Detection and Mitigation of SYN and HTTP flood DDoS attacks in Software Defined Networks

by

Amandeep Singh Dhaliwal

A thesis
presented to Ryerson University
In partial fulfillment of the
requirements for the degree of
Master of Applied Science
In the program of
Computer Networks

Toronto, Ontario, Canada 2017

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Abstract

Detection and Mitigation of SYN and HTTP flood DDoS attacks in Software Defined Networks

Master of Applied Science
in Dept. of Computer Networks
Faculty of Engineering and Architectural Science
Ryerson University, Toronto, 2017

Distributed Denial of Service (DDoS) constitutes major threat to both traditional and SDN networks. An attacker can launch a DDoS attack to exhaust either the controller or other network resources, such as switches, or both. There are different DDoS attacks such as UDP flood, SYN flood, Ping of death, ICMP flood and HTTP flood. Among these, SYN and HTTP flood are the most common attacks these days.

In this thesis, we focus on developing a security scheme to alleviate the DDoS attacks with spoofed and non-spoofed IP addresses in the SDN environment. First we use a simple detection mechanism that utilizes a time series window-based traffic statistic measurement to detect possible SYN flood and/or HTTP flood DDoS attacks. To reduce false positives, further investigation of traffic is done based on valid source IP address scheme and single flow packet scheme to separate legitimate traffic from attack traffic. Once the attack is detected, the security scheme deploys a number of mitigation methods to alleviate the attack. For the SYN flood attack, the mitigation method of Source IP address filtering is used to permit traffic only with valid source IP addresses to enter the network. For HTTP flood attack mitigation, a mitigation method is used to identify the attack sources and discard the traffic from those sources. We test our proposed scheme with other DDoS attacks such as ICMP flood attack and UDP flood attacks. We also compare our scheme with other security schemes found in the literature. The result shows that our proposed scheme can effectively protect controller and other network resources from some common DDoS attacks, and that our scheme allows more legitimate traffic connections with less false positives in comparison with other schemes.

Acknowledgements

A sincere acknowledgement and gratitude to my supervisor Dr. Ngok-Wa-Ma, and my co-supervisor Dr. Xiaoli Li, for their continuous support, patience, motivation, encouragement and time throughout my graduate studies. It was great experience to study and work under their supervision throughout my research work and thesis writing.

Special thanks to the Computer Networks department and the Yeats School of Graduate Studies at Ryerson University for providing the opportunity to complete my Master's degree and support of Ryerson graduate scholarship and Graduate assistantship.

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

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

LIST OF FIGURES 4

LIST OF TABLES 5

ABBREVATIONS 6

Chapter 1 7

1 Introduction 7

1.1 Problem Statement 7

1.2 Research Objectives and Contribution 7

1.3 Thesis Outline 8

Chapter 2 9

2 Background 9

2.1 SDN: DEFINITION, BENEFITS 9

2.2 BENEFITS 10

2.3 OpenFlow Protocol and OpenFlow Switch 11

2.4 SDN Controllers 12

2.5 DDoS attacks 13

2.5.1 SYN attack 13

2.5.2 HTTP flood attack 14

2.5.3 UDP flood attack 15

2.5.4 ICMP flood attack 15

2.6 Background and Related Works 15

2.6.1 Traditional DDoS attack detection and mitigation Approaches 15

2.6.2 SDN-Based DDoS attack detection and mitigation approaches 19

2.7 Chapter Summary 23

Chapter 3 24

3 Introduction 24

3.1 DDoS Attack Detection with Spoofed IP addresses 24

3.1.1 SYN Attack Detection Threshold Measurements 24

3.2 DDoS attack detection with Non-Spoofed IP Addresses 32

3.2.1 HTTP Attack Detection Measurement 32

3.3 Mitigation Methods 36

3.3.1 SYN Flood Attack Mitigation 36

3.3.2 HTTP Flood Attack Mitigation 38

3.4 Organization of the Detection and Mitigation Modules 39

3.5 Chapter summary 41

Chapter 4 42

4 Performance Evaluation and Results 42

4.1 Testbed 42

4.2 Network Setup 42

4.3 Results and Analysis 43

4.3.1 Normal Traffic Pattern 44

4.3.2 SYN Flood Attack Detection and Mitigation 46

4.3.3 HTTP Flood Attack Detection and Mitigation 51

4.3.4 ICMP Flood Attack Detection and Mitigation 53

4.3.5 UDP Flood Attack Detection and Mitigation 55

4.4 False Positive and False Negative Attack Detection 56

4.4.1 SYN Traffic 56

4.4.2 ICMP traffic 59

4.5 Comparison with other Approaches 61

4.5.1 DDoS attack detection comparison 61

4.5.2 DDoS mitigation comparison 64

4.6 Chapter Summary and Discussion 66

Chapter 5 67

5 Conclusion and Future work 67

5.1 Conclusion 67

5.2 Future Work 67

LIST OF FIGURES

Figure 1: SDN architecture [1] 9
Figure 2: OpenFlow Switch 11
Figure 3: OpenFlow 1.0 Flow table entry [1] 12
Figure 4: Packet forwarding with OpenFlow [6] 12
Figure 5: SYN flood attack 14
Figure 6 : Permanent IP Address Database (PAD) 28
Figure 7 : OpenFlow Statistics in Flow Stat Database 30
Figure 8 : SYN Detection Algorithm Flowchart 31
Figure 9 : HTTP flood detection flowchart 35
Figure 10 : Pass flows and Drop flows 37
Figure 11 : Mitigation Algorithm Flowchart 38
Figure 12 : Attack Detection and Mitigation Operation 40
Figure 13 : Network topology 43
Figure 14 : Legitimate traffic before attack 45
Figure 15 : SYN Flood Attack without defense mechanism 47
Figure 16 : Legitimate Traffic under SYN Flood Attack with a defense mechanism 49
Figure 17 : Detection and Mitigation Attack process 50
Figure 18 : Number of packets arriving at the controller 51
Figure 19 : HTTP response time under Normal Traffic and Attack Traffic 52
Figure 20: HTTP Response time with Mitigation Scheme 53
Figure 21 : Bandwidth consumption (Normal traffic, ICMP flood attack w/o mitigation, ICMP flood attack w mitigation) 54
Figure 22 : Number of Flow Rules Installed – OF 1.0 55
Figure 23 : False Positive and False Negative Reports 58
Figure 24: False Positive and False Negative Reports 60
Figure 25: False Positive Reports Traffic Pattern A 63
Figure 26: False Positive Reports Traffic Pattern B 64
Figure 27: Legitimate HTTP connections (Comparison of Mitigation Schemes) 65

LIST OF TABLES

Table 1: SDN vs Traditional Networking [4] 10
Table 2: Normal traffic pattern 44
Table 3: Attack Traffic Pattern 46
Table 4: Mitigation method effectiveness 50
Table 5 : Traffic Pattern A 57
Table 6 : Traffic Pattern B 57
Table 7: FP and FN reports under different traffic patterns 59
Table 8 : ICMP traffic 60
Table 9 FP and FN Reports under both cases 61

ABBREVATIONS

ACK	Acknowledgement
API	Application Program Interface
CUSUM	Cumulative Sum
CPU	Central Processing Unit
DARPA	Defence Advanced Research Projects Agency
DDoS	Distributed denial-service-attack
HTTP	Hypertext Transfer Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
LOIC	Low Orbit Ion Cannon
SDN	Software Defined Networking
SYN	Synchronize Sequence Number
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
OVS	Open vSwitch
ICMP	Internet Control Message Protocol
ONF	Open Networking Foundation

Chapter 1

1 Introduction

1.1 Problem Statement

In Software Defined Networking (SDN), the controller is a single point of failure and a prime target of DDoS attacks. In DDoS attack, a single system, such as the controller, can be targeted by multiple compromised systems sending high volume of traffic. An attacker might use fictitious IP addresses (IP address spoofing) to hide the locations of the attacking sources. During the attack, the switch that receives the incoming attacking packets will usually forward the packets to the controller. If the arrival rate of incoming spoofed packets is high, the processing of these packets may exhaust the controller resources, thus overwhelming the controller and make it unable to process the legitimate traffic. Since the controller represents a single point of failure, this type of attacks may bring down the whole network. SYN flood is one such DDoS attack in which an attacker sends a large number of SYN requests to a target victim thus consuming network bandwidth and/or computing resources [2][3].

Another popular type of attacks is the HTTP flood attack in which an attacker floods a web server with HTTP GET messages, thus makes it unavailable to legitimate users. To launch the HTTP flood attack, the attacking machine must establish a TCP connection with the web server. Consequently, the attacking machine cannot use a spoofed IP address. This characteristic is different from SYN flood attack. In this thesis, we propose two detection and mitigation methods to handle SYN flood attack and HTTP flood attacks separately. It should be noted that the proposed methods that deal with SYN flood attack and HTTP flood attack can also be used to deal with general classes of address-spoofing and non-address-spoofing attacks, respectively.

1.2 Research Objectives and Contribution

This thesis research focuses on Distributed Denial of Service (DDoS) attacks on SDN networks with and without spoofed source IP addresses, in particular, it focuses SYN attack and HTTP flood attacks. Two detection and mitigation methods are proposed by adding a lightweight detection and mitigation mechanism at the controller to protect SDN switches and controller from DDoS attacks. The thesis has following main objectives:

- Show the impact of SYN and HTTP flood DDoS attacks.
- Propose and design lightweight DDoS detection schemes by utilizing statistical measures with high accuracy and less false positives and negatives.
- Propose and design mitigation schemes to efficiently block malicious sources while allowing legitimate traffic to connect to the network.
- Evaluate the effectiveness of the proposed methods against SYN and HTTP flood DDoS attacks.
- Evaluate the performance of the proposed methods against other DDoS attacks such as UDP and ICMP attacks.
- Compare the proposed methods with existing approaches found in the literature.
- Successfully implement both methods using SDN POX controller.
- Show the effectiveness of the solution through simulations based on different traffic patterns and attack scenarios.

1.3 Thesis Outline

This thesis comprises of following chapters:

- Chapter 2 covers an overview of SDN and its architecture, followed by introduction of OpenFlow protocol and OpenFlow Switch. Various DDoS attacks are introduced and methods to deal with these attacks found in the literature are briefly described.
- Chapter 3 proposes and describes the methods used to detect and mitigate SYN and HTTP flood DDoS attacks.
- Chapter 4 presents the implementation of proposed methods, simulation setup, results, analysis and comparison with existing approaches.
- In Chapter 5 provides conclusions and directions for future work.

Chapter 2

2 Background

This chapter covers Software defined networking (SDN) architecture in detail. We will take a look of SDN benefits, SDN-capable switching devices, current implementations, SDN controller, and OpenFlow protocol. Finally, previous work in SDN security will be reviewed.

2.1 SDN: DEFINITION, BENEFITS

Software Defined Networking (SDN) is an emerging technology that simplifies network management by separating network control and forwarding functions. With the separation of control and data planes, network control can be done solely on the centralized control plane. Furthermore, by making the centralized network plane programmable, SDN is well-suited for dynamic and high bandwidth network applications deployed nowadays. OpenFlow protocol is one of the popular control protocols used in the SDN control plane. Figure 1 shows the architecture of SDN with OpenFlow.

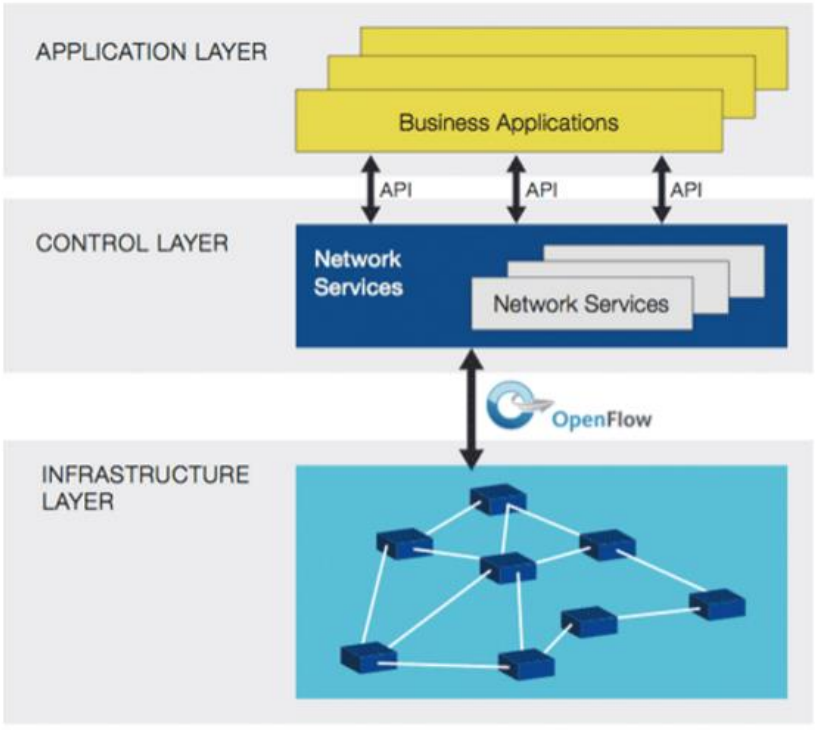


Figure 1: SDN architecture [1]

SDN architecture consists of three layers: application top layer, control middle layer and bottom infrastructure layer. Communication between infrastructure layer and control layer is through Southbound APIs and between application and control layer is through Northbound APIs. The bottom infrastructure layer consists of network devices such as switches that contain flow entries and communicate with controller located in the control layer. Each flow entries in the flow table of the switch contains matching rules and associated actions. When a packet arrives to a switch its header field is matched with flow entry rules, if the packet is matched with one of the existing flow entries in the flow table, the associated action is taken. The action may include forwarding the packet to a specified host/router/switch or discarding of the packet. If no match can be found the switch sends the packet to the controller for further processing. The Controller processes the packet header, identifies the flow the packet belongs to and then instructs the switch to install an entry in the flow table to handle the packet and the subsequent packets of the flow.

2.2 BENEFITS

Traditional network configuration involves a specific level of manual processing, which sometimes is tedious and error prone. Also, a significant effort is required to troubleshoot network misconfigurations. With the decoupling of control and data plane, SDN offers potential benefits of improved performance, enhanced configuration and encouraged innovation of network architecture and operations [4].

In SDN, switching devices can be configured and controlled externally from a single point, via a software controller. In addition, SDN improves network performance globally by exploiting the flow based paradigm, logically centralized intelligence and automation. An SDN offers cloud technology solutions such as virtual networks in a multi-tenant clouds based on OpenFlow standards. Table 1 shows comparisons between SDN and traditional networking.

	SDN	**Traditional Networking**
Features	Decoupled control and data plane, and programmability	Complex network control
Configuration	With centralized validation, automated configuration	Manual configuration, error prone and time-consuming
Performance	Dynamic global control	Static configuration, limited info.
Innovation	Easy software implementation for new ideas, quick deployment using software upgrade	Difficult hardware implementation for new ideas, limited testing environment, long

Table 1: SDN vs Traditional Networking [4]

2.3 OpenFlow Protocol and OpenFlow Switch

OpenFlow is a standard defined by the Open Networking Foundation (ONF) for southbound SDN control. The OpenFlow protocol describes the interaction between SDN/OpenFlow controller and OpenFlow-compliant switches. The OpenFlow protocol allows the SDN/OpenFlow controller to install flows to and gather traffic statistic from switch/routers as shown in Figure 2 below.

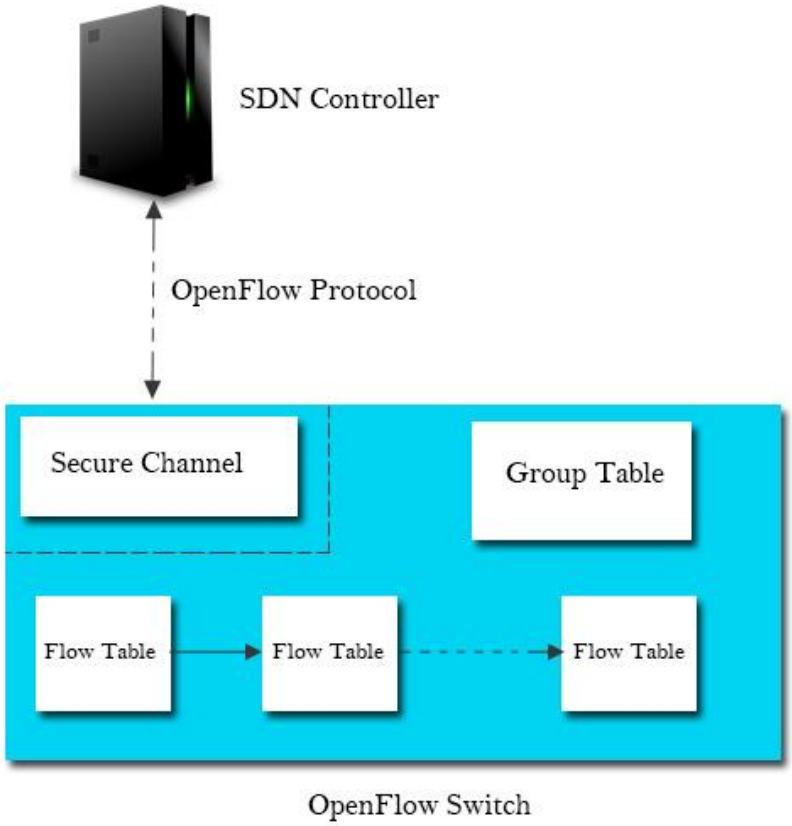


Figure 2: OpenFlow Switch

An OpenFlow Switch consists of: (1) A Flow Table, with an action associated with each flow entry, to tell the switch how to process the flow, (2) A Secure Channel, that connects switch to a remote controller, allowing exchange of packets and commands between switch and controller, (3) The OpenFlow Protocol, provides a standard way for communication between controller and switch [5]. The flow table has flow table entries, each of which consists of: (1) A Header field, which defines the flow, (2) Counters or Statistics, which keep track of number of bytes, packets, duration for each flow, (3) Actions, which defines how packets of that flow should be processed, as illustrated in Figure 3.

Figure 3: OpenFlow 1.0 Flow table entry [1]

The OpenFlow switch stores flows in the Ternary Content-Addressable Memory (TCAM) which is limited by the size. Any packets entering the switch, the switch parses the packet header, and match against flows in the table. If there is a match, the action associated with the flow is applied and the flow statistics for the entry are updated accordingly. Otherwise, if there are no matching flows, the packet will be sent to the controller via Packet_In message. The controller process the packet, identifies the correct action and sends back a Packet_Out message to instruct the switch to take the required action. Figure 4 shows the basic packet forwarding mechanism:

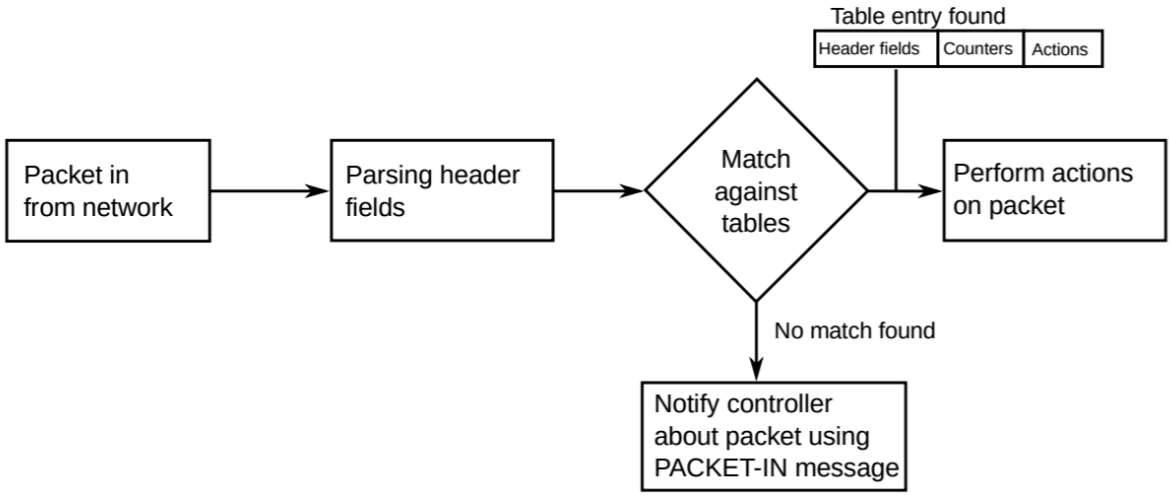


Figure 4: Packet forwarding with OpenFlow [6]

2.4 SDN Controllers

Controller is the most important component in the SDN networks. An SDN controller contains variety of modules that can perform different network tasks such as network controlling, network monitoring, collecting network status and updates packet forwarding rules to switching devices through south-bound interface, receiving policies from SDN applications and providing a synchronized global view through north-bound interface [4]. The controllers are software-based. Today, a number of open source controller implementations are written with different programming languages such as Java, Python and C++.

In this thesis we use POX controller with OpenFlow version 1.0, which is a suitable platform for SDN research. POX is written in Python language. POX officially supports Linux, Windows and Mac OS operating systems and it is easy to install and run. A POX installation includes a number of components having different core functionality and features. The proposed algorithm in this thesis is implemented based on the *l2_multi* module of POX. L2_multi module is sort of a learning switch, and works with the *openflow.discovery* component that uses the Link Layer Discovery Protocol to discover the network topology.

2.5 DDoS attacks

Distributed Denial of Service (DDoS) attacks are becoming an increasingly frequent problem for big organizations to individual internet users. The DDoS attack is an attempt to consume network or host resources by sending large amount of malicious traffic. There are various types of DDoS attacks such as, SYN, UDP, ICMP and HTTP flood. In Q2 2011, Kaspersky's labs botnet monitoring system intercepted over 20,000 web-borne commands to attack on different sites and reported HTTP flood is the most popular (88.9%) method of attacking a website, while SYN flood attacks are the second most popular type of attack (5.4%). In another report by Kaspersky Labs [46], there were about 75.33% SYN attacks, 2.19% UDP attacks, 10.71% HTTP and 1.41% ICMP DDoS attacks reported in 2016. It is clear that SYN and HTTP DDoS attacks are the highly reported ones. Therefore, we focused on SYN and HTTP flood attacks in this thesis and proposed two methods to deal with such attacks. We will also test the proposed methods against UDP and ICMP attacks. These attacks will be briefly described below:

2.5.1 SYN attack

TCP uses 3 way handshake process in which a client or intruder sends connection initiate request by sending SYN packet to server, and server or target victim replies back by SYN+ACK packet. The victim/server has created half-open TCP connection at this point and is waiting for an ACK reply from client or attacker to complete the 3-way handshaking process. The half-connection is kept in the connection table of the server until timeout. Since the size of the table is restricted, inundating half open connections established in a brief time frame and not being able to complete the connections expeditiously will saturate the connection table, incapacitating the establishment of new legitimate TCP connections. The SYN flood attack happens when the attacker sends a large number of SYN packets with spoofed source IP addresses to overflow the connection table with

massive half connections. This attack strategy will hamper the session establishment of client or legitimate user leading to denial of service. In some occasion, the server may even malfunction or crash.

In the SDN environment, the attack also affects the controller and the switches. As explained in section 1.1, to attack a SDN networks, an attacker can send large number of malicious packets or faked traffic to a host or server. In the case an attacker makes use of spoofed IP addresses to hide identity, the switch will not find a match and forward the spoofed packet to the controller which will then normally install a flow in a switch to direct the spoofed packet to the target destination. A large number of spoofed packets will create a large number of flows in the switch, thus consume the switch TCAM capacity [9]. Meanwhile by receiving a large number of legitimate and malicious packets, the controller may eventually run out of resources and breakdown. Figure 5 shows the principle of SYN flood attack.

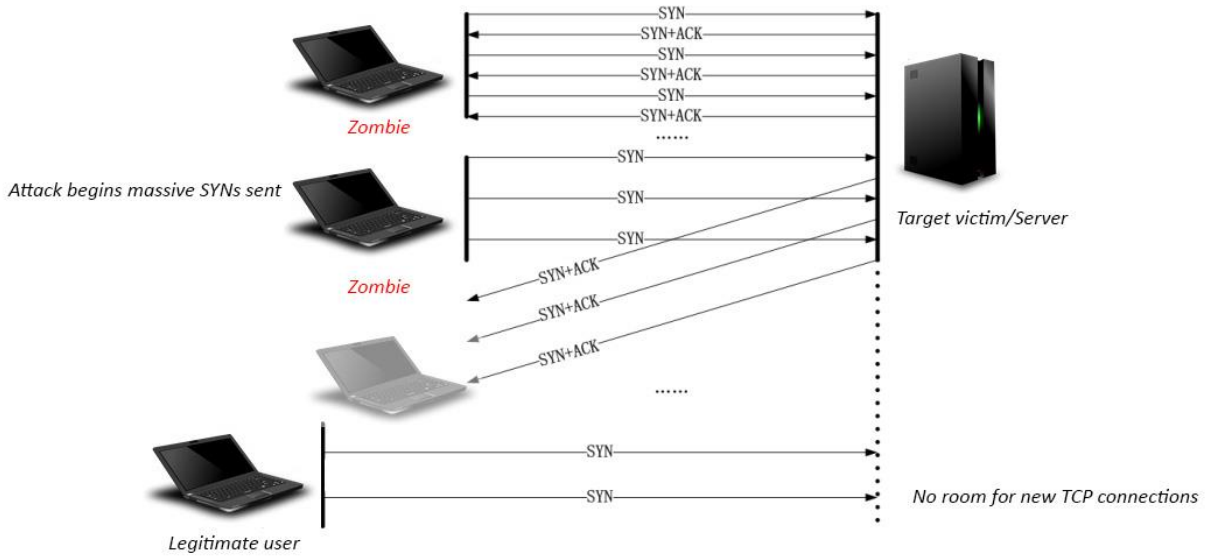


Figure 5: SYN flood attack

2.5.2 HTTP flood attack

An HTTP flood attack is an application layer attack that targets web server and applications. In this type of attack an attacker first establish a TCP connection with the web server. After the connection, the attacker floods the web server with HTTP GET and POST requests [8]. Here, the attacker intends to deplete the bandwidth or significant amount of the server's resources. Note that

in order to establish a TCP connection, the attacker cannot use the spoofed IP source addresses. Consequently, this attack does not affect the controller performance.

2.5.3 UDP flood attack

In this type of DDoS attack, the victim server is flooded with UDP packets with spoofed source IP addresses at a very high rate thus consuming all the available bandwidth and resources in the network. This type of attacks is similar to the SYN attack. Thus, the method we proposed to deal with SYN attack can also be used for this type of attacks.

2.5.4 ICMP flood attack

Like UDP flood attack, this DDoS attack floods the target server with tremendous amount of spoofed or un-spoofed ICMP Echo Request (ping) packets at a fast rate without waiting for any replies. The goal of this attack is to consume network bandwidth. Since, the victim server tries to respond large number of ICMP requests with ICMP Echo Reply packets, resulting in overall system slowdown [41].

2.6 Background and Related Works

In this section, we discuss existing methods of DDoS attack detection and mitigation. We present a brief overview of traditional network as well as SDN-based DDoS attack detection and mitigation techniques with their limitations.

2.6.1 Traditional DDoS attack detection and mitigation Approaches

Distributed Denial of Service (DDoS) is a type of DoS attack and is a common threat to internet today in which multiple compromised systems launch attack on a single system specially server resulting in temporarily interrupting or suspending of services. One such attack is SYN Flood attack. W. Chen [10], H. Wang [11] use a statistical anomaly detection method called Cumulative Sum (CUSUM) for detecting SYN Flood. W. Chen proposes a simple mechanism that can detect SYN flood attack under different IP spoofing types such as random, subnet and fixed spoofing. To differentiate between complete and incomplete TCP handshakes, this method uses change point detection method (CUSUM) and storage efficient data structure using Bloom Filter [12] which requires a fixed-length table for recording traffic information. After some traffic information is extracted and stored in Bloom filter, a change point detection method, CUSUM, is applied to detect

abrupt changes in the traffic. When malicious events are detected, a hash table for the source IP characteristics is analyzed to classify them further into random spoofing, subnet spoofing and fixed spoofing. Similar method is proposed by H.Wang [11] in which Cumulative Sum (CUSUM) is used for the detection of SYN flood attack. Instead of monitoring traffic at front end such as firewall or proxy, detection of SYN flood attack is done at leaf routers that connect end hosts to internet. It is an instance of a non-parametric CUSUM, Sequential Change Point Detection method and also based on TCP Synchronized-Finish (SYN-FIN) behaviour. In order to differentiate normal TCP traffic from malicious, a TCP flow with three types of SYN pairs (SYN, FIN), (SYN/ACK, FIN), and (SYN, $RST_{active)}$ are considered as normal. The difference between the number of SYNs and FINs (RSTs) is used for SYN flood attack detection. This mechanism not only helps in finding out the flooding sources location, but also sets alarm on detection of SYN flooding attacks. Simulation experiments show that the method is both effective and efficient in defending against SYN flood attacks under different IP spoofing. However, the inadequacy of SYN-FIN pair's scheme is, if the attacker is aware of the presence of such a detection system, it can fail the detection mechanism by flooding a mixture of SYNs and FINs (RSTs) packets.

In [13], Se Gi Hong et al. proposed a signalling architecture for network traffic authorization, called Permission Based Sending (PBS) that requires a sender to get authorization from the receiver before sending a packet flow. The permission states are installed within the end hosts and routers, called PBS nodes, for subsequent data flow. However, the proposal requires changes in both the network nodes and end hosts. Another mechanism is to detect the mismatch between bidirectional packets [14]. At the point when there is no attack on the server, packet advent rate for both directions are virtually the same or of the same order. On the off chance that the packet arrival rate for one direction is much higher than that for the other direction, the traffic in the high rate direction may incorporate attack packets. The victim server uses backlog queue which is a memory structure to handle incoming packets with a SYN flag set. In this method, however, the attack cannot be detected by the router until the server has a vacancy in backlog queue the server uses to store can reply SYN/ACK packets for spoofed SYN packets.

The time variation of arrival traffic is taken into account in [15] for accurately detecting SYN flood traffic. Author presents attack detection method based on statistics of SYN arrival rate by periodically measuring SYN packets arrival rates and calculating the variance and average of those

rates. The Researcher illustrates how to model arrival rate of normal TCP SYN packets as normal distribution by introducing a new attack detection method. The author analyze data traffics of the Osaka University and describe the SYN arrival rates of normal traffic normally distributed, while the distribution of the SYN arrival rates of network traffic with attack traffic doesn't follow normal distribution. An attack is identify by observing the difference between the distribution of SYN arrival rates and a normal distribution. However, the author used a fixed threshold in his work.

In [16] author shows the effectiveness of TCP control packet counts in TCP SYN scanning detection on a single machine. To capture behavioural characteristics of TCP control packets a neural network is trained for port scan data. The method is based on counting occurrence of half open and failed connections. For modelling of TCP control behaviour Multilayer Feed Forward Network with Backpropagation algorithm is used. The artificial network aggregates counts of TCP control packets accurately.

The authors of [17] use Linear prediction (LP) analysis for SYN flooding DDoS attack detection. The proposed method uses exponential backoff property of TCP. To analyse network traffic this Linear prediction (LP) analysis is very useful as well as by using appropriate parameters this method can be used to detect TCP, UDP and ICMP flooding attacks too. However, this LP based detection system approach is only efficient for low intensity SYN flood attacks, whereas in case of high intensity DDoS attacks this solution introduces high overhead computation.

In [18] authors have proposed a methodology in which attack detectors are installed in the network. The traffic flow pattern is monitored by each router and if any anomaly is detected by a router in traffic pattern, it will raise alarm that will propagate to all routers through which attack flows in network.

Lersak Limwiwatkul et al. [19] distinguish between normal and abnormal traffic and discover DDoS attacking signature based on analysing TCP/IP packet header against well-defined conditions and rules. The proposed model takes input network packet and matches it with designed rules and conditions, and if not matched, it declines the connection.

The primary objective of post attack investigation is to dissect an attack, discover patterns in it to sustain the IDS with, and on the other hand, to follow back the attacker [20]. Song et al. exhibit in [21] a technique to follow back a spoofed IP address to its authentic source. Nonetheless, Zhang

et al. [20] demonstrate that it is definitely not practical to follow back immensely colossal Botnets at the moment of the attack. One reason is that immensely colossal, cutting edge Botnets comprises of thousands of agents; and the second one is that the worldwide web is too enormous that all administrators can work together to exchange trace back information.

A traditional blackholing methodology can be utilized to block traffic which is bound to a specific victim [22]. In this way, all packets from a specific IP address that causes high activity are directed to a supposed null route. Contingent upon the geographical location of the source address, the traffic could hypothetically additionally begin from legitimate request, e.g. some advertisement. Having this as a main priority, one noteworthy disadvantage of this methodology is that malicious as well as legitimate traffic also filtered out. In [23] authors propose a method with a window of 0.1 seconds and three levels of threshold concerned with evading false positives and false negatives in the network. In any case, the authors themselves specify, the strategy is tedious and utilizes more resources.

Blazek et al. [24] propose a statistical technique to detect a change in the network traffic. The technique is based on change point detection theory and consists of two methods: batch sequential and adaptive sequential. Both the methods utilize statistical analysis of training data to achieve a fixed rate of false alarms. The technique uses TCP and UDP type traffic in change detection modelling. The authors claim that their technique detect the attacks with small delay and the methods are self –learning. However, the high computational complexity is the main drawback of this technique.

Tao Peng et al. [25] proposed History-based IP Filtering (HIF) scheme to defend against highly distributed DDoS attacks. In this scheme, an edge router keeps a history of all legitimate IP addresses of the previous successful network connections in an IP address database. In case of a DDoS attack, this history is used to filter out attack packets by discarding packets whose source address does not appear in the database. The main idea of this scheme is to record all the IP addresses of the previous successful connections in order to build an IP address database. The author analyzed data traffics of the University of Auckland network and verified nearly 90% of all IP addresses observed under normal traffic conditions have sent a request before. The author also indicated any TCP flow with less than 3 packets is an indication of an anomalous flow.

2.6.2 SDN-Based DDoS attack detection and mitigation approaches

In [26], the researchers develop the current OpenFlow information plane with two new modules. The first is a connection relocation module which empowers the information plane to shield the control plane from DDoS SYN flood attacks. The second is an activating trigger module which is a technique for rapidly reacting to the network threat. An embedded layer 2-4 DDoS mitigation solution has been made in [27]. Similar to trigger module, rate-constrained flow rules are programmed in the switch for the automatically detected layer 2-4 flows. The inadequacy of this type of methodology is that diverse hosts may need to set distinctive values and based on the traffic character of the network threshold need to be calculated. E.g. thresholds such as allowable bandwidth and observation interval. Some of the large flows whose bandwidth requirements can be high and/or some bursty flows with long duration of in activity which requires an observation interval greater than default value. So these thresholds may need to be set differently for DDoS attack determination.

Tommy Chin Jr. et al. [28] discuss a novel attack detection approach that inspects network packets on demand. They utilized a system of sensory Monitors distributed over a network and attack Correlators residing with SDN controllers at OVS. A Monitor quickly detect anomalies in network traffic. An attack Correlator alerted by the Monitor, collects more information about suspicious traffic with the help of controller. Followed by verification of high accuracy, this approach issues an alert when potential threat is detected. This approach is tested on the detection and mitigation of SYN flood attacks. In their design Client is a computer node that either generates regular traffic or attack traffic of malicious intent. Monitor node is a sensing node that sends alerts or other additional information to correlator (controller). The correlator verifies the attack by correlating the alerts with information maintained in OVS switches and identifies the ports and computer that generated the attack and drop the flow for mitigation. A control threshold is set after empirical analysis of normal traffic. If the threshold is crossed, the monitor raises an alert and communicates to correlator that will then take the appropriate action. However, this mechanism doesn't consider external DDoS attack with non-spoofed addresses and can only detect large scale network attacks with high volume of traffic.

The authors of [29] propose the technique well known as AVANT-GUARD, an extension of OpenFlow. First, author introduced a method called connection migration, an extension to the

OpenFlow data plane, which dramatically reduces the interactions between the data and control planes to provide prevention against DoS attack on the controller. Second, a technique called an actuating trigger proposed by the authors that addresses the responsiveness challenge. The actuating trigger activates a flow rule under some predefined conditions to help the control plane manage network flows without delays. In any case, the authors themselves acknowledge that the technique cannot defend against application layer DDoS attacks or attacks based on HTTP, ICMP and UDP protocols. This method is also not efficient and feasible, as every switch in the network need to be Avant guard enabled.

LineSwitch [30] is an efficient and effective solution to tackle control plane buffer saturation attack and overcomes the limitations of AVANT-GUARD. LineSwitch proxies all incoming TCP connections from a given IP until one is completed by using SYN proxy technique. However, for mitigating buffer saturation attack, LineSwitch employs probabilistic proxying and blacklisting of network traffic. From a given IP LineSwitch proxies all incoming TCP connections until one is completed. This decreases with memory size needed for storing the state of ongoing connections. Like AVANT-GUARD, when applying LineSwitch mechanism, it is required to upgrade all switches.

Nhu et al. [31] propose a method to protect the network against DDoS attacks in SDN environment. The method is based on the proposed method described by the author in [25]. The author used the proposed scheme in SDN environment. The method differentiate the attack sources from legitimate sources based on number of connections and number of packets. The author describes, the characteristics of DDoS addresses that initiate less than k connections and generate less than n packets per connection. However, this method is not efficient when the amount of attack traffic is very huge and not suitable for real-time traffic applications.

Mehdi et al. [32] estimate the benign traffic distribution by using maximum entropy estimation to detect network anomalies in a home networks, a small-office/home-office network. Traffic is divided into a number of packet classes and maximum entropy estimation is used to develop a baseline benign distribution for each class. Packet classes are derived from protocols and port numbers. The authors implemented traffic anomaly detection algorithms using OpenFlow switches and NOX as a controller. However, the authors only focused on a home environment and used the

low rate network traffic to do the experiments. Wang et al. [33] proposed an entropy based lightweight DDoS flooding attack detection method to determine the randomness of the flow data and point out potential DDoS attacks. In that, DDoS detection module runs in every edge switch thereby reducing the heavy communication overhead between switch and controller without the frequently flow collection. The detection module running in every OpenFlow edge switch monitors the input flow and does analysis on the collected data. An alert information is sent to the controller once a DDoS flooding attack is confirmed. Entropy based techniques suffer with a few limitations such as, relevant information about the distribution of analyzed feature is lost, resulting in masking of anomaly effects. Additionally, the distinctive distributions with the same amount of uncertainty cannot be distinguished by entropy values [34].

Braga et al. [35] proposed Self-organizing map (SOM) flow-based DDoS attack detection method. This method is divided into three modules: (1) The Flow collector is responsible for collecting flow entries from flow tables of OpenFlow switches, (2) The Feature Extractor extracts features of collected flows to be passed to the classifier. These features are Average of Bytes per flow, Average of Packets per flow, Average of Duration per flow, Percentage of Pair-flows, Growth of Single flows and Growth of Different Ports. (3) The classifier analyze the collected flow stats by using some statistical or learning method to determine whether given traffic pattern belongs to DDoS flooding attack or to legitimate traffic. This method shows high rate detection with low rate false alarms. However, the author didn't mention any mitigation strategy.

Dillon et al. [36] proposed a DDoS mitigation solution that utilizes OpenFlow. The authors have presented a three step solution for DDoS attack detection and mitigation. In the first step, anomaly detection algorithm computes standard deviation of packet/byte rate for a certain interval by utilizing OpenFlow statistics to detect anomalies in the network traffic. Identification of sources that are performing an attack is done in the second step through the analysis of packet samples. Third step drops identified attackers with OpenFlow drop flows. The performance of this method typically dependent upon the dataset used in this work.

NM Sahri et al. [37] presented *CAuth*, a client-server based novel mechanism that block the spoofing packet while authenticate the legitimate query. This method detect spoofed DNS query

packets against DNS servers by blocking all queries from clients that failed to reply back the authentication packet sent previously by the server controller.

NICE [38] is a multiphase distributed vulnerability intrusion detection system to prevent vulnerable virtual machines from being compromised in the cloud environment. The proposed mechanism, NICE is built on attack graph based analytical models for attack detection and prevention. Multiple attributes are considered for attack graph generation including vulnerability information, configuration information, cloud system information, virtual network topology. After some VMs are identified as suspicious or vulnerabilities are discovered, countermeasures including traffic isolation, port blocking, network reconfiguration and traffic redirection are taken. Commonly used mitigation techniques including blocking ports, dropping packets are used in SDN [38, 28]. However, in any case legitimate port or node is compromised, it is completely blocked resulting in dropping all the legitimate traffic coming from that port [34].

Lim at el. [39] deployed the traffic redirection technique and proposed a blocking scheme for botnet-based attacks. In this scheme, the traffic redirection technique forwards the traffic to new IP address after the attack is detected on an existing server. Additionally, a CAPTCHA mode is implemented on the server side under attack such that service to legitimate users is not effected. The proposed scheme is implemented as a SDN application that runs on POX SDN controller. However, this technique increases the overall complexity and processing time of the network traffic. In [40] authors have proposed FlowFence, a congestion avoidance mechanism system for mitigating DDoS attacks on SDN. The proposed FlowFence architecture is composed of routers that monitor the average occupation of their interfaces and to detect congestion and an SDN controller that coordinates bandwidth assignment usage on the congested interfaces. A flow is classified as well behaved flow, if the flow transmitted bit-rate is less than an estimated fair share. A fair share, is defined as the ratio between the link capacity and the number of flows using that link. Flows exceeding the fair usage bandwidth consumption are penalized. This attack mitigation scheme by controlling bandwidth at each router interface prevents user's starvation. However, legitimate flows that do not adhere to network usage policies may also suffer.

2.7 Chapter Summary

In this chapter, we first introduced SDN. We also studied diverse papers describing strategies of detection and mitigation against DDoS attacks on traditional networks as well as the SDN based networks. The above discussion has highlighted that most of the DDoS detection and mitigation techniques has been used in traditional networking. In the recent years, traditional networking is by and large superseded by Software Defined Networking (SDN) and therefore number of challenges need to be addressed to mitigate DDoS in SDN based networks. Although similar works are found in [31] and [36], they are not suitable for real world scenarios and required complex configuration [39]. Additionally, most of the techniques [34, 35] are limited to detection only without offering mitigation. There is still no single solution for dealing with different kinds of DDoS attacks.

Therefore, in this research our goal is to analyse Software Defined Networks (SDN) from the viewpoint of Distributed Denial of Service Attacks (DDoS). We develop an effective comprehensive solution that can detect and defend against different types of DDoS attacks more effectively with less false positive and negative alarms.

Chapter 3

3 Introduction

This chapter is divided into two parts. The first part explains the statistical method that is applied to detect DDoS attack with spoofed IP addresses e.g. SYN attack and DDoS attack with non-spoofed IP addresses e.g. HTTP flood attack. The second part describes mitigation methods that can be used to protect the controller and network switches after an attack is detected. For detection and mitigation, a set of lightweight code is added to the controller. The main objective of our study is to find a solution to detect and mitigate DDoS attacks in SDN networks before the attack overwhelms the controller.

3.1 DDoS Attack Detection with Spoofed IP addresses

Identifying DDoS attacks include first knowing normal profile of the traffic pattern and then to detect deviations from this normal profile. In this section, the proposed detection measurements are discussed to find abnormal traffic pattern associated with SYN flood attack with spoofed IP addresses.

3.1.1 SYN Attack Detection Threshold Measurements

In the SYN flood attack detection mode, the detection mechanism identifies anomaly traffic patterns as the signs of attacks. The detection mechanism is divided into three attack detection phases. The first attack detection phase utilizes a time series window-based traffic statistic measurement. The detection algorithm continuously monitors the rate of SYN packets arriving in each fixed window of size Δw, and compares it to a threshold at the end of each time window. The detection algorithm quickly raises a warning when the rate of SYN packets exceeds the threshold. However, a surge of legitimate traffic can also increase the rate of SYN packets arriving at the controller. Thus, in the second attack detection phase, further analysis is done by comparing the source IP addresses of all the flows to a database that contains valid source IP addresses (We will explain the definition of valid source address in the subsequent section). Traffic flows that have the match are considered legitimate. After that, the controller proceeds to the third phase where it computes the ratio of single-packet flows to the total number of flows and compares this ratio to a threshold. The idea of using single packet flows for attack detection came from [31], in which the

author observed that flows that contain less than 3 packets are anomalous, thus, could be generated by the DDoS attack. In our detection method, flows that contain single packet are considered anomalous because in DDoS attacks with spoofed addresses, such as SYN flood attacks, only generate one packet per flow. If for three consecutive windows, the percentage ratio is over the threshold, SYN attack is declared and controller will take action accordingly.

There are two essential parameters to DDoS detection: Window size Δw and thresholds. In this thesis, the window size is fixed. The size of monitoring window affects the detection algorithm performance. If the window size is too large, the system will be slow to respond to attacks. If the window size is too small there will not be enough data samples collected to analyse the traffic pattern accurately. Additionally, it also uses up more bandwidth and requires more processing resources. In this thesis, a window size of 3 seconds is chosen. This value seems to provide a good compromise. .

The proposed detection algorithm requires two adaptive thresholds. Instead of using predefined threshold values, the detection algorithm adaptively calculates the threshold values. These threshold parameters can be tuned by the network administrator to adapt to changing traffic conditions in the network. In order to make the thresholds to be adaptive to the traffic characteristics, we will use the mean and standard deviation of the traffic parameters to derive the appropriate thresholds. To calculate the mean and standard deviation, the controller collects the specific traffic parameters in each window and use Exponential Weighted Moving Average (EWMA) and Exponential Weighted Moving Standard Deviation (EWMSTD) to compute them.

In the first phase of detection, the detection algorithm continuously monitors the number of SYN packets arriving at the controller in a three-second window and count the number of SYN packets. Let X_n be the number of SYN packets arrived in the n^{th} window, then the algorithm updates the mean and standard deviation at the end of the n^{th} window using following equations:

$$Mean = \text{EWMA} = \bar{X}_n = \bar{X}_{n-1} + \alpha(X_n - \bar{X}_{n-1}) \qquad (3.1)$$

$$Std.\ Dev. = \text{EWMSTD} = S_n = \sqrt{\alpha * (X_n - \bar{X}_{n-1})^2 + (1-\alpha) * (S_{n-1})^2} \qquad (3.2)$$

where the tuning parameter $0 < \alpha \leq 1$, is a constant smoothing factor. The choice of α value has a significant impact on the detection performance. The value of $\alpha = 1$ gives more weight to recent data and less weight to older data; a small value of α gives more weight to older data. In this thesis we set the value of α to 0.5, which gives an equal weight to the current and the past data. $\bar{X}_n$ and S_n are the mean and the standard deviation calculated at the n^{th} window, respectively.

The dynamic threshold δ_n calculated in the n^{th} window is set according to equation (3.3):

$$\delta_n = \bar{X}_n + kS_n, \qquad (3.3)$$

Where tuning parameter k is a constant. The choice of k represents the trade-off between false positive and false negative. Larger value of k increases the rate of false negative while smaller value of k increases the rate of false positive. In this thesis, k is set to 1. We choose a small value because this is used in the first phase of detection. Any false positives introduced in this stage may be corrected by the later phases of the detection. By choosing a smaller value, on the other hand, the algorithm can detect the attack earlier and cut down the false negative rate.

In order to derive second threshold that the detection algorithm uses in the third phase of detection, the detection system looks for flows with single data packets. We define a parameter Pr, as the ratio of the number of single-packet flows to the total number of flows. Let C_{Single} be the count of single-packet flows and C_{Total} the total count of flows. Then, the percentage ratio (Pr) is calculated using equation 3.4:

$$Pr = \left(\frac{C_{Single}}{C_{Total}}\right) * 100 \qquad (3.4)$$

The parameter Pr, is computed at the end of each time window. Using Pr the detection algorithm calculate mean and standard deviation to derive an adaptive second threshold. Let Pr_n be the percentage ratio of single-packet flows in the n^{th} time window, then the algorithm updates the mean and standard deviation at the end of the n^{th} window using following equations:

$$Mean = \overline{Pr}_n = \overline{Pr}_{n-1} + \alpha(Pr_n - \overline{Pr}_{n-1}) \qquad (3.5)$$

$$Std.\ Dev. = Sr_n = \sqrt{\alpha * (Pr_n - \overline{Pr}_{n-1})^2 + (1-\alpha) * \left(Sr_{n-1prev}\right)^2} \qquad (3.6)$$

In the above equations (3.5) and (3.6), $\overline{Pr_n}$ and Sr_n are the mean and standard deviation, respectively.

The second threshold ϕ_n computed in the n^{th} window is set according to following equation (3.7):

$$\phi_n = \overline{Pr_n} + Sr_n \qquad (3.7)$$

Let us discuss in detail the three attack detection phases.

3.1.1.1 First Attack Detection Phase

Let X_n be the number of SYN packets received by the controller in the n^{th} time window. For every received SYN packet, a counter C_{count} will be incremented by one. Let $\bar{X}_n$ and S_n be the exponential mean and standard deviation as defined in equation 3.3 and 3.2. At the end of the n^{th} time window, $\bar{X}_n$ and S_n are computed. By updating $\bar{X}_n$ and S_n after each time window, we can track the rate of traffic changes over time and set the appropriate threshold. For attack detection, the number of SYN packets (X_n) in each window is compared with the threshold, $\delta_{n-1.}$ An attack warning is generated if

$$X_n > \delta_{n-1}.$$

The detection algorithm then goes into the second attack detection phase discussed next for further analysis. Note that if the attack is confirmed after the third phase of the detection, δ_n will not be updated and δ_{n-1} will be continuously used as the threshold during the rest of the attack period.

3.1.1.2 Second Attack Detection Phase

This is the second phase in SYN detection algorithm. This phase is triggered if the first phase issues an alert of potential attack (i.e.$X_n > \delta_{n-1}$)). The controller does further analysis of traffic based on valid IP address scheme.

For this, an IP Monitor module is added in the POX controller. This module stores the source IP address of each IP packet received by the controller into a Temporary IP address Database (TAD). After each time window (Δw), if the traffic rate is normal, the temporary IP addresses are moved

from temporary IP address database (TAD) to a permanent IP address Database (PAD). The IP Monitor module keeps the PAD updated by adding new legitimate IP addresses and deleting expired IP addresses. Every IP address stored in the PAD has a time stamp. The time stamp is used to remove the expired IP addresses to keep the information in the database fresh. In our experiments, we set the expiration time to 30 seconds.

When the second phase is triggered, the controller checks if the source IP address of a single-packet traffic flows matches one of the IP addresses in PAD database. An IP address is considered to be valid, if there is a match. This matching scheme can be used to match exact IP addresses or subnet addresses (e.g. 24-bit mask). In this thesis, we use the exact match approach. This scheme helps the controller to add valid flows to the network even during the DDoS attack as described in section 3.4. A snapshot of PAD from our simulation is shown in figure 6.

```
mysql> select * from permanentipaddress;
+------+----------------+---------------------+
| P_Id | IPADDRESS      | Datetime            |
+------+----------------+---------------------+
|   25 | 34.10.20.1     | 2017-06-26 22:49:08 |
|   26 | 24.10.20.1     | 2017-06-26 22:49:13 |
|   27 | 25.10.20.1     | 2017-06-26 22:49:21 |
|   28 | 161.110.120.1  | 2017-06-26 22:49:25 |
|   29 | 66.10.20.1     | 2017-06-26 22:49:25 |
|   30 | 27.10.20.1     | 2017-06-26 22:49:25 |
|   31 | 37.10.20.1     | 2017-06-26 22:49:25 |
|   32 | 155.110.120.1  | 2017-06-26 22:49:25 |
|   33 | 28.10.20.1     | 2017-06-26 22:49:25 |
|   34 | 64.10.20.1     | 2017-06-26 22:49:25 |
|   35 | 26.10.20.1     | 2017-06-26 22:49:25 |
|   36 | 38.200.45.185  | 2017-06-26 22:49:31 |
|   37 | 197.137.94.97  | 2017-06-26 22:49:32 |
|   38 | 199.198.110.225| 2017-06-26 22:49:33 |
|   39 | 171.145.156.80 | 2017-06-26 22:49:33 |
|   40 | 178.110.5.59   | 2017-06-26 22:49:35 |
|   41 | 175.142.129.181| 2017-06-26 22:49:38 |
|   42 | 50.145.147.137 | 2017-06-26 22:49:38 |
|   43 | 117.22.33.44   | 2017-06-26 22:49:39 |
+------+----------------+---------------------+
19 rows in set (0.00 sec)
```

Figure 6 : Permanent IP Address Database (PAD)

3.1.1.3 Third Attack Detection Phase

This is the last detection phase in which the detection system compares the percentage of single-packets flows to a threshold to decide if there is an attack. The detection system confirms an attack if the following conditions are satisfied.

$$Pr_n > \phi_{n-1}$$

$$Pr_{n+1} > \phi_{n-1}$$

$$Pr_{n+2} > \phi_{n-1}$$

In other words, an attack is declared only if the percentage ratio of single-packet flow (P_{Ratio}) is greater than the threshold ϕ_{n-1} for three consecutive windows. Otherwise, it is considered a false alarm generated due to surge of legitimate traffic.

The reason of using three consecutive windows to determine the attack is to reduce false positives when there is a temporary surge of legitimate TCP connection requests within one or two time windows. The reason to choose single packet is because in SYN flood attack with spoofed addresses, there is only single packet sent from the attacker to the targeted victim in a flow.

Note that the detection method uses the OpenFlow statistics to determine Pr_n and the threshold. Usually these statistics are stored in OpenFlow switch tables. Each entry of the table contains various fields associated with a flow such as number of bytes transmitted, number of packets transmitted, flow duration, source and destination IP addresses, and source and destination port numbers. The controller requests the statistics from switches at the end of each window. For this a *Flow-Stat-Collector* module is added into POX controller. The module polls these statistics every 3 seconds from OpenFlow switches and stores them into a database called Flow Stat Database (FSD) in this thesis. Figure 7 shows Flow Stat Database.

P_Id	Switch	Bytes	Packets	Flows	Duration	nw_src	nw_dst	tp_src	tp_dst	Datetime
1961	00-00-00-00-00-01	66	1	1	0	10.0.0.2	10.0.0.5	54209	80	2017-02-22 08:52:21
1962	00-00-00-00-00-01	516	7	1	3	10.0.0.2	10.0.0.5	54209	80	2017-02-22 08:52:24
1963	00-00-00-00-00-01	516	7	1	6	10.0.0.2	10.0.0.5	54209	80	2017-02-22 08:52:27
1964	00-00-00-00-00-01	648	9	1	9	10.0.0.2	10.0.0.5	54209	80	2017-02-22 08:52:30
1965	00-00-00-00-00-01	440	6	1	2	10.0.0.2	10.0.0.5	54210	80	2017-02-22 08:52:30
1966	00-00-00-00-00-01	440	6	1	2	10.0.0.2	10.0.0.5	54211	80	2017-02-22 08:52:33
1967	00-00-00-00-00-01	648	9	1	12	10.0.0.2	10.0.0.5	54209	80	2017-02-22 08:52:33
1968	00-00-00-00-00-01	572	8	1	5	10.0.0.2	10.0.0.5	54210	80	2017-02-22 08:52:33
1969	00-00-00-00-00-01	648	9	1	15	10.0.0.2	10.0.0.5	54209	80	2017-02-22 08:52:36
1970	00-00-00-00-00-01	506	7	1	5	10.0.0.2	10.0.0.5	54211	80	2017-02-22 08:52:36
1971	00-00-00-00-00-01	740	10	1	2	10.0.0.2	10.0.0.5	54212	80	2017-02-22 08:52:36
1972	00-00-00-00-00-01	572	8	1	8	10.0.0.2	10.0.0.5	54210	80	2017-02-22 08:52:36
1973	00-00-00-00-00-01	290	4	1	0	10.0.0.2	10.0.0.5	54213	80	2017-02-22 08:52:39
1974	00-00-00-00-00-01	648	9	1	18	10.0.0.2	10.0.0.5	54209	80	2017-02-22 08:52:39
1975	00-00-00-00-00-01	572	8	1	8	10.0.0.2	10.0.0.5	54211	80	2017-02-22 08:52:39
1976	00-00-00-00-00-01	806	11	1	5	10.0.0.2	10.0.0.5	54212	80	2017-02-22 08:52:39
1977	00-00-00-00-00-01	572	8	1	11	10.0.0.2	10.0.0.5	54210	80	2017-02-22 08:52:39
1978	00-00-00-00-00-01	590	8	1	3	10.0.0.2	10.0.0.5	54213	80	2017-02-22 08:52:42
1979	00-00-00-00-00-01	572	8	1	11	10.0.0.2	10.0.0.5	54211	80	2017-02-22 08:52:42
1980	00-00-00-00-00-01	872	12	1	8	10.0.0.2	10.0.0.5	54212	80	2017-02-22 08:52:42
1981	00-00-00-00-00-01	572	8	1	14	10.0.0.2	10.0.0.5	54210	80	2017-02-22 08:52:42
1982	00-00-00-00-00-01	590	8	1	6	10.0.0.2	10.0.0.5	54213	80	2017-02-22 08:52:45
1983	00-00-00-00-00-01	572	8	1	14	10.0.0.2	10.0.0.5	54211	80	2017-02-22 08:52:45
1984	00-00-00-00-00-01	872	12	1	11	10.0.0.2	10.0.0.5	54212	80	2017-02-22 08:52:45
1985	00-00-00-00-00-01	722	10	1	9	10.0.0.2	10.0.0.5	54213	80	2017-02-22 08:52:48
1986	00-00-00-00-00-01	440	6	1	2	10.0.0.2	10.0.0.5	54214	80	2017-02-22 08:52:48
1987	00-00-00-00-00-01	872	12	1	14	10.0.0.2	10.0.0.5	54212	80	2017-02-22 08:52:48
1988	00-00-00-00-00-01	722	10	1	12	10.0.0.2	10.0.0.5	54213	80	2017-02-22 08:52:51
1989	00-00-00-00-00-01	740	10	1	5	10.0.0.2	10.0.0.5	54214	80	2017-02-22 08:52:51
1990	00-00-00-00-00-01	722	10	1	15	10.0.0.2	10.0.0.5	54213	80	2017-02-22 08:52:54
1991	00-00-00-00-00-01	440	6	1	2	10.0.0.2	10.0.0.5	54215	80	2017-02-22 08:52:54
1992	00-00-00-00-00-01	872	12	1	8	10.0.0.2	10.0.0.5	54214	80	2017-02-22 08:52:54
1993	00-00-00-00-00-01	722	10	1	18	10.0.0.2	10.0.0.5	54213	80	2017-02-22 08:52:57
1994	00-00-00-00-00-01	590	8	1	5	10.0.0.2	10.0.0.5	54215	80	2017-02-22 08:52:57
1995	00-00-00-00-00-01	872	12	1	11	10.0.0.2	10.0.0.5	54214	80	2017-02-22 08:52:57
1996	00-00-00-00-00-01	722	10	1	9	10.0.0.2	10.0.0.5	54215	80	2017-02-22 08:53:00
1997	00-00-00-00-00-01	872	12	1	14	10.0.0.2	10.0.0.5	54214	80	2017-02-22 08:53:00
1998	00-00-00-00-00-01	590	8	1	2	10.0.0.2	10.0.0.5	54216	80	2017-02-22 08:53:00
1999	00-00-00-00-00-01	722	10	1	12	10.0.0.2	10.0.0.5	54215	80	2017-02-22 08:53:03

Figure 7 : OpenFlow Statistics in Flow Stat Database

It should be noted that this phase can also be used for the detection of UDP attack with spoofed source addresses. It is because, like the SYN attack, only a single packet is sent from the attacker to the targeted victim for each flow. Figure 8 below shows the SYN attack detection flowchart.

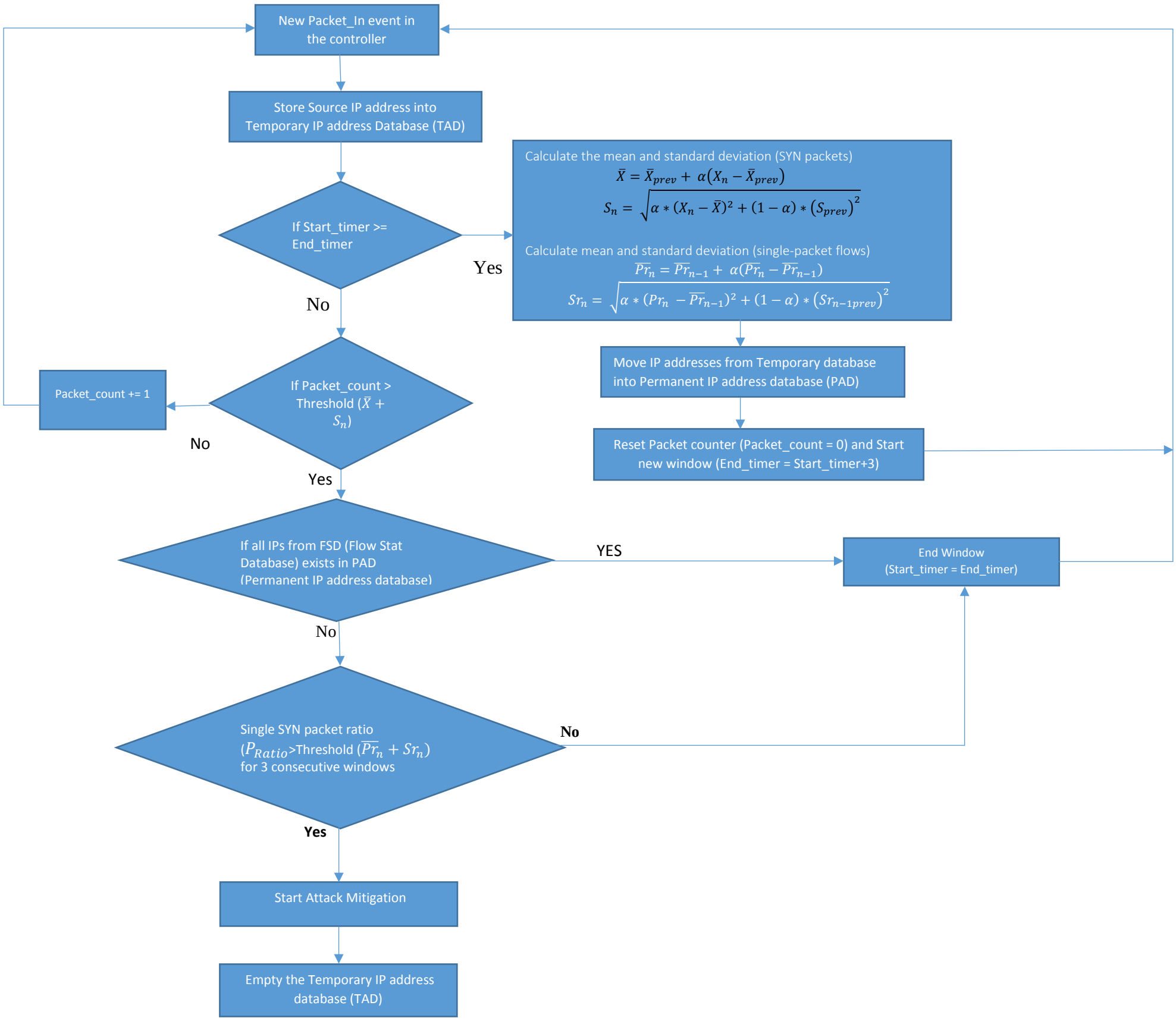


Figure 8 : SYN Detection Algorithm Flowchart

3.2 DDoS attack detection with Non-Spoofed IP Addresses

In this section, the proposed detection measurements are discussed for DDoS attack detection with non-spoofed IP addresses e.g. HTTP flood attack.

3.2.1 HTTP Attack Detection

In HTTP attack, once the connection is established to the web server, an attacker can saturate the data plane bandwidth, switch flow tables, and consume server's resources by sending excessive packet requests continuously, leading to access denial to legitimate clients. A straight forward way to detect such attack is to measure the data rate of each flow. If the data rate of a flow exceeds certain threshold, then the system assumes that flow is an attacking flow and appropriate mitigation method is used to block the flow. A simplistic way to choose a threshold is to set it to a fixed percentage (such as 10%) of the total link bandwidth [47]. Choosing a fixed threshold, however, does not provide adequate attack detection. For example, if an attacker can control a number of attacking hosts, that attacker can adjust the attacking data rate in each attacking host such that it will not cross the threshold while the overall attacking traffic can still overwhelm the network. In what follows, we will derive an adaptive threshold that provides a better detection.

The anomaly detection system is divided into two attack detection phases. The first attack detection phase utilizes a time series window-based traffic statistic measurement. The detection algorithm continuously monitors the total traffic rate generated by all the hosts in the network in each time window and compares it to a threshold at the end of the window. As in the previous case, the detection algorithm uses mean and standard deviation of the total traffic data rate to derive the appropriate threshold. The detection algorithm quickly raises a warning when the total traffic rate exceeds the threshold. To avoid false alarms because of surge of normal traffic, the detection algorithm shifts into a second detection phase, in which it further analyses the traffic by comparing individual traffic load generated by each host to a second threshold (We will describe second threshold in detail in the second attack detection phase in next section). If for three consecutive windows the load traffic generated by a host is over the threshold, attack is declared. Let us discuss in detail how the thresholds used in the two attack detection phases are derived.

3.2.1.1 First Attack Detection Phase

Let T_n be the total data rate generated by all the hosts in the n^{th} time window. The algorithm updates the mean and standard deviation of the total traffic at the end of the n^{th} window using following equations:

$$Mean = \bar{T}_n = \bar{T}_{n-1} + \alpha(T_n - \bar{T}_{n-1}) \qquad (3.8)$$

$$Std.\ Dev. = D_n = \sqrt{\alpha * (T_n - \bar{T}_{n-1})^2 + (1-\alpha) * (D_{n-1})^2} \qquad (3.9)$$

where the tuning parameter $0 < \alpha \leq 1$, is a constant smoothing factor. The value of $\alpha$is again set to 0.5 in this thesis. $\bar{T}_n$ and D_n are the mean and standard deviation computed at the n^{th} window.

The dynamic threshold $\delta_n$used in the n^{th} window is set according to equation (3.10):

$$\delta_n = \bar{T}_n + 3 * D_n \qquad (3.10)$$

If the total data rate of traffic T_n exceeds the threshold δ_n, the detection algorithm shifts into the second detection phase. Note that we choose 3 standard deviations to derive the threshold because we expect the total traffic flow may have large fluctuation in a short time window in practice [36].

3.2.1.2 Second Attack Detection Phase

In this section, we will derive an appropriate threshold for the second phase of detection. Our aim here is to derive an adaptive threshold to detect attacking flows even if the attacker can launch an attack in a number of attacking machines. Let A and L be the number of attacking hosts and the number of legitimate hosts, respectively. Furthermore, let $\bar{R}$ be the average traffic load generated from legitimate host. $\bar{R}$ is measured and updated in each time window. During the attacking period, T_n can be approximated as follows:

$$T_n \approx L\bar{R} + Am\bar{R} \qquad (3.11)$$

The first term of the right hand side of eq (3.11) represents average total data rate generated by all the legitimate hosts while the second term represents the average total data rate generated by the

attacking hosts. The average traffic generated by an attacking host is assumed to be m times of that generated by the legitimate host. To determine the detection threshold, we just need to find m and set the threshold to $m\bar{R}$. From eq. (3.11), m can be expressed as:

$$m \approx \frac{T_n - L\bar{R}}{A\bar{R}} \qquad (3.12)$$

In general, we do not know L and A. However, we can make the following observation. Since non-address-spoofing attack, such as HTTP flood attack, requires the attacking host to use its own IP address to launch the attack, the attacker must have a full control of the attacking host. Because of this constraint, we expect the number of attacking hosts that can launch the attack simultaneously usually will be small. Based on this observation, we set

$$A \leq aN \qquad (3.13)$$

Where $N\ (= L + A)$ is the total number of hosts sending traffic to the network and $a \ll 1$. In this thesis, we choose $a = 0.1$. Clearly, m is the smallest when A is the largest? Thus,

$$m \geq \frac{T_n}{0.1N\bar{R}} - 9 \qquad (3.14)$$

Note that T_n, N and $\bar{R}$ are computed or computed and updated in each window. We use the minimum value of m to set the threshold:

$$\text{threshold} = \left(\frac{T_n}{0.1N\bar{R}} - 9\right)\bar{R}$$

Using this threshold, we cover all the attack cases where $A \leq 0.1N$. In the case where $A > 0.1N$, the detection may fail to identify an attack. If we anticipate a massive attack from a very large botnet, we can choose a larger value of a, such as $a = 0.2$. However, a larger value of a gives a smaller m, thus, increases the false positives. We speculate that $a = 0.1$ is an appropriate value for most practical situation.

In order to confirm attack, the detection algorithm compares each individual host traffic load to threshold $m\bar{R}$. Let F_d be the traffic load generated by each individual flow or host. For each flow or host, if $F_d > m\bar{R}$ for three consecutive windows, an attack is declared. On the other hand, in

the case where the average traffic loads of all the flow are smaller than the threshold $m\bar{R}$ then it is a false alarm generated by some spike in normal traffic.

Figure 9 below shows flow chart of proposed detection scheme:

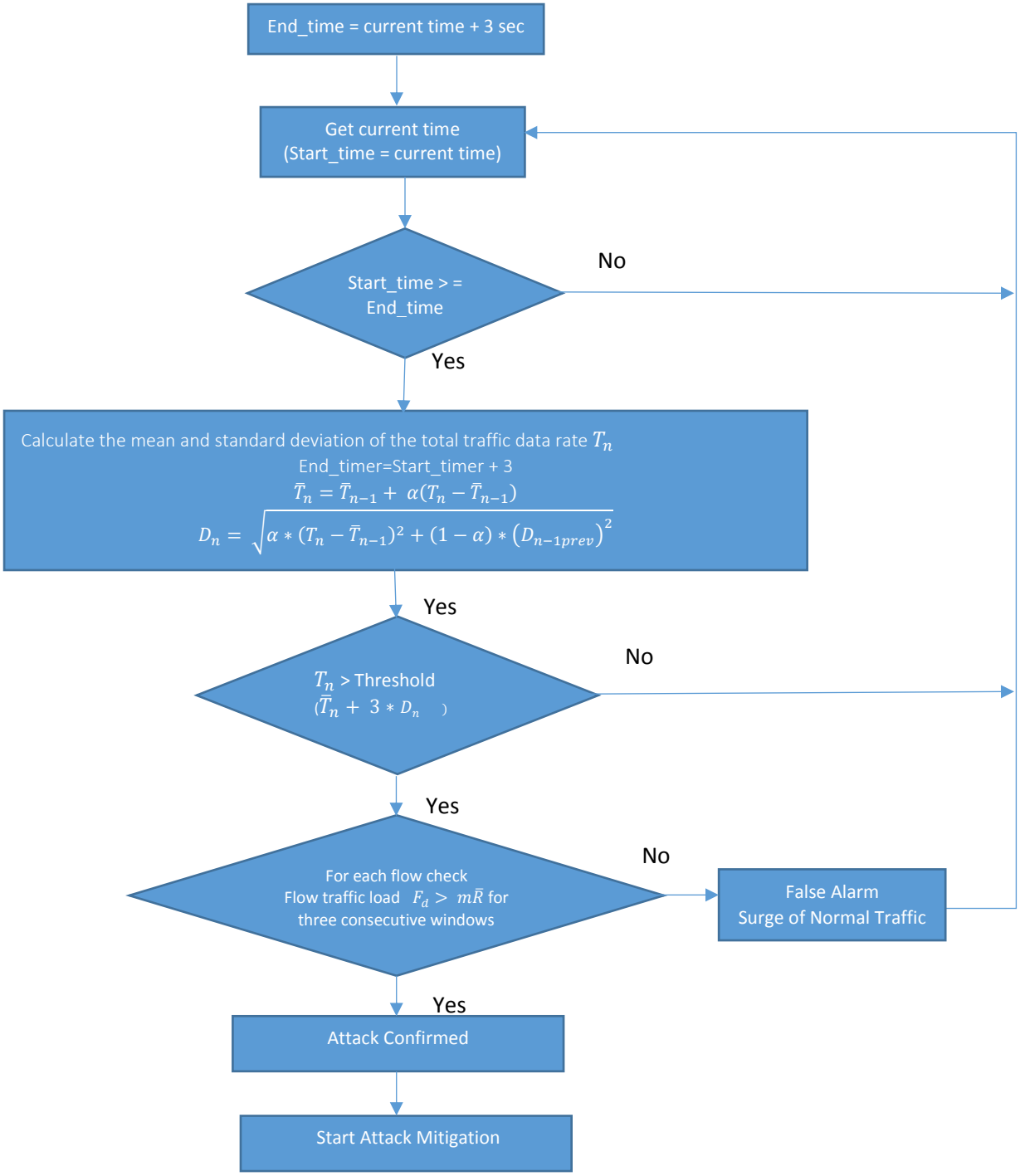


Figure 9 : HTTP flood detection flowchart

3.3 Mitigation Methods

After an attack is detected by the controller in an SDN environment, the main goal is to find a method to mitigate the effect of the attack efficiently. Due to limited resources switches are not very robust against attacks like DDoS and breaks down eventually when large number of short flows get installed in the flow table very quickly in SYN flood attack. One possible solution to mitigate the attack is installing flows to block incoming ports where the attack traffic is arriving at until the attack is stopped. But this is not an optimal solution because it will also block legitimate traffic to pass through. Our proposed mitigation approach uses Source IP address filtering scheme to mitigate SYN flood attack with spoofed addresses by allowing traffic with valid source IP addresses to pass to the web server and dropping rest of the packets. For HTTP flood attack mitigation with non-spoofed IP addresses, our proposed algorithm simply blocks the attacking source IP address.

3.3.1 SYN Flood Attack Mitigation

As discussed above in section 3.1.2, we are assuming that permanent IP address database (PAD) contains legitimate source IP addresses. As soon as the attack is detected (3.1), the POX controller push two specific types of flow entries to OpenFlow switches. We name these flows as Pass Flows and Drop Flow.

Pass flows will be given the higher priority than the Drop flow, ensuring these flows to be matched first before the Drop flow. A Pass flow specifies legitimate source IP address in the matching field. If the source IP address of a SYN packet matches anyone of these Pass flows, the packet will be redirected to the controller. The controller then set up a flow to the destination and OpenFlow rules matching each observed flow are installed with the highest priority to the switches along aforementioned path thus allowing legitimate source IP addresses traffic to reach to destination.

The drop flow is used to drop all the other SYN packets that do not find any matches. With drop flow, the switch starts dropping all the network traffic by redirecting the traffic to a sinkhole interface (port no. 3). A switch can also drop the packets if no actions are specified in a flow. In this way, the drop flow prevent DDoS sources from attacking the controller and the target destination. Figure 10 shows the permanent IP address database, the corresponding pass flows and the drop flow in one of our simulation runs.

```
mysql> select * from permanentipaddress;
+------+-----------------+---------------------+
| P_Id | IPADDRESS       | Datetime            |
+------+-----------------+---------------------+
|   25 | 34.10.20.1      | 2017-06-26 22:49:08 |
|   26 | 24.10.20.1      | 2017-06-26 22:49:13 |
|   27 | 25.10.20.1      | 2017-06-26 22:49:21 |
|   28 | 161.110.120.1   | 2017-06-26 22:49:25 |
|   29 | 66.10.20.1      | 2017-06-26 22:49:25 |
|   30 | 27.10.20.1      | 2017-06-26 22:49:25 |
|   31 | 37.10.20.1      | 2017-06-26 22:49:25 |
|   32 | 155.110.120.1   | 2017-06-26 22:49:25 |
|   33 | 28.10.20.1      | 2017-06-26 22:49:25 |
|   34 | 64.10.20.1      | 2017-06-26 22:49:25 |
|   35 | 26.10.20.1      | 2017-06-26 22:49:25 |
|   36 | 38.200.45.185   | 2017-06-26 22:49:31 |
|   37 | 197.137.94.97   | 2017-06-26 22:49:32 |
|   38 | 199.198.110.225 | 2017-06-26 22:49:33 |
|   39 | 171.145.156.80  | 2017-06-26 22:49:33 |
|   40 | 178.110.5.59    | 2017-06-26 22:49:35 |
|   41 | 175.142.129.181 | 2017-06-26 22:49:38 |
|   42 | 50.145.147.137  | 2017-06-26 22:49:38 |
|   43 | 117.22.33.44    | 2017-06-26 22:49:39 |
+------+-----------------+---------------------+
19 rows in set (0.00 sec)
```

(a) Permanent IP Address Database

```
cookie=0x0,duration=30.91s,table=0,n_packets=0,n_bytes=0,idle_age=30,priority=65123,ip,nw_src=64.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.953s,table=0,n_packets=5,n_bytes=370,idle_age=12,priority=65123,ip,nw_src=24.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.859s,table=0,n_packets=0,n_bytes=0,idle_age=30,priority=65123,ip,nw_src=178.110.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.945s,table=0,n_packets=0,n_bytes=0,idle_age=10,priority=65123,ip,nw_src=64.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.9s,table=0,n_packets=0,n_bytes=0,idle_age=11,priority=65123,ip,nw_src=66.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.877s,table=0,n_packets=5,n_bytes=370,idle_age=11,priority=65123,ip,nw_src=161.110.120.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.91s,table=0,n_packets=0,n_bytes=0,idle_age=30,priority=65123,ip,nw_src=197.137.94.97 actions=CONTROLLER:65535
cookie=0x0,duration=30.989s,table=0,n_packets=0,n_bytes=370,idle_age=12,priority=65123,ip,nw_src=199.197.11.2 actions=CONTROLLER:65535
cookie=0x0,duration=30.91ss,table=0,n_packets=0,n_bytes=0,idle_age=12,priority=65123,ip,nw_src=26.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.91s,table=0,n_packets=5,n_bytes=370,idle_age=30,priority=65123,ip,nw_src=34.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.945s,table=0,n_packets=0,n_bytes=0,idle_age=10,priority=65123,ip,nw_src=155.110.120.1 actions=CONTROLLER:65535
cookie=0x0,duration=31.028s,table=0,n_packets=5,n_bytes=0,idle_age=30,priority=65123,ip,nw_src=37.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.91s,table=0,n_packets=0,n_bytes=370,idle_age=30,priority=65123,ip,nw_src=28.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.953s,table=0,n_packets=5,n_bytes=370,idle_age=12,priority=65123,ip,nw_src=27.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.859s,table=0,n_packets=0,n_bytes=0,idle_age=30,priority=65123,ip,nw_src=171.145.10.11 actions=CONTROLLER:65535
cookie=0x0,duration=30.945s,table=0,n_packets=0,n_bytes=370,idle_age=30,priority=65123,ip,nw_src=192.168.56.23 actions=CONTROLLER:65535
cookie=0x0,duration=30.9s,table=0,n_packets=0,n_bytes=0,idle_age=10,priority=65123,ip,nw_src=15.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.877s,table=0,n_packets=5,n_bytes=370,idle_age=12,priority=65123,ip,nw_src=25.10.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.91s,table=0,n_packets=0,n_bytes=0,idle_age=10,priority=65123,ip,nw_src=178.110.20.1 actions=CONTROLLER:65535
cookie=0x0,duration=30.989s,table=0,n_packets=5,n_bytes=0,idle_age=30,priority=65123,ip,nw_src=38.200.45.185 actions=CONTROLLER:65535
```

(b) Pass Flows

```
mininet > dpctl dump-flows
* * * s 1 ----------------------------------------------------------------------------------------------------
cookie=0x0, duration=8.23s, table=0, n_packets=277, n_bytes=14958, idle_age=0, priority=1000, in_port=1 actions=output : 3
:23,dl_dst=d6:42:9a:65:6c:c2,nw_src=75.240.32.185,nw_dst=192.168.56.23,nw_tos=0,tp_src=3146,tp_dst=80 actions=output:2
 cookie=0x0,duration=3.859s,table=0,n_packets=0,n_bytes=0,idle_timeout=5,idle_age=3,priority=65535,tcp,in_port=2
:c2,dl_dst=42:ed:le:5c:5d:aa,nw_src=192.168.56.2,nw_dst=121.4.54.187,nw_tos=0,tp_src=80,tp_dst=1623, actions=output:1
```

(c) Drop Flow

Figure 10 : Pass flows and Drop flows

Figure 10(a) shows the Permanent IP Address database (PAD) contains legitimate clients IP address at the time of attack. After attack is detected, controller pushes Pass flows (Figure 10(b)) with highest priority 65123 and the Drop flow (Figure 10(c)) with lowest priority 1000 in the OpenFlow switch. Pass flows contain all the IP addresses that are in the PAD database as matching source address field. Pass flows allow the traffic with matching source IP address to pass through the network where the drop flow drops the rest of the traffic by redirecting it to a sinkhole interface port 3.

It is also important to withdraw the pass flows and the drop flow from the flow tables when DDoS attack is stopped. The network administrator can use monitoring tools to check status of attack and can manually clear the flows from OVS switch.

3.3.2 HTTP Flood Attack Mitigation

Once HTTP flood attack is detected (3.2) and abnormal flows are identified, the proposed mitigation algorithm gets the attack source IP address of abnormal flows from Flow stat database (FSD). The controller then immediately blocks the attack source IP address by installing flow entry in OpenFlow switch. The flowchart in figure 11 illustrates the proposed SYN and HTTP flood mitigation algorithm.

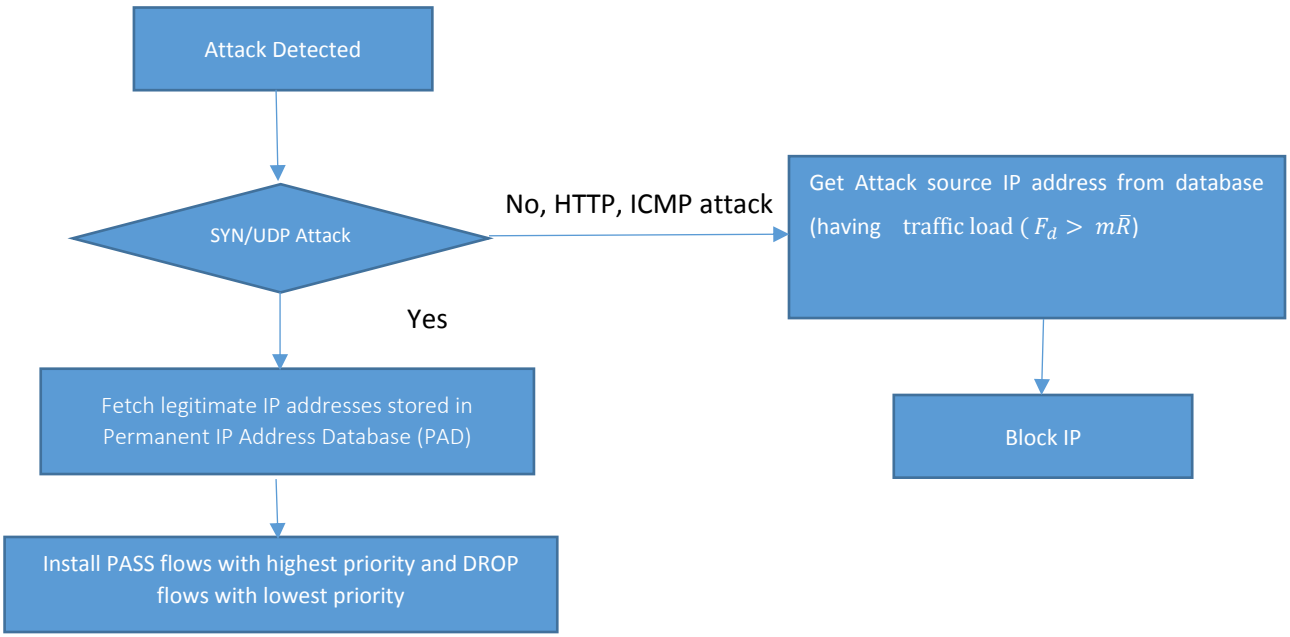


Figure 11 : Mitigation Algorithm Flowchart

Like HTTP flood attack detection and mitigation, we use the same approach for ICMP flood attack detection and mitigation.

3.4 Organization of the Detection and Mitigation Modules

Figure 12 shows an overview of the architecture of our proposed scheme. The proposed scheme consists of three modules placed within POX controller. These three modules are described below:

1. *The Flow-Stat-Collector* module is responsible for collecting flow statistics from the Flow Tables of OpenFlow switches and store them into a Flow Stat Database (FSD). FSD keeps statistics for active flows such as packet count, byte count, flows duration, source-destination IP addresses and port numbers and update them periodically with new flow statistics. These flow statistics can be used to detect potential DDoS attacks. The Flow-Stat-Collector module collects data from switches through a Secure Channel and send them to Flow-Stat-Analyzer module.
2. *The Flow-Stat-Analyzer* module is running the statistical anomaly detection algorithm and does analysis on the collected data stored in FSD database. This is the most important step, as the controller need to figure out whether there is an attack or just a legitimate flash crowd traffic. This module is further divided into two components. The first component *Analyzer 1*, issues warning for suspicious traffic when there is a surge of traffic exceeds control threshold. Further analysis is done by the second component *Analyzer 2* to find out whether a given traffic pattern corresponds to legitimate traffic or a DDoS attack. If there is an attack, the *Flow-Stat-Analyzer* issues alert to *Attack Mitigator* module.
3. *The Attack-Mitigator* immediately takes action once an alert is received. The Mitigator push OpenFlow rules to switches based on attack type. These rules may range from dropping the attack traffic to allowing legitimate traffic with certain source or destination IP addresses.

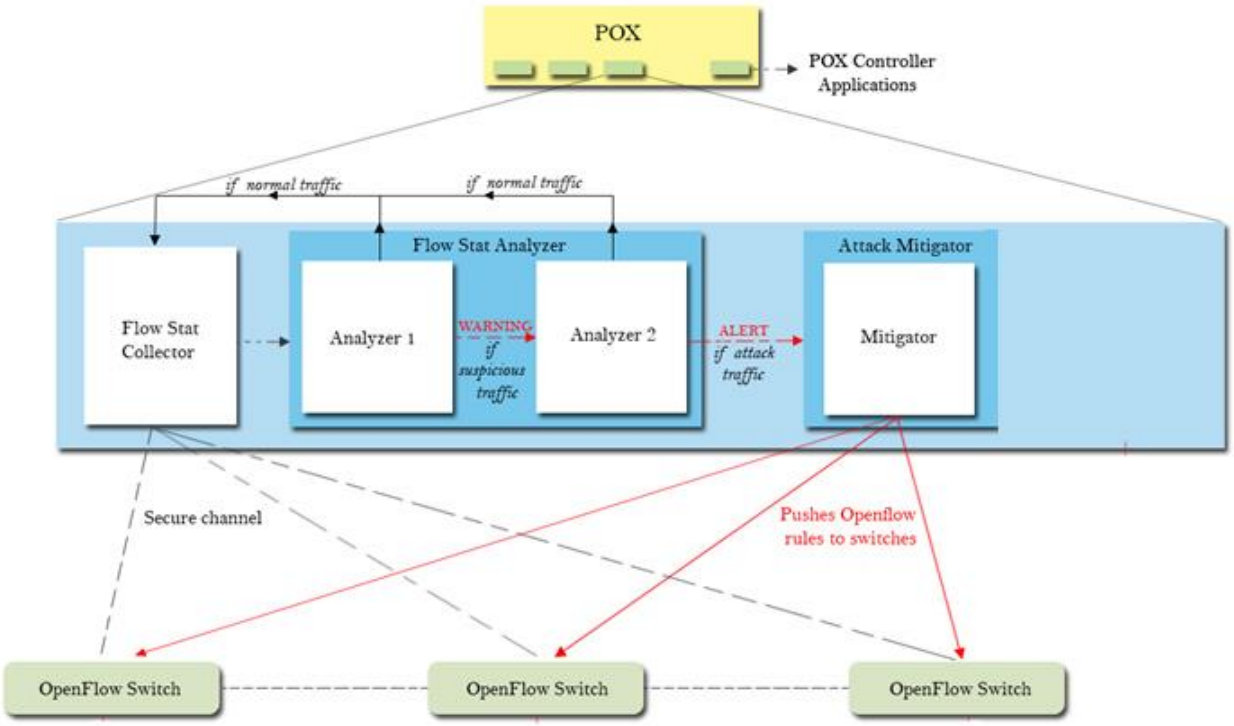


Figure 12 : Attack Detection and Mitigation Operation

3.5 Chapter summary

In this chapter, we discussed two methods for DDoS attack detection and mitigation with spoofed and non-spoofed addresses. The proposed detection methods employ a system of three set of modules, The Flow-Stat-Collector, the Flow-Stat-Analyzer and the Attack-Mitigator. The scheme is based on statistical analysis of data packets collected within a fixed window time period in case of SYN and UDP flood DDoS attacks using spoofed IP addresses. We also discussed how the presented scheme identifies abnormal flows in HTTP and ICMP flood DDoS attacks with non-spoofed addresses. Furthermore, in section 3.3, the mitigation scheme is shown to mitigate effects of an attack, once an attack is confirmed.

In the next chapter, results of the proposed DDoS detection and mitigation method will be examined.

Chapter 4

4 Performance Evaluation and Results

Having described proposed SYN and HTTP flood attack detection and mitigation algorithms in the previous chapter, we now proceed to evaluate the performance of both the algorithms in this chapter.

4.1 Testbed

In order to construct a simple testbed, we used Mininet [42] as a network emulator with OpenvSwitch as the switching devices. Mininet is the standard network emulation tool that can be used to simulate the Software Defined Networks. For the SDN controller, we used POX controller to control the flows in the Mininet environment. POX [43] is an improved version of its predecessor NOX and it is fast, lightweight and designed as a platform so a custom controller can be built on top of it. To generate legitimate TCP SYN traffic Python client-server socket programming is used. The attack detection and mitigation algorithms are implemented at the POX controller. For SYN, UDP and ICMP flood attacks we used Hping3 [44] tool to generate attack traffic. With Hping3 tool we can set different parameters such as number of packets, interval duration, random source-destination IP addresses etc. For HTTP flood attack, an attacker python script is used to generate large number of GET requests to the web server.

4.2 Network Setup

The experiment is done on a Dell laptop Intel® Core ™ i7-5500U CPU, 2.40 GHz with 8.00 GB RAM. The operating system is Linux Ubuntu 14.04 and Mininet version 2.2.1.

Using python API, a custom topology is created for the current network scenario. The network consists of two switches, 53 clients or hosts and a controller as shown in figure 13. The switches used in the network are open virtual switches or OVS. The *L2_multi* module of POX is used for the controller. Among 53 hosts, 45 hosts are acting as clients on which python client script is running to generate legitimate TCP SYN connections to the web server, one host is acting as victim/web server running python server script, two hosts are acting as attackers which uses hping3 tool to generate SYN/UDP flood DDoS attacks to the web server. An attack script and Hping3 tool are running on the other five hosts to generate Distributed HTTP flood attack and ICMP flood

attack respectively. The web server is residing inside the SDN network and all legitimate clients and attackers are in the external network, accessing the web server through internet. A gateway edge router s4 in figure 13 below is connecting to the SDN internal network to the outside environment.

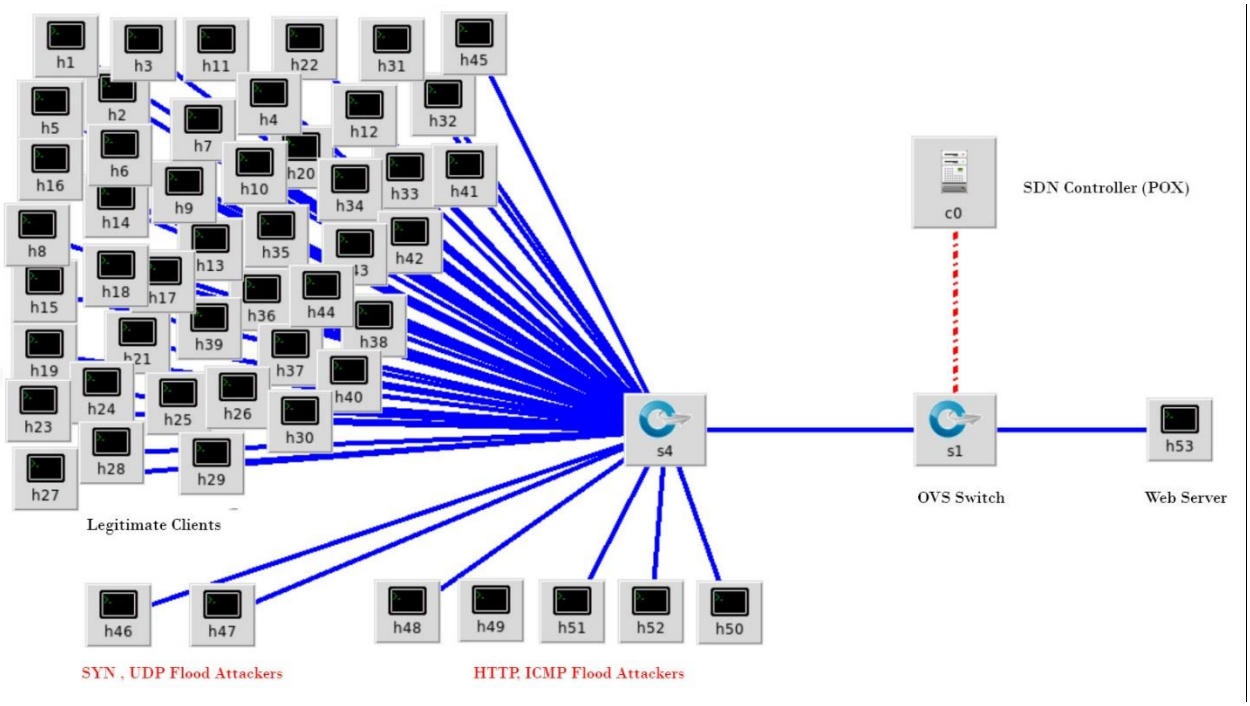


Figure 13 : Network topology

In the next section 4.3, we study the performance of the proposed scheme on identifying different DDoS attacks.

4.3 Results and Analysis

In this section, first we will have a look at the normal traffic pattern. In the subsequent subsections, we will conduct different experiments to check the validity of our approach based on the type of attack. We will investigate different scenarios using different traffic parameters such as, a) Average number of TCP connections at the web server, b) Total number of packets going to the controller, and c) HTTP response time for SYN and HTTP flood attacks. Later, we will test our scheme with other attacks such as UDP and ICMP flood attacks and analyse the effectiveness of our scheme in terms of bandwidth consumption and the number of flow entries installed in the switch flow table. The parameters of the regular traffic will be the same in all the experiments as

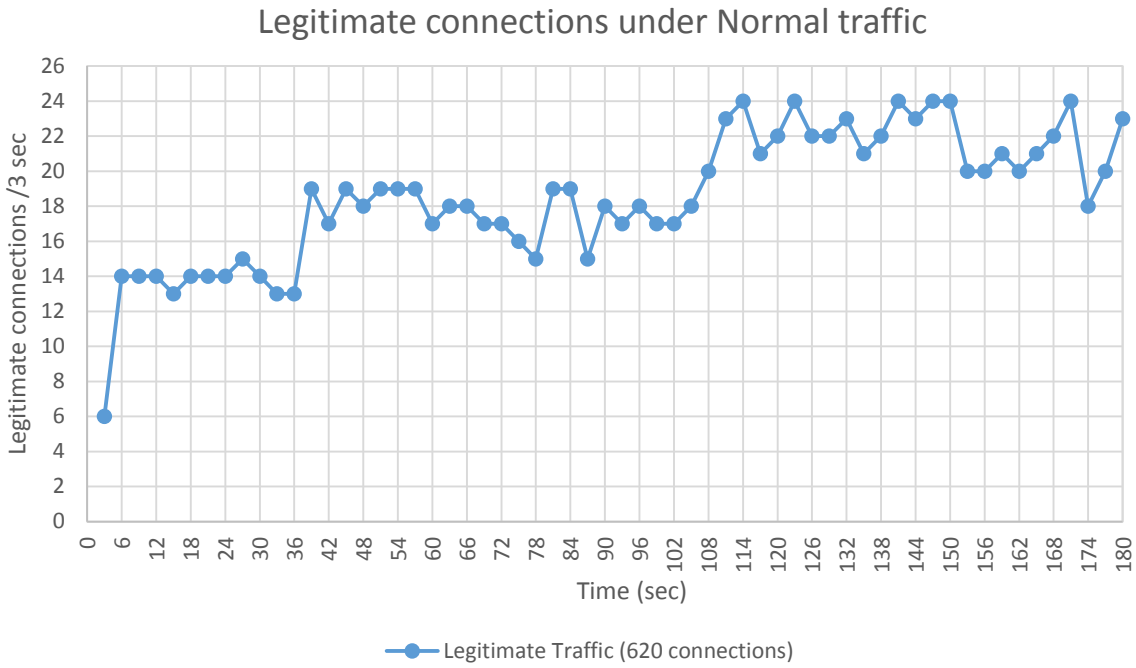
shown in table 2 below. The traffic characteristics, such as connection inter-arrival time, connection duration and number of packets generated per connection are generated randomly.

4.3.1 Normal Traffic Pattern

In order to accomplish legitimate SYN connections to generate normal traffic, a python client script is running on 45 hosts (hosts h1 to h45) and a server script on single host (h53) for at least three minutes of simulation. To make the scenario to be as realistic as possible, we divided the legitimate clients into three groups, Group A, Group B and Group C. Each group of clients running multiple web connections in parallel with different parameters. Traffic is generated randomly by each group at different time intervals.

Parameters	Group A	Group B	Group C
No. of clients	10	15	20
Traffic type	HTTP	HTTP	HTTP
No. of Servers	1	1	1
No. of Connections per client	8	10	12
Avg. HTTP Connection duration	8 s	13 sec	10 sec
Avg. Inter-arrival time between connections	1 s	1.2 s	1.3 s
No. of Get requests per connection	5	6	7

Table 2: Normal traffic pattern



(a) Legitimate connections under normal traffic (3 sec window)

Client	Server	State	Idle	A	Speed
16.10.20.1:59852	192.168.56.23:80	ESTABLISHED	0s		0 B/s
133.110.120.1:49173	192.168.56.23:80	CLOSED	1s		0 B/s
62.10.20.1:55554	192.168.56.23:80	CLOSING	0s		0 B/s
16.10.20.1:59842	192.168.56.23:80	CLOSING	1s		0 B/s
62.10.20.1:55543	192.168.56.23:80	CLOSED	2s		0 B/s
32.10.20.1:57662	192.168.56.23:80	ESTABLISHED	0s		68 B/s
131.110.120.1:53334	192.168.56.23:80	CLOSED	0s		0 B/s
10.10.20.1:46718	192.168.56.23:80	ESTABLISHED	0s		0 B/s
133.110.120.1:49183	192.168.56.23:80	ESTABLISHED	1s		34 B/s
10.10.20.1:46699	192.168.56.23:80	CLOSED	2s		0 B/s
16.10.20.1:59832	192.168.56.23:80	CLOSED	2s		0 B/s
10.10.20.1:46708	192.168.56.23:80	CLOSING	0s		0 B/s
131.110.120.1:53345	192.168.56.23:80	ESTABLISHED	0s		68 B/s
11.10.20.1:43483	192.168.56.23:80	CLOSED	0s		0 B/s
32.10.20.1:57652	192.168.56.23:80	CLOSED	0s		0 B/s
11.10.20.1:43494	192.168.56.23:80	ESTABLISHED	0s		68 B/s
58.10.20.1:47578	192.168.56.23:80	CLOSED	0s		0 B/s
62.10.20.1:55564	192.168.56.23:80	ESTABLISHED	0s		0 B/s
30.10.20.1:54338	192.168.56.23:80	CLOSED	1s		0 B/s
59.10.20.1:53507	192.168.56.23:80	ESTABLISHED	0s		68 B/s
58.10.20.1:47586	192.168.56.23:80	ESTABLISHED	0s		68 B/s
TOTAL					408 B/s
Connections 1-21 of 23		Unpaused	Unsorted		

(b) Server backlog

Figure 14 : Legitimate traffic before attack

Figure 14 illustrates the average number of legitimate http connections in a 3 second window (figure 14 (a)) and the state of the connections in the server backlog (figure 14 (b)). It is clear from the above figures, the normal traffic behaviour of a web server and all the connections are either in established state or closing state after successful data connections. We will use this normal traffic pattern (figure 14(a)) as a baseline to analyse our proposed scheme in different attack scenarios.

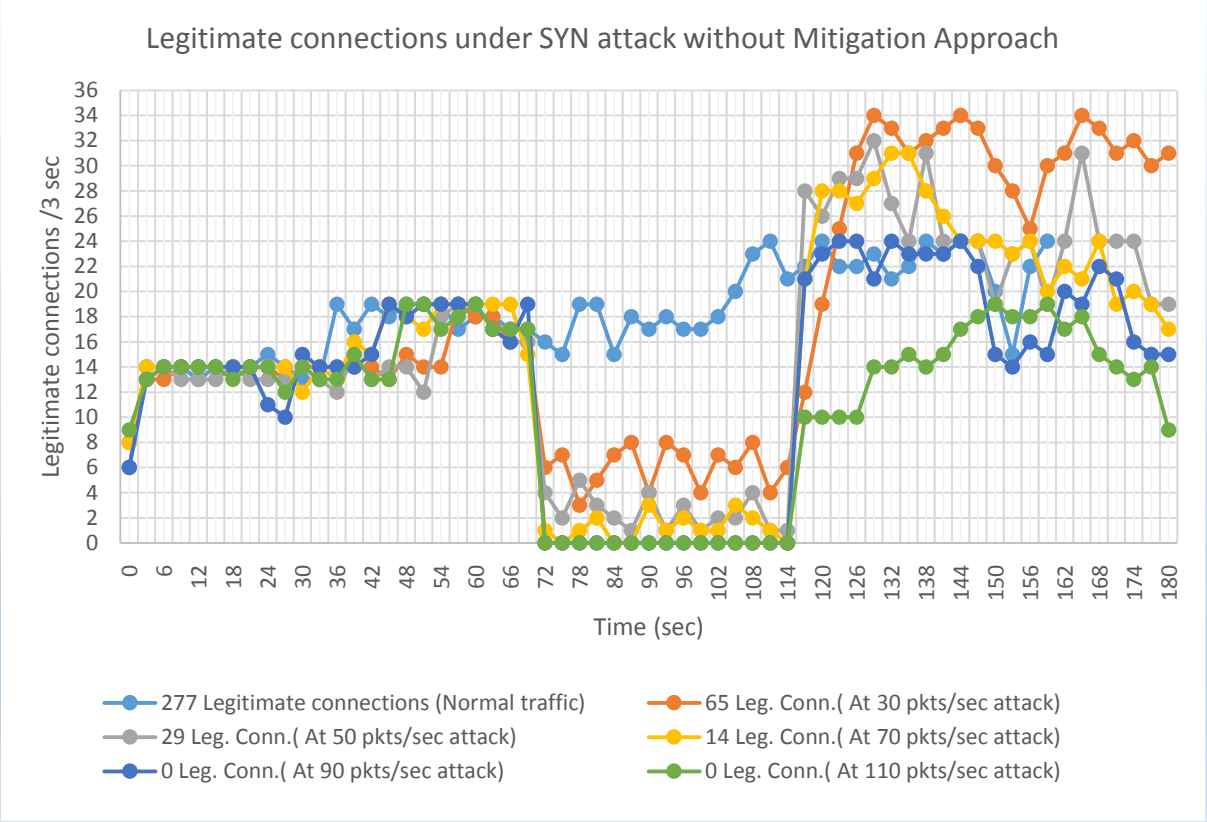
4.3.2 SYN Flood Attack Detection and Mitigation

4.3.2.1 Attack Traffic

To test this solution, SYN flood attack traffic is generated with different intensities. For generating particular SYN DDoS attack, hping3 tool is used at two attacker nodes h46 and h47 with random source IP spoofing. Five simulation runs are performed consisting of five different attack scenarios and each simulation lasts around three minutes. Both the attackers start the attack at t=72 seconds and ending at t=114 second, in all the simulations. Starting with a 30 packets/sec rate of attack, traffic rate is then increased to 50 packets/sec, 70 packets/sec, and 90 packets/sec. In the fifth attack scenario the attack traffic rate is ramped up to 110 packets/sec to completely suppress the normal traffic. Under different attack scenarios the number of legitimate connections will be counted in a three second window during the attacking window time period from t=72 seconds to t=114 seconds. Table 3 summarizes the traffic specifications in each attack scenario.

Attack Rate (pkts/sec)	Attack Type	No. of Attackers	No. of Clients (Legitimate)	No. of servers
30	SYN flood	2	45	1
50	SYN flood	2	45	1
70	SYN flood	2	45	1
90	SYN flood	2	45	1
110	SYN flood	2	45	1

Table 3: Attack Traffic Pattern



(a) No. of Legitimate connections under SYN flood attack

Client	Server	State	Idle	A	Speed
13.241.131.168:3017	192.168.56.23:80	SYN_RECV	15s		0 B/s
143.6.247.222:3013	192.168.56.23:80	SYN_RECV	16s		0 B/s
144.169.220.197:3019	192.168.56.23:80	SYN_RECV	15s		0 B/s
108.240.60.206:3026	192.168.56.23:80	SYN_RECV	11s		0 B/s
250.19.154.110:3009	192.168.56.23:80	SYN_RECV	16s		0 B/s
239.134.153.20:3003	192.168.56.23:80	SYN_RECV	16s		0 B/s
184.188.134.89:3016	192.168.56.23:80	SYN_RECV	15s		0 B/s
233.199.20.246:3011	192.168.56.23:80	SYN_RECV	18s		0 B/s
182.99.68.182:3024	192.168.56.23:80	SYN_RECV	12s		0 B/s
129.222.112.169:3015	192.168.56.23:80	SYN_RECV	15s		0 B/s
194.136.194.64:3014	192.168.56.23:80	SYN_RECV	16s		0 B/s
141.212.207.38:3006	192.168.56.23:80	SYN_RECV	18s		0 B/s
129.248.220.191:3004	192.168.56.23:80	SYN_RECV	17s		0 B/s
173.128.59.108:3023	192.168.56.23:80	SYN_RECV	12s		0 B/s
4.4.13.124:3005	192.168.56.23:80	SYN_RECV	18s		0 B/s
220.242.108.207:3028	192.168.56.23:80	SYN_RECV	11s		0 B/s
112.131.241.83:3001	192.168.56.23:80	SYN_RECV	16s		0 B/s
151.85.117.102:3027	192.168.56.23:80	SYN_RECV	11s		0 B/s
144.12.121.188:3029	192.168.56.23:80	SYN_RECV	11s		0 B/s
161.130.70.94:3018	192.168.56.23:80	SYN_RECV	15s		0 B/s
245.183.74.147:3021	192.168.56.23:80	SYN_RECV	15s		0 B/s
TOTAL					0 B/s
Connections 1-21 of 29		Unpaused	Unsorted		

(b) Server backlog

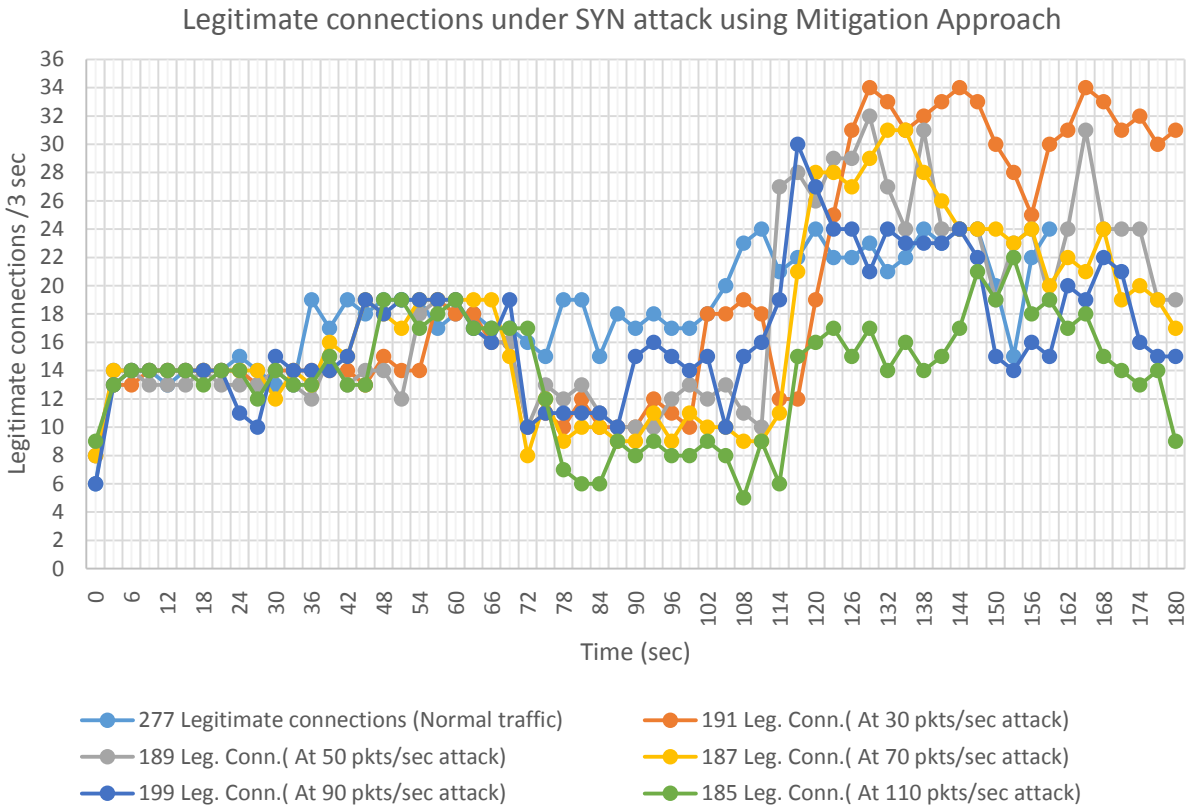
Figure 15 : SYN Flood Attack without defense mechanism

As the figure 15(a) shows, 72 seconds after the attack has started, the average number of legitimate connections in a 3 second window starts decreasing. This means that the server will stop receiving data from legitimate hosts. We can clearly see that the legitimate connections are brought down to 0, under SYN flood attack rate of 110 packets/second. However, due to low impact of attack at 30 packets/sec, some of the clients are still able to make successful connections to the server. Where this number is decreasing continuously as the attack rate is increasing. By increasing attack rate from 30 packets to 50 packets and to 70 packets per second, the average number of legitimate connections fell down to 6, 3 and then 1 approximately, with almost 0 at 90 and 110 packets per second attack rate. This is because DDoS traffic completely suppresses the legitimate HTTP traffic. Figure 15(b) shows the connection states at the backlog of server where a large number of state connections from random IP addresses in a SYN_RECV (half-connection) state.

Next, we will see the effectiveness of our mitigation approach.

4.3.2.2 Attack mitigation approach effectiveness

In this section we will check the effectiveness of our mitigation approach based on source IP address filtering by counting the http connections in three second window. As soon as the attack is detected, the controller starts mitigating the attack by pushing Pass flows and Drop flows in OVS switch as discussed in chapter 3 section 3.3. Once the new flows are pushed, the switch starts dropping traffic by allowing legitimate http sessions to continue to the web server.



(a) No. of Legitimate connections under SYN flood attack

Client	Server	State	Idle	A Speed
120.80.114.221:3979	192.168.56.23:80	SYN_SENT	8s	0 B/s
245.39.39.151:3975	192.168.56.23:80	SYN_SENT	14s	0 B/s
237.25.46.193:3974	192.168.56.23:80	SYN_SENT	14s	0 B/s
27.10.20.1:56345	192.168.56.23:80	ESTABLISHED	1s	68 B/s
24.10.20.1:47378	192.168.56.23:80	CLOSING	1s	0 B/s
128.172.204.220:3958	192.168.56.23:80	SYN_SENT	14s	0 B/s
129.110.120.1:57641	192.168.56.23:80	ESTABLISHED	1s	68 B/s
24.10.20.1:47421	192.168.56.23:80	ESTABLISHED	1s	68 B/s
216.179.150.74:3970	192.168.56.23:80	SYN_SENT	14s	0 B/s
201.43.88.74:3971	192.168.56.23:80	SYN_SENT	14s	0 B/s
15.10.20.1:55221	192.168.56.23:80	RESET	1s	0 B/s
49.4.89.163:3955	192.168.56.23:80	SYN_SENT	15s	0 B/s
161.110.120.1:54256	192.168.56.23:80	ESTABLISHED	1s	68 B/s
25.10.20.1:46457	192.168.56.23:80	CLOSING	1s	0 B/s
26.10.20.1:33507	192.168.56.23:80	ESTABLISHED	1s	34 B/s
90.28.137.179:3981	192.168.56.23:80	SYN_SENT	8s	0 B/s
158.226.151.87:3983	192.168.56.23:80	SYN_SENT	8s	0 B/s
206.36.12.163:3973	192.168.56.23:80	SYN_SENT	14s	0 B/s
28.167.142.168:3980	192.168.56.23:80	SYN_SENT	8s	0 B/s
131.234.80.106:3954	192.168.56.23:80	SYN_SENT	15s	0 B/s
161.110.120.1:54250	192.168.56.23:80	CLOSING	1s	0 B/s
TOTAL				578 B/s
Connections 1-21 of 37		Unpaused	Unsorted	

(b) Server backlog

Figure 16 : Legitimate Traffic under SYN Flood Attack with a defense mechanism

From figure 16(a), we can see the effectiveness of our scheme. Legitimate traffic is started at the beginning of the experiment, and SYN flood attack is launched starting at t=72 seconds and ending at t=114 second, by both the attackers h46 and h47. As can be seen at approximately t=74 second, our defense scheme comes into action and starts mitigation. After the defense mechanism has been activated, an average of 14 legitimate clients can still send and receive information to and from the web server based on Source IP address filtering scheme discussed in section 3.3 above. This can be seen between t=72 second to t=114 second period of time. Attack is stopped at t=114 second to allow new legitimate connections to reach to the web server. Figure 16(b) shows the server backlog few seconds after the attack has been blocked.

Table 4 illustrates the total number of successful connections recorded during the attack period, from t=72 sec to t=114 sec, with and without defense mechanism. It is to be noted that, at attack rate 30 pkts/sec, and 50 pkts/sec without using mitigation approach few of the clients are still able to make http connections because of low rate/impact of SYN attack.

	With Mitigation method	**Without Mitigation method**
Attack Rate	**Total no. of Legitimate connections**	**Total no. of Legitimate connections**
0	277	0
30	191	65
50	189	29
70	187	14
90	199	0
110	185	0

Table 4: Mitigation method effectiveness

We can see in figure 17 the overall process of attack detection and mitigation.

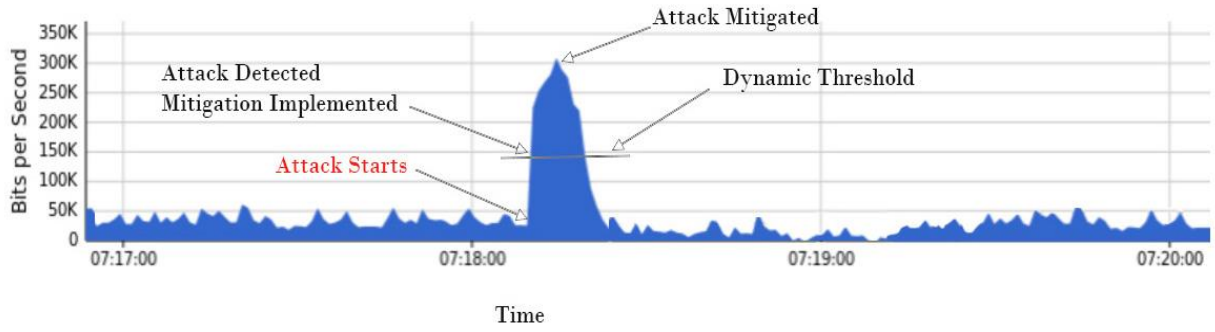


Figure 17 : Detection and Mitigation Attack process

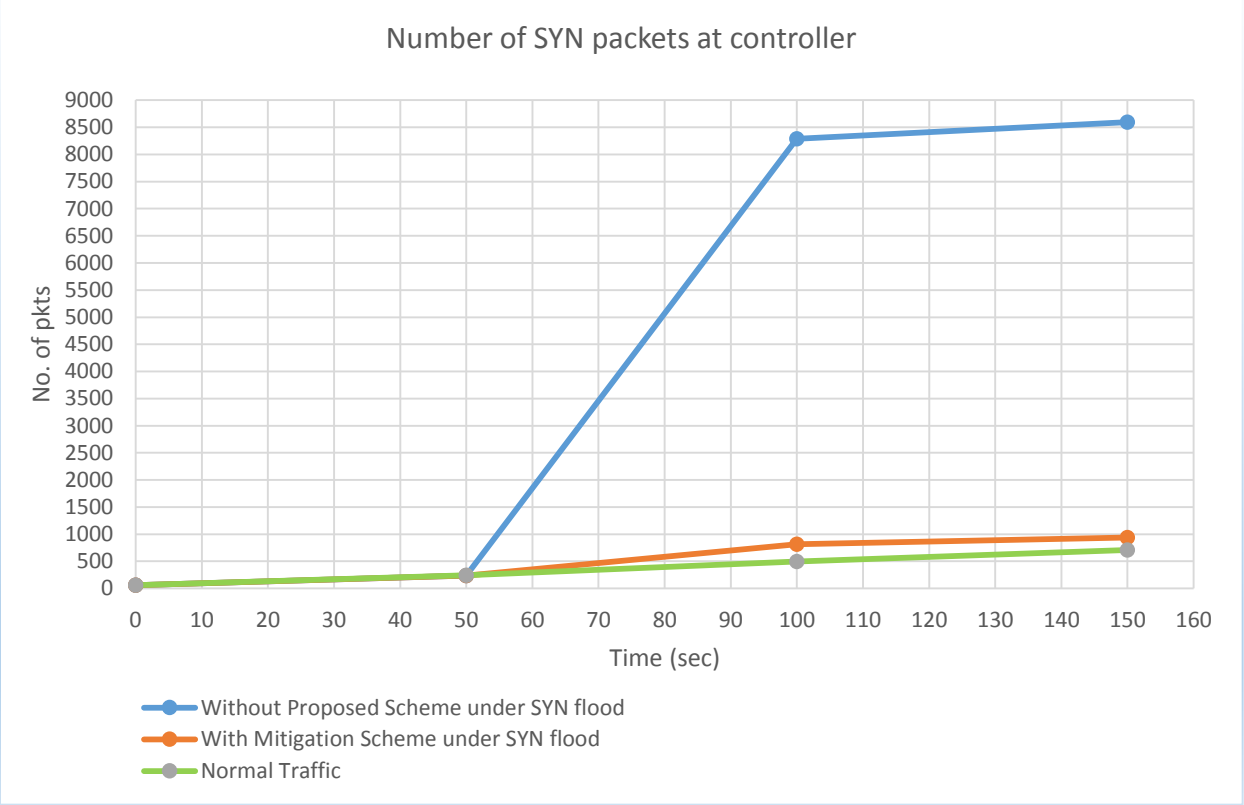


Figure 18 : Number of packets arriving at the controller

Figure 18 shows the total number of packets arriving at the controller after every 50 seconds. It is clear from the figure, during the first 50th seconds, when traffic rate was normal the total number of SYN packets arrived at the controller was approximately 800. The SYN flood started at 50th second. As soon as the attack was started, the number of SYN packets rose sharply to 8288. This is because the switch didn't find any match for the packets and forward all the packets to the controller. In contrast, when mitigation takes place, instead of reaching a peak of eight thousand packets, the attack is limited to 1000 packet slightly more than the normal traffic because few attack packets arrived at the controller before detection.

4.3.3 HTTP Flood Attack Detection and Mitigation

This experiment evaluates the effectiveness of the proposed defense mechanism in mitigating HTTP flood DDoS attack. We evaluate this experimental result using HTTP response time as a performance metric.

4.3.3.1 Attack Traffic

To test this solution, HTTP flood attack traffic is generated in two attack scenarios. In the first attack scenario, 5 attacker hosts generate HTTP flood GET requests to the destination web server. In the second scenario, 10 attacker hosts are generating HTTP flood GET requests. In order to generate HTTP flood GET requests, we run a simple python based script on attacker hosts. Each one of the attacking hosts will send 30000 GET requests with time interval of 0.1s between successive requests to the web server.

4.3.3.2 Attack mitigation approach effectiveness

The experiment in this scenario measures the response time for a legitimate client when it requests HTTP content under HTTP flood attack. HTTP response time is the time between requesting a web page by a legitimate user and receiving it from the attacked web server.

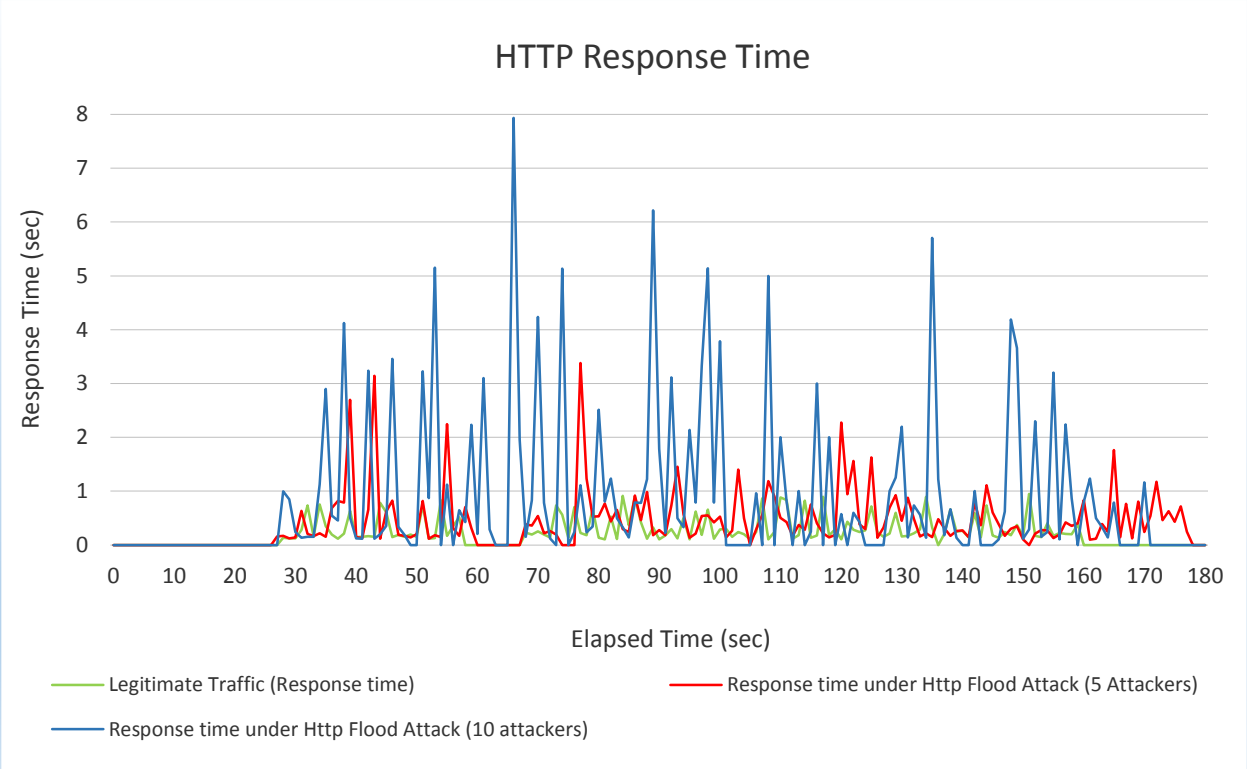


Figure 19 : HTTP response time under Normal Traffic and Attack Traffic

Figure 19 shows the average HTTP response time for the legitimate user's requests. In the first case, during normal traffic the average response time is between 0.3 seconds and 0.6 seconds. On the other hand, the average response time is increased almost 3 to 7 times during the first attack

scenario (5 attacker hosts). As the number of attacker hosts increased from 5 to 10 in second attack scenario, the response time rose to 8 seconds where average response time is 4 seconds approximately, because with ten attacker hosts and load of 300000 GET requests, a legitimate client ends up using very small share of the bandwidth.

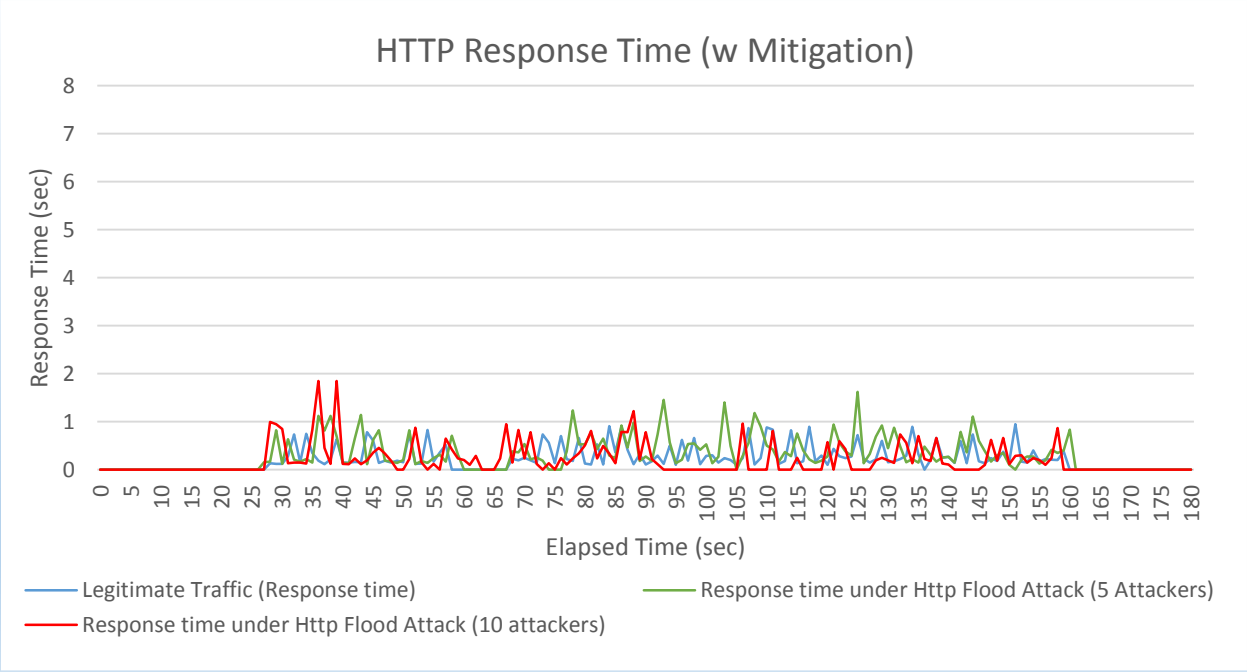


Figure 20: HTTP Response time with Mitigation Scheme

The average HTTP response time using mitigation scheme is shown in figure 20. With mitigation scheme, the HTTP flood attack is quickly detected by the Flow Stat Analyzer, which notifies the Attack Mitigator. The Attack Mitigator retrieves details of the attacker IP address and blocks it quickly. This can be seen in the figure, the average response time is almost the same as normal traffic with mitigation scheme in both the attack scenarios.

4.3.4 ICMP Flood Attack Detection and Mitigation

This experiment tests the proposed mitigation scheme against ICMP flood attack in terms of bandwidth consumption. The detection and mitigation scheme in this experiment is same as that of the HTTP flood attack. The goal of the attacker in ICMP flood attack is to consume bandwidth in the network by sending a large number of ICMP packets targeting random servers or a specific server [45].

Attack Traffic

Since our testbed is based on Mininet and according to the normal traffic the link speed given in the network is 1Mbit/s. ICMP flood attack is generated on five hosts h48, h49, h50, h51 and h52 as shown in Figure 13. Attack traffic has been generated using hping3 tool by using command *hping3 --flood -C -a --target-ipaddress.* This command floods the target ipadddress (web server) with ping requests. All attackers keeps generating attack traffic to the web server. We evaluate the impact on bandwidth with and without using the proposed mitigation scheme.

4.3.4.1 Attack mitigation approach effectiveness

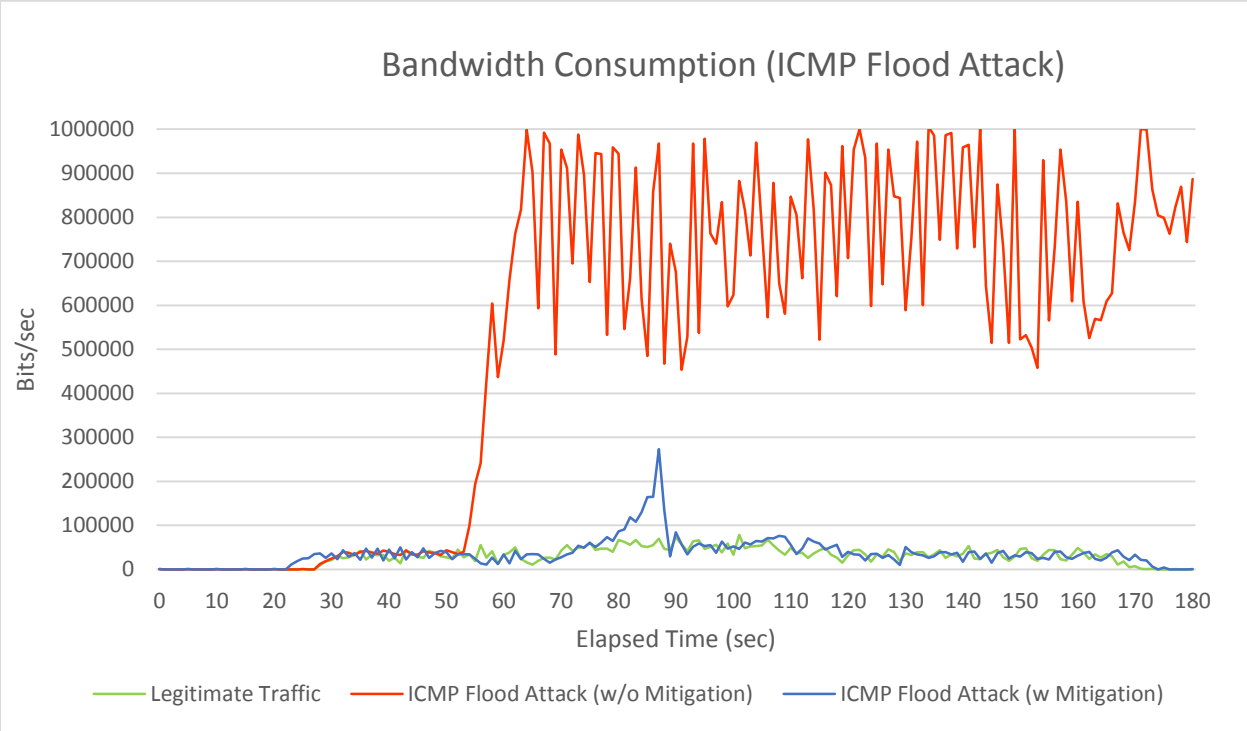


Figure 21 : Bandwidth consumption (Normal traffic, ICMP flood attack w/o mitigation, ICMP flood attack w mitigation)

The results of bandwidth consumption in Bits/sec is shown in figure 21. The figure shows that under normal traffic the overall bandwidth consumption is about 700000 Bits/sec. All five attackers start attack at 50th second. As soon as attack is started, the bandwidth consumption increased sharply to 1000000 Bits/sec which is 1Mbit/sec, thus saturating the whole link bandwidth. On the other hand using mitigation scheme, as the bandwidth consumption reaches to 270000 Bits/sec (0.27Mbit) of the total link bandwidth at 87th second, our defense mechanism

mitigate the attack by installing block flow entry in the OpenFlow switch, thus returning back the bandwidth consumption to normal state. This can be seen after 90th second in the figure.

4.3.5 UDP Flood Attack Detection and Mitigation

In this section, we discuss and analyse the results of the proposed scheme against UDP flood attack. We investigate the number of flow entries installed in OVS switch with mitigation and without mitigation scheme under UDP flood attack.

4.3.5.1 Attack Traffic

We generated UDP flood attack from two attacker hosts h46 and h47 (figure 13). Both the attackers send 100 packets/sec spoofed UDP packets to the target web server from random source IP addresses. UDP flood attack is generated using Hping3 tool using command *hping3 -i u1000 --rand-source --udp --target-ipaddress*, where –i is interval wait uX for X microseconds, e.g. –i u1000 generates 100 packets for second.

4.3.5.2 Attack mitigation approach effectiveness

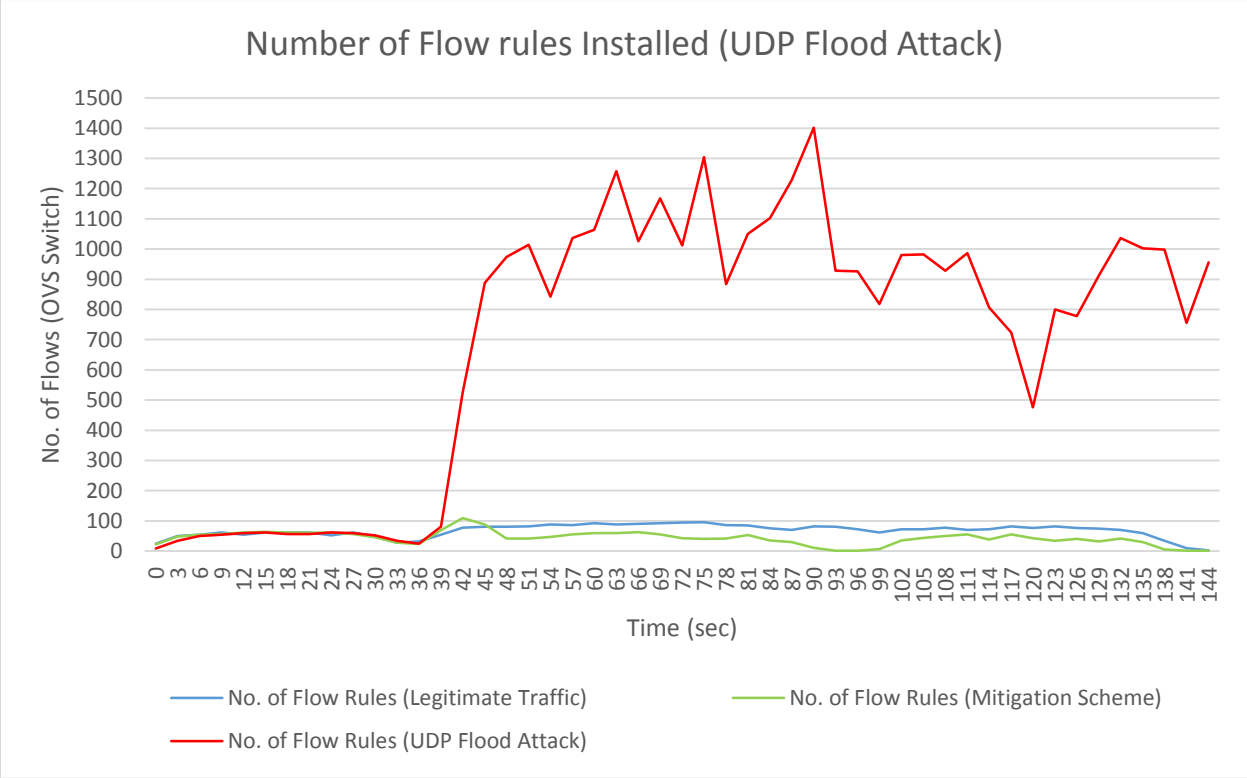


Figure 22 : Number of Flow Rules Installed – OF 1.0

The impact of UDP flood on the number of flow entries in OVS switch table is measured using time series analysis with a time window of 3 seconds. The result of this attack is shown in figure 22 in the form of number of flow entries installed in OpenFlow switch as a performance metric. From this it is clear that when UDP flood attack started at 39th second, the number of flow entries in the switch increased dramatically to 1000 and then varies between 600 and 1400. When defense scheme is enabled, the Attack Mitigator module in the controller blocked all the UDP packets targeted to the web server and allows those IP addresses that are in PAD to continue traffic. The number of flow entries in the switch are 50 approximately with mitigation scheme.

4.4 False Positive and False Negative Attack Detection

To analyze the detection performance of the proposed algorithm, we focus on two metrics: FN (false negative) and FP (false positive). We calculate the false positive (FP) and false negative (FN) occurrences in our detection results. False positive detection occurrences are those where the detection algorithm raised an alarm under normal traffic. On the other hand, false negative detection occurrences are those when detection algorithm raised no alarm when there was attacks.

To calculate false positives and false negatives for SYN flood attack detection, the algorithm is run under two different scenarios. Both scenarios consists of different normal traffic patterns with attack traffic similar to the normal traffic. In both the scenarios false positive and false negative detection rates are observed. The detection accuracy is observed under SYN traffic and ICMP traffic.

4.4.1 SYN Traffic

To test the performance of our proposed detection algorithm, we generate legitimate traffic whose traffic pattern is not stationary. We use two legitimate traffic patterns. The characteristics of the traffic patterns are as follows:

Traffic Pattern A

In this traffic pattern normal traffic is run on 45 hosts. All the 45 hosts are divided into three groups, Group A (10 clients), Group B (15 clients), and Group C (20 clients) to make scenario more realistic as possible. All three different groups generate TCP traffic by making HTTP connections to a web server with different parameters as shown in table 5 and table 6 below. When

the simulation starts, Group A clients start sending traffic to the web server. After Group A finishes Group B starts sending traffic followed by Group C. Table 5 shows the traffic pattern A parameters.

Parameters	Group A	Group B	Group C
Packet type	TCP SYN	TCP SYN	TCP SYN
No. of clients	10	15	20
Traffic type	HTTP	HTTP	HTTP
No. of Servers	1	1	1
No. of Connections per client	8	10	12
Avg. HTTP Connection duration	8 s	13 sec	10 sec
Avg. Inter-arrival time between connections	1 s	1.2 s	1.3 s
No. of Get requests per connection	5	6	7

Table 5 : Traffic Pattern A

Traffic Pattern B

In this traffic pattern, both Group A and Group B clients start sending traffic at the same time to generate burst of traffic followed by Group C at the end. We also increase the number of clients in each group by 15, 20 and 25 with a total of 60 clients. When simulation starts total of 35 hosts starts sending traffic while the remaining 25 hosts in group C are activated to send after hosts in Group A and B finish sending. Table 6 shows the traffic pattern B parameter.

Parameters	Group A	Group B	Group C
Packet type	TCP SYN	TCP SYN	TCP SYN
No. of clients	15	20	25
Traffic type	HTTP	HTTP	HTTP
No. of Servers	1	1	1
No. of Connections per client	8	10	12
Avg. HTTP Connection duration	8 s	13 sec	10 sec
Avg. Inter-arrival time between connections	0.5 s	0.7 s	0.9 s
No. of Get requests per connection	5	6	7

Table 6 : Traffic Pattern B

Attack Traffic

In order to generate attack traffic similar to normal traffic pattern A in first scenario, we issued the following hping3 command on two attacker hosts:

hping3 -i u200000 -S 192.168.56.23 -p 80 --rand-source

The above command sends 5 pkts/sec SYN packets, which will be 10 pkts/sec in total from both attacking machines, to the address of web server (192.168.56.23) on port 80 with random source

address, where –i is interval wait uX for X microseconds. In our setup, –i u200000, and so each machine generates 5 packets per second.

To generate attack traffic similar to normal traffic pattern B in second scenario, following hping3 command is used on two attacker hosts:

hping3 -i u100000 -S 192.168.56.23 -p 80 --rand-source

For each scenario 50 simulations runs are performed. Each simulation lasts around 3 minutes. While the simulation is running if an attack is suspected, an attack event is created in a log file.

4.4.1.1 Results

The algorithm performs very well in the first scenario with traffic pattern "A" without any false positive and false negative reports. Note FPR and FNR are 0 when Detection accuracy is 100%.

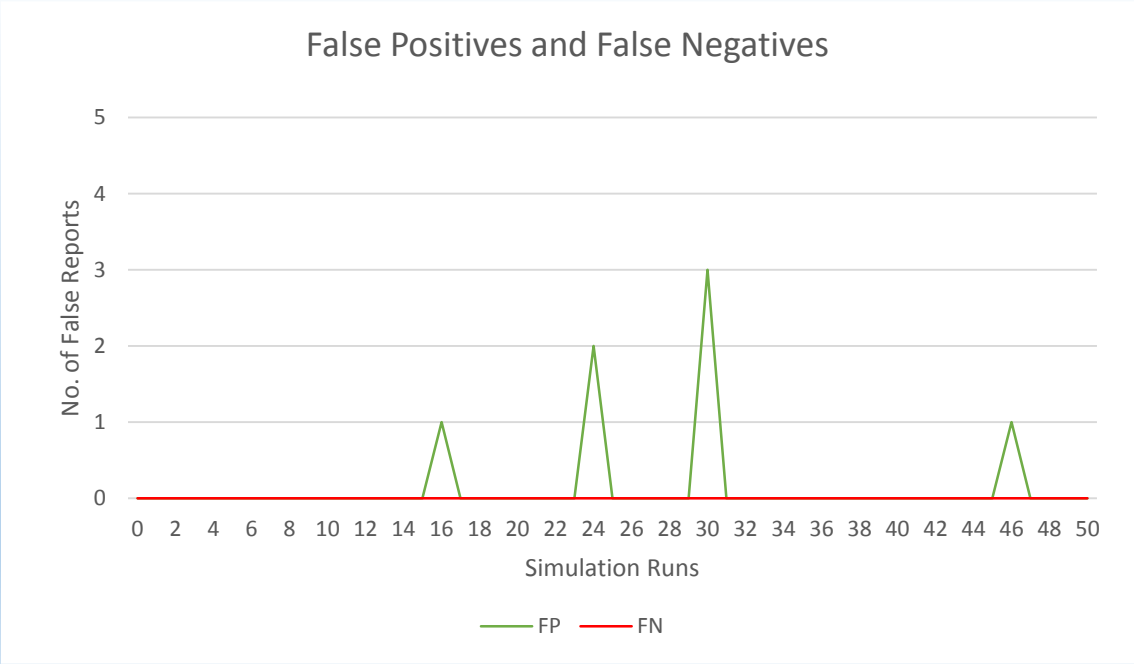


Figure 23 : False Positive and False Negative Reports

Figure 23 illustrates the false negatives and false positive reports under 10 pkts/sec attack rate with pattern B. As we can see in the above figure, when the traffic becomes heavy in traffic pattern B, a total of 7 false positive reports are detected with a false positive ratio (FPR) being 0.14%. However, no false negative reports are detected, hence FNR is 0%.

The table below summarizes the number of False Positive (FP) and False Negative (FN) reports on while traffic patterns ‘A’ and ‘B’ are tested.

	Traffic Pattern A		Traffic Pattern B	
No. of Attacks	50		50	
	Error Counts	Error Probability	Error Counts	Error Probability
FP	0	0%	7	14%
FN	0	0%	0	0%
Algorithm Detection Rate	100%		86%	
Algorithm Failure Rate	0.00%		14%	

Table 7: FP and FN reports under different traffic patterns

As we see in table 7, the results obtained in the traffic pattern “A”, show a high detection rate of 100% and the algorithm proves to perform extremely well in this case. The high rate of detection is a result of dynamic threshold and the ability of the algorithm to adjust to different traffic characteristics. Although the detection rate in traffic pattern “B” shows a little decline, the dynamic threshold update helps to keep the detection rate at 86%. In traffic pattern “B”, due to inherently bursty nature of traffic, a sudden increase of in traffic may be mistaken as an attack e.g. “flash crowd” events, where a large number of legitimate users access the same website simultaneously.

4.4.2 ICMP traffic

To test the method in terms of false positives and negatives under DDoS attack with non-spoofed IP addresses, we generate normal ICMP traffic on 45 hosts whose traffic pattern is not stationary. Again, to make traffic pattern scenario more realistic as possible all the 45 hosts are divided into three groups, Group A (10 clients), Group B (15 clients), and Group C (20 clients). All three different groups generate ICMP traffic to a web server with different parameters as shown in table 8 below. A python script is running on 45 hosts that generates ICMP traffic. We will test the approach in two cases. In the first case, similar to traffic pattern A discussed above in section 4.4.1, Group A clients starts ICMP traffic to the web server. After Group A finishes Group B starts

sending traffic followed by Group C. Where in second case, when simulation starts both group A and group B hosts starts sending ICMP traffic to destination. The idea is to generate bursty traffic in second scenario. While group C hosts activated to send after group A and group B finish.

In each simulation 5 hosts are generating attack traffic similar to normal traffic. 50 simulation runs are performed and false negative and false positives are observed in each simulation.

Parameters	Group A	Group B	Group C
No. of clients	10	15	20
Traffic type	ICMP	ICMP	ICMP
No. of Servers	1	1	1
No. of Ping requests per client	3	5	7
Avg. Inter-arrival time between each ping request	1 s	1.5 s	2 s

Table 8 : ICMP traffic

4.4.2.1 Results

In the first case, when Pattern A was used with 50 simulations runs, the algorithm does not generate any false positive and negative reports. FPR and FNR, thus, are 0 with detection accuracy of 100%.

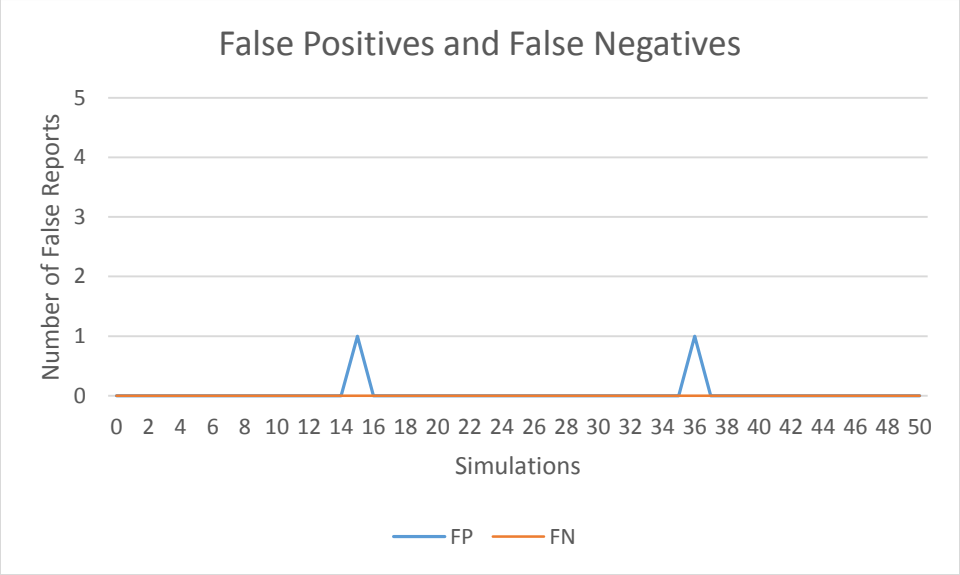


Figure 24: False Positive and False Negative Reports

Figure 24 illustrates the false negatives and false positive reports in the second case with Pattern B. As we can see, a total of 2 false positive reports are observed with a false positive ratio (FPR) being 0.04%. No false negative reports are observed, hence FNR is 0%.

The table below summarizes the results.

	CASE I		CASE II	
No. of Attacks	50		50	
	Error Counts	Error Probability	Error Counts	Error Probability
FP	0	0%	2	4%
FN	0	0%	0	0%
Algorithm Detection Rate	100%		96%	
Algorithm Failure Rate	0.00%		4%	

Table 9 FP and FN Reports under both cases

As shows in table 9 when all the clients in different groups generate ICMP traffic using traffic pattern "A", the algorithm is able to perform with a 100% detection rate. The algorithm also performs very well in traffic pattern "B". Even though the legitimate traffic is heavy still only two cases of false positive reports is seen in fifty simulation runs that results in an error probability of 4%.

4.5 Comparison with other Approaches

In this part, we compare the performance of the proposed algorithm against DDoS detection and mitigation mechanisms as given in [28], [31] and [38].

4.5.1 DDoS attack detection comparison

In [31], the author proposed FMAD, a feasible method to differentiate the attack sources from legitimate sources based on number of connections and number of packets. The author describes, the characteristics of DDoS addresses that initiate less than k connections and generate less than n packets per connection. In order to make comparison with FMAD scheme, we use the same traffic

pattern A and traffic pattern B as discussed above in section 4.4.1 to calculate false positive reports. Based on normal traffic, we set the values of k and n to 2 and 6.

Tommy Chin Jr. et al. [28] discussed a novel attack detection approach (SDN-DFDC) that inspects network packets on demand. The authors have presented two step solution for DDoS attack detection. In the first phase, anomaly in network is detected. The detection algorithm monitors the TCP SYN arrival rates by using Snort IDS and calculates mean and standard deviation to set threshold. An alert is raised by the IDS if the threshold is exceeded. In the second phase, an additional comparison is made by comparing the source IP address information found in SYN packets to the source IP addresses in the flows of OVS switch flow table. To make a fair comparison of our detection approach with the authors approach SDN-DFDC, we set the threshold to regular traffic rate plus one standard deviation. If the threshold is crossed, an additional comparison is made by comparing the source IP address information found in SYN packets to the source IP addresses in the flows of OVS switch flow table.

The comparison is done in two scenarios. First scenario consists of traffic pattern A and false positive reports are computed. Similarly, in second scenario false positive reports are observed under traffic pattern B. For each scenario 50 simulations runs are performed. Each simulation lasts around 3 minutes.

4.5.1.1 Results

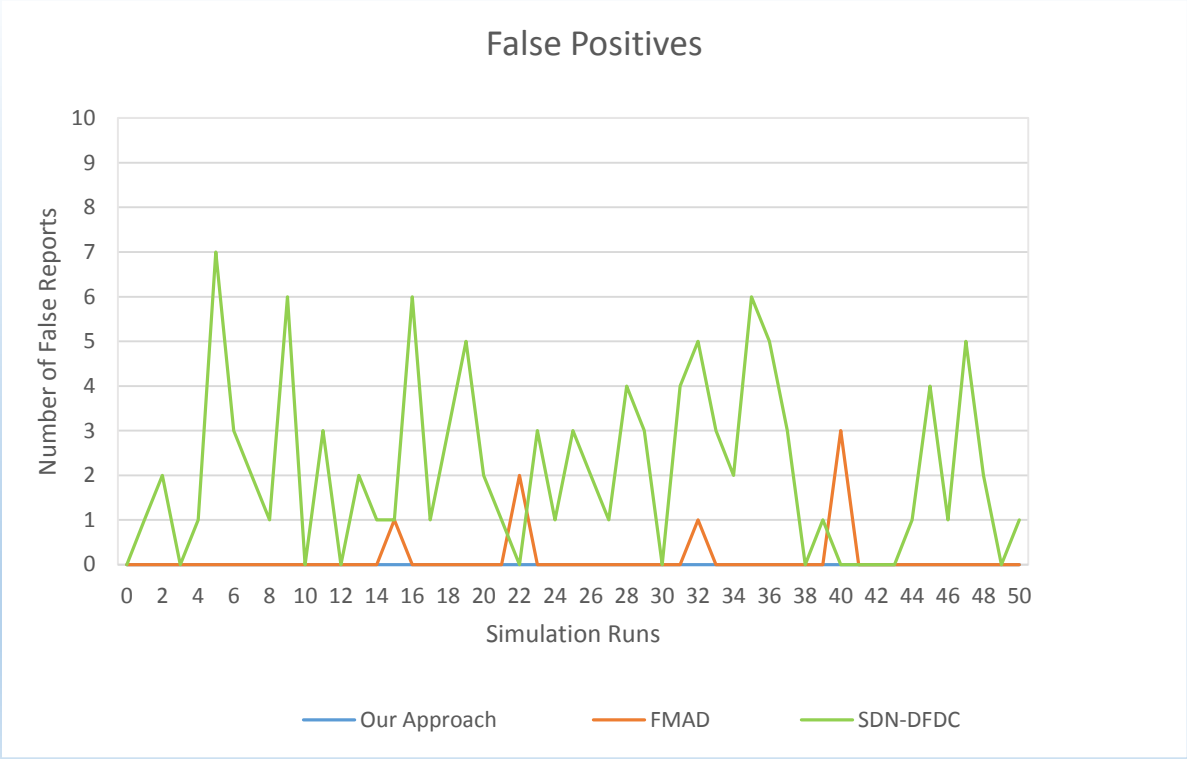


Figure 25: False Positive Reports Traffic Pattern A

Figure 25 depicts the false positive reports for the 50 simulation runs in the first scenario under normal traffic pattern A. It can be seen from the above figure, a large number of false positive reports from 0 to 7 are detected in each simulation run when using SDN-DFDC scheme. On the other hand, a total of 7 false positive reports are detected when using FMAD scheme. However, there is no false positive report detected using our proposed scheme.

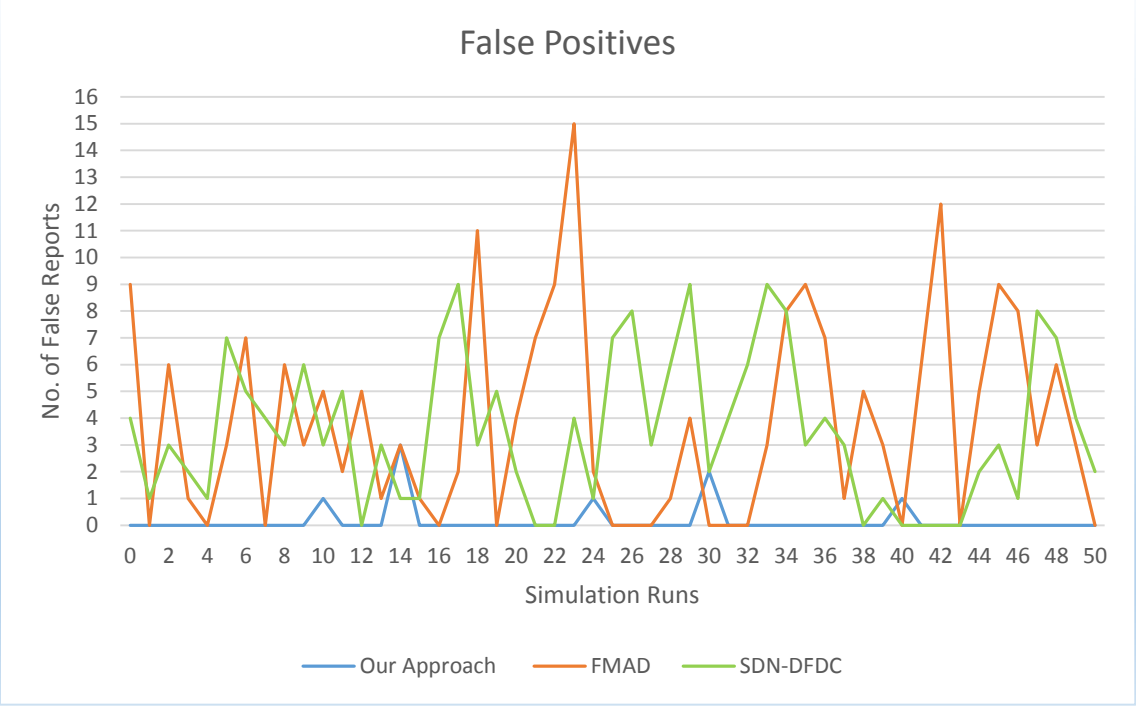


Figure 26: False Positive Reports Traffic Pattern B

Figure 26 shows false positive reports for the 50 simulation runs in the second scenario under normal traffic pattern B. The figure clearly indicates, when there is a burst of normal traffic generated using traffic pattern B, it is observed that the false positive reports detected using our proposed scheme is the smallest among the three schemes. On the other hand, the number of false positive reports detected in FMAD scheme varies between 0 and 15 in each simulation. By contrast, the number of false positives is between 0 and 9 when using SDN-DFDC scheme. This indicates that our proposed scheme outperforms the results obtained by FMAD and SDN-DFDC in terms of false positive reports.

4.5.2 DDoS mitigation comparison

In this section we will compare the proposed mitigation scheme with NICE [38]. In NICE, after an attack is detected, the network traffic from attacking port is completely blocked [34]. We will investigate the average number of TCP connections in a 3 sec window at the web server as a performance metric under SYN flood attack. Using same normal traffic pattern A described above in table 5 section 4.4.1, a total of 45 hosts accomplish legitimate SYN connections to the web server. SYN flood attack at 100 pkts/sec is generated on two attacker hosts with random source IP

spoofing. Each simulation lasts around three minutes and both the attackers start the attack after 72 seconds.

4.5.2.1 Results

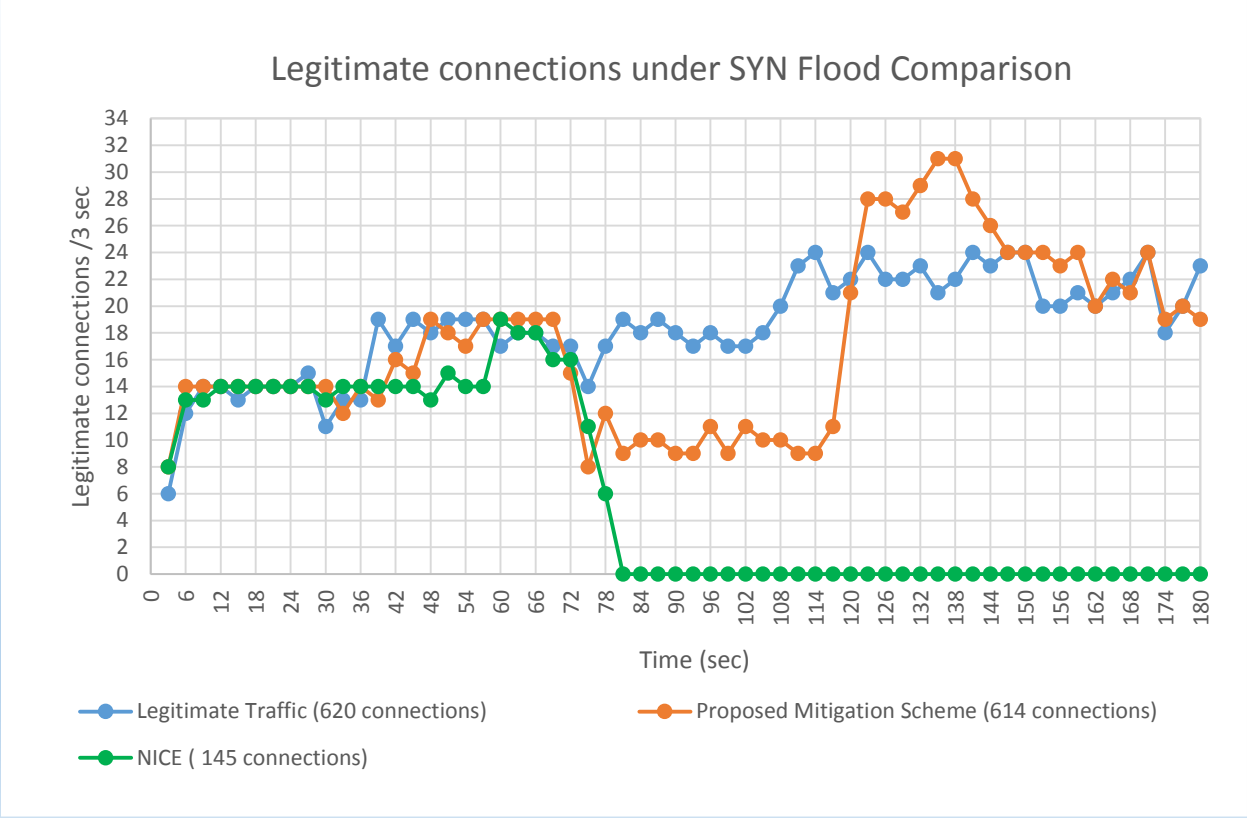


Figure 27: Legitimate HTTP connections (Comparison of Mitigation Schemes)

As the figure 27 shows, legitimate traffic starts at the beginning of the simulation. Both the attacking machines launch SYN flood attack at t=72 seconds and ending at t=114 second. At approximately t=75 second, our defense scheme comes into action and start mitigation. At this point, it is observed that an average of 10 legitimate clients can still send and receive data to and from the web server based on Source ip address filtering scheme. On the other hand, when using NICE mitigation scheme not a single host is able to make connection to the web server. This is because the mitigation scheme completely block all the incoming packets, resulting in dropping of any legitimate traffic coming to that port.

We experimentally showed the performance of our proposed detection algorithm by comparing it against other schemes FMAD [31] and SDN-DFDC [28] using different normal traffic patterns.

We showed that our proposed detection scheme outperforms than the other methods with less number of false positives. Furthermore, we also compared our mitigation scheme against NICE [38]. In this context, our mitigation scheme also outperforms with large margin.

4.6 Chapter Summary and Discussion

The results in this chapter show how the proposed defense scheme effectively detects different DDoS attacks and mitigate the attacks. We summarize the results as follows.

- The results show that the proposed scheme enhances the average number of successful connections to the web server under DDoS attack.
- The proposed algorithm limits the number of packets arriving at the controller under SYN and UDP flood attack thus protecting the controller.
- The algorithm also limits the number of flow entries installed in the OVS switch thus protecting the switches in SDN network under SYN and UDP flood attack.
- The algorithm can prevent congestion caused by HTTP and ICMP flood attacks using bandwidth control over every flow and mitigate the impact caused by badly behaved flows, thus reduce the response time of the legitimate client.

Chapter 5

5 Conclusion and Future work

5.1 Conclusion

In this thesis, we presented a SDN based lightweight DDoS defense mechanism. The objective of this thesis is to show how SDN technology can help to defend against various types of DDoS attacks. We proposed two detection and mitigation methods to defense SYN flood attack and HTTP flood attack. We implement these methods in three set of modules, the Flow-Stat-Collector that collects flow statistics from OpenFlow switches, the Flow-Stat-Analyzer module that detects anomalies in network traffic and an Attack-Mitigator that runs the attack mitigation algorithm when it is alerted by the Flow-Stat-Analyzer to mitigate an attack.

The results show that our proposed methods are able to perform well under different network traffic scenarios and are every efficient at detecting and mitigating different types of DDoS attacks with a minimal code addition to the controller program. We demonstrated the capability of these methods against other DDoS attacks such as ICMP flood and UDP attacks. One of the main advantages of the methods is that they do not block all the legitimate traffic during attack. The proposed methods are shown to be able to successfully protect the controller, servers and OpenFlow switches in SDN environment.

5.2 Future Work

Having addressed the detection and mitigation methods in a single controller network, we will consider larger experiment topologies in multi-controller environment. Also, we have tested the proposed scheme in a virtual environment with Mininet. It is hard for Mininet to support realistic IP-based network sessions. In future we will extend our work by taking into consideration real traffic scenarios and perform the experiments on real hardware switches.

References

[1] https://www.opennetworking.org

[2] Nenekazi N. P. Mkuzangwe, Andre McDonald, Fulufhelo V. Nelwamondo, "Implementation of Anomaly Detection Algorithms for Detecting Transmission Control Protocol Synchronized Flooding Attacks" *International Conference on Fuzzy Systems and Knowledge Discovery (FSKD),* 2015, pp. 2137-2141.

[3] Abdulaziz Aborujilah, Mohd Nazri Ismail, Shahrulniza Musa, "Detecting TCP SYN based Flooding Attacks by Analyzing CPU and Network Resources Performance," *ACSAT 14*, pp. 157-161. 2014.

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

[5] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, Jonathan Turner, "Openflow:Enabling Innovation in Campus Networks", March 14, 2008

[6] Wolfgang Braun, Michael Menth, "Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices", Future Internet, ISSN 1999-5903, Pages 302-336, DOI: :10.3390/fi6020302,2014

[7] Ola Salman, Imad Elhajj, Ayman Kayssi, Ali Chehab, "SDN Controllers: A Comparative Study", Conference , April 2016

[8] www.security.radware.com

[9] Pascoal T.A., Dantas Y.G., Fonseca I.E., Nigam V. (2017), "Slow TCAM Exhaustion DDoS Attack", In: De Capitani di Vimercati S., Martinelli F. (eds) ICT Systems Security

and Privacy Protection. SEC 2017. IFIP Advances in Information and Communication Technology, vol 502. Springer, Cham

[10] Wei Chen, Dit-Yan Yeung, "Defending Against TCP SYN Flooding Attacks Under Different Types of IP Spoofing," ICNICONSMCL 06, 2006, pp. 38

[11] H.Wang, D. Zhang, and K. G. Shin, "Detecting SYN Flooding Attacks" *in Proc. Conference INFOCOM IEEE Communications society, 2002.*

[12] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors". Communications of the ACM, 13(7):422-426, July 1970

[13] S. G. Hong and H. Schulzrinne, "PBS: Signalling architecture for network traffic authorization," –in IEEE Communications Magazine, vol. 51 (7), Jul. 2013.

[14] J. Mirkovic, D-WARD, "DDoS network attack recognition and defence", Ph.D. thesis, Computer Science Department, University of California, Los Angeles, and June 2003.

[15] Y. Ohsita, S. Ata, M. Murata, "Detecting distributed denial-of-service attacks by analysing TCP SYN packets statistically", *Global Telecommunications Conference (GLOBECOM)*, 2004, pp. 2043-2049

[16] Soniya Balram, M. Wiscy, "Detection of TCP SYN Scanning Using Packet Counts and Neural Network", *IEEE International Conference on Signal Image Technology and Internet Based Systems SITIS'08, 2008*, pp. 646-649

[17] Dinil Mon Divakaran, Hema A. Murthy, Timothy A. Gonsalves, "Detection of SYN Flooding Attacks using Linear Prediction Analysis", IEEE International Conference on Networks, 2006, pp. 1-6

[18] S. Mercy Shalinie, M. P. Manoj Kumar, M. Karthikeyan, J. Deepa Sajani, V. Abirami Nachammai, K. Sundarakantham, K. Narasimha Mallikarjunan, "CoDe – An Collaborative

Detection algorithm for DDoS attacks*". International Conference on Recent Trends in Information Technology (ICRTIT),* Tamil Nadu. 2011, pp. 113-118.

[19] L. Limwiwatkul, Rungsawang, "Distributed Denial of Service Detection using TCP/IP Header and Traffic Measurement Analysis," *IEEE International Symposium on Communication and Information Technology (ISCIT)*, vol. 1, pp. 605 – 610

[20] G. Zhang and M. Parashar, "Cooperative defence against DDoS attacks," Journal of Research and Practice in Information Technology, vol. 38, no. 1, pp. 69–84, 2006.

[21] D. X. Song and A. Perrig, "Advanced and authenticated marking schemes for ip traceback," in INFOCOM 2001. *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*. Proceedings. IEEE, vol. 2, pp. 878–886, IEEE, 2001.

[22] M. Caesar and J. Rexford, "Bgp routing policies in isp networks," Network, IEEE, vol. 19, no. 6, pp. 5–11, 2005

[23] Z. Qin, L. Ou, J. Liu, A. X. Liu J. Zhang, "An Advanced Entropy-Based DDoS Detection Scheme," International Conference on Information, Networking and Automation, 2010, pp. 67-71.

[24] R. B. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky, "A novel approach to detection of denial-of-service attacks via adaptive sequential and batch-sequential change-point detection methods," 2001, pp. 220-226.

[25] Tao Peng, Christopher Leckie, Kotagiri Ramamohanarao, "Proactively Detecting Distributed Denial of Server Attacks Using Source IP Address Monitoring", International conference on Research in Networking, Vol. 3042, pp 771-782, 2004

[26] Shin S, Yegneswaran V, Porras P, et al., "Avant-guard: Scalable and vigilant switch flow management in software-defined networks." In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM, 2013: 413-424.

[27] Krishnan R, Krishnaswamy D, Mcdysan D., "Behavioural Security Threat Detection Strategies for Data Center Switches and Routers." In 2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW), IEEE, 2014: 82-87.

[28] Tommy Chin ,Xenia Mountrouidou , Xiangyang Li, Kaiqi Xiong, "An SDN –supported collaborative approach for DDoS flooding detection and containment", *MILCOM, 2015 Proceedings IEEE.* Tampa, 2015, pp. 659 – 664

[29] Seungwon Shin, Vinod Yegneswaran, Philip Porras, Guofei Gu, "AVANT-GUARD: scalable and vigilant switch flow management in software-defined networks", Proceedings of the 2013 ACM SIGSAC conference on Computer & Communications security, pp. 413-424, ISBN: 978-1-4503-2477-9,November 04-08, 2013, Berlin, Germany

[30] Moreno Ambrosin, Mauro Conti, Fabo De Gaspari, Radha Poovendran, "LineSwitch: Tackling Control Plane Saturation Attacks in Software-Defined Networking", IEEE/ACM Transactions on Networking,pp 1206-1219, Volume:25, April 2016

[31] Nhu-Ngoc Dao, Junho Park, Minho Park, Sungrae Cho, "A feasible method to combat against DDoS attack in SDN network", International Conference on Information Networking (ICOIN), Cambodia, 12-14 Jan 2015

[32] Mehdi, S.,A.,S.; Khalid, J.; Khayam, S.,A.,S., " Revisiting traffic anomaly detection using software defined networking", In: Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection, pp. 161–180 (2011)

[33] Wang, R.; Jia, Z.; Ju, L, "An entropy-based distributed DDoS detection mechanism in software-defined networking", In: 2015 IEEE Trustcom/BigDataSE/ISPA, pp. 310–317 (2015)

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

[35] Braga, R.; Mota, E.; Passito, A, "Lightweight DDoS flooding attack detection using NOX/OpenFlow", In: LCN '10 Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks, pp. 408–415. IEEE, Washington (2010)

[36] Dillon, C.; Berkelaar, M., "OpenFlow (D) DoS Mitigation", Technical Report (Feb 2014). http://www.delaat.net/rp/2013-2014/p42/ report.pdf (2014)

[37] NM Sahri, Koji Okamura, "Collaborative spoofing detection and mitigation- SDN based looping authentication for DNS services".

[38] Chung, C.-J.; Khatkar, P.; Xing, T.; Lee, J.; Huang, D.,"NICE: Network intrusion detection and countermeasure", IEEE Trans. Dependable Secure Computer. 10(4), 198–211 (2013)

[39] S. Lim, J. Ha, H. Kim, Y. Kim, S. Yang, "A SDN-oriented DDoS blocking scheme for botnet-based attacks", Sixth International Conference on Ubiquitous and Future Networks (ICUFN), July 2014, Shanghai China

[40] Piedrahita, A.F.M.; Rueda, S.; Mattos, D.M.F.; Duarte, O.C.M.B., "FlowFence: a denial of service defense system for software defined networking", In: 2015 Global Information Infrastructure and Networking Symposium (GIIS), Guadalajara, pp. 1–6. (2015)

[41] https://www.incapsula.com/DDoS/DDoS-attacks/

[42] M. Team, "Mininet, "Octopress , 2015. [Online]. Available: http://mininet.org/. [Accessed 07 July 2015].

[43] M. McCauley, "POX," 2012. [Online]. Available: http://www.noxrepo.org/

[44] http://www.hping.org/hping3.html

[45] https://javapipe.com/DDoS/blog/DDoS-types

[46] https://securelist.com/DDoS-attacks-in-q1-2017/78285

[47] Muhammad Afaq, Shafqat Rehman, Wang-Cheol Song, "Large Flows Detection, Marking, and Mitigation based on sFlow Standard in SDN", Journal of Korea Multimedia Society, February 2015, Vol. 18, pp. 189-198.