Simulation Protect the Pox Controller from DDoS Attacks

Sally A. Al-Hasnawi, Mahmood K. Ibrahem

Abstract— Distributed Denial of Service is one of the harmful attackers which send a lot of requests to the target until it can control that target. The controller is the heart of the network topology in Software-Defined Network (SDN), and because the controller centralization control so must protect the controller from that attacks to increase the security of the controller. In this paper, the mechanism of protection from DDoS attacks in software-defined networks (SDNs) is investigated. A software-defined network that contains the mechanism of protection from DDoS attacks by specifying the max number of requests that reach per interval from each host is simulated and analyzed. The simulated topology consists of a Pox Controller, an open virtual switch, six clients, and one server. The OpenFlow protocol is utilized to connect the mentioned topology with the Pox controller. The results show that when this mechanism applies the controller being more secure because the controller refused the request from the host when the requests more than the max request.

Index Terms— Distributed Denial of Service (DDoS), Software Defined Network (SDN), OpenFlow.

IJSER

1 INTRODUCTION

Distributed denial-of-service (DDoS) attacks [1] make online services unavailable by overwhelming victims with traffic from multiple attackers. As more and more businesses migrate their operations online, significant financial losses DDoS attacks have caused [2]. Different forms of DDoS attacks have, for example, disruption of configuration information, consumption of computational resources. Also, there are different DDoS detection methods [1]. From multiple hosts connected to the SDN switches, DDoS attacks are launched [3]. DDoS is one of the urgent and hardest security issues. Furthermore, DDoS attacks will often carry out by attackers, which increased effectiveness by employ different machines, by trying to tie up all of the tellers at all of the open windows. So, it can often be harder to block attackers manually and detect. For that special defenses are important to defend and detect against such large-scale attacks. Additionally, attackers almost never legitimately control their attacking machines. They infect thousands of computers spread across the world with specialized malware in order to gain unauthorized access to such machines. A collection of thousands or hundreds of compromised machines acting as an army under the control of one attacker is known as a "botnet", and oftentimes the actual owners of machines that are part of a botnet are unaware that their computers have been compromised and are being used to launch DDoS attacks [4].

2 DOS AND DDOS

Denial of Service (DoS) attack is when an attacker uses a single machine's resources to exhaust those of another machine, in order to prevent it from functioning normally. Large web servers are robust enough to withstand a basic DoS attack from a single machine without suffering performance loss. There are a number of different ways that DoS attacks can be used [4]. These include the following:

1. Teardrop Attack: an attacker sends IP data packet fragments to a network. The network then attempts to recompile these fragments into their original packets. The process of compiling these fragments exhausts the system and it ends up crashing. It crashes because the fields are designed to confuse the system so that it cannot put them back together.
2. Ping of Death (POD) or Internet Message Control Protocol (ICMP) flood: is used to take misconfigured network devices and uses them to send spoof packets to ping every computer within the aim network.
3. Buffer overflow attacks: this type of attack is the most common DOS attack experienced. Under this attack, the

attacker overloads a network address with traffic so that it is put out of use.

4. SYN flood: is send requests to connect to a server but don't complete the handshake.

Where Distributed Denial of Service (DDoS) attacks are more complex because they use a range of devices that increase the severity of attacks. Being attacked by one computer is not the same as being attacked by a botnet of one hundred devices. DDoS attacks can come in different forms including:

1. Ping Flood – Much like a UDP flood attack, a ping flood attack uses ICMP Echo Request or ping packets to derail a network's service. The attacker sends these packets rapidly without waiting for a reply in an attempt to make the target network unreachable through brute force. These attacks are particularly concerning because bandwidth is consumed both ways with attacked servers trying to reply with their own ICMP Echo Reply packets. The end result is a decline in speed across the entire network [4].
2. UDP Floods: is a DDoS attack that floods the victim network with User Datagram Protocol (UDP) packets. The attack works by flooding ports on a remote host so that the host keeps looking for an application listening at the port. When the host discovers that there is no application it replies with a packet that says the destination wasn't reachable. This consumes network resources and means that other devices can't connect properly, as illustrates in figure 1 [4].

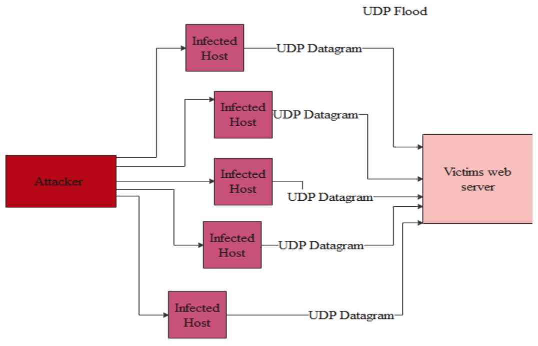


Figure.1: UDP Flood [4].

3. Slowloris: is a type of DDoS attack software that was originally developed by Robert Hansen or RSnake to take down web servers. A Slowloris attack occurs when the attacker sends partial HTTP requests with no intention of completing them. To keep the attack going, Slowloris periodically sends HTTP headers for each request to keep the computer network's resources tied up. This continues until the server can't make any more connections. This form of attack is used by attackers because it doesn't require any bandwidth.

3 LITERATURE REVIEW

Mikhail Belyaev, Svetlana Gaivoronski, in 2014 [5] in their work considered the opportunities of SDN for a "survival" mitigation during Distributed Denial of Service (DDoS) attacks, the load balancing problem. They propose a two-level balancing solution in SDN networks, which includes traditional balancing between servers and load balancing between network devices Experiments show that their solution in caseload balancing between network devices increases the "survival" time of a system during the DDoS attack in times compared to the existing balancing solution in SDN networks. Faisal Shahzad, Muazzam A. Khan, Shoab A. Khan, Saad Rehman, and Monis Akhlaq, in 2016 [6] proposed a methodology to automatically detect multiple types of (DDoS) attacks within few seconds using sampling techniques for continuous monitoring site-wide traffic and block attacking source with the help of OpenFlow protocol. Wisam H. Muragaa, Kamaruzzaman Seman, and Mohd Fadzli Marhusin in 2018 [7] the main objective of this paper is to simulate DDoS attacks in SDN in terms of the number of packets and time of generating. Thus, this paper gives a chance for developers to tackle the problems of this attack by concentrating only on the packets numbers either it high or low, and the time each packet takes to reach its destination. But they didn't propose a mechanism to protect from DDOS attacks. Narmeen Zakaria Bawany, and Jawwad A. Shamsi, in 2019 [8] they presented a mechanism which detects DDOS attack on the controller by identifying the switch that is sending the highest amount of control traffic. The algorithm is

IJSER

designed to mitigate an attack by load balancing and/or blocking the excessive control traffic from the switch, while this thesis used another mechanism by identifying the max number of requests that allow per interval from each host and prevent the hosts which do that and can block them by removing their IP from the list of IPs which allow accessing the controller.

4 THE CONTROLLER

The control plane manages different services. The control plane defines the regulation through the flow regulations on network and SDN switches instruments are extracted and applied in the data plane [9]. The South Bound Interface (SBI) which is the interface between the networking devices and the controller. By the North Bound Interface (NBI), the interface between the controller and the applications is represented. The decision module offers paths to define the best paths among network nodes. The Link Discovery Module makes organized queries for external ports for messages sent by packets. The controller's main tasks primarily apply to topology and traffic flow. These test messages are returned throughout the shape of a communications packet that helps the controller construct the network topology. By the director of topology, the topology is controlled. Flow Manager is among the main modules that directly communicate with flow tables and the flow of the data plane. The controller also consists of a various queue administrator and information administrator for handling and the processing of various outgoing and incoming queues. Several SDN controllers [10] are obtainable in a variety of settings and programs. For example, Python Network Operating System (POX) is based on Python, the Network Operating System (NOX) is C++-based and Python-based, Floodlight, and Beacon. POX requires more physical resources to perform efficiently, it shows a minimal change in latency as the switches increase [11]. Beacon controller: it has also released a robust Java app. Floodlight controller: it is a Java-based free-flow system with origins from the Beacon framework. This is the most advanced free flow controller [12] that is simple to set up and has excellent performance. OpenFlow provides access to the forwarding plane of a network router or switch over the network, facilitating more sophisticated traffic management, especially for cloud and virtualized environments. OpenFlow is an open, standards-based communications protocol, and an example of device-based SDN. The OpenFlow targets at separating the data plane from the control plane. Another well-known effort to separate the control plane from the data plane and standardize information exchange between the data planes and control plane is ForCES [13] [14] proposed by Internet Engineering Task Force (IETF) [15]. By the Open Networking Foundation (ONF), The OpenFlow protocol is managed and standardized. The Mission also includes the promotion of SDN technologies as a whole [16].

5 PROPOSED ALGORITHM OF PROTECT FROM DDOS

The protection is attained through anti-mechanism for the DDOS attacks made by specifying the max requests for each host per interval. If the requests from the host exceed the specified value the controller will deny the requests, and be treated as DDOS and the host will be blocked and treat it as the DDOS attack, as illustrates in figure 2.

Algorithm: Protection from DDOS

○ **Input** (No. of Requests from the Host=x)

○ **Output** (The requests accept or reject)

➢ **Start**

➢ Send Request Stage:

Step-1:

• Host: send the requests to the controller.

➢ Check the Number of Requests Stage:

Step-2:

• If the number of requests from a host per second more than the allowed max number the requests reject.

• Otherwise: the requests reach the controller and retrieve the answer to the host.

➢ **End**

IJSER

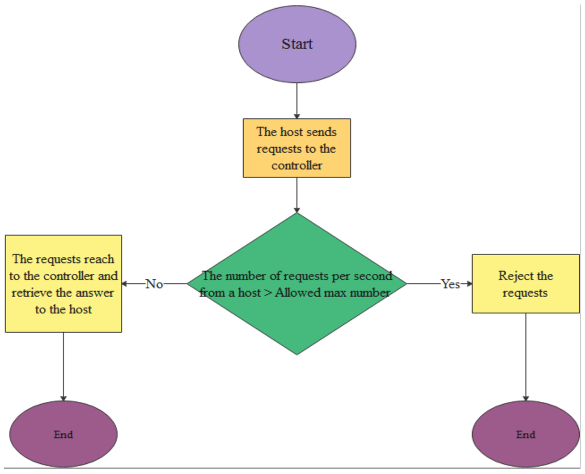

Start
The host sends requests to the controller
The number of requests per second from a host > Allowed max number
No
Yes
The requests reach to the controller and retrieve the answer to the host
Reject the requests
End
End

Figure.2: Flowchart protects the controller from DDOS.

6 PROPOSED TOPOLOGY

Figure.3 illustrates protection from DDOS attacks topology. The proposed topology consists of one POX controller, one server, six clients, and one open virtual switch. When the requests from the host reach the server, the server knows the max request that allows from each host per interval that the controller specified before. So, the server refuses the requests that more than the max request of packet per interval. For that, the controller will be more secure.

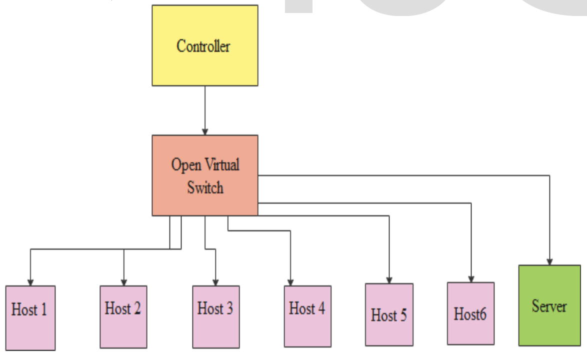
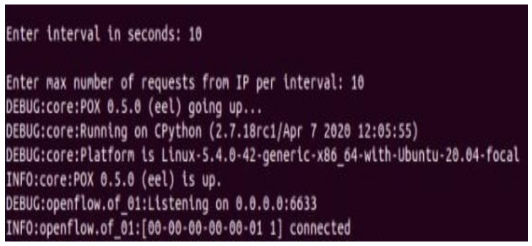


Figure.3: The topology of DDOS protection.

7 IMPLEMENTATION OF THR PROPOSED ALGORITHM

1. The protection is happened by anti-mechanism for the DDOS attacks made by specifying the max requests for each host per interval (the specified time). If the requests from the host increase, the value that specifies the requests denies and blocks that host and treats it as the DDOS attack. In the figure below, the specified max number (10 packets per 10 second) received from each host per interval is shown. If the number of packets received from the host is more than the max number of packets, the request refuses and specifies the host as DDOS attacks, as illustrated in figure 4.

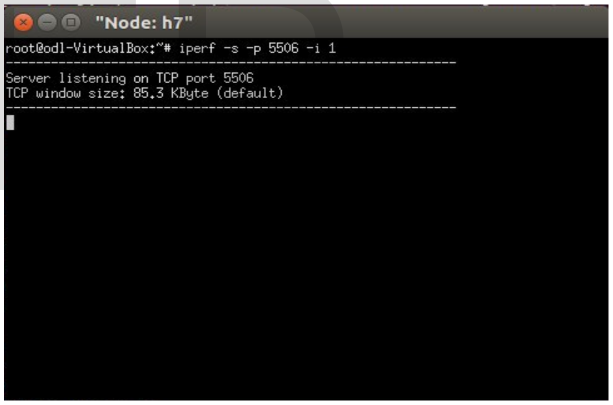


```
Enter interval in seconds: 10

Enter max number of requests from IP per interval: 10
DEBUG:core:POX 0.5.0 (eel) going up...
DEBUG:core:Running on CPython (2.7.18rc1/Apr 7 2020 12:05:55)
DEBUG:core:Platform is Linux-5.4.0-42-generic-x86_64-with-Ubuntu-20.04-focal
INFO:core:POX 0.5.0 (eel) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
```

Figure 4: Pox controller configuration.

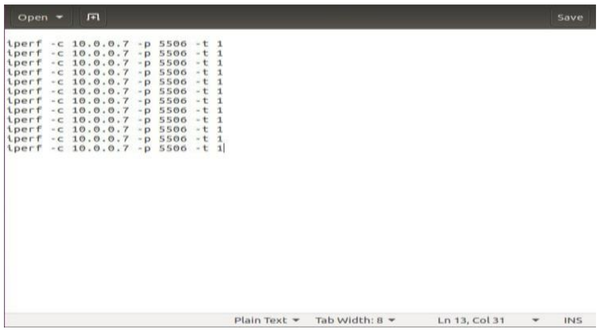
2. Activate host seven to work as a server, as illustrated in figure 5.



```
root@odl-VirtualBox:~# iperf -s -p 5506 -i 1
------------------------------------------------------------
Server listening on TCP port 5506
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

Figure 5: Activate host7 as a server.

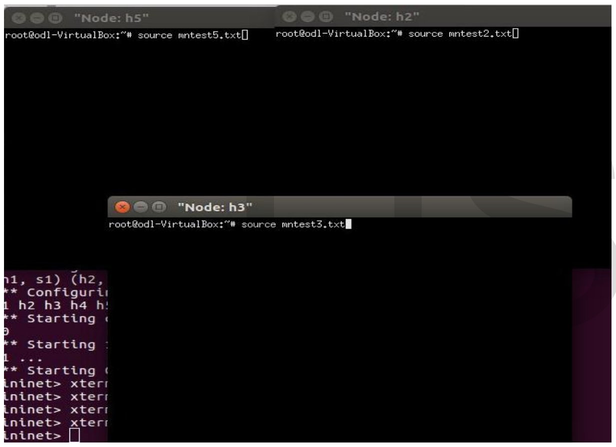
3. Specifies a folder for each host and the number of requests that the host sends per interval as shown in the figure below. In host two and host three and host five, the specified number of requests per interval is more than the max number of packets, as illustrated in figure 6.



```
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
```

Figure 6: Host2 and host3 and host5requests.

4. When host two or host three or host five send requests to the server(Host7), the controller treats them as DDOS attacks and refuses their connection, as illustrated in figure 7.



```
root@odl-VirtualBox:~# source mntest5.txt
```

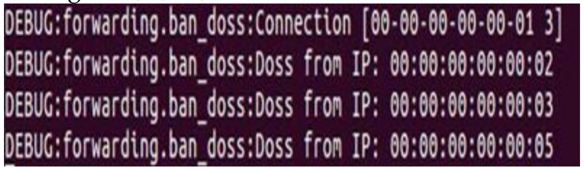
```
root@odl-VirtualBox:~# source mntest2.txt
```

```
root@odl-VirtualBox:~# source mntest3.txt
```

```
h1, s1) (h2,
** Configuri
1 h2 h3 h4 h
** Starting
0
** Starting
1 ...
** Starting
ininet> xter
ininet> xter
ininet> xter
ininet> xter
ininet>
```

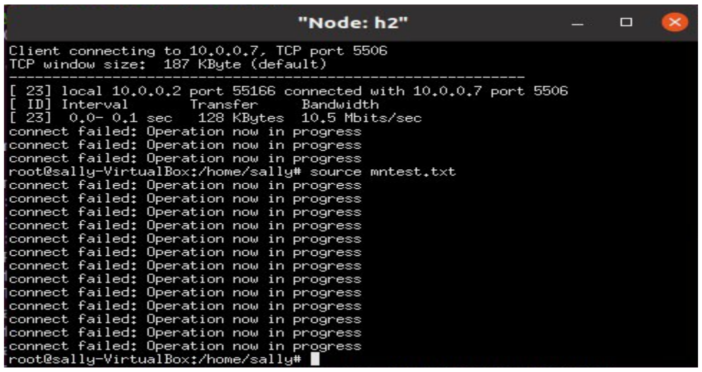
Figure 7: Host2 and host3 and host5 send requests to the server (Host7).

5. The controller treats them as DDOS attacks and refuses their connection, as illustrated in figure 8, 9, 10, and 11.



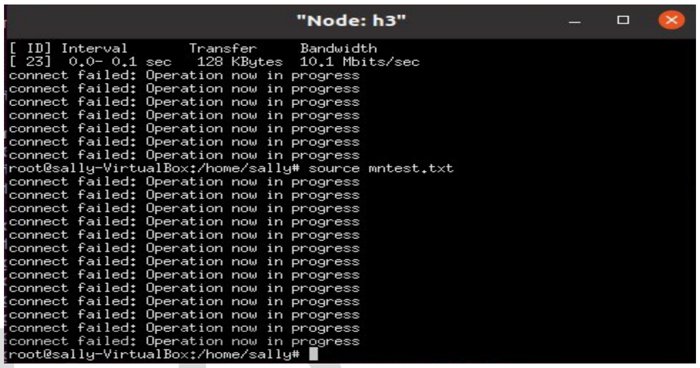
```
DEBUG:forwarding.ban_doss:Connection [00-00-00-00-00-01 3]
DEBUG:forwarding.ban_doss:Doss from IP: 00:00:00:00:00:02
DEBUG:forwarding.ban_doss:Doss from IP: 00:00:00:00:00:03
DEBUG:forwarding.ban_doss:Doss from IP: 00:00:00:00:00:05
```

Figure 8: Host 2 and host3 and host5 DDOS attacks.



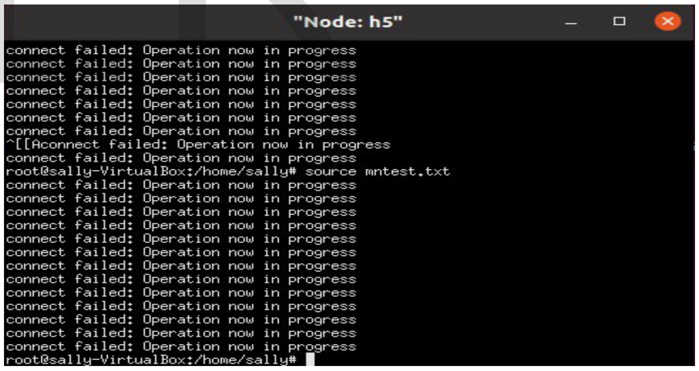
```
Client connecting to 10.0.0.7, TCP port 5506
TCP window size:  187 KByte (default)
------------------------------------------------------------
[ 23] local 10.0.0.2 port 55166 connected with 10.0.0.7 port 5506
[ ID] Interval       Transfer     Bandwidth
[ 23]  0.0- 0.1 sec   128 KBytes  10.5 Mbits/sec
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
root@sally-VirtualBox:/home/sally# source mntest.txt
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
root@sally-VirtualBox:/home/sally#
```

Figure 9: Host 2 connection refused.



```
[ ID] Interval       Transfer     Bandwidth
[ 23]  0.0- 0.1 sec   128 KBytes  10.1 Mbits/sec
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
root@sally-VirtualBox:/home/sally# source mntest.txt
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
root@sally-VirtualBox:/home/sally#
```

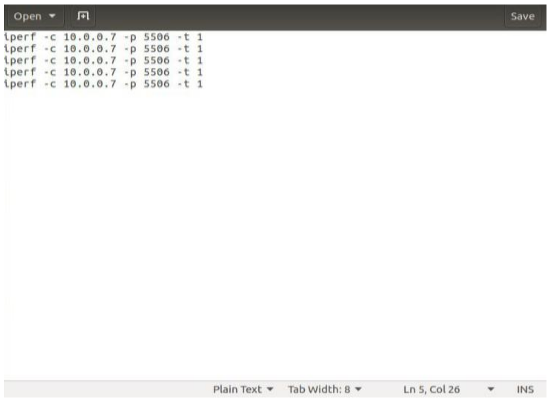
Figure 10: Host 3 connection refused.



```
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
^[[Aconnect failed: Operation now in progress
connect failed: Operation now in progress
root@sally-VirtualBox:/home/sally# source mntest.txt
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
connect failed: Operation now in progress
root@sally-VirtualBox:/home/sally#
```

Figure 11: Host 5 connection refused.

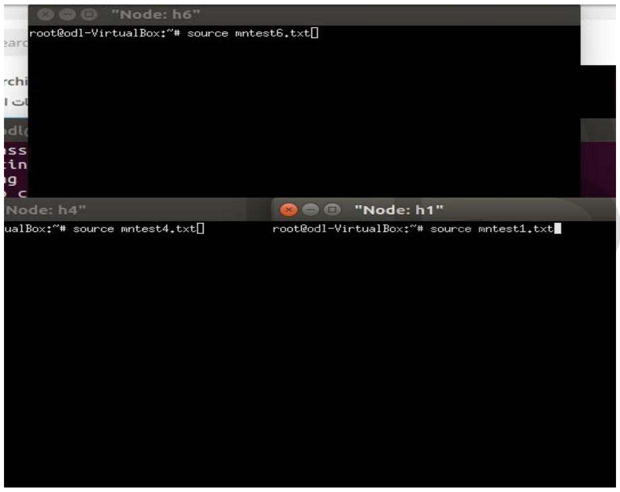
6. Now, it specified a folder for hosts one, four, six, and the number of requests that the host sends per interval, as shown in the figure below. In host one and host four and host six, the specified number of requests per interval is less than the max number of packets, as illustrated in figure 12.



```
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
iperf -c 10.0.0.7 -p 5506 -t 1
```

Figure 12: Host1and host4 and host6 requests.

7. When host one or host four or host six sends requests to the server (Host7), the controller accepts their connection, as illustrated in figure 13, 14.

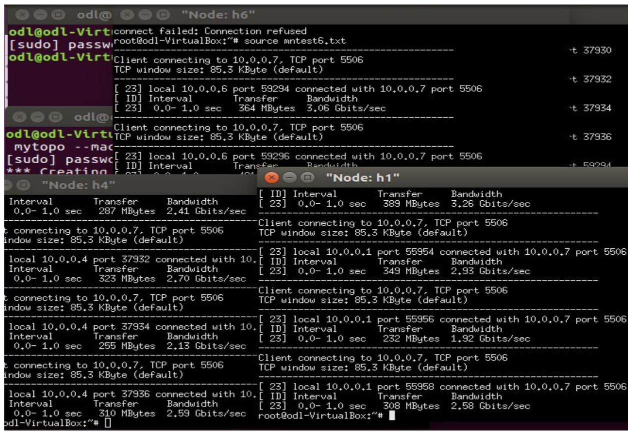


```
root@odl-VirtualBox:~# source mntest6.txt
```

```
ualBox:~# source mntest4.txt
```

```
root@odl-VirtualBox:~# source mntest1.txt
```

Figure 13: Host1, 4, 6 send requests to the server (Host7).



```
connect failed: Connection refused
root@odl-VirtualBox:~# source mntest6.txt
------------------------------------------------------------
Client connecting to 10.0.0.7, TCP port 5506
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 23] local 10.0.0.6 port 59294 connected with 10.0.0.7 port 5506
[ ID] Interval       Transfer     Bandwidth
[ 23]  0.0- 1.0 sec   364 MBytes  3.06 Gbits/sec
------------------------------------------------------------
Client connecting to 10.0.0.7, TCP port 5506
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 23] local 10.0.0.6 port 59296 connected with 10.0.0.7 port 5506
[ ID] Interval       Transfer     Bandwidth
```

```
 Interval       Transfer     Bandwidth
  0.0- 1.0 sec   287 MBytes  2.41 Gbits/sec
------------------------------------------------------------
t connecting to 10.0.0.7, TCP port 5506
indow size: 85.3 KByte (default)
------------------------------------------------------------
 local 10.0.0.4 port 37932 connected with 10.
 Interval       Transfer     Bandwidth
  0.0- 1.0 sec   323 MBytes  2.70 Gbits/sec
------------------------------------------------------------
t connecting to 10.0.0.7, TCP port 5506
indow size: 85.3 KByte (default)
------------------------------------------------------------
 local 10.0.0.4 port 37934 connected with 10.
 Interval       Transfer     Bandwidth
  0.0- 1.0 sec   255 MBytes  2.13 Gbits/sec
------------------------------------------------------------
t connecting to 10.0.0.7, TCP port 5506
indow size: 85.3 KByte (default)
------------------------------------------------------------
 local 10.0.0.4 port 37936 connected with 10.
 Interval       Transfer     Bandwidth
  0.0- 1.0 sec   310 MBytes  2.59 Gbits/sec
odl-VirtualBox:~# 
```

```
[ ID] Interval       Transfer     Bandwidth
[ 23]  0.0- 1.0 sec   389 MBytes  3.26 Gbits/sec
------------------------------------------------------------
Client connecting to 10.0.0.7, TCP port 5506
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 23] local 10.0.0.1 port 55954 connected with 10.0.0.7 port 5506
[ ID] Interval       Transfer     Bandwidth
[ 23]  0.0- 1.0 sec   349 MBytes  2.93 Gbits/sec
------------------------------------------------------------
Client connecting to 10.0.0.7, TCP port 5506
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 23] local 10.0.0.1 port 55956 connected with 10.0.0.7 port 5506
[ ID] Interval       Transfer     Bandwidth
[ 23]  0.0- 1.0 sec   232 MBytes  1.92 Gbits/sec
------------------------------------------------------------
Client connecting to 10.0.0.7, TCP port 5506
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 23] local 10.0.0.1 port 55958 connected with 10.0.0.7 port 5506
[ ID] Interval       Transfer     Bandwidth
[ 23]  0.0- 1.0 sec   308 MBytes  2.58 Gbits/sec
root@odl-VirtualBox:~# 
```

Figure 14: Host1 and host4 and host6 accepted their connection and received an answer.

8 DISCUSSION

Software-Defined Network (SDN) is a computer networking model that supports programmable interfaces that provide a suitable way to adapt the network traffic control. In SDN they separate control plane from data plane. The data plane forwards traffic to the next hop along the path to the specific destination network. The control plane decides where the traffic is forward. SDN network depends on the OpenFlow protocol to exchange information with the network. The controller is a single point of failure which can make a very dangerous issue for the network that connects with that controller. So, this paper simulates a mechanism protect the controller from DDOS attacks. By the strategy of specify the max number of requests from each host per interval to make the controller more secure.

9 CONCLUSION

In this paper, a mechanism of DDOS protection for the software defined networks is elaborated. The proposed mechanism has been evaluated through a simulation, which was implemented using Python programming language. The simulated mechanism depends on specifying the max number of requests per interval from each host. The results showed that when this mechanism has applied the controller refused the requests that more than the max value. So, the controller made more secure.

10 REFERENCES

[1] J. Mirkovic and P. Reiher, "A taxonomy of ddos attack and ddos defense mechanisms", ACM SIGCOMM Computer Communication Review, vol. 34, no. 2, pp. 39–53, 2004.

[2] Xu, Yang, and Yong Liu, "DDoS attack detection under SDN context", IEEE INFOCOM 2016-the 35th annual IEEE international conference on computer communications. IEEE, 2016.

[3] Sarwan Ali, Maria Khalid Alvi, Safi Faizullah, Muhammad Asad Khan, Abdullah Alshanqiti, Imdadullah Khan, "Detecting DDoS Attack on SDN Due to Vulnerabilities in OpenFlow",

International Conference on Advances in the Emerging Computing Technologies (AECT). IEEE, 2020.

[4] Kenig, R., Manor, D., Gadot, Z., & Trauner, D., "DDoS survival handbook", 2013.

[5] Mikhail Belyaev, Svetlana Gaivoronski, "Towards Load Balancing in SDN-Networks During DDoS-attacks", in the 2014 International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC), Moscow, Russia, 2014.

[6] Faisal Shahzad, Muazzam A. Khan, Shoab A.Khan, Saad Rehman, and Monis Akhlaq, "AutoDrop: Automatic DDoS Detection and Its Mitigation with Combination of OpenFlow and sFlow", International Conference on Future Intelligent Vehicular Technologies. Springer, pp.112-122, Cham, 2016.

[7] W.Muragaa, K.Seman, and M.Fadzli Marhusin, "SIMULATING DDOS ATTACK IN SDN NETWORK USING POX CONTROLLER AND MININET EMULATOR", Proceedings of 134th The IRES International Conference, Malaysia, 2018.

[8] Narmeen Zakaria Bawany, and Jawwad A. Shamsi, "SEAL: SDN based secure and agile framework for protecting smart city applications from DDoS attacks", Journal of Network and Computer Applications, 2019.

[9] V. Varadharajan, K. Karmakra, U. Tupakula, and M. Hitchens, "A policy-based security architecture for software-defined networks", IEEE Transactions on Information Forensics and Security, Vol. 14(4), pp. 897-912, 2018.

[10] D.Kreutz, F. M. V. Ramos, P.Verissimo, Fellow, C.Esteve Rothenberg, Member, S.Azodolmolky, and S.Uhlig, "Software-defined networking: A comprehensive survey", Proceedings of the IEEE, Vol. 103(1), pp. 14-76, 2014.

[11] Liehuang Zhu, , Md Monjurul Karim, Kashif Sharif, , Fan Li, Xiaojiang Du, and Mohsen Guizani, Fellow, "Sdn controllers: Benchmarking & performance evaluation". Available: http://arxiv.org/pdf/1902.04491.pdf, 2019.

[12] N.McKeown, Tom Anderson, Hari Balakrishnan , Guru Parulkar , Larry Peterson , Jennifer Rexford , cott Shenker , Jonathan Turner , "OpenFlow: enabling innovation in campus networks", ACM SIGCOMM Computer Communication Review , Vol. 38(2), pp.69-74, 2008.

[13] Ali Akbar Neghabi1, Nima Jafari Navimipour, Mehdi Hosseinzadeh, Ali Rezaee, "Nature-inspired meta-heuristic algorithms for solving the load balancing problem in the software-defined network", International Journal of Communication Systems , vol.32 (4), pp.3875, 2017.

[14] L. Yang, R. Dantu, T. Anderson, and R. Gopal, Forwarding and Control Element Separation (ForCES) Framework, Apr. 2004, RFC 3746. [Online]Available:http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[15] W.Xia, Y.Wen, C.Heng Foh, D.Niyato, and H.XieXia, "A survey on software-defined networking", IEEE Communications Surveys and Tutorials, Vol. 17(1), pp. 27-51, 2014.

[16] A. Neeraja, Dr. N. Chandra Sekhar Reddy, Mukund, "Improving Network Management with Software Defined Networking", International Journal of Science and Research (IJSR), Vol.3(7), pp.1-4, 2012.

Author's Details:

Sally A. Al-Hasnawi, and Mahmood K. Ibrahem.
Collage of Information Engineering / Networks Department
Al-Nahrain University/ Baghdad-Iraq.
Sally.a.karim@gmail.com
mahmoodkhalel@coie-nahrain.edu.iq

IJSER