DDoS Attack Detection System Using Semi-supervised Machine Learning in SDN

By

Mohamed Ahmed Azmi Etman

B.Sc., Helwan University, Egypt, 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

DDoS Attack Detection System Using Semi-supervised Machine Learning in SDN

Master of Applied Science, 2018

Mohamed Ahmed Azmi Etman

Computer Networks

Ryerson University

Abstract

Distributed Denial of Service (DDoS) attacks is one of the most dangerous cyber-attack to Software Defined Networks (SDN). It works by sending a large volume of fake network traffic from multiple sources in order to consume the network resources. Among various DDoS attacks, TCP SYN flooding attack is one of the most popular DDoS attacks. In this attack, the attacker sends large amounts of half-open TCP connections on the targeted server in order to exhaust its resources and make it unavailable. SDN architecture separates the control plane and data plane. This separation makes it easier to the controller to program and manage the entire network from single device to make better decisions than when the control is distributed among all the switches. These features will be utilized in this thesis to implement our detection system. Researchers have proposed many solutions to better utilize SDN to detect DDoS attacks, however, it is still a very challenging problem for quick and precise detection of this kind of attacks. In this thesis, we introduce a novel DDoS detection system based on semi-supervised algorithm with Logistic Regression classifier. The algorithm is implemented as a software module on POX SDN controller. We have conducted various test scenarios, comparing it with the traditional approach in the literature. The approach presented in this thesis manages to have a better attack detection rate with a lower reaction time.

Acknowledgments

First and foremost, praises and thanks to the God, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.

My special and heartily thanks to my supervisor, Dr. Ngok-Wah Ma sincerely for his support, patience, motivation, encouragement and time throughout my research work and thesis writing.

I am also deeply thankful 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.

A special thanks to my parents and my wife, who constantly supported and encouraged me throughout the course of my studies.

Finally, my thanks go to all the people who have supported me to complete the research work directly or indirectly.

Table of Contents

List of Figures

List of Tables

Abbreviations

ACK	Acknowledgement
API	Application Program Interface
CPU	Central Processing Unit
DDoS	Distributed denial-service-attack
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
LR	Logistic Regression
ML	Machine Learning
SDN	Software Defined Networking
SYN	Synchronize Sequence Number
SVM	Support Vector Machine
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
OVS	Open vSwitch
ICMP	Internet Control Message Protocol
ONF	Open Networking Foundation

Chapter 1

1 Introduction

1.1 Problem Statement

Software Defined Network (SDN) is an emerging networking paradigm that improves on the limitation of traditional network infrastructure. By moving the control plane to a centralized controller, SDN simplifies the policy enforcement and network configuration. In addition, the networking devices are made simple with lower costs.

Despite the fact that SDN has obvious advantages such as centralization and network flexibility, it is as susceptible as traditional networks to network security attacks, a case in point, DDoS attack. Among DDoS attacks, TCP SYN flooding attacks is one of the most popular and effective attacks. TCP SYN attack exploits the weakness of the Transmission Control protocol (TCP). The attacker produces many half-open TCP connections on the targeted server in order to degrade its availability. Furthermore, when applied to SDN, TCP SYN flooding attack also introduces control plane saturation attack. In particular, the attacker generates a significant number of TCP SYN packets and elicits the data plane switches to forward them to the controller. As a result, the performance of the controller degrades and may not be able to respond to genuine requests in an acceptable time. In the worst case scenario the attack may lead to a network outage.

In this thesis, we will propose semi-supervised machine learning algorithm by using Logistic Regression classifier to detect TCP SYN flooding attack in SDN, which utilizes the dynamic programmability nature of SDN to detect attacks. We have employed four important features (Packets, Bytes, Flows and One_Packet) to train our LR classifier off-line. The predictive model produced from off-line training is then used in on-line attack detection, and simultaneously continue to classify the incoming traffic, to update the training dataset to retrain our classifier. We will implement extension modules on POX controller and evaluate it under different attack scenarios.

1.2 Research Objective and Contribution

The target of this research is to detect DDoS attacks by utilizing the unique feature of SDN architecture. The main contributions of this thesis are:

1. Propose a semi-supervised machine learning methodology for detecting DDoS attacks.
2. Implement the proposed semi-supervised algorithm in the SDN environment.
3. Compare, analyze and evaluate the proposed detection system with statistical approach [16] found in the literature.

1.3 Thesis structure

The rest of this thesis is organized in the following order:

Chapter 2 provides necessary background information of SDN architecture and the current research of DDoS attacks in SDN networks.

Chapter 3 describes the details of the proposed machine learning classifier in an off-line mode as well as an on-line mode.

Chapter 4 determines the effectiveness of machine learning classification in an SDN/Open Flow environment by evaluating a Semi-supervised algorithm by using Logistic Regression classifier on an on-line network testbed.

Chapter 5 closes the main body of the thesis by presenting the final conclusions and proposing

Future work.

Chapter 2

2 Background and Related Work

This chapter covers Software Defined Networking (SDN) architecture in detail. We will take a look at SDN benefits, SDN-capable switching devices, SDN controller, OpenFlow protocol, and finally review previous work in SDN security will be reviewed.

2.1 Architecture of SDN

Software Defined Network (SDN) is an emerging architecture that moves the control plane from the networking devices, such as routers and switches to a centralized controller [2]. To increase the flexibility, the controller is also made programmable, the programmability of SDN provides a platform to researchers and network engineers to deploy new ideas and network applications.

SDN network architecture can be defined as four pillars [2]:

1. The control plane and the date plane are separated.
2. Network traffic forwarding decisions are flow-based, instead of destination-based.
3. Control logic is moved to an external entity, namely, SDN controller or Network Operating System NOS.
4. The network is programmable through software applications running on top of the controller, which interacts with the underlying data planes devices.

As shown in figure 1, the architecture consists of three layers. The infrastructure layer also known as data plane, comprises the forwarding elements for data traffic. The control layer, is an external control plane which manages and programs the infrastructure layer. Finally, the application layer contains network applications for the network features, such as network management [1].

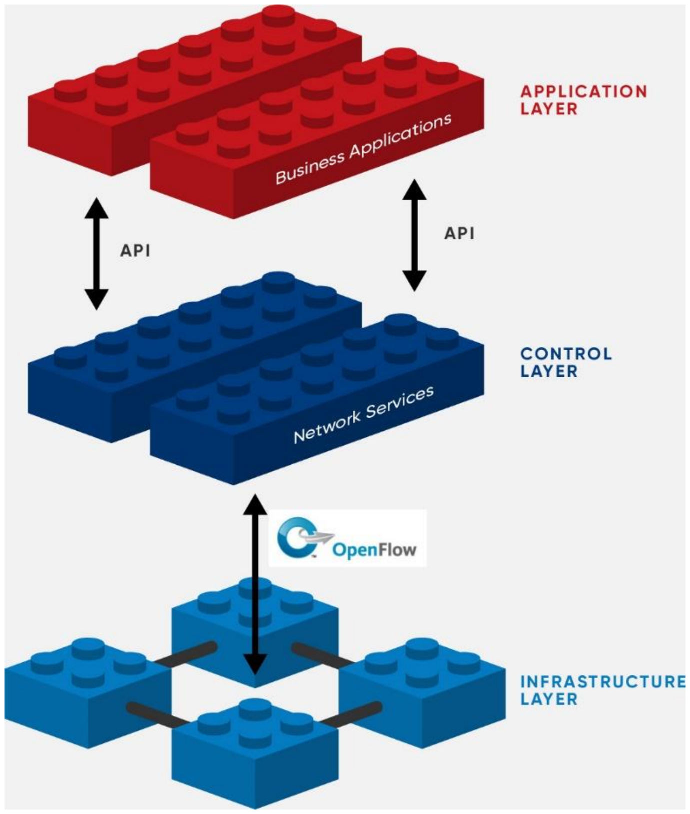


Figure 1: SDN Architecture [1]

2.2 SDN Security benefits

SDN can be utilized to manage the entire network as if it was a single device, it will be considered as the hardware independent next generation networking paradigm in which the networking device from any vendors could be controlled through SDN. Network capabilities such as, global view of the network, dynamic updating of forwarding rules and so on, can facilitate DDoS attacks detection, however the separation of the control plane from the data plane leads to new emerging types of attacks. For instance, an attacker can exploit the characteristics of SDN to launch DDoS attacks against the control, infrastructure and application layers of SDN. We can observe some good features of SDN in defeating DDoS attacks in the following table [32]:

Good features of SDN	**Benefits for defending DDoS attacks**
Decoupled control and data plane	It is able to establish large scale attacks and defence experiments easily.
Controller centralized view of the network	It helps to build a consistent security policy
Programmability of the network by external applications	It supports a process of harvesting intelligence from existing IDSs and IPSs
Software based traffic analysis	It improves the capabilities of a switch using any software-based technique.
Dynamic updating of forwarding rules and flow abstraction	It helps to respond promptly

Table 1: Good features of SDN in defeating DDoS attacks [32]

2.3 OpenFlow Protocol and OpenFlow Switch

OpenFlow is a standard protocol that is used to provide communication between a controller and a network device. The OpenFlow architecture is made up of 3 main components: an OpenFlow switch/router, OpenFlow controller and a secure channel to connect the switch to the controller. The OpenFlow protocol allows the SDN/OpenFlow controller to install flows for hosts connected, and gather traffic statistic from switch through a secure communication channel as shown in figure 2.

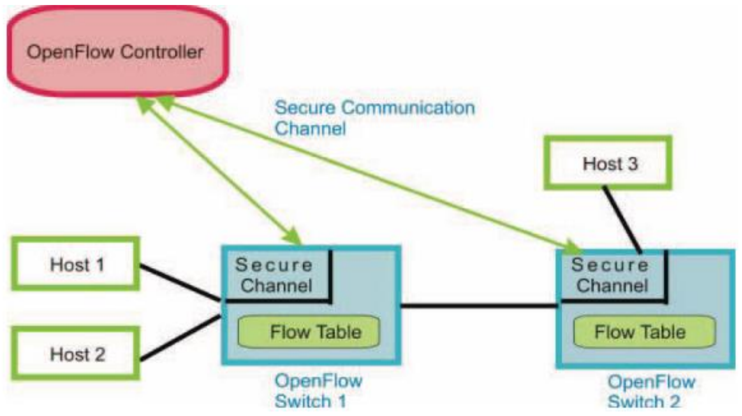
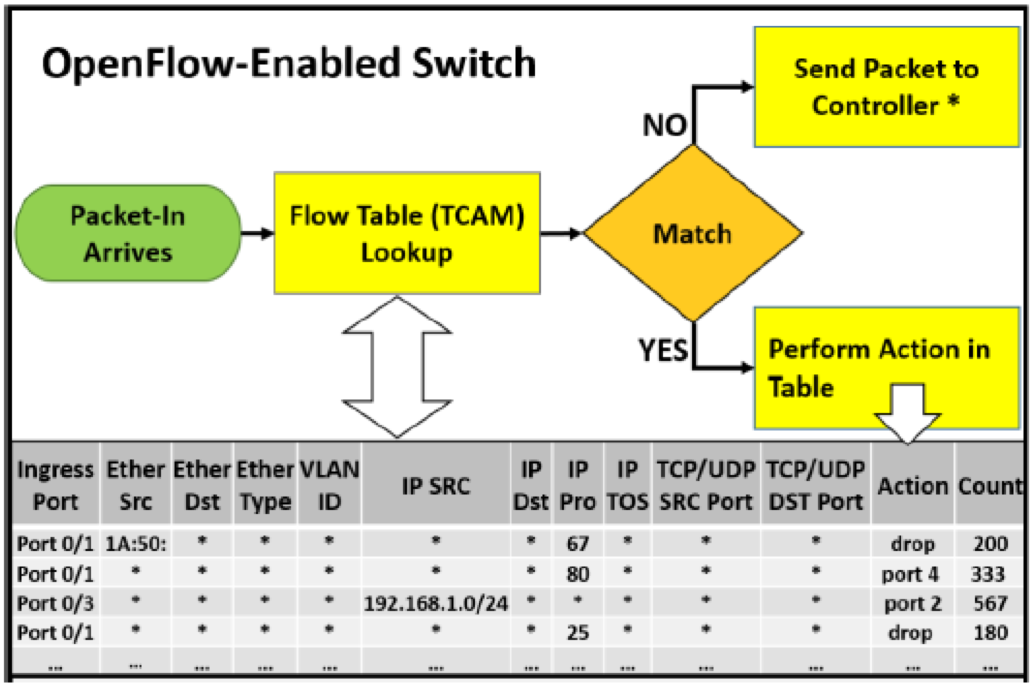


Figure 2: OpenFlow Protocol [29]

The network devices which is usually an OpenFlow switch, is only responsible to forward data. No proprietary vendor specific hardware is required. The forwarding behavior of a switch is controlled by the controller, which installs flow entries in the flow tables of the switch to steer the traffic to appropriate destinations [6, 23]. In such devices Ternary Content Addressable Memory (TCAM) is used to build flow table [30].

An example of OpenFlow enabled switch is shown in figure 3. In this figure, the header fields of arriving packets are matched against header fields located in the TCAM or flow table. The action column indicates what to do with the packet if a match is found, the action assigned to that entry is applied and the counter for the entry will be updated. If no match is found, then the switch forwards Packet_IN message to the OpenFlow controller over secure channel to obtain new flow rules. The controller processes the packet and sends Packet_OUT / Flow_MOD message back to the switch to be added to the flow table.

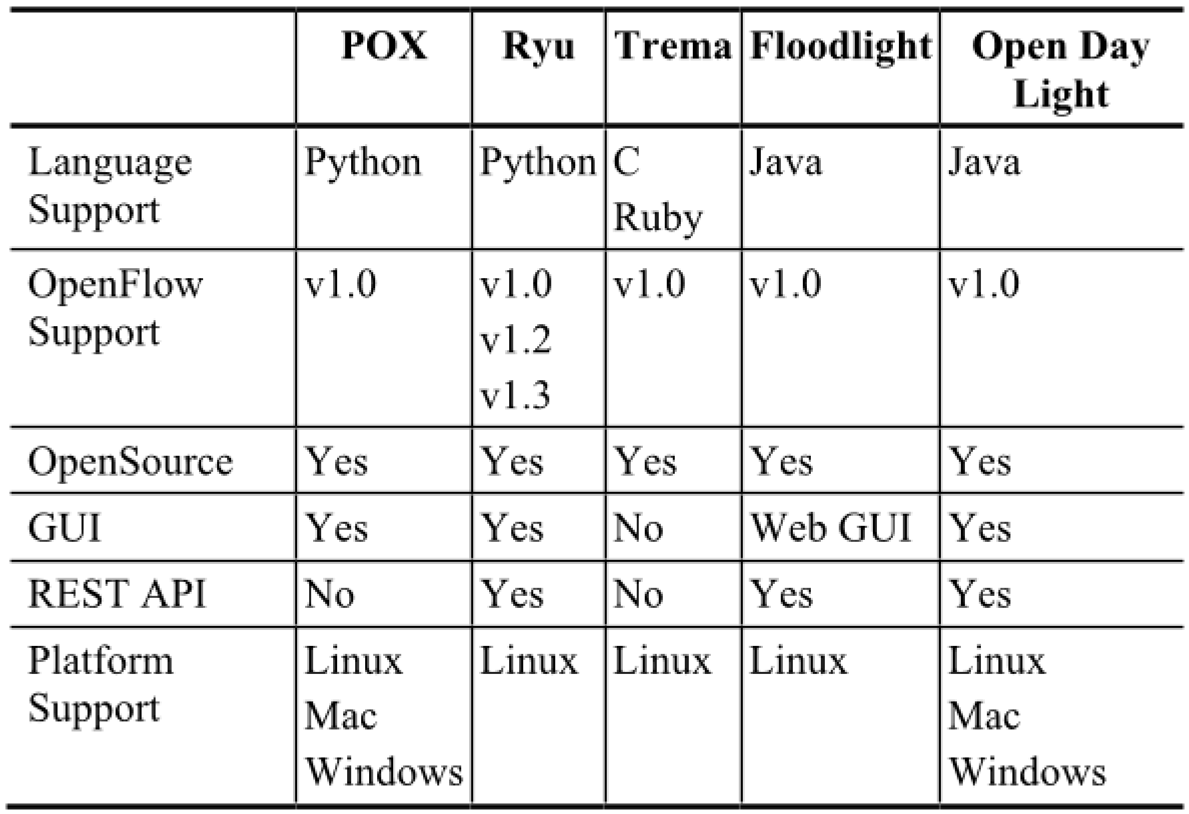


Ingress Port	Ether Src	Ether Dst	Ether Type	VLAN ID	IP SRC	IP Dst	IP Pro	IP TOS	TCP/UDP SRC Port	TCP/UDP DST Port	Action	Count
Port 0/1	1A:50:	*	*	*	*	*	67	*	*	*	drop	200
Port 0/1	*	*	*	*	*	*	80	*	*	*	port 4	333
Port 0/3	*	*	*	*	192.168.1.0/24	*	*	*	*	*	port 2	567
Port 0/1	*	*	*	*	*	*	25	*	*	*	drop	180
...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 3: OpenFlow-enabled switch [31]

2.4 SDN Controllers

SDN Controllers in a software-defined network is the brain of the network. SDN controllers use southbound protocol, such as OpenFlow, to communicate with the network devices. Figure 4 shows current well-known SDN controllers including POX, ONOS, Floodlight and OpenDaylight.



	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

Figure 4: Different SDN Controllers [30]

POX

POX [33] is an open source controller for developing SDN applications. POX controllers provides an efficient way to implement the OpenFlow protocol which is the de facto communication protocol between controllers and switches. We have validated the proposed algorithm in our thesis in POX controller.

2.5 DDoS attacks

DDoS attacks are one of the most proper kind of network attacks. The aim of the attacks is to exhaust network resources and make the network unavailable to the legitimate users by sending a large amount of attack traffic. The attacker usually sends a massive volume of packets to the attack target with spoofed source IP addresses to make it look like normal traffic. The differentiation between legitimate and malicious traffic sent by an attacker is one of the biggest challenges to detect DDoS attacks as shown in figure 5

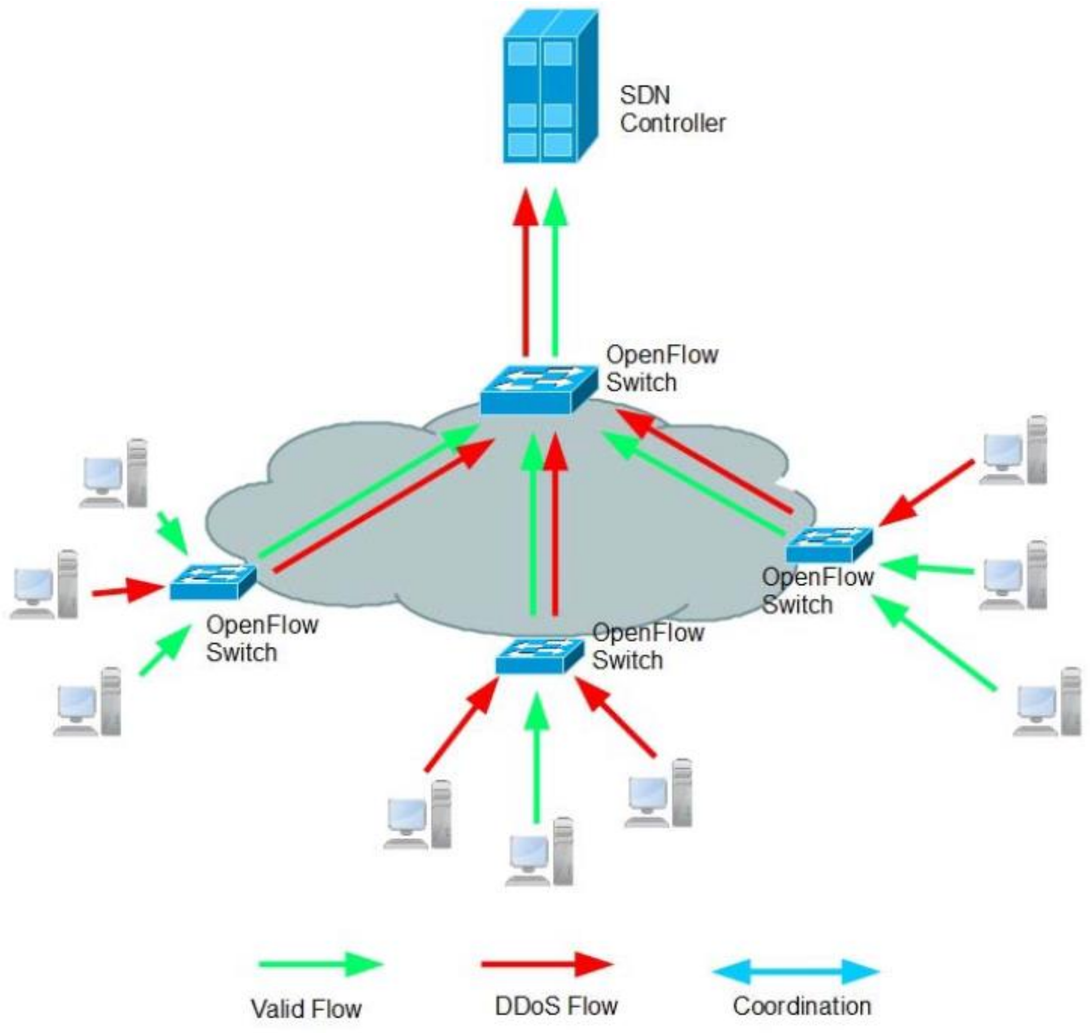


Figure 5: DDoS Attack on SDN controller [28]

2.5.1 TCP SYN attack

In TCP connection, the connection between legitimate host and server are established by using three way TCP handshake. As illustrated in figure 6, the client initiates SYN packet and server replies with SYN-ACK. The server entering the listening state allocates a transmission control block (TCB) to store user data and connection information. The server stays on this state for a certain amount of time even if it does not receive the ACK from the client. If the client responds with ACK, the TCP connection will be established successfully [27].

However, this mechanism may be exploited by attackers using SYN spoofed attack. The attacker could launch SYN flooding that create half-open connections using forged IP addresses.
As a result, the server queues will be congested, denying new TCP connections to legitimate hosts.

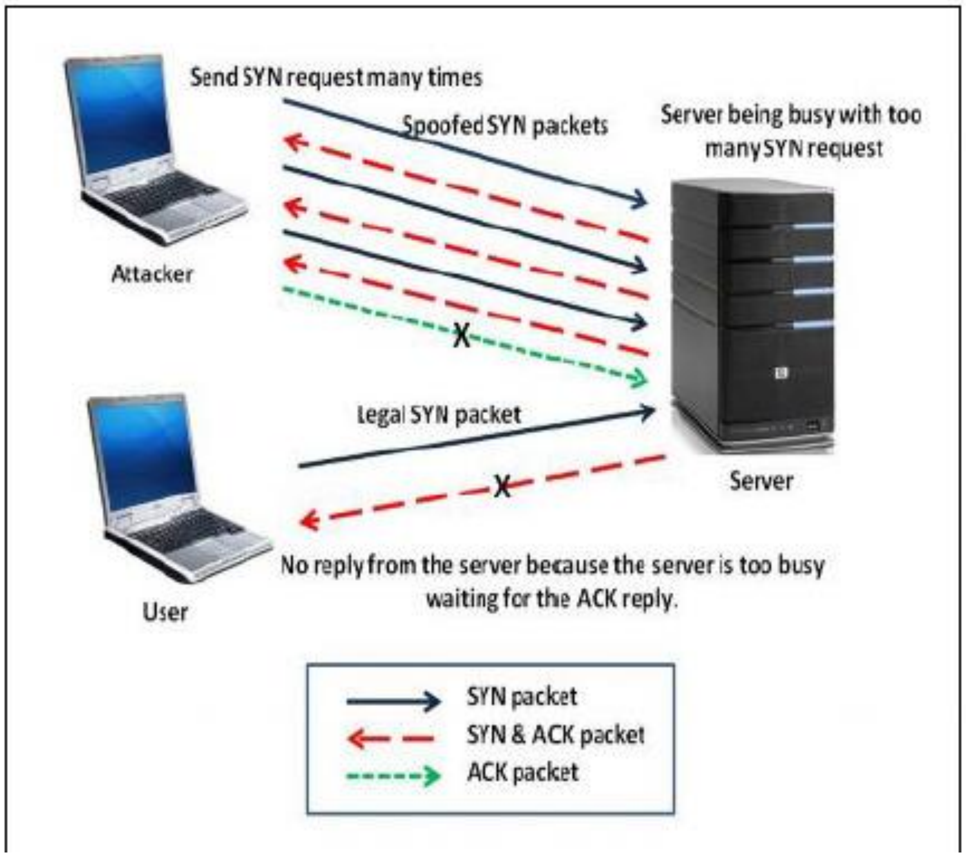

Send SYN request many times
Spoofed SYN packets
Server being busy with too many SYN request
Attacker
Legal SYN packet
Server
No reply from the server because the server is too busy waiting for the ACK reply.
User
SYN packet
SYN & ACK packet
ACK packet

Figure 6: TCP SYN Flood [27]

2.5.2 UDP flood attack

The fact that UDP is a connectionless networking protocol can make UDP more exposed to exploitation. It’s similar to TCP SYN flood attack, UDP flood attack can be initiated by sending a large number of UDP packets with spoofed source IP addresses to hide the source of attack. The attacker overwhelms the victim server leading it to be unreachable by other legitimate clients.

2.5.3 ICMP flood attack

Normally, ICMP is used in testing connectivity between two hosts, one of the hosts initiates an ICMP echo request to solicit an echo reply from the other host. Like TCP and UDP flood attacks, the attackers use spoofed or un-spoofed fabricated ICMP echo requests to overload a target network with data packets that leads to consume victim bandwidth and prevent access to the legitimate users [27].

2.6 Related Works

The logically centralized control-plane in SDN architecture is a sword with of two edges. On one hand, its global view of the network and dynamic updating of forwarding rules, make it easier to detect and respond to DDoS attacks. On the other hand, it provides single point of failure in the network that make it a target for DDoS attacks. In this section, we will present a selection of previous research regarding DDoS attack detection and countermeasures using SDN.

AVANT-GUARD [4] is a framework that rely on two modules to identify TCP SYN flood attacks. The first module named Connection migration, which is installed as an extension in data plane to act as a proxy for ingress SYN packets to prevent saturation attack from reaching control plane. The second module named actuating triggers which installed in the controller. If network traffic were identified as malicious, actuating triggers module enables an event for the controller to install flow rule in switch to decrease response time. The most conspicuous side effect of this mechanism is the performance penalty, since it utilizes connection migration, each flow needs to be classified. AVANT-GUARD requires upgrading all switches in data plane to support AVANT-GUARD features.

The authors of [5] proposed solution called LineSwitch as an extension of AVANT-Guard. LineSwitch utilize SYN proxy technique in data plane switches for all TCP connections from given IP source. Once it detects the SYN flood attack, it will apply probabilistic black-listing and prevent all TCP packets from this IP source. This solution can effectively prevent the SYN flood attack against SDN network with little expense, however, similar to AVANT-GUARD, it is required to upgrade data plane switches when applying LineSwitch mechanism.

Chin et al. [26] proposed collaborative approach for detection and containment of TCP SYN flood attacks. They have implemented a system with two main components, the first component is sensory monitors which are distributed across the network for each open virtual switch (OVS). The second component is called correlator residing in the controllers. Once the monitor detects an attack, it informs the correlator by sending an alert message which holds a number of source IP addresses found in SYN packets. As a result, the controller manages further actions to either update normal traffic profiles, in order to avoid future false alerts or to mitigate effects of an attack by blocking malicious hosts. However, this work in a large scale network will create an overhead on the network, because it will need Monitors and Correlators for every OVS and Controllers respectively.

Mohan Dhawan et al. in [7] proposed SPHINX as a solution to detect suspicious activity in the network using real-time flow graph. SPHINX is implemented as a module in the controller and can detect TCP flooding attack, by validating the rate of packet-in messages which correspond to the new SYN requests. SPHINX raises an alert if the rate of PACKET_IN messages is above the administrator-specified threshold. However, setting a control threshold for detecting SYN flooding attack is a static solution and would likely generate false alarms [8].

SLICOTS [9] proposed as lightweight extension which is installed as a module in the controller to detect and mitigate SYN flooding attacks in SDN Environment. It monitors all ongoing TCP connections requests, however, SLICOTS validates 3-way handshake by installing temporary forwarding rules during TCP handshaking process. If the traffic is benign, it will install permanent forwarding rules between the client and server. Otherwise, SLICOTS blocks the attacker that creates a high volume number of half-open TCP connections. The main disadvantage of this solution is the installation of temporary or new forwarding rules that might increase the bandwidth consumption in case there is a flash crowd user's request.

Nhu et al. [10] proposed a method to protect SDN network against DDoS flooding attacks. The authors analyzed the user behavior in terms of number of packets per connection using the fact that the number of packets per connection generated by the malicious users is usually smaller than the number of packets per connection generated by normal users. Therefore, DDoS attacks can be detected based on the average number of connections and minimum number of packets per connection. If an attack is detected, the controller mitigates the attack by changing the default hard timeouts and idle timeouts of attacker flows to a minimum value to quickly remove them from the switch. However, the authors admit their approach will not scale for large traffic attacks.

FloodGuard [11] system is divided to two modules to detect data-to-control plane saturation attacks. When the saturation attack is detected by FLOODGUARD, the packet migration module will redirect the table-miss packets in the OpenFlow switch to the data plane cache. At the same time, the proactive flow rule analyzer module will dynamically track the runtime value of the state sensitive variables from the running applications, convert generated path conditions to the proactive flow rules dynamically and install these flow rules into the OpenFlow switches. Then the data plane cache will slowly send the table-miss packets as packet_in messages to the controller by using rate limiting and round-robin scheduling algorithm. However, this approach needs additional devices to accommodate table-miss traffic and switches upgrades.

Wang et al. [12] proposed Entropy-Based Distributed DDoS Detection Mechanism. The entropy algorithm is distributed in every OpenFlow switch to make it more intelligent. Within a predefined time period, it takes a copy of the packet number counter of the flow entry in switch table and calculate the probability distribution of destination IP address. Once a DDoS flooding attack is confirmed, this algorithm will send an alert information to the controller. The motivation is to detect attacks at the data-plane level in order to reduce overheads introduced by classifying traffic on the controller level.

Seyed and Marc [13] proposed entropy to detect DDoS attacks against SDN controller. The solution utilizes lightweight module implemented at the controller side and mainly rely on two parameters: window size and threshold. For every 50-packet window, an entropy is calculated based on the frequencies of the destination IP addresses. An attack is declared if the entropy is below certain threshold for 5 consecutive windows. However, this approach may not be able to detect a low attack rate since it uses a predefined threshold value.

StateSec [15] is a novel approach that is added to switch processing capabilities to detect DDoS attacks. This approach extends the stateful SDN concept [14], and on top of it, DDoS detection and mitigation mechanism. StateSec detection process consists of three main stages: Traffic monitoring, anomaly detection and mitigation. The Traffic monitoring module is installed in the switch, it uses traffic features such as destination IP address, destination port, source IP address and source port. A lightweight anomaly detection module is integrated into the switch to calculate the entropy of each monitored traffic feature associated counters. If the entropy of one or multiple traffic features generates a drop or raise that crosses predefined thresholds, an anomaly is detected. The controller mitigation module installs mitigation rules into the switch through standard OpenFlow functionalities. However, software upgrade is needed for the switches.

The author in [6] made a survey on current SDN solutions to different detection and mitigation techniques and highlighted pros and cons of each category. He proposed ProDefense framework an efficient system for DDoS attack detection and mitigation. It has three major components, namely, traffic flow collector, policy engine, attack detector and mitigation. The flow collector is used to collect traffic statistics from OpenFlow switches. The policy engine is used to define attack detection policy and mitigation policy. The detection utilizes exponentially weighted moving average (EWMA) filters based on three different variants of filters to control the false positive. These filters are highly reactive (HR) filter, intermediate reactive (IR) filter, and least reactive (LR) filter, which are used for smart grid applications in catastrophic, critical, and marginal categories respectively. The attack detector module takes the input from Traffic Flow collector and generates security alerts with respect to the policy defined in Policy Engine. These security alerts trigger the mitigation module for taking appropriate action. However, details on the detection and mitigation mechanism itself were not provided.

Dhaliwal's [16] proposed statistical approach by implementing Exponential Weighted Moving Average (EWMA) and Exponential Weighted Moving Standard Deviation (EWMSTD) to detect and mitigate SYN and HTTP flood DDoS attacks. It uses the means and standard deviations of SYN packet hit counts and single-packets flows to compute various thresholds. The detection mechanism is divided into three attack detection phases. The first attack detection phase utilizes a time series window-based traffic statistic measurement. The detection algorithm continuously monitors the rate of SYN packets arriving in each fixed 3-second time window, and compares it to a threshold at the end of each time window. The detection algorithm quickly raises a warning when the rate of SYN packets exceeds the threshold. However, a surge of legitimate traffic can also increase the rate of SYN packets arriving at the controller. Thus, in the second attack detection phase, further analysis is done by comparing the source IP addresses of all the flows to a database that contains valid source IP addresses. Traffic flows that have the match are considered legitimate. After that, the controller proceeds to the third phase where it computes the ratio of single-packet flows to the total number of flows and compares this ratio to a threshold. However, this method introduces some delay in attack detection. The statistical method based on time window, which is three consecutive time windows, each time window is 3 which equates to 9 seconds is considered to be inefficient in detecting large rate of attacks. Consequently, the network or controller could be broken before the detection mechanism reaches the ninth second in order to confirm an attack.

Rodrigo Braga et al. [7] proposed a lightweight method for DDoS attack detection. The method utilizes the feature which natively exists in OpenFlow protocol to collect flows from switches periodically using flow collector module. The feature extractor module extract 6 tuple features from collected flow as follows: Average of Packets per flow (APf), Average of Bytes per flow (ABf), Average of Duration per flow (ADf), Percentage of Pair-flows (PPf), Growth of Single-flows (GSf), and Growth of Different Ports (GDP). Finally, classifier module makes use of unsupervised machine learning algorithm Self Organization map, to classify the traffic pattern to benign or malicious traffic. However, the author methods can’t detect low rate attacks.
Saurav Nanda et al. [18] proposed anomaly detection using four well-known machine learning algorithms to identify potential malicious connections. The author trained the following machine learning algorithms on historical data: C4.5, Bayesian Network (BayesNet), Decision Table (DT), and Naive-Bayes using WEKA framework. He implemented the trained model in SDN environment to predict anomalies in real time network. He showed that among the four algorithms, Baysian network achieved prediction accuracy 91.68 % on average.

In [19] Restricted Boltzmann Machine (RBM) based Detection System for DDoS is proposed. The authors used unsupervised machine learning algorithm to self-learn defined network metrics. RBM get activated to classify the DDoS attacks after computing the threshold values of two features, the first feature is the hit count of incoming packets to controller with same source IP address and MAC address of the switch. The second feature is energy consumption rate of switch. When Hit count of IP addresses is greater than predefined threshold value and energy consumption rate is higher for same MAC address, the flows of same source IP address and Destination IP address passed to RBM to classify the traffic to DDoS attack based on protocol type. If attack is declared by RBM, the packet will be dropped, otherwise the packets will be allowed to the controller. The RBM algorithm achieved detection rate 92%.

In [22] proposed detection for intrusions and DDoS attacks using two machine learning classifier, SVM and Neural Network (NN). These two classifier model generated by using one feature which is counting the number of hosts per second. This value passed to the classifier for labeling. If the packet is classified as anomaly it will be labeled accordingly and the connection appeared to be anomaly detected, otherwise it will be treated as normal traffic and passed to the destination. However, the accuracy achieved is 80% and this is due to noise in the training dataset as data accumulated for every second.

Trung et. al. [20] proposed an OpenFlowSIA protection technique to SDN from flooding attacks. OpenFlowSIA utilizes SVM machine learning algorithm to detect the attacks based on extracted features, which are collected from network switches statistics. They extract two features from network flows, the first feature is packet number per flow and the second feature is the flow duration. Once DDoS attack confirmed, the mitigation decides the type of the attack based on two types. The first type is attackers send a large number of flows and each flow consists of high packet numbers. The second type when flooding attackers generate a vast number of flows using fake source IP addresses and each flow transmits only form 1 packet. If it is the first type, the idle-timeout of the flow change to 0. If the attack is the second type, DeleteFlowEntry function is used to delete the flows from switch flow-tables. Otherwise, the default flow idle-timeout value will be set to normal flows. However, the SVM model used is not adaptive.

Ashraf et al. [21] provided a survey for different machine learning schemes for mitigating DDoS attacks in SDN environments. The authors studied advantages and disadvantages of supervised machine learning algorithms such as neural network, support vector machine, genetic algorithms, fuzzy logic, Bayesian networks, and decision trees to differentiate between the benign traffic and attack traffic.

Kokila et al [23] used SVM as a classifier to detect DDoS attacks. They found that SVM algorithm accuracy is 95% which outperform the other machine learning algorithms in their literature. However, their algorithm could only detect attacks without trying to mitigate these attacks [22]

In [24] SVM trained model used for detecting malware by using only traffic features that can be extracted from Openflow messages. The controller collects flows periodically from the switches. Once it receives the feature that have number of packets, number of bytes, and duration of the flows. It processes this information to extract other features: Byte rate, Packet rate, and Average packet length. Finally, it passes these features to the SVM classifier to classify the traffic to normal or malware, the author achieved 95% accuracy. However, the trained model is not adaptive

In [25] DDoS detection system implemented in SDN environment. The authors compared various methodologies of supervised machine learning algorithms such as Naive Bayes, K-Nearest neighbor, K-means and K-medoids to classify the traffic as normal and abnormal. Then these algorithms are trained on a dataset which consists of a feature counting the number of hosts connected in 10 seconds to the switch. It was tested and the results shows that Naïve Bayes is the best algorithm with 94 % accuracy. However, the classifier is not adaptive.

Merlin [17] proposed two detection approaches, statistical approach and machine-learning approach. The statistical approach utilizes entropy computation for new flow arrival rate, packets per flow and flow duration to compute various thresholds. These thresholds are then used to differentiate normal and attack traffic. The machine learning approach implemented by Random Forest algorithm to detect the DDoS attack. The author showed that the proposed machine-learning approach outperform the statistical approach. However, the trained model wasn't adaptive.

2.7 Chapter Summary

This chapter covered SDN, Openflow, and controller background. We have presented a comprehensive survey on SDN. We have discussed the benefits of SDN, explained its main concepts and how it differs from traditional networking.

We surveyed different approaches for detecting DDoS attacks and showed how SDN has been used as a platform specific to detect and mitigate DDoS attacks.

The next chapter will focus on developing an effective solution using Semi-Supervised machine learning algorithm that can detect DDoS attacks more effectively with less false positive and negative alarms.

Chapter 3

3 Proposed DDoS Detection System

In this chapter, we will first introduce different types of machine learning. We then introduce our proposed method which utilizes semi- supervised learning algorithm by using Logistic Regression classifier to detect DDoS attacks. We define four features from the training datasets and use it to train our LR classifier off-line. Once the classifier is trained, it will be used for on-line attack detection, and at the same time, continue to classify the incoming traffic to update the training dataset to retrain our classifier in the background.

3.1 Introduction to Machine learning

According to Tom M. Mitchell [34], the machine learning (ML) can be defined as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E".

An example of a Machine Learning program is email spam detection system that can learn from given samples of spam emails and nonspam emails called ham. The spam emails pattern may be contained in the senders email address or in the email subject phrases such as (“4U,” “credit card,” “free,” and “amazing”)[36].The prepared samples that detection system uses called the training dataset. Each training example is known as training datapoint. In this detection system we can define the task T to flag spam for new emails based on the experience E obtained from the training dataset and the performance measure P defined by accuracy which is the ratio of successfully classified emails to the total emails.

Similarly we can utilize the machine learning capability in DDoS attack detection. Since the network traffic load is fluctuating, it’s hard to find a solution that identify the attack traffic from benign traffic with high accuracy using the traditional approach [16]. However, in machine learning, we can learn network traffic patterns and adapt to new traffic information with high accuracy by utilizing semi-supervised algorithm.

The machine learning tasks can be classified to supervised, unsupervised and semi-supervised learning as shown in figure 7.

3.1.1 Supervised learning

In supervised learning, input data, called training data, is fed to the algorithm. The training data is pre-classified and labeled. It is called supervised learning because the process of an algorithm learning from the training dataset can be considered as a tutor supervising the learning process. The training process continues until the model achieves a desired level of accuracy on the training data. This approach requires having a dataset that represents the system under consideration and can be used to estimate the performance of the selected method. A typical supervised learning task is classification. The spam filter is a good example of this, it is trained with many example emails along with their classes (spam or ham), and it must learn how to classify new emails [35].

3.1.2 Unsupervised learning

Unsupervised learning is the opposite of supervised learning. The system tries to learn without a tutor. Input data is not labeled, thus, not classified. Therefore, the system mainly focuses on finding specific patterns in the input. An example of the unsupervised learning approach is clustering, which is used for detecting useful clusters in the input data based on similar properties defined by a proper distance metric such as Euclidian, Jaccard, and cosine distance metrics [36].

3.1.3 Semi-supervised learning

Semi-supervised learning algorithms has been growing in popularity over the last decade. They are considered somewhere between unsupervised and fully supervised learning algorithms. The input data for the learning algorithm is a mixture of labeled and unlabeled data [35]. Semi-supervised learning algorithms have the ability to save the time and effort to process the training data labeling. In addition, semi-supervised eliminates the human error that might occur during the labeling of the data. It has been found that unlabeled data, when used in combination with a small amount of labeled data, can produce considerable improvement in learning accuracy. In our research we will utilize semi-supervised algorithm with Logistic Regression classifier.

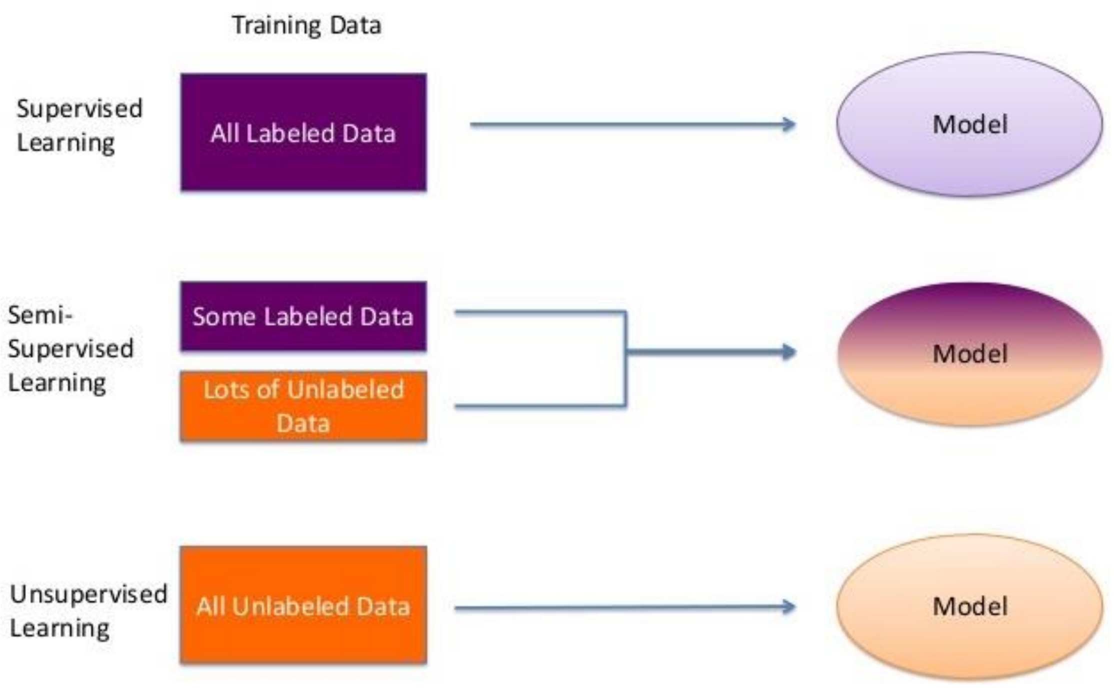


Figure 7: Types of Machine Learning for DDoS Detection [37]

3.2 Feature Selection

Data is required to train the machine learning algorithm to produce predictive model. The first step to create accurate predictive model is feature selection. By choosing best features we can get better detection accuracy whilst requiring less data. Fewer Features is desirable because it reduces the complexity of the model, and a simpler model is easier to implement and would have smaller overheads. We employ four features to build the training and testing data. We name these features as Packets, Bytes, Flows and One_Packet. The meanings of these features are given in table 2.

Features	**Description**
Packets	Total number of packets transmitted to server per time window
Bytes	Total number of Bytes transmitted to server per time window
Flows	Total number of flows destined to the server per time window
One_Packet	Total number of single packet flows destined to server per time window

Table 2: Feature Selection of Dataset

The use of Packets and Bytes features are obvious. A DDoS attack will always increase the values of these features significantly. However, these two features alone cannot be relied on 100% in identifying that there is an attack for the occasional flash crowd traffic, which may also cause the increase of the values of these features. To improve the accuracy of the attack detection, we also include two extra features, One-Packet and Flows. The idea of using One-Packet feature derived from [16]. The detection method proposed by the author observed that flows that contain single packet are considered as a sign of DDoS attack with spoofed addresses, such as TCP SYN flood attacks. We can further improve the accuracy by including in our detection method the Flows feature as well. Reason being, in the DDoS attack, the attacker sends a large number of SYN packets to overflow the switch connection table with half-open connections, increasing the total number of flows or the value of the Flows feature. The values of these features are extracted from the traffic statistics stored in the Openflow switch every three seconds. The network traffic statistics collection will be discussed in more details in chapter 4.

3.3 Semi-Supervised Machine learning

Many algorithms and techniques have been proposed for DDoS attack detection. But less work has been done in the field of SDN networks based on machine learning, in particular semi-supervised machine learning. In [37] the author proposed a new framework for QoS-aware traffic classification in SDN based on semi-supervised learning. This approach can classify the network traffic according to the QoS requirements. The system employs deep packet inspection (DPI) for network traffic to train semi-supervised learning based on Laplacian SVM classifier. The Laplacian SVM classifier achieved 90% accuracy which surpassed a previous semi-supervised approach based on k-means classifier. The other machine learning algorithms for SDN DDoS attack detection in [17] [18] [20] [23] [24] [25] were using supervised machine learning algorithms, which means the classifiers are trained with a fixed labeled network traffic dataset, which represents a specific stationary traffic pattern. However, real life traffic is never stationary. To solve this issue, unsupervised ML was proposed for DDoS Attacks [17] [19], which employs unlabeled dataset as the training set. However, it is difficult for unsupervised ML-based approaches to achieve a good performance with low complexity.

We propose a novel DDoS detection system which uses a semi-supervised algorithm with Logistic Regression classifier. The objective of a semi-supervised algorithm is to combine information from unlabeled data with that of labeled data to iteratively improve the learning model for DDoS detection.

The semi-supervised algorithm contains the following steps:

1. Train Logistic Regression classifier on labeled dataset to produce a preliminary detection model.
2. Use the trained model to classify unlabelled dataset.
3. Add new classified data with high confidence to labeled dataset and reject data with lower confidence.
4. Retrain classifier on previously labeled dataset plus the newly added labeled data.
5. Iterate through steps 1 to 4 until the detection rate converges.

3.4 Logistic Regression

Logistic Regression (LR) is one of the most popular machine learning classifiers for binary classification [38] [39]. This is because it is a simple algorithm that performs very well on a wide range of problems. In addition to, it has probabilistic framework that make it easy to adjust classification thresholds or get confidence intervals, unlike SVM classifier [37] which doesn't give probabilistic interpretation to the data.

LR measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function. The independent variables X are the features or Input data we are going to use to predict the target, the dependent variable is the target class variable or output data Y we are going to predict with values 0 or 1.
LR is commonly used to estimate the probability that an instance belongs to a specific class, for example, what is the probability that this email is spam? If the estimated probability is greater than 50%, then the classifier predicts that the instance belongs to spam with class labeled "1", otherwise it predicts that it does not and is labeled "0". In this case, the classifier is a binary classifier.

LR can be formulated as the conditional probability of the dependent variable $Y = C$ given X, where C is the label or class. For attack detection, the classifier is also binary with $C = 1$ if there is an attack and $C = 0$ if there is no attack.

LR probability is calculated by equation (1) and (2) using the logistic function $\frac{1}{1+e^{-z}}$, which is also known as the sigmoid function:

$$P(Y = 1 \mid X) = \frac{1}{1+e^{-z}} \quad (1)$$

$$P(Y = 0 \mid X) = 1 - P(Y = 1 \mid X) \quad (2)$$

Where

$$Z = \omega_0 + \omega_1 X_1 + \omega_2 X_2 + \cdots + \omega_n X_n \quad (3)$$

Where $X_{1,}\ X_{2,} \ldots X_n$ are the n features inputed to the classifier. The coefficients $\omega_{0,\ \ldots,}\ \omega_n$ are determined by utilizing maximum likelihood estimation (MLE) [38] based on the complete training dataset. Once the coefficients are determined, the LR probability can be calculated and the datapoint will be classified subsequently to attack “1” or no attack “0” as shown in figure 8.

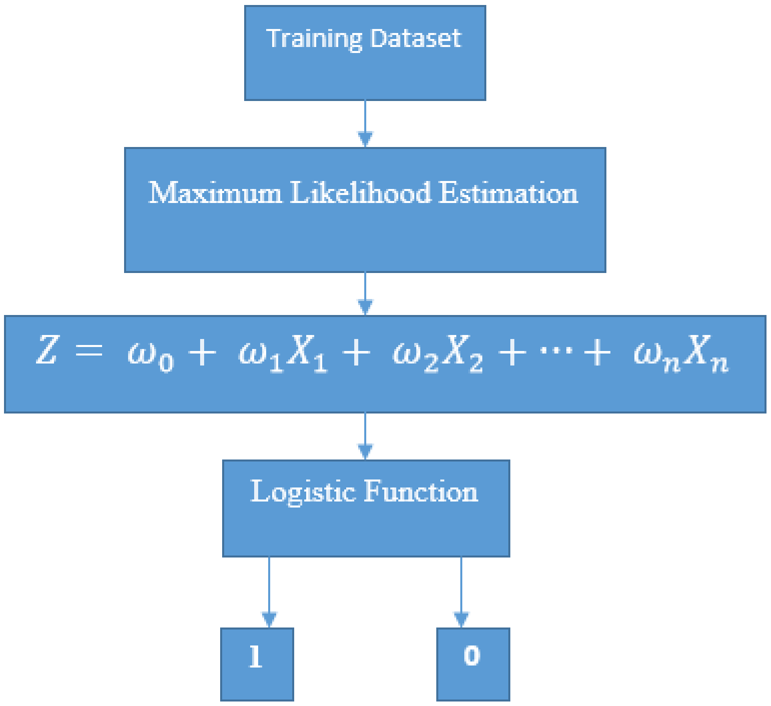

Training Dataset
Maximum Likelihood Estimation
$Z = \omega_0 + \omega_1 X_1 + \omega_2 X_2 + \cdots + \omega_n X_n$
Logistic Function
1
0

Figure 8: Logistic Regression Classifier

3.5 Overview of Attack detection

The proposed detection methodology is divided into two phases, the off-line training phase and on-line adaptation phase. In off-line training, we have collected dataset from various simulation in the SDN environment to train LR machine learning classifier and export a predictive model. In the on-line phase, we have added to the POX controller three modules: data collection, traffic prediction, and classifier retraining. In the next sections we will discuss in more details our proposed detection system.

3.5.1 Off-line Training Phase

The proposed detection system exploits the semi-supervised machine learning (ML) algorithm by using LR classifier. In semi-supervised ML algorithm, we will use two datasets to produce the predictive model. The first dataset is labeled and it will be used to train the LR classifier, we denoted it as the training dataset. The second dataset is also labeled and is used for validation, we denoted it as the validation dataset. The learning algorithm doesn't use the labels of the validation dataset, instead, the labels are used to validate the predicted output of the classifier. The details of creation for both datasets are provided in chapter 4.

In step 1 of the off-line training, the training dataset, whose content has the format as shown in Table 3, is loaded to the training algorithm. Every row in the training dataset is a datapoint consisting of four features (X1, X2, X3, or X4) and a label (Y). The features, as mentioned before, are total no. of Packets, total no. of Bytes, total no. of Flows, and total no. of One_Packet flows. The label has the value of either 1 or 0 (attack or benign).

X1 Packets	**X2 Bytes**	**X3 Flows**	**X4 One_Packet**	**Y class**
358	26591	26	0	0
290	21540	44	17	0
212	13632	110	100	1
214	13790	111	101	1
202	12969	107	98	1

Table 3: Examples of Training Dataset

With the training dataset, the training algorithm computes a set of coefficients (ω) using maximum likelihood estimation.

In step 2, the coefficients derived in step 1 are used in eq. (3) to compute a value of Z for every datapoint from the validation dataset. From Z, we calculate the logistic function and thus, the probability of that datapoint belongs to class 1 using eq. (1). We can also compute the probability of that datapoint belongs to class 0 using equation (2). In addition of being classified, the newly labeled datapoint with high confidence will be added to the training dataset for future retraining.

Here we define a labeled datapoint that belongs to a given class C with high priority as the datapoint with high confidence. For example, given a datapoint X with

$$P(Y = C|X) \geq P_{th}$$

Where P_{th} is a predetermined threshold (in this thesis, P_{th} is set to 0.9, as described in chapter 4), then we will say that X belongs to class C with high confidence. The datapoints associated with low confidence will not be added to the training dataset.

The off-line mechanism will go through a number of iterations. In each iteration, it repeats steps 1 and 2, but with the updated training dataset which includes the previous training dataset plus the newly added labeled datapoints. At the end of step 2, the mechanism checks the attack detection accuracy. If the detection accuracy of the current iteration is within 0.5% of the previous iteration, then the mechanism concludes that the iteration converges.

Figure 9 below shows the flow chart of proposed off-line training mechanism.

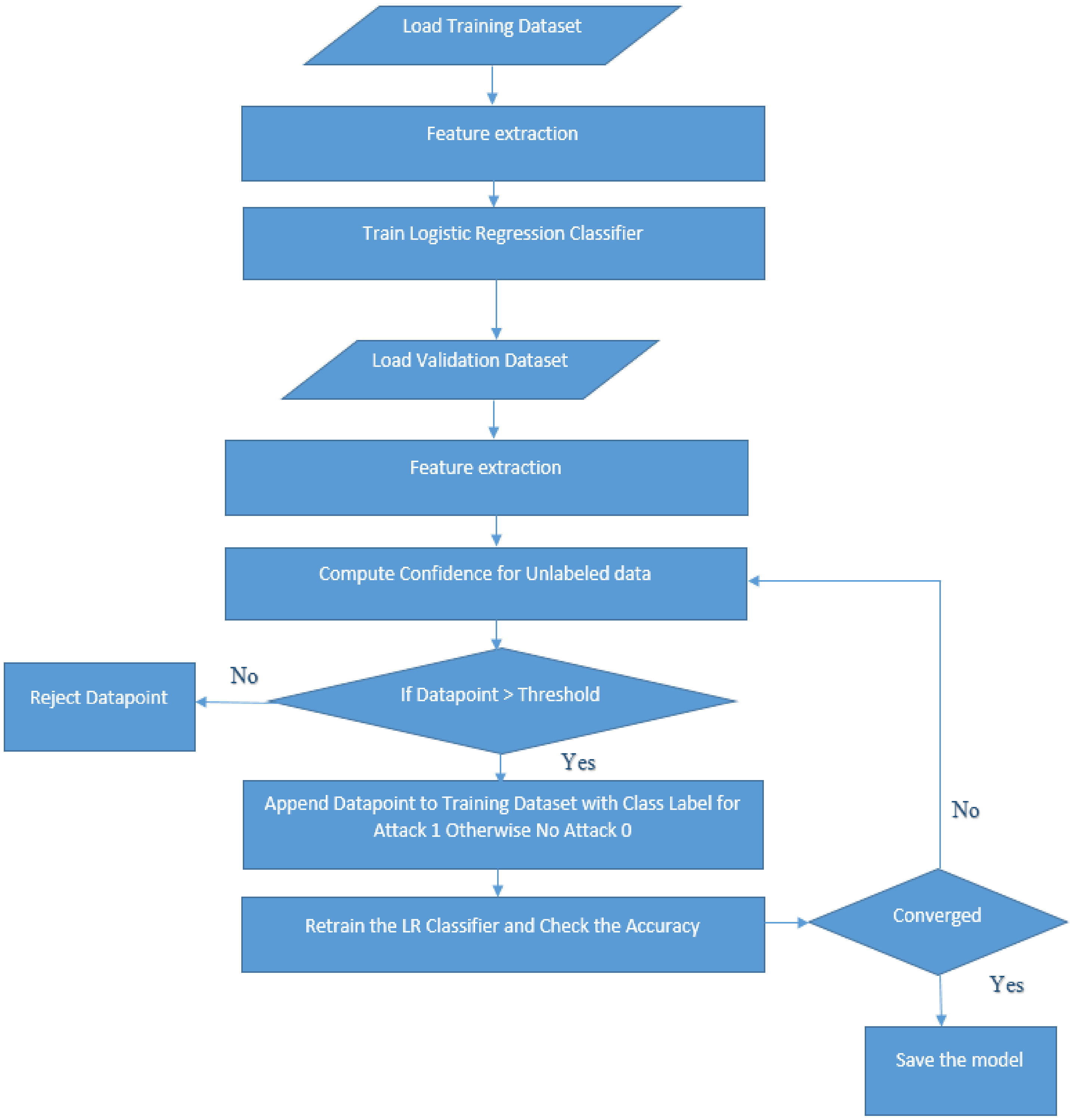

Load Training Dataset
Feature extraction
Train Logistic Regression Classifier
Load Validation Dataset
Feature extraction
Compute Confidence for Unlabeled data
If Datapoint > Threshold
No
Reject Datapoint
Yes
Append Datapoint to Training Dataset with Class Label for Attack 1 Otherwise No Attack 0
Retrain the LR Classifier and Check the Accuracy
Converged
No
Yes
Save the model

Figure 9: Offline Detection Flowchart

3.5.2 On-line detection phase

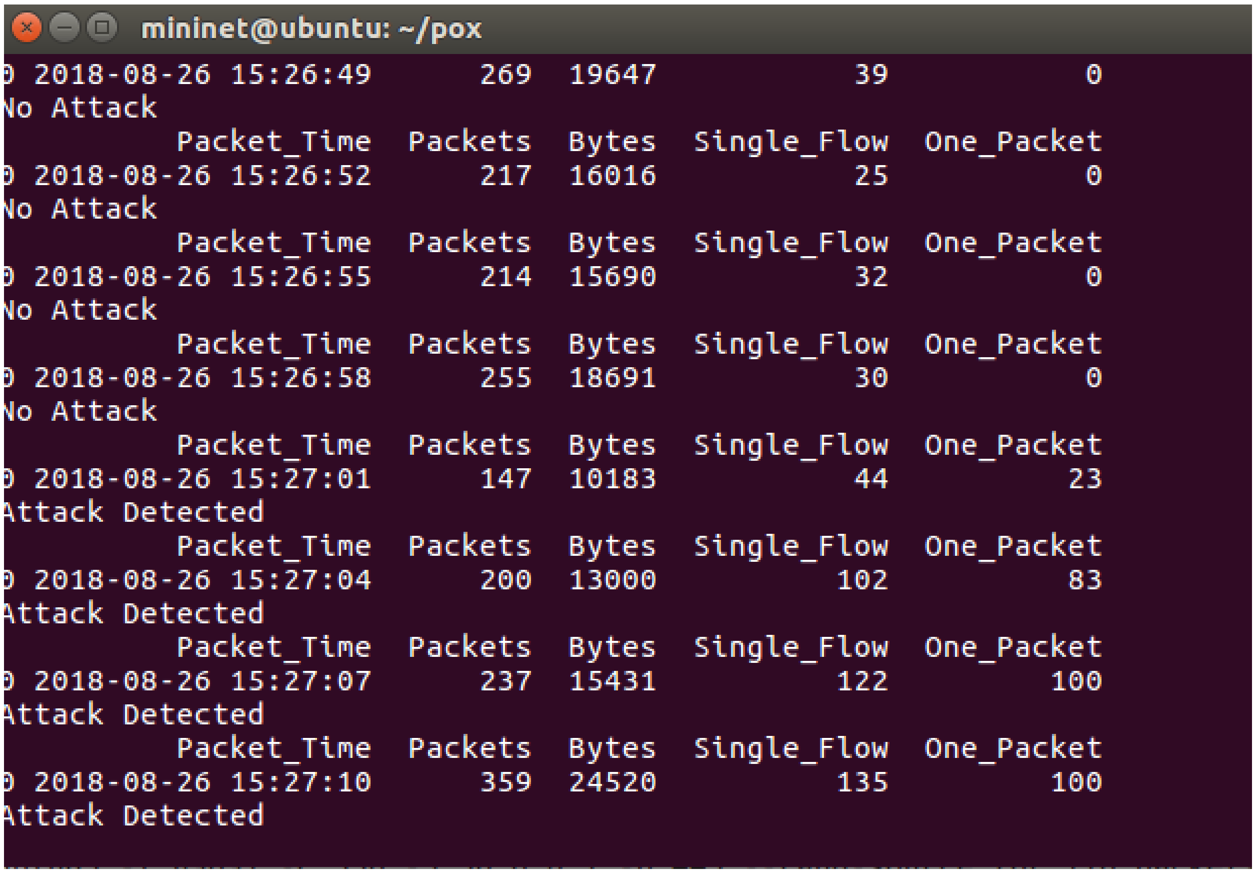
Our proposed on-line detection method is divided into three modules implemented within the POX SDN controller. These three modules are described below:

Traffic collector: The POX controller manages traffic of network switches in a centralized way, it has global view of all connected switches and has the ability to analyze their traffic from the point of view of a DDoS attack. The collection of flow entries from an Openflow switch is performed at predetermined time intervals by traffic controller module. From this collection, important features are extracted and used to classify traffic as normal or as an attack. The definition of the time interval to collect flow entries is of great importance. If collection is made at over a longtime intervals, then there will be a substantial delay to detect an attack and consequently a reduction of the time available for a possible mitigation. On the other hand, if the time interval for the collection is too short, there will be an increase of overhead caused by feature collection. In our scheme we use 3 seconds interval. The module collects these statistics every 3 seconds from OpenFlow switches through a secure channel and stores them into a database called Flow Database.

Traffic Predictor: In this module, for every three seconds a datapoint with the four extracted features (Packets, Bytes, Flows and One_Packet) are passed as the input to our predictive model obtained from the off-line training phase. The logistic functions (1) and (2) are computed. An attack is declared if $P(Y = 1|X) > 0.5$. Furthermore, the datapoint is inputted to the training dataset only if $P(Y = 1|X) > 0.9$. On the other hand, if $P(Y = 0|X) > 0.5$, then the system declares no attack. Similarly the datapoint is inputted to the training dataset only if $P(Y = 0|X) > 0.9$.

Classifier Retraining: In this module, semi-supervised algorithm will work in the background to retrain LR classifier using the same approach similar to step 2 of the off-line mechanism. The only difference is that the input datapoints are derived from the on-line traffic instead of the verification dataset. The module starts the retraining at the controller every 90 sec. To limit the size of the dataset, we delete the oldest data from previous simulations whenever the size of the data exceeds certain threshold.

Figure 10 below shows DDoS detection system implementation.



```
mininet@ubuntu: ~/pox
0 2018-08-26 15:26:49      269  19647          39           0
No Attack
          Packet_Time  Packets  Bytes  Single_Flow  One_Packet
0 2018-08-26 15:26:52      217  16016          25           0
No Attack
          Packet_Time  Packets  Bytes  Single_Flow  One_Packet
0 2018-08-26 15:26:55      214  15690          32           0
No Attack
          Packet_Time  Packets  Bytes  Single_Flow  One_Packet
0 2018-08-26 15:26:58      255  18691          30           0
No Attack
          Packet_Time  Packets  Bytes  Single_Flow  One_Packet
0 2018-08-26 15:27:01      147  10183          44          23
Attack Detected
          Packet_Time  Packets  Bytes  Single_Flow  One_Packet
0 2018-08-26 15:27:04      200  13000         102          83
Attack Detected
          Packet_Time  Packets  Bytes  Single_Flow  One_Packet
0 2018-08-26 15:27:07      237  15431         122         100
Attack Detected
          Packet_Time  Packets  Bytes  Single_Flow  One_Packet
0 2018-08-26 15:27:10      359  24520         135         100
Attack Detected
```

Figure 10: DDoS Detection System Implementation

3.6 Chapter summary

In this chapter, we discussed our proposed method which utilizes Logistic Regression classifier based on semi supervised learning to detect DDoS attack. We employed four features (Packets, Bytes, Flows and One_Packet) to train our LR classifier off-line. The predictive model produced from the off-line training is then used in the on-line attack detection in SDN controller, and at the same time, continue to classify the incoming traffic to update the training dataset for the retraining of our classifier in SDN background.

In the next chapter, results of the proposed DDoS detection method will be evaluated.

Chapter 4

4 Simulations and Performance Analysis

In this chapter, we will conduct a comprehensive simulation study to evaluate our proposed semi-supervised machine learning algorithm, later we will compare our approach to the statistical approach from Dhaliwal et al. [16].

4.1 Experimental environment

To obtain the dataset and test the performances of our proposed DDoS attack detection system, a virtual network is constructed using the same experiment and setting used for [16]. In order to make a fair comparison, we have setup the same simulation environment using the same network tools as in [16]. In the following subsections, we give a brief introduction of these tools.

4.1.1 POX controller

The POX controller allows an easy way to implement OpenFlow/SDN experiments [33]. It is a Python-based SDN controller. POX comes with a number of switch learning modules. The one that is used in our solution is L2_multi module.

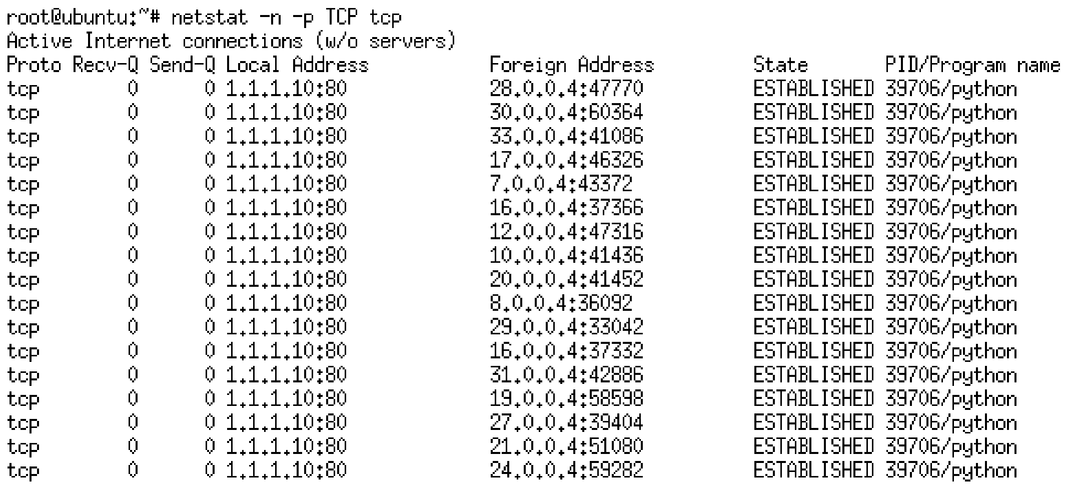
4.1.2 Mininet

Mininet [34] is the network emulator that we used in our thesis to create a desirable underlying network. It is a popular network emulation tool for SDN research. In the Mininet environment, virtual hosts and switches can be created and interconnected to form any given network topology. This tool also has a Python API, which facilitates the creation of custom topologies and experiments, providing a powerful method to rapidly prototype a network.

4.1.3 Network Traffic Generator

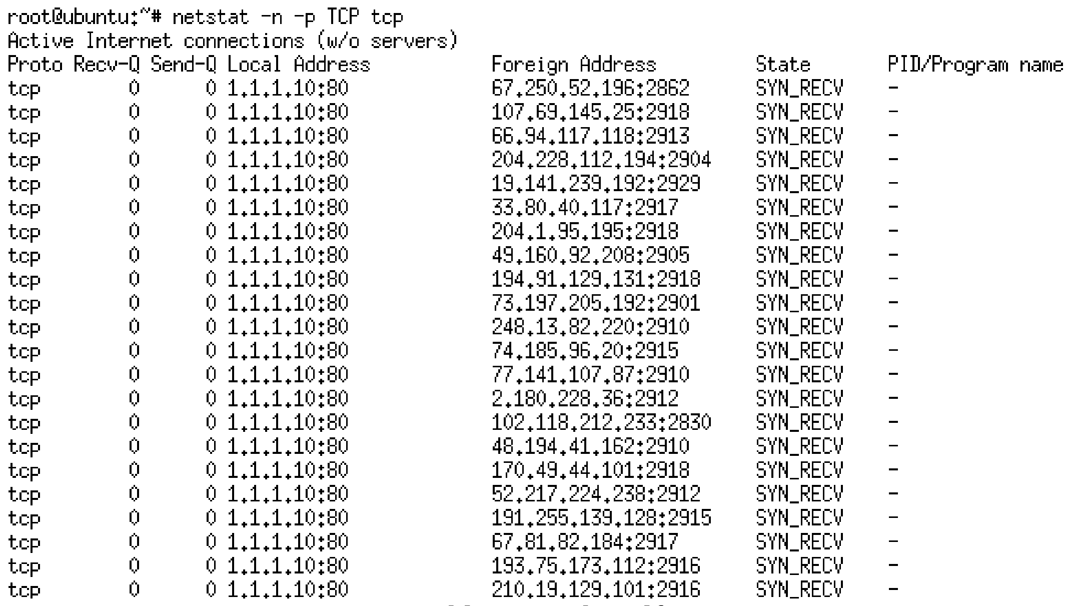
Two python scripts were created to generate benign TCP SYN traffic based on client-server socket programming, one runs on the client and the other on the server [16]. The time durations to open the socket connection and send get-request packet are randomly generated. Figure 11 shows the state of TCP connections in web server backlog 1.1.1.10 (Local Address) on port 80 and no. of benign hosts (Foreign Address) in an established state, which means that the connection is fully established and 3 way-handshake between hosts and server have been completed.

For SYN flood attacks we used Hping3 [40] tool to generate attack traffic. We have created an attacker python script which ran Hping3 tool with various network parameters such as number of packets, interval duration, and random source-destination IP addresses. Figure 12 illustrates the connection states at web server backlog where a large number of state connections from random IP addresses in a SYN_RECV state. The SYN_RECV state indicates that the connection is only half-open and the legitimacy of the requested host is still not completed.



```
root@ubuntu:~# netstat -n -p TCP tcp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 1.1.1.10:80             28.0.0.4:47770          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             30.0.0.4:60364          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             33.0.0.4:41086          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             17.0.0.4:46326          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             7.0.0.4:43372           ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             16.0.0.4:37366          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             12.0.0.4:47316          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             10.0.0.4:41436          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             20.0.0.4:41452          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             8.0.0.4:36092           ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             29.0.0.4:33042          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             16.0.0.4:37332          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             31.0.0.4:42886          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             19.0.0.4:58598          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             27.0.0.4:39404          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             21.0.0.4:51080          ESTABLISHED 39706/python
tcp        0      0 1.1.1.10:80             24.0.0.4:59282          ESTABLISHED 39706/python
```

Figure 11: Server backlog Benign Connections



```
root@ubuntu:~# netstat -n -p TCP tcp
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 1.1.1.10:80             67.250.52.196:2862      SYN_RECV    -
tcp        0      0 1.1.1.10:80             107.69.145.25:2918      SYN_RECV    -
tcp        0      0 1.1.1.10:80             66.94.117.118:2913      SYN_RECV    -
tcp        0      0 1.1.1.10:80             204.228.112.194:2904    SYN_RECV    -
tcp        0      0 1.1.1.10:80             19.141.239.192:2929     SYN_RECV    -
tcp        0      0 1.1.1.10:80             33.80.40.117:2917       SYN_RECV    -
tcp        0      0 1.1.1.10:80             204.1.95.195:2918       SYN_RECV    -
tcp        0      0 1.1.1.10:80             49.160.92.208:2905      SYN_RECV    -
tcp        0      0 1.1.1.10:80             194.91.129.131:2918     SYN_RECV    -
tcp        0      0 1.1.1.10:80             73.197.205.192:2901     SYN_RECV    -
tcp        0      0 1.1.1.10:80             248.13.82.220:2910      SYN_RECV    -
tcp        0      0 1.1.1.10:80             74.185.96.20:2915       SYN_RECV    -
tcp        0      0 1.1.1.10:80             77.141.107.87:2910      SYN_RECV    -
tcp        0      0 1.1.1.10:80             2.180.228.36:2912       SYN_RECV    -
tcp        0      0 1.1.1.10:80             102.118.212.233:2830    SYN_RECV    -
tcp        0      0 1.1.1.10:80             48.194.41.162:2910      SYN_RECV    -
tcp        0      0 1.1.1.10:80             170.49.44.101:2918      SYN_RECV    -
tcp        0      0 1.1.1.10:80             52.217.224.238:2912     SYN_RECV    -
tcp        0      0 1.1.1.10:80             191.255.139.128:2915    SYN_RECV    -
tcp        0      0 1.1.1.10:80             67.81.82.184:2917       SYN_RECV    -
tcp        0      0 1.1.1.10:80             193.75.173.112:2916     SYN_RECV    -
tcp        0      0 1.1.1.10:80             210.19.129.101:2916     SYN_RECV    -
```

Figure 12: Server Backlog Attack Half-Open Connections

4.1.4 Network Setup

The experiment is done on a Dell laptop Intel® Quad Core i7-3740QM CPU, 2.70 GHz with 16.0 GB RAM. The operating system is Linux Ubuntu 16.04 LTS and Mininet version 2.3.0d1.

Using Mininet python API, a custom topology is created for our experiments network topology as shown in figure 13. The network consists of two switches, 57 hosts and a controller as shown in figure 13. There are 49 benign hosts, each runs a python client script. They are divided into three groups: groups A, B and C. In addition, 7 hosts are setup as the attackers, each runs an Hping3 attacking tool. All the benign hosts send requests to the web server, which is connected to S1 inside the network. The web server is also the attack target. All the benign and attack hosts are located at the external network. The traffic generated by them goes through S2 to reach the web server.

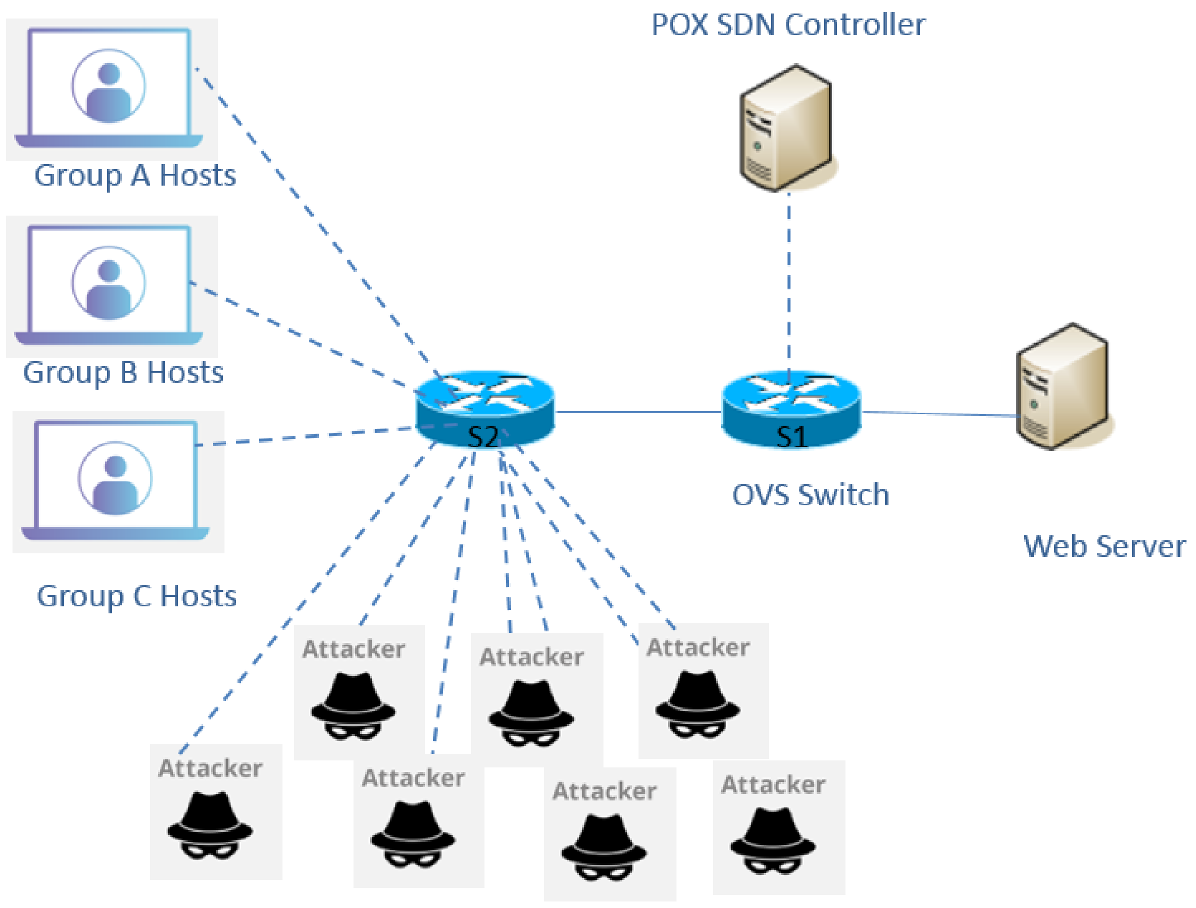


Figure 13: Network Topology

4.2 Off-line Training Implementation

The proposed semi-supervised algorithm is implemented using Python Scikit-learn open source machine learning libraries [41]. Scikit-learn is used in implementation of classification methods in Python. Thus our classifier LR can be easily integrated with POX controller as a module. To our best knowledge, currently there is no publicly available SDN traffic dataset in the research community. Therefore, we used private dataset to train our detection model and tested the DDoS defense through real-time DDoS attacks. In the subsequent sections, we will discuss the dataset generation.

4.2.1 Dataset

Network traffic dataset is the most important part in training our model. We run different scenarios using the network topology as shown in figure 13. We will use python client and server socket scripts to generate data flows with "realistic behaviors" at the network or transport level, mixed with attack traffic using python script that run Hping3 tool generate TCP SYN flood attack traffic.

4.2.2 Network Traffic Patterns

The traffic pattern in real life is non-stationary. To mimic this characteristic, we introduce 3 traffic groups, Group A, Group B and Group C. The clients of each group use the same traffic parameters as shown in table 4 to generate random traffic. To generate a non-stationary traffic pattern, Clients of different groups generate traffic at different periods as shown in figure 14.

In figure 14, the benign TCP traffic generation is divided into 5 periods. In the first period, only the clients of Group A generate web traffic to communicate with the server. In the second period, clients in both Groups A and B generate traffic. In the third period, the clients in Group B stop and clients in Group A continue. In the fourth period, the clients in Group C start along with those in Group A to send traffic to the web server. In the final period, Group A traffic generation stops and clients in Group C continue until the end of the experiment. The period of benign traffic generation is around three minutes.

In the attack traffic pattern, the attack is generated by Hping3 attack tool. The number of attacking hosts, which generate spoofed SYN packets, ranges from 2 to 7 attackers. By changing the number of attacking hosts, we can control the level of attack as shown in table 5. We repeat the same benign traffic scenario (Figure 13) mixed with the attack traffic in some periods for an additional 3 minutes. The total time for benign and attack traffic generation is around 6 minutes.

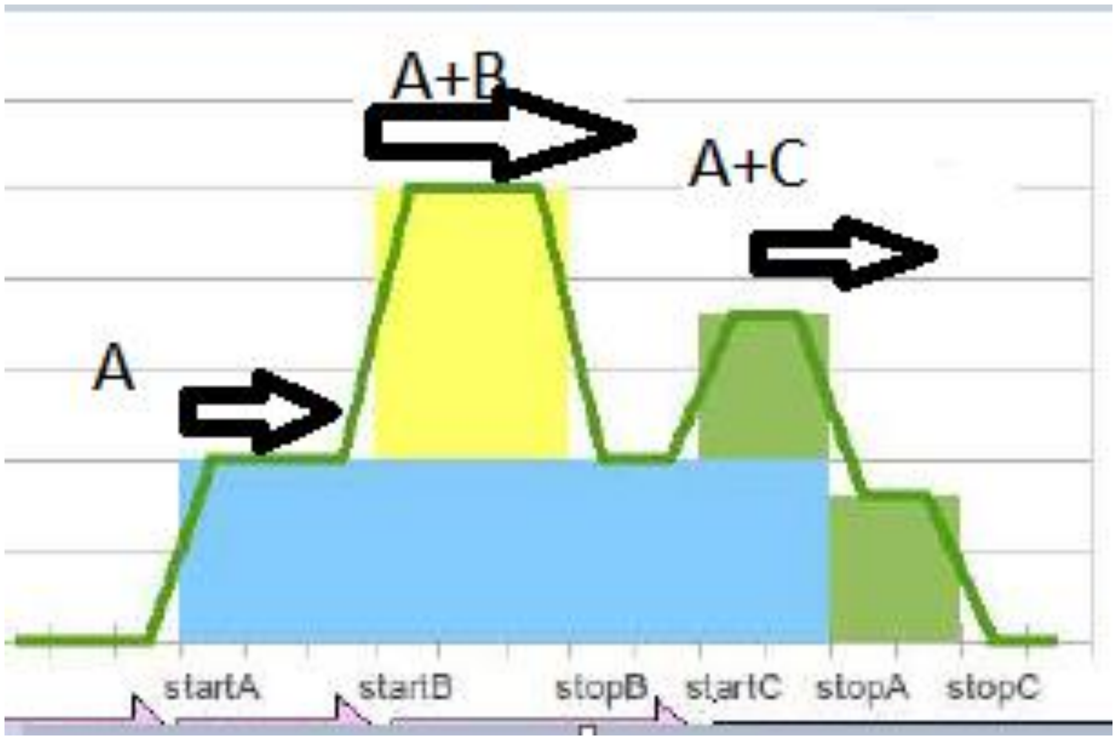


Figure 14: Network Traffic Pattern [42]

Parameters	**Group A**	**Group B**	**Group C**
No. of clients	14	20	15
Traffic type	HTTP	HTTP	HTTP
Avg. Inter-arrival time between connections	1 sec	1 sec	1 sec
No. of Connections per client	10	15	12
No. of Get requests per connection	6	8	7

Table 4: Benign Traffic Pattern

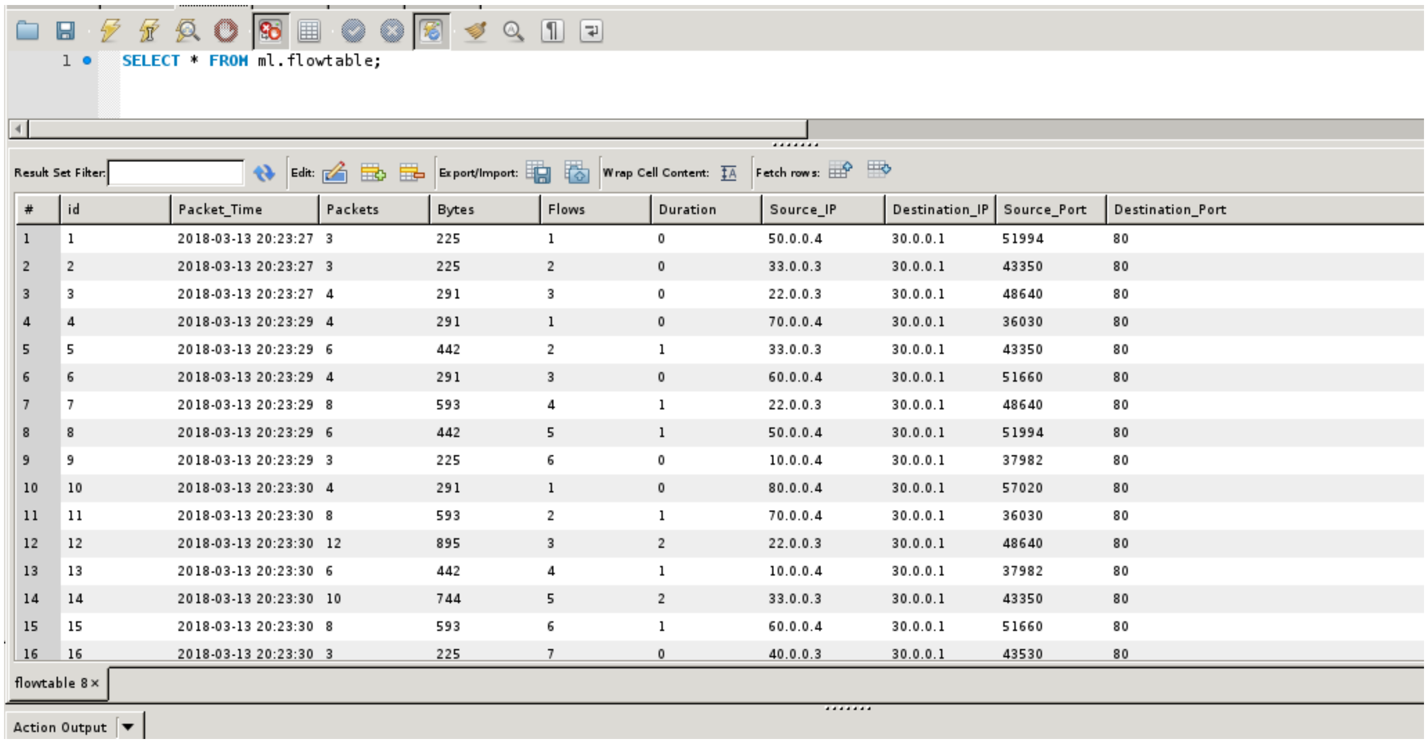
Attack Rate(pkts/sec)	Attack Type	No. of Attackers	Group A	Group B	Group C
20	TCP SYN	2	10	16	23
30	TCP SYN	3	10	16	23
40	TCP SYN	4	10	16	23
50	TCP SYN	5	10	16	23
60	TCP SYN	6	10	16	23
70	TCP SYN	7	10	16	23

Table 5: Attack Traffic Pattern

4.2.3 Network Traffic Statistics Collection

In order to feed our classifier with required network traffic data in offline and on-line mode, we have created module in the controller to collect network traffic statistics from Open Flow switches and store them into a Flow Database (FD). FD keeps network traffic statistics for active flows such as packet count, byte count, source IP address, destination IP address, source port and destination port. The traffic statistics are updated every 3 seconds. Figure 15 (a) below shows the statistics collected according to the traffic pattern as described in Figure 14.

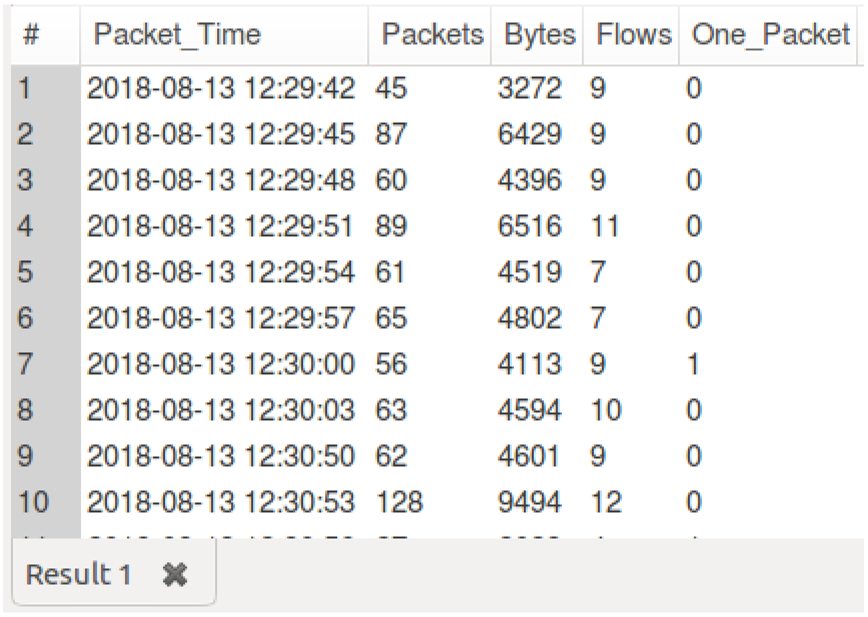
All traffic statistics are then filtered to extract the most important features that will be used as the training dataset for LR classifier to detect DDoS attack, Figure 15 (b) shows extracted features from traffic statistics collections. In our thesis, we employ four features which are Packets, Bytes, Single Flow and Single Packet as discussed in section 3.5.2. Figure 15 shows the traffic statistics collections and the most important features in one of our simulations run.



```
SELECT * FROM ml.flowtable;
```

#	id	Packet_Time	Packets	Bytes	Flows	Duration	Source_IP	Destination_IP	Source_Port	Destination_Port
1	1	2018-03-13 20:23:27	3	225	1	0	50.0.0.4	30.0.0.1	51994	80
2	2	2018-03-13 20:23:27	3	225	2	0	33.0.0.3	30.0.0.1	43350	80
3	3	2018-03-13 20:23:27	4	291	3	0	22.0.0.3	30.0.0.1	48640	80
4	4	2018-03-13 20:23:29	4	291	1	0	70.0.0.4	30.0.0.1	36030	80
5	5	2018-03-13 20:23:29	6	442	2	1	33.0.0.3	30.0.0.1	43350	80
6	6	2018-03-13 20:23:29	4	291	3	0	60.0.0.4	30.0.0.1	51660	80
7	7	2018-03-13 20:23:29	8	593	4	1	22.0.0.3	30.0.0.1	48640	80
8	8	2018-03-13 20:23:29	6	442	5	1	50.0.0.4	30.0.0.1	51994	80
9	9	2018-03-13 20:23:29	3	225	6	0	10.0.0.4	30.0.0.1	37982	80
10	10	2018-03-13 20:23:30	4	291	1	0	80.0.0.4	30.0.0.1	57020	80
11	11	2018-03-13 20:23:30	8	593	2	1	70.0.0.4	30.0.0.1	36030	80
12	12	2018-03-13 20:23:30	12	895	3	2	22.0.0.3	30.0.0.1	48640	80
13	13	2018-03-13 20:23:30	6	442	4	1	10.0.0.4	30.0.0.1	37982	80
14	14	2018-03-13 20:23:30	10	744	5	2	33.0.0.3	30.0.0.1	43350	80
15	15	2018-03-13 20:23:30	8	593	6	1	60.0.0.4	30.0.0.1	51660	80
16	16	2018-03-13 20:23:30	3	225	7	0	40.0.0.3	30.0.0.1	43530	80

(a) Traffic Statistics Collection



#	Packet_Time	Packets	Bytes	Flows	One_Packet
1	2018-08-13 12:29:42	45	3272	9	0
2	2018-08-13 12:29:45	87	6429	9	0
3	2018-08-13 12:29:48	60	4396	9	0
4	2018-08-13 12:29:51	89	6516	11	0
5	2018-08-13 12:29:54	61	4519	7	0
6	2018-08-13 12:29:57	65	4802	7	0
7	2018-08-13 12:30:00	56	4113	9	1
8	2018-08-13 12:30:03	63	4594	10	0
9	2018-08-13 12:30:50	62	4601	9	0
10	2018-08-13 12:30:53	128	9494	12	0

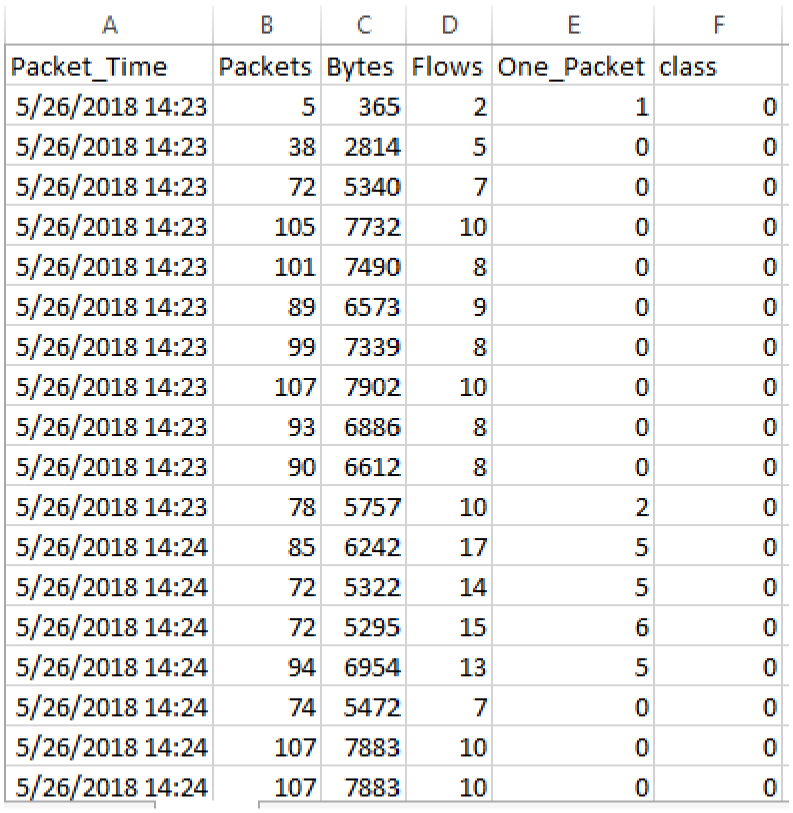
(b) Feature Extraction

Figure 15: Traffic Statistics Collection and Feature Extraction

4.2.4 Dataset Preparation

We have conducted 7 simulation scenarios using the network traffic patterns described in the previous section. For each scenario we recorded in FD (Flow Database) around 6 minutes of normal traffic mixed with attack traffic. After each scenario, we have extracted the required feature and labeled datapoint according to the timestamp. Then we have exported each dataset in CSV file format as illustrated in Figure 6. Finally, we have combined two simulation scenarios which last for 12 minute in one dataset CSV file and the remaining 5 scenarios which last for 30 minutes in separate dataset CSV file.

Figure 16 below shows part of the dataset which consists of 4 extracted features: Packets, Bytes, Flows and One_Packet. The last column of the dataset is the class label which is either 0 for benign traffic or 1 for attack traffic. We divided our dataset into two datasets. The first dataset is labeled and it will be used to train the LR classifier. The second dataset is also labeled but it is used for validation. The semi-supervised learning algorithm doesn't use the labels of the validation dataset, instead, the labels are used to validate the predicted output of the classifier.



A	B	C	D	E	F
Packet_Time	Packets	Bytes	Flows	One_Packet	class
5/26/2018 14:23	5	365	2	1	0
5/26/2018 14:23	38	2814	5	0	0
5/26/2018 14:23	72	5340	7	0	0
5/26/2018 14:23	105	7732	10	0	0
5/26/2018 14:23	101	7490	8	0	0
5/26/2018 14:23	89	6573	9	0	0
5/26/2018 14:23	99	7339	8	0	0
5/26/2018 14:23	107	7902	10	0	0
5/26/2018 14:23	93	6886	8	0	0
5/26/2018 14:23	90	6612	8	0	0
5/26/2018 14:23	78	5757	10	2	0
5/26/2018 14:24	85	6242	17	5	0
5/26/2018 14:24	72	5322	14	5	0
5/26/2018 14:24	72	5295	15	6	0
5/26/2018 14:24	94	6954	13	5	0
5/26/2018 14:24	74	5472	7	0	0
5/26/2018 14:24	107	7883	10	0	0
5/26/2018 14:24	107	7883	10	0	0

Figure 16: Generated Dataset

4.3 Evaluation Metrics

In general, the performance of our proposed detection system is evaluated in terms of accuracy (AC), False Positive (FP), False Negative (FN). A good detection requires high accuracy and low no. of false positive and false negative. A confusion matrix is used to calculate these parameters. In the confusion matrix, True Positive (TP) is the number of attacks recorded correctly by classifier. True Negative (TN) is the number of times the normal traffic recorded correctly by classifier. False Positive (FP) is the number of times the normal traffic was classified as attack traffic. False Negative (FN) is the number of times attack traffic were classified as normal traffic.
Accuracy (AC): Accuracy rate = No. of correct Predictions / Total No. of Predictions
Which is,

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \qquad (1)$$

In the subsequent sections, we will present the results of off-line and on-line detection results based on FP, FN, and Accuracy Rate.

4.4 Off-line Training Results

The performance of the semi-supervised ML algorithm depends on the value of confidence threshold. The key to our algorithm working successfully is the accurate selection of threshold. We selected the threshold empirically by optimizing the accuracy on the validation dataset. We add datapoints with high confident to the labeled dataset, then we use them to retrain our classifier and get our final model.

With the training dataset, we started to train the LR classifier using maximum likelihood estimation. Then, the model is optimized by finding the desirable threshold which leads to optimal classification accuracies on the validation dataset. To find the desirable threshold, we validate our semi-supervised algorithm with four different confidence thresholds: 60%, 70%, 80%, and 90%. Based on the confidence threshold, we add the datapoints with high confidence to the training dataset, otherwise we reject the less confident datapoints. We retrain classifier on previously labeled dataset plus the newly added labeled datapoints. For each iteration, we calculate the accuracy. We found that the accuracy converges within 8 iterations.

Table 6 shows the results of accuracy for confidence threshold based on the no. of iteration. We found that lower threshold of 60% or 70% will results in decline in the accuracy for our model. On the other hand, we obtained higher accuracy for the threshold of 80 % or 90 %. In our experiments, we will employ 90 %. The advantage of choosing a 90% threshold is that the system will admit lesser number of datapoints to the training dataset. The disadvantage is that the convergence is slower.

Threshold	**Iteration 1**	**Iteration 2**	**Iteration 3**	**Iteration 4**	**Iteration 5**	**Iteration 6**	**Iteration 7**	**Iteration 8**
90	91.53%	94.35%	96.02%	96.40%	97.43%	97.81%	98.71%	99.10%
80	91.53%	95.77%	96.66%	98.72%	99.10%	99.10%	99.10%	99.10%
70	91.53%	95.89%	97.81%	98.84%	99.10%	99.10%	98.97%	98.97%
60	91.53%	96.02%	98.33%	98.97%	99.10%	99.10%	98.79%	98.84%

Table 6: Comparison of 4 different Confidence threshold

4.5 On-line Detection Results

In order to evaluate our proposed model, we compare the performances of our approach with the approach from [16]. The test environment is shown previously in figure 13. In the next sections we will discuss two scenarios with two different characteristics of the network traffic patterns as follow:

4.5.1 Benign Traffic Pattern 1

In this scenario, we have tried to make the benign network traffic pattern realistic as possible. A python client script is running on 45 hosts and a server script on a single host for at least three minutes of simulation. We divided these hosts to three groups, Group A (10 hosts), Group B (15 hosts), and Group C (20 hosts). Each group of hosts are sending multiple web connections to the server randomly at different times as shown in table 7. When the experiment begins, Group A hosts start sending traffic to the web server. After Group A finishes Group B starts sending traffic followed by Group C. Table 7 shows the traffic pattern “1” parameters.

Parameters	**Group A**	**Group B**	**Group C**
No. of clients	10	15	20
Traffic type	HTTP	HTTP	HTTP
No. of Connections per client	10	12	15
Avg. HTTP Connection duration time	8	13	10
Avg. Inter-arrival time between connections in sec.	1	1.2	1.3
No. of Get requests per connection	6	7	8

Table 7: Traffic Pattern 1

4.5.2 Traffic Pattern 2

In this scenario, both Group A and Group B hosts start sending traffic at the same time to generate flash crowd event followed by Group C at the end. We also increase the number of hosts in each group to 15, 20 and 25 with a total of 60 hosts. When simulation starts a total of 35 hosts starts sending traffic, while the remaining 25 hosts in group C are activated to send after hosts in Group A and B finish sending. Table 8 shows the traffic pattern “2” parameters.

Parameters	Group A	Group B	Group C
No. of clients	15	20	25
Traffic type	HTTP	HTTP	HTTP
No. of Connections per client	8	10	12
Avg. HTTP Connection duration time.	8 sec.	13 sec.	10 sec.
Avg. Inter-arrival time between connections in sec.	0.5 sec.	0.7 sec.	0.9 sec.
No. of Get requests per connection	5	6	7

Table 8: Traffic Pattern 2

4.5.3 Attack Traffic

In order to generate TCP SYN DDoS attack traffic mixed with normal traffic pattern "1" in first scenario, Hping3 tool is used at two attacker hosts to generate constant low rate attack traffic with random source IP spoofing. Both attackers start the attack after 60 seconds in all the simulations for one minute. We issued the following hping3 command on two attacker hosts:

```
hping3 -i u200000 -S '1.1.1.10 -p 80 --rand-source
```

The above command sends 5 pkts/sec SYN packets, which will be 10 pkts/sec in total from both attacking machines, to the address of web server ('1.1.1.10) on port 80 with random source address, where –i is interval wait uX for X microseconds. In our setup, –i u200000, and so each machine generates 5 packets per second.

To generate constant low rate attack traffic mixed with normal traffic pattern "2" in second scenario, following hping3 command is used on two attacker hosts:

```
hping3 -i u100000 -S '1.1.1.10 -p 80 --rand-source
```

The above command sends 10 pkts/sec SYN packets, which will be 20 pkts/sec in total from both attacking machines.

For each scenario 50 simulations runs are performed. Each simulation lasts around 3 minutes. While the simulation is running if an attack is suspected, an attack alarm is raised.

4.5 Results

The table below summarizes the number of False Positive (FP) and False Negative (FN) reports on while traffic patterns “1” and “2” are tested.

	Traffic Pattern 1		Traffic Pattern 2	
No. of Simulations	50		50	
	Error Counts	Error Probability	Error Counts	Error Probability
FP	1	2%	3	6%
FN	0	0%	0	0%
Algorithm Detection Rate	98%		94%	
Algorithm Failure Rate	2%		6%	

Table 9: FP and FN detection rate reports under different traffic patterns

As shown in table 9, the results obtained in the traffic pattern “1”, show a high detection rate of 98% and our approach demonstrates to perform very well in this scenario. The high rate of detection is due to confidence threshold that gives the ability of the algorithm to learn new traffic pattern. Although the detection rate in traffic pattern “2” declines a little, the semi-supervised algorithm adaptation helps to keep the detection rate at 94%.

4.6 Comparison between Statistical and Machine Learning Approach

In this section we will compare the performance of our proposed machine learning based DDoS Detection algorithm with the statistical approach of Dhaliwal et al [16].

In [16], the author proposed a DDoS attack detection and mitigation algorithm. He computed threshold using Exponential Weighted Moving Average (EWMA) and Exponential Weighted Moving Standard Deviation (EWMSTD) to detect SYN flood DDoS attacks. The detection algorithm run in three phases, each phase run in a 3-second window. In the first phase, the detection algorithm raises a warning when the rate of SYN packets that arrives at the controller exceeds the threshold. In the second phase, further analysis is done by comparing the source IP addresses of all the flows to a database that contains valid source IP addresses. Traffic flows that have the match are considered legitimate. In the third detection phase, the algorithm proceeds to computes the ratio of single-packet flows to the total number of flows and compares this ratio to a threshold.
The comparison is done in two scenarios. First scenario consists of traffic pattern “1” and false positive reports are computed. Similarly, in second scenario false positive reports are observed under traffic pattern “2”. For each scenario 50 simulations runs are performed. Each simulation lasts around 3 minutes.

4.7 Results

Figure 17 illustrates the first scenario when Traffic Pattern “1” was used with 50 simulations runs. In Our detection approach based on a one 3 seconds window, the number of false positive reports detected in our approach is 98%, which is slightly lower by 2% than Dhaliwal’s algorithm that does not generate any false positive reports with FPR 0 and detection accuracy of 100%. However, we have repeated the simulations with three consecutive windows similar to the statistical approach [16] and we have achieved 100 %.

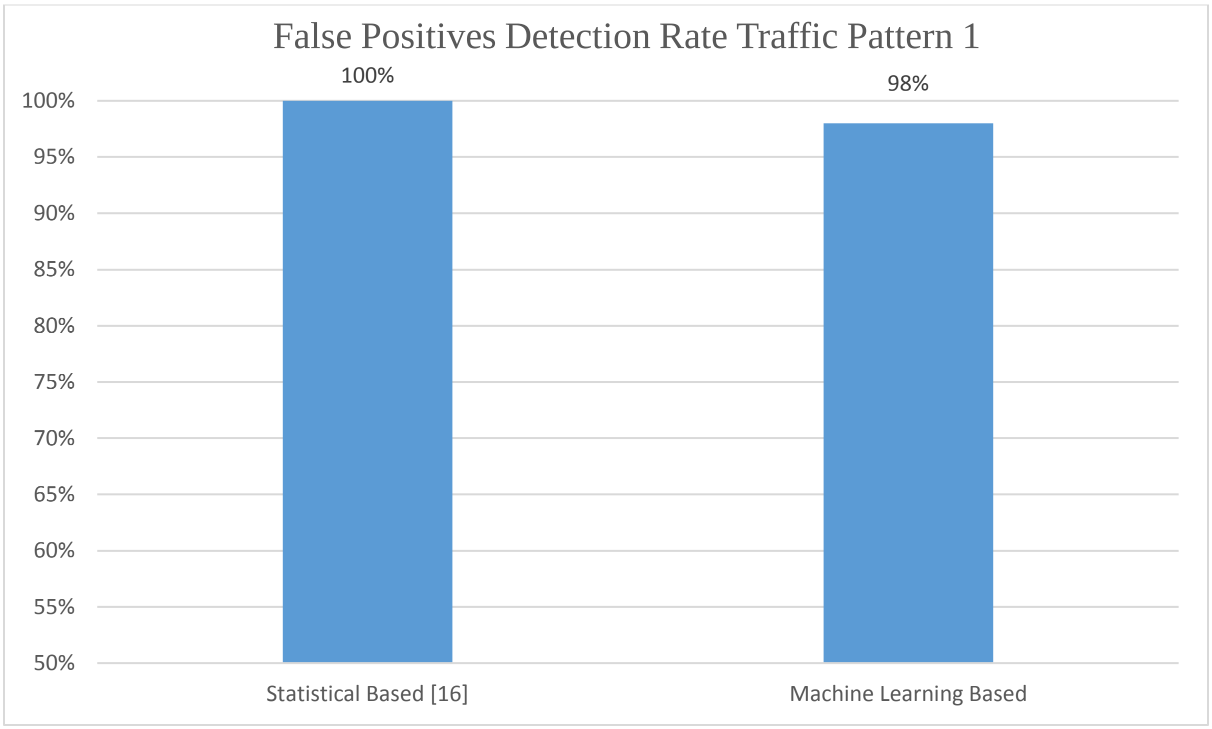


Figure 17: Traffic Pattern 1 Comparison Between Statistical and ML Approaches

The table below summarizes the results.

	Machine Learning Based		Statistical Based [16]	
No. of Simulations	50		50	
	Error Counts	Error Probability	Error Counts	Error Probability
FP	1	2%	0	0%
FN	0	0%	0	0%
Algorithm Detection Rate	98%		100%	
Algorithm Failure Rate	2%		0%	

Table 10: FP and FN Reports Comparison Between Statistical and ML Approaches Pattern 1

As shown in table 10, in traffic pattern “1”, our algorithm is able to perform with a 98% detection with one false positive report. However, in Dhaliwal’s approach the algorithm is able to perform with a 100% detection rate. Even though we are slightly lower in FPR detection rate, we use a 3 sec window to detect the attack. In contrast, work in [16] method introduces delay that is 3 times longer in attack detection. This is due to the fact that his method needs to examine the traffic in three consecutive windows (9 seconds) before an attack can be confirmed. This restriction would be ineffective in dealing with large attacks, which could overwhelm the network or controller in a very short time before the controller could take any countermeasures. We repeated the simulations using our approach with the modification that an attack is declared only if the datapoints in three consecutive windows are all tested attack positive. With this modification, our approach can also achieve 100 % FPR detection rate.

Figure 18 shows false positive reports for the second scenario when traffic Pattern “2” was used with 50 simulations runs. It is observed that our detection approach has an FPR of 94%. On the other hand, Dhaliwal’s approach has an FPR of only 84%. This clearly indicates that our proposed approach outperforms Dhaliwal’s approach in terms of false positive reports.

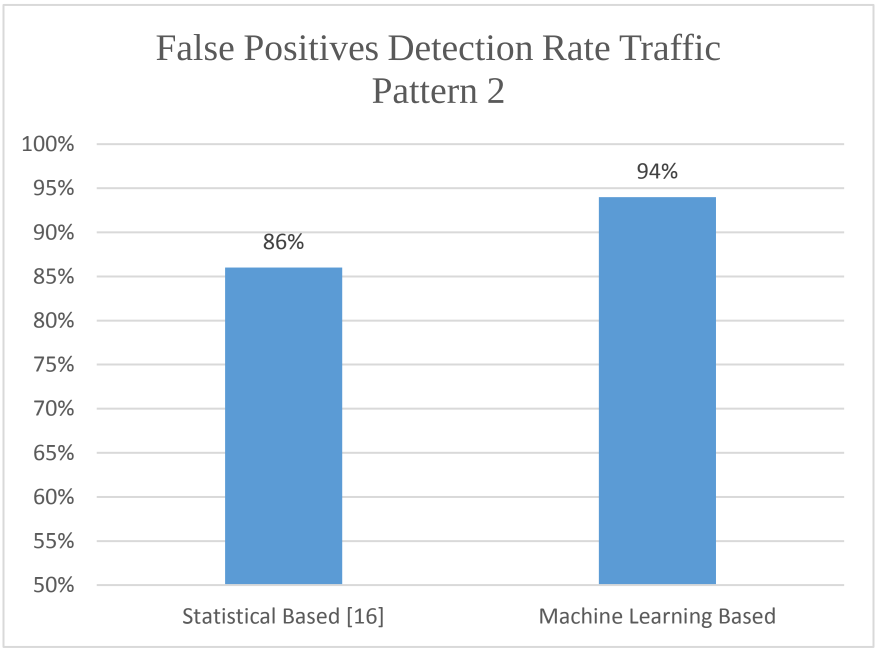


Figure 18: Traffic Pattern 2 Comparison Between statistical and ML Approaches

The table below summarizes the results.

	Machine Learning Based		Statistical Based [16]	
No. of Simulations	50		50	
	Error Counts	Error Probability	Error Counts	Error Probability
FP	3	6%	7	14%
FN	0	0%	0	0%
Algorithm Detection Rate	94%		86%	
Algorithm Failure Rate	6%		14%	

Table 11: FP and FN Reports Comparison Between Statistical and ML Approaches Pattern 2

As shows in table 11, for traffic pattern "2", our algorithm is able to perform with a 94% detection (3 false positive reports) rate; while Dhaliwal's approach has a 86% detection rate (7 false positive reports). It is because the flash crowd traffic is highly non-stationary. The statistical method in Dhaliwal's approach relies on mean and standard deviation for detection that cannot capture the non-stationary traffic pattern effectively. The machine learning algorithm in our approach can learn the non-stationary traffic pattern through off-line and on-line training. Consequently, our approach identifies flash crowd traffic much more accurately.

4.8 Chapter Summary and Discussion

The results in this chapter show how the proposed defense scheme effectively detects TCP SYN flood DDoS attacks in different traffic patterns. We summarize the results as follows.

- The results show that the proposed approach can detect different attack traffic pattern and detect the TCP SYN flood attack with high accuracy and with no false negative.
- The comparison between statistical approach from Dhaliwal's and our approach shows that our approach outperforms Dhaliwal's approach in terms of false positive detection rate, especially for flash crowd traffic.

Chapter 5

5 Conclusion and Future Work

5.1 Conclusion

In this thesis, we have employed the features of SDN architecture such as centralization and Programmability of the network to detect difficult type of DDoS attacks, TCP SYN flooding attack.

We have successfully implemented semi-supervised machine learning algorithm by using Logistic Regression classifier to detect TCP SYN flooding attack in real-time SDN environment. We have employed four important features (Packets, Bytes, Flows and One_Packet) to train our LR classifier off-line. The predictive model produced from off-line training has been used in on-line attack detection, and at the same time, continue to classify the incoming traffic to update the training dataset to retrain our classifier. Our results have clearly proved that our detection system is reliable with no false negative and very few false positive.

5.2 Future work

In our future work, we will develop a semi-supervised machine learning algorithm that is able to detect more types of DDoS attacks. In addition to, we will propose countermeasures to mitigate various types of attacks without the sacrifice of benign traffic. Since our work is done on simple testbed virtual environment, we will try in the future to cover more topologies to the performance of our proposed approach.

References

[1] opennetworking.org, “SDN Definition”, [Online]. Available: https://www.opennetworking.org. [Accessed July 2017].

[2] Diego Kreutz, Fernando M. V. Ramos, Paulo Verissimo,Christian Esteve Rothenberg Siamak Azodolmolky, Steve Uhlig, “Software-Defined Networking: A Comprehensive Survey”, Proceedings of the IEEE | Vol. 103, No. 1, January 2015

[3] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, Haiyong Xie, “A survey on Software-Defined Networking”, IEEE Communication Surveys & Tutorials, Vol. 17, No. 1, First Quarter 2015.

[4] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, Guofei Gu†, “AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks,” in Proc. ACM SIGSAC Conf. Comput. Commun. Security, Berlin, Germany, 2013, pp. 413–424.

[5] Moreno Ambrosin, Mauro Conti , Fabio De Gaspari, and Radha Poovendran, “Lineswitch: Efficiently managing switch flow in software-defined networking while effectively tackling dos attacks,” in Proc. 10th ACM Symp. Inf. Comput. Commun. Security, Singapore, 2015, pp. 639–644.

[6] Narmeen Zakaria Bawany, Jawwad A. Shamsi, Khaled Salah, “DDoS Attack Detection and Mitigation Using SDN Methods, Practices, and Solutions”, Arabian Journal for Science and Engineering, February 2017, Volume 42, Issue 2, pp 425–441

[7] Tommy Chin, Xenia Mountrouidou, Xiangyang Li, Kaiqi Xiong, “An SDN-Supported Collaborative Approach for DDoS Flooding Detection and Containment,” in Proc. IEEE Mil. Commun. Conf. (MILCOM), Tampa, FL, USA, 2015, pp. 659–664.

[8] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, Vijay Mann, “Sphinx: Detecting security attacks in software-defined networks,” in Proc. NDSS, San Diego, CA, USA, 2015.

[9] Reza Mohammadi, Reza Javidan, and Mauro Conti, “SLICOTS: An SDN-Based Lightweight Countermeasure for TCP SYN Flooding Attacks”, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, VOL. 14, NO. 2, JUNE 2017

[10] Nhu-Ngoc Dao, Junho Park, Minho Park, Sungrae Cho, "A Feasible Method to combat against DDoS Attack in SDN Network," International Conference on Information Networking (ICOIN), pp. 309-311, January 2015

[11] 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), pp. 239-250, 2015.

[12] Rui Wang, Zhiping Jia, Lei Ju, “An entropy-based distributed DDoS detection mechanism in software-defined networking”, In 2015 IEEE Trustcom/BigDataSE/ISPA, 2015 IEEE,vol 1, pp. 310–317 (2015)

[13] Seyed Mohammad Mousavi , Marc St-Hilaire, "Early Detection of DDoS Attacks against SDN Controllers," International Conference on Computing, Networking and Communications, Communications and Information Security Symposium, pp. 77-81, 2015.

[14] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, “OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch,” ACM SIGCOMM Computer Communication Review, vol. 44, no. 2, pp. 44–51, 2014.

[15] Julien Boite, Pierre-Alexis Nardin, Filippo Rebecchi, Mathieu Bouet, and Vania Conan, “StateSec: Stateful Monitoring for DDoS Protection in Software Defined Networks”, IEEE Conference on Network Softwarization (NetSoft), July 2017.

[16] Amandeep Singh Dhaliwal, “Detection and Mitigation of SYN and HTTP flood DDoS attacks in Software Defined Networks”, Master’s thesis, Ryerson university, Canada 2017.

[17] Merlin James Rukshan Dennis,"Machine-Learning and Statistical Methods for DDoS Attack Detection and Defense System in Software Defined Networks", Master’s thesis, Ryerson university, Canada 2017.

[18] Saurav Nanda, Faheem Zafari, Casimer DeCusatis, Eric Wedaa ,Baijian Yang, “Predicting network attack patterns in SDN using machine learning approach,” in Proc. IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN), pp. 167–172, Nov. 2016.

[19] P.MohanaPriya, S.Mercy Shalinie, “Restricted Boltzmann Machine based Detection System for DDoS attack in Software Defined Networks”, 4th International Conference on Signal Processing, Communications and Networking (ICSCN -2017), IEEE, 2017.

[20] Trung V. Phan, Truong Van Toan, Dang Van Tuyen, Truong Thu Huong , Nguyen Huu Thanh “OpenFlowSIA: An Optimized Protection Scheme for Software-Defined Networks from Flooding Attacks”, IEEE Sixth International Conference on Communications and Electronics (ICCE),pp. 13-18,2016.

[21] Javed Ashraf, Seemab Latif ,“Handling Intrusion and DDoS Attacks in Software Defined Networks Using Machine Learning Techniques Javed”, National Software Engineering Conference, IEEE, pp. 55 – 60,2014.

[22] Kübra Kalkan, Gürkan Gür, and Fatih Alagöz, "Defense Mechanisms against DDoS Attacks in SDN Environment," IEEE Communications Magazine, vol. 55, no. 9, pp. 175-179, 2017.

[23] Kokila RT', s. Thamarai Selvi', Kannan Govindarajan, “Ddos detection and analysis in sdn-based environment using support vector machine classifier,” in Advanced Computing (ICoAC), 2014 Sixth International Conference on, pp. 205–210, IEEE, 2014.

[24] Luca Boero, Mario Marchese, Sandro Zappatore, “Support vector machine meets software defined networking in ids domain,” in 2017 29th International Teletraffic Congress (ITC 29), vol. 3, pp. 25–30, 2017.

[25] Lohit Barki, Amrit Shidling, Nisharani Meti, Narayan D G, Mohammed Moin Mulla "Detection of distributed denial of service attacks in software defined networks". In IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2016 IEEE; pp. 2576-2581,2016.

[26] incapsula.com, “PING FLOOD(ICMPFLOOD)”,[Online].Available, http://www.incapsula.com/ddos/ attack-glossary, [Accessed July 2017].

[27] S.H.C. Haris, R.B. Ahmad, and M.A.H.A. Ghani, “Detecting TCP SYN Flood Attack based on Anomaly Detection,” Proceeding of The 2nd International Conference on Network Applications, Protocols and Services, pp. 240-244, 2010.

[28] I Gde Dharma N., M. Fiqri Muthohar, Alvin Prayuda J. D., Priagung K., Deokjai Choi, “Time-based DDoS detection and mitigation for SDN controller,” in Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific. IEEE,pp. 550–553, 2015.

[29] Vipin Gupta, Karamjeet Kaur, Sukhveer Kaur,” Network Programmability using Software Defined Networking”, 2016 International Conference on Computing for Sustainable Global Development (INDIACom),IEEE, pp 1170-1173,2016.

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

[31] Jacob H. Cox, Joaquin Chung, Sean Donovan, Jared Ivey, Russell J. Clark, George Riley, and Henry L. Owen,” Advancing Software-Defined Networks: A Survey”,IEEE Access, vol. 5, pp. 25487–25526, 2017.

[32] Qiao Yan, F. Richard Yu, “Distributed denial of service attacks in software-defined networking with cloud computing,” IEEE Communications Magazine, Vol.53, Issue: 4, pp: 52 - 59, 2015

[33] M. McCauley, “POX,” 2012. [Online]. Available: http://www.noxrepo.org/ [Accessed September 2017]

[34] Tom M. Mitchell (1997). Machine Learning. McGraw Hill. [Online]. Available :https://www.cs.ubbcluj.ro/ [Accessed August 2018]

[35] Aurélien Géron, (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow, USA: O'Reilly

[36] slideshare.net, "semi-supervised learning", [Online]. Available:https://www.slideshare.net/Dataiku/dataiku-hadoop-summit-semisupervised-learning-with-hadoop-for-understanding-user-web-behaviours [Accessed July 2018]

[37] Pu Wang, Shih-Chun Liny, and Min Luoz, "A Framework for QoS-aware Traffic Classification Using Semi-supervised Machine Learning in SDNs",2016 IEEE International Conference on Services Computing (SCC), pp. 760 - 765, 2016.

[38] David W. Hosmer and Stanley Lemeshow, Applied Logistic Regression, Wiley, 2nd edition, 2000.

[39] machinelearningmastery.com, "logistic-regression-for-machine-learning", [Online] Available: https://machinelearningmastery.com/ [Access June 2018]

[40] https://linux.die.net/man/8/hping3

[41] scikit-learn.org, "Sklearn", [Online]. Available: https://scikit-learn.org/. [Accessed Feb 2018].

[42] A. Varet and N. Larrieu, "How to generate realistic network traffic," in Proc. IEEE 38th Annual International Computers, Software & Applications Conference (COMPSAC), 2014.