SLOW DISTRIBUTED DENIAL OF SERVICE DETECTION AND MITIGATION IN SOFTWARE DEFINED NETWORKS USING SUPPORT VECTOR MACHINE AND SELECTIVE ADAPTIVE BUBBLE BURST ALGORITHM APPROACHES

BY

AKANJI, Oluwatobi Shadrach

MTECH/SICT/2018/8644

DEPARTMENT OF COMPUTER SCIENCE

FEDERAL UNIVERSITY OF TECHNOLOGY

MINNA

AUGUST, 2021

SLOW DISTRIBUTED DENIAL OF SERVICE DETECTION AND MITIGATION IN SOFTWARE DEFINED NETWORKS USING SUPPORT VECTOR MACHINE AND SELECTIVE ADAPTIVE BUBBLE BURST ALGORITHM APPROACHES

BY

AKANJI, Oluwatobi Shadrach

MTECH/SICT/2018/8644

A THESIS SUBMITTED TO THE POSTGRADUATE SCHOOL FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA, NIGERIA IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF MASTER OF TECHNOLOGY IN COMPUTER SCIENCE

AUGUST, 2021

ABSTRACT

Distributed Denial of Services (DDoS) has been used by attackers over the years to disrupt the availability of services in a networked environment. However, the increased attention in detecting and mitigating DDoS by security researchers has made attackers resort to an application layer attack known as slow DDoS which mimics the behaviour of a legitimate client using a slow connection or which has limited message window size thus making the attack difficult to detect. Although some researchers have examined the detection and mitigation of slow Hypertext Transfer Protocol (HTTP) DDoS, a form of slow DDoS, their research focused on either slow read or slow post and get attacks only without considering attack detection for the three types of slow HTTP DDoS. Furthermore, other researchers who have achieved competitive results in detecting slow read, post, and get attacks examined slow Denial of Service (DoS) attack which originates from one attacker. Since the slow DoS originates from an attacker, it is relatively easy to detect and, consequently, mitigate. Therefore, this research examined a machine learning-based slow HTTP DDoS detection and a Selective Adaptive Bubble Burst (SABB) mitigation of detected slow HTTP DDoS attacks, while considering slow read, post and get attacks in a Software-Defined Network (SDN) environment. The SDN environment was simulated in Graphical Network Simulator-3 (GNS3) where the Ryu controller was used to collect attack and benign Netflow flowsets for feature selection using Genetic Algorithm (GA) and attack detection using Radial Basis Function (RBF) kernel-based Support Vector Machine (SVM). Consequently, the trained SVM model was uploaded to the controller for real-time detection and activation of the SABB mitigation mechanism. Results obtained showed that the SVM classification of Netflow flowsets into attack and benign categories achieved an Area Under the Receiver Operating Characteristic Curve (AUC), accuracy, True Positive Rate (TPR), False Positive Rate (FPR), and False Negative Rate (FNR) of 99.89%, 99.89%, 99.95%, 0.18%, and 0.05% respectively. Furthermore, the SABB mitigation mechanism achieved an average response time and percentage of the completed request of 387.743 milliseconds (ms) and 92% respectively when eight slow HTTP DDoS attackers launched the assault compared to an average response time and percentage of the completed request of 1121.369 ms and 76% respectively when SABB was not utilized with the same number of attackers. The effectiveness of the SVM slow HTTP DDoS attack detection and the proposed SABB mitigation mechanism contributes to ongoing research into the use of SDN to enhance network security. Further studies into enhancing the average response time and the percentage of completed requests through a multi-controller SDN setup is recommended.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

Abbreviation	**Meaning**
5-NN	5-Nearest Neighbour
ABB	Adaptive Bubble Burst
ABC	Artificial Bee Colony
ANN	Artificial Neural Network
AUC	Area Under the Curve
CPU	Central Processing Unit
CSV	Comma-Separated Values
DDoS	Distributed Denial of Service
DNS	Domain Name System
DoS	Denial of Service
DT	Decision Trees
FTP	File Transfer Protocol
GA	Genetic Algorithm
GNS 3	Graphical Network Simulator-3
GOA	Grasshopper Optimization Algorithm
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol

IMAP	Internet Message Access Protocol
IP	Internet Protocol
KNN	K-Nearest Neighbour
LR	Logistics Regression
MLP	Multilayer Perceptron
NS3	Network Simulator-3
OMNeT++	Objective Modular Network Testbed in C++
OSI	Open Systems Interconnection
PSO	Particle Swarm Optimization
RBF	Radial Basis Function
RF	Random Forest
SABB	Selective Adaptive Bubble Burst
SDN	Software Defined Networks
SMTP	Simple Mail Transfer Protocol
SVM	Support Vector Machine
SYN	Synchronize
TCAM	Ternary Content Addressable Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

CHAPTER ONE

1.0 INTRODUCTION

1.1 Background to the Study

Digital networks are formed when a group of digital devices are connected through a medium that enables them to communicate using defined protocols. On such networks, valuable information is resident and essential services are rendered to various clients who need that information or services to complete diverse tasks. The valuable information and essential services provided in the network are considered the assets of the network. Just as legitimate clients need access to such services, illegitimate clients seek to either gain access or deny the legitimate clients access to the services on the network (Jaafar *et al.*, 2019). Among the numerous schemes used by attackers to deny legitimate clients access to the services provided in a network is the use of a Distributed Denial of Service (DDoS) attack because of the difficulty to trace and stop a DDoS attack.

Distributed Denial of Service (DDoS), also known as flooding attacks, refers to an attack on an asset or assets within a network that seeks to exhaust the limited resources of the asset or assets by sending requests from several other network devices thereby overwhelming the target's capacity to respond to the malicious requests or any other request (Rawat & Reddy, 2017; Swami *et al.*, 2019a). The need to launch DDoS attacks which overwhelms the target swiftly has made volumetric DDoS the attack of choice among malicious network users. Most volumetric DDoS attacks target the network layer of the Open Systems Interconnection (OSI) model. Next to volumetric DDoS attacks which operates at the network layer of the OSI model is the application layer DDoS attack of which Hypertext Transfer Protocol (HTTP) and Simple Mail Transfer Protocol (SMTP) are the prominent application layer protocols being attacked in this category (Swami *et al.*, 2019a). As attackers establish new ways of launching volumetric DDoS attacks, security researchers have been able to keep up with the trend and detect those attacks because the detection of volumetric DDoS attacks is relatively easy since volumetric attacks involve sending large requests from several devices simultaneously (Cambiaso *et al.*, 2013). Due to the pace with which security researchers have been able to identify the gimmicks of volumetric DDoS attackers and since the goal of the attackers is the disruption of service availability, the attackers now resort to launching slow DDoS attacks which are hard to detect and mitigate. The difficulty in detecting and mitigating slow DDoS attacks stems from the attack's similarity to requests from slow legitimate clients as opposed to the behaviour exhibited by volumetric attacks (Cambiaso *et al.*, 2017; Jaafar *et al.*, 2019; Muraleedharan & Barnabas, 2018).

Slow DDoS is a low-rate attack which is mostly application layer attacks that are difficult to detect because it behaves like legitimate traffic sent over a slow connection or from a legitimately slow client (Cambiaso *et al*., 2013; Dhanapal & Nithyanandam, 2019; Suroto, 2017; Swami *et al*., 2019b). Since the goal of a slow DDoS attack is to cause the unavailability of the targeted service, the attacker seizes available connections at the application layer which results in a busy service queue where legitimate traffic is dropped. Furthermore, the attacker sustains the assault by sending data at a low rate per unit time to keep the seized connections for as long as possible (Cambiaso *et al*., 2013). Since these attacks affect the application layer, protocols such as HTTP, SMTP, Internet Message Access Protocol (IMAP), and File Transfer Protocol (FTP) are also affected (Swami *et al*., 2019b). Although launching a slow DDoS attack requires less bandwidth and resources, it is capable of causing service disruption on a large scale (Cambiaso *et al.*, 2013; Shtern *et al.*, 2014; Suroto, 2017). Given the wide spread use of web servers to provide hypermedia, slow DDoS attackers have focused on attacking the application layer of webservers by disrupting the operations of the HTTP using an attack known as slow

HTTP DDoS which causes the web server to be unable to respond to HTTP requests. Slow HTTP DDoS are DDoS attacks launched against web servers by sending data or forcing the server to send data in a manner that prolongs the connection requesting for the resource thus denying legitimate clients access (Park, 2015). A slow HTTP attack is launched after a Transmission Control Protocol (TCP) connection has been established with the web server (Idhammad *et al.*, 2018; Tayama & Tanaka, 2018). Based on the operations obtainable in an HTTP request, there are three major types of slow HTTP DDoS attacks namely: slow HTTP get, slow HTTP post, and slow read attack (Yevsieieva & Helalat, 2017).

The slow HTTP get attack sends HTTP get requests to a victim server without transmitting two carriage return and line feed characters to denote the end of the request thus forcing the victim not to start processing until a complete header is received (Muraleedharan & Barnabas, 2018; Suroto, 2017; Idhammad *et al*., 2018). Similarly, slow HTTP post attacks take advantage of the adherence of web servers to the volume of data defined in the *Content-Length* field by advertising a large value and then sending the post message in several packets at a low rate to the server (Idhammad *et al*., 2018; Swami *et al.*, 2019a). Unlike the other two slow HTTP attacks, a slow read attack sends a normal HTTP message to the web server but forces it to send a reply at a slow rate based on the number of bytes the attacker specifies to be readable as defined by the TCP window size (Kemp *et al*, 2018). Therefore, to protect web servers from this hard-to-detect form of DDoS, emphasis is laid on the accuracy of the methods used in identifying such attacks. Although researchers have been able to use deterministic methods, such as calculating the distance-metrics, to detect the presence of slow HTTP DDoS traffic, the deterministic methods have exhibited several drawbacks which includes being error prone and the inability to perform real-time detection. Consequently, the focus on machine learning detection of slow HTTP DDoS attacks has gained prominence in recent times because it addresses the drawbacks of the deterministic detection method for slow HTTP DDoS attacks.

Machine learning, a sub-field of artificial intelligence, has been used in different fields including SDN in identifying different attributes of interest (Sen *et al.*, 2020). A machine is deemed as intelligent when it learns from its experience concerning the data available in its domain and uses it to enhance decisions to be taken in the future (Latah & Toker, 2019). Four groups of machine learning formed according to the learning methods of their constituent algorithms exist: supervised, unsupervised, reinforcement, and semi-supervised learning. Supervised learning implies the use of predefined knowledge to perform tasks such as classification on a new set of data that has not been analysed previously of which Support Vector Machine (SVM), Artificial Neural Network (ANN), and Decision Trees (DT) are algorithms developed on supervised learning concept (Agarwal, 2014). Unlike supervised learning, unsupervised learning finds and maps the relationships present among the data provided to make decisions based on those relationships discovered whenever new data is introduced. While supervised learning performs classification tasks on data, unsupervised learning performs clustering operations (Latah & Toker, 2019). Aside from detection of malicious traffic by statistical analysis, supervised machine learning algorithms on classification have been applied in classifying traffic into normal and malicious categories thus resulting in higher accuracy than statistical analysis but also increasing the computational cost compared to statistical analysis (Swami *et al.*, 2019a). For a machine learner to perform better, feature selection and classifier parameter optimization are executed using algorithms such as genetic, ant colony optimization, artificial bee colony, and particle swarm optimization algorithms which are meta-heuristic algorithms (Alshamrani *et al.*, 2017; Kamarudin *et al.*, 2019;

Latah & Toker, 2019). Detection of attack traffic is not worthwhile unless it is backed up with mechanisms that stop the attack and restores the service to normal levels for legitimate users. Therefore, mitigation of slow HTTP DDoS attacks is paramount in protecting the availability of the HTTP service offered by web servers (Sattar *et al.*, 2016).

Mitigation of slow HTTP DDoS attacks entails the application of techniques which protects the web server from service degradation and its resources from exhaustion by slowing or stopping the attack entirely (Jaafar *et al*., 2019; Yeasir *et al.*, 2015). When slow HTTP DDoS attacks are launched, the processing power of the victim web server's Central Processing Unit (CPU) is the resource under the threat of exhaustion (Swami *et al.*, 2019b). Given the prominence of DDoS among malicious users as an attack of choice, researchers have been able to develop several DDoS attack mitigation techniques. For instance, Jaafar *et al*. (2019) and Shafieian *et al.* (2015) worked on mitigating DDoS attacks by blocking the illegitimate traffic while Luo *et al.* (2014) examined increasing the buffer size of the bottle-neck device under DDoS attack. Furthermore, Lukaseder *et al.* (2018) explored the redirection of traffic to a Turing verification server. Similarly, Schehlmann and Baier (2013) investigated the redirection of attack traffic to a honeypot while a shark tank which is a separate cluster with full application capabilities designed to monitor suspicious users was used by Beigi-Mohammadi *et al.* (2017) to mitigate DDoS attacks. Also, Bhunia and Gurusamy (2017) researched into rate-limiting of suspicious traffic while Fonseca and Nigam (2016) approached DDoS mitigation by selectively dropping some attack traffic. Ameyed *et al*. (2015) and Sattar *et al.* (2016) focused on spreading the attack traffic across multiple replicas while Yuan *et al.* (2017) evaluated the queuing of requests using a scheduling algorithm and Yeasir *et al*. (2015) used a reverse proxy server to curb DDoS attacks. Most slow DDoS mitigation techniques rely on attack recognition measures implemented on the server such as connection timeout and the number of a concurrent connection made from an Internet Protocol (IP) address which cannot guarantee attack detection before reaching the target (Hong *et al.*, 2018). As a result, a robust attack recognition system detects the attack before it reaches the target web server and applies an effective mitigation mechanism which is evident by the response time to requests and the ratio of completed to timed-out requests of the web server (Sattar *et al*., 2016). Furthermore, having a global view of the network enables and enhances the slow HTTP DDoS detection and mitigation technique applied to the network (Lukaseder *et al*., 2018).

The emergence of Software Defined Networks (SDN) addresses the absence of unified network management and flexible device configuration observed in a traditional network by combining logically centralized network management with network programmability through the separation of the control plane from the data plane (Benzekki *et al.*, 2016). In SDN, the control plane is situated in a device called the controller which defines rules that govern the forwarding of data and the data plane is situated in devices called the switch. The switch receives and forwards data received based on the rules specified by the controller (Dabbagh *et al.*, 2015). Although the SDN now makes network monitoring and updating easier, it is also prone to malicious attacks such as DDoS (Xu *et al.*, 2017). Devices within the SDN prone to DDoS attacks include servers, switch flow tables, and the controller (Ali *et al.*, 2015). However, researchers have identified that proactive defence of networks against attacks including DDoS is achievable with SDN due to the controller's ability to collect traffic statistics from the switches (Ali *et al*., 2015). The controller's ability to gather traffic information is made possible with the OpenFlow protocol which is one of the first SDN standards responsible for intercommunication between the control and data planes (Hamad *et al.*, 2016; Swami *et al*., 2019b). Although OpenFlow is capable of sending flow statistics, the communication is not as lightweight as Netflow considering the request and associated response format it follows (Schehlmann & Baier, 2013). That is, obtaining flow statistics using OpenFlow entails the request of flow statistics from the switch by the controller and the receipt of the response from the switch by the controller. In a situation where flow statistics are needed constantly, the flow request message from the controller serves as an overhead thereby affecting the processing capability of the controller. To reduce the controller processing overhead, Netflow has been the flow statistics aggregator of choice. Network flows (Netflow), which is a Cisco Systems technology that monitors and exports network flows, refer to a unidirectional stream of network packets between a source and destination application (Schehlmann & Baier, 2013). Netflow is relatively efficient in terms of storage as it groups packets into flow summaries making it easier to query large historical traffic data sets (Kemp *et al*., 2018). A combination of SDN and Netflow gives a global view of the network, aids in fast processing of the flow records, and easy propagation of rules for slow HTTP DDoS attack mitigation.

1.2 Statement of the Research Problem

The increased attention given to DDoS by security researchers has forced attackers to consider other attack methods which are less prone to detection and mitigation but are capable of disrupting the availability of the targeted services. As a result, attackers have been able to exploit the HTTP using slow HTTP DDoS attacks which requires low bandwidth, fewer resources, and can originate from mobile phones which presently connects about 3.5 billion people globally to the Internet (Farina *et al.*, 2015). Given the complexity of the HTTP, a slow HTTP DDoS attack could be any or a combination of the following three types of attacks: slow read, slow get, or slow post DDoS attacks. Although researchers have examined the detection of slow HTTP DDoS attacks, they were able to explore only one or a combination of two types of slow HTTP DDoS attacks.

That is, some researchers explored the detection of slow read attacks only while others explored the detection of slow get and slow post attacks only without evaluating the detection of all the three types of slow HTTP DDoS attacks: slow read, slow get, and slow post. It can be inferred that the researchers who examined the detection of slow get and slow post HTTP DDoS attack pairs were able to perform the detection task seamlessly because of the similarity between both attacks (Muraleedharan & Barnabas, 2018). That is, both slow get and slow post HTTP DDoS attacks entail the sending of malicious packets to the target at a slow rate, unlike slow read which involves reading the contents of packets received from the webserver at a slow rate. Furthermore, the dissimilarity between the slow read DDoS attack and the other two slow HTTP DDoS attacks has made researchers evaluate its detection separately.

On the one hand, researchers who have examined the detection of slow HTTP DDoS attacks explored either one or two of the slow HTTP DDoS attack types without considering all three. On the other hand, however, researchers who have explored the detection of the three types of slow HTTP attacks focused on Denial of Service (DoS) without evaluating the effect of a DDoS. What this means is that the researchers who explored the three types of slow HTTP attacks evaluated the attack from a single source without checking the possibility of multiple attackers.

Furthermore, apart from the work by Ameyed *et al*. (2015) and Sattar *et al*. (2016) which explored protecting the availability of the targeted service during slow read and volumetric DDoS attacks respectively, other researchers focused on techniques that either does not guarantee some level of service availability or will eventually degrade the availability of services as the mitigation mechanisms become a bottle-neck. The methods used by researchers which do not guarantee service availability are rate-limiting, reverse proxy, traffic redirection, and time-out intervals mitigation techniques.

In this work, the detection of the three types of slow HTTP DDoS – slow read, slow get, and slow post – which originates from multiple attackers was performed using Radial Basis Function (RBF) kernel-based Support Vector Machine (SVM) which addresses the dissimilar behaviour of slow HTTP DDoS types while relying on Genetic Algorithm (GA) to select the appropriate features that signify the presence of attack traffic in a Netflow export. Besides, GA was used to tune the RBF kernel parameters to obtain optimal values that guarantee competitive classification. Unequivocally, the effectiveness of the classifier in detecting the three types of slow HTTP DDoS is hinged on the optimal selection of features that signify the presence or absence of attack traffic and robust classification that addresses the nonlinearity of the three types of slow HTTP DDoS attacks (Aziz *et al.*, 2013; Barati *et al.*, 2014; Li *et al.*, 2015; Xingzhu, 2015; Kamarudin *et al.*, 2019). Once an attack is detected, it is mitigated using the Selective Adaptive Bubble Burst (SABB) mitigation technique – a concept synthesized from the work by Ameyed *et al*.(2015) and Sattar *et al*. (2016). SABB isolates the traffic flagged as malicious onto a replica server for further observance while other legitimate traffic continues communicating with the primary webserver. Once the isolated traffic is flagged as malicious again, the selective adaptive bubble burst isolates the traffic onto another replica webserver. This process is repeated until the number of times the traffic has been flagged as malicious is greater than the number of replica webservers. Then, all traffic from the malicious Internet Protocol (IP) address is blocked at the gateway switch. The setup is tested in an SDN simulation environment such that detected attack traffic is flagged for mitigation using the selective adaptive bubble burst technique. The use of the mitigation technique highlights the effectiveness evaluation aspect suggested as a future work by Ameyed *et al*.(2015).

1.3 Aim and Objectives

This research aims to mitigate slow HTTP DDoS attacks using selective adaptive bubble burst approach while relying on radial basis function kernel support vector machine to detect the attack based on the features of interest selected by the genetic algorithm. The objectives are to:

i. Select features that signify the presence or absence of a slow DDoS attack using genetic algorithm.
ii. Classify the traffic in the flowset into benign or anomalous using RBF SVM and serialize the object obtained.
iii. Formulate a selective adaptive bubble burst model to ensure the availability of web services to legitimate users whether slow or not.
iv. Simulate the RBF SVM serialized object obtained in (ii) with selective adaptive bubble burst in an SDN environment.
v. Evaluate the performance of RBF SVM and the selective adaptive bubble burst mitigation technique.

1.4 Scope of the Study

This research focuses on mitigating slow HTTP DDoS attacks of slow HTTP get, slow post, and slow read attacks using genetic algorithm and support vector machine for attack recognition and a selective adaptive bubble burst technique to curb the attack. Volumetric or flooding attacks were not considered likewise slow attacks against other application layer protocols were not explored.

1.5 Significance of the Study

This study would be of benefit to network professionals, cloud security professionals, and researchers that are focused on improving the resilience of traditional and software-defined networks against slow HTTP DDoS attacks. Furthermore, this study will aid researchers in understanding the methods of DDoS mitigation which may be applicable to slow HTTP DDoS thus enhancing the decision making process towards selecting the appropriate mitigation technique to implement or to modify towards curbing slow HTTP DDoS.

1.6 Thesis Organisation

This thesis is comprised of five chapters. The background to the study, statement of the research problem, aims and objectives, and the significance of the study are covered in Chapter One. Review of literature related to slow DDoS mitigation techniques, slow DDoS machine-learning detection techniques, and optimization algorithms used to improve the detection accuracy of the machine learning techniques are contained in Chapter two. The methodology used for the research, the description of the dataset generated, the use of Graphical Network Simulator-3 (GNS3) to simulate an SDN network, and the metrics used to evaluate the performance of the RBF SVM approach to detecting and the selective adaptive bubble burst approach to mitigating are examined in Chapter three. Chapter four consists of the analysis of the results obtained from the execution of the RBF SVM algorithm and the implementation of the selective adaptive bubble burst mechanism. The thesis ends with chapter five which consists of the conclusion, the recommendations, and the contribution to the knowledge of this work.

1.7 Definition of Terms

The terms used in this study are defined below:

Mitigation: Mitigation can be described as the act of making the effect of an action to be less severe. Mitigation of slow DDoS as it applies to this work refers to the actions taken or to be taken which will lessen the effect of the attack on resources on the network.

Simulation: Simulation refers to the use of representative objects, features, and environments to describe another object, feature, or environment. Simulation of the SDN using GNS3 creates a representational network on a computer.

OpenFlow: OpenFlow is an SDN communication protocol that enables the control plane to communicate with other networking devices. This protocol can be used to define the behaviour of networking devices by the controller in addition to monitoring the network through flow statistics collection.

Netflow: Netflow is a flow collection protocol used to obtain aggregates of flows that move through a network via a Netflow compatible switch using the source address, a destination address, source port, destination port, and transport protocol to obtain an aggregate known as a record. It gives an abstract view of network communications.

CHAPTER TWO

2.0 LITERATURE REVIEW

2.1 Introduction

The previous research on slow HTTP DDoS mitigation is presented in this chapter while evaluating other topics of interest such as slow HTTP DDoS detection, support vector, machines, genetic algorithm, and the origin of the selective adaptive bubble burst mitigation approach being proposed. It is important to examine previous statements on the methods and tools used in this research. A review of works regarding slow DDoS detection, mitigation, and feature selection is presented at the end of the chapter.

2.2 Support Vector Machines (SVM)

SVM is an algorithm that classifies both linear and nonlinear data by searching for an optimal linear hyperplane. A hyperplane refers to the boundary of decision that effectively separates the tuples in one class from another (Agarwal, 2014). With the aid of a hyperplane, SVM is used to separate training datasets. Each data item feature can be represented in n-dimensional space and SVM can be used to discover the hyperplane that splits the dataset into two classes (Singh & Rai, 2019). Also, it provides a maximum distance (classifier margin) from itself to the closest training point. In working with nonlinear data, a nonlinear mapping, kernel function, is utilized to transform the data points into high dimensional space. Although the training time of SVM is slow, they exhibit high accuracy due to the ability to model complex nonlinear hyperplanes and are less susceptible to overfitting (Agarwal, 2014). Due to the similarity of the slow attack traffic with benign traffic, the radial basis function kernel is applied to transform the nonlinear data points into linear data points in a high dimensional space. An optimal hyperplane gives better result and is represented mathematically under the condition of linear separability and linear separable dataset of two points $(x_i, y_i), i = 1,2, \dots, n$ where

$x_i \in \mathbb{R}^n$ and $y_i \in \{+1, -1\}$. Equation 2.1 represents the correct classification of dataset (Su *et al.*, 2018; Singh & Rai, 2019; Ye *et al.*, 2019),

$$y_i((w.x_i) + b) - 1 \geq 0 \; for \; i = 1,2, \ldots\ldots, n \qquad (2.1)$$

where *y* represents two classes that have a binary value, *w* is a weight vector, *x* is an input vector, and b is a threshold value. Generalized to n-dimensional space, minimizing the structural risk of constructing the optimal classification hyperplane is equivalent to solving the constrained optimization problem with the formula expressed in equation 2.2 (Liu *et al.*, 2018; Ye *et al.*, 2018):

$$\min[\varphi(w,\varepsilon] = \min(\frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\varepsilon_i) \qquad (2.2)$$

$$s.t. \, y_i(w.x_i + b) \geq 1 - \varepsilon_i, \varepsilon_i \geq 0, i = 1,2, \ldots., N$$

where the classifier margin is maximized by minimizing $\frac{1}{2}||w||^2$ and the variables ε_i denote the extent to which the samples, x_i, violate the margin and the penalty parameter C > 0 adjusts the trade-off between minimizing the sum of the slack violation errors and maximizing the margin (Ma & Guo, 2014).

The optimization problem in equation 2.2 can be expressed through the introduction of the Lagrange multiplier $\alpha_i$and the kernel function by the formula in equation 2.3 (Liu *et al.*, 2018):

$$\max[Q(\alpha)] = \max[\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j k(x_i, x_j)] \qquad (2.3)$$

$$s.t. \sum_{i=1}^{N}\alpha_i y_i = 0; \; 0 \leq \alpha_i \leq C, i = 1,2, \ldots, N$$

where $k(x_i, x_j)$ is the kernel function. Several kernel functions exist such as linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid. However, in this work, RBF is selected for use because it works well and it is relatively easy to tune. Equation 2.4 describes the RBF kernel (Ma & Guo, 2014):

$$k(x_i, x_j) = \exp(-\gamma|x_i - x_j|^2) \tag{2.4}$$

where γ is a constant value to adjust the width of the Gaussian function.

2.3 Genetic Algorithm

Genetic algorithm is a stochastic algorithm that performs global search operations by leveraging the concept of genetics and the natural selection process (Kannan, 2018). In genetics, genes - the basic unit of heredity – are studied with their behaviours. Natural selection, as developed by Charles Darwin, is the process by which living organisms change and adapt within their population thus introducing variation among individuals in the population. Consequently, genetic algorithm mimics biological processes of the survival of the fittest to develop a solution to a problem. The basic building block of the genetic algorithm is the gene. A collection of genes forms a chromosome (individual) while a collection of chromosomes forms a population. Given a set of chromosomes (initial population), genetic algorithm evolves the population into a new population using selection operators while applying mutation and crossover operations to the selected chromosomes. A new population contains the fittest individuals based on the fitness function definition. The selection operator chooses the fittest individuals to form a new population based on the fitness function defined while crossover and mutation introduce diversity to the new population by swapping genes from chromosome pairs in the case of crossover and inverting randomly selected genes in chromosomes in the case of mutation. The crossover operation is also known as the mating operation where two chromosomes that are of good fitness based on the fitness function undergo a gene exchange operation to produce two more chromosomes that are assumed to be good. Genetic algorithm assumes that mating two good chromosomes will yield two good chromosomes (Agarwal, 2014). The parameters of a genetic algorithm are: population, fitness function, selection criteria, crossover operator, and mutation operator (Katoch *et al.*, 2021; Mirjalili *et al.*, 2020).

2.3.1 Population

A population refers to a collection of chromosomes that are potential solutions to the problem at hand (Aziz *et al.*, 2013). Since a chromosome is formed from a gene, the encoding of the individual genes is dependent on the type of problem to be solved and has to be within the constrained values to have chromosomes that represent meaningful solutions. To determine the genes, properties that affect the result of the problem at hand are to be considered. For instance, in a feature selection problem, the features are the properties that affect the result if selected or not. Therefore, every feature in the dataset is regarded as a gene. In a dataset with 31 features, a chromosome is represented by 31 genes and the chromosome is said to be of length 31. Having determined the selected genes, the representation of the genes is the next activity. Various forms of representing a gene exist such as the use of decimal, binary, string, and float. In line with the feature selection instance, determining the selection of a gene (feature) is the problem of interest hence the need to encode it using two possible values. One value signifies that a gene (feature) is selected while the other signifies that the gene is not selected. Hence, encoding the genes using binary values is the best option based on the feature selection problem.

In a parameter optimization problem, the number of genes in a chromosome is defined by the number of parameters to be optimized. That is if two parameters are to be optimized, then, there will only be two genes per chromosome. Generally, defining the number of chromosomes in a population is left to the researcher to decide (Kannan, 2018; Katoch *et al.*, 2021; Mirjalili *et al.*, 2020).

2.3.2 Fitness Function

A fitness function refers to an algorithm or a method used to calculate the fitness value of a solution (chromosome). The fitness value is an indicator that determines the ranking of a solution among all other solutions. That is, the fitness value represents the closeness of a solution to the expected result. The fitness value is used to differentiate good solutions from bad solutions. In a feature selection problem, the accuracy of the classifier based on the selected features can be used as the fitness value for the selected features (chromosome) (Agarwal, 2014; Aziz *et al*., 2013; Kannan, 2018).

2.3.3 Selection Criteria

Selection is the process of choosing chromosomes from a population for later breeding which will produce off-springs for the next generation (Anusha & Sathiyamoorthy, 2016). The convergence rate, how fast an optimal solution is obtained, of genetic algorithm is hinged on the ability to select good parents that produces better individuals. Also, maintaining the diversity of the population is important in genetic algorithm to prevent premature convergence. Premature convergence occurs when the entire population is over shadowed by one extremely fit solution. The types of parent selection methods are: roulette wheel selection, tournament selection, rank selection, and random selection (Mirjalili *et al*., 2020). Selection is an exploitation process. Exploitation is the process of accessing regions of a search space around the area of previously accessed points.

2.3.3.1 Roulette Wheel Selection

In roulette wheel parent selection method, a circular wheel is used which is divided into pies equivalent to the number of individuals in the current population. Each space occupied by each individual of the population is proportionate to the fitness value of that individual. Then, a fixed point on the circumference of the wheel is chosen. The pie section that stops in front of the fixed point when the wheel is rotated is then chosen to be the first parent. The same process is repeated to choose the second parent. Given that the size of the pie each individual occupies on the wheel is directly proportional to the individual’s fitness value, the individual with the highest fitness value has a greater probability to be selected as a parent compared to the other individuals (Mirjalili *et al*., 2020).

2.3.3.2 Tournament Selection

In a tournament selection, a specified number of individuals is selected at random from the population. Then, the best individual out of the selected individuals becomes a parent (Katoch *et al*., 2021). That is if the specified way in the tournament selection is 3, then, three individuals in the population are selected at random and the individual with the best fitness value becomes a parent. The same process is repeated for selecting the second parent. Tournament selection is well known for its low susceptibility to premature convergence due to the presence of dominant individuals.

2.3.3.3 Rank Selection

Rank selection is most appropriate when individuals in the population have close fitness values. Therefore, each individual is ranked based on their fitness. Then, the individuals with higher rankings are selected for the breeding operation (Katoch *et al*., 2021; Mirjalili *et al*., 2020).

2.3.3.4 Random Selection

In random selection, the parents are selected at random from the population. This strategy does not consider the fitness of individuals hence the need to avoid this selection strategy. This is because, the random selection of parents could yield sub-optimal or non-optimal generations which will cause the solution search process to suffer (Katoch *et al*., 2021; Mirjalili *et al*., 2020).

2.3.4 Crossover Operator

Genetic algorithm assumes that breeding two good parents will yield a good individual for the next population. This breeding happens when the genes of both parents are combined in a predefined manner to yield another individual (Mirjalili et al., 2020). Therefore, a crossover operator is applied to two parents to create a new individual. The types of crossover operators are: one-point crossover, multi-point crossover, and uniform crossover. Crossover is an exploration process. Exploration is the process of gaining access to entirely new areas of a search space (Crepinsek *et al.*, 2013).

2.3.4.1 One-point Crossover

In one point crossover, a random point for crossover is picked and the other half of the parents are exchanged to get two new off-springs. This is the most applied form of crossover (Crepinsek et al., 2013; Mirjalili *et al*., 2020).

2.3.4.2 Multi-point Crossover

In a multi-point crossover, several points are selected on each parent based on the number of points specified. Then, alternate parts of each parent are exchanged to yield new off-springs (Crepinsek et al., 2013; Mirjalili *et al*., 2020).

2.3.4.3 Uniform Crossover

In a uniform crossover, each gene is treated separately by computing the probability of a gene being part of an offspring. The probability may be biased towards a parent so that the off-springs contain more genes from one parent than the other (Crepinsek et al., 2013; Mirjalili *et al*., 2020).

2.3.5 Mutation Operator

To maintain diversity in a population, the mutation operation is performed after a crossover operation. The mutation operation changes one or more genes in an individual from its original state. A mutation probability is always defined and needs to be set to a low value else the high mutation probability converts the search process to a random search. Mutation is an exploration process (Agarwal, 2014; Crepinsek *et al*., 2013; Kannan, 2018).

2.4 Selective Adaptive Bubble Burst

Selective Adaptive Bubble Burst (SABB) is a concept created from the work by Ameyed *et al*. (2015) and Sattar *et al*. (2016). Since both works focused on ensuring the availability of services, the advantages of attack connection deletion as observed in the work by Ameyed *et al*. (2015) and the use of SDN to spread attack traffic as observed in the work by Sattar *et al*. (2016) were merged and modified to yield the SABB slow HTTP DDoS mitigation approach. In SABB, there is a primary webserver with at least one replica webserver which hosts the same content as the primary webserver. Once the first occurrence of a slow HTTP DDoS attack traffic is detected from an IP address, the connections from the detected IP address are moved from the primary webserver to the first replica server (in a situation where there are more than one replica servers). The attack detection occurrence for the IP address is updated and SABB continues to monitor the network. On subsequent flagging of the same IP address as an attacker, SABB continues to update the attack detection occurrence and moves the attack traffic to the next replica server until the attack detection occurrence exceeds the number of replica servers in the setup. Once the attack detection occurrence for an IP address exceeds the number of replica servers, the defaulting IP address is blocked from accessing any service on the network. The movement of new connections from an IP address flagged as a slow HTTP DDoS attack to replica servers is being performed so as to verify the detection result multiple times given that traffic from slow legitimate traffic bears similarity with slow HTTP DDoS attacks. The works from which the SABB was synthesized are presented and comparisons drawn.

Ameyed *et al*. (2015) proposed an availability model to curb slow read DDoS having considered the limitations of existing slow read DDoS mitigation models such as using connection timeouts, total connections on a webserver, and the total number of attack connections. The proposed availability model incorporated multiple web servers through a failure isolation zone. The failure isolation zone involves the distribution of web services on webservers among two zones. In the first zone, once the total number of connections the webserver can process has reached its peak, new connections are sent to the second webserver in another zone. Then, the connections on the first webserver are analyzed and connections deemed to be slow which have originated from the same IP address are deleted. The proposed model was not implemented. Similarly, SABB consists of replica servers and it moves the flagged attack connections from the primary server to the replica server. However, on the contrary, SABB is focused on mitigating slow get, slow post and slow read HTTP DDoS attacks and it blocks the attack traffic completely as opposed to deleting each connection established. The traffic SABB blocks would have been flagged as a slow HTTP DDoS attack traffic multiple times as opposed to the technique by Ameyed *et al.* (2015) where slow connections from the same IP address are deleted while assuming that the multiple slow connections originated from an attacker.

Another availability model developed by Sattar *et al*. (2016) explored the use of SDN to mitigate volumetric DDoS by ensuring the availability of services during an attack. The approach used in the work was called Adaptive Bubble Burst (ABB). In the work, the detection of volumetric DDoS was performed by setting a threshold on incoming traffic based on the serving capacity of the webserver or file server. Once the threshold is reached, the ABB bursts the DDoS attack bubble by spreading the traffic across several nodes serving the same resource in the network. It was noted in the work that the attack might still cause damage to some resources even after the ABB is activated. The advantages of the ABB as highlighted by the authors are the reduction of the impact of volumetric DDoS on a single node in the network and the boosting of service availability during volumetric DDoS attacks. On the one hand, SABB bursts the slow HTTP DDoS attack which is similar to the operations of ABB. On the other hand, however, SABB does not spread all incoming traffic to replica servers. SABB selects the connections made by the flagged IP address with subsequent connections from the flagged IP address and maps their destination to be one of the replica servers without disrupting other legitimate traffic traversing the network.

2.5 Software Defined Network Simulation and Emulation Tools

Simulation refers to the creation of a synthetic version of a real-world process or system thereby mimicking the operation and behaviours of the system, especially at the software level. In contrast, emulation refers to the development of a superficial environment that mimics both the hardware and the software of the target environment. Due to the issues around the access to a physical SDN topology, several simulators and emulators have been developed to aid in performing research and experimental operations on SDN. A simulator is a tool developed to mimic an environment that contains the necessary software variables and configurations that will exist in an application production environment. In contrast, an emulator attempts to mimic both hardware and software features of the production environment. In this work, the production environment is SDN. SDN simulators and emulators include Mininet, Network Simulator 3(NS-3), Objective Modular Network Testbed in C++ (OMNeT++), Estinet, and Graphical Network Simulator-3 (GNS3).

2.5.1 Mininet

Mininet is a virtual network emulator which creates a network of links, switches, controllers, and virtual hosts. The operating system of the virtual hosts in a Mininet network is based on the standard Linux network software. The switches in Mininet support the OpenFlow protocol. Interaction with a network designed using Mininet is performed using a Command Line Interface (CLI). Mininet provides a simple network testbed for creating OpenFlow applications which are easily reconfigured and restarted however, the settings and configurations on the virtual hosts are lost on exiting the emulator (Li *et al*., 2020).

2.5.2 Network Simulator 3 (NS-3)

NS-3 is a discrete-event simulator for networks to model the behaviour of packet networks. It is open-source and has a set of libraries that can be merged with other libraries to provide visualization. It also supports the OpenFlow protocol (Jevtic *et al*., 2018).

2.5.3 Objective Modular Network Testbed in C++ (OMNeT++)

Objective Modular Network Testbed in C++ (OMNeT++) is an extensible simulation library in C++ for developing network simulations that support the OpenFlow protocol. The OMNeT++ simulator supports SDN through an extension. It also consists of a Graphical User Interface (GUI) which makes the simulation visible to the user.

2.5.4 Estinet

Estinet is a licensed OpenFlow network simulator and emulator that supports the creation, testing, and evaluation of SDN. That is, Estinet combines the advantages of a simulator which are low cost, flexibility, repeatability, and controllability with the real-time execution advantage of an emulator. The free trial license of Estinet provides a network with a maximum of 15 nodes with a five minute simulation time (Wang *et al.*, 2013).

2.5.5 Graphical Network Simulator-3 (GNS3)

Graphical Network Simulator-3 (GNS3) is an emulator software for networks that execute real operating system images in a network. Virtualisation of the host computer is needed for GNS3 to run. It also has a GUI which makes it easy to configure and design network topologies. Furthermore, GNS3 supports the persistence of data and commands on the hosts in the network. This makes it easy to transfer external files into hosts for execution and perform complex configurations on the hosts without losing the configuration on shutdown. In this work, GNS3 was the simulator of choice because of the data and command persistence it offers given the need to configure each host with a real operating system image and load dependencies to support the simulation task. Besides, since GNS3 is an emulator, experiments performed on it can be transferred easily with little to no configuration onto a real-life system. Furthermore, the intuitive GUI makes it easy to utilize (Polanco & Guerrero, 2020).

2.5 Review of Slow DDoS Mitigation Techniques

Slow read DDoS mitigation mechanism that ensures high availability of resources to legitimate users was explored by Ameyed *et al*. (2015). In the work, an approach that ensures high availability and redundancy in the cloud was proposed by implementing a failure isolation zone in distributing web services instances across two zones. Once the webserver in the first zone reaches its maximum number of connections, new connections are redirected to the other zone. Then slow connections coming from the same IP address in the first zone are deleted from the server. The main focus of this approach is ensuring the availability of web services. Evaluating the effectiveness of their approach through its implementation was outlined as an area of further research.

Park (2015) evaluated the strength of slow read DDoS against ModSecurity mitigation mechanism which is based on limiting the number of connections made from an IP address. Unlike the work by Ameyed *et al.* (2015), monitoring of the number of connections from IP addresses was used to detect a slow read attack that stops the attacker before the maximum number of available connections is reached. In a situation where the number of connections made from an IP address exceeds the limit set, ModSecurity was able to mitigate the attack effectively. However, in a situation where the number of attackers was increased but the number of connections made was below the limit set, ModSecurity was ineffective. Adaptively changing the number of connections made by an attacker per second with the window size and deciding the value of window size based on concurrent measurement of packet arrival time were marked for further studies on attack propagation and attack mitigation.

Reverse proxy was used in the work by Yeasir *et al.* (2015) as a defence mechanism against slow HTTP get DDoS. The concept employed in their work differs from that of Ameyed *et al.* (2015) and Park (2015) because slow get attacks involve sending data to the server as opposed to slow read which involves receiving data from the server. The reverse proxy is a proxy server that handles requests for resources by clients by retrieving the resource from the server then sending the resource to the client thus hiding the existence of the main server. It also balances request load among available servers. Since the reverse proxy server caches requests, unless properly configured with timeouts and thresholds, its resources could also be exhausted thus causing a denial of service scenario. Finding the appropriate timeout or threshold so as not to disconnect legitimate slow clients is difficult. Extending their approach to other web servers which are not HTTP-based was suggested as an aspect for future study.

Adaptive Bubble Burst (ABB) was used in SDN to mitigate DDoS attacks in the research by Sattar *et al*. (2016). The work differed from that of Ameyed *et al*. (2015), Park (2015), and Yeasir *et al*. (2015) as it focused on DDoS launched at a high or fast rate, not slow DDoS. ABB takes advantage of the logically centralized nature of SDN to mitigate DDoS by diffusing incoming traffic. ABB aims to ensure service availability lends web server anonymity and DDoS mitigation in the network. Threshold based on the serving limit of the webserver for incoming traffic rate measured at the gateway switch was used to trigger the ABB mechanism which spreads the traffic across multiple replicas of the target node in the network. Virtual IP addresses were used in ABB to enforce anonymity. High response time with 4% legitimate request completion was recorded without ABB but when ABB was activated, request completion rose to 81% because the traffic was spread to two servers. When the traffic spread was increased to three servers, no timed-out request was recorded. Although ABB increases web service availability, it comes with a packet processing overhead. Without ABB, the average packet processing time was 542µs while when ABB was activated, processing time increased to 776µs per packet. A faster method of collecting statistics to reduce packet processing overhead was suggested by the researchers as the controller in the work collected statistics every one second.

Hong *et al*. (2018) examined the use of SDN in the defence against slow DDoS based on connection threshold, similar to the method used by Park (2015), and timeout mechanisms, similar to the method employed by Yeasir *et al*. (2015). Whenever the number of connections made has exceeded the number of connections the web server can handle, it alerts the controller that it is under attack so that the controller begins a slow DDoS check on subsequent connections. Then continuation of the incomplete HTTP requests is received by the controller. The controller determines that a client is an attacker when the number of connections made does not complete the HTTP request within a certain timeframe. Table 2.1 presents the various mitigation techniques used in mitigating DDoS and slow DDoS.

Failure isolation zone, proposed by Ameyed *et al*. (2015), ensures the availability of web services by splitting the traffic on the primary web server once it has reached its maximum amount of connections to its replica in another zone. Then slow traffic on the primary server is analyzed and deleted. The framework ensures availability for legitimate clients of the web server however because it waits for the maximum connection limit of the web server to be reached, web clients experience very slow data transfers in the buildup to the threshold. Furthermore, the removal of traffic from the primary web server is executed at a pace that may be considered slow compared to the way an attacker saturates the replica with connections that send low rate requests. Rate limiting, a concept explored by Park (2015) and Bhunia and Gurusamy (2017), limits the number of connections made by an IP address thus stopping attack traffics but risks disconnecting benign slow clients. Yeasir *et al*. (2015) used reverse proxy for mitigation of slow attacks by caching slow requests until they are completed. The reverse proxy server acts as a buffer for slow request thus shielding the webserver but risks becoming a target. Hong *et al*. (2018) used timeout intervals to halt slow attacks by defining a time for request completion else the client, benign or malicious, get disconnected from the server however, slow benign traffic is affected by the timeout. Beigi-Mohammadi *et al*. (2017) and Lukaseder *et al*. (2018) explored redirecting attack traffic to a scrubbing server, a shark tank - a copy of the application's topology - which ensures that more insight is obtained from the attack although, it may also slow down the traffic of slow benign clients.

Table 2.1: Weaknesses and Strengths of Various DDoS and Slow DDoS Mitigation Approaches

MITIGATION APPROACHES	**Failure-Isolation Zone**	**Rate Limiting**	**Reverse Proxy**	**Timeout Intervals**	**Redirection**	**Adaptive Bubble Burst**
Characteristics	Splits traffic to a replica web server	Reducing the number of connections made by an IP address	Caches slow requests until they are completed	Disables a connection once timeout for request completion has exceeded the predefined value	Redirection of traffic to a shark tank	Splits the traffic to replica servers
Strength	Ensures availability	Removes attack traffic	Enhances protection of web server	Removes slow traffic	Learns more from attack scenario	Ensures availability
Weakness	Removes attack traffic slowly	May remove slow client's connections	Reverse proxy resources may become exhausted	May remove slow client's connections	Slows down connections made by slow client's	Does not remove attack traffic

Sattar *et al*. (2016) utilized adaptive bubble burst in mitigating attacks by spreading all new connection traffic to replica servers once a threshold is reached. The technique ensures availability but does not remove attackers hence making it a matter of time before all replica servers become saturated with connections.

2.6 Review of Slow DDoS Detection Techniques

The use of the web server's performance was explored by Shtern *et al.* (2014) to detect low and slow DDoS attacks in which the CPU utilization, disk utilization, disk time, waiting time, throughput, workload, and CPU time of the server was measured in a non-attack scenario. The application characteristics measured initially is then compared with subsequent characteristics thus creating a discrepancy measure. Once the discrepancy measure is greater than the discrepancy threshold, an attack is signalled. Offline establishment of the performance model could yield high false positives while an online establishment of the performance model could cause the bypass of the detection module leading to high false negatives. A shark tank mitigation mechanism was employed as it learns more from the attack. A shark tank is a copy of the application's topology hosted in an isolated environment which enables the observance of suspicious traffic. It is a separate cluster with full application capabilities designed to monitor suspicious users. For future research, a comparison of the offline and online establishment of the performance model was identified.

Detection of slow read attacks in the cloud using machine learning technique, as opposed to a performance measurement technique used by Shtern *et al*. (2014), was the basis of research by Shafieian *et al.* (2015). The detection of an attack was based on the random forest classifier trained on TCP logs of attack and benign traffic. Accuracy of 99.37% and false negative rate of 1.90% was recorded when pre-pruning of trees was not applied while when pre-pruning was applied, accuracy of 83.34% and false negative rate of

50.10% was recorded. A false positive rate of 0 % was recorded in both cases. It was observed that an increase in the number of trees increases the true positive rate and decreases the false positive rate. The stability of results obtained in the work makes the method employed to be superior to the performance measurement model evaluated by Shtern et al. (2014) although, detection may not be in real-time as it depends on TCP logs for attack detection. Slow get and post attacks were not explored in their work.

Mobile devices are not spared in slow DDoS attacks as described by Cusack and Tian (2016). Euclidean distance-based similarity metric was employed in the detection of an attack by evaluating the similarity between a previous log file and the current log file to determine whether an attack has occurred. The execution of the method is similar to the performance-based method employed by Shtern *et al.* (2014) as the connection between initially calculated values and the test values is determined. The accuracy and reliability of their approach were not evaluated. However, it can be inferred that real-time detection of a slow attack is not feasible as log files need to be collected and compared. This makes the technique employed useful for auditing logs rather than monitoring. Use of the algorithm proposed on a larger data set was included in their future work.

Kumar (2016) applied one-class SVM to protect virtual machines by detecting DoS attacks in the cloud. The researcher did not use the Knowledge Discovery in Databases (KDD) Cup 1999 dataset citing ageing factor and imbalance but opted to generate a dataset on the Eucalyptus cloud platform. Attacks of Internet Control Message Protocol (ICMP) flood, ping of death, User Datagram Protocol (UDP) flood, TCP SYN flood, TCP Land, Domain Name System (DNS) flood and slowloris also known as slow HTTP get were simulated. SVM classifier performed poorly in detecting slowloris with an accuracy of 68% and sensitivity of 43% compared to ICMP flood, ping of death, UDP flood, TCP SYN flood, TCP Land, and DNS flood which achieved accuracy values of 100%, 94%, 97%, 96%, 98%, and 99% respectively with sensitivity values of 100%, 100%, 97%, 100%, 100%, and 100% respectively. The researcher concluded that the weak performance of the classifier was due to the nature of slow attacks that transmit HTTP malicious flows at a slow rate without them being detected effectively. Also, it can be noted that the slowloris attack is different from the other types of attacks in the work thus the classifier's accuracy must have been skewed due to the presence of flooding attacks in the training dataset.

Tripathi *et al.* (2016) employed the measurement of the Hellinger distance between two probability distributions of the normal and attack traffic generated during testing and training phases. This method is similar to the method used by Cusack and Tian (2016) and Shtern *et al.* (2014) which were based on the similarity between two metrics. The SlowHTTPTest tool was used to generate attack traffic of which the Hellinger distance in a DoS scenario was high compared to that of normal traffic. The Hellinger distances of 0.0006/0.0118, 0.3980, and 0.3971 were recorded for normal, slow get, and slow message body simulated HTTP traffic respectively while Hellinger distances of 0.0191/0.1273, 0.2812, 0.3562 were recorded for normal, slow get, and slow message body traffic respectively using real traffic normal interval. The detection system proposed can be evaded when an attacker generates HTTP requests with probability distributions similar to that of the normal traffic used in the training phase.

The perspective of having a slow attack on an OpenFlow switch was examined by Dantas *et al.* (2017) unlike the focus on web servers in the work of Kumar (2016), Shafieian *et al*. (2015), Shtern *et al*. (2014) and Tripathi *et al*. (2016). Slow exhaustion of the Ternary Content Addressable Memory (TCAM) of OpenFlow switches in SDN which involves sending new flows to the switch and maintaining the flow entry in the table by sending data at intervals smaller than the timeout interval was examined in the work. Rule aggregation, dynamic timeouts, and improving TCAM usage by storing fewer data were the mitigation schemes highlighted. 95.6% of clients connected were able to obtain a response after the attacker launched an attack for every 100 packets per 10 seconds with an attacking intensity of 5.8 unique packets per second. A median time to service of 2,454 milliseconds was recorded. The applicability of alternative defence mechanisms was highlighted as future work.

Tripathi and Hubballi (2018) adopted the use of chi-square statistics to detect slow rate HTTP/2 DoS attacks which was used as a distance measurement technique similar to their earlier work (Tripathi *et al*., 2016). In the training phase, they collected legitimate traffic over some time, ΔT, and compared the traffic generated in the training phase with traffic obtained in the testing phase using the chi-square distance measurement technique. The challenge encountered was choosing the appropriate threshold significance level (α) and time interval (ΔT). Recall rate of 100% was recorded for $\Delta T = 20$ and 25 minutes independent of α value but an increase in ΔT and α results in a change in recall and false positive rate. A large time interval provides better recall rate and a higher false positive rate but a small time interval affects recall adversely and improves the false positive rate. The technique used failed to detect slow get and slow post attacks because the attacks behaved like legitimate clients. As observed in the research by Cusack and Tian (2016), Tripathi and Hubballi (2018), and Tripathi *et al*. (2016), a distance-based measurement for detection of slow DoS or DDoS attacks has a lot of drawbacks. DDoS was not taken into account.

Kemp *et al*. (2018) used a generated dataset based on Netflow data and features which is capable of handling a growing amount of traffic. Netflow was selected considering the storage and resource-intensive nature of full packet captures. Another alternative was the use of web server logs but the logs are created when TCP connections get closed which means that logs are not available until the damage has been done and the attack called off. A varying number of connections and connection time intervals were used to generate the dataset using the SlowHTTPTest tool which was executed from a single machine with each attack variation running for about an hour. Eight different classifiers were used to model slow read DoS attack detection of which random forest, C4.5 N, 5-Nearest Neighbour (5-NN), C4.5D, Multilayer Perceptron (MLP), JRip, SVM, and Naïve Bayes achieved an Area Under the Curve (AUC) detection of 96.76%, 96.72%, 96.69%, 96.62%, 95.06%, 94.71%, 89.22%, and 88.94% respectively. The high AUC recorded in the work has similarity with the high accuracy recorded in the research by Shafieian *et al*. (2015). Evaluation of slow post attacks was suggested as an area for further exploration.

Analysis of support vector machine techniques in detecting intrusion formed the crux of the research by Singh and Rai (2019). NSL-KDD dataset was used in evaluating the performance of linear SVM, quadratic SVM, fine gaussian SVM, and medium gaussian SVM. Attacks evaluated were DoS, remote 2 user, user 2 root, and probing. Linear SVM, quadratic SVM, fine gaussian SVM, and medium gaussian SVM produced an accuracy of 96.1%, 98.6%, 98.7%, and 98.5% respectively with an overall error rate of 3.9%, 1.4%, 1.3%, and 1.5% respectively. The use of a real-time dataset and the use of SVM optimization techniques to evaluate the SVM techniques outlined were highlighted for further research.

Calvert & Khoshgoftaar (2019) worked on the use of machine learning techniques to detect slow HTTP DoS get and post attacks and how class distribution affects detection performance. The attacks ran on a single host machine for one hour. In the slow post attack, a default content-length value of 1000 was used in executing the attack. Netflow features were extracted from the full packet captures because it gives a high-level summary of communications over the network and lessens resource impact if full packet analysis were to be utilized. Eight classification algorithms which are K-Nearest Neighbour (KNN), Naïve Bayes, MLP, SVM, C4.5 decision trees, Random Forest (RF), JRip and Logistics Regression (LR) were used to build predictive models using five class distribution ratio of normal to attack instances which are 99:1, 90:10, 75:25, 65:35, and 50:50. It was noted that most of the learners achieved high AUC which was attributed in part to the feature set used by Netflow. RF achieved the highest AUC value of 0.99905 with a class ratio of 50:50. RF also achieved the second highest AUC of 0.99904 with a class ratio of 65:35. The work by Calvert and Khoshgoftaar (2019) has shown that RF detects slow attacks seamlessly with results similar to those obtained in the work by Kemp *et al*. (2018) and Shafieian *et al*. (2015). Evaluation of the methodology proposed using other datasets and performance metrics were marked down as aspects for future work.

Measurement of the average network delay was employed by Dhanapal and Nithyanandam (2019) in detecting slow HTTP attacks. Since attackers feign slow network as a reason for slow requests, the mechanism implemented in their paper measures the network delay of a slow client by sending five ping requests to the client. Once the time between HTTP requests exceeds the average network delay of the client, the client is placed in the blocked list. Clients that persistently advertise a TCP window size of zero and those that send out few bytes of HTTP post requests after 80% of the connection keep-alive time interval is exceeded are treated as attackers and further requests are blocked. Advertisement of TCP window sizes greater than 0 but small enough to execute a slow read attack was not examined. The method employed showed the real-time monitoring and detection capability it has compared to the methods used by Cusack and Tian (2016) and Shafieian *et al*. (2015).

Rahman *et al.* (2019) evaluated machine learning techniques in the detection and blockage of DDoS attacks in an SDN network. The authors applied J48, RF, SVM, and KNN to detect flooding attacks. In obtaining the dataset, the hping3 tool was used in a Python script and tshark was used to capture both malicious and benign traffic. Capture for malicious traffic ran for 30 minutes while that of the normal traffic was for three hours. Captured files were then converted to Comma-Separated Values (CSV) format in which data preprocessing was performed using Weka. In their analysis, J48 performed better than the other machine learning techniques considering the training time of 17.43 seconds and testing time of 3.03 seconds which is accomplished. RF, SVM, and KNN had training time of 171.11 seconds, 168.59 seconds, and 0.13 seconds respectively with a testing time of 5.19 seconds, 1.97 seconds, and 1,5957.7 seconds respectively. The trained J48 model was exported for online classification of traffic in their SDN network simulated using Mininet. Detection and mitigation of attack using their approach take approximately 10-15 seconds to complete. Reduction of attack detection time by minimizing the number of steps in classifying traffic and use of a honeypot server to analyze the attack in an in-depth manner were marked for further research.

Detection of attacks in cloud computing using machine learning techniques formed the core of the work by Wani *et al.* (2019). The DDoS attack was launched against OwnCloud, an open-source private server, using Tor Hammer as the attack tool and tshark tool for recording both suspicious and normal traffic. Snort, an intrusion detection system, was used to attach class attributes of “normal” and “suspicious” to the dataset generated by tshark. Machine learning algorithms of RF, Naïve Bayes, and SVM were utilized for data classification which was performed in Weka. SVM had the highest accuracy of 0.997 compared to random forest and Naïve Bayes that had an accuracy of 0.976 and 0.980 respectively. The results recorded buttresses the findings on the accuracy of SVM in the research by Singh and Rai (2019) however, the result random forest obtained does not dispel the conclusions reached about it as a good choice for DDoS detection as noted by Calvert and Khoshgoftaar (2019), Kemp *et al*. (2018), and Shafieian *et al*. (2015). Further research on the inclusion of more attack types and feature selection techniques was highlighted. Selected methods of detecting slow DDoS is presented in Table 2.2.

In Table 2.2, the nonlinear data refers to data without a distinct difference between the class' different data points to be classified. The stability of a machine learning algorithm refers to how well it fares when new data aside from the ones used in training and testing is introduced. Table 2.2 lists the properties of the random forest algorithm as identified by Ali *et al.* (2012) showing that it is less sensitive to outliers and it produces high accuracy due to its ability to handle missing values. The random forest can be used for both classification and regression (Calvert & Khoshgoftaar, 2019). Also, it produces high accuracy as substantiated by the literature reviewed by Latah and Toker (2019). However, Kemp *et al*. (2018) highlighted the long training times random forest requires due to the number of the decision trees generated.

Similar to the random forest, the features of SVM enumerated in Table 2.2, as supported by Latah and Toker (2019), shows that it can deal with non-linear data seamlessly although using kernel functions. This makes SVM classification accuracy to be high but in some cases, not as high as random forest. Also, it yields higher accuracy when a limited dataset is involved (Bhunia & Gurusamy, 2017). However, the extensive time required to create the models together with the problem of selecting the appropriate kernel function were identified as some of SVM's weaknesses (Bhunia & Gurusamy, 2017).

MLP as a choice of slow DDoS machine learning detection technique can handle non-linear data properly through the use of its activation function which enables it to yield high accuracy although the accuracy may not be as high as that of RF and SVM. It is also suitable for classification and regression problems. However, the time to train the algorithm which is dependent on the strength of the hardware it executes upon is a major drawback (Latah & Toker, 2019; Ramchoun *et al.*, 2016).

Unlike other machine learning techniques, Calvert and Khoshgoftaar (2019) noted that the assumption that features are independent serves as a disadvantage in using Naïve Bayes. However, it has been known to outperform some other machine learning techniques. It is suitable for applications that are critical in terms of time and storage (Kaviani & Dhotre, 2018).

Random forest is a collection of decision trees as opposed to C4.5N which is a decision tree algorithm that represents the machine learning model as a single tree. C4.5N achieves good accuracy as shown in Table 2.2 by making the completed decision tress generalized through pruning. It generates a simple and accurate decision tree especially when a small dataset is involved. However, when a large amount of data is involved, it can be expensive to build (Ali et al., 2012; Lakshmi, 2015).

5-NN is a variation of KNN supervised machine learning algorithm where the value of K is 5. In general, KNN does not require training of the dataset as it computes the distance between instances in the dataset using predefined distance metrics. However, it is time inefficient when new data is introduced because it computes the distances between all the instances present in the dataset again which makes it less stable. Also, it cannot handle data with high dimension properly as the distance may be dominated by unrelated attributes. The optimal choice for K is 5 as substantiated in the finding by Calvert and Khoshgoftaar (2019) and Najafabadi *et al.* (2016) as it gives the best result by avoiding overfitting and bias.

Based on the properties defined in Table 2.2, logistics regression performs on an average and has good training time with accuracy however, it cannot capture complex relationships and cannot solve non-linear problems since its decision surface is linear (Donges, 2018).

2.7 Review of Optimization Techniques Applied to Machine Learning Approaches Towards Intrusion Detection

In optimizing the execution of detection algorithms, several algorithms such as genetic algorithm (GA), artificial bee colony (ABC), particle swarm optimization (PSO), and grasshopper optimization algorithm (GOA) have been used.

ABC is a population-based meta-heuristic optimization technique inspired by the foraging behaviour of honeybee swarms. It was used to optimize SVM parameters and select features to be used in an intrusion detection system by Wang *et al.* (2010). The SVM parameters selected by the ABC algorithm were: C, 2238.2041; σ, 1.1037; and ε, 0.01275. It was observed that the use of feature selection methods improved the overall accuracy by 1.31% to 2.65%. ABC-SVM achieved an accuracy of 100% compared to PSO with SVM and GA with SVM with an accuracy of 98.69% and 97.35% respectively. Improvement of the feature selection algorithm on search strategy and evaluation criteria was identified for further studies.

Detection of application layer attacks using MLP with GA to train the neural network and select weights instead of using gradient descent was the basis of the work by Singh and De (2017). GA is a meta-heuristic optimization algorithm inspired by the method of natural selection and evolution with its roots in Charles Darwin's theory. MLP with GA had a high accuracy of 98.04% and false positive of 2.21% compared to that of MLP, RBF, Naïve Bayes, J48, and C4.5 which had 96.92%, 89.64%, 82.91%, 97.75%, and

94.68% accuracy respectively and 2.95%, 7.38%, 22.14%, 2.57%, and 5.45% false positive respectively. Singh *et al.* (2016) also used MLP to detect application layer attacks using GA to train the network in which MLP with GA had an accuracy of 98.31% compared to MLP which had an accuracy of 95.23%.

Su *et al*. (2018) used PSO to optimize SVM parameters in detecting illegal network access in power monitoring systems. Since there is no predefined standard of choosing the disciplinary factor C and other kernel function parameters on which the accuracy of SVM is dependent, PSO was used to arrive at the optimal value. PSO is based on the social behaviour of groups. The PSO with SVM algorithm achieved an average accuracy of 89.46% with an average error rate of 4.6% compared to SVM algorithm which achieved an average accuracy of 80.67% with an average error rate of 9.2%.

GOA, a meta-heuristic algorithm, was used in the work by Ye *et al*. (2019) to identify the optimal parameters to be used to improve the accuracy of SVM in detecting network intrusions. GOA is inspired by the swarm intelligence behaviour exhibited by grasshoppers. The fitness function of the optimization algorithm was based on the classification accuracy of the SVM training sets. GOA-SVM had an average accuracy of 97.84% compared to the average accuracy of SVM of 91.31%. Their future work entails the use of GOA with SVM to verify other intrusion detection data sets. The review of related work is summarized in Table 2.3.

Table 2.2: Weaknesses and Strengths of Various Slow DDoS Detection Machine Learning Approaches

Machine Learning Methods	**Random Forest**	**Support Vector Machine**	**Multi-layer Perceptron**	**Naïve Bayes**	**C4.5N**	**5-Neural Network**	**Logistic Regression**
Training Time	Requires longer training due to the number of trees generated	Require long training time on large datasets	Long training time	Less training time due to less training data	Time efficient with less data	Does not require training but is time inefficient for every prediction	Efficient
Classification Accuracy	High	High	High but trails behind support vector machine and random forest	Good as it has chances of loss of accuracy	Good	High than other values of K	Good
Nonlinear Data	Handles it properly	Handles it efficiently	Handles it efficiently based on the activation function	Handles it inefficiently	Can discover nonlinear relationships	Can learn non-linear boundary	Handles it inefficiently
Stability	Stable	Stable	Stable	Stable	Not stable	Not stable	Stable
Problem Type	Classification and regression	Classification and regression	Classification and regression	Classification and prediction	Classification	Classification and regression	Classification and regression
Weakness	Complex	Selecting appropriate kernel function	Hardware dependence and unknown duration of network	Assumes that feature are independent	Expensive in complexity when data is large	Does not work well in high dimensions	Cannot capture complex relationships

Table 2.3: Review of related work

S/N	Author/Year	Techniques	Strengths	Weaknesses
		Mitigation Techniques		
1	Ameyed *et al*. (2015)	Failure isolation zone	Ensures availability	Slowly removes attacks from the primary webserver
2	Park (2015)	Rate limiting	Blocks attack traffic	May cut off slow benign traffic
3	Yeasir *et al*. (2015)	Reverse proxy	Enhances web server security by serving as a proxy	Reverse proxy server's resource depletion and becoming a slow DDoS target
4	Sattar *et al*. (2016)	Adaptive bubble burst	Ensures availability	Does not block or stop the attack. Makes web services available for a while before all replica servers become overwhelmed
5	Hong *et al*. (2018)	Connection threshold and timeout	Removes slow attacks	Removes slow benign clients whose connection exceeds the timeout set

Table 2.3: Review of related work (continued)

S/N	Author/Year	Techniques	Strengths	Weaknesses
		Detection Techniques		
6	Shtern *et al*. (2014)	Web server performance	Identifies changes in web server resource usage	The dilemma of when to establish web server performance: offline or online
7	Shafieian *et al*. (2015)	RF with and without pre-pruning	Without prepruning, RF achieves an accuracy of 99.37%, a false negative rate of 1.90%, false positive rate of 0%	Expensive to deploy RF with large numbers of trees and cannot differentiate between DDoS types
8	Cusack and Tian (2016)	Euclidean distance-based similarity metric	Detects protocol used to engage in attacks	Cannot detect DDoS attacks in real-time
9	Kumar (2016)	One class SVM	SVM was able to detect flooding attacks	Low performance in detecting slowloris attacks
10	Tripathi *et al*. (2016)	Hellinger distance	Utilizes simple probability distributions and Hellinger distances to detect attacks	Evasion of detection by generating attack similar to the HTTP profile created during training is possible

Table 2.3: Review of related work (continued)

S/N	Author/Year	Techniques	Strengths	Weaknesses
11	Dantas *et al*. (2017)	Rule aggregation, dynamic timeouts, and storing less data	Switch availability of 95.6% when attack traffic is at 100 packets per ten second with a median time to service of 2454ms	Unused legitimate rules are also removed
12	Tripathi and Hubballi (2018)	Chi-square statistics	100% recall rate for ΔT = 20 and 25 minutes and 0% false positive rate for ΔT = 5 minutes	Large ΔT increases the false positive rate and low ΔT reduces recall rate
13	Kemp *et al*. (2018)	RF, C4.5N, 5NN, C4.5D, MLP, JRIP, SVM, Naïve Bayes	RF achieved the highest AUC, 96.76%, for detection of DoS compared to the other seven classifiers	Detection of DoS is easy compared to DDoS
14	Singh and Rai (2019)	Linear, Quadratic, Fine Gaussian, and Medium Gaussian SVM	Fine Gaussian SVM achieved high accuracy of 98.7% and a low error rate of 1.3% compared to the other four techniques.	No optimization of SVM parameters
15	Calvert and Khoshgoftaar (2019)	KNN, Naïve Bayes, MLP, SVM, C4.5D, RF, JRIP, LR	RF achieved the highest and second highest AUC values of 0.99905 and 0.99904 respectively	Detection of DoS is relatively easy compared to DDoS

Table 2.3: Review of related work (continued)

S/N	Author/Year	Techniques	Strengths	Weaknesses
16	Dhanapal and Nithyanandam (2019)	Network delay measurement using five pings to slow client to determine the uniqueness of the slow behaviour	Differentiates actual slow clients from slow HTTP DDoS attackers	Slow read attacks with low TCP window greater than 0 escapes detection
17	Rahman *et al*. (2019)	J48, RF, SVM, KNN	J48 had the lowest training and testing time aggregate	One performance metric of time was used
18	Wani *et al*. (2019)	SVM, RF, Naïve Bayes	High SVM accuracy of 0.997	Small datasets of few megabytes can be handled with Weka
		Optimization Techniques		
19	Wang *et al*. (2010)	ABC with SVM, PSO with SVM, GA with SVM	ABC with SVM achieved an accuracy of 100%	Premature convergence in later search period of ABC
20	Singh and De (2017)	MLP, RBF, NB, MLP with GA, J48, C4.5	MLP with GA achieved an accuracy of 98.04%	Computationally expensive nature of MLP with GA
21	Su *et al*. (2018)	SVM, LS with SVM, PSO with SVM	An average accuracy of 89.46% with an average error rate of 4.6% was achieved with PSO with SVM algorithm	Easy to fall into local optimum in high dimensional space of PSO Low convergence rate in the iterative process of PSO

Table 2.3: Review of related work (continued)

S/N	Author/Year	Techniques	Strengths	Weaknesses
22	Ye *et al*. (2019)	SVM, PSO with SVM, GA with SVM, GOA with SVM	GOA with SVM had an average accuracy of 97.84%	Easy to fall into local optimum and slow convergence speed of GOA

2.8 Chapter Summary

This chapter presents an overview of DDoS and slow DDoS mitigation techniques, machine learning approaches for detecting slow DDoS, and optimization methods employed to enhance the detection accuracy of the machine learning techniques. Various machine learning approaches have been employed to detect slow DoS/DDoS attacks in the literature reviewed and mitigation techniques were proffered and applied. Therefore, this study is centred on the use of genetic algorithm for optimizing the kernel parameters of support vector machine for slow HTTP DDoS detection and the use of selective adaptive bubble burst, a variation of adaptive bubble burst, to mitigate the attack.

CHAPTER THREE

3.0 RESEARCH METHODOLOGY

3.1 Introduction

This chapter presents the methods used in performing this research. It provides a detailed explanation of the steps used in achieving the outlined objectives. The process of generating the Netflow dataset, preprocessing the generated dataset, training the SVM classifier for feature selection, training the SVM classifier for RBF parameter tuning, testing the performance of the classifier on the selected features and RBF parameters, and simulating the SABB mitigation mechanism was described. The performance evaluation metrics used to evaluate the performance of the RBF SVM slow HTTP DDoS attack detection and the SABB slow HTTP DDoS attack mitigation were also discussed. The proposed solution is presented in Figure 3.1. The diagram gives an overview of the major steps that culminated in achieving the objectives of this research.

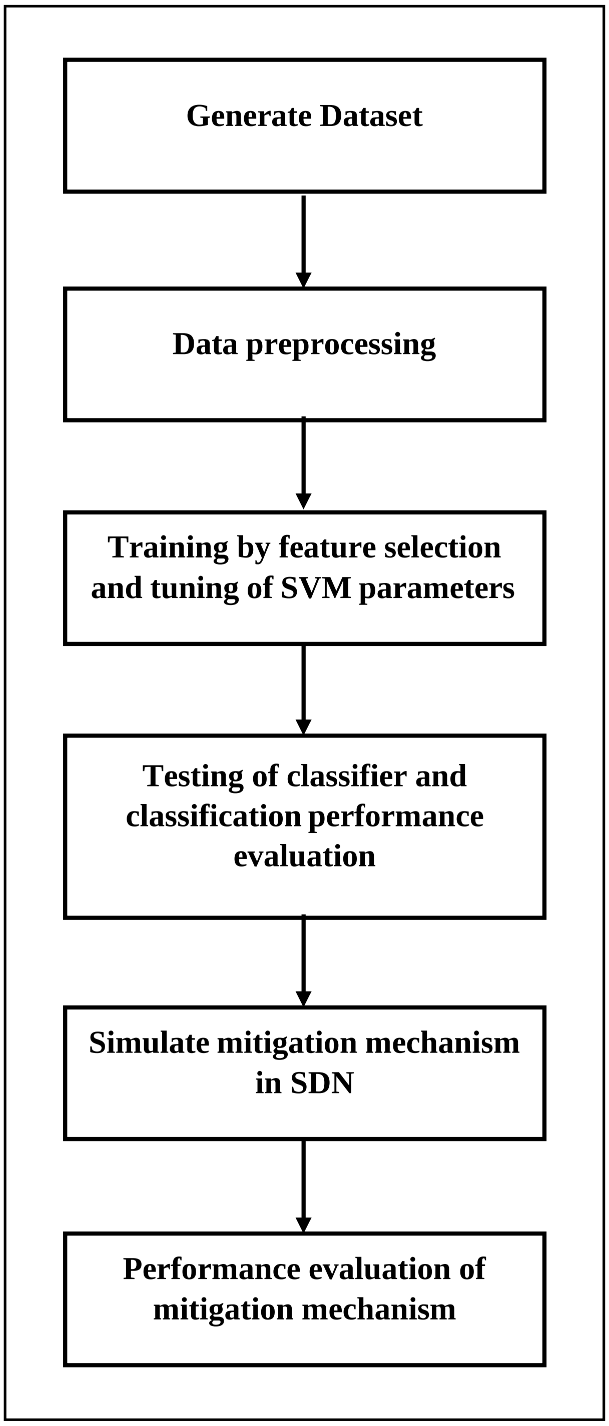

Generate Dataset
Data preprocessing
Training by feature selection and tuning of SVM parameters
Testing of classifier and classification performance evaluation
Simulate mitigation mechanism in SDN
Performance evaluation of mitigation mechanism

Figure 3.1: Proposed Solution Block Diagram

3.2 Generate Dataset

In this study, the GNS3 tool was used to set up the SDN environment. The network topology in GNS3 contained 20 virtual machines, five switches, and 24 links. Of the 20 virtual machines, it is comprised of eight slow HTTP attackers, eight legitimate clients, three web servers, and a Ryu controller. The visualization of the SDN environment is presented in Figure 3.2.

The five switches in the SDN topology is comprised of one OpenVSwitch and four datalink layer switches. The OpenVSwitch supports both OpenFlow and Netflow. The OpenFlow switch was configured to export Netflow records to the Ryu controller and acts as the gateway switch. That is, the OpenVSwitch determines which traffic accesses the internal network as it sits between the internal and external network. A Netflow record is created on the OpenVSwitch when traffic that passes through it reaches the ageing criteria for active and inactive flows. Unlike the OpenVSwitch, the datalink layer switches connect multiple virtual machines for seamless communication. The datalink switches minimized the use of several redundant network links in the setup. Once the datalink switch receives a packet, it checks its Content Addressable Memory (CAM) table for a mapping of the Media Access Control (MAC) address on the frame in the received packet to an output port. If such mapping does not exist, it sends the frame out all ports except the incoming port. If such a mapping exists, it sends the packet out of the port associated with the MAC address.

The Ryu controller written in python was used. Apart from Ryu, another controller which supports Netflow is the OpenDaylight controller which is written in Java. However, compared to the OpenDaylight controller, the Ryu controller is agile and can handle a higher traffic rate. Also, the development of applications for the Ryu controller is faster given that Ryu uses python as opposed to OpenDaylight controller which is based on Java. The Ryu controller was configured to receive Netflow exports from the OpenVSwitch and store the Netflow records received in an excel file format (xlsx) on its disk. The Netflow collector was implemented using python.

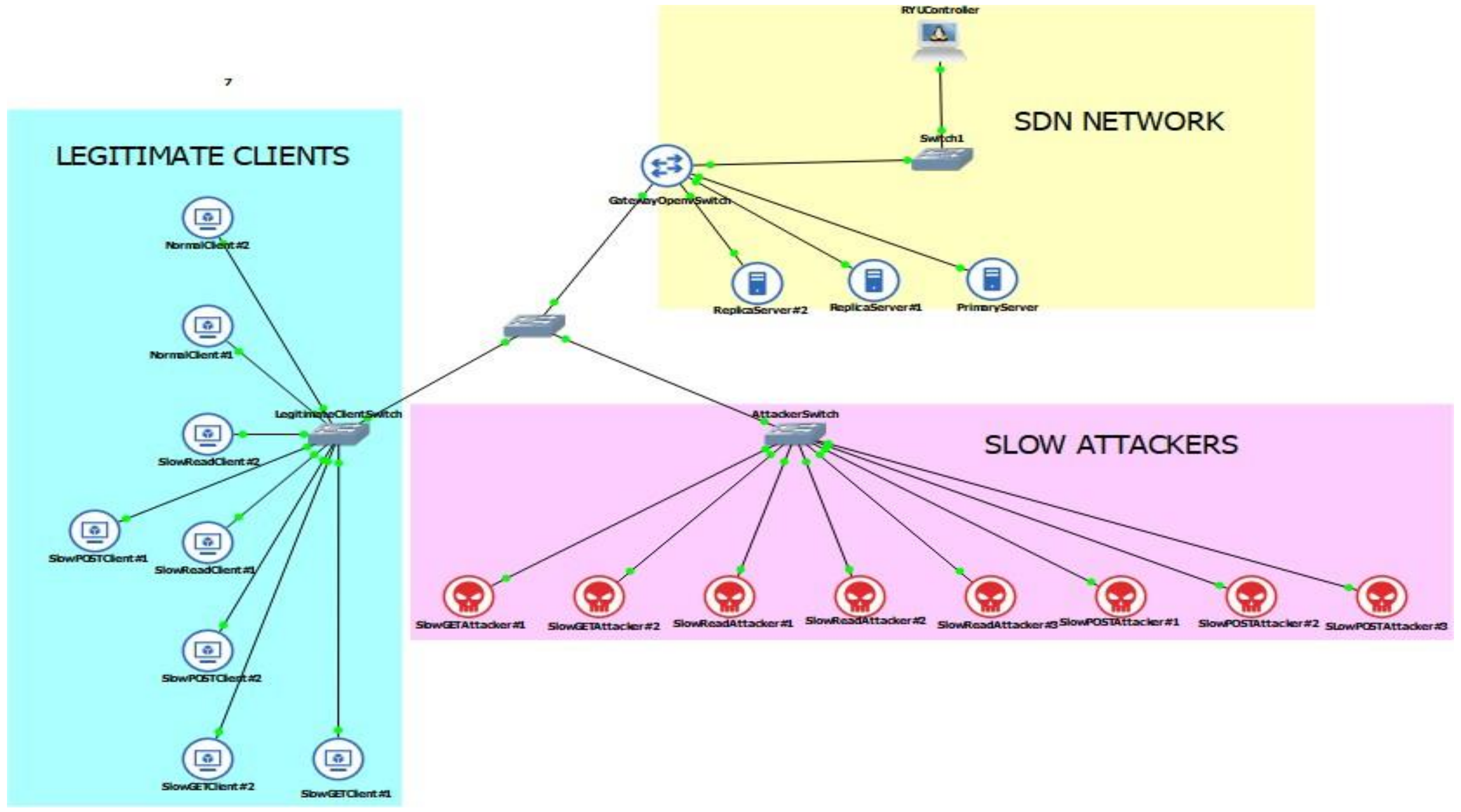


Figure 3.2: SDN Setup in GNS3

The three webservers consist of two replica servers and a primary server. The webservers are based on Lighttpd which is a light-weight web server. The content hosted on the webservers is a simple webpage written in Hypertext Markup Language (HTML) and PHP Hypertext Preprocessor (PHP) which displays a text with an image. The primary web server hosts all the content needed by the users of the network and it is the target of the attack. The replica servers serve the same content present on the primary webserver. However, they are not available to all the users of the network. An attack occurrence count is maintained to determine which replica webserver flagged attack traffic should be migrated to. A replica webserver is only available to an IP address that has been flagged as malicious while communicating with another webserver. The previous webserver the attacker was communicating with before the traffic was flagged as malicious could be the primary if the attack occurrence count is zero or any other replica server based on the attack occurrence count. Neither the attackers nor the legitimate clients are aware of the replica servers.

The eight slow HTTP DDoS attacker virtual machines are comprised of two slow get attackers and three slow post and slow read attackers each running Ubuntu 16.04 Long Term Support (LTS) operating system. The SlowHTTPTest tool was selected as the tool to generate attack traffic over Slowloris.py and PyLoris. As observed by Calvert and Khoshgoftaar (2019), the traffic patterns across SlowHTTPTest, Slowloris.py, and PyLoris attack generation tools are similar. However, Slowloris.py and PyLoris can only launch slow get attacks whereas, SlowHTTPTest can launch slow get, slow post, and slow read attacks. Furthermore, configuring the SlowHTTPTest tool from the command-line is easy to perform.

The eight legitimate client virtual machines are comprised of two clients that send get, post, and read requests each to the primary web server in a slow manner. In addition to the slow clients, two legitimate clients that send requests and retrieve responses from the primary web server at the rate allowed by the network bandwidth were included. The custom python script which utilises the http.client library was used to send requests and retrieve responses from the primary webserver. To simulate packet loss and delay, the Linux traffic control tool was installed on the slow legitimate clients. The Linux traffic control tool was used to change the configurations of the kernel packet scheduler to simulate packet delay, loss, and limit the bandwidth usage to create scenarios legitimately slow clients may be facing. The Linux traffic control mechanism was deployed from within the python script through the use of the sub-process module in python which allows the script to make operating system command calls and create new operating system processes. The egress bandwidth, the bandwidth of the outgoing packets, of the slow legitimate traffic clients was manipulated using the Linux traffic control tool by defining the maximum bandwidth rate and the latency of each outgoing packet.

As shown in the first step in Figure 3.3, to generate the slow HTTP DDoS attack dataset, slow read, get, and post attacks were launched from all the eight slow HTTP DDoS attackers using the SlowHTTPTest tool. The parameters used in the attack tool varied according to attack type as shown in Table 3.1, 3.2, and 3.3. The Ryu controller which was configured to receive and store the exported Netflow records that originated from the OpenVSwitch according to the second and third steps in Figure 3.3 was powered on before the attack was launched. Once each attacker reached the attack duration termination criteria specified in seconds, the attacker stopped sending malicious packets to the primary webserver. Consequently, the saved Netflow records on the Ryu controller were exported to the physical machine on which the simulation was performed. Next, the dataset of legitimate client traffic – slow client and normal client – was generated.

Similarly, according to the first step in Figure 3.3 on the legitimate client dataset creation, the legitimate clients first established a TCP connection with the primary web server using a custom python script. Then, requests for the webpage on the web server were sent and the responses received. Since the Ryu controller is still on, the resulting Netflow records generated by the OpenVSwitch were transmitted to the controller which captured and saved the records as an excel file according to the second and third steps in Figure 3.3 respectively. Consequently, the saved legitimate client Netflow records were also exported to the physical machine on which the simulation was performed.

The final step towards generating the dataset was achieved by consolidating all attack and legitimate client traffic records which were obtained independently. The client and attack traffic were not executed together so as not to hamper easy identification and labelling of attack and legitimate traffics in preparation for classification. The attack and legitimate client traffic Netflow records which were exported to the physical machine on which the simulation was performed were labelled as attack and legitimate traffic respectively. The label “0” represented legitimate client traffic Netflow instances while the label “1” represented the attack traffic Netflow instances. Consequently, the attack and legitimate client traffic Netflow instances were manually combined into a single excel file and the content of the resultant file was shuffled.

Table 3.1: Slow Get Parameters and Flags

Variable	**Value**
Number of connections (-c)	1000
Test duration in seconds (-l)	1000
Interval in seconds between follow up data (-i)	10
Connections per second (-r)	5

Table 3.1: Slow Get Parameters and Flags (continued)

Variable	Value
The maximum length of follow up data in bytes (-x)	10
Interval in seconds to wait for HTTP response onprobe connection(-p)	2

Table 3.2: Slow Post Parameters and Flags

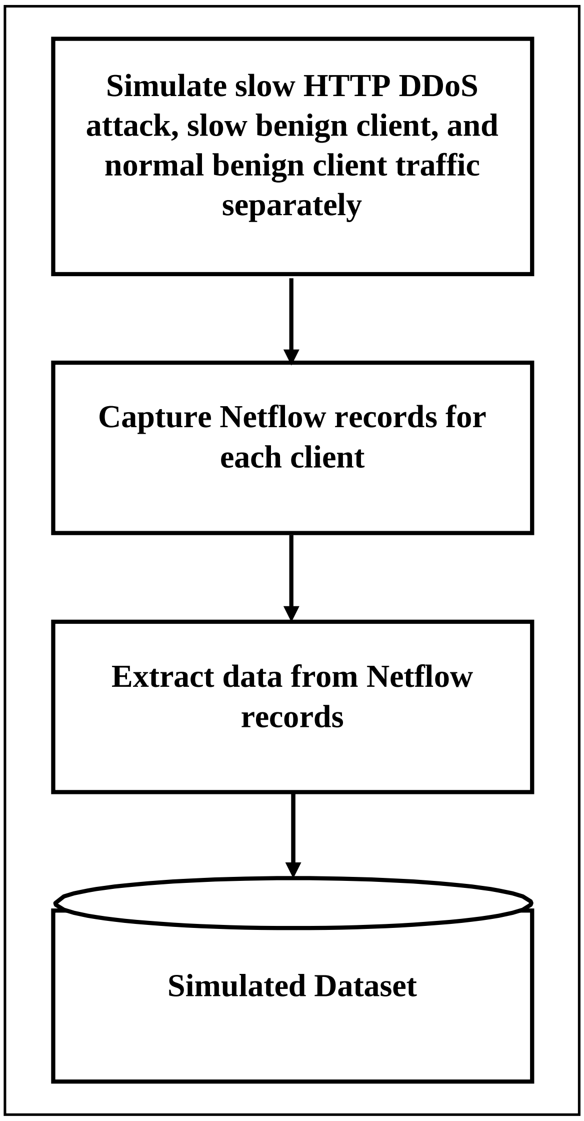
Variable	Value
Number of connections (-c)	1000
Test duration in seconds (-l)	1000
Interval in seconds between follow up data (-i)	4
Connections per second (-r)	5
The maximum length of follow up data in bytes (-x)	24
Interval in seconds to wait for HTTP response onprobe connection(-p)	3
Value of Content-length header in bytes (-s)	8192

Table 3.3: Slow Read Parameters and Flags

Variable	Value
Number of connections (-c)	1000
Test duration in seconds (-l)	1000
Start of the range for the TCP window size in bytes (-w)	3
Connections per second (-r)	5
End of the TCP advertised window range in bytes (-y)	24

Table 3.3: Slow Read Parameters and Flags (continued)

Variable	Value
Interval in seconds to wait for HTTP response onprobe connection (-p)	3
The interval between read operations in seconds (-n)	5
Number of bytes to read from receive buffer (-z)	15
Number of times the resource would be requested per socket (-k)	3



Simulate slow HTTP DDoS attack, slow benign client, and normal benign client traffic separately
Capture Netflow records for each client
Extract data from Netflow records
Simulated Dataset

Figure 3.3: Block Diagram of the dataset creation

3.3 Data Preprocessing

Data preprocessing refers to the transformation of data to a form suitable for the classifier to utilize. Aside from the data transformation step of feature construction implemented during Netflow export collection, other data preprocessing techniques were used to improve the quality of results to be obtained and prevent the overfitting of data. The correctness of any classifier depends on the quality of data supplied to it. A low-quality dataset will yield a poor classifier in that it may not be able to generalize classification when previously unseen data is introduced. Also, a low-quality dataset may cause the classifier to overfit the training dataset. This means that the model obtained from the classification process describes the training data too well which hampers its ability to classify the untrained dataset. The dataset obtained is described in this section and preprocessing operations performed on the dataset outlined.

3.3.1 Dataset

The Netflow dataset export in excel file contained 27 features which represents a high-level abstraction of the packets in each flow as shown in Table 3.4 according to their position in the dataset represented as the serial number. Feature construction was applied to each Netflow export during capture which added four more features, as shown in Table 3.5, bringing the total number of features in the dataset to 31. Feature construction was performed to improve the mining process given the features used by Calvert and Khoshgoftaar (2019) and Kemp *et al*. (2018). The dataset contained a total of 56,892 tuples where a 50:50 ratio was adopted for the attack to legitimate client traffic. The total amount of attack and legitimate client traffic tuples that constitute the dataset are 28,446 and 28,446 respectively. The 50:50 ratio adopted was aimed at ensuring the creation of a balanced SVM model which is to be applied to the simulated network at the controller. Also, Calvert and Khoshgoftaar (2019) in their work asserted that the 50:50 ratio of attack to legitimate client traffic instances resulted in the highest AUC values across most of the classifiers used including SVM.

Table 3.4: Netflow Version 5 Features

S/N	Feature Name	Description
1	Version	Netflow export format version number
2	Count	Number of flows exported (1-30)
3	Sys_Uptime	Current time in milliseconds since the export device booted
4	Unix_secs	The current count of seconds since 0000 UTC 1970
5	Unix_nsecs	Residual nanoseconds since 0000 UTC 1970
6	Flow_sequence	Counter of total flow sequence seen
7	Engine type	Type of flow switching engine
8	Engine_ID	Slot number of the flow switching engine
9	Sampling Interval	Interval of Netflow export sampling
10	Srcaddr	Source IP address
11	Dstaddr	Destination IP address
12	Nexthop	IP address of next-hop router
13	Input	Simple Network Management Protocol (SNMP) index of the input interface
14	Output	SNMP index of output interface
15	dPkts	Packets in the flow
16	dOctets	Total number of layer 3 bytes in the packets of the flow
17	First	Sysuptime at start of the flow
18	Last	SysUpTime at the time the last packet of the flow was received

Table 3.4: Netflow Version 5 Features (continued)

S/N	Feature Name	Description
20	Srcport	TCP/UDP source port number
21	Dstport	TCP/UDP destination port number
22	Tcp_flags	Cumulative OR of TCP flags
23	prot	IP protocol type (TCP = 6; UDP =17)
24	tos	IP type of service
25	Src_as	Autonomous system number of the source
26	Dst_as	Autonomous system number of the destination
27	Src_mask	Source address prefix mask bits
28	Dst_mask	Destination address prefix mask bits

Table 3.5: Constructed Features

S/N	Feature Name	Description
19	Diff.	The time difference in seconds between the last and first feature in the Netflow version 5 feature set
29	Packets/second	The number of packets per second (dPkts divided by diff.)
30	Bytes/second	The number of bytes per second (dOctets divided by diff)
31	Bytes/Packet	The number of bytes per second (dOctets divided by dPkts)

3.3.2 Data Reduction

Before feature selection, a form of data reduction, using genetic algorithm was performed, some features were removed from the dataset manually. The removed features pose the risk of model overfitting if the classifier were to be trained using a dataset that includes them. Table 3.6 shows the features that were removed manually with their corresponding position in the dataset represented as the serial number. Thirteen features were removed manually either due to zero values contained in all the tuples for the feature or the potential of the feature causing the classifier to overfit due to values that make a prediction of an attack or legitimate client traffic trivial. After the data reduction, the remaining 18 features were normalized.

Table 3.6: Features Removed Manually

S/N	Feature Names	Rationale
3	Sys_Uptime	The time interval between the start and end of attack traffic can be generalized which would not always be the case
4	Unix_secs	The time interval between the start and end of attack traffic can be generalized which would not always be the case
5	Unix_nsecs	The time interval between the start and end of attack traffic can be generalized which would not always be the case
6	Flow_sequence	The flow sequence numbers can be generalized which would not always be the case
9	Sampling Interval	All tuples contained zero value for the feature
10	Srcaddr	The same set of attack and legitimate client traffic IP address or from the same web server to a set of attack and legitimate client IP address.
11	dstaddr	Responses to the same set of attack and legitimate client traffic IP address or the same web server from a set of attack and legitimate client IP address.
12	Nexthop	All tuples contained zero value for the feature
24	tos	All tuples contained zero value for the feature

Table 3.6: Features Removed Manually (continued)

S/N	Feature Names	Rationale
25	Src_as	All tuples contained zero value for the feature
26	Dst_as	All tuples contained zero value for the feature
27	Src_mask	All tuples contained zero value for the feature
28	Dst_mask	All tuples contained zero value for the feature

3.3.3 Data Normalization

The dataset with 18 features was normalized using the standard scaler of sklearn preprocessing to give all features equal weight. Data normalization increases the training speed of a classifier and also prevents any feature in the dataset from outweighing the other features with smaller ranges. The standard scaler implementation, as shown in equation 3.1, uses the zero-mean (z-score) normalization to standardize the values of each feature in a tuple. In z-score normalization, the values of a feature are standardized based on the mean and standard deviation of that feature. A value v_i of a feature is normalized to v_i' by determining

$$v_i' = \frac{v_i - A}{\sigma_A} \tag{3.1}$$

where A and σ_A are the mean and standard deviation respectively (Agarwal, 2014).

3.4 Training and Testing of Slow DDoS RBF SVM Classifier

In Figure 3.4, the breakdown of the steps taken for the selection of relevant features from the simulated dataset and the appropriate SVM regularization strength (C) and gamma (γ) parameters is illustrated. The selection of relevant features and the Radial Basis Function (RBF) SVM kernel parameters helps to provide a model that is suitable for generalizing on datasets.

3.4.1 Feature Selection

Genetic algorithm was executed on the normalized dataset to extract features that are most relevant to determining whether a flow in a Netflow record is attack traffic or legitimate client traffic whether normal or slow due to the bandwidth of the network. The fitness function of the features selected is determined by the accuracy of a linear SVM classifier subjected to four-fold cross-validation. The rationale behind the usage of a linear SVM classifier is that a linear SVM will ensure that the features selected are free from the bias of a regularization parameter or gamma as evident in a radial basis function kernel-based SVM classifier. The selected features were stored and used to determine the best regularization parameter, C, and gamma for the radial basis function kernel-based SVM used for the model creation.

3.4.2 RBF SVM Kernel Parameter Selection

The dataset containing tuples of the selected features was used with genetic algorithm to determine the best parameters to use in creating the final SVM model which distinguishes attack from legitimate client traffic. A range of possible parameter values was defined for the SVM regularization parameter C and the gamma parameter. For the C parameter, a range of integer values from 1 to 10 was defined while for the gamma parameter, a real-valued range from 0.1 to 1 was defined using numpy’s arange function. The parameter range was defined based on the research work by Kokila *et al.* (2015).

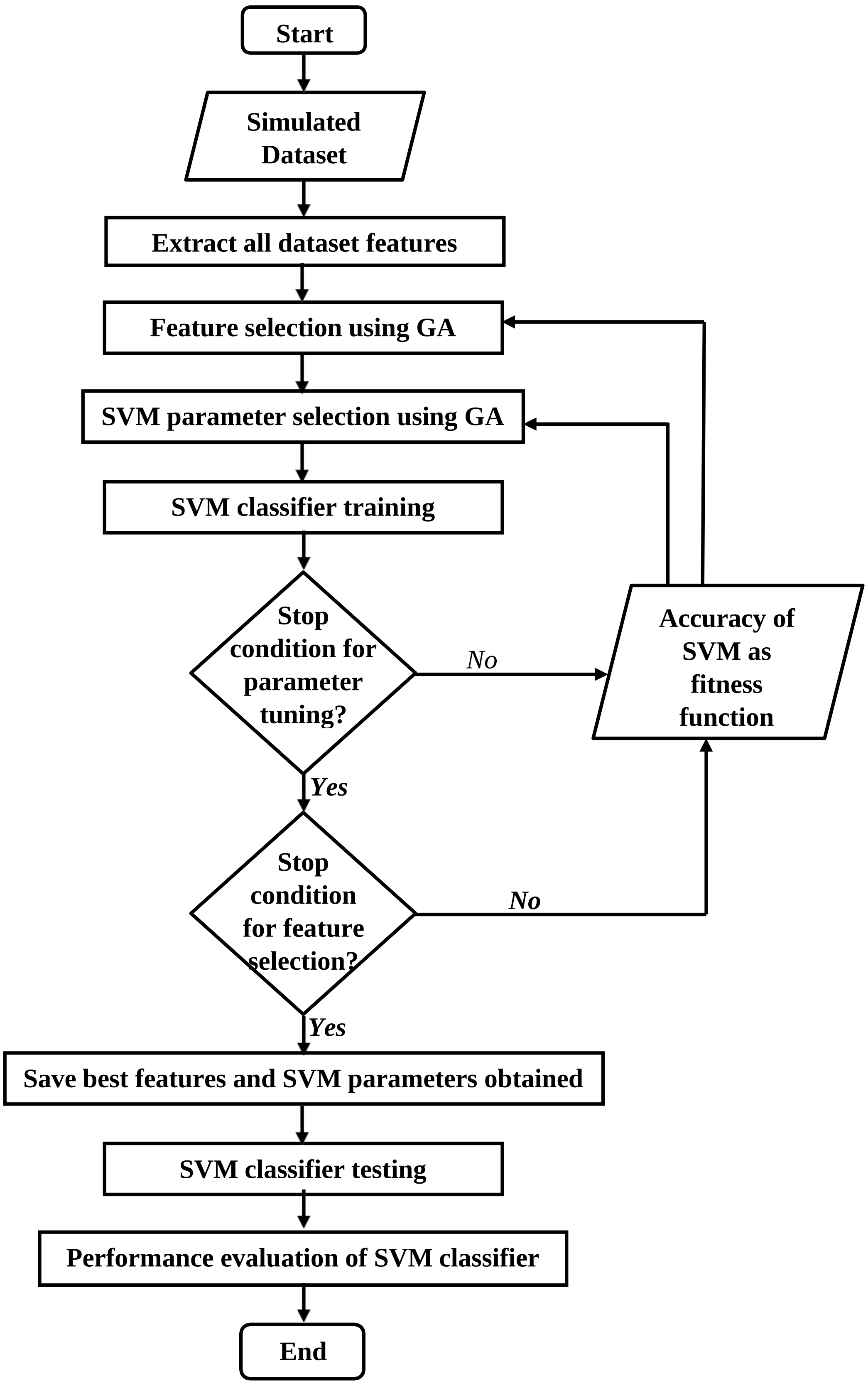

Start
Simulated Dataset
Extract all dataset features
Feature selection using GA
SVM parameter selection using GA
SVM classifier training
Stop condition for parameter tuning?
No
Accuracy of SVM as fitness function
Yes
Stop condition for feature selection?
No
Yes
Save best features and SVM parameters obtained
SVM classifier testing
Performance evaluation of SVM classifier
End

Figure 3.4: Training and Testing flowchart for RBF SVM

3.4 Simulation of RBF SVM and Mitigation Mechanism

Simulation of the RBF SVM detection model and selective adaptive bubble burst mitigation mechanism is executed as shown in Figure 3.5. Internal operations of the SABB mitigation module is described in Figure 3.6. The RBF kernel SVM model was extracted using the python programming language object serialization package, pickle. Object serialization is the conversion of an object, in a programming language, to a series of bytes for storage or transmission over a network. The serialized SVM model was transmitted to the RYU controller in the SDN simulation for deserialization and use to analyze real-time Netflow records of traffic traversing the network. Deserialization is the conversion of a series of bytes into a replica of the original object which was serialized.

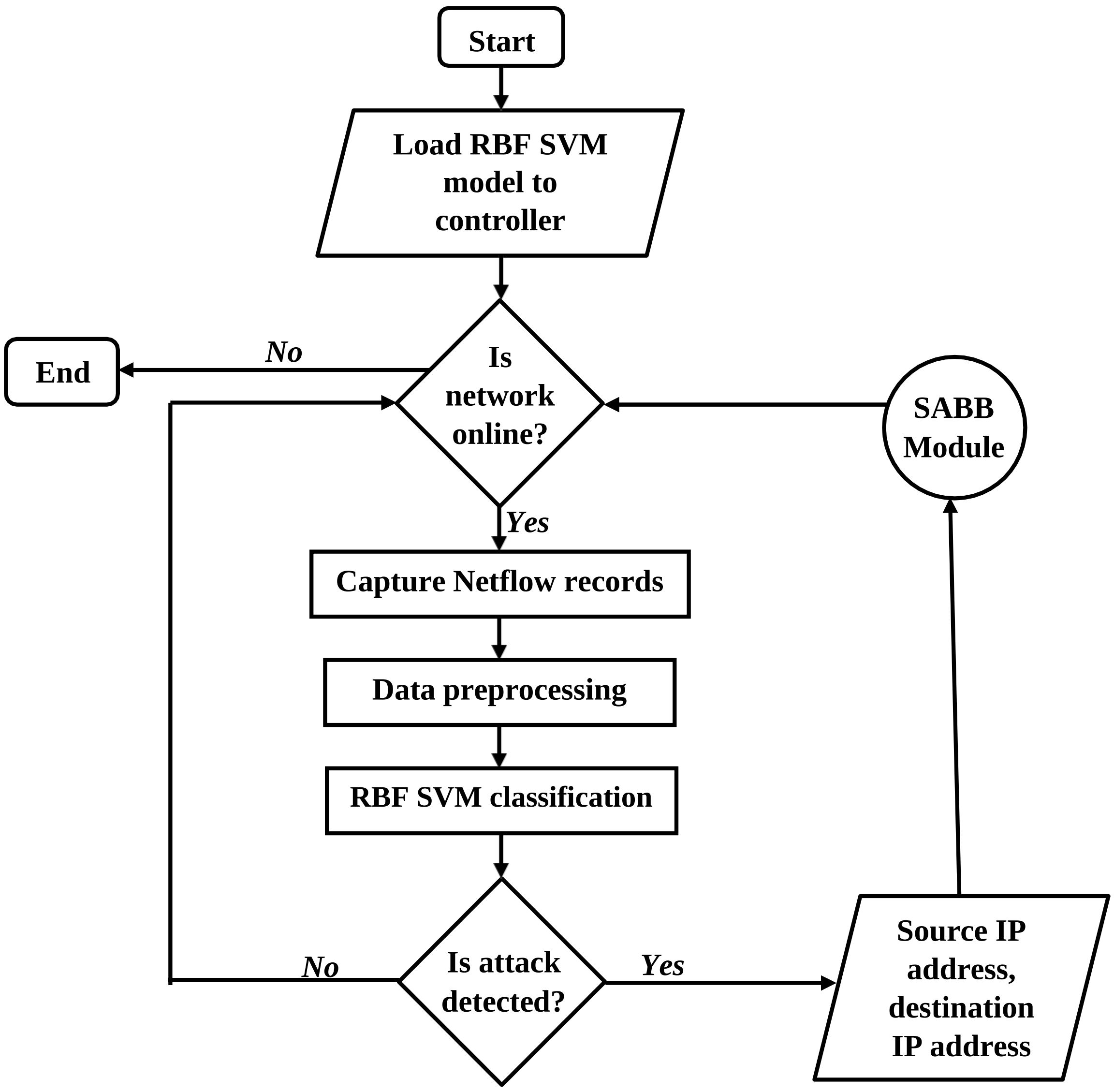


Figure 3.5: Simulation flowchart of RBF SVM in SDN

3.4.1 Selective Adaptive Bubble Burst (SABB) Module

As shown in Figure 3.5, once the last IP address of the webserver replica set is the destination address of the suspicious client, the IP address of the client gets blocked at the gateway switch. Conversely, if the destination IP address is not the last in the replica set, the next destination address in the replica set after the current destination address is assigned as the new address for all new incoming connections from the suspicious IP address.

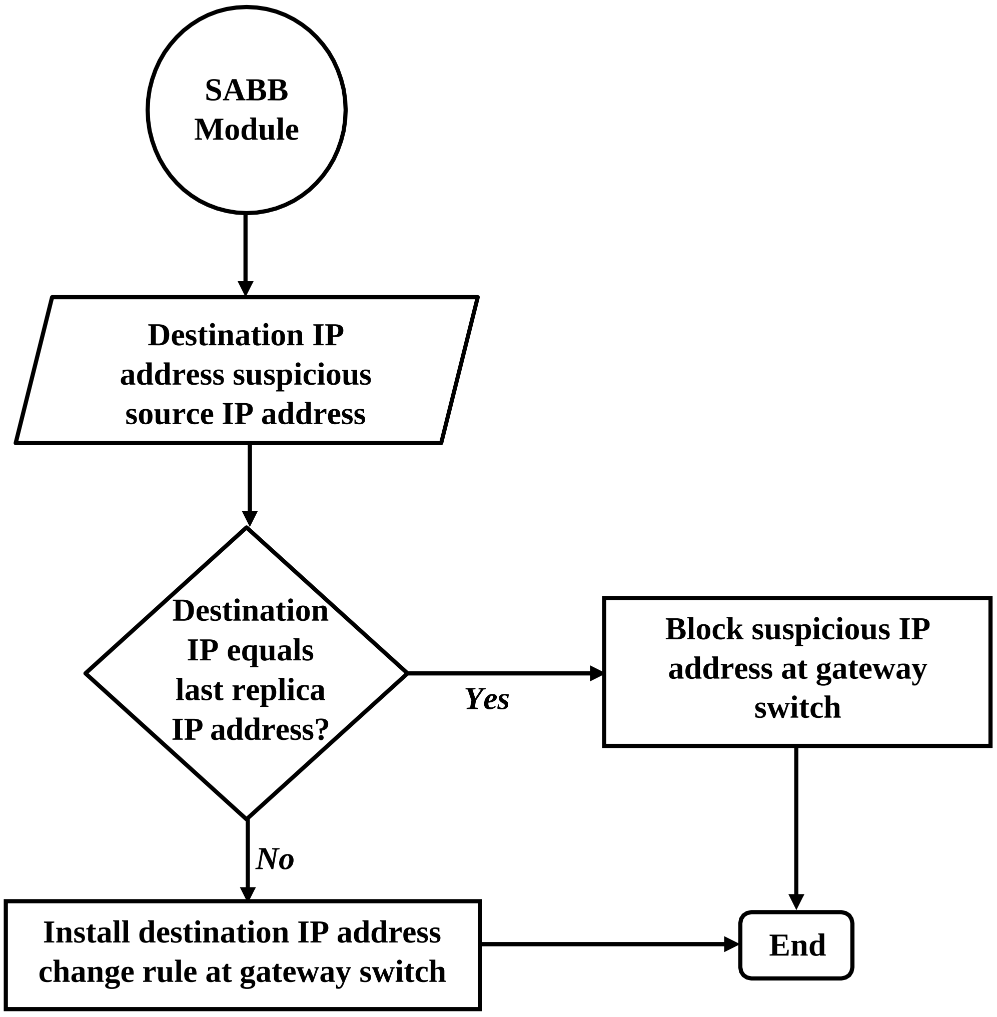

SABB Module
Destination IP address suspicious source IP address
Destination IP equals last replica IP address?
Yes
Block suspicious IP address at gateway switch
No
Install destination IP address change rule at gateway switch
End

Figure 3.6: SABB Mitigation Module flowchart

The operations of the SABB module is modelled mathematically by taking into consideration the number and addresses of the web servers within the network and the connections made to the primary server by various devices sending either legitimate or attack traffic. The mathematical model of the SABB module is given in equation 3.9. Since the webservers are the network asset being protected, they are represented as a set in equation 3.2 as:

$$\{y_1, y_2, y_3, \dots, y_n\} \tag{3.2}$$

where y_1 is the primary web server and $\{y_2, y_3, \dots, y_n\}$ are the replica servers. In equation 3.2, n represents the total number of web servers such that $n \geq 2$.

Since the RBF SVM model handles attack detection, when an attack is not detected, equation 3.3 holds while when an attack is detected, equation 3.4 holds.

$$\beta_x = 0 \tag{3.3}$$

$$\beta_x = \beta_x + 1 \tag{3.4}$$

where β_x is the attack occurrence count from IP address x.

Furthermore, the connection i from the IP address x to the webserver y_1 when equation 3.3 holds is given in equation 3.5 as:

$$\sigma_{ci(x)} = (x, y_1) \tag{3.5}$$

Similarly, for every new connection, j, from IP address x when equation 3.4 holds is given in equation 3.6 as:

$$\sigma_{cj(x)} = (x, y_{\beta_x + 1}) \tag{3.6}$$

where $j = i + 1, i + 2, \dots, m$.

Therefore, to block a connection, the connection block parameter represented as $\mu_x^{\beta x}$ is a function of the attack occurrence count, β_x, of a given IP address x. Equation 3.7 presents the connection block parameter model as:

$$\mu_x^{\beta_x} = \begin{cases} 1, & \beta_x > n \\ 0, & \beta_x \leq n \end{cases} \tag{3.7}$$

where 1 means blocked and 0 means open.

The connections made from IP address x which is visible to the system is given in equation 3.8 as:

$$\rho_x = \sigma_{ci(x)} + \sum_{k=0}^{\beta_x} (1 - \mu_x^k) \cdot \sigma_{cj(x)} \tag{3.8}$$

The major function of the SABB module is to monitor the flow status from IP x given in equation 3.9 as:

$$\delta_x = min\,(\bigcup_{k=0}^{\beta_x} (1 - \mu_x^k)) \tag{3.9}$$

When the flow is blocked, the flow status is zero. That is, $\delta_x = 0$. But when the flow is active, the flow status returns one. That is, $\delta_x = 1$.

The algorithmic model for the execution of SABB is given as follows.

Algorithm 1: SABB Online Execution Model

```
Input: SVM Model $\theta$, Netflow record of IP source IP address $x$ is $N_{fx}$, primary
        webserver $y_1$, replica set of webservers $\{y_\pi\}$ where $\{y_\pi\} = \{y_2, y_3, \cdots, y_n\}$ :
        $n \geq 2 \wedge n = |y_1 \cup \{y_\pi\}|$
Output: Boolean block value $\mu_x$ for IP address $x$
Procedure $onlineSABB()$
1       $while\ network$ is $online\ \ do$

2           Connection $i$ from $x$ is $\sigma_{ci(x)} \leftarrow (x, y)$
3           $\mu_x \leftarrow false$
4           Netflow class $N_c \leftarrow predictClassSVM\ \theta(N_{fx})$

5           $if\ N_c\ is\ attack$
6               Sender-destination pair $(x, y) \leftarrow getSenderDestPair(N_{fx})$ : $x \notin$
                $\{y_1 \cup \{y_\pi\}\}$

7               $if\ y \neq y_n$
8                   $y' \leftarrow serverAfter(y)$
9                   New connections $j$ from $x$ is $\sigma_{cj(x)} \leftarrow (x, y')$
10              $else\ if\ y = y_n$
11                  $\mu_x \leftarrow true$

12              $switchPropagate(\mu_x)$
```

The Selective Adaptive Bubble Burst Mitigation Mechanism operations given in Algorithm 1 is a function of the number and IP addresses of the web servers ($y_1...y_n$) within the network and the connection requests $\sigma_{ci(x)}$ (where σ_c is the connection symbol, *i* is the connection number, and *x* is the client's IP address), attack or legitimate, sent to the primary web server (y_1). Here, *x* is the client's source IP address and the Netflow record of source IP *x* is depicted by N_{fx}. The SVM model generated after the classification of the dataset is denoted by θ. A change in the value of destination IP address *y* expressed as y' occurs when the classification of N_{fx}, Netflow record of *x*, gives a class category N_c which is an attack. Once the number of times N_c yields as attack traffic causes y' to equal the last replica server y_n, the block parameter μ_x for the defaulting IP address x is propagated to the switch which blocks all traffic from IP address x.

3.5 Performance Evaluation

Performance evaluation of the methods employed to achieve the objectives outlined measures the degree to which the classifier predicts the class labels of instances and the extent to which web services are available to legitimate clients.

3.5.1 RBF SVM Performance Evaluation

Performance evaluation of RBF SVM represents the degree to which the RBF SVM classifier detected the attack and legitimate client traffic instances. The performance was measured using metrics of accuracy, False Positive Rate (FPR), False Negative Rate (FNR), and Area Under the Receiver Operating Characteristics Curve (AUC).

3.5.1.1 Accuracy

The accuracy of a classifier refers to how well the classifier distinguishes attack traffic from normal traffic. Mathematically, accuracy is represented in equation 3.10 (Agarwal, 2014).

$$Accuracy = \frac{TP+TN}{P+N} \tag{3.10}$$

where True Positive (TP) corresponds to the number of attack instances in the dataset that were correctly classified as attack traffic. Similarly, True Negative (TN) refers to the number of legitimate traffic instances in the dataset that were correctly classified as legitimate traffic. Positive (P) corresponds to the total number of instances in the dataset which are attack traffic while Negative (N) corresponds to the total number of legitimate traffic in the dataset.

3.5.1.2 False Positive Rate (FPR)

The False Positive Rate (FPR) of a classifier refers to the percentage of negative instances incorrectly classified as positive. Here, since the positive instances are the attack traffic instances and the negative instances are the legitimate traffic instances, FPR refers to the percentage of normal traffic incorrectly classified as attack traffic. Mathematically, FPR is expressed in equation 3.11.

$$FPR = \frac{FP}{FP+TN} \times 100 \tag{3.11}$$

where False Positive (FP) corresponds to the number of legitimate client instances labelled by the classifier as attack instances.

3.5.1.3 False Negative Rate (FNR)

The False Negative Rate (FNR) of a classifier refers to the percentage of positive instances incorrectly classified as negative. Here, since the positive instances are the attack traffic instances and the negative instances are the legitimate traffic instances, FNR defines the percentage of attack traffic labelled as normal traffic. FNR is mathematically represented in equation 3.12 as:

$$FNR = \frac{FN}{FN+TP} \times 100 \tag{3.12}$$

where False Negative (FN) corresponds to the number of attack traffic instances labelled by the classifier as legitimate client traffic instances.

3.5.1.4 Area Under the Receiver Operating Characteristics Curve (AUC)

Area Under the Receiver Operating Characteristics (AUC) refers to the degree of the classifier’s separability based on the Receiver Operating Characteristic (ROC) curve. ROC curves show the trade-off between true positive rate and false positive rate. The

AUC is defined by calculating the area under the curve created when the True Positive Rate (TPR), the percentage of attack traffic classified as attack instances, is plotted against the FPR.

3.5.2 SABB Mitigation Performance Evaluation

The performance of the SABB mitigation mechanism was measured using the average response time of the webservers and the ratio of completed to timed-out requests for legitimate clients.

3.5.2.1 Average Response Time

The web server's response time refers to how fast it replies to requests from legitimate clients during attack and non-attack scenarios. In this work, 100 requests are sent to the primary web server and the average response time of the total requests is obtained. The formula is expressed in equation 3.13 as:

$$\textit{Average Response Time} = \frac{\sum_{i=1}^{n} Rt_i}{n} \tag{3.13}$$

where n refers to the number of requests sent to the webserver and Rt_i refers to the response time of the ith request.

3.5.2.2 Ratio of Completed to Timed-out Requests

The ratio of completed to timed-out requests was used to measure the availability of services to legitimate clients during attack and non-attack scenarios. In this work, requests that receive an associated response is considered completed.

3.6 Chapter Summary

This chapter presents the methodology involved in dataset creation, classification, and slow DDoS mitigation module in SDN. The dataset was generated in an SDN network simulated in GNS3 using OpenVSwitch as the Netflow exporter and the Ryu controller as the Netflow collector. The dataset created was preprocessed using various preprocessing methods. The preprocessed dataset was used to train an SVM classifier to generate an SVM model. The SVM model was transferred to the Ryu controller on the SDN network to carry out real-time slow DDoS detection. The performance analysis of the slow DDoS classification and mitigation with the discussion of the results obtained are presented in chapter four.

CHAPTER FOUR

4.0 RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter presents the results obtained by the feature selection and RBF SVM parameter tuning process by GA, the performance of the classifier, and the performance of the SABB slow HTTP DDoS attack mitigation mechanism. The results obtained are discussed and insights are drawn from each.

4.1 Feature Selection Result

In the dataset generated, 31 features were consisting of 27 Netflow version 5 specific features and 4 constructed features. After performing data reduction and normalization on the dataset which resulted in 18 features, genetic algorithm was used to select the best features that aid in detecting a slow DDoS attack. The genetic algorithm was initialized with population size and generation number of 10 using a tournament selection of size 3 after extensive experimentation. The accuracy obtained as a fitness function for all 10 generations is described in Table 4.1. Furthermore, eleven features out of the 18 features were selected to have significance in aiding the detection of slow DDoS attacks. The features selected are listed in Table 4.2 where the serial number represents the position of the feature in the raw dataset of 31 features.

Table 4.1: Genetic Algorithm Feature Selection Generation Accuracy

Generation	Maximum Accuracy per Generation
0	98.35%
1	98.64%
2	98.64%
3	98.64%
4	98.64%
5	98.67%
6	98.67%
7	98.72%
8	98.72%
9	98.72%
10	98.72%

Table 4.2: Selected Features

S/N	Features
2	Count
13	Input
14	Output
15	dPkts
16	dOctets
18	Last
19	Diff
20	Srcport
22	Tcp_flags

Table 4.2: Selected Features (continued)

S/N	Features
29	Packets/second
31	Bytes/packet

4.2 RBF SVM Parameter Selection Result

The regularisation and gamma parameters of the radial basis function kernel-based SVM were selected using the chosen features with genetic algorithm. The accuracy of the estimator, an SVM classifier, was used as the fitness function of the genetic algorithm. A generation number of 5 was used with a population size of 10 and tournament size of 3 after extensive experimentation using other parameters. The accuracy of the estimator per generation is described in Table 4.3. The initial generation had the best individual which was selected by the genetic algorithm with an accuracy of 99.71% and SVM regularization and gamma parameters of 8 and 0.798 respectively.

Table 4.3: **Genetic Algorithm RBF SVM Parameter Generation Accuracy**

Generation	Maximum Accuracy
0	99.71%
1	99.68%
2	99.68%
3	99.68%
4	99.68%
5	99.68%

4.3 RBF SVM Classification Result

The 80:20 ratio of training to testing dataset was utilised in executing the classification task. On training of the RBF SVM classifier using the kernel parameters C and gamma as 8 and 0.798 respectively, the testing of the model was performed using the test dataset ratio, which subsequently resulted in an accuracy of 99.89% with an AUC of 99.89%. Table 4.4 shows the percentages recorded for the performance metrics of accuracy, AUC, TPR, FNR, and FPR which were used to gauge the performance of the classifier. A total of 11,379 tuples constituted the test dataset of which 5,650 tuples and 5,729 tuples were legitimate client and attack traffic respectively. For the legitimate client traffic, 5,640 tuples were correctly classified as legitimate traffic while 10 tuples were misclassified as attack traffic by the classifier model. For the attack traffic, 5,726 tuples were correctly classified as attack traffic while 3 tuples were misclassified as legitimate client traffic. Therefore, the true positive, true negative, false positive, and false negative tuple count which is the summary of the confusion matrix are shown in Table 4.5. Figure 4.1 provides the details presented by the summary in Table 4.5.

Table 4.4: RBF SVM Classifier Performance Metric

Performance Metric	**Percentage**
Accuracy	99.89%
AUC	99.89%
TPR	99.95%
FPR	0.18%
FNR	0.05%

Table 4.5: Confusion Matrix Summary of RBF SVM Classification

TP	TN	FP	FN	Total
5,726	5,640	10	3	11,379

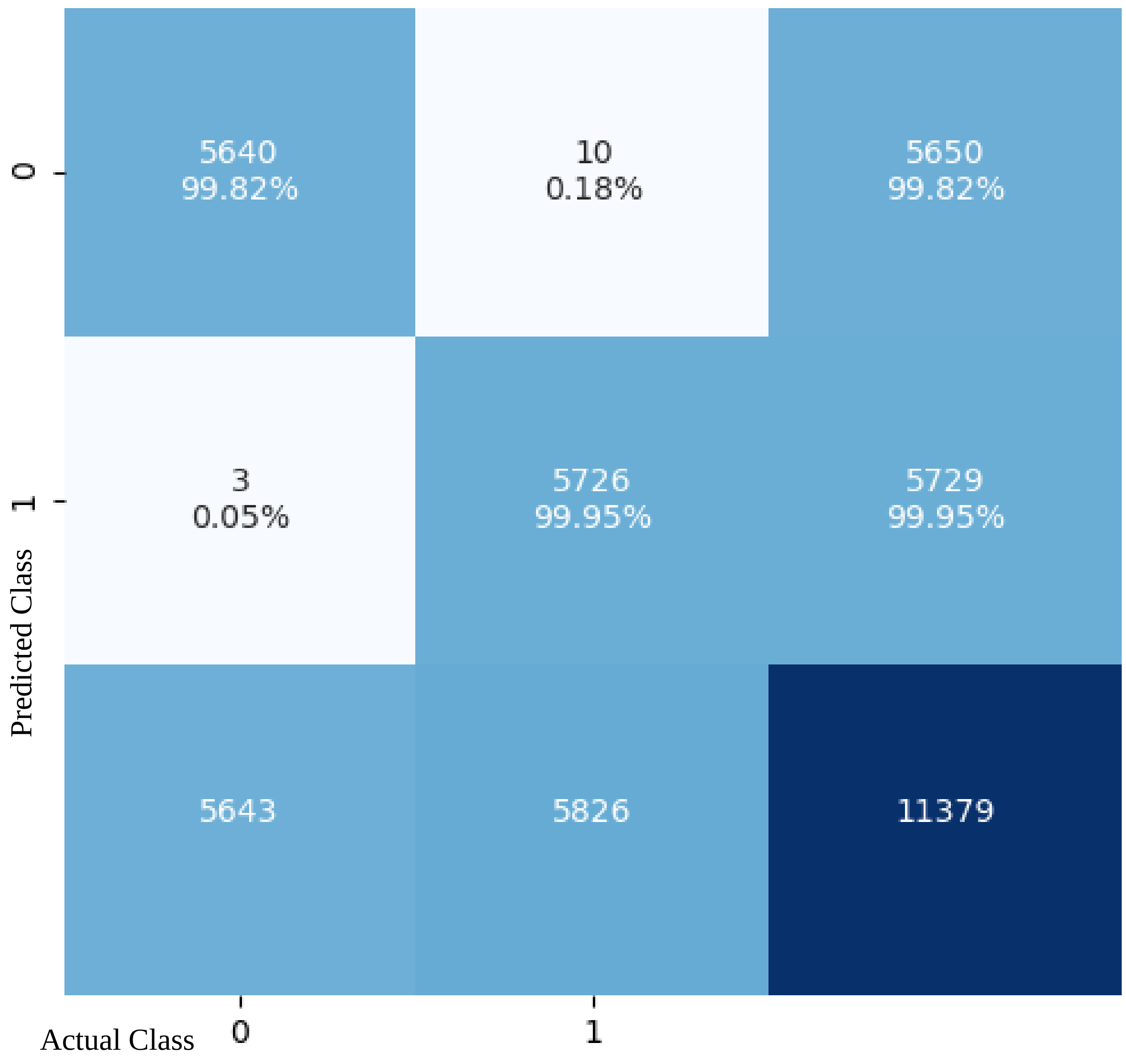


Figure 4.1: RBF SVM Classification Confusion Matrix

In Figure 4.1 which presents the details of the RBF SVM performance in a confusion matrix, the horizontal axis represents the actual class while the vertical axis represents the predicted class. The “0” on both axes represents actual and predicted legitimate client instances while the “1” on both axes represents actual and predicted attack traffic instances. As shown in the confusion matrix in Figure 4.1, the classifier achieved a false positive rate, false negative rate, true positive rate, and true negative rate percentage of 0.18%, 0.05%, 99.95%, and 99.82% respectively. The classifier's false positive rate is identified in the confusion matrix highlighting the point of intersection between the "1" on the actual class horizontal axis and the "0" on the predicted class vertical axis. Similarly, the classifier's false negative rate is identified in the confusion matrix by highlighting the point of intersection between the "0" on the actual class horizontal axis and the "1" on the predicted class vertical axis. Furthermore, the classifier's true positive rate is identified in the confusion matrix by highlighting the point of intersection between the "1" on the actual class horizontal axis and the "1" on the predicted class vertical axis while the true negative rate is identified in the confusion matrix by highlighting the point of intersection between the "0" on the actual class horizontal axis and the "0" on the predicted class vertical axis.

For the FPR result of 0.18%, it implies that for every 10,000 records which are legitimate client traffic, 18 out of the 10,000 tuples would be misclassified as attack traffic. In this work, the RBF SVM classifier resident in the controller collects Netflow traffic in real-time and processes them immediately thus, classifying each Netflow record of a particular IP address. Each Netflow record can only contain a maximum of 30 flow records which corresponds to 30 tuples in the dataset. Therefore, the possibility of misclassification of traffic is negligible. Although in some exceptional cases, a flow in the record can be classified as malicious, switching the flow traffic from one of the webservers to the other provides room for better analysis of subsequent flows from the IP address to the newly allocated webserver.

The FNR of 0.05% shows that it would be difficult for attack traffic, slow get, post, or read, to circumvent the detection module. Since in every Netflow record export only 30 flows are transmitted to the controller based on the transmission criteria, the possibility of misclassifying attack traffic to be a legitimate client's traffic is negligible. With this performance percentage, attackers are identified swiftly and the appropriate action is performed.

4.4 SABB Mitigation Process Result

For the SABB slow HTTP DDoS mitigation process to function, the RBF SVM model obtained from the training phase was saved and uploaded to the Ryu controller in the GNS3 SDN simulated environment for real-time slow HTTP DDoS detection. The slow HTTP DDoS attacks were launched using eight computers running on Ubuntu operating system using the SlowHTTPTest tool as shown in Figure 3.2. The average response time of the webservers to HTTP requests together with the ratio of completed to timed-out requests were measured before and after the SABB module was activated. The average response time of the webservers to 100 legitimate requests before and after the activation of the SABB module during the attack is shown in Table 4.6.

Table 4.6: Average Request Response Time with and without SABB during slow HTTP DDoS attack

Number of Attackers	Average Response Time without SABB (ms)	Average Response Time with SABB (ms)
1	35.948	4.411
2	40.191	15.370
3	89.392	120.076
4	137.802	128.374
5	379.855	167.264
6	436.211	412.714
7	504.328	275.73
8	1121.369	387.743

As observed in Table 4.6, the average response time increased for scenarios when the SABB module was active as well as when it was active. However, when the SABB module is inactive during the attack, the average response time increased at a faster rate than when the SABB module was active. This difference in average response time is attributed to the activity of the SABB module which inspects the traffic received and blocks malicious traffic promptly thereby freeing up the bandwidth occupied by the attackers which reduce the response time. A graphical view of the average response time presented in Table 4.6 is shown in figure 4.2.

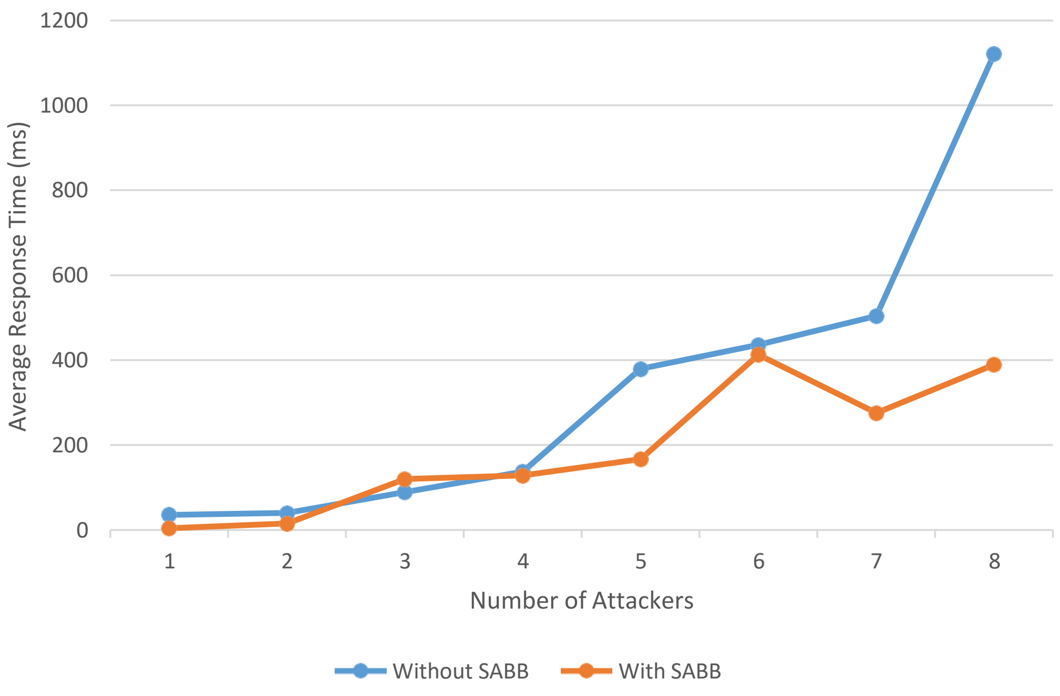


Figure 4.2: Average Response Time of the webserver

In addition to measuring the effectiveness of the SABB module through the response time of requests made by legitimate clients, the percentage ratio of requests completed by the webserver to the percentage ratio of requests which timed-out was measured. A request is completed if an associated response to the request is sent by the webserver and received by the requesting client. Table 4.7 shows the percentage ratio of completed to timed-out legitimate requests when the SABB module was inactive. It can be observed that as the number of attackers increased, the percentage of completed requests reduced while the percentage of timed-out requests increased. The reduction in the percentage of completed requests is due to the utilization of the webserver's resources by the attackers thus making those resources unavailable to legitimate clients. The graphical illustration of the data presented in Table 4.7 is shown in figure 4.3.

Table 4.7: Completed to Timed-out Request Ratio without SABB

Number of Attackers	Completed Requests (%)	Timed-out Requests (%)
1	100	0
2	100	0
3	98	2
4	98	2
5	91	9
6	93	7
7	91	9
8	76	24

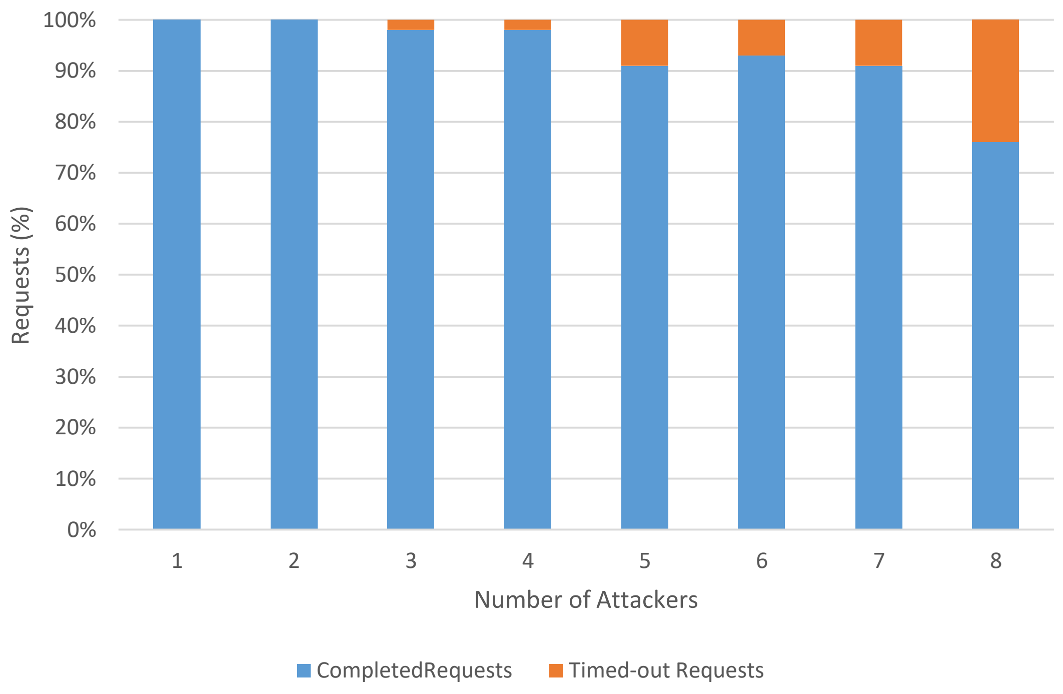


Figure 4.3: Completed to Timed-out Request Ratio without SABB

Furthermore, the percentage ratio of completed to timed-out requests when SABB was active was recorded. As shown in Table 4.8, it is observed that although the percentage ratio of completed requests reduces with an increase in the number of slow HTTP DDoS attackers, the rate of reduction is lower compared to when SABB was inactive. The lower rate of reduction in completed requests as the number of attackers increase is a result of the blocking and flow modification operations of the SABB module. Once an attacker is detected, the SABB module modifies the flow of the attacker’s requests from the primary webserver to another replica server in a manner transparent to the attacker. This frees up the resources on the primary web server thereby reducing the number of incomplete requests recorded by legitimate clients of the primary webserver. Figure 4.4 shows the graphical representation of the data presented in Table 4.8.

Table 4.8: Completed to Timed-out Request Ratio with SABB

Number of Attackers	Completed Requests (%)	Timed-out Requests (%)
1	100	0
2	100	0
3	99	1
4	98	2
5	97	3
6	95	5
7	94	6
8	92	8

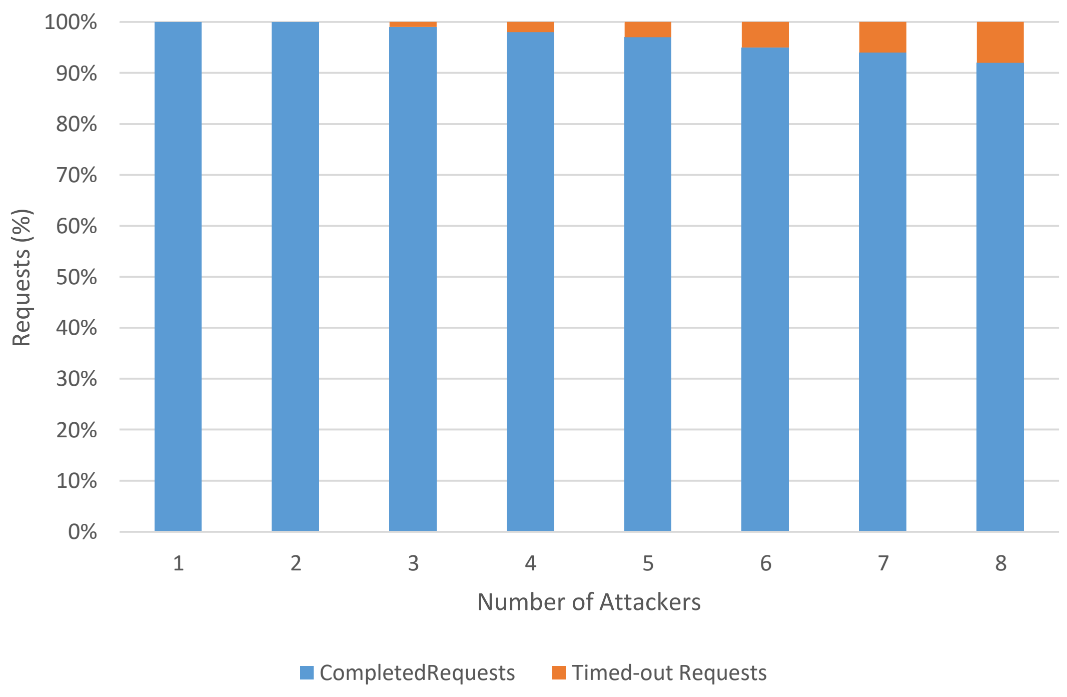


Figure 4.4: Completed to Timed-out Request ratio with SABB

To further illustrate the significance of the SABB module in ensuring a high rate of request completion, the percentage of completed requests when SABB was active as well as when it was inactive were juxtaposed in Figure 4.5. As shown in Figure 4.5, the percentage of completed requests when SABB was active supersedes the percentage of completed requests when SABB was inactive for most attack scenarios. Attack scenarios where the number of slow HTTP DDoS attackers was either one or two recorded a 100% completed legitimate request rate when SABB was active as well as when it was inactive. This occurred because the capacity of the webserver's resources had not been exhausted by the attackers hence the legitimate clients were able to complete their requests. However, the difference between both scenarios, when the number of attackers was either one or two, is evident in the average response time recorded in Table 4.6.

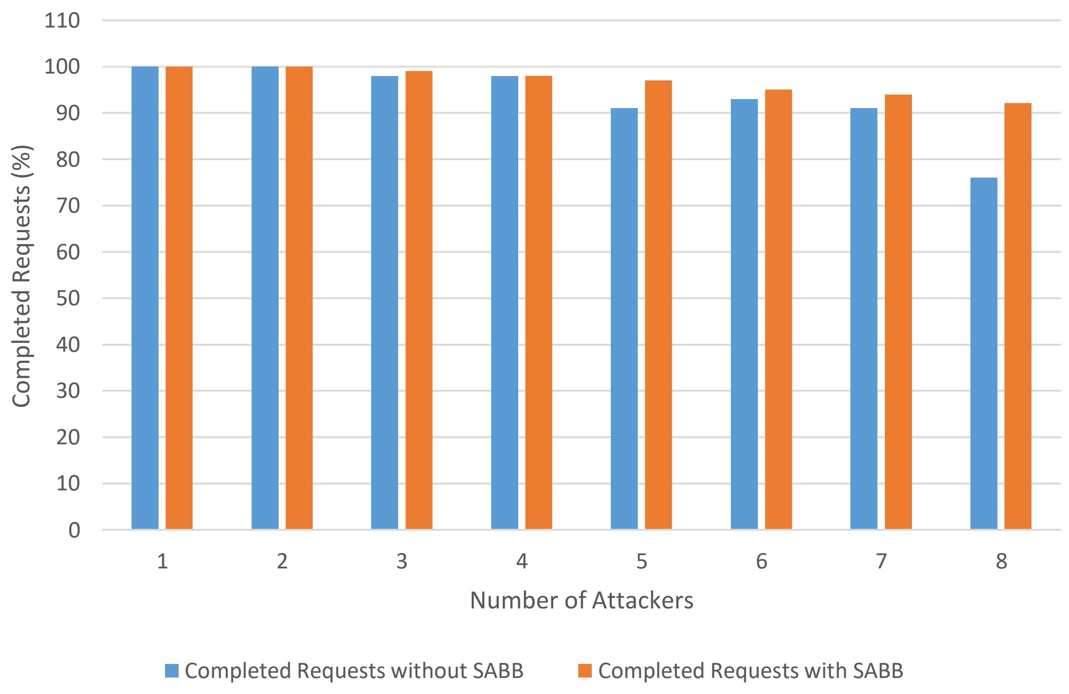


Figure 4.5: Comparison between Completed Requests of without and with SABB

In Table 4.6, when one attacker launched the slow HTTP DDoS attack on the webserver, the average response time when SABB was inactive was eight times higher than the average response time recorded when SABB was inactive. However, when two attackers launched the slow HTTP DDoS attack, the average response time recorded when SABB was inactive doubled the average response time recorded when SABB was active. As much as this highlights the differences between the 100% completed request rate recorded when either one or two attackers were launched when SABB was either active or inactive, it also points to the rate at which a webserver's resource can be consumed with a unit increase in the number of slow HTTP DDoS attackers.

4.5 Comparison with Other Published Work

Results and methods employed by other authors to detect and mitigate slow HTTP attacks were compared in Table 4.9. The analysis of existing work on detecting and mitigating slow HTTP attacks shows that this study, to the best of our knowledge, might be the first application of Support Vector Machine with Genetic algorithm on slow HTTP DDoS which consists of slow get, slow post, and slow read HTTP DDoS attacks. The studies examined either explored slow HTTP DoS or a variation of slow HTTP DDoS which consists of slow get and slow post HTTP DDoS attacks only. Also, the slow HTTP DDoS mitigation technique of SABB used within this work has not been utilised in any of the studies analysed. Only two studies, Calvert and Khoshgoftaar (2019) and Kemp *et al*. (2018) utilised Netflow records as the dataset for analysis. This work further lends the scholarly community insight into using Netflow records as a means of SDN dataset collection for analysis.

Table 4.9: Classification Accuracy Comparison with other Published Work

Method	**Dataset**	**Accuracy**	**Reference**
RBF SVM	Netflow records	99.89%	This study
RF	Netflow records	99.90%	Calvert and Khoshgoftaar (2019)
5NN		99.88%	
C4.5		99.87%	
LR		99.34%	
SVM		99.14%	
JRIP		99.29%	
MLP		98.92%	
Naïve Bayes		97.46%	
RF	Netflow records	96.76%	Kemp *et al*. (2018)
C4.5N		96.72%	
5-NN		96.69%	
C4.5D		96.62%	
MLP		95.06%	
JRip		94.71%	
SVM		89.22%	
Naïve Bayes		88.94%	
RF	TCP Logs	99.37%	Shafieian *et al*. (2015)

4.6 Chapter Summary

The chapter presents the result of feature selection using Genetic Algorithm on the Netflow dataset and selection of the appropriate Support Vector Machine parameter in section 4.1 and 4.2. Section 4.3 presents the result of the classification task on the Netflow dataset. The result obtained from using the SABB module to mitigate slow HTTP DDoS attacks is shown in section 4.4 while section 4.5 shows the comparison of the slow HTTP DDoS detection method used in this work with other methods used in previous studies.

CHAPTER FIVE

5.0 SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This chapter concludes this research on mitigating slow HTTP DDoS in SDN. A summary of the research is presented and conclusions are drawn from the results of the work. Furthermore, recommendations were drawn from the results obtained and the research's contribution to knowledge was outlined.

5.2 Summary

This study developed the use of RBF SVM to detect slow HTTP DDoS attacks which consists of slow get, slow post and slow read attacks. Besides, a unique approach to mitigating the slow HTTP DDoS attacks named Selective Adaptive Bubble Burst was also created. The first objective of selecting the relevant features that signify the presence of attack traffic from a Netflow export generated in a simulated SDN environment in GNS3 was achieved using Genetic Algorithm and Support Vector Machine. First, the population was initialised and crossover operations performed on the parent population to yield offsprings. Then, each offspring was tested against the fitness function defined as the accuracy of the Support Vector Machine which was obtained by classifying the instances whose features are defined by the offspring. Consequently, eleven features were selected through this process out of the initial 31 features which consist of 27 Netflow version 5 features and 4 constructed features. Also, the parameters of the radial basis function kernel of the Support Vector Machine were tuned using Genetic Algorithm.

The classification of Netflow flowsets into benign and anomalous, the second objective, was achieved through the extraction of instances in the Netflow flowset using the eleven features selected and performing a standard scale normalization on the extracted flowsets.

Radial Basis Function kernel-based Support Vector Machine was used to classify the extracted instances which resulted in a 99.89% accuracy. The high accuracy achieved was instrumental in the functioning of the SABB module developed because the ability to mitigate attacks effectively is dependent on the effectiveness of attack detection.

The mathematical and algorithmic models of the SABB module were formulated thus achieving the third objective. Subsequently, the SABB models were translated into the python programming language and uploaded to the controller for real-time attack mitigation which achieved the fourth and fifth objectives. The performance of the real-time mitigation of slow HTTP DDoS attack launched by eight attackers was measured using the average response time and the percentage of completed to timed-out requests sent by a legitimate client to the primary webserver. The result indicates the effectiveness of the SABB module in achieving a fast average response time and a high percentage of completed requests relative to when SABB was not utilised.

5.3 Conclusion

In conclusion, the accuracy of the RBF SVM slow HTTP DDoS classifier presented in this work outperforms the accuracy of the classifiers used in the research by Calvert and Khoshgoftaar (2019), Kemp *et al*. (2018), and Shafieian *et al*. (2015). However, the accuracy of the random forest classifier used in the work by Calvert and Khoshgoftaar (2019) exceeds the accuracy obtained in this work by 0.01. Therefore, RBF SVM is highly competitive in detecting slow HTTP DDoS attacks in Netflow records.

For the SABB slow HTTP DDoS mitigation, the ratio of completed to timed-out requests when SABB was activated exceeds the ratio of completed to timed-out requests when SABB was not activated. Also, the average response time when SABB was activated remained in the 10^2 milliseconds range while when SABB was not activated, the response time reached the 10^3 milliseconds range. Therefore, SABB mitigates slow HTTP DDoS attacks effectively by ensuring the availability of services evident in the high number of completed requests and low response time.

5.4 Recommendations

This study utilised Genetic Algorithm with Radial Basis Function kernel-based Support Vector Machine to detect slow HTTP DDoS attacks which consist of slow get, slow post and slow read and also mitigate such attacks using a technique called Selective Adaptive Bubble Burst. It is recommended that further research should explore the use of multiple controllers for managing the flow of packets in the network to further reduce the latency and increase the percentage of completed requests recorded when the SABB module is activated.

5.5 Contribution to Knowledge

This study has achieved a two-fold contribution to knowledge. First, the study established an effective two-staged approach to detecting slow HTTP DDoS attacks. Second, the study developed a new method of mitigating slow HTTP DDoS attacks which can be applied to various DDoS and DoS attack scenarios.

REFERENCES

Agarwal, S. (2014). Data mining: Data mining concepts and techniques. *Proceedings - 2013 International Conference on Machine Intelligence Research and Advancement, ICMIRA 2013*, 203–207. https://doi.org/10.1109/ICMIRA.2013.45

Ali, J., Khan, R., Ahmad, N., & Maqsood, I. (2012). Random Forests and Decision Trees. *International Journal of Computer Science Issues*, *9*(5), 272–278. Retrieved from https://www.ijcsi.org/articles/Random-forests-and-decision-trees.php

Ali, S. T., Sivaraman, V., Radford, A., & Jha, S. (2015). A Survey of Securing Networks Using Software Defined Networking. *IEEE Transactions on Reliability, vol. 64, no. 3,* pp. 1086-1097. https://doi.org/10.1109/TR.2015.2421391

Alshamrani, A., Chowdhary, A., Pisharody, S., Lu, D., & Huang, D. (2017). A defense system for defeating DDoS attacks in SDN based networks. *MobiWac 2017 - Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access, Co-Located with MSWiM 2017*, 83–92. https://doi.org/10.1145/3132062.3132074

Ameyed, D., Jaafar, F., & Fattahi, J. (2015). A slow read attack using cloud. *Proceedings of the 2015 7th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2015*, SSS33–SSS38. https://doi.org/10.1109/ECAI.2015.7301202

Anusha, K., & Sathiyamoorthy, E. (2016). Comparative study for feature selection algorithms in intrusion detection system. *Automatic Control and Computer Sciences, 50*(1), 1–9. https://doi.org/10.3103/S0146411616010028

Aziz, A. S. A., Azar, A. T., Salama, M. A., Hassanien, A. E., & Hanafy, S. E. O. (2013). Genetic algorithm with different feature selection techniques for anomaly detectors generation. *2013 Federated Conference on Computer Science and Information Systems, FedCSIS 2013*, 769–774. Retrieved from https://annals-csis.org/Volume_1/pliks/106.pdf

Barati, M., Abdullah, A., Udzir, N. I., Mahmod, R., & Mustapha, N. (2014). Distributed Denial of Service Detection Using Hybrid Machine Learning Technique. 2*014 International Symposium on Biometrics and Security Technologies (ISBAST),* Kuala Lumpur, Malaysia, pp. 268–273. https://doi.org/ 10.1109/ISBAST.2014.7013133

Beigi-Mohammadi, N., Barna, C., Shtern, M., Khazaei, H., & Litoiu, M. (2017). CAAMP: Completely automated DDoS attack mitigation platform in hybrid clouds. *International Workshop on Green ICT and Smart Networking, GISN 2016*, 136–143. https://doi.org/10.1109/CNSM.2016.7818409

Benzekki, K., El Fergougui, A., & Elbelrhiti Elalaoui, A. (2016). Software-defined networking (SDN): a survey. *Security and Communication Networks*, *9*(18), 5803–5833. https://doi.org/10.1002/sec.1737

Bhunia, S. S., & Gurusamy, M. (2017). Dynamic attack mitigation using SDN. *2017 27th International Telecommunication Networks and Applications Conference, ITNAC 2017*, 1–6. https://doi.org/10.1109/ATNAC.2017.8215430

Calvert, C. L., & Khoshgoftaar, T. M. (2019). Impact of class distribution on the detection of slow HTTP DoS attacks using Big Data. *Journal of Big Data*. https://doi.org/10.1186/s40537-019-0230-3

Cambiaso, E., Papaleo, G., & Aiello, M. (2017). Slowcomm: Design, development and performance evaluation of a new slow DoS attack. *Journal of Information Security and Applications*, *35*, 23–31. https://doi.org/10.1016/j.jisa.2017.05.005

Cambiaso, E., Papaleo, G., Chiola, G., & Aiello, M. (2013). Slow DoS attacks: definition and categorisation. *International Journal of Trust Management in Computing and Communications*, *1*(3/4), 300. https://doi.org/10.1504/ijtmcc.2013.056440

Crepinsek, M., Liu, S. H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys*, *Vol. 45*, pp. 1–33. https://doi.org/10.1145/2480741.2480752

Cusack, B., & Tian, Z. (2016). Detecting and tracing slow attacks on mobile phone user service. *Proceedings of the 14th Australian Digital Forensics Conference, ADF 2016*, 4–10. https://doi.org/10.4225/75/58a54b013185a

Dabbagh, M., Hamdaoui, B., Guizani, M., & Rayes, A. (2015). Software-defined Networking Security: Pros and Cons. *IEEE Communications Magazine, vol. 53, no. 6*, pp. 73-79. https://doi.org/10.1109/MCOM.2015.7120048

Dantas, Y. G., Fonseca, I. E., & Nigam, V. (2017). Slow TCAM Exhaustion DDoS Attack. *IFIP International Conference on ICT Systems Security and Privacy Protection, vol 502*. pp. 17-31. https://doi.org/10.1007/978-3-319-58469-0_2

Dhanapal, A., & Nithyanandam, P. (2019). The slow http distributed denial of service attack detection in cloud. *Scalable Computing*, *20*(2), 285–298. https://doi.org/10.12694/scpe.v20i2.1501

Donges, N. (2018). The Logistic Regression Algorithm. Retrieved April 21, 2020, from https://machinelearning-blog.com/2018/04/23/logistic-regression-101/

Farina, P., Cambiaso, E., Papaleo, G., & Aiello, M. (2015). Understanding DDoS Attacks From Mobile Devices. *3rd International Conference on Future Internet of Things and Cloud*, Rome, Italy, 2015, pp. 614-619, https://doi.org/10.1109/FiCloud.2015.19

Fonseca, I. E., & Nigam, V. (2016). *Mitigating High-Rate Application Layer DDoS Attacks in Software Defined Networks*. Retrieved from http://nigam.info/docs/sdn-flooding.pdf

Hamad, D. J., Yalda, K. G., & Okumus, I. T. (2016). Getting traffic statistics from network devices in an SDN environment using OpenFlow. *Information Technology and Systems 2015*, (April), 951–956. Retrieved from https://itas2015.iitp.ru/pdf /1570195931.pdf

Hong, K., Kim, Y., Choi, H., & Park, J. (2018). SDN-Assisted Slow HTTP DDoS Attack Defense Method. *IEEE Communications Letters*, *22*(4), 688–691. https://doi.org/10.1109/LCOMM.2017.2766636

Idhammad, M., Afdel, K., & Belouch, M. (2018). Detection System of HTTP DDoS Attacks in a Cloud Environment Based on Information Theoretic Entropy and Random Forest. *Security and Communication Networks*, *2018*. https://doi.org /10.1155/2018/1263123

Jaafar, G. A., Abdullah, S. M., & Ismail, S. (2019). Review of Recent Detection Methods for HTTP DDoS Attack. *Journal of Computer Networks and Communications*, Vol. 2019. https://doi.org/10.1155/2019/1283472

Jevtic, S., Lotfalizadeh, H., & Kim, D. S. (2018). Toward network-based DDoS detection in software-defined networks. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3164541.3164562

Kamarudin, M. H., Maple, C., & Watson, T. (2019). Hybrid feature selection technique for intrusion detection system. *International Journal of High Performance Computing and Networking*, *13*(2), 232–240. https://doi.org/10.1504/IJHPCN.2019 .097503

Kannan, V. (2018). *Feature Selection using Genetic Algorithms* (Masters Thesis, San Jose State University, California, United States). Retrieved from https://scholar works.sjsu.edu/etd_projects/618

Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications* Vol. 80. https://doi.org/10.1007/s11042-020-10139-6

Kaviani, P., & Dhotre, S. (2018). Short Survey on Naive Bayes Algorithm. *International Journal of Advance Research in Computer Science and Management*, 4(11). Retrieved from https://www.researchgate.net/publication/323946641_Short_Survey _on_Naive_Ba yes_Algorithm

Kemp, C., Calvert, C., & Khoshgoftaar, T. M. (2018). Utilizing netflow data to detect slow read attacks. *Proceedings - 2018 IEEE 19th International Conference on*

Information Reuse and Integration for Data Science, *IRI 2018*, 108–116. https://doi.org/10.1109/IRI.2018.00023

Kokila, R. T., Thamarai Selvi, S., & Govindarajan, K. (2015). DDoS detection and analysis in SDN-based environment using support vector machine classifier. *6th International Conference on Advanced Computing, ICoAC 2014*, 205–210. https://doi.org/10.1109/ICoAC.2014.7229711

Kumar, R. (2016). Detecting Denial of Service Attacks in the Cloud. *2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, Auckland, New Zealand, 2016, pp. 309-316, https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2016 .70

Lakshmi, B. N. (2015). An Empherical Study on Decision Tree Classification Algorithms. *International Journal of Science, Engineering and Technology Research*, *4*(11), 3705–3709. Retrieved from http://ijsetr.org/wp-content/uploads /2015/11/IJSETR-VOL-4-ISSUE-11-3705-3709.pdf

Latah, M., & Toker, L. (2019). Artificial intelligence enabled software-defined networking: A comprehensive overview. *IET Networks*, Vol. 8, pp. 79–99. https://doi.org/10.1049/iet-net.2018.5082

Li, X., Yuan, D., Hu, H., Ran, J., & Li, S. (2015). *DDoS Detection in SDN Switches using Support Vector Machine Classifier*. *Proceedings of the 2015 Joint International, Mechanical, Electronic and Information Technology Conference* (JIMET-15), 344–348. https://dx.doi.org/10.2991/jimet-15.2015.63

Li, Y., Guo, X., Pang, X., Peng, B., Li, X., & Zhang, P. (2020). Performance Analysis of Floodlight and Ryu SDN Controllers under Mininet Simulator. *2020 IEEE/CIC International Conference on Communications in China, ICCC Workshops 2020*, 85–90. https://doi.org/10.1109/ICCCWorkshops49972.2020.9209935

Liu, S., Wang, L., Qin, J., Guo, Y., & Zuo, H. (2018). An intrusion detection model based on IPSO-SVM algorithm in wireless sensor network. *Journal of Internet Technology*, *19*(7), 2125–2134. https://doi.org/10.3966/160792642018121907015

Lukaseder, T., Maile, L., Erb, B., & Kargl, F. (2018). SDN-assisted network-based mitigation of slow DDoS attacks. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST*, *255*, 102–121. https://doi.org/10.1007/978-3-030-01704-0_6

Luo, J., Yang, X., Member, S., Wang, J., & Xu, J. (2014). On a Mathematical Model for Low-Rate Shrew DDoS. *IEEE Transactions on Information Forensics and Security*, *9*(7), 1069–1083. https://doi.org/10.1109/TIFS.2014.2321034

Ma, Y., & Guo, G. (2014). Support vector machines applications. *Support Vector Machines Applications* (Vol. 9783319023). https://doi.org/10.1007/978-3-319-02300-7

Mirjalili, S., Song Dong, J., Sadiq, A. S., & Faris, H. (2020). Genetic algorithm: Theory, literature review, and application in image reconstruction. In *Studies in Computational Intelligence* (Vol. 811). https://doi.org/10.1007/978-3-030-12127-3_5

Muraleedharan, N., & Barnabas, J. (2018). Behaviour analysis of HTTP based slow denial of service attack. *Proceedings of the 2017 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2017, 2018-Janua*, 1851–1856. https://doi.org/10.1109/WiSPNET.2017.8300082

Najafabadi, M. M., Khoshgoftaar, T. M., Napolitano, A., & Wheelus, C. (2016). RUDY attack: Detection at the network level and its important features. *Proceedings of the 29th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2016*, 282–287.

Park, J. (2015). Analysis of Slow Read DoS Attack and Countermeasures on Web servers. *International Journal of Cyber-Security and Digital Forensics, 4*(2), 339–353. https://doi.org/10.17781/p001550

Polanco, O., & Guerrero, F. G. (2020). Virtualised Environment for Learning SDN-based Networking. *IETE Journal of Education*, *61*(2), 90–100. https://doi.org/10.1080 /09747338.2020.1838337

Rahman, O., Quraishi, M. A. G., & Lung, C. H. (2019). DDoS attacks detection and mitigation in SDN using machine learning. *Proceedings - 2019 IEEE World Congress on Services, SERVICES 2019, 2642-939X*, 184–189. https://doi.org/10 .1109/SERVICES.2019.00051

Ramchoun, H., Amine, M., Idrissi, J., Ghanou, Y., & Ettaouil, M. (2016). Multilayer Perceptron : Architecture Optimization and Training. *International Journal of Interactive Multimedia and Artificial Intelligence*. https://doi.org/10.9781 /ijimai.2016.415

Rawat, D. B., & Reddy, S. R. (2017). Software Defined Networking Architecture, Security and Energy Efficiency: A Survey. *IEEE Communications Surveys and Tutorials*, Vol. 19, pp. 325–346. https://doi.org/10.1109/COMST.2016.2618874

Sattar, D., Matrawy, A., & Adeojo, O. (2016). Adaptive Bubble Burst (ABB): Mitigating DDoS attacks in Software-Defined Networks. *2016 17th International Telecommunications Network Strategy and Planning Symposium, Networks 2016 - Conference Proceedings*, 50–55. https://doi.org/10.1109/NETWKS.2016.7751152

Schehlmann, L., & Baier, H. (2013). COFFEE : a Concept based on OpenFlow to Filter and Erase Events of botnet activity at high-speed nodes. *Horbach, M. (Hrsg.), INFORMATIK 2013 – Informatik angepasst an Mensch, Organisation und Umwelt. Bonn: Gesellschaft für Informatik,* 2225–2239. Retrieved from https://dl.gi.de/bitstream/handle /20.500.12116/20651/2225.pdf

Sen, S., Gupta, K. D., & Ahsan, M. (2020). Leveraging Machine Learning Approach to Setup Software-Defined Network (SDN) Controller Rules During DDoS Attack. *Proceedings of International Joint Conference on Computational Intelligence,* pp.49-60 https://doi.org/10.1007/978-981-13-7564-4_5

Shafieian, S., Zulkernine, M., & Haque, A. (2015). CloudZombie: Launching and detecting slow-read distributed denial of service attacks from the Cloud. *Proceedings - 15th IEEE International Conference on Computer and Information Technology, CIT 2015*, 1733–1740. https://doi.org/10.1109/CIT/IUCC/DASC /PICOM.2015.261

Shtern, M., Sandel, R., Litoiu, M., Bachalo, C., & Theodorou, V. (2014). Towards mitigation of low and slow application DDoS attacks. *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014*, (Vm), 604–609. https://doi.org/10.1109/IC2E.2014.38

Singh, B., & Rai, B. C. S. (2019). Analysis of Support Vector Machine-based Intrusion Detection Techniques. *Arabian Journal for Science and Engineering*. https://doi.org/10.1007/s13369-019-03970-z

Singh, K. J., & De, T. (2017). Journal of Information Security and Applications MLP-GA based algorithm to detect application layer DDoS attack. *Journal of Information Security and Applications*, *36*, 145–153. https://doi.org/10.1016/j.jisa.2017.09.004

Singh, K. J., Thongam, K., & De, T. (2016). Entropy-based application layer DDoS attack detection using artificial neural networks. *Entropy*, *18*(10), 350. https://doi.org/10.3390/e18100350

Su, Y., Zhang, W., Tao, W., & Qiao, Z. (2018). A Network Illegal Access Detection Method Based on PSO-SVM Algorithm in Power Monitoring System. *4th International Conference, International Conference on Cloud Computing and Security,* Vol. 2, pp.450-459. https://doi.org/10.1007/978-3-030-00009-7

Suroto, S. (2017). A Review of Defense Against Slow HTTP Attack. *JOIV : International Journal on Informatics Visualization*, *1*(4), 127. https://doi.org/10.30630/joiv.1.4.51

Swami, R., Dave, M., & Ranga, V. (2019a). Defending DDoS against Software Defined Networks using Entropy. *Proceedings - 2019 4th International Conference on Internet of Things: Smart Innovation and Usages, IoT-SIU 2019*, 1–5. https://doi.org/10.1109/IoT-SIU.2019.8777688

Swami, R., Dave, M., & Ranga, V. (2019b). Software-defined Networking-based DDoS Defense Mechanisms. *ACM Computing Survey*, *52*(2), 36. https://doi.org/10.1016/B978-0-12-375000-6.00124-5

Tayama, S., & Tanaka, H. (2018). Analysis of Slow Read DoS Attack. *International Conference on Mobile and Wireless Technology,* pp. 350-359. https://doi.org/10.1007/978-981-10-5281-1

Tripathi, N., & Hubballi, N. (2018). Slow rate denial of service attacks against HTTP/2 and detection. *Computers and Security*, *72,* 255–272. https://doi.org/10.1016 /j.cose.2017.09.009

Tripathi, N., Hubballi, N., & Singh, Y. (2016). How Secure are Web Servers? An empirical study of Slow HTTP DoS attacks and detection. *Proceedings - 2016 11th International Conference on Availability, Reliability and Security, ARES 2016*, 454–463. https://doi.org/10.1109/ARES.2016.20

Wang, J., Li, T., Ren, R. (2010). A Real Time IDSs Based on Artificial Bee Colony-Support Vector Machine Algorithm. *Third International Workshop on Advanced Computational Intelligence*, Suzhou, China, 2010, pp. 91-96. https://doi.org/10.1109/IWACI.2010 .5585107

Wang, S.-Y., Chih-Liang, C., & Chun-Ming, Y. (2013). EstiNet Network Simulator and Emulator. *IEEE Communications Magazine*, *51*(9), 110–117. Retrieved from http://www.estinet.com/products.php?lv1=1&sn=2

Wani, A. R., Rana, Q. P., Saxena, U., & Pandey, N. (2019). Analysis and Detection of DDoS Attacks on Cloud Computing Environment using Machine Learning Techniques. *Proceedings - 2019 Amity International Conference on Artificial Intelligence, AICAI 2019*, 870–875. https://doi.org/10.1109/AICAI.2019.8701238

Xingzhu, W. (2015). ACO and SVM Selection Feature Weighting of Network Intrusion Detection Method. *International Journal of Security and its Applications*, *9*(4), 259-270. https://doi.org/10.14257/ijsia.2015.9.4.24

Xu, X., Yu, H., & Yang, K. (2017). DDoS Attack in Software Defined Networks: A Survey. *Journal of Hunan University of Technology*, *15*(3). https://doi.org /10.3969/j.issn.1673

Ye, J., Cheng, X., Zhu, J., Feng, L., & Song, L. (2018). A DDoS Attack Detection Method Based on SVM in Software Defined Network. *Security and Communication Networks*, *2018*. https://doi.org/10.1155/2018/9804061

Ye, Z., Sun, Y., Sun, S., Zhan, S., Yu, H., & Yao, Q. (2019). Research on Network Intrusion Detection Based on Support Vector Machine Optimized with Grasshopper Optimization Algorithm. *2019 10th IEEE International Conference on Intelligent*

Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), *1*(41301371), 378–383. https://doi.org/10.1109/IDAACS.2019.8924234

Yeasir, M., Morshed, M., & Fakrul, M. (2015). A Practical Approach and Mitigation Techniques on Application Layer DDoS Attack in Web Server. *International Journal of Computer Applications, 131*(1), 13–20. https://doi.org/10.5120 /ijca2015907209

Yevsieieva, O., & Helalat, S. M. (2017). Analysis of the Impact of the Slow HTTP DoS and DDoS Attacks on the Cloud Environment. *4th International Scientific-Practical Conference Problems of Infocommunications. Science and Technology*, pp. 519-523. https://doi.org/10.1109/INFOCOMMST.2017.8246453

Yuan, B., Zou, D., Jin, H., Yu, S., & Yang, L. T. (2017). HostWatcher: Protecting hosts in cloud data centers through software-defined networking. *Future Generation Computer Systems*. https://doi.org/10.1016/j.future.2017.04.023

APPENDIX

Selective Adaptive Bubble Burst Implementation in Python

```
import eventlet
import math
import pickle
import socket
from Ryu.base import app_manager
from Ryu.controller import ofp_event
from Ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from Ryu.controller.handler import set_ev_cls
from Ryu.ofproto import ofproto_v1_3
from Ryu.lib.packet import packet
from Ryu.lib.packet import ipv4
from Ryu.lib.packet import ethernet
from Ryu.lib.packet import ether_types
from Ryu.lib.xflow import netflow
from Ryu.lib.ip import ipv4_to_str
from Ryu.lib.mac import haddr_to_str
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from collections import Counter

class NetFlowSwitch(app_manager.RyuApp):

    NETFLOW_UDP_PORT = 2055

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        match = parser.OFPMatch()
        actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                          ofproto.OFPCML_NO_BUFFER)]
        self.add_flow(datapath, 0, match, actions)

    def add_flow(self, datapath, priority, match, actions, buffer_id=None):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                             actions)]
        if buffer_id:
            mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                    priority=priority, match=match,
                                    instructions=inst)
        else:
```

```
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)

#store values in a global dictionary to use for modifying flow instructions
def storeValues(self,ip_src, eth_src,in_port,dst,datapath,buffer_id=None):
    #not a web server source packet
    if (ip_src not in self.websvr_ip):
        if (ip_src not in self.recordedAddr):
            self.flowDetails[ip_src]={}
            self.flowDetails[ip_src]['eth_src']=eth_src
            self.flowDetails[ip_src]['in_port']=in_port
            self.flowDetails[ip_src]['dst']=dst
            self.flowDetails[ip_src]['datapath']=datapath
            self.flowDetails[ip_src]['buffer_id']=buffer_id
            self.recordedAddr.append(ip_src)
        elif (self.flowDetails[ip_src]['buffer_id']!=buffer_id):
            self.flowDetails[ip_src]['buffer_id']=buffer_id
    return

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    self.parser=parser
    self.ofproto=ofproto
    self.datapath=datapath

    pkt = packet.Packet(msg.data)
    ip_src = pkt.get_protocol(ipv4.ipv4)
    if (ip_src):
        ip_src=ip_src.src
    else:
        ip_src=None
    eth = pkt.get_protocols(ethernet.ethernet)[0]

    if eth.ethertype == ether_types.ETH_TYPE_LLDP:
        # ignore lldp packet
        return
    dst = eth.dst
    src = eth.src
```

```
        dpid = datapath.id
        self.mac_to_port.setdefault(dpid, {})
        self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)
        # learn a mac address to avoid FLOOD next time.
        self.mac_to_port[dpid][src] = in_port

        if dst in self.mac_to_port[dpid]:
            out_port = self.mac_to_port[dpid][dst]
        else:
            out_port = ofproto.OFPP_FLOOD

        actions = [parser.OFPActionOutput(out_port)]

        # install a flow to avoid packet_in next time
        if out_port != ofproto.OFPP_FLOOD:
            match = parser.OFPMatch(in_port=in_port, eth_dst=dst, eth_src=src)
            # verify if we have a valid buffer_id, if yes avoid to send both
            # flow_mod & packet_out
            if msg.buffer_id != ofproto.OFP_NO_BUFFER:
                self.add_flow(datapath, 1, match, actions, msg.buffer_id)
                eth_src = src
                self.storeValues(ip_src, eth_src,in_port,dst,datapath,msg.buffer_id)

                return
            else:
                self.add_flow(datapath, 1, match, actions)
                self.storeValues(ip_src, eth_src,in_port,dst,datapath)

        data = None
        if msg.buffer_id == ofproto.OFP_NO_BUFFER:
            data = msg.data
        out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                                  in_port=in_port, actions=actions, data=data)
        datapath.send_msg(out)

    def getFlowDataset(self, nfObj):
        net_ip=""
        flowCountFlag = 0
        for nfMainFlowObj in nfObj.flows:
            net_ip = ipv4_to_str(nfMainFlowObj.srcaddr)
            if((net_ip in self.attackIP.keys()) and (self.attackIP[net_ip] > len(self.websvr_ip)-
1)):
                print(net_ip + " BLOCKED!! ALREADY")
                break
            if ((net_ip not in self.websvr_ip) and (net_ip not in self.whitelist)):
                #extract other necessary details
                difference = nfMainFlowObj.last - nfMainFlowObj.first
                diffSeconds = difference/1000 if difference != 0 else 0      #converts to seconds
from milliseconds
```

```
            packetsPerSec = math.floor(nfMainFlowObj.dpkts/diffSeconds) if diffSeconds!=0 else 0
            bytesPerSec = math.floor(nfMainFlowObj.doctets/diffSeconds) if diffSeconds!=0 else 0
            bytesPerPacket = math.floor(nfMainFlowObj.doctets/nfMainFlowObj.dpkts)

            nfMainFlowValues = [nfObj.count, nfMainFlowObj.input, nfMainFlowObj.output, nfMainFlowObj.dpkts, nfMainFlowObj.doctets, nfMainFlowObj.last, difference, nfMainFlowObj.srcport, nfMainFlowObj.tcp_flags, packetsPerSec, bytesPerPacket]

            initData=[]
            if(net_ip not in self.netflow_dataset.keys()):
                #create a new key in the netflow dataset
                initData.append(nfMainFlowValues)
                self.netflow_dataset[net_ip] = {"data": initData, "flowCount": len(nfObj.flows)}
            else:
                #get previous data and flowCount values
                #append to the netflow dataset dictionary
                flowCountFlag = 1
                prevData = self.netflow_dataset[net_ip]["data"]
                prevData.append(nfMainFlowValues)
                self.netflow_dataset[net_ip]["data"] = prevData

        if((net_ip not in self.websvr_ip) and (net_ip not in self.whitelist)):
            prevFlowCount = self.netflow_dataset[net_ip]["flowCount"]
            self.netflow_dataset[net_ip]["flowCount"] = (len(nfObj.flows) + prevFlowCount ) if flowCountFlag else len(nfObj.flows)

        flows = nfObj.flows
        print("Message: ", vars(nfObj))
        for flow in flows:
            print("Flow Content:", vars(flow))

    def normalizeDataset(self, dataset):
        std_data = StandardScaler()
        std_data.fit(dataset)
        return std_data.transform(dataset)

    def runSVM(self, model, normData):
        y_pred=model.predict(normData)
        return y_pred

    def modifyFlow(self, attackCount, net_attackIP):
        #modify flow to point to next server
        #update attack count
        ofp_parser=self.parser
        ofp=self.ofproto
        datapath=self.datapath
        match = ofp_parser.OFPMatch(in_port=2,
                ipv4_src=net_attackIP,
                ipv4_dst=self.websvr_ip[0],
                eth_type=0x0800)
        dst_websvr = self.websvr_ip[attackCount]
        actions = [ofp_parser.OFPActionSetField(ipv4_dst=dst_websvr),
        ofp_parser.OFPActionSetField(eth_dst=self.websvr_mac[dst_websvr]),

ofp_parser.OFPActionOutput(self.websvr_ip_to_port[self.websvr_ip[attackCount]])]
        inst      =      [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
actions)]
        mod = ofp_parser.OFPFlowMod(
            datapath=datapath,
            priority=attackCount+1,
            buffer_id=ofp.OFP_NO_BUFFER,
            match=match,
            instructions=inst)
        datapath.send_msg(mod)
        print("Modified flow from "+str(net_attackIP)+"to dst " + str(dst_websvr))
        match                                                                    =
ofp_parser.OFPMatch(in_port=self.websvr_ip_to_port[self.websvr_ip[attackCount]],
                ipv4_src=self.websvr_ip[attackCount],
                ipv4_dst=net_attackIP,
                eth_type=0x0800)
        actions = [ofp_parser.OFPActionSetField(ipv4_src=self.websvr_ip[0]),

ofp_parser.OFPActionSetField(eth_dst=self.flowDetails[net_attackIP]['eth_src']),
                   ofp_parser.OFPActionOutput(2)] #check here for input port for hosts
        inst      =      [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
actions)]

        mod = ofp_parser.OFPFlowMod(
            datapath=datapath,
            priority=attackCount+1,
            buffer_id=ofp.OFP_NO_BUFFER,
            match=match,
            instructions=inst)

        datapath.send_msg(mod)

    def blockTraffic(self, attackCount, net_attackIP):
        #modify flow to point to next server
        #update attack count
        ofp_parser=self.parser
        ofp=self.ofproto
        datapath=self.datapath

        match = ofp_parser.OFPMatch(in_port=2,
                ipv4_src=net_attackIP,
            ipv4_dst=self.websvr_ip[0],
            eth_type=0x0800)

        actions = [ofp_parser.OFPActionOutput(2)]
        inst       =       [ofp_parser.OFPInstructionActions(ofp.OFPIT_APPLY_ACTIONS,
actions)]

        mod = ofp_parser.OFPFlowMod(
            datapath=datapath,
            priority=attackCount+1,
            buffer_id=ofp.OFP_NO_BUFFER,
            match=match,
            instructions=inst)
        print("Blocked flow from "+str(net_attackIP))
        datapath.send_msg(mod)

    def runServer(self):

        try:
            serverSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            serverSock.bind(("192.168.0.18", 2055))
            model = pickle.load(open("SVM_Model2.sav",'rb'))

            while True:
                data, addr = serverSock.recvfrom(6024)
                netflowObj = netflow.NetFlow.parser(data)

                self.getFlowDataset(netflowObj)

                for net_dataset_ip in self.netflow_dataset:

                    if((net_dataset_ip in self.attackIP.keys()) and (self.attackIP[net_dataset_ip]
> len(self.websvr_ip)-1)):
                        continue
                    if(self.netflow_dataset[net_dataset_ip]["flowCount"] < 100):
                        continue

                    normalizeDataset =
self.normalizeDataset(self.netflow_dataset[net_dataset_ip]["data"])
                    predictionResult = self.runSVM(model,normalizeDataset)
                    #delete dataset present in a particular IP
                    del self.netflow_dataset[net_dataset_ip]["data"][:]
                    #modify flowCount to zero
                    self.netflow_dataset[net_dataset_ip]["flowCount"] = 0
                    print(Counter(predictionResult))

                    #count of prediction result
                    #if count of prediction result for 0 is grater than 30
                    # else, it is an attack traffic
                    #propagate flow modification to openflow switch
                if (Counter(predictionResult)[0] < 30):
                    if (net_dataset_ip in self.attackIP.keys()):
                        attackCount=self.attackIP[net_dataset_ip]
                        if (attackCount >= len(self.websvr_ip)-1):
                            #block traffic using flow modification
                            self.attackIP[net_dataset_ip] += 1
                            eventlet.spawn_n(self.blockTraffic,attackCount, net_dataset_ip)
                        else:
                            #modify flow to point to next server
                            #update attack count
                            self.attackIP[net_dataset_ip] += 1
                            attackCount=self.attackIP[net_dataset_ip]
                            eventlet.spawn_n(self.modifyFlow,attackCount, net_dataset_ip)
                    else:
                        #add the attack IP to attackIP and initialize to 1
                        #modify traffic to point to next server
                        self.attackIP[net_dataset_ip] = 1
                        attackCount=self.attackIP[net_dataset_ip]
                        self.modifyFlow(attackCount, net_dataset_ip)

            print("Terminated")
        except Exception as e:
            print("Keyboard Interrupt recieved")
            print(e)
        finally:
            print("Terminated 2")

    def___init_(self, *args, **kwargs):
        super(NetFlowSwitch, self)._init_(*args, **kwargs)
        self.mac_to_port = {}
        self.flowDetails = {}
        self.netflow_dataset = {}
        self.recordedAddr= []
        self.websvr_ip = ["192.168.0.24", "192.168.0.22", "192.168.0.20"]
        self.whitelist = []
        self.websvr_mac = {"192.168.0.24":"",
                           "192.168.0.22":"",
                           "192.168.0.20":""}
        self.websvr_ip_to_port = {"192.168.0.24":1,
                           "192.168.0.22":11,
                           "192.168.0.20":10}

        self.attackIP={}
        self.netflow_svrip=["24","22","20"]
        self.parser=None
        self.ofproto=None
        self.datapath=None
        eventlet.spawn_n(self.runServer)
```