



PROJECT REPORT ON

DDoS Detection System Using Chi-Square Statistics in Software-Defined Networks

RESEARCH AREA: FINANCIAL NETWORKS AND APPLICATIONS

Submitted by :
Aditya Prakash
2nd Year, CE
IIIT, Bhubaneswar

Under the guidance of :
Dr. V. Radha, Associate Professor
IDRBT

**INSTITUTE FOR DEVELOPMENT AND RESEARCH IN
BANKING TECHNOLOGY, HYDERABAD**

10th July 2018

Institute for Development and Research in Banking Technology,

Road No. 1, Castle Hills, Masab Tank,

Hyderabad-500057

CERTIFICATE

This is to certify that Mr. Aditya Prakash, pursuing B. Tech in Computer Engineering at IIIT, Bhubaneswar had undertaken a project as an intern at IDRBT, Hyderabad from May 14th,2018 to July 14th,2018. He was assigned the project of "**Security in Software-Defined Networks**" under my guidance.

Dr. V. Radha
Associate Professor,
IDRBT, Hyderabad

ACKNOWLEDGEMENT

I am greatly indebted to the authorities of Institute for Development and Research in Banking Technology (IDRBT), Hyderabad for providing opportunity to work on the project titled "Security in Software-Defined Networks".

Firstly, I thank and express my solicited gratitude to Dr. V. Radha, Associate Prof., IDRBT for her invaluable help and support which helped me a lot in successfully completing the project. Her depth of knowledge in her field helped me a lot in keeping up with the project.

This project was beneficial for me at each and every step, at the same time it gave me the confidence to work in the real life and professional set up. I feel the experience gained during the project would lead me towards a good professional life.

I would also thank IIIT Bhubaneswar, for allowing me to participate in this Summer Internship Program.

Aditya Prakash
2nd Year, CE
IIIT, Bhubaneswar

Abstract

The evolution in networking technologies and the need for a secure network led to the birth of Software-Defined Networks. Software-defined networking originally defined an approach to designing, building, and managing networks that separates the network's control and forwarding planes thus enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services for applications as SDN cloud computing or mobile networks. Although being secure, SDN and DDoS Attacks have a contradictory relationship. There are several ways in which the different layers of the network can be attacked by hackers, making the resources unavailable to the client(s). This project throws light on various kinds of attacks that can be launched in a SDN architecture and discusses the existing ways in which a DDoS is handled. This also implements an efficient DDoS detection method by invoking a statistic approach that compares source IP addresses' normal and current packet statistics to discriminate whether there is a DoS/DDoS attack. It first collects all resource IPs' packet statistics so as to create their normal packet distribution. Once some IPs' current packet distribution suddenly changes, very often it is an attack. After the detection of any attack, flowmod messages are sent to the controller to block specific port(s) in the switch(s).

Contents

CERTIFICATE	i
ACKNOWLEDGEMENT	ii
Abstract	iii
Introduction	1
1 Software-Defined Networks	2
1.1 What is SDN	2
1.2 SDN Architecture	3
1.2.1 Layers of SDN	3
1.2.2 Interfaces of Contoller	3
1.3 OpenFlow & SDN	4
1.4 Goals and Challenges	7
2 Important Terminologies	9
2.1 DDoS Attack and its types	9
2.2 SDN & DDoS	10
2.3 Statistical DDoS Detection Methods	11
2.3.1 Pearson's Chi-square Goodness-of-fit Test:	11
2.3.2 Entropy	12
3 Proposed Methodology	13
3.1 Algorithms	15

3.2	Flowchart	17
4	Experiment	18
4.1	Software and Tools Used	18
4.1.1	VirtualBox	18
4.1.2	POX	18
4.1.3	Mininet	19
4.1.4	OpenFlow	19
4.1.5	Scapy	20
4.1.6	PyPy Interpreter	20
4.1.7	PuTTY	20
4.1.8	XMing	20
4.2	VM Setup	21
4.3	Experiment Walkthrough	24
4.3.1	Overview	24
4.3.2	Results	27
	Conclusion	28
	References	29

Introduction

Traditionally, computer networks are typically constructed from a large number of network devices with complex protocols, which are implemented or embedded on them. Network engineers are responsible for configuring the policies to respond to a wide range of network events and application scenarios. They manually transform these high-level policies into low-level configuration commands.

The administrators have to configure the network devices (eg. switch, routers,etc) manually on a device-to-device basis. Next step is to use device-level management tools to update numerous configuration settings such as QoS, VLANs, ACLs, etc. Thus, this makes the system too complex to handle, error-prone and vulnerable to security breaches.

There are a variety of equipments of different vendors each having their own configuration methods. Hence, to deploy a configuration a administrator has to have a complete knowledge about all the devices available.

This is where Software-Defined Network (SDN) comes into play. SDN completely overcomes these issues with its concept of plane separation i.e, abstracting the network from the hardware, which means network policies no longer had to be configured manually into the network devices. A key characteristic of the SDN approach is automating the process of configuring the devices, enabling an administrator to manage the entire network as if it were a single device.

Although the centralized controller brings about flexibility, dynamic control of forwarding rules & software-based traffic-analysis but this makes it the potential single point of attack and failure. The attackers could take down a network by exploiting the central controller of a network and likewise bring the whole network down.

Chapter 1

Software-Defined Networks

1.1 What is SDN

As the Open Networking Foundation (ONF) defines, SDN is the physical separation of control plane from the forwarding plane and where a control plane controls several devices. This separation paves the way for a more flexible, programmable, vendor-agnostic, cost-effective, and innovative network architecture. An SDN network is characterized by five fundamental traits: plane separation, a simplified device, centralized control, network automation and virtualization, and openness. So the network devices no more have to decide where to send a new unknown packet. It is now the job of a centralised controller, or more precisely the ‘brain’ of the SDN architecture, to decide where and whom to send the packets to.

SDN architecture is:

- Directly programmable-Decoupling the network control and the forwarding functions enables the network control to be directly programmable.
- Centrally Managed-Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single, logical switch.
- Programmatically Configured-SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
- Open Standards-based and Vendor-Neutral-When implemented through open standards, SDN simplifies network design and operation because instructions are provided

by SDN controllers instead of multiple, vendor-specific devices and protocols.

1.2 SDN Architecture

1.2.1 Layers of SDN

ONF describes a high level architecture of SDN, which functionally and vertically split into three layers.

- **Infrastructure layer:** This layer consists of forwarding devices like the physical switch, router, etc. Software switches which can be accessible via open interfaces, also part of this layer. This layer is considered as forwarding layer since it allows packet switching and forwarding.
- **Control Layer:** The control layer is also referred as control plane that comprises a set of software-enabled SDN controllers. This layer allows the network administrator to apply custom policies to the physical layer devices. About the controller functionalities will be briefly discussed next.
- **Application layer:** Application layer deals with enduser business applications that utilizes the SDN services. Business application such as energy efficient networking, security monitoring, network virtualization, etc

1.2.2 Interfaces of Controller

- **Northbound Interface :** The north bound APIs represents are interfaces between the controller and the applications application layer. This interface helps the application developers to manage the network through the program.
- **Southbound Interface :** Southbound interface creates a channel to interact with the controller and underlying forwarding elements. OpenFlow is the standardized protocol supported by ONF, is the widely used southbound interface, which establishes a secured link between the controller and forwarding devices.
- **East-West Interface :** The east-west protocol helps in the intra-domain communication by exchanging the network view among the controllers in a multi-controller based architecture.

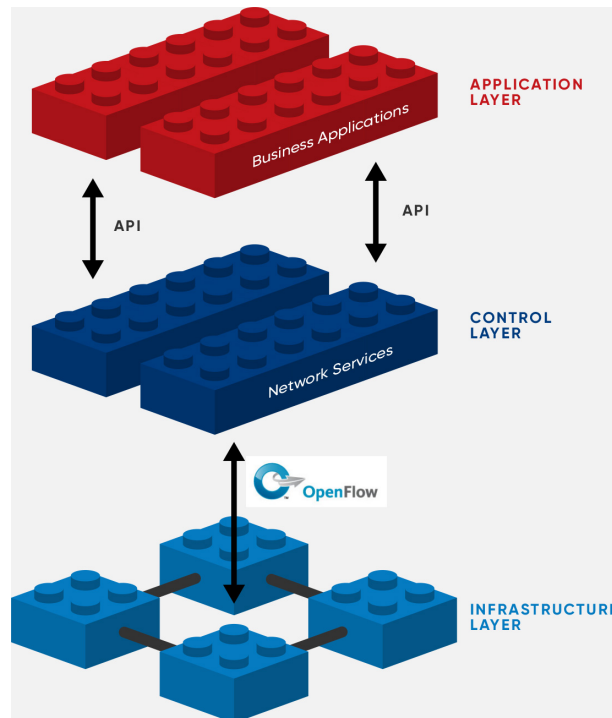


Figure 1.1: SDN Architecture

Source: ONF

1.3 OpenFlow & SDN

The OpenFlow protocol is a key enabler of Software-Defined Networks. It is the first standardized southbound interface. It allows the direct manipulation of the forwarding planes of the OpenFlow-based switches. OpenFlow is based on an Ethernet switch, with an internal flow-table, and a standardized interface to add and remove flow entries.

An OpenFlow Switch consists of at least three parts:

- A Flow Table, with an action associated with each flow entry, to tell the switch how to process the flow.
- A Secure Channel that connects the switch to a remote control process (called the controller), allowing commands and packets to be sent between a controller and the switch using
- The OpenFlow Protocol, which provides an open and standard way for a controller to communicate with a switch. By specifying a standard interface (the OpenFlow

Protocol) through which entries in the Flow Table can be defined externally, the OpenFlow Switch avoids the need for researchers to program the switch.

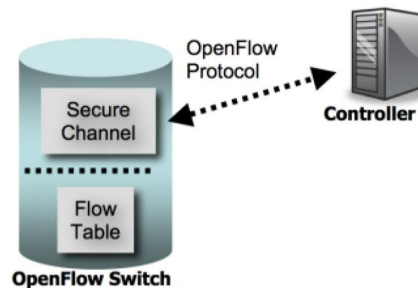


Figure 1.2: A OF Switch communicating with controller over TLS

Source: RSAConference

The OpenFlow protocol supports three message types, controller-to-switch, asynchronous, and symmetric, each with multiple subtypes.

- **Controller-to-Switch:**

Controller/switch messages are initiated by the controller and may or may not require a response from the switch.

1. **Features:** Upon Transport Layer Security (TLS) session establishment, the controller sends a features request message to the switch. The switch must reply with a features reply that specifies the capabilities supported by the switch.
2. **Configuration:** The controller is able to set and query configuration parameters in the switch. The switch only responds to a query from the controller.
3. **Modify-State:** Modify-State messages are sent by the controller to manage state on the switches. Their primary purpose is to add/delete and modify flows in the flow tables and to set switch port properties.
4. **Read-State:** Read-State messages are used by the controller to collect statistics from the switch's flow-tables, ports and the individual flow entries.
5. **Send-Packet:** These are used by the controller to send packets out of a specified port on the switch.
6. **Barrier:** Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.

- **Asynchronous**

Asynchronous messages are sent without the controller soliciting them from a switch.

Switches send asynchronous messages to the controller to denote a packet arrival, switch state change, or error. The four main asynchronous message types are described below.

1. **Packet-in:** For all packets that do not have a matching flow entry, a packet-in event is sent to the controller (or if a packet matches an entry with a “send to controller” action). If the switch has sufficient memory to buffer packets that are sent to the controller, the packet-in events contain some fraction of the packet header (by default 128 bytes) and a buffer ID to be used by the controller when it is ready for the switch to forward the packet. Switches that do not support internal buffering (or have run out of internal buffering) must send the full packet to the controller as part of the event.
2. **Flow-Removed:** When a flow entry is added to the switch by a flow modify message, an idle timeout value indicates when the entry should be removed due to a lack of activity, as well as a hard timeout value that indicates when the entry should be removed, regardless of activity. The flow modify message also specifies whether the switch should send a flow removed message to the controller when the flow expires. Flow modify messages which delete flows may also cause flow removed messages.
3. **Port-status:** The switch is expected to send port-status messages to the controller as port configuration state changes. These events include change in port status (for example, if it was brought down directly by a user) or a change in port status as specified by 802.1D (Spanning Tree).
4. **Error:** The switch is able to notify the controller of problems using error messages.

- **Symmetric**

Symmetric messages are sent without solicitation, in either direction.

1. **Hello:** Hello messages are exchanged between the switch and controller upon connection startup.
2. **Echo:** Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. They can be used to indicate the latency, bandwidth, and/or liveness of a controller-switch connection.
3. **Vendor:** Vendor messages provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space. This is a staging area for features meant for future OpenFlow revisions.

1.4 Goals and Challenges

The goal of Software-Defined Networking is to enable cloud computing and network engineers and administrators to respond quickly to changing business requirements via a centralized control console. Social media, mobile devices, and cloud computing are pushing traditional networks to their limits. Compute and storage have benefited from incredible innovations in virtualization and automation, but those benefits are constrained by limitations in the network. Administrators may spin up new compute and storage instances in minutes, only to be held up for weeks by rigid and oftentimes manual network operations.

Software-defined networking has the potential to revolutionize legacy data centers by providing a flexible way to control the network so it can function more like the virtualized versions of compute and storage today.

Challenges:

The following challenges continue to be relevant today in SDN:

- **Latency:** The purpose of having a central controller being satisfied, this leads to a certain number of decisions suffering round-trip latency as the networking element requests policy directions from the controller. Also, with the central controller providing policy advice for a number of network devices, it is unknown whether the conventional servers on which the controller runs will be able to service these requests at sufficient speed to have minimal or no impact on network operation.
- **Scale:** Having a centralized controller means that responsibility for the topological organization of the network, determination of optimal paths, and responses to changes, must be handled by the controller. As has been argued, this is the appropriate location for this functionality; however, as more and more network devices are added to the network, questions arise of scale and the ability of a single controller to handle all those devices. It is difficult to know how well a centralized system can handle hundreds, thousands, or tens of thousands of network devices, nor what is the solution when the number of network devices outgrows the capacity of the controller to handle them.
- **High Availability (HA):** The centralized controller must not constitute a single point of failure for the network. This implies the need for redundancy schemes in a number of areas. First, there must be redundant controllers such that processing power is available in the event of failure of a single controller. Secondly, the actual data used by the set of controllers needs to be mirrored such that they can program the network devices in a consistent fashion. Thirdly, the communication paths to the different controllers need to be redundant to ensure that there is always a functioning communications path between a switch and at least one controller.

- Security: Having a centralized controller means that security attacks are able to focus on that one point of failure, and, thus, the possibility exists that this type of solution is more vulnerable to attack than a more distributed system. It is important to consider what extra steps must be taken to protect both the centralized controller and the communication channels between it and the networking devices.

Been introduced to SDN, OpenFlow and the challenges faced by the current day SDN network, in the next chapter we discuss the security challenges more broadly - DDoS attacks in SDN.

Chapter 2

Important Terminologies

2.1 DDoS Attack and its types

A distributed denial of service (DDoS) attack is a malicious attempt to make an online service unavailable to users, usually by temporarily interrupting or suspending the services of its hosting server. A DDoS attack is launched from numerous compromised devices, often distributed globally in what is referred to as a botnet. It is distinct from other denial of service (DoS) attacks, in that it uses a single Internet-connected device (one network connection) to flood a target with malicious traffic.

Broadly speaking, DoS and DDoS attacks can be divided into three types:

1. **Volume Based Attacks:** Includes UDP floods, ICMP floods, and other spoofed-packet floods. The attack's goal is to saturate the bandwidth of the attacked site, and magnitude is measured in bits per second.
2. **Protocol Attacks:** Includes SYN floods, fragmented packet attacks, Ping of Death, Smurf DDoS and more. This type of attack consumes actual server resources, or those of intermediate communication equipment, such as firewalls and load balancers, and is measured in packets per second.
3. **Application Layer Attacks:** Includes low-and-slow attacks, GET/POST floods, attacks that target Apache, Windows or OpenBSD vulnerabilities and more. Comprised of seemingly legitimate and innocent requests, the goal of these attacks is to crash the web server, and the magnitude is measured in Requests per second.

2.2 SDN & DDoS

A Possible DDoS Attacks on SDN SDN itself may be a target of DDoS attacks. Since SDN is vertically split into three main functional layers, including infrastructure layer, control layer, and application layer, potential malicious DDoS attacks can be launched on these three layers of SDN's architecture. Based on the possible targets, we can classify the DDoS attacks launching on SDN into three categories: application layer DDoS attacks, control layer DDoS attacks, and infrastructure layer DDoS attacks.

- Application layer DDoS attacks: There are two methods to launch application DDoS attacks. One is to attack some applications, the other is to attack northbound API. Since isolation of applications or resources of SDN is not well solved, DDoS attacks on one application can affect other applications.
- Control layer DDoS attacks: The controllers could potentially be seen as a single point of failure risk for the network, so they are a particularly attractive target for DDoS attack in the SDN architecture. The following methods can launch control plane DDoS attacks: attacking controller, northbound API, southbound API, westbound API or eastbound API. For example, many conflicting flow rules from different applications may cause DDoS attacks on the control layer. Within the operation of SDN, data plane will typically ask the control plane to obtain flow rules when the data plane sees new network packets that it does not know how to handle. There are two options for the handling of a new flow when no flow match exists in the flow table: either the complete packet or a portion of the packet header is transmitted to the controller to resolve the query. With a large volume of network traffic, sending the complete packet to the controller would occupy high bandwidth.
- Infrastructure layer DDoS attacks: There are two methods to launch data plane DDoS attacks. One is to attack some switches, the other is to attack southbound API. For example if only header information is transmitted to the controller, the packet itself must be stored in node memory until the flow table entry is returned. In this case, it would be easy for an attacker to execute a DDoS attack on the node by setting up a number of new and unknown flows. As the memory element of the node can be a bottleneck due to high cost, an attacker could potentially overload the switch memory. The generated fake flow requests can produce many useless flow rules that need to be held by the data plane, thus making the data plane hard to store flow rules for normal network flows.

2.3 Statistical DDoS Detection Methods

The statistic method of anomaly detection methods firstly defines the independent variables based on the packet features such as the source IP address or the destination port number, secondly extracts the symbols in packet header, finally calculate the entropy value and chi-square value based on the frequency of features.

2.3.1 Pearson's Chi-square Goodness-of-fit Test:

Pearson's Chi-square (χ^2) goodness of fit test is a non-parametric test that is used to determine how the observed value of a phenomena is significantly different from the expected value. Pearson's chi-square test is used for distribution comparison in cases where the measurements involved are discrete values. For example, it could be used to test the distribution of TCP SYN flag values (0 or 1) or protocol numbers. The test works best when the number of possible values is small. This can often be achieved through "binning", that is combining a set or range of possible values and treating them as one.

For a sample of some packets, let k be the number of available bins. Define O_i as the number of packets whose value falls in the i^{th} bin and E_i as the expected number of packets in the i^{th} bin under the typical distribution.

For the χ^2 goodness-of-fit computation, with the data divided into k bins, the test static is defined as:

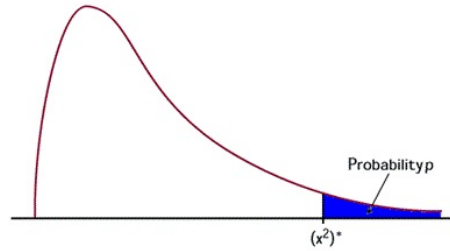
$$\chi^2 = \sum_{i=1}^k \frac{(O_i - E_i)^2}{E_i} \quad (2.1)$$

where, O_i is the observed frequency for the bin i and E_i is the expected frequency for the bin i .

For the test, computational procedure includes the following steps.

1. Calculate the χ^2 test statistic, which resembles a normalized sum of squared deviations between observed and theoretical frequencies.
2. Determine the degrees of freedom, df , of that statistic. For a test of goodness-of-fit, this is essentially the number of categories reduced by the number of parameters of the fitted distribution. Here, $df = \text{Number of groups}(/bins) - 1$
3. Select a desired level of confidence (significance level, p-value or alpha level) for the result of the test.
4. Compare χ^2 value to the critical value from the chi-squared distribution with df degrees of freedom and the selected confidence level.

5. If the test statistic exceeds the critical value of χ^2 , the null hypothesis (H_0 = there is no difference between the distributions) can be rejected, and the alternative hypothesis (H_a = there is a difference between the distributions) can be accepted, both with the selected level of confidence.

Figure 2.1: χ^2 and p -value

Source: Internet

2.3.2 Entropy

By definition entropy is the randomness of the occurrence of an event. Let an information source have independent symbols each with the probability of P_i . Then the entropy H is defined as:

$$H = - \sum_{i=1}^n P_i \log P_i \quad (2.2)$$

where,

$$P_i = \frac{x_i}{n}$$

and x_i , is the frequency of the occurrence of parameter i in the observation of on n events.

NOTE: The entropy value increases when each symbol has the uniform frequency. On the other hand, the entropy value decreases when the specific symbol appears with concentration.

SUMMARY: Having discussed various types of DDoS attacks and the DDoS attacks that can be induced on present SDN architecture, we have proper knowledge to distinguish a normal traffic from an anomalous traffic. We also come to know about two main statistical methods to detect anomaly in network traffic. In the next chapter, we discuss about a DDoS detection model based on the χ^2 goodness-of-fit test and propose the methodology.

Chapter 3

Proposed Methodology

Every new packet that comes to a openflow-enabled switch is send to the controller as a PacketIn event. During a DDoS the attacker sends a huge amount of traffic from numerous compromised devices, often distributed globally in what is referred to as a botnet, through the network consuming the resource as well as the bandwidth. A large number of nodes send packets to the victim host.

Hence, a detector placed at the controller collects the packets sent to a switch. Each new packet in the network is sent to the controller for specifying a flow for the packet and similar ones. This analyses the source IPs of the packets and counts the number of packets that each sender (source IP) sends to a host in a given network to produce a source-IP distribution table. The table is later then used to detect DoS/DDoS and determine who is issuing the attack.

Source-IP Distribution Table:

The source-IP distribution table is based on the attribute IP. Each source-IP has it's own tuple in the table which records various packet information. This table consists of Source-IP, group number, packet-information, 10-secbase, group-percentage (G%), N-percentage (N%), chi-square % and chi-square for amount where Source-IP records source IP of a received packet. The packet information field has 10 sub-fields, including past 7 days' count, one-week-count, current-day-count and past-10s-count.

Source-IP-addr	Port #	Group #	Packet-information										10-sec-base	G%	N%	Chi-square %	Chi-square for amount
			Past 7 th -day count	Past 6 th -day count	Past 5 th -day count	Past 4 th -day count	Past 3 rd -day count	Past 2 nd -day count	Past 1 st -day count	One-week count= $\sum_{i=1}^7$ day count	Current-day count	Past-10-sec count					

Figure 3.1: A source-IP distribution table

Past i^{th} day counts the past i^{th} day's count, one-week-count counts the number of pack-

ets that have been recieved from an IP in the past 7 days of the week, current-day-count counts the number of packets that have been recieved in the last 24 hours and past-10s-count counts the number of packets in last 10 seconds. 10-sec-base records the 10 second average of an IP's normal traffic in the past one week.

Group-percentage represents the percentage of packet counts that a group received previously under normal circumstance over the total number of packets that the system has received, N-percentage is defined as the number of packets that a group has currently received, e.g., Q_i , over total number of packets the network has received, i.e., Q , in the past 10 seconds (i.e., $N - Percentage = \frac{Q}{Q_i}$), chi-square-percentage defined as $\frac{(N\% - G\%)^2}{G\%}$ is used to analyze whether or not there is a resource consumption attack.

Before detection, the statistics are collected for at least a week, attack packets filtered out and their normal packet counts are calculated. The IP with the largest amount of packets is ranked number one. The second largest amount is ranked number two, and so on. The baseline profile is updated every 24 hours. Those ranked number two and three are group 1. The classification is shown in the following figure.

Group#	Range of source IPs	Ranked no.
Group 0	2^0	1
Group 1	$2^1 \sim 2^2 - 1$	2,3
Group 2	$2^2 \sim 2^3 - 1$	4,5,6,7
Group 3	$2^3 \sim 2^4 - 1$	8,9,10,11,12,13,14,15
Group 4	$2^4 \sim 2^5 - 1$	16,17,18,.....31
Group 5	$2^5 \sim 2^6 - 1$	32,33,34,.....63
Group 6	$2^6 \sim 2^7 - 1$	64,65,66,.....127
Group 7	$2^7 \sim 2^8 - 1$	128,129,130,.....255
Group 8	$2^8 \sim 2^9 - 1$	256,257,258,.....511
Group 9	$2^9 \sim 2^{10} - 1$	512,513,514,.....1023
Group 10	$2^{10} \sim 2^{11} - 1$	1024,1025,1026,.....2047
Group 11	$2^{11} \sim 2^{12} - 1$	2048,2049,.....4095
Group 12	2^{12} and up	4096 and remaining

Figure 3.2: Classification of collected Source-IPs

3.1 Algorithms

ALGORITHM 1:

For every packetIn event the packet information (packet count for current day, count for last 10 seconds, etc) is collected against each source IP from the events and maintains a table for it. This forms a current model for the statistics to be compared with the baseline model. It also checks if there is an attack, every 10 seconds.

Input: An incoming packet to the controller with source IP ' $S-IP$ ' in table T_{S-IP} .

Output: Update the source IP's information in the table, or insert a new tuple against S-IP into the table.

Algorithm 1 To establish a Source IP distribution table T_{S-IP} :

```

if  $S - IP \notin T_{S-IP}$  then
     $t.past\_10s\_packet\_count = 0$ 
     $t.current\_day\_count = 0$ 
end if
 $t.past\_10s\_packet\_count++ = 1$ 
 $t.current\_day\_count++ = 1$ 
if Timer times out then
    Call Algorithm 3
     $t.past\_10s\_packet\_count = 0$ 
    Set Timer to 10s
end if

```

ALGORITHM 2:

It is used to establish a baseline model from the past network traffic data, summing up the past 7 days packet count and classifying the source IPs in the S-IP table into groups. Grouping is done on the basis that the S-IP with the largest number of packets in current day field is considered as group 1. The next two IPs with large current day packet count in the list are considered as group 2. The next four in the list in group 3 and so on.

Input: It takes the source IP table as the input.

Output: Tuples in the table are classified into groups numbered from 1.

Algorithm 2 To establish a baseline profile using past 7 days stats:

```

if Timer times out then
  for  $i = 6$  to  $1$  do
    shift  $past\_i^{th}\_day\_count$  to  $past\_ (i + 1)^{th}\_day\_count$ 
  end for
  shift  $current\_day\_count$  to  $past\_1^{th}\_day\_count$ 
  for all  $tuples \in T_{S-IP}$  do
     $t.one\_week\_count = \sum_{i=1}^7 past\_i^{th}\_day\_count$ 
  end for
  call  $SortData(current\_day\_count, T_{S-IP})$ 
  for all  $tuples \in T_{S-IP}$  do
    Fill in the group number to which the each tuple belongs to group# field
  end for
  Set  $Timer=24$  hr
end if

```

SortData(x,y)

```

Sort tuples in  $y$  based on field  $x$ 
The IP ranked top one is  $x - group\ 0$ 
The IPs ranked the top  $2^{nd}$  and top  $3^{rd}$  are  $x - group\ 1$ 
The IPs ranked the top  $4^{th}$  to top  $7^{th}$  are  $x - group\ 2$ 
The IPs Ranked the top  $8^{th}$  to top  $15^{th}$  are  $x - group\ 3$ 
....
The IPs ranked the top  $(2k)^{th}$  to top  $(2k + 1)^{th}$  are  $x - group\ k$ 

```

ALGORITHM 3:

A method to detect DoS/DDoS is as follows. Let G_i be the average number of packets that group i i.r, N_i has received per 10 seconds under the circumsatnce of no attacks,

$$G_i = \frac{\sum_{t \in N_i} t.one-week-count}{\frac{7*24*60*60}{10}} .$$

$\chi^2 = \sum_{i=1}^n \frac{(N_i - G_i)^2}{G_i}$ where N_i is the number of packets that a group i has received currently in the past 10 seconds.

In this experiment we choose the level of confidence $p = 0.05$ for which $\chi^2=11.07$. If the $\chi^2 > \chi_{p=0.05}^2$, then we say there is a DoS/DDoS attack. If it is the otherway round, then the network is not under a DoS/DDoS attack.

Input: The current model and the baseline model

Output: Returns attack information if there is a DoS/DDoS attack.

Algorithm 3 Detecting DDoS at end of every 10s using chi-square statistics.

```

for all Group  $G_i$  do
   $G_i = \frac{\sum_{t \in N_i} t.one-week-count}{\frac{7*24*60*60}{10}}$ 
end for
 $\chi^2_{count} = \sum_{i=1}^n \frac{(N_i - G_i)^2}{G_i}$ 
if  $\chi^2_{count} > \chi^2_{p=0.05}$  then
  if  $(packet - count)_{dpid} > threshold$  then
    Send an attack information and a flowmod message to block ports of switch
  end if
end if

```

3.2 Flowchart

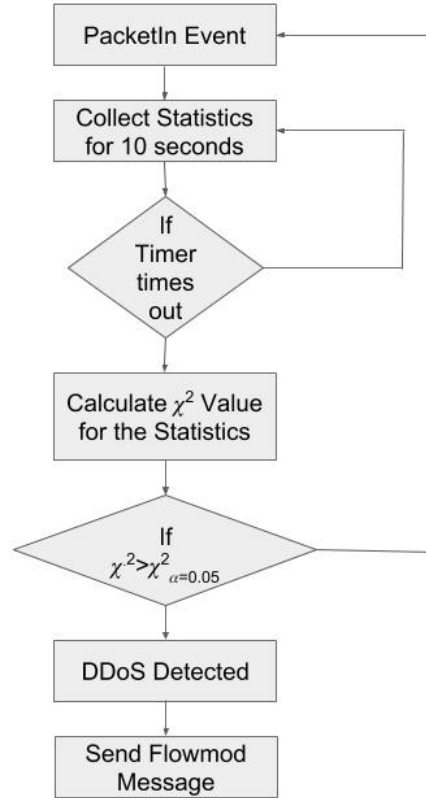


Figure 3.3: Detecting and Mitigating DDoS

Next chapter gives a step-by-step procedure to implement and test the model we just discussed on a SDN architecture built with POX controller and describes various dependencies before carrying out the experiment.

Chapter 4

Experiment

4.1 Software and Tools Used

4.1.1 VirtualBox

VirtualBox is a powerful x86 virtualization software. It helps in the creation and management of virtual machines (VM) in a host machine. It is free and open-source. A virtual machine is an emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination.

VirtualBox has an extremely modular design with well-defined internal programming interfaces and a client/server design. This makes it easy to control it from several interfaces at once: for example, you can start a virtual machine in a typical virtual machine GUI and then control that machine from the command line, or possibly remotely.

4.1.2 POX

POX is yet another OpenFlow based SDN controller completely written in python. POX can be used with the "standard" Python interpreter (CPython), but also supports PyPy. Here are a few modules of POX that will be used in the experiment.

openflow.keepalive:

This component causes POX to send periodic echo requests to connected switches. This addresses two issues. First, some switches (including the reference switch) will assume that an idle control connection indicates a loss of connectivity to the controller and will disconnect after some period of silence (often not particularly long). This behavior is almost certainly broken: one can easily argue that if the switches want to disconnect when the

connection is idle, it is their responsibility to send echo requests, but arguing won't fix the switches.

Secondly, if you lose network connectivity to the switch, you don't immediately get a FIN or a RST, so it's hard to say exactly when you'll notice that you've lost the switch. By sending echo requests and tracking their responses, you get a bound on how long it will take to notice.

forwarding.l3_learning :

This component is not quite a router, but it's also definitely not an L2 switch. It's an L3-learning-switchy-thing. Perhaps the most useful aspect of it is that it serves as a pretty good example of using POX's packet library to examine and construct ARP requests and replies.

`l3_learning` does not really care about conventional IP stuff like subnets – it just learns where IP addresses are. Unfortunately, hosts usually do care about that stuff. Specifically, if a host has a gateway set for some subnet, it really wants to communicate with that subnet through that gateway. To handle this, you can specify "fake gateways" in the commandline to `l3_learning`, which will make hosts happy. For example, if you have some machines which think they're on 10.x.x.x and others that think they're on 192.168.0.x and they think there are gateways at the ".1" addresses.

4.1.3 Mininet

Mininet is a network emulator which creates a network of virtual hosts, switches, controllers, and links. Mininet hosts run standard Linux network software, and its switches support OpenFlow for highly flexible custom routing and Software-Defined Networking.

Mininet creates virtual networks using process-based virtualization and network namespaces - features that are available in recent Linux kernels. In Mininet, hosts are emulated as bash processes running in a network namespace, so any code that would normally run on a Linux server (like a web server or client program) should run just fine within a Mininet "Host". The Mininet "Host" will have its own private network interface and can only see its own processes. Switches in Mininet are software-based switches like Open vSwitch or the OpenFlow reference switch. Links are virtual ethernet pairs, which live in the Linux kernel and connect our emulated switches to emulated hosts (processes).

4.1.4 OpenFlow

One of the primary purposes for using POX is for developing OpenFlow control applications – that is, where POX acts as a controller for an OpenFlow switch (or, in more proper terminology, an OpenFlow datapath). Refer to previous chapters for a detailed idea about OpenFlow.

4.1.5 Scapy

Scapy is a powerful interactive packet manipulation tool, packet generator, network scanner, network discovery, packet sniffer, etc. It can for the moment replace hping, parts of nmap, arpspoof, arp-sk, arp-ing, tcpdump, tshark, p0f, etc.

These kind of tools do two things : sending packets and receiving answers. That's what scapy does : you define a set of packets, it sends them, receives answers, matches requests with answers and returns a list of packet couples (request, answer) and a list of unmatched packets.

Scapy uses the python interpreter as a command board. That means that you can use directly python language. If you give a file as parameter when you run scapy, your session will be saved when you leave the interpreter, and restored the next time you launch scapy.

4.1.6 PyPy Interpreter

While POX is not as heavily tested as the normal Python interpreter, it's a goal of POX to run well on the PyPy Python runtime. There are two advantages of this. First, PyPy is generally quite a bit faster than CPython. Secondly, it's very easily portable – you can easily package up POX and PyPy in a single tarball and have them ready to run.

PyPy is a replacement for CPython. It is built using the RPython language that was co-developed with it. The main reason to use it instead of CPython is speed: it runs generally faster. PyPy implements Python 2.7.13 and 3.5.3. It supports all of the core language, passing the Python test suite. It supports most of the commonly used Python standard library modules.

4.1.7 PuTTY

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. It can also connect to a serial port.

4.1.8 Xming

Xming provides a minimalist yet functional X11 server for use in a Windows environment. This allows users to utilize graphical applications on a remote workstation without the need for large amounts of hard drive space.

4.2 VM Setup

1. Download and install a virtualization program such as: VMware Workstation for Windows or Linux, VMware Fusion for Mac, VirtualBox for any platform, or qemu for Linux and download the Mininet VM from the given link. (For the experiment, VirtualBox is used)

<https://github.com/mininet/mininet/wiki/Mininet-VM-Images>

And follow the installation and booting instructions from the link itself.

2. After importing the VM's .ovf image in VirtualBox, select "settings," and add an additional host-only network adapter that you can use log in to the VM image. Start the VM.

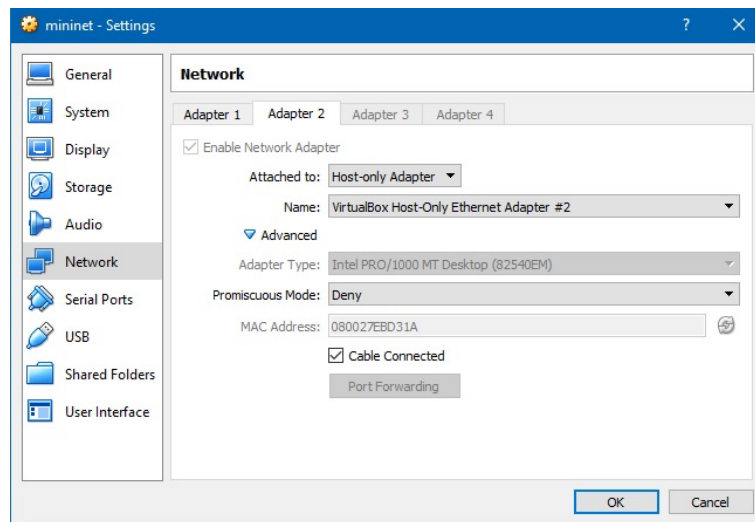


Figure 4.1: Network Adapter Configuration(VirtualBox)

3. Boot into your VM and make sure that this new interface has showed up as eth1:

```
$ifconfig -a
```

```

mininet [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
eth0      Link encap:Ethernet  HWaddr 08:00:27:63:67:30
          inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:47 errors:0 dropped:0 overruns:0 frame:0
          TX packets:226 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6178 (6.1 KB)  TX bytes:20126 (20.1 KB)

eth1      Link encap:Ethernet  HWaddr 08:00:27:eb:d3:1a
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:2480 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2480 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:124000 (124.0 KB)  TX bytes:124000 (124.0 KB)

ovs-system Link encap:Ethernet  HWaddr 12:ce:55:12:59:27
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
mininet@mininet-vm:~$

```

Figure 4.2: Configure ethX to connect to VM from host

To allow network access in the VM, execute:

```
$sudo dhclient eth1
```

- Download, install and start Xming; nothing exciting will happen but it will show an icon on the right of the toolbar.

Download and copy the putty.exe file into an accessible location (This step is required only if your host system is Windows).

Next open a command prompt and type in the following commands. The option `-X` will enable X11 forwarding for the host machine.

```
>cd <dir>
>putty.exe -X mininet@<ip-of-the-virtual-machine>
```

A terminal window should appear. If you have succeeded, you are done with the basic setup. Enter the password as *mininet*.

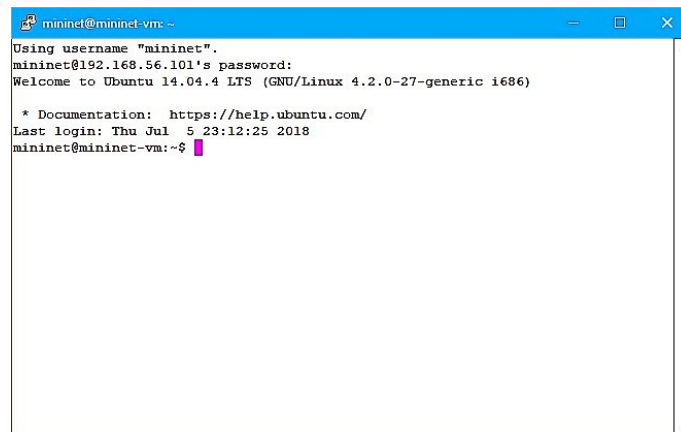


Figure 4.3: SSH from puTTY

5. Since we are using the PyPy interpreter along with POX for performance boost, download the already built PyPy package from the given link.

<https://bitbucket.org/pypy/pypy/downloads/pypy2-v6.0.0-linux32.tar.bz2>

Now, follow the procedure to completely set-up the dependencies:

```
#Download built tar.gz file for PyPy from the site according to
#your system.
$tar -xf pypy-xxxxxx.tar.bz2
$cd pypy-xxxxxx
$mkdir ~/pox/pypy
$cp -rf * ~/pox/pypy

$sudo apt-get install gfortran
$sudo apt install libblas-dev liblapack-dev

$cd ~/pox
$./pypy/bin/pip install cython
$./pypy/bin/pip install numpy==1.14.5
$./pypy/bin/pip install pandas
$./pypy/bin/pip install scipy
```

Modify the Signed to int on the line inside pypy/include/pypy_decl.h (fourth argument to the function)(line : 851), to build pandas successfully :

```
PyAPI_FUNC(PyObject *) PyUnicode_DecodeUTF16(const char *arg0,
Signed arg1, const char *arg2, Signed *arg3);    to
PyAPI_FUNC(PyObject *) PyUnicode_DecodeUTF16(const char *arg0,
Signed arg1, const char *arg2, int *arg3);
```

4.3 Experiment Walkthrough

The experiment was done on a HP Pavilion laptop with a dual core processor, 2.3GHz clock speed and 8GB of RAM. The host system runs on Windows 10.

4.3.1 Overview

In this particular experiment we are trying to distinguish between DDoS and normal traffic using the χ^2 statistics. So, using Mininet, a tree-type network of depth two with five switches and 16 hosts was created, upon which we would test this model. POX is used as the controller of choice. The normal and attack packets are generated from a fixed number of hosts (IPs) outside the network we created. The detector module deployed on top of the controller captures the packets and determines if there is an attack every 10 seconds.

NOTE: For the chi-square test we select the level of confidence of 5% and χ^2 threshold as 11.07 .

- Make a copy of the modified l3_learning module (l3_new.py) and detect_new.py into the POX's folder.

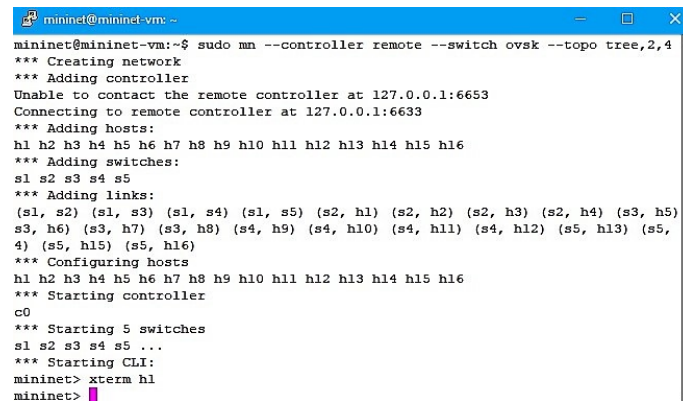
```
$cp ./l3_new.py ./detect_new.py ~/pox/pox/forwarding
```

- Make a copy of the attack.py and traffic.py into accessible location in you VM.
- In a SSH Window start the POX controller with the l3_new module.

```
$cd pox  
$./pox.py forwarding.l3_new openflow.keepalive
```

- In m another SSH Window create a mininet topology by entering the following command

```
$sudo mn --controller remote --switch ovsk --topo tree,2,4
```



```

mininet@mininet-vm:~$ sudo mn --controller remote --switch ovsk --topo tree,2,4
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s2, h1) (s2, h2) (s2, h3) (s2, h4) (s3, h5)
(s3, h6) (s3, h7) (s3, h8) (s4, h9) (s4, h10) (s4, h11) (s4, h12) (s5, h13) (s5,
4) (s5, h15) (s5, h16)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> xterm h1
mininet>

```

Figure 4.4: Mininet Tree Topology

- Open the Xterm of host h1 and launch the normal traffic in the mininet network. Let it run for few minutes to generate a base line profile and then observe the χ^2 values and *p-values* in the controller.

```
$python traffic.py -s 2 -e 17
```



```

mininet@mininet-vm:~/pox
INFO:openflow.of_01:[00-00-00-00-00-05 75] closed
INFO:openflow.of_01:[None 85] disconnected
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:1.77463411881
INFO:forwarding.detect_new:0.939215207046
INFO:openflow.of_01:[None 86] disconnected
INFO:openflow.of_01:[None 85] closed
INFO:openflow.of_01:[00-00-00-00-00-04 77] disconnected
INFO:openflow.of_01:[None 86] closed
INFO:openflow.of_01:[None 87] disconnected
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:4.57083576538
INFO:forwarding.detect_new:0.59990940727
INFO:openflow.of_01:[00-00-00-00-00-04 77] closed
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:3.18317515088
INFO:forwarding.detect_new:0.785530187499
INFO:openflow.of_01:[None 88] disconnected
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:1.76523934979
INFO:forwarding.detect_new:0.939974386325
INFO:openflow.of_01:[None 87] closed
INFO:openflow.of_01:[00-00-00-00-00-01 74] disconnected
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:0.370721813219
INFO:forwarding.detect_new:0.999075706419
INFO:openflow.of_01:[None 88] closed
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:2.84683661205
INFO:forwarding.detect_new:0.827810384773
INFO:openflow.of_01:[None 89] disconnected
INFO:openflow.of_01:[00-00-00-00-00-01 74] closed
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:2.32720134858
INFO:forwarding.detect_new:0.887283233893
INFO:openflow.of_01:[None 90] disconnected
INFO:openflow.of_01:[None 89] closed
INFO:openflow.of_01:[None 91] disconnected
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:1.50641795552
INFO:forwarding.detect_new:0.959067098156
INFO:openflow.of_01:[None 90] closed
INFO:openflow.of_01:[None 91] closed

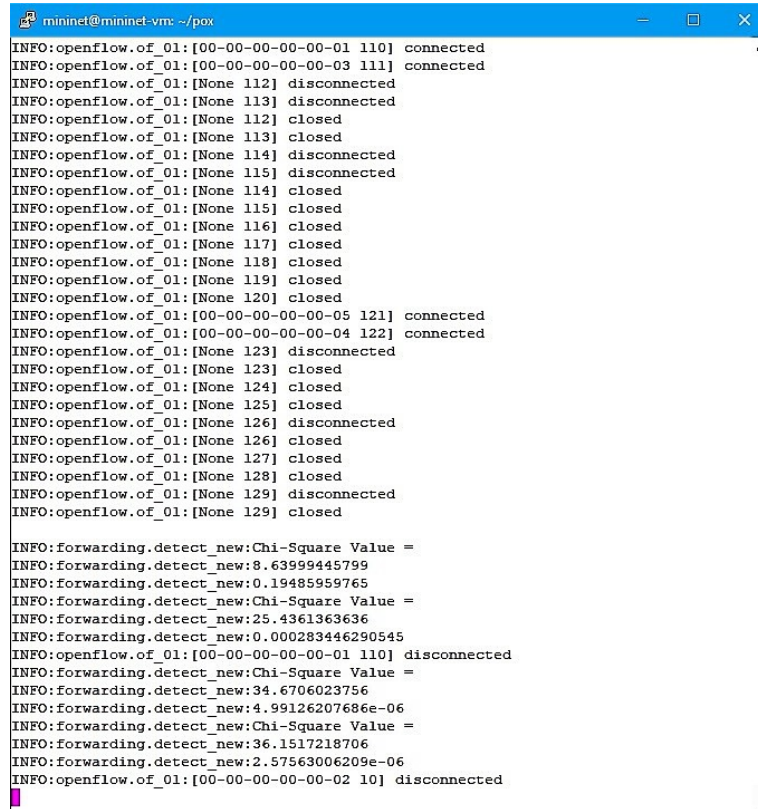
```

Figure 4.5: Chi-square value under normal traffic

- Now repeat the same on the Xterm with the attack.py script and launching the attack on any node of the network.

```
$python attack.py 10.0.0.6
```

You will observe that the p-value has fallen below 0.05 and the χ^2 value has risen significantly.



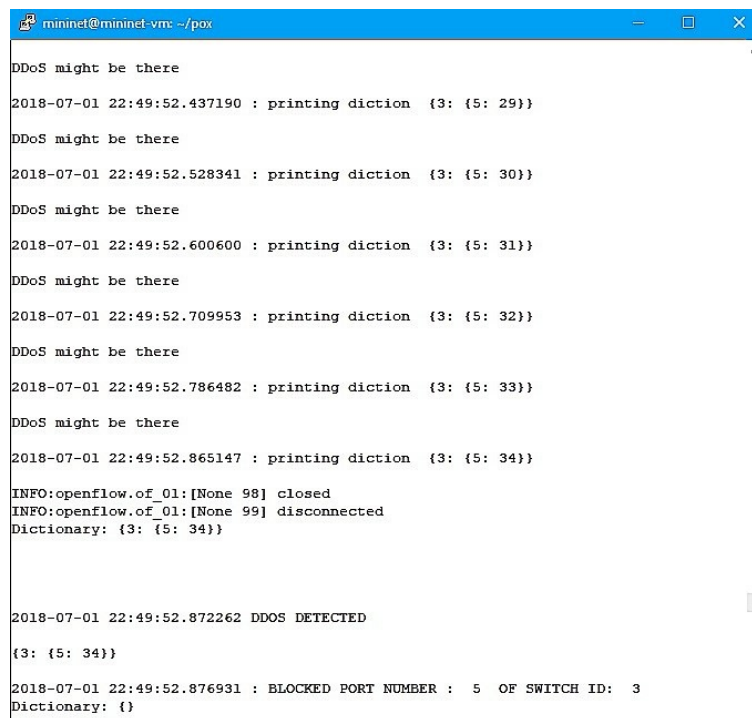
```
mininet@mininet-vm: ~/pox
INFO:openflow.of_01:[00-00-00-00-00-01 110] connected
INFO:openflow.of_01:[00-00-00-00-00-03 111] connected
INFO:openflow.of_01:[None 112] disconnected
INFO:openflow.of_01:[None 113] disconnected
INFO:openflow.of_01:[None 112] closed
INFO:openflow.of_01:[None 113] closed
INFO:openflow.of_01:[None 114] disconnected
INFO:openflow.of_01:[None 115] disconnected
INFO:openflow.of_01:[None 114] closed
INFO:openflow.of_01:[None 115] closed
INFO:openflow.of_01:[None 116] closed
INFO:openflow.of_01:[None 117] closed
INFO:openflow.of_01:[None 118] closed
INFO:openflow.of_01:[None 119] closed
INFO:openflow.of_01:[None 120] closed
INFO:openflow.of_01:[00-00-00-00-00-05 121] connected
INFO:openflow.of_01:[00-00-00-00-00-04 122] connected
INFO:openflow.of_01:[None 123] disconnected
INFO:openflow.of_01:[None 123] closed
INFO:openflow.of_01:[None 124] closed
INFO:openflow.of_01:[None 125] closed
INFO:openflow.of_01:[None 126] disconnected
INFO:openflow.of_01:[None 126] closed
INFO:openflow.of_01:[None 127] closed
INFO:openflow.of_01:[None 128] closed
INFO:openflow.of_01:[None 129] disconnected
INFO:openflow.of_01:[None 129] closed

INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:8.63999445799
INFO:forwarding.detect_new:0.19485959765
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:25.4361363636
INFO:forwarding.detect_new:0.000283446290545
INFO:openflow.of_01:[00-00-00-00-00-01 110] disconnected
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:34.6706023756
INFO:forwarding.detect_new:4.99126207686e-06
INFO:forwarding.detect_new:Chi-Square Value =
INFO:forwarding.detect_new:36.1517218706
INFO:forwarding.detect_new:2.57563006209e-06
INFO:openflow.of_01:[00-00-00-00-00-02 10] disconnected
```

Figure 4.6: Chi-square value under attack traffic

4.3.2 Results

In the experiment, during normal traffic the χ^2 value remains well below the threshold (which according to our experiment is 11.07). Once we launch the attack traffic, the χ^2 test static significantly increases (i.e, > 11.07). This implies that there is a difference between the observed traffic and the expected traffic in the network. Hence, a DDoS is detected and the controller sends a flow modification message to the switch(s) to block the port(s) having a significantly large number of packets being flowing through them. Figure 4.7 shows DDoS attack being detected and flowmod message sent to block ports.



```

mininet@mininet-vms ~/pox
DDoS might be there
2018-07-01 22:49:52.437190 : printing diction {3: {5: 29}}
DDoS might be there
2018-07-01 22:49:52.528341 : printing diction {3: {5: 30}}
DDoS might be there
2018-07-01 22:49:52.600600 : printing diction {3: {5: 31}}
DDoS might be there
2018-07-01 22:49:52.709953 : printing diction {3: {5: 32}}
DDoS might be there
2018-07-01 22:49:52.786482 : printing diction {3: {5: 33}}
DDoS might be there
2018-07-01 22:49:52.865147 : printing diction {3: {5: 34}}
INFO:openflow.of_01:[None 98] closed
INFO:openflow.of_01:[None 99] disconnected
Dictionary: {3: {5: 34}}

2018-07-01 22:49:52.872262 DDOS DETECTED
{3: {5: 34}}
2018-07-01 22:49:52.876931 : BLOCKED PORT NUMBER : 5 OF SWITCH ID: 3
Dictionary: {}

```

Figure 4.7: Switch Ports Blocked on DDoS Detection

Conclusion

As a result of the experiment, Chi-Square Test for DDoS detection turns out to be a effective solution for the detection of anomaly in network traffic in a SDN architecture. This can be a efficient method to adopt at software defined data centers,

DDoS detection and mitigation is all about where the detector is placed in a network. In the case where the detector is placed near the victim, it does not make much difference (keeping aside the fact that network is now safe from the malicious attacks), because it no more communicate with other non-malicious networks, as the attacker sending traffic through the route.

So, it's always better to place the detector near the source of attack so that the resource/bandwidth of the attack path is not consumed by the attacker, even after the attack is detected and blocked. When we place the detector near the source, the attack is blocked and the resource/bandwidth of the route is not used anymore. So, the victim network can communicate with hosts on other network without any difficulties.

References

1. Feinstein, Laura, et al. "Statistical approaches to DDoS attack detection and response." null. IEEE, 2003.
2. Leu, Fang-Yie, and I-Long Lin. "A DoS/DDoS Attack Detection System Using Chi-Square Statistic Approach." Systemics, Cybernetics And Informatics 8.2 (2010).
3. Lim, Sharon, et al. "A SDN-oriented DDoS blocking scheme for botnet-based attacks." Ubiquitous and Future Networks (ICUFN), 2014 Sixth International Conf on. IEEE, 2014.
4. Oshima, Shunsuke, Takuo Nakashima, and Toshinori Sueyoshi. "Comparison of properties between entropy and chi-square based anomaly detection method." Network-Based Information Systems (NBIS), 2011 14th International Conference on. IEEE, 2011.
5. Yan, Qiao, et al. "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 & Tutorials 18.1 (2016): 602-622.
6. Azodolmolky, Siamak. Software Defined Networking with OpenFlow: Get Hands-on with the Platforms and Development Tools Used to Build OpenFlow Network Applications. Packt Pub., 2013.
7. Lim, Anthony. "Security Risks in SDN and Other New Software Issues." RSA Conference. 2015.
8. POX Wiki
<https://openflow.stanford.edu/display/ONL/POX+Wiki>
9. OpenFlow V.1.0 Specifications Whitepaper
<https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf>
10. Open Networking Foundation
<https://www.opennetworking.org/>