Predicting DDoS Anomaly Patterns in SDN Controller using Hidden Markov Model

Abdul-wadud Alhasan
Nanjing University of Science and Technology
200 Xiaolingwei, Nanjing
Jiangsu, PR. China

Sonjie Wei
Nanjing University of Science and Technology
200 Xiaolingwei, Nanjing
Jiangsu, PR. China

ABSTRACT
The introduction of Software Defined Networking (SDN) as a panacea to the global demand for a more secure and highly dependable internet infrastructure has also brought along security issues. The adoption of OpenFlow Protocol (OFP) by SDN as the way of communication between controllers and switches, has not only brought about easy and direct manipulation of data for enhanced packet forwarding policies, but also renders the network vulnerable to security issues (DDoS attacks) since the OpenFlow (OF) switch has to ask the controller to install new rules for any new incoming packet.

In this work, the capability of SDN in handling security threats that arise from the above vulnerability is proven. This work seeks to design and implement a DDoS detection model that uses Hidden Markov Model (HMM) for detecting abnormal traffic (OpenFlow flooding attacks) directed towards the SDN controller aimed at destabilizing the flow of normal network traffic among users in a software-defined networking environment.

The experiment achieved an accuracy of 94.3% in classifying network traffic with 5.7% false positive rate. The feasibility of this approach is proven by building a test scenario to simulate the approach with POX controller and OpenFlow switches.

General Terms
Software Defined Networking (SDN), Anomaly Detection, Distributed Denial of Service (DDoS), Network Security

Keywords
OpenFlow, Mininet, OpenFlow (OF), SDN, Hidden Markov Model (HMM)

1. INTRODUCTION
Innovative internet ideas are increasingly the interest of today`s world due to the rise in technological demands. As a result, the need for an equally innovative internet security architecture has become the focus for researchers. Software Defined networking (SDN) is one of the few areas that has proven to be the future of networking since it is dynamic and programmable due to the separation of its control and data plane. This makes application of security approaches more convenient as compared to traditional networks where networking devices are vertically integrated hence, reducing network configuration flexibility, policy enforcement and evolution of the network. SDN enhances network utilization and management, ensuring availability and security as well as promoting evolution whiles maintaining performance with the use of a well-defined application programming interface (API)[1], [2]. OpenFlow is the most notable API in the world of SDN today since it is the first and widely used standardized protocol which defines the way of communication between controllers and switches. OpenFlow adopts the concept of flows to determine traffic flow based on a per-flow basis through network devices[3]. This allows for direct manipulation of data in order to enhance policies for packet forwarding, unlike current traditional networks. In a typical SDN environment, the controller manages a set of flow tables in an OF switch. These flow tables contain flow entries made up of counters and match fields as well as a set of actions to apply to matching packets.

This separation however, is also subject to vulnerability issues with the central control logic prone to failure due to attacks from malicious users. One of such vulnerabilities that has been prominent is Distributed Denial of Service (DDoS) attacks[4]. In SDN, DDoS attacks occur as a result of a malicious user sending numerous numbers of anomalous packets with different header fields which are forwarded by the OpenFlow switch to the controller for further actions since no matching rules are found. Hence, the controller uses much resources that prevents it from answering requests from legitimate users. Also, once an attacker successfully compromises an SDN switch, flow table modification becomes easy for an attacker to change rules to attack the whole network [5]. The above issues as well as several threat vectors identified by [6] warrants the need for constant improvements in the security of SDN. With numerous literature on SDN and its security aspects including but not limited to the works of [7]–[11], there is still the need for improvements in enhancing SDN security.

The rise of machine learning in network anomaly detection has also paved a way for the development and improvement of anomaly detection systems. Therefore, in this paper, an attempt is made to identify anomalous flow patterns by observing the behavior of packet flow in a software defined network and using the characteristics of this behavior to design a system for detection of distributed denial of service (DDoS) attacks aimed at the SDN controller. Hidden Markov model[12] is applied to predict the state of the network. This detection model will help in identifying effective techniques to handle these attacks.

Specifically, this study:

1. Seeks to show the effects DDOS attacks have on SDN by employing a few threat vectors identified by [6] i.e., forged traffic flows and vulnerabilities in controller communication.

2. Uses a Software Defined Network (SDN) to achieve complete control of network traffic for intrusion detection and utilize OpenFlow's message mechanism to increase the flexibility of the detection period.

3. Proposes an anomaly detection model using hidden Markov model (HMM) to define a variety of states with double stochastic processes of hidden states and observed states to improve the true positives and reduce the false positives.

2. SYSTEM DESIGN AND MECHANISM

In this chapter, an in-depth description of the system design and mechanism process is given. As depicted in figure 1, the detection scheme consists of three phases which will be discussed in detail in the preceding chapters: Data collection, Data preprocessing/Feature Selection, and Data modelling with HMM.

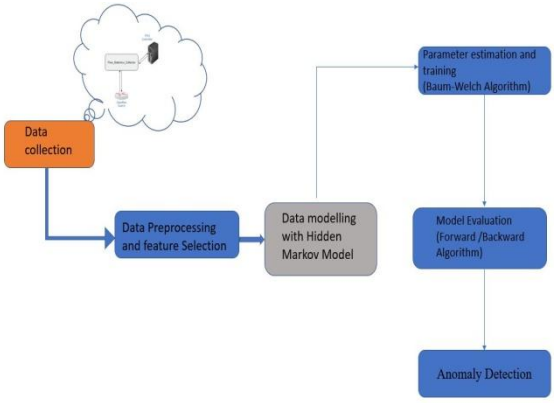

Figure 1 System Design Process

2.1 Data Collection

Data is obtained by deploying an SDN network where the controller is connected to the switches that serve as points of data collection.

The proposed system uses a Flow_Statistics_collector that records the flow information in an active flow table in the controller which consists all current active flows in the network. Table 1 shows the structure of the Flow_Statistics_collector. For better performance, the statistics collector is only responsible for gathering data and merging port and flow statistics into a single list and stored in a CSV file.

Data statistics are gathered from controlled switches that are authenticated by the controller (POX) by generating IDs for each switch. This allows for exchange of packets between hosts connected to each switch. Since the headers of each packet arriving at the switch is matched against flow entries of the switches flow table, statistics of flow entries are updated in the case of a successful match. However, if no flow entry (in the switch's Flow Table) matches the packet's header, our application proactively uses OpenFlow to install a Flow Entry that instructs the controlled switches to forward all traffic to the controller. In its turn, the controller may add, according to the defined policy, a new flow entry to the Flow Table of every switch as required for policy enforcement by using FlowMod control message. Thus, the traffic generated by all hosts connected to a given OF switch will populate the Flow Table of said switch.

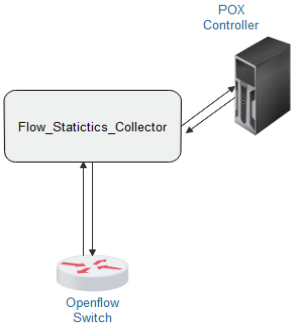

Figure 2 Data Collection System

Table 1 How Flow information is stored in Flow_Statistics_Collector

Flow	Counter
flow[1]	i. Packet count ii. Total bytes iii. Duration

Two main fields are contained in Table 1: flow and counter with the counter further divided into Packet_Count, Total_Bytes and Duration field. When the controller performs the actions explained earlier after the arrival of an incoming packet, the new information is added to the list of active flows(flow[1]) by our Flow_Statistics_collector in the flow field. The controller then installs the associated rules as a flow entry in the flow table of the switch via FlowMod message. Figure 2 depicts the nature of our data collection module.
The collection of flow entries from an OpenFlow switch is performed at predetermined time intervals by the controller. From this collection important features are extracted to classify traffic as normal or as an attack. As the collector gathers samples from all OF switches authenticated by POX, the switch ID is used to help the classifier module pinpoint in which OF switches DDoS flooding attacks were detected.

2.2 Data preprocessing

Preprocessing converts network traffic into a series of observations, where each observation is represented as a feature vector. Observations are optionally labeled with its class, such as "normal" or "anomalous". These feature vectors are then suitable as input to HMM algorithm. The algorithm is able to generalize from these labelled data(observations), hence allowing future observations to be automatically classified.
Collected data usually contain redundant, noisy and different scales of feature values which presents challenges that are critical to data modelling. In order to fix these challenges, data are treated to scale by calculating the standard scores for each feature. The standard score x^i, of a data feature x is given by:

$$x^i = \frac{x-\mu}{\sigma(x)} \qquad 2\text{-}1$$

Where $\sigma(x)$ represents the standard deviation and μ the distribution mean value for x. the issues of high variability and scaling effect is removed and reduced in standardized features since they have approximately zero mean and unit standard deviation.

2.3 Feature Selection

Several features of dataset have been identified by several authors for developing an intrusion detection and evaluation system. However, the most effective and significant features need to be extracted in order to build a suitable detection

algorithm with high detection accuracy. Reducing the feature set to a smaller set of features and selecting the significant features will improve the computation time, obtain higher accurate detection rates and minimize the effect of noise when training occurs. Since our work is focusing on OpenFlow flooding attacks, we consider the most obvious features with reference to the works of[7], [10].

In this method, the feature set includes 6 different features whose data are extracted and stored in a CSV file format. These features include:

1. Average number of packets per flow (APf): In order to increase an attack's efficiency during an OpenFlow flooding attack, source IP spoofing is usually employed by generating a limited number of flows with small number of packets, this creates a difficulty in tracing the attack source. Considering that normal traffic usually contains a much larger number of packets, we calculate the median value. Based on the number of packets per flow, flows are ordered in ascending order prior to calculating the median value. This computation is done using equation 3-2, where *K* is the number of packets per flow and *n* is the sequence of flows, thus, k(1), k(2),…,k(n) represents the sequence of packets.

$$\boldsymbol{md(K)} = \begin{cases} \boldsymbol{K(\frac{n+1}{2}, \quad if\ n\ is\ odd,} \\ \boldsymbol{\frac{K(n/2)+K((n+1)/2,}{2} \quad otherwise} \end{cases} \qquad 2\text{-}2$$

2. Average number of bytes per flow (ABf): This can also be computed as a median value using equation 3-2 with K now representing the number of bytes. This number of bytes can be obtained from the flow stats in the OpenFlow switch.

3. Growth rate of different Ports (GDP): Just like IP spoofing, ports can also be generated randomly during attacks. Thus, the growth rate of different ports can be computed using equation 3-3.

$$\boldsymbol{GDP} = \frac{\boldsymbol{Num\_ports}}{\boldsymbol{interval}} \qquad 2\text{-}3$$

4. Growth rate of flows (Gof): when a flood attack starts, there can be a quick rise in the number of flows. This can be seen through programming the OpenFlow switch to record the flow of every packet in the flow table. To compute this growth, we use equation 3-4.

$$\boldsymbol{Gof} = \frac{\boldsymbol{Difference\ in\ flow\ number}}{\boldsymbol{time\ interval}} \qquad 2\text{-}4$$

5. Percentage of Pair-flows (PPf): In order for flows (flow^1 and flow^2) to be considered pair-flows, they should meet the following conditions, flows that do not meet the following conditions are called single-flows:
   - SrcIP(flow^1) = DstIP(flow^2)
   - DstIP(flow^1) = SrcIP(flow^2)
   - Both flows must have the same communication protocol.

The attacker's use of fake IPs results in an increase in single-flows hence the decrease in percentage of pair-flows can be an indication of attack in the SDN environment. This feature helps determine the number of pair-flows occurring in a flow stream over a certain interval**.** We use equation 3-5 to compute the percentage of pair-flows.

$$\boldsymbol{PPF} = \frac{\boldsymbol{2*Num\_Pair-flows}}{\boldsymbol{Num\_flows}} \qquad 2\text{-}5$$

6. Average of Duration per flow (ADf): The duration of time a flow spends in a flow table decreases the number of false positives during packet exchange (small number) between applications. In order to measure the duration of time a flow spends in a flow table; we also employ the use of median value. Equation 3-2 is used for this computation with *X* now representing the duration of flow in the switch and *n* the number of flows.

2.4 Data modelling with HMM

In this section, we explain how we employed the Hidden Markov Model in our data modelling.

Traditional HMM specifies five entities in the model property: the set of states' S, the observation set V, the initial probability state π, transition probabilities A and the observation probability *B*. For the purpose of this project, we define 6 entities as follows:

$\Lambda = S, V, \pi, A, B, C$ where:

S represents the number of different states assumed to be in the system. Each individual state is represented by $S_1, \ldots, S_N$. In this work, we use the following 4 classes as states:

Normal(N): indicates that there is no case of malicious activity in the system.

TCP flood(T): Indicated that there is a TCP flooding attack occurring in the system.

ICMP flood(I): indicates that there is an occurrence of ICMP flooding attack.

UDP flood(U): indicates the occurrence of UDP attacks.

Each state is represented by the following characters hence: $S_i = \{S_1 = N, S_2 = T, S_3 = I, S_4 = U\}$.

The security state of a network changes over time, and the sequence of state from time 1 to T is denoted by $x = x_i, \ldots, x_T$, where $x_t \in S$. The Markov Model used for our HMM is a first order model. That is, the state transition probabilities only depend on the previous state. Figure 3 shows the topology of the transition states of our HMM model from one state to another. S_1, S_2, S_3 and S_4 represent the states of our HMM model. The connecting link in the figure indicates the state transition probability for each state.

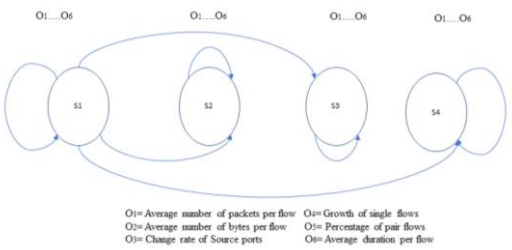

Figure 3 HMM model topology of transition states

V denotes the set of all possible feature observations, generally expressed as $V = \{V_1, \ldots, V_M\}$. The observation feature set in our work includes 6 different features as

explained in section 3.3; Average number of packets per flow (APf): Average number of bytes per flow (ABf): Change rate of source ports (Gof): Growth of Single-flows (GSf): Percentage of Pair-flows (PPf): Average of Duration per flow (ADf). *M* is the number of distinct feature observations per state which corresponds to the physical output of the modeled state. Thus, V_M = 12 since each feature has two values which can be normal or abnormal. For better illustration of the various attack features

$V = \{T_i(1),\ T_i(2),\ U_i(1), U_i(2),\ I_i(1), I_i(2)\}$. $V_i(1)$ indicates a normal status whiles $V_i(2)$ indicates an abnormal status. The observation value sequence is $O = \{O_1, O_2, \ldots, O_T\}$, where $O_i = (i = 1,2 \ldots T)$ represents the observed feature set at certain time. Each observed feature set consists of 6 feature values, $O_i = \{O^1, O^2, O^3, \ldots, O^6\}$. $O^j \in V$. *A* is a matrix which represents the state transition probability matrix usually represented as:

$$\boldsymbol{A = [\,a_{ij}] = P(qt+1 = S_j | qt = S_i),\quad 1 \le i,j \le N} \qquad 2\text{-}6$$

where, *qt* is the state at time $t, S_i \in S$(set of all possible states). The state transition probability describes the probability of transitions between the states of the models where each model a_{ij} represents the probability of the network state changing to the state S_j at time $t+1$, given that it is in the state S_i at time t. Parameter *B* defines the observation symbol probability matrix as;

$$\boldsymbol{B = [\,bi,j]\,,\ b_i(k) = P(Vk\ at\ t | qt = Si),\quad 1 \le i,j \le N} \qquad 2\text{-}7$$

The Observation Probability Matrix B describes the probability of observing a network feature value given that the network is in a certain state. Each entry $b_i(k)$ represents the probability of observing V_p in a network status S_i at a time t. And the initial state probability vector $\pi = p(S_i)$ as defined in the equation below.

$$\boldsymbol{\pi = \pi i},\ \text{where},\ \boldsymbol{\pi = p[Sj], 1 \le j \le N} \qquad 2\text{-}8$$

The decision whether the TCP, ICMP or UDP packet type is normal or an attack is based on the combined features and their values, these feature values represent observations that can be taken as observation sequences and classified with the right observation symbol numbers. Hence to reduce dimensionality of the model and improve performance, we treat all features and their values separately. The actual values of the selected features as well as the discrete observation symbol of the corresponding values in our session are shown in Table 3.

Table 2 Actual Values and Discrete Observation Symbol values of the Features

Features	Value 1	Value 2
Average number of packets per flow	9	4
Average number of bytes per flow	534	248
Change rate of source ports	0	0
Growth of Single-flows	0	0
Percentage of Pair-flows	0	0
Average of Duration per flow	1308	2945
Observation Symbol number	1	2

From Table 3.3, it can be seen that the above-described values have adapted as observation sequence of $O = O_1, O_2, O_3, O_4, O_5$ and O_6which correspond to the observation symbols for each sequence. Thus, after these values are classified with their observation symbol numbers, the observation sequences qualify for classification of an instance with type normal or attack and divided into two types: (1) a known observation sequence extracted from the training dataset and used for training model, and (2) an unknown observation sequence extracted from the test dataset and used for evaluating trained model.

Once the observation sequences of type normal or an attack are generated, the next step is to set up the HMM parameters for training and testing model as explained in the next section.

2.4.1 Parameter Estimation and Training

In this section, the use of HMM's to learn the typical behaviors in the SDN network environment is discussed. In order to train a model, we first need to get the hidden state space π, the observed state A, and the initial probability matrix B.

We use Baum-Welch algorithm to train O(observed value sequence) and S(corresponding state sequence), and for parameter estimation of the values $A, B, and\ \pi$. This algorithm calculates the parameters by initializing a model and updating the model into a better one until the quantity of the model cannot be improved further. The Baum-Welch algorithm contains two steps:

i- E-step (Expected step) where the expected state occupancy count γ and the expected state transition count ξ from the initial transition and emission probabilities are computed.

ii- M-step (maximization step): In this step, the expected state occupancy count γ and the expected state transition count ξ are used to recompute new A and B probabilities.

Given π, the expected number of times that you are in a state Si, at a time t, the probability that you transition from state i, to state j, and b_j, V_k, the steps of Baum-Welch algorithm applied are shown in Algorithm 1 where i, j represent different status in status set S, and $\xi_t(i,j)$ is calculated by equation 3-9.

$$\boldsymbol{\xi_t(i,j) = \frac{a_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=0}^{6}\sum_{j=1}^{6} a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}} \qquad 2\text{-}9$$

Algorithm 1 Baum-welch Algorithm

Input: Observed value sequence $O = \{O_1, O_2, ..., O_T\}$

Output: the parameters of the Hidden Markov Model:
$\lambda = \pi, A, B;$

Step 1: Initialization
for n = 0, select $\pi_i^{(0)}$, $a_{ij}^{(0)}$, $b_j^{(0)}$ to obtain the initial model $\lambda^{(0)} = (\pi^{(0)}, A^{(0)}, B^{(0)})$

Step 2: Iterative calculation
for n = 1,2, …,

Update a_{ij}: $a_{ij}^{(n+1)} = \frac{\sum_{t-1}^{T=1} \xi_t(i,j);}{\sum_{t=1}^{T=1} y_t(i,j);}$

Update b_j: $b_j(k)^{(n+1)} = \frac{\sum_{t=1, o_t=v_k}^{T} y_t(j)}{\sum_{t=1,}^{T} y_t(j)};$

Set π_i: $\pi_i^{(n+1)} = y_t(j);$

***Step 3*:** Termination. Obtain the parameters value where

$\lambda = \pi, A, B = \pi^{(n+1)}, A^{(n+1)}, B^{(n+1)}$.

Two types of models are considered in this approach: Model for normal HMM λ_1 and model for Attack HMM λ_2. Each model is assigned to each distinguishable state with initial values of π, A and B carried out to be uniformly distributed so that the local maximum becomes the global maximum. The initial values of π, A, and B for each session in our model are initialized to be the same for both HMM for "normal" and HMM for an "attack" model. The corresponding initial values of π and A parameter which correspond to the initial state probability distribution and state transition probability distribution respectively are illustrated in Table 4 and 5.

Table 3 Initial State distribution (Parameter 'π' of HMM)

States	Initial State Distribution Value
1	0.000582
2	0.261901
3	0.089831
4	0.375827

Table 4 State transition probability distribution (Parameter 'A' of HMM)

States	1	2	3	4
1	0.1455	0.1061	0.2719	0.2496
2	0.0679	0.3353	0.2774	0.2004
3	0.0191	0.1165	0.4647	0.1878
4	0.6308	0.2843	0.0760	.0030

The Baum-Welch algorithm is applied to re-estimate the HMM parameters to get a new set of parameters until a point is reached where the sample likelihood is locally maximal. For updating and re-estimating the HMM parameters once they are initialized, we use $\xi_t(i,j)$ and the probability $\gamma t(i)$ generated by the training process of Baum-Welch algorithm to get the optimality criterion which maximizes the expected number of correct individual states. $\gamma_t(i)$ is the probability of status S_i at time *t*, as shown in equation 3-10.

$$\boldsymbol{\gamma_t(i) = \frac{a_t(i)\beta_t(i)}{P(Y|\lambda)} = \frac{a_t(i)\beta_t(i)}{\sum_{j=1}^{N} a_t(j)\beta_t(j)}} \qquad 2\text{-}10$$

The following steps were employed in the re-estimation stage of the training algorithm:

i- Re – estimating initial state distribution values

$$\boldsymbol{\pi_i = \gamma_t 1(i)\ where, 1 \le i \le N} \qquad 2\text{-}11$$

ii- Re – estimating state transition probability distribution

$$\boldsymbol{a_{ij} = \sum_{t-1}^{T-1} \xi_t(i,j) / \sum_{t-1}^{T-1} \gamma_t(i)} \qquad 2\text{-}12$$

iii- Re – estimating observation symbol probability distribution

$$\boldsymbol{b_j(k) = \frac{[(s,t)O_t = V_k] \sum_{t=1}^{N} \gamma_t(j)}{\sum_{t-1}^{N} \gamma_t(j)}} \qquad 2\text{-}13$$

We continue the process of re-estimation until we cannot reach the desired limiting point. At the end of the training phase, we generate one model for the trained values of the HMM parameters A, B and π as depicted in Tables 6, 7, and 8 below.

Table 5 Trained values of Parameter 'A' of HMM

States	1	2	3	4
1	0.2458	0.1249	0.3470	1.5571
2	0.0066	0.01015	0.9816	0.2005
3	0.0191	0.0951	0.5727	0.2117
4	0.7104	0.3854	0.1670	0.1084

Table 6 Trained values of Parameter 'B' of HMM

States	1	2	3	4
1	0.3156	0.2164	0.1714	0.0394
2	0.0679	0.4355	0.2774	0.2005
3	0.0191	0.1165	0.4648	0.1878
4	0.6308	0.2844	0.0760	0.0029

Table 7 Trained Initial State distribution (Parameter 'π' of HMM)

States	Initial State Distribution Value
1	0.15610
2	0.38650
3	0.09963
4	0.48582

2.4.2 Model Evaluation

The issue of calculating the probability of the unknown observation sequence $P = O|\lambda$ where the model λ of type normal or attack and the observation sequences $O = O_1, O_2, O_3, O_4, O_5$ and O_6 of type normal or attack are given already, can be solved using Forward algorithm and Backward algorithm. The process is described in a and b below:

a- Forward procedure: The forward variable $\alpha_t(i) = O_1, O_2, O_3, O_4, O_5, O_6, qt = Si\,|\lambda,$ indicates the probability of the partial observation sequence $O_1, O_2, O_3, O_4, O_5, O_6$ and the state S_i at time t, given the model λ. The stages in the forward procedure is given in the equations below:

i- Forward variable value initialization

$$\boldsymbol{a_t(i) = \pi_i b_i(O_1)} \qquad 2\text{-}14$$

where $1 \le i \le N$

ii- Induction

$$\boldsymbol{a_{t+1}(j) = [\sum_{i-1}^{N} a_t(i) a_{ij}] b_j(O_{t+1})} \qquad 2\text{-}15$$

where $1 \le t \le T-1;\ 1 \le j \le N$

iii- Termination

$$\boldsymbol{P(O|\lambda) = \sum_{i=j}^{N} a_T(i)} \qquad 2\text{-}16$$

Thus, $P\ (O|\lambda)$ is the sum of all the $\alpha_t(i)$ values.

b- Backward procedure: Backward variable $\beta_t(i)$ refers to the probability of the partial observation sequence from $(t+1)$ to the end, given state S_i at time *t* and the model λ. the main steps involved in Backward Procedure are described using the equations below.

i- Initialization

$$\boldsymbol{\beta_t = (i) = 1} \qquad 2\text{-}17$$

ii- Induction step

$$\boldsymbol{B_t(i) = \sum_{j=1}^{N} a_{ij}\, b_j(O_{t+1}) \beta_{t+1}(j)} \qquad 2\text{-}18$$

where $\boldsymbol{t = T-1, T-2, T-3, \ldots 1\ and\ 1 \le i \le N}$

2.4.3 Risk assessment

Each of the states in the system is associated with a risk vector C which indicates the potential consequences of the state in question. After obtaining the probability $\gamma_t(i)$, as well as the risk vector $C = \{C_1, C_2, C_3, C_4\}$, we calculate the total risk value Rt at time t using the following equation;

$$\boldsymbol{Rh, t \sum_{i=1}^{N} \gamma t\,(i) C(i)} \qquad 2\text{-}19$$

where $\gamma_t(i)$ is the probability that the system is in security state S_i at time t, N is the number of security states, and $C(i)$ is the risk value associated with state S_i.

2.4.4 Anomaly Detection

We employed the Viterbi algorithm for our attack detection. This algorithm is used in finding the most likely sequence of hidden states as a result of unobserved sequence events. As a dynamic programming algorithm, it can be used to solve HMM issues thus, making it fit for predicting the state of the network in our work.

In this work, we employ the following process in applying Viterbi algorithm where $\delta t(i)$ is defined as a maximum probability among all the paths ($x_1, x_2, ..., x_t$) in status i at time t, as denoted in the equations below.

$$\delta_t(i) = \max_{x_1,x_2,\dots,x_{t-1}} P(x_t = x, x_t - 1, \dots, x_1, o_t, \dots o_1|\lambda), x = 1,2, \dots, N \quad 2\text{-}20$$

The recursion formula of δ is as Eq.

$$\delta_t(i) = \max_{x_1,x_2,\dots,x_{t-1}} P(x_{t+1} = x, x_t, \dots, x_1, o_{t+1}, \dots o_1|\lambda),$$

$$= \max_{1\le j\le N}[\boldsymbol{\delta_t(j)a_{ij}}]\boldsymbol{b_i(o_{t+1})}, \boldsymbol{x = 1,2,\dots,N;\ t = 1,2,\dots,T-1} \quad 2\text{-}21$$

$\psi_t(x)$ is the $t-1th$ node in δ_t, as shown below;

$$\boldsymbol{\psi_t(x)} = \underset{1\le j\le N}{\boldsymbol{arg\ max}}\ [\boldsymbol{\delta_{t-1}(j)a_{ij}}] \quad 2\text{-}22$$

Viterbi algorithm recursively calculates $\delta t(i)$ and then trace back to obtain the optimal path, using the observed value sequence $O = \{O_1, O_2, \dots, O_T\}$ to predict the status sequence $X = \{x_1, x_2, \dots, x_T\}$, where $x_T \le 1 \le t$, T is the status value at time t corresponding to the observed sequence. The process is summarized in algorithm 2.

Algorithm 2 Viterbi Algorithm

Input: HMM: π, A, B
Observed sequence: $O = \{O_1, O_2, \dots, O_t\}$:
Output: Optimal path: $X^* = \{x_1^*, x_2^*, \dots x_t^*\}$;
1: Begin $\delta t(i) = \pi_i b_i(o_1)$ and $\psi_1(i) = 0, I = 1, 2, \dots N;$
2: **for** t = 2; t ≤ T: t + + do:
3: $\delta t(i) = \max_{1\le j\le N}[\delta_{t-1}(j)a_{ji}]b_i(o_t),\ i = 1,2, \dots, N$
4: $\psi_1(i) = \arg\max_{1\le j\le N}[\delta_{t-1}(j)a_{ji}],\ i = 1,2, \dots, N$
5: **end for** $P^* = \max_{1\le j\le N} \delta_t(i),\ i_T^* = \arg\max_{1\le j\le N}[\delta_T(i)]$
6: **for** $t = T-1: t > 0, t - - do:$
7: $x_t^* = \psi_{t+1}(x_{t+1})$
8: **end for**.

3. IMPLEMENTATION AND EVALUATION

In this study, POX controller will be used to implement our detection module. It contains several in-built components that can be invoked to control various functionalities in the network.

One of the components that is vital to our work is the $forwarding.l3\_learning$ ($l3\_learning\ module$). It serves as a good example of using POX`s packet library in examining and constructing ARP requests and replies.

For the virtual testbed in this experiment, we will be using Mininet network emulator installed on VMWARE emulated machine running on a computer equipped with Ubuntu Linux OS and an Intel Core i5-5200U @2.20GHz resources all available to the virtual machine. Mininet simulated networks run real Linux network applications and provides Linux kernel and networking stack for further development (real-world testing, performance evaluation, and deployment).

Scapy was used to generate both attack and normal traffic during this test. The code for generating random source IP addresses and host IP addresses is in Python.

Our model includes a tree-type network of depth two with three OpenFlow virtual switches (OVS) and 8 hosts with separate IP addresses all connected with a virtual Ethernet cable. The OpenFlow switches are configured to connect to our remote POX controller (running l3_learning module) as depicted in figure 4.

Our simulation consists of varying attack parameters for our DDoS attack as well as different types of legitimate traffic. The traffic generated during tests is a composition of several different protocols: 85% is TCP, 10% is UDP, and 5% of ICMP (ping). After successfully setting up our network topology and running a successful ping command to test connectivity between hosts, we start our detection module on the controller alongside a l3_learning using; $./pox.py\ forwarding.l3\_learning\ detection$. Normal traffic is run using $sudo\ python\ traffic.py$. Attack traffic is run with the command $sudo\ python\ attack.py$. From figure 4, hosts 1, 2 and 4 are the attackers launching TCP, ICMP and UDP attack traffic respectively directed towards the controller whiles the remaining hosts represent normal hosts.

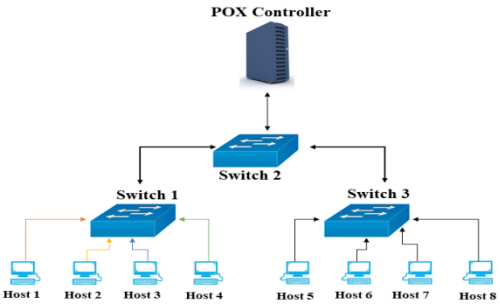

Figure 4 Tree-Type Network Topology Setup

Due to the use of flow-based information for classification of traffic patterns, we set the interval of detection loop to 10 seconds for approximately 1,100 seconds, hence every switch is programmed to obtain flow tables every 10 seconds for feature selection and exported into a CSV format file. Timestamp and duration of each flow was used to sort out flow. Thus 1,000 flow tables were considered at 1,100 different times. During time 151 to 300, we simulated TCP flooding; during time 451 to 650, ICMP attacks were launched; UDP attacks were from time 801 to 950; other remaining time periods had no attacks (i.e. normal status). The 6 features as specified in section 3.3 were extracted by analyzing the flow tables.

A total of 3000 flows were generated; 2000 flows were collected during the interval of attack whiles 1000 were collected during normal traffic. We used 2100 for training and 900 for testing. In Table 4.1 we present the types of attacks launched for the training and testing phases, along with their respective estimated number of generated flows.

Table 8 Attack traffic used for training and testing

Attack types	Training	Testing
TCP/SYN	1720	455
UDP	380	300
ICMP	100	145

An example on the various values of one feature at different times is depicted in Figure 5 and 6. We selected two features as a potential attack indicator to determine their values at different times as depicted below. Average number of packets per flow (APf) and percentage of pair-flows (PPF) were considered most likely to influence the decision of whether a traffic flow is normal or an attack.

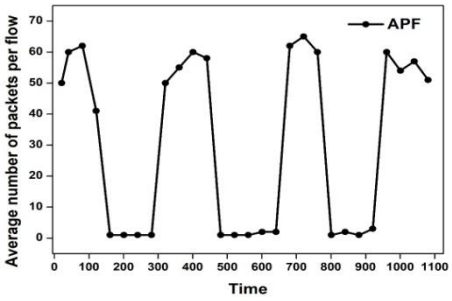

Figure 5 APf of normal and attack traffic

Figure 5 shows the average number of packets per flow (APf) in OpenFlow flooding attack. It can be observed that, there is a rise in the variation of number of packets during normal packets sessions from 40 to 65 which occurred during times 1 to 150, 301 to 450, 651 to 800 and 951 to 1100 whiles flooding attacks occurred between times 151 and 300, 451 and times 650 as well as 801 and 950 with a decrease in packet number between 5 to 0 per flow. This decrease in packet number increases the speed of sending numerous packets and also aids in better distinguishing of feature values.

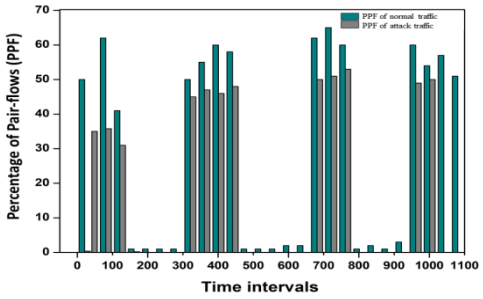

Figure 6 PPF of normal and abnormal traffic

As depicted in Figure 6, PPF is near 0 during attack times 151 to 300, 451 to 650 and 801 to 950 whiles PPF of normal traffic falls between 30 to 60 at times 1 to 150, 301 to 450, 651 to 800 and 951 to 1100.

In order to adopt the features into our training model, we set two values {0,1} for each feature, 1 represents a normal feature value and 0 represents an abnormal value, from our 6 features, we can deduce that the HMM model has 12 possible feature observations.

All the observation sequence values were tagged as normal and abnormal after which they were trained to obtain the parameters π, A, B using the Baum-Welch Algorithm as depicted in section 3.3 earlier. The probability γ_t at different times were also obtained as shown in Figure 7. The times 1-7 at the x-axis represents the time periods 1-150, 151-300, 301-450, 451-650, 651-800, 801-950, 951-1100 respectively. Each bar on the graph denotes the average probability of being in a certain state at 150 different times. It can be observed that the probabilities of SDN in a normal state N is much higher at times 1-150, 301-450, 651-800, 951-1100. Whiles TCP attack is at time 151 to 300, the probability of state T is higher; ICMP attack is at time 451-650, hence, a higher probability of state I; UDP attack at time 801-950 thus a higher probability of state U.

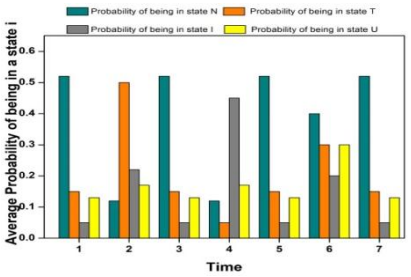

Figure 7 Average probability of being in a state i at various time periods

To get the total risk value, corresponding to the status set {N, T, I, U}, the risk vector is defined as C = {0,20,60,80}. This is done according to the damages and threats of state on the entire network with 0 indicating that there is no risk when the network is in a normal state, and 20 to 80 indicates that the SDN network is under attack under the remaining states which is large enough to set status C with the maximum risk value. The average risk values from time 0 to 1100 are depicted in Figure 8. Since there are different attacks in the network, the risk value changes with regards to the risk vector, thus the higher the risk values, the higher there is a security risk in the network. The times 1-7 at the x-axis represents the time periods 1-150, 151-300, 301-450, 451-650, 651-800, 801-950, 951-1100 respectively.

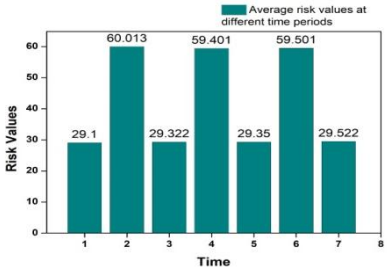

Figure 8 Average risk values at various time periods

From the Figure 8, it can be observed that the risk values conform with the values we set previously with TCP attack having the highest value of 60.013, UDP attack comes second with a value of 59.501 and ICMP has a value of 59.401. Normal values are stable around 29.

3.1 Evaluation

In this section, we depict our results based on different evaluation measures under two broad categories (Accuracy and Efficiency). The measures are shown in a 2d-axis diagram in Figure 9. Various labels are used in our evaluation method and these labels are True Positive (TP): The number of attacks instances classified as attacks, True Negative (TN): The number of non-attacks instances classified as non-attacks, False Negative (FN): The number of attacks instances classified as non-attacks and False Positive (FP): The number of non-attacks instances classified as attacks.

We also used a subset of the KDD Cup 1999 intrusion detection data set prepared to further evaluate our model. The attack launched in our own simulation is constant rate attack, we choose the constant rate TCP attack samples in KDD Cup data. We employed the Weka (Waikato Environment for Knowledge Analysis) [13] tool for this evaluation.

3.1.1 Accuracy

To evaluate the performance of our proposed model in terms of correctness, we calculate its accuracy. Accuracy measures the detection, failure rates as well as the number of false alarms produced by the system. In this work accuracy is measured by using the Viterbi algorithm to generate a likely state sequence and compare it to the known state sequence to get TP, FP, FN, and TN. This form of evaluation helps determine how correctly our model is in classifying and predicting the class label of attack and normal. Our experiment achieved an accuracy of 94.3% with 5.7% false positive rate. In order to access the accuracy of our model, we considered the following measures:

i. Sensitivity and Specificity: Sensitivity is the ratio of the total number of detected true positive that are correctly identified as attack to total number of positive instances. Specificity is the ratio between TN and (FP+TN). The Sensitivity and Specificity can be calculated by using the following equations respectively:

$$\boldsymbol{Sensitivity(TPR)} = \frac{\boldsymbol{TP}}{\boldsymbol{TP+FP}} \qquad 3\text{-}1$$

$$\boldsymbol{Specificity} = \frac{\boldsymbol{TN}}{\boldsymbol{FP+TN}} \qquad 3\text{-}2$$

ii. Precision, Recall and F-measure: Precision is defined as the fraction of retrieved objects (e.g., documents) that are relevant to a given query, Recall is the fraction of the objects that are relevant to a given query or search request and are correctly retrieved. F-measure is calculated by combining precision and recall into a simple metric. Figure 10 depicts the sensitivity and specificity as well as the Precision, Recall and F-measure of our model on SDN generated data and KDD Cup dataset.

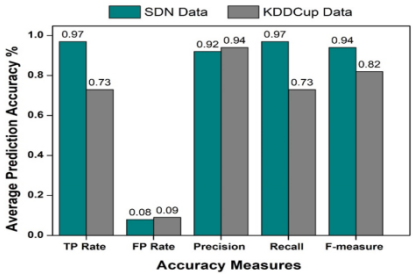

Figure 9 Accuracy of proposed model on SDN data and KDD Cup data

i. Confusion matrixes: A confusion matrix can be used to show the overall performance of an Intrusion Detection System. The confusion matrix can be used in the case of n class problems, where n = number of problems. Table 10 shows the results of our experiment.

Table 9 Confusion matrix of proposed model

Detected / Actual	Benign	Attacks
Benign	TP = 43	FP = 6
Attacks	FN = 1	FN = 50

ii. Receiver operating characteristic (ROC) curve**:** The Receiver Operating Characteristics (ROC) is used in this work for representing the relation between True Positive Rate (TPR) and False Positive Rate (FPR) for the different attack rates tested on our detection model. As depicted in the figures below, x-axis represents TPR which is the fraction of attack traffic correctly as attack traffic whereas y-axis represents FPR which is the fraction of normal traffic incorrectly predicted as attack traffic. Figure shows the ROC Curve of our model and figure shows the ROC Curve of one session of KDD Cup dataset used. From these figures, the curves are closer to both the Y-axis and the point (0, 1) which implies that low false positives were obtained.

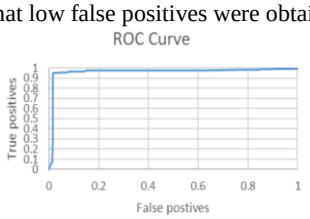

Figure 10 ROC Curve from SDN generated Data

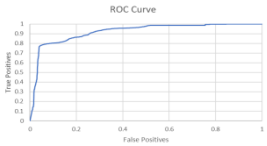

Figure 11 ROC Curve from one session of KDD Cup data

3.1.2 Performance

With the use of flow-based information within a given time interval for collection of samples every 10 seconds and classification of traffic patterns, the issue of minimal effect on the controller in terms of resource usage needs to be examined, remarkable overhead reduction of our detection mechanism something we considered during or design.
In view of this, we compare the CPU time for extraction of features by our system to other DDoS attack detection approaches in SDN.

Table 10 Detection of feature extraction overhead

	Our method	**Braga et al**	**KDD99 based method**
No. of features	6	6	9
CPU times	156	154	237

The results from Table 11 shows that our system is much faster that KDD 99 based method but almost same as the method employed by Braga et al [7].

In chapter 4 of our work, we demonstrated how our proposed mechanism can be implemented and evaluated it in terms of accuracy and performance. We demonstrated that our mechanism extracts significant features with low overhead as compared to other methods. Our model demonstrated 94.3 % accuracy and 5.7% false positive rate with minimum overhead as compared with other methods. Finally, we showed that the flexibility and programmable nature in Software Defined Networking that makes it the new norm for networking in today's world.

4. CONCLUSION

This work presented a mechanism for Distributed Denial of Service anomaly detection of the controller in Software

Defined networking environment. The mechanism employed Hidden Markov Model technique for anomaly detection.
An in-depth description of the methodology design and mechanism and how this model can handle DDoS attacks in SDN was given in chapter 2: employing the Flow_Statistics_Collector for data collection, Baum-Welch algorithm for training and Viterbi algorithm for anomaly detection just to mention a few.

Chapter 3 demonstrated how the proposed mechanism can be implemented and evaluated it in terms of accuracy and performance. We demonstrated that our mechanism extracts significant features with low overhead as compared to other methods. Our model demonstrated 94.3 % accuracy and 5.7% false positive rate with minimum overhead as compared with other methods. Finally, we showed that the flexibility and programmable nature in Software Defined Networking that makes it the new norm for networking in today's world.
Since machine learning techniques are becoming relevant in anomaly detection [14], [15], it`s principles running alongside SDN will make network security vulnerability issues much easier to handle and improve. Therefore, for our future work, we will attempt to identify and mitigate anomalous flow patterns, by applying hybrid machine learning methods on results obtained from backtracking attack traffic generated and designing a highly effective mitigation technique at the cost of low resource usage and a rigorous dataset to handle threats

5. ACKNOWLEDGMENTS

I would like to thank my supervisor Sonjie Wei for the guidance, numerous and helpful discussion throughout this research.

6. REFERENCES

[1] F. Mattern and C. Floerkemeier, "From the internet of computers to the internet of things," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2010, vol. 6462 LNCS, pp. 242–259, doi: 10.1007/978-3-642-17226-7_15.

[2] Z. Wan, "Cloud Computing infrastructure for latency sensitive applications," in International Conference on Communication Technology Proceedings, ICCT, 2010, pp. 1399–1402, doi: 10.1109/ICCT.2010.5689022.

[3] D. K. Bhattacharyya and J. K. Kalita, Network Anomaly Detection. 2013.

[4] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDOS) attacks in cloud computing environments: A survey, some research issues, and challenges," IEEE Communications Surveys and Tutorials, vol. 18, no. 1. Institute of Electrical and Electronics Engineers Inc., pp. 602–622, Jan. 01, 2016, doi: 10.1109/COMST.2015.2487361.

[5] M. Suh, S. H. Park, B. Lee, and S. Yang, "Building firewall over the software-defined network controller," in International Conference on Advanced Communication Technology, ICACT, 2014, pp. 744–748, doi: 10.1109/ICACT.2014.6779061.

[6] D. Kreutz, F. M. V. Ramos, and P. Verissimo, "Towards secure and dependable software-defined networks," in HotSDN 2013 - Proceedings of the 2013 ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, 2013, pp. 55–60, doi: 10.1145/2491185.2491199.

[7] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in Proceedings - Conference on Local Computer Networks, LCN, 2010, pp. 408–415, doi: 10.1109/LCN.2010.5735752.

[8] S. Shin, S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and Vigilant Switch Flow Management in Software-Defined Networks," Accessed: Apr. 20, 2020. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.645.5293.

[9] S. Shin et al., "Rosemary: A Robust, Secure, and High-Performance Network Operating System," Accessed: Apr. 20, 2020. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.650.2722.

[10] Z. Fan, Y. Xiao, A. Nayak, and C. Tan, "An improved network security situation assessment approach in software defined networks," Peer-to-Peer Netw. Appl., vol. 12, no. 2, pp. 295–309, Mar. 2019, doi: 10.1007/s12083-017-0604-2.

[11] R. Mohammadi, R. Javidan, and M. Conti, "SLICOTS: An SDN-based lightweight countermeasure for TCP SYN flooding attacks," IEEE Trans. Netw. Serv. Manag., vol. 14, no. 2, pp. 487–497, Jun. 2017, doi: 10.1109/TNSM.2017.2701549.

[12] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," Proc. IEEE, vol. 77, no. 2, pp. 257–286, 1989, doi: 10.1109/5.18626.

[13] "Weka tutorial: machine learning & data mining." https://wekatutorial.com/ (accessed May 17, 2020).

[14] R. Swami, M. Dave, and V. Ranga, "Software-defined Networking-based DDoS Defense Mechanisms," ACM Comput. Surv., vol. 52, no. 2, pp. 1–36, May 2019, doi: 10.1145/3301614.

[15] D. K. Bhattacharyya, Network anomaly detection?: a machine learning perspective. 2013.