DDoS MITIGATION AND FLOW TABLE MANAGEMENT IN SDN

Thesis

Submitted in partial fulfillment of the requirements for the degree of

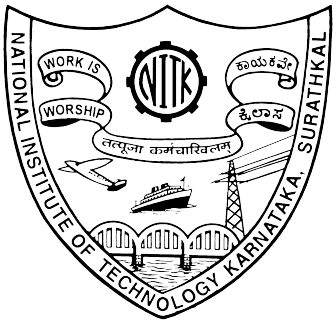
MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING
INFORMATION SECURITY

by

Azaruddhin Khan
(Reg. No.: 16IS05F)



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL
MANGALORE - 575025

JULY 2018

DDoS MITIGATION AND FLOW TABLE MANAGEMENT IN SDN

by

Azaruddhin Khan
(Reg. No.: 16IS05F)

Under the Guidance of

Vani M
Associate Professor
Department of CSE

THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING
INFORMATION SECURITY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL
MANGALORE - 575025

JULY 2018

DECLARATION

I hereby *declare* that the Thesis entitled **DDoS Mitigation and Flow Table Management in SDN** which is being submitted to **National Institute of Technology Karnataka, Surathkal**, in partial fulfillment for the requirements of the award of degree of **Master of Technology** in **Computer Science and Engineering-Information Security** in the Department of **Computer Science and Engineering**, is a *bonafide report of the work carried out by me*. The material contained in this report has not been submitted at any other University or Institution for the award of any degree.

16IS05F, Azaruddhin Khan

.....

(Register Number, Name and Signature of the student)

Department of Computer Science and Engineering

Place : NITK, Surathkal

Date :

CERTIFICATE

This is to *certify* that the Thesis entitled **DDoS Mitigation and Flow Table Management in SDN** submitted by **Azaruddhin Khan** (Register number:16IS05F) as the record of the work carried out by him, is *accepted as the P.G Major Project Thesis submission* in partial fulfillment for the requirements of the award of degree of **Master of Technology** in **Computer Science and Engineering-Information Security** in the department of **Computer Science and Engineering** at **National Institute of Technology Karnataka, Surathkal** during the academic year 2017-2018.

.....	.....
Vani M	**Dr. Alwyn R. Pais**
Project Guide	**Chairman-DPGC**
Department of CSE,	**Department of CSE,**
NITK Surathkal	**NITK Surathkal**

ACKNOWLEDGEMENT

I would really like to show my gratitude to my guide Vani M, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal for her continuous inspiration and help in completing my project work. She has continuously helped me in my work and correct me in my concepts, without which it would have been difficult to carry out this project. Whenever I had face any problem in my work, i discussed with her and she explained the problem in detail. I would like to thank her for sharing her great experience and knowledge with me.

I express my deep gratitude to project coordinator Mahendra Pratap Singh, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal for his constant co-operation, support and for providing necessary facilities throughout the M.Tech program.

I would like to say thanks all the assisting staff members of the Department of Computer Science and Engineering and all my classmates and friends. Without their consistent help, it would not have been possible to complete my project.

Place: Surathkal Azaruddhin Khan
Date: July 2018

ABSTRACT

Software Defined Networking (SDN) provides security features that were hard to develop, deploy and manage in network devices for the traditional network. The main aim of the SDN is to program the network devices present in the network and remove the vendor specificity from the network devices. The network devices in the SDN is mainly OpenFlow Switches which can act as switches, routers, firewalls and load balancers.

In networks, the attacker can attack the network devices, host and server. One popular attack is the DDoS attack which can make the servers' resource unavailable in the network. Many techniques are already presented to mitigate the DDoS attack but these techniques are having processing overhead. We propose the lightweight statistical algorithm Correlation and Shannon entropy for the mitigation of DDoS attack. These algorithms mitigate the DDoS attack by generating the threshold value for the attacker if the generated value is less than the threshold value. Our proposed scheme is CDM which stands for collection detection and mitigation.

In flow table management, flow entries are deleted because of idle time or hard time irrespective of the space present in the flow table or their reusability. If the same packet is coming to OpenFlow switch with no lookup entry in flow table then the packet is sent to the controller. Controller installs the flow rules in the flow table whether this packet has to drop or forward. The controller is remotely placed in your network and has information about the network topology information. Our aim is to exploit the idle space and minimize the table miss and overhead on the controller. We propose the modified LRU for the flow table management so that more likely frequently used entry can be placed in flow table as long as possible.

Keywords: DDoS, CDM module, SDN, OpenFlow Switch, Flow Table, Idle and Hard timeout, LRU, Idle space.

Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

In networking, switch or router handles (e.g. networking devices) the traffic of all types and based on the traffic. The router has to take the decision, what to do with traffic e.g. forward, drop or modify the traffic. In the traditional network, both the control and forward functionality is done by the router or switch only. Each and every network device has to configure separately more often, devices are vendor specific. Each vendor has their own set of rules and commands to configure the network devices. To implement the network policies there is a manual process for each and every network devices (Kreutz et al., 2015).

Therefore it is hard to manage and configure the traditional IP network because the manual configuration is required for networking devices and they are vendor specific as well. The problem in traditional IP network devices is that control plane and data plane are vertically integrated.

So we come up with new technologies named Software Defined Networking. SDN becomes an apparent or prominent technology to resolve the problem of the traditional IP network. SDN offers you an interface to program the network remotely. It means in SDN, the network is programmable. SDN allows you to remove the limitation of the current traditional network by breaking the vertical integration of control plane (decision making) and data plane (forwarding plane).

Now we can set up an interface between the network devices and SDN controller so that well-defined programming is possible. The SDN controller has direct access to all the network devices. It means we can program network devices by well-defined programming interface API. The first API in SDN technology is OpenFlow protocol. This protocol is responsible for communication between SDN controller and the Open flow switch. The OpenFlow Switch has one or more tables for packet lookups. These tables in SDN are flow tables which contains some set of rules for handling the packets. These rules are flow rules and an incoming packet can be matched with subsets of rules

in flow table and certain actions can be taken like dropping, forwarding and modifying etc.

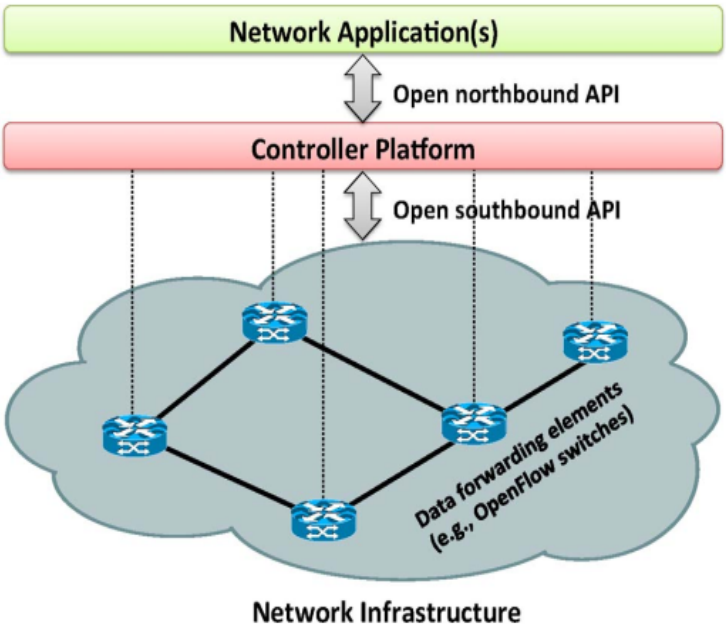


Figure 1.1: Network Architecture in SDN Environment.

1.1 WHAT IS SOFTWARE DEFINED NETWORKING

The term SDN was initially found at Stanford University to work on OpenFlow. In the initial phase of SDN, it was started to manage forwarding state in data plane remotely by SDN control plane which was decoupled from the traditional IP network. Separation of control plane and data plane means thinking capability (control logic) is removed from the routers or switches and network devices are now only forwarding devices.

According to (Shenker et al., 2011) definition of SDN consists of three key abstractions term which is *forwarding*, *distribution* and *specification*. The abstractions are the most crucial tools in computer applications to facilitate the configuration of networking devices. These abstractions are given below.

You can abstract your *forwarding* behavior from the underlying hardware and this abstraction is provided by the control plane. What type of forwarding behavior you want that can be provided by *forwarding* abstractions. OpenFlow is one type of such abstraction. This is same as the driver in OS.

The *distribution* abstraction is responsible for making all the control distribution platform into logically centralized one. For doing this distribution, layer is required in SDN. This layer provides you two things the first one is installing control commands in forwarding devices and the second one is, gathers network topology information from the underlying data plane means which device is connected to which and status of the links, up or down and this global network information is used by the network applications.

Specification abstraction allows a network application to show the desired network behavior without being responsible for implementing that behavior itself. Fig. 1.2 shows the abstractions, architecture of the SDN.

- In SDN, forwarding is based on flow instead of decision-based forwarding. A flow is nothing but the set of packets between source and destination which has the certain field in common that compose a match and certain kind of actions. All the packets which match with the same flow are going to take same network services from the forwarding device.

- Control logic which was the part of network devices in traditional network, is now moved to an external entity called SDN controller also called Network operating system. NOS is a software platform which provides you network resources and abstractions to make easy to program the forwarding devices.

- The last one is the management plane which provides you to write software application on top of the SDN controller to program the network devices.
- The network is programmable by writing application programs on the network operating system. The application program provides you to change the behavior of network devices.

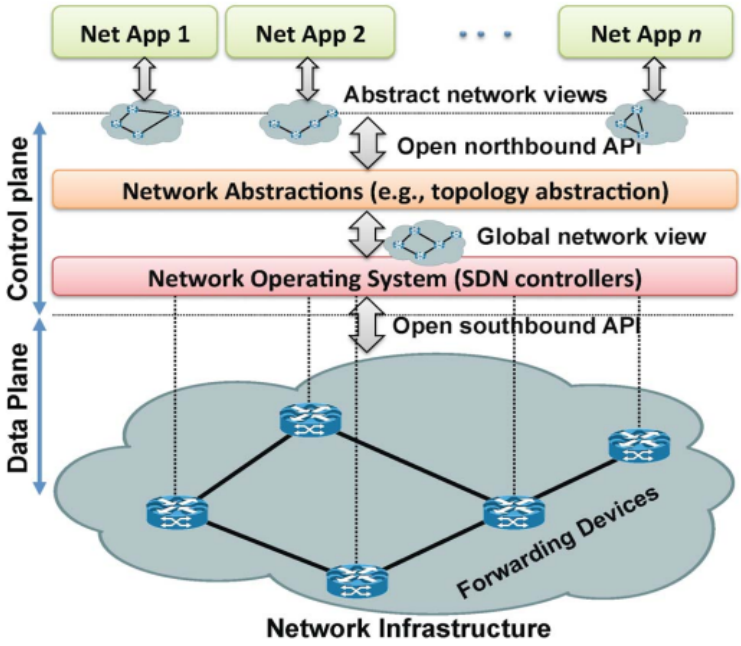


Figure 1.2: Abstraction provides by the Northbound and Southbound Interfaces in SDN.

1.1.1 Elements of SDN

- **Forwarding Devices (FD):**
  The forwarding devices can be hardware or software and they have a certain set of operations. In forwarding devices, there are set of instructions called rules that are defined by the southbound interfaces like OpenFlow (McKeown et al., 2008). These operations can be forwarded to a specific port, forwarding to the controller, modifying headers, drop and modify the incoming packet.

- **Data Plane (DP):**
  The data plane is nothing but the interconnection of networking devices. Every device in data plane is connected to the controller by a secure communication link generally. The communication links can be of any type wired or wireless.

- **Southbound Interface (SI):**
  Southbound Interface is responsible to define the communication protocol between the control plane and the data plane. There is a Southbound API which is a part of the SI defines the instructions sets in data plane devices and because of this API CP(control plane) and DP(data plane) interact.

- **Control Plane (CP):**
  Control plane is the brain of the network. The data plane devices are programmed by this control plane through the SI APIs.

- **Northbound Interface (NI)**
  This component of SDN offers the facility to Application Developer to write application programs through Network Operating System.

- **Management Plane:**
  The management plane is nothing but the set of applications which takes the advantages of the functions provided by the control plane. In this, there are so many applications like firewall, load balancers, routing.

1.1.2 Traditional Networking versus SDN

The difference between these two paradigms is the placement of control plane and data plane. In a traditional network both control and data plane is vertically integrated because of that it is hard to add new functionality. Fig. 1.3 shows the physical embedding of control and data plane in network devices and because of that coupling, it is hard to manage network application like routing algorithm because it requires a

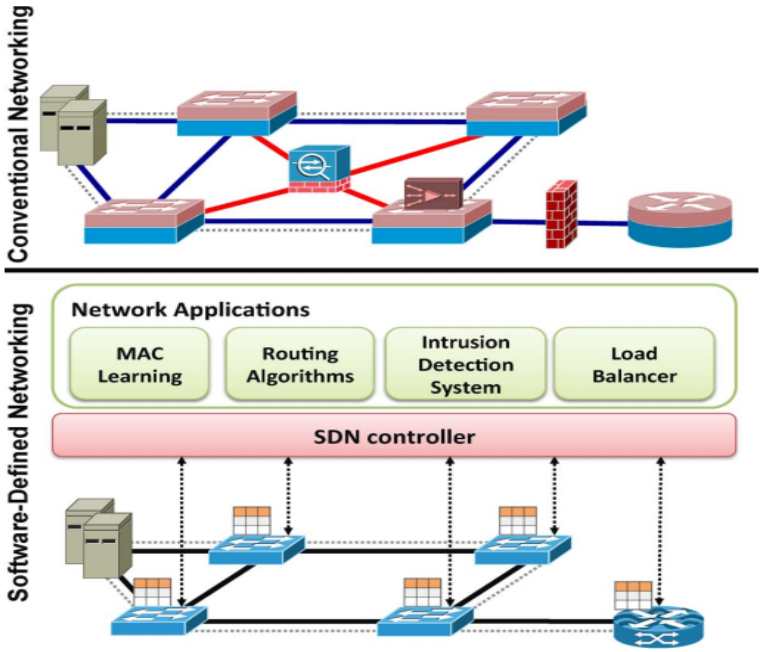


Figure 1.3: Traditional networking versus Software Defined Networking.

modification of control plane of all the network devices in case of hardware upgrades.

But in SDN, Control plane and Data plane are decoupled and Control logic is moved to an external entity called NOS(Network Operating System). There are so many benefits of SDN which are given below.

- We have the advantage to program these applications because abstraction like (e.g. forwarding, distribution and specification) provided by the control plane and programming can be shared.
- The global network view information can be used by all the network applications to develop more regular and efficient network.
- These applications can configure the network devices from any part of the network.
- The integration of network applications (e.g. firewall, NAT or routing and load balancing) can be possible in network devices.

1.1.3 Vulnerabilities of SDN

- Centralization of Controller: Everything in SDN is managed by SDN controller or we can say that decision making is done by the controller. If somehow, some of the attackers target that controller then the whole network can be collapsed. (ElDefrawy and Kaczmarek, 2016).
- Switches and Controller both use memory for their flow tables which is very limited. The name of that memory is TCAM.

1.2 OPENFLOW SWITCH

The concept of switches in traditional networking is similar to OpenFlow switch in SDN. Ethernet switch forwards the packets based on mac address only it means it works till data link layer of OSI Model and this is the only behaviour of switch in traditional network but in case of open flow switch, it works on all layer, it means now this OpenFlow switch can works like firewall, router, gateway depending on how you program the OpenFlow Switch (McKeown et al., 2008).

Inside the OpenFlow switch, there is a memory which is called TCAM (e.g. Ternary Content Addressable Memory) which is very fast, costly and very limited. This memory contains Flow Rules stored in flow Table and these rules contain some matching fields and with each rule, there is an action associated (Naous et al., 2008).

The OpenFlow Switch has at least three components.

- **Flow table** contains flow entries and with each entry action is associated which indicates how to handle the packet.
- **A Secure Channel** which tells you how to communicate with remotely placed SDN Controller and OpenFlow switches that allows you to send commands and packets.
- **An OpenFlow Protocol** which allows the SDN Controller to communicate with network devices.

1.2.1 OpenFlow Protocol:

The OpenFlow protocol which is responsible for exchanging message between OpenFlow controller and OpenFlow Switch in secure manner (Shalimov et al., 2013). This protocol is implemented on top of the SSL or TLS protocol. This protocol is also responsible to enable SDN controller to do the operation like add, delete or modify actions associated with flow entries. This protocol provides three types of messages (Hu, 2014).

- **Controller-to-Switch:** This is the message which begins from the controller to switch and in this message logical state and configuration of the OpenFlow switch is enabled. Global Network Information (e.g. topology of the network, states of links) about the whole network is known to the controller. This message is also used when incoming packet not matched with flow entries then controller tell the switch to forward to specific port.

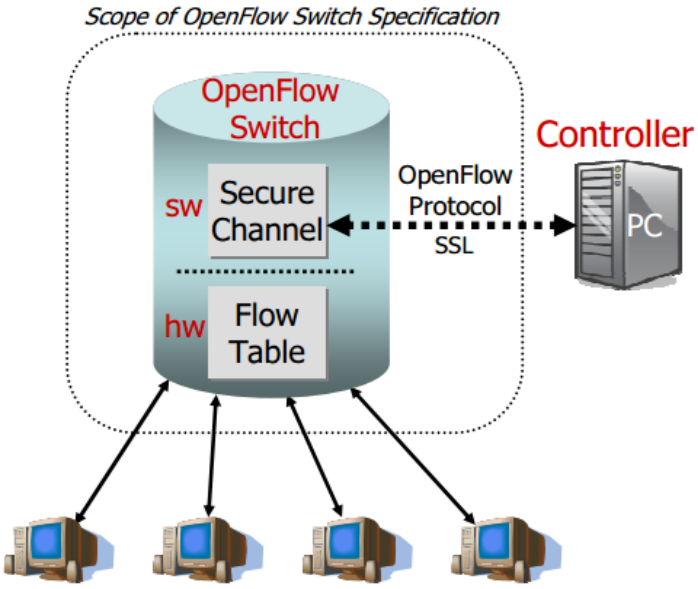


Figure 1.4: Interaction between OpenFlow Controller and OpenFlow Switch

- **Asynchronous:** In this type of message, switch communicate with the controller when the switch does not have information regarding incoming packet. In this case, only header of the packet is sent to the controller. Packet-in message is sent by this method.

- **Symmetric:** This is the message between controller and OpenFlow Switch to establish a secure communication and in this establishment, one entity sent the message without informing the other. The echo request and reply message are sent by any one of the controller or the switch. The hello message is the first message between the controller and the switch.

1.2.2 Idle and Hard Time Out:

These are the two parameters for managing the flow entries. Flow entries are going to remove by the controller when any of the time is going to expire.

- **Idle Time Out** This time is responsible for removing the flow entry from the flow table when there is no traffic is received within that period of time. If the traffic is matched before that time then *idle time* is reset again.

- **Hard Time Out** This time is responsible for removing the flow entry from the flow table whether traffic is received or not within that period of time. This is the

largest time for a flow entry to remain in the flow table.

Table 1.1: Idle and Hard Time out

Idle Timeout	**Hard Timeout**	**Behaviour**
Non-zero	Zero	Entry is deleted because of idle timeout if no traffic is received
Zero	Non-zero	entry is deleted because of hard time whether traffic is received or not
Non-zero	Non-zero	entry is deleted either of time is expired first
Zero	Zero	entry is permanent in the flow table until not deleted by controller

1.2.3 Reactive and Proactive Process in SDN

The below Fig. 1.5 shows that OpenFlow switch handles the the incoming packet in *reactive* and *proactive* process based on the packet look up in flow table. In *reactive* process, OpenFlow switch sent the packet to the controller because there is no match in flow table regarding the incoming packet so this process is slow but in *proactive* process, the packet is handled by the OpenFlow switch only because there is match regarding the incoming packet.

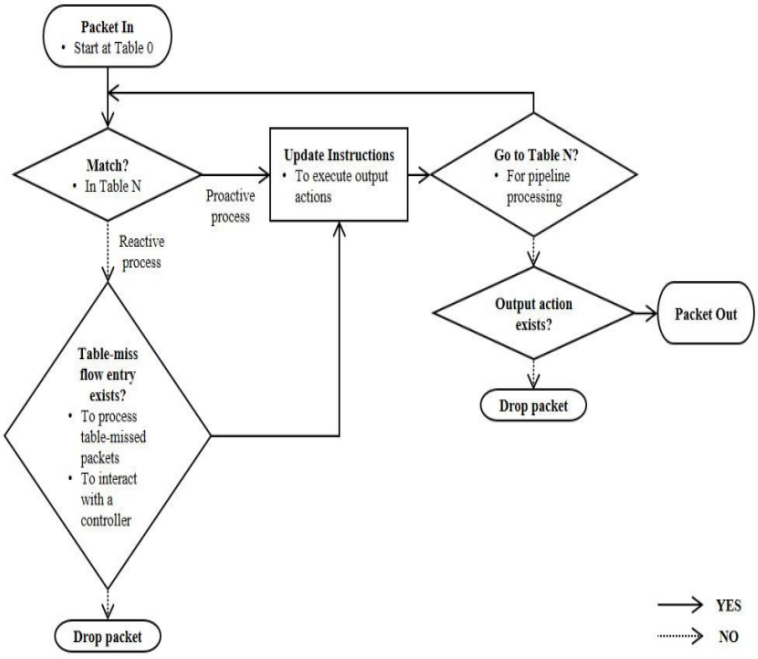


Figure 1.5: Handling of incoming packet by OpenFlow switch in proactive or reactive manner.

Fig. 1.6 shows that how the OpenFlow enabled device to communicate with NOS and what are the fields present in the flow table. According to the OpenFlow Specification, the flow table has mainly three fields which are given below.

- **Rules:** As the Fig. 1.6 shows rules made up of src_mac, dst_mac, src_ip, dst_ip etc.
- **Actions:** With each of the rule action is associated like.
  1. Forwarding.
  2. Encapsulate and forward to the controller.
  3. Drop packet.
  4. Send to normal pipeline processing.
- **Stats:** This field is used to keep the statistics about the packets.

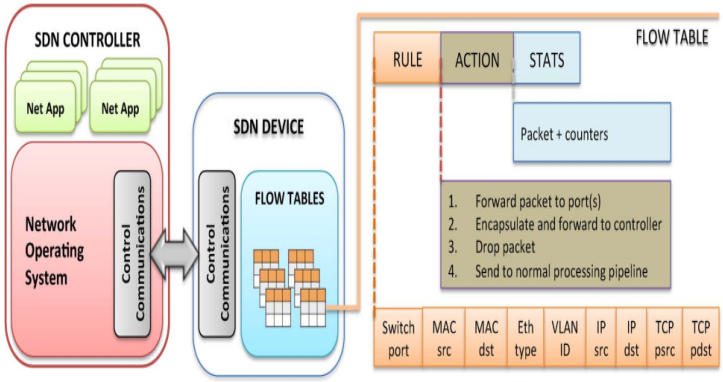


Figure 1.6: Fields in Flow Tables with OpenFlow

1.3 DDOS ATTACK

This is the attack where multiple compromised system target a server so that server runs out of resources and genuine systems run denial of service. The main target of the DDoS attack to prevent the genuine users from accessing the services from a particular server as shown in Fig. 1.7. This attack mainly consumes your desired resources and bandwidth, therefore many users which are legitimate, prevented from taking the bandwidth and services from the servers.

Analysis of DDoS attack in terms of size and number increase every year. It can be found that 250 attacks in 2011, 770 in 2012 and increases till 2018 according to some of the specialized companies.

1.3.1 Mitigation Phase

As the nature of DDoS, it can be mitigated but cannot be prevented. The effect of the DDoS attack can be minimized to some extent by configuring OpenFlow switches. The

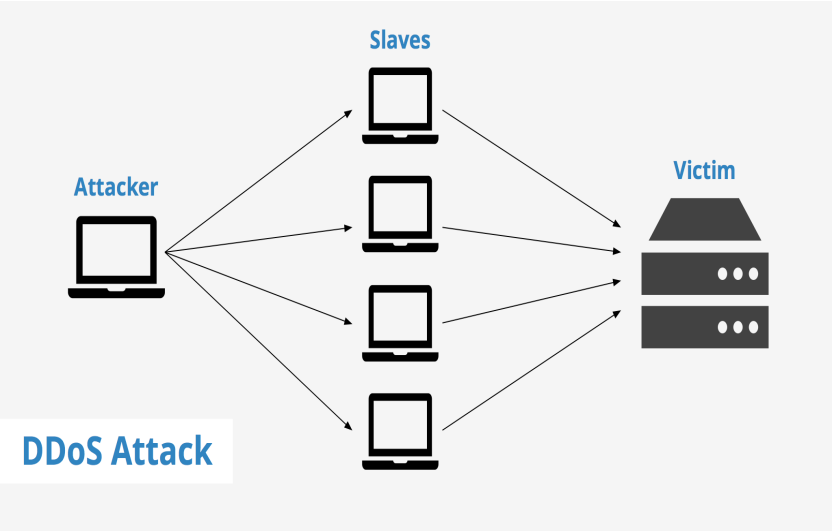


Figure 1.7: DDoS Attack

nature of the DDoS attack is unpredictable. The reason is that attacker is smarter than the System Admin of the network. There's no common pattern detected in DDoS, that's why it's hard to defend and detect, it can only be mitigated.

1.4 ORGANIZATION OF THE THESIS

The organization of this project is as follows:

- **Chapter 1: Introduction**

  This chapter gives the Introduction to The Software Defined Networking and OpenFlow Standard. How SDN and OpenFlow Standard are related to each other. In this chapter, we present the comparison of SDN and Traditional Networking.

- **Chapter 2: Literature Survey**

  This chapter gives the brief idea about the related work carried out till now in the area of DDoS mitigation and Flow Table Management in SDN environment.

- **Chapter 3: Problem Statement**

  In this chapter, we define the Problem Statement. There are two problems on which we are working. The first one is DDoS Mitigation in SDN environment and the second one is Flow Table Management in SDN environment.

- **Chapter 4: Design and Implementation**

  We propose CDM Module for the Mitigation of DDoS attack in which CDM stands for collection, detection and mitigation. These are the phases of CDM module and for the flow table management, we exploit the *idle space* present in the OpenFlow switch.

- **Chapter 5: Experimental Analysis and Results**

  In this chapter, we are creating topology of the network with some hosts, attacker and one web server. We are generating the correlation coefficient values for the genuine hosts and the attacker.

  In Flow table management, we compare the results of the simple LRU and the proposed scheme.

- **Chapter 6: Conclusion and Future Work**

  At last, this chapter summarizes the work done in this project and guides about the opportunities in this field and the future work which needs to be resumed from the current status of this project.

CHAPTER 2
LITERATURE SURVEY

Open flow is the fastest interface to get access to and manipulation between routers or switches. Control plane and data plane communicate by this protocol only (Kreutz et al., 2015). Deep knowledge about Open Flow is highly required doing project in SDN (McKeown et al., 2008). In this paper, it is given that two buildings at Stanford University used Open Flow protocol. They are also encouraging other universities to use that protocol too. Because of the OpenFlow protocol network is programmable and the action can be taken (forwarding, dropping, modifying) in OpenFlow switches. No requirement of vendor specificity for the switches or the routers in SDN.

One solution proposed in the mitigation of DDoS attack (Dharma et al., 2015) is the "Time Based DDoS mitigation and detection in SDN". Consider malicious packet for D-DOS detection and mitigation as well as time characteristics of the DDoS attack. They did not include the threshold to identify the attack's pattern so there is a requirement of threshold for detection and mitigation of DDoS attack.

Separation of control and data plane in SDN comes up with many benefits as well as introduces new attacks but writing application on top of SDN helps us to mitigate DDoS attack in cloud computing environment but SDN is also a victim of D-DOS attack (Yan et al., 2016).

In this paper "Flow-Based Security for IoT device with the help of SDN Gateway" (Bull et al., 2016) assumption is that for a small network 1.5 Mbps link is used and an attacker tries to send traffic of 2 Mbps means saturation of link. Traffic generated by the attacker is analyzed by SDN gateway so that it can be blocked and genuine host can communicate to the server.

Active Queue Management is used to mitigate Congestion based DDOS attack. AQM is used to verify that each network flow going to get its fair share of bandwidth and identify attacks based on flow so that action can be taken against them Bedi et al. (2013).

We have already seen limitation of flow table capacity in OpenFlow switches. To satisfy all the requirement of an enterprise we require more flow entries than the flow table capacity. So now to solve this problem several work has been proposed which is given below.

According to (Curtis et al., 2011) and (Lee et al., 2013), the incoming traffic is classified in elephant flows and mice flows in OpenFlow Switch. An OpenFlow switch selectively communicate with SDN controller in their proposed scheme and with their scheme, interaction with OpenFlow controller significantly reduces, means we can say that table hit is more and table miss are less. These two models Curtis et al. (2011) and Lee et al. (2013) for caching flow entries are slightly different. According to Curtis et al. (2011) the mice flows are groups according to set of least energy paths before being added to flow table and rules for elephant flows are separated individually. According to Lee et al. (2013) the cached flow entries are handled separately and OpenFlow switch does not share the cache memory for elephant and mice flows. In both the techniques and models for managing flow, table are significantly improved by classifying traffic according to their flows pattern, but there is a requirement of training sets to classify traffics.

According to Yu et al. (2010), switches which are having authority, have the placement between OpenFlow controllers and OpenFlow switches on behalf of the OpenFlow controller. Duplicate flow rules are placed in the authority switch by interacting with SDN controller. By doing so, OpenFlow switch does not increase overhead on the Controller and table misses are now reduced. As of now, there is no concept of authority switch so this technique requires the large change in existing OpenFlow switch and model of SDN.

According to Zhu et al. (2015) and Kim et al. (2014), their proposed scheme is to dynamically change the idle timeout and hard timeout of the flow entries according to flows coming in OpenFlow switch. For the repeated flows, the idle timeout going to be large and for remaining flows idle timeout is going to less. This dynamic adjustment is done by the controller by collecting traffic from the OpenFlow switch and it has to decide which one is periodic flow and which one is not. The controller has to insert periodic flows with high idle timeout to prevent early deletion of the flow entry from the flow table.

As we have already seen caching techniques to increase the performance in terms of table miss and idle timeouts and we compare several page replacement algorithms like FIFO, LRU and Random page replacement then we say that LRU is the best algorithm which decreases the table miss compared to other algorithms Zarek et al. (2012). So in our proposed scheme, we apply an LRU algorithm on expired flow entries if space is available in flow table and overhead is decreased on the controller. This is the simple technique that we are dividing the flow entries into active and inactive entries and we are able to decrease the table misses.

CHAPTER 3
PROBLEM STATEMENT

Decoupling of control and data plane in SDN comes up with many advantages and it is an emerging technology. We can manage and program the networking devices from the SDN controller. As we know that the SDN controller is centralized in the network, so it can be targeted by the attackers by sending spoofed IP addresses and the whole network can be collapsed and the same thing can be done with the web server present in the network, it can also be targeted by the attacker and the web service is unavailable to all the genuine users. Several techniques have been proposed to mitigate the DDoS attack but they require training of collected data that results in overhead on the OpenFlow switches. So for this type of attack (i.e. DDoS), we are proposing the CDM module which collects the data, analyzed the data and based on the analysis, we mitigate the the DDoS attack by using some lightweight statistical algorithm.

Management of the flow table in SDN is required to decrease the table miss because when there is no matched entry in the flow table regarding incoming packet, then in the case OpenFlow switch need to communicate with SDN Controller to ask what to do with the packet. So somehow we have to manage flow table. In flow table, there are *idle* and *hard timeout* associated with each flow entry and entries are deleted when any one of these two time expired disregarding *idle space* (e.g. empty space) and if the new incoming packet matched with recently deleted entry then there is table miss, which increases overhead on SDN Controller because now OpenFlow switch communicate with SDN controller. So we propose the scheme in which we divide our flow table entries in three parts which are *active* entries and *inactive* entries and *idle space* in OpenFlow switch and we apply LRU algorithm first on *inactive* entries and then we apply LRU algorithm on *active* entries in case, flow table is full.

CHAPTER 4
DESIGN AND IMPLEMENTATION

In this project, we detected and mitigated DDoS attack by using statistical algorithm e.g. correlation and co-variance in SDN environment. So first we will see the introduction of DDoS attack and some of the statistical algorithm, their definition and their formula.

4.1 FLOOD BASED DDOS ATTACK

Some of the DDoS attacks can be classified as flooding, brute force attack according to the weakness. We mainly focus on flood-based DDoS attack and it can be of any type like TCP SYN flooding, UDP, ICMP and DNS flooding. Also, they can use multiple compromised client with spoofed IP address and they send so many requests to a targeted server and this server responds to all the request until its all resources are exhausted (Mirkovic and Reiher, 2004). There is some variation of DNS flooding and DNS amplification, they change source address to targeted source address and all the request goes to a targeted server increases the bandwidth in the network (Meitei et al., 2016).

In the TCP three way handshaking, client ans server establish a connection by exchanging some kinds of packets like SYN, SYN-ACK, ACK and connection is established as shown in below Fig. 4.1 and they are ready to transfer the data.

These flags are given below.

- **SYN Flag**
  This flag is sent by the client to initialize the the connection with the server.
- **SYN-ACK Flag**
  This is the flag is sent by the server to acknowledge the client and initiate the connection with client.
- **ACK Flag**
  Sent by the client to acknowledge the server.
- **FIN flag**
  It is used to close the connection by the client.
- **FIN-ACK**
  It is used by the server to acknowledge the client and also initiate the close connection to client.

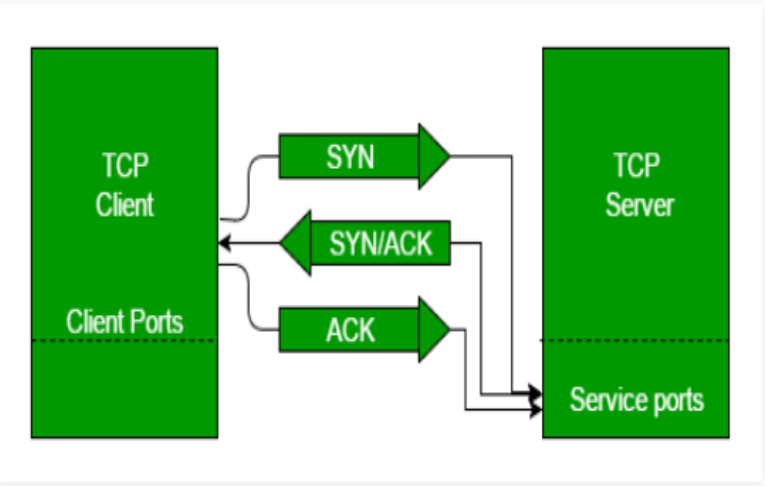


Figure 4.1: Three way handshaking

But one attack is possible which is TCP SYN flooding. In this attack, attacker sends many of the SYN packets and does not send acknowledgement back and attacker reserve the resource for so many connection and after some time server runs out of resources.
SDN controller has the ability to build the complete configuration and state of the network and also able to detect and mitigate the DoS and DDoS and controller send some temporary rules into the OpenFlow switches to drop the traffic regarding malicious flows. A controller has the ability to remove the temporary rules when the attack stops so that memory can be freed and load on the forwarding devices can be reduced.
In our work, we present a CDM module that is used to detect and mitigate the DDoS attack. We are mainly focused on the TCP SYN flooding because this attack more used by the attacker and it has been seen 23% of the attacks registered in (Godlovitch et al., 2015).

4.2 STATISTICAL ALGORITHMS

4.2.1 Co-variance

Co-variance is used to measure the change in one variable is related to change in another variable or we can say that it is the degree to which the two variable is associated. It is same as variance but the difference is, the variance is the measure of how one variable varies and the Co-variance is the measure how two variables are varied to each other. It means variance is stick to one variable only and Co-variance is stick to two variables. The formula for Co-variance is

$$Cov(X,Y) = \frac{\sum_{i=1}^{n} (x-\bar{x}) * (y-\bar{y})}{n-1}$$

where
x = the given value in data set
y = the given value in data set corresponding to y
$\bar{x}$ = mean of data set of x
$\bar{y}$ = mean of data set of y
n = number of data points in a sample

4.2.2 Correlation

Correlation is the measure of the strength of linear relation between two variables. The value for the correlation is ranges from -1 to +1. The correlation is the scaled version of the co-variance.

To understand the the correlation, we take two variable, one is the number of hunters in a region and second is the deer population. If the number of hunters increases then definitely number of deer population decreases, therefore this is the **Negative Correlation**. For the **Positive Correlation** if one variable increases then another variable should also be increased, means both the variable should react in the same way. The formula for the Correlation

$$Correlation = r_{xy} = \frac{Cov(X,Y)}{S_x * S_y}$$

where S_x = Standard deviation of X
S_y = Standard deviation of Y

The other formula for the correlation is

$$Correlation = r_{xy} = \frac{\sum {}_zX_i * {}_zY_i}{n-1}$$

where

$$ {}_zX_i = \frac{X_i - \bar{X}}{S_x}$$

$$ {}_zY_i = \frac{Y_i - \bar{Y}}{S_y}$$

${}_zX_i$ and ${}_zX_i$ be the standardized versions of X and Y, respectively, that is, zX and zY are both re-expressed to have means equal to 0 and standard deviations (s.d.) equal to 1. According to the (Ratner, 2009) the correlation values over the range -1 to +1 are divided.

- If the correlation is zero it means that no linear relationship.
- +1 of correlation value means **perfect positive linear relationship**. It means, if one variable increases the other variable is also increases in exact same manner by linear rule.
- -1 of correlation value means **perfect negative linear relationship**. It means, if one variable increases the other variable is decreases in exact same manner by linear rule.
- correlation value between 0.3 and 0.7 (-0.3 and -0.7) means **moderate positive (or negative) linear relationship**.
- correlation of value between 0.7 and 1.0 (-0.7 and -1.0) is the **strong positive (or negative) linear relationship** via a linear firm rule.

4.2.3 Shannon Entropy

Shannon entropy is nothing but the randomness of variables (Mousavi and St-Hilaire, 2015). If one of the flags is increases then entropy is going to decrease. Entropy is going to calculate based on the probability of TCP flags.

$$entropy = -\sum_{i=1}^{n} P_i * \log_2 P_i$$

entropy has the highest value when all the flags are going to have same probability and decreases when the probability is going to different-different for the TCP flags.

4.2.4 Detection and Mitigation

The tool for the detection and mitigation should satisfy the following requirements.

- Automation: The tool should be automated so that it can check incoming traffic

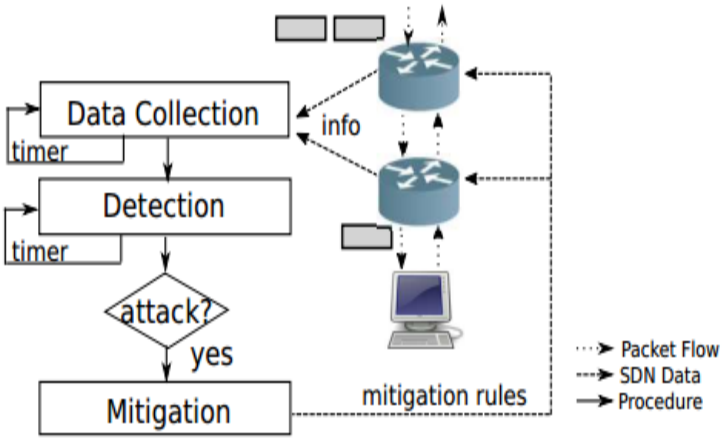


Figure 4.2: Phases in CDM module which stands for Collection, Detection and Mitigation.

regularly and store the required data and DDoS attack can be detected and mitigated and after stopping the attack it should come in normal behavior again.

- Flexible: The designed tool should allow a network administrator to select one of the detection algorithm and change whenever it is required.

As in the above Fig. 4.2, CDM module has the three phases in which first phase is the collection which calls the *receive()* method to handle the incoming packets and collects data for each host present in the network. The second phase is detection which uses the *statistical* algorithms to generate the thresh-hold value. If detection phase indicates an attack then in the mitigation phase, flow rules are created and sent to the Openflow switches to drop the malicious traffic. Now we see every phase in detail.

1. ***Collection Phase:*** In this phase, we focus on the 3-way handshaking of TCP and what are the packets exchanged between the client and server to establish the connection. So packets SYN, SYN-ACK, ACK are handled during TCP connection establishment and packets FIN, FIN-ACK and ACK are exchanged between client and server.

   So in the collection phase, we are going to maintain the statistics of the TCP flags in the incoming flows on the network. Therefore, regarding each host statistics of the TCP flags is maintained.

   To implement the collection phase we created two java classes named **HeaderExtract** and **ServerInfo**. The class **HeaderExtract** is used to extract the incoming packets and the second class is used to keep counters and active connections from the TCP packets. First payload of data link layer is extracted

and then from that payload, IP packet is extracted and from the IP payload, TCP segment is extracted and from the TCP segment, required data is collected like TCP flags.

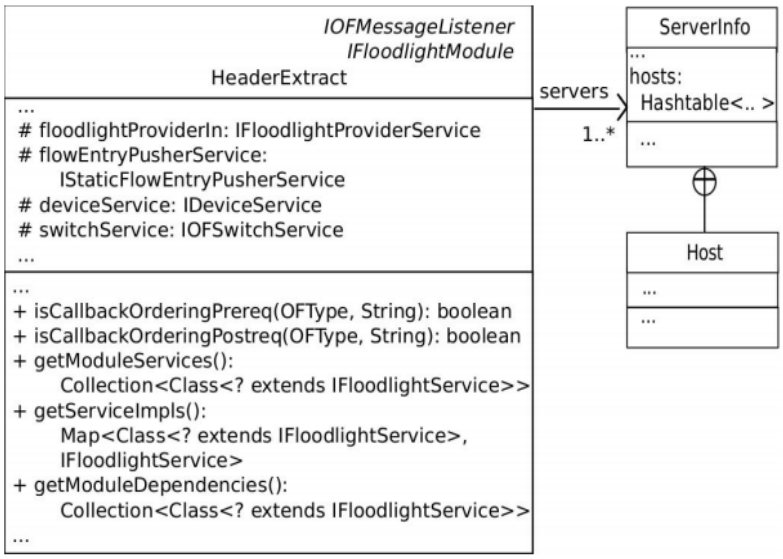


Figure 4.3: Interfaces used in Floodlight Controller to communicate with OpenFlow Switches

Apart from that two java interface is also required named *IFloodlightModule* and *IOFMessageListener* as mentioned in Fig. 4.3. The first interface allows controller to find new class as a module and the second one is used to analyze the packets.

The algorithm for receiving the packets.

Algorithm 1: receive(packet)

```
1 if (isTCP(packet) and flagTocheck(packet) then
2 |   srcIP = TCPpacket.getSrcIP()
3 |   dstIP = TCPpacket.detDstIP()
4 |   server = lookupProtectedServer(srcIP,dstIP)
5 end
6 if server is not null then
7 |   processPacket(server,TCPpacket)
8 end
```

Above receive function runs every time whenever a packet comes, then the

protocol of the incoming packet is checked, if protocol mentioned in the packet is TCP, then it has TCP Flags (e.g. SYN, SYN-ACK, ACK, FIN, FIN-ACK, ACK). If the packet goes to the Web Server or coming from the Web Server present in the network then statistics of the TCP flag is maintained by running the processPacket for a client.

We also created an independent thread for sending collected data to detector every five seconds and also send a signal to start detection analysis every 20 seconds.

2. ***Detection Phase:*** In the detection phase, data collected in the first phase are taken by the algorithm either by the *correlation matrix* or the *Shannon entropy*. These algorithms analyze the collected data and generate a threshold value for the mitigation phase.

   Correlation Matrix: This the algorithm which is used to find the dependency among the TCP Flags. For the valid TCP session, SYN, SYN-ACK and ACK have the strong relationship or we can say one to one relation. In the same way, the FIN, FIN-ACK and ACK have the strong relation or one to one relation. For the strong relationship between two variable have the correlation of value of +1 refer section 4.2.2. To calculate correlation among more than two variables, we require two-dimensional matrix to put all the TCP flags and correlate each variable with every other variable.

   In two dimensional array, the columns are TCP Flags which are SYN, SYN-ACK, ACK, FIN. So our two-dimensional matrix is a square matrix of dimension 4X4. Then we pass that matrix into Co-variance class and Correlation Coefficient and the threshold value is generated for each TCP flag with every other flag. If any of the threshold value is less than 0.7, we treated that host as attacker and value greater than 0.7 indicates the host is genuine.

   Shannon entropy : In this algorithm, we are mainly focused on the SYN, SYN-ACK, and FIN flags with same probability (1/3) and we are applying the Shannon entropy formula $H(x) = -(3 * 1/3\ log_2(1/3))$. H(x) should be approx to 1.583 for the valid TCP connection. H(x) should be decreased if one of the TCP flags increases. If H(x) value is less than 1.31 then detection phase in the CDM module generates an alert for the mitigation phase.

   - **Choosing the Threshold Value**

     The range of the correlation values is $\left[\text{-}1,+1\right]$. The correlation between 0.7 and 1 indicates a **strong positive linear relationship** and 0.3 to 0.7 indicates a **moderate positive linear relationship** refer section 4.2.2. Therefore we have 0.7 is the common value for the moderate and strong linear

relationship. Suppose if we take the 0.8 as a threshold value which is the strong linear relationship then some of the genuine users can be the attacker and if we take 0.6 as the threshold value which is the moderate linear relationship then some of the attackers can be genuine users.

In case of Shannon entropy the thresh-hold value is 1.31 (Mousavi and St-Hilaire, 2015) based on experimental analysis. They are calculating the rate of attack packets according to the formula.

$$R = \frac{P_a}{P_n} * 100\%$$

Where R is rate of attack packets to normal packets. P_a and P_n number of attack packets and number of normal packets respectively.

They are assuming that out of 100% traffic 25% traffic is of attacker. They are choosing the highest entropy value as threshold of all the cases.

3. ***Mitigation Phase:*** In the mitigation phase, if there is an alert for the attack to happen the CDM module start to block the malicious traffic. First, the information about the registered client is checked, particularly checks unfinished three-way handshaking as they are the indicators of the packets with spoofed IP addresses and second it generates mitigation rules to drop the traffic from the malicious hosts and at lats, the Floodlight uses the interface **IStaticFlowEntryPusherService** to sent the rules to OpenFlow Switches. After the attack has stopped, The CDM module deletes the flow rules from the OpenFlow Switch to free up space.

4.2.5 Tools and Commands

- **Controller** : Floodlight
- **Tools** : Mininet, hping3, Eclipse.
- **Protocols** : Open Flow Protocol in Open Flow Switch.
- **Language** : Java.
- **Commands** :
  1. This the command for the mininet emulator to create the topology. This the tree topology with height 2.
     sudo mn –controller=remote, ip=127.0.0.1, port=6653 –topo=tree, fanout=3, depth=2–switch=ovsk,protocols=OpenFlow13
  2. **sudo mn –controller=remote, ip=127.0.0.1, port=6653 –topo=tree, fanout=5, depth=1 –switch=ovsk, protocols=OpenFlow13**
  3. Command to create linear topology with switch are connected in line and has one host connected to each host.
     sudo mn –test pingall –topo single,3
  4. To run pox Controller with learning switch
     ./pox.py forwarding.l2_learning
  5. This is the command for running a web server on a host present in the network.
     python -m SimpleHTTPServer 80
  6. This is the command to request a web page from a web server.
     wget -O - 10.0.0.1
  7. This is the command to dump the flows present in flow table of open flow switch.
     sudo ovs-ofctl dump-flows s1
  8. The command to send so many SYN packets to a server.
     hping3 -i uX -S -p 5001 10.0.0.1
     where i is used to denote interval
     S denotes SYN packets
     p denotes port number from which packet has to send
     uX means it has to wait for X microseconds to send each packet.
  9. Command to install flow in ovsk switch
     sudo dpctl add -flow tcp:127.0.0.1:6633 in_port=1, actions=output:2

4.3 FLOW TABLE MANAGEMENT

As we have already discussed the flow table and how the packet is handled in OpenFlow switch in section 1.2.3. The management of the flow entries are done by idle timeout and hard time out irrespective of the reusability of the entries. Now we discuss to manage the flow table in efficient manner without the programming code complexity.
As in the below figure 4.4, the flow table space is divided into two parts, one is Idle space (i.e. empty space) and the second one is active entries (i.e. used space by the flow entries). Time (e.g. idle time and hard time) is associated with each of the flow entries. When this time expires the entry has to be deleted even if we have enough space in the flow table. If the same entry comes again then, we have to ask from the controller for that entry. So table miss and overhead on the controller increases.

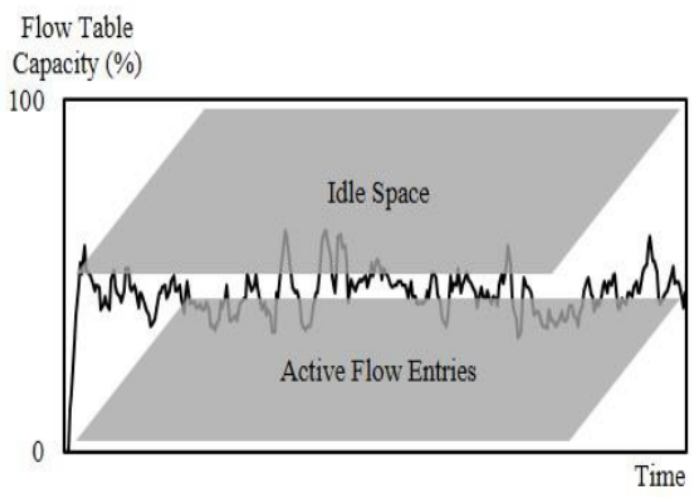


Figure 4.4: Traditional Approach to cache the flow entries

Now we propose a scheme in which we exploit the idle space present in the flow table. Therefore we divide the flow table into three parts as mentioned in Fig. 4.5.

- Active entries
- Inactive entries
- Idle space (remaining space after exploiting the Idle space).

So our intention is to have the flow entries into flow space as long as possible. When the entry is expired we move that entry into the inactive entry till we have idle space in open flow switch. So we propose a scheme in which certain operation has to consider to achieve the desired flow table management. So operations are given below.

1. If the flow table size is less than the max size then.
    - Idle and hard time out of each entry are decreased every second.
    - When this time is expired, the entry is moved to *inactive_list*, the *active*

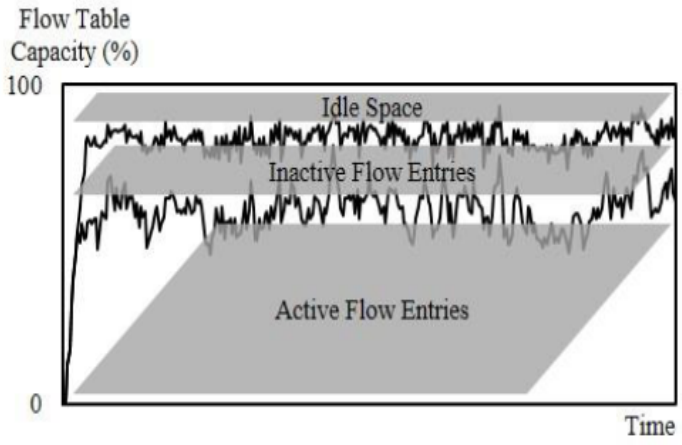


Figure 4.5: Proposed scheme to cache the flow entries

field is set to zero (denoting entry is *inactive*) instead of removing from the *flow_table* and we remove it later from the *inactive_list*.

- So we have a mix of the *active* and *inactive* entries in the flow table.
- It means the new incoming entry can be matched with either the *active* entries or *inactive* entries.
  (a) If matches with the active entries then idle time is going to reset and *active_list* is maintained appropriately (i.e. in LRU fashion).
  (b) If matches with the *inactive* entries then the entry will become *active* entry again with the fresh idle and hard time out and entry is moved from the *inactive_list* to *active_list*.

2. If the flow table size is full then.
   - In this case, we start deleting *inactive* entries from the flow table in LRU fashion when new flow entry came to OpenFlow switch because we have already maintained a list for the *inactive* entries (e.g. *inactive_list*). In *inactive_list*, we maintain least frequently used entries on the front end and most frequently used entries on the rear end.
   - Otherwise we have to delete from the *active* entries from the flow table in same LRU fashion because we have already maintained an *active_list*.

The above explanation shows that how to handle incoming flow entries and dividing them into *active* and *inactive* entries. The algorithm is given below.

Algorithm 2: Handling of entries which are coming to OpenFlow switches

```
1  while entry is coming to OpenFlow switch do
2      if entry is present in flow_table then
3          if entry is active then
4              reset the idle_time
5              remove and append entry from active_list
6          else
7              reset the idle_time and hard_time
8              remove entry from active_list and append into inactive_list
9          end
10     else
11         if flow_table is not full then
12             insert the new entry in flow_table and append in the active_list
13         else
14             increment the pagefault by one
15             if inactive_list is not empty then
16                 remove one entry from flow_table and the front end of the
                     inactive_list
17                 insert the new entry in flow_table and append in the active_list
18             else
19                 remove one entry from the flow_table and the front end of the
                     active_list
20                 insert the new entry in flow_table and append in the active_list
21             end
22         end
23     end
24 end
```

The *update()* method operation are given below.

- For the *active* entries, every second we decrease the *idle time* and the *hard time* one second from their value. If any one of the *idle time* and the *hard time* is zero, whichever come first then entry is moved from the *active_list* to *inactive_list*.
- Otherwise we have to *update* the *idle time* and the *hard time* of the flow entries.

The algorithm for the Update function is given below.

Algorithm 3: Update() method

```
1 for each entry in flow_table do
2     if entry is active then
3         decrease the idle_time and the hard_time by one
4         if flow_table is not full and (idle_time ≤ 0 or hard_time ≤ 0) then
5             reset the idle_time and hard_time
6             move entry from active_list to inactive_list
7         end
8     end
9 end
```

CHAPTER 5
EXPERIMENTAL ANALYSIS AND RESULTS

Mininet is the network emulator tool to represent the whole network as a realistic virtual network. The topology of the network can be created on the mininet by running the command with instances of OpenFlow switch refer section 4.2.5. In mininet router and switches are represented as OpenFlow switch and services are the hosts. The environment runs on a machine with Ubuntu 16.04 LTS, 8 GB RAM, and an Intel processor at 3.4 GHz and eight cores. Figure 5.1 shows the topology of the network. This topology is created by the Miniedit present in the Mininet.

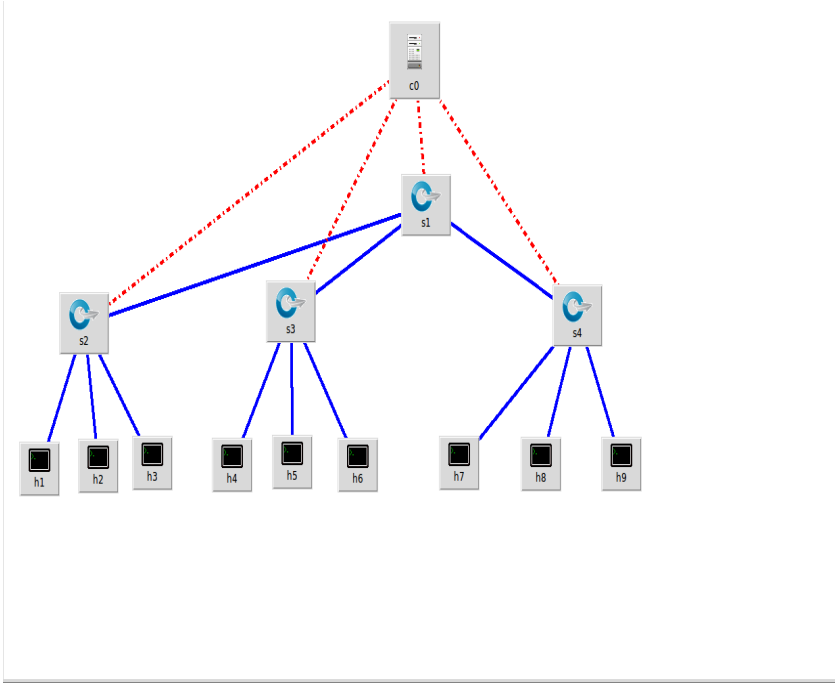


Figure 5.1: Tree topology of the network with one server, two attackers and some legitimate hosts

In the above topology, there are 9 hosts and four OpenFlow switches in which *s1* is connected to *s2*, *s3* and *s4* and three-three host are connected to each of *s2*, *s3*, and *s4*. In this network, we are assuming that host h8 are registered web server and h1 and h4 are attackers and all the other host are legitimate users. All the legitimate hosts want

the web services from the registered Server h8 and h1 and h4 which are the attackers, are trying to stop the legitimate user from accessing the web services present in the network.

In CDM module, we maintained an individual thread for each and every host and for every 5 seconds we are collecting the data and send that data every 20 seconds to detection phase regarding each host. Our detection phase takes a very few time approx 60 ms. to 180 ms. to generate the threshold value. After the mitigation phase, the OpenFlow switch drops the traffic regarding the attacker. After the mitigation, the genuine host is able to take the request from the web server in 15 sec. to 20 sec. (i.e. because the server is already handling the malicious packets.) The attacker runs *hping3* tool (Bozhinovski et al., 2013) to send malicious packets to Server in the network. The command is given below.

hping3 -i <time> -S -p <port> <ip>;

The Correlation Coefficient values for the genuine host is.

- The below table shows that the correlation coefficient values for the genuine host and these correlation values shows that the relation is **Strong positive linear**.

Table 5.1: Correlation matrix of the genuine user.

Correlation	**SYN**	**SYN-ACK**	**ACK**	**FIN**
SYN	1.0	0.99	0.996	0.998
SYN-ACK	0.996	1.0	0.996	0.998
ACK	0.996	0.996	1.0	0.998
FIN	0.996	0.996	0.998	1.0

- The below table shows that the correlation coefficient values for the attacker and these correlation values shows that the relation is **Moderate positive linear relationship**.

Table 5.2: Correlation matrix of the attacker.

Correlation	**SYN**	**SYN-ACK**	**ACK**	**FIN**
SYN	1.0	0.998	0.472	0.472
SYN-ACK	0.998	1.0	0.472	0.472
ACK	0.467	0.472	1.0	0.998
FIN	0.476	0.0472	0.998	1.0

The below figure shows the comparison between simple LRU algorithm and proposed

scheme on flow table. Our main intention is to have the flow entries as long as possible so that overhead on the controller can be minimized.
The configuration details for proposed scheme and simple LRU algorithm :
Flow table size is 256 entries.
idle_time is 10 seconds.
hard_time is 60 seconds and we are reading 16 entries per seconds.
Fig. 5.2 shows the result and better results comes in proposed scheme.

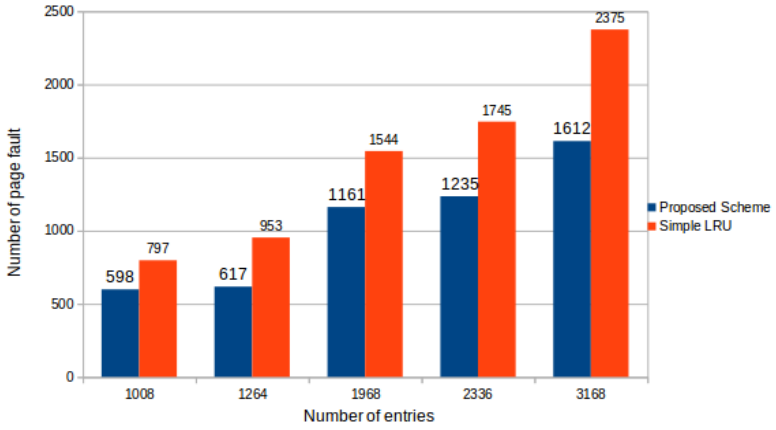


Figure 5.2: The results comparison of LRU and proposed scheme with flow table size 256 entries

Now we increase the *idle_time* and *hard_time* so that we can have the flow entries for long time and we will see what are the results. The configuration details for proposed scheme and simple LRU algorithm :
Flow table size is 512 entries.
idle_time is 20 seconds.
hard_time is 100 seconds and we are reading 16 entries per seconds. We are getting more better results than the previous one.
Fig. 5.3 shows the result and better results comes in proposed scheme.

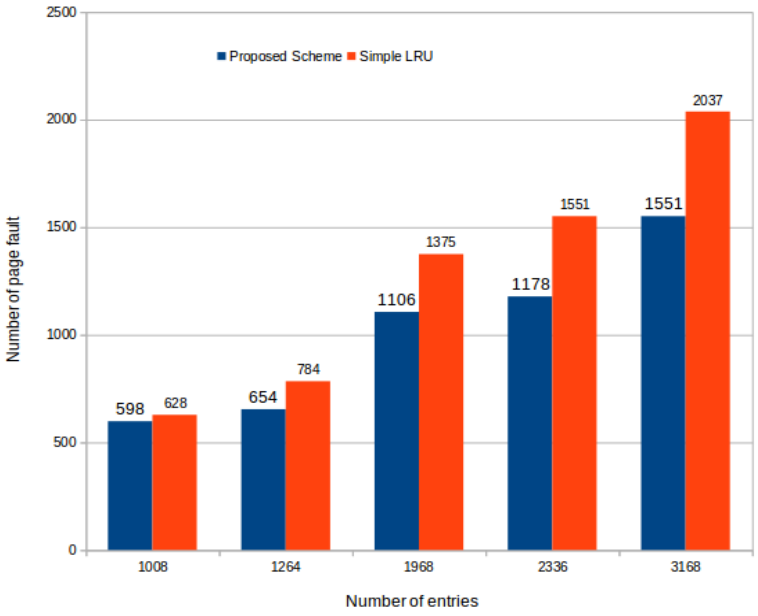


Figure 5.3: The results comparison of LRU and proposed scheme with flow table size 512 entries

CHAPTER 6
CONCLUSION AND FUTURE WORK

This work developed a CDM Module for the mitigation of TCP-SYN flood attack. It has three phases which are the collection, detection and mitigation. The detection part of the CDM module allows the administrator to select one of the statistical algorithm like Shannon entropy or Correlation Matrix.

The proposed CDM module takes very few time (i.e. approx 60 ms. to 180 ms.) to detect the DDoS attack in detection phase but it takes time to provide the services from the server approx 15 sec. to 20 sec. (i.e. the server is already handling the malicious packets) for the genuine users because it has to drop all the malicious requests which were already handled.

In flow table management, we proposed a modified LRU algorithm in which we exploited idle space present in OpenFlow switch and minimize the table miss than the simple LRU algorithm on flow table management. Our main intention was to have the more likely frequently used entries as long as possible so that overhead on the controller can be minimized. We experimented our proposed scheme as the number of page fault are 598 for 1008 entries in proposed scheme and the flow table can have the maximum of the 256 entries and in simple LRU page fault was 797. So the results are better in the proposed scheme.

6.1 FUTURE WORK

For the flow table management, our proposed scheme was trying to have the more likely frequently entries but not always the frequently used entry. So in future, we can have a scheme which is trying to have the most frequent entries in flow table. But to implement this technique the processing time is going to increase since we have to sort all the entries as the new flow entry comes.

REFERENCES

Bedi, H., Roy, S., and Shiva, S. (2013). Mitigating congestion-based denial of service attacks with active queue management. In *Global Communications Conference (GLOBECOM), 2013 IEEE*, pages 1440–1445. IEEE.

Bozhinovski, R., Dimitrova, V., Jakimovski, B., and Ristov, S. (2013). Security penetration test on fcse's it services.

Bull, P., Austin, R., Popov, E., Sharma, M., and Watson, R. (2016). Flow based security for iot devices using an sdn gateway. In *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th International Conference on*, pages 157–163. IEEE.

Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. (2011). Devoflow: scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 254–265. ACM.

Dharma, N. G., Muthohar, M. F., Prayuda, J. A., Priagung, K., and Choi, D. (2015). Time-based ddos detection and mitigation for sdn controller. In *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*, pages 550–553. IEEE.

ElDefrawy, K. and Kaczmarek, T. (2016). Byzantine fault tolerant software-defined networking (sdn) controllers. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 2, pages 208–213. IEEE.

Godlovitch, I., Henseler-Unger, I., and Stumpf, U. (2015). Competition & investment.

Hu, F. (2014). *Network Innovation through OpenFlow and SDN: Principles and Design*. CRC Press.

Kim, T., Lee, K., Lee, J., Park, S., Kim, Y.-H., and Lee, B. (2014). A dynamic timeout control algorithm in software defined networks. *International Journal of Future Computer and Communication*, 3(5):331.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Lee, B.-S., Kanagavelu, R., and Aung, K. M. M. (2013). An efficient flow cache algorithm with improved fairness in software-defined data center networks. In *Cloud Networking (CloudNet), 2013 IEEE 2nd International Conference on*, pages 18–24. IEEE.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74.

Meitei, I. L., Singh, K. J., and De, T. (2016). Detection of ddos dns amplification attack using classification algorithm. In *Proceedings of the International Conference on Informatics and Analytics*, page 81. ACM.

Mirkovic, J. and Reiher, P. (2004). A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53.

Mousavi, S. M. and St-Hilaire, M. (2015). Early detection of ddos attacks against sdn controllers. In *Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 77–81. IEEE.

Naous, J., Erickson, D., Covington, G. A., Appenzeller, G., and McKeown, N. (2008). Implementing an openflow switch on the netfpga platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 1–9. ACM.

Ratner, B. (2009). The correlation coefficient: Its values range between+ 1/- 1, or do they? *Journal of targeting, measurement and analysis for marketing*, 17(2):139–142.

Shalimov, A., Zuikov, D., Zimarina, D., Pashkov, V., and Smeliansky, R. (2013). Advanced study of sdn/openflow controllers. In *Proceedings of the 9th central & eastern european software engineering conference in russia*, page 1. ACM.

Shenker, S., Casado, M., Koponen, T., McKeown, N., et al. (2011). The future of networking, and the past of protocols. *Open Networking Summit*, 20:1–30.

Yan, Q., Yu, F. R., Gong, Q., and Li, J. (2016). Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments:

A survey, some research issues, and challenges. *IEEE Communications Surveys & Tutorials*, 18(1):602–622.

Yu, M., Rexford, J., Freedman, M. J., and Wang, J. (2010). Scalable flow-based networking with difane. *ACM SIGCOMM Computer Communication Review*, 40(4):351–362.

Zarek, A., Ganjali, Y., and Lie, D. (2012). Openflow timeouts demystified. *Univ. of Toronto, Toronto, Ontario, Canada*.

Zhu, H., Fan, H., Luo, X., and Jin, Y. (2015). Intelligent timeout master: Dynamic timeout for sdn-based data centers. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 734–737. IEEE.

BIODATA

Name	Azaruddhin Khan
Address	Village Kanchana, Mursan, Hathras, Uttar Pradesh PIN code - 204213
E-mail	khanazharuddin740@gmail.com
Mobile	9900476396
Qualification	B.Tech. in Information Technology (Kamla Nehru Institute of Technology, Sultanpur) M.Tech. in Computer Science and Engineering - Information Security (NITK Surathkal)