NOVATEUR PUBLICATIONS
INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY
[IJIERT] ISSN: 2394-3696 Website: ijiert.org
VOLUME 8, ISSUE 9, Sep. -2021

IMPLEMENTING SOFTWARE DEFINED NETWORKING (SDN) BASED FIREWALL USING POX CONTROLLER

Siddhesh Deshmukh
Department of Electronics and Telecommunications,
K. J. Somaiya College of Engineering, Mumbai, India
siddhesh.dd@somaiya.edu

Amey Gawde
Department of Electronics and Telecommunications,
K. J. Somaiya College of Engineering, Mumbai, India
amey.gawde@somaiya.edu

Nitin Nagori
Department of Electronics and Telecommunications,
K. J. Somaiya College of Engineering, Mumbai, India
nitinnagori@somaiya.edu

ABSTRACT

A firewall's main function is to limit unwanted traffic. It will track and manage the flow of data that comes from various sources into the network and functions on the principle of preconfigured rules. Firewalls are one of the important elements of the network infrastructure. In order not to increase the packet delay in the network, they should guarantee the correct level of protection and, at the same time, satisfactory efficiency. According to security policy, the firewall is interposed between two networks to buffer traffic between them. By implementing rule-based control on packets, a firewall gives security protection. With either hardware or software, or a fusion of both, firewalls may be implemented.

Software-Defined Networking (SDN) is an evolving technology that will drive the networks of the next generation. Network managers are given the freedom to introduce their networks. But at the same time, it brings with its new security problems. We need effective firewall solution to protect SDN networks. The SDN provides network managers with a simple description of the whole layout of the network. It decouples the control and forwarding mechanisms of a network so that it is possible to handle the physical and logical networks separately. This approach facilitates the programmatic and efficient reallocation of network traffic flows to fulfil increasing needs. SDN makes networks completely managed by software applications and provides the hope of shifting the limits of traditional network infrastructures.

For implementation of firewall POX controller is used. POX is an open source OpenFlow/Software Oriented Networking (SDN) Controller built on Python. For quicker design and development of experimental network technologies, POX is used. The POX controller arrives with the Mininet virtual machine pre-installed.

Keywords: SDN, Firewall, POX, OpenFlow, Mininet.

INTRODUCTION

A computer firewall is a piece of software that stops unauthorised users from accessing or leaving a private network. To increase computer security on networks like a local area network, or the Internet, firewalls are software applications that may be installed on computers that are connected to those networks. They're an important element of your network's overall security strategy. By implementing a "wall of code" that inspects every "packet" of data that comes inbound to or outgoing from your computer to determine whether it should be allowed to pass or not, a firewall totally isolates your computer from the outside network. A firewall may be thought of as a device that obstructs traffic and filters it according to certain criteria. To protect a network from the internet, a generic firewall can be used. We'll utilise the OpenFlow controller in an SDN-based firewall to filter traffic between hosts according to specified criteria and then allow it flow through or not. We'll achieve this by utilising the POX controller to set up our needed policies or rules and using the switches to filter traffic across hosts. Hardware firewalls are installed at the main hub that links a secured intra-net and the rest of the system. A software firewall is a machine-run software program that safeguards a system by limiting external attempts to obtain access to the network. Despite their benefits, there are three big disadvantages of hardware firewalls. First, the expense of hardware firewalls is always high. Second, hardware firewall repair and maintenance are usually concerned with complex and vendor-specific setups. In addition, it is not always feasible to obtain connectivity within hardware firewalls produced by different vendors. Third, hardware firewall failures will lead to the several hardware firewalls being removed and reconfigured to maintain a clear policy through a network. software-defined firewall offers stronger performance than hardware or conventional firewall since the SDN has a software-based central traffic control system to handle traffic and provide automated and scalable traffic control. SDN is based on the principle of decoupling the packet or frame forwarding feature from the control logic at the lower layer that determines how and where to intelligently and efficiently forward the traffic. This decoupling has allowed us to explore new and innovative ways of handling traffic. The SDN consists of 3 pieces. Physical or virtual switches supported by OpenFlow, second controllers which can be commercially available or open source based and third is network technologies such as routers, firewall, load balancers.

RELATED WORK AND TECHNICAL BACKGROUND

A. Related work

The Firewall is a network security framework that, based on predetermined security rules, tracks and manages incoming and outgoing network traffic. Usually, a firewall creates a buffer between a trusted internal network and an external network that is not trusted. Sometimes, firewalls are listed as either network firewalls or firewalls depending on the host. Network firewalls buffer traffic and run on network hardware from two or more networks. On host computers, host-based firewalls run and monitor network traffic in and out of those devices. Insiders can very easily perform attacks. This issue may be readily handled with OpenFlow by installing a firewall at any one or several locations throughout a network as presented in [1].
For building a distributed firewall that can manage traffic, Kaur [2] used Mininet emulator as a testbed software and POX Controller based on Python as the SDN Controller. SDN will lower network administration costs and improve programmability, allowing networks to be setup and maintained more simply. The Data Center is one of the areas where SDN is most beneficial. Hundreds of thousands of virtual machines move over the underlying architecture of data centres [3]. Instead of having to manually configure each switch to match the new virtual network, they may use programmable switches that are controlled by a central database.
In [4], author is given some information about POX controller. He also given brief information about different SDN controllers and their compatible versions, support of GUIs and language support. But he preferred POX controller because, it is possible to use a POX controller to transform cheap, useless Hub, transform, modem or merchant silicon gadgets Firewall, load balancer, middleboxes. POX is a free and open - source controller for SDN application development. The OpenFlow protocol, which is the de facto communication protocol between controllers and switches, is implemented efficiently by the POX controller. POX is the Perfect platform for the implementation and testing of SDN Applications. Its great power is that it can be used. To capture and examine the packets flowing between the POX controller and OpenFlow devices, use the Tcpdump packet capture programme. True hardware, in testbeds or with Mininet with an emulator.

B. Firewall

Firewall is a kind of cybersecurity mechanism that is used on a network to filter traffic. Firewalls are used to differentiate network nodes from external traffic sources, internal traffic sources, and even specific apps. Firewalls can be software, hardware, or cloud-based, and each has its own set of benefits and drawbacks. A firewall's primary purpose is to prevent malicious traffic requests and data packets while enabling legitimate traffic to move through. There are several different types of firewalls based on their function [5]:
1. Packet filtering: Depending on user-defined criteria, each packet entering or departing the network is inspected and allowed or denied.
2. Circuit-gateway Firewall: When a TCP or UDP connection is established, this procedure implements security measures. Even though incredibly resource-efficient, the packet itself is not tested by these firewalls. So, if a packet was carrying malware, but had the right handshake for TCP, it would go right through firewall this is the major disadvantage of this firewall.

3. Stateful Inspection Firewall: To establish a level of security better than that of packet filtering firewalls or circuit gateway firewalls, these firewalls incorporate both packet inspection technologies and TCP handshake authentication.
4. Proxy Firewall: In the application layer, proxy firewalls run to filter incoming data between the network and the source of traffic. These firewalls are distributed by a cloud-based solution or another proxy system. The proxy firewall first creates a connection to the source of the traffic and inspects the incoming data packet, rather than having traffic link directly.

C. Software defined networking (SDN)

Software defined networking (SDN) is leading to prototype shift in networking through the ideas of programmable network infrastructure and decoupling of network control and data planes. It assures simplified network management and easier introduction of new services or changes into the network. Because of its centralised controller and distinct data and control planes, SDN may handle different functions. The SDN’s functionalities, beside with its layers and planes, the general functionalities of SDN are as follows:
Programmability: Network control is directly programmable as the control plane is disassociate from the forwarding or data plane. SDN enables the control plane to be programmed using different software development tools along with the function of customization of the control network just as user requirements.
Centrally Managed: In an SDN, the controller network is logically centralized, thus provide a complete view of the network that appears to the applications or users as a logical device.
Flexibility: Network administrators benefit from SDN's flexibility. Through dynamic, automated SDN programmes, network managers may swiftly manage, configure, secure, and optimise network settings. This makes it easier for the controllers to respond to traffic changes. As controllers run in software, SDN provides the flexibility of synchronization through the network operating system (NOS) perspective on different physical or virtual hosts.
This paradigm is made up of three layers: an infrastructure layer, a control layer, and an application layer, all of which are stacked on top of each other. The infrastructure layer mainly made up of forwarding elements (e.g., physical and virtual switches, routers, wireless access points) that comprise the data plane. These devices are mainly responsible for

(i) Fetching network status, keeping them temporally in local network devices and sending the stored data to the network controllers.
(ii) For managing packets based on the rules given by the network controllers or administrators. The control layer, also known as control plane, sustains the link between the application layer and the infrastructure layer via open interfaces.

The controller can communicate with other levels through three communication interfaces: the southbound interface for communicating with the infrastructure layer, the northbound interface for interacting with the application layer and east/westbound interfaces for communicating with bunch of controllers. The application layer is designed mainly to fulfil user requirements. It comprises of the end-user business applications that consume network services. SDN applications are capable of control and access switching devices at the data layer via control plane interfaces. SDN contains network visualization, dynamic access control, security, mobility and migration, cloud computing, and load balancing (LB).

IMPLEMENTATION & RESULT

A. Implementation

Firewall design may be summarised as, the controller is loaded onto the switch ahead of time to serve as a main filter for incoming packets. In response, only packets satisfying the switch's criteria are routed to the controller, which then enables bidirectional flows on the switch to allow that traffic to pass through. This design requires the switch to handle most of the traffic as it should and only send data to the controller as necessary. A flow entry would have to be installed for each packet that needs to pass through a switch so that the switch can forward this traffic without further controller interference.
In this implementation we made use of a virtual Ubuntu Linux instance created and run on VMware Workstation. The Mininet network emulator and POX controller were installed on this Ubuntu machine. We have created a network topology in this we have 8 hosts (Pc’s) which acts as trusted hosts and one host which

is directly connected to the controller which acts as an outsider to the network or untrusted host. The hosts in the network are connected to the controller via open vswitches (OVS), in our network we have 4 open vswitches. We have an one OpenFlow controller that can manage any number of switches through the OpenFlow protocol, thereby turning them into L2 MAC-learning switches or hubs. We used the POX controller, which is an open source development platform for Python-based software-defined networking (SDN) control applications like OpenFlow SDN controllers. The controller is directly connected to data centre which is connected to the server.

To test our firewall rules, we made one topology in python script which includes all the network components including remote POX controller. In our module import lines, we must first import Remote Controller instead of the standard Controller, and then pass that through to the Mininet class. Adding MAC addresses to the hosts during initialization allows us to quickly build Layer-2 rules for our Firewall. We run python script with POX controller under MININET environment.

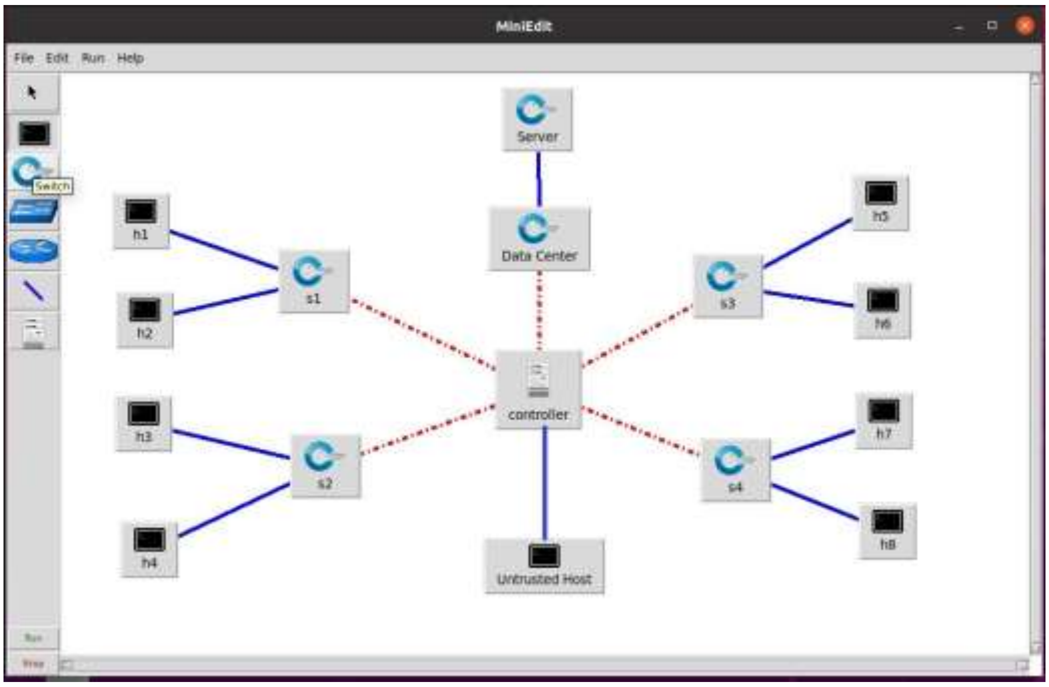


Figure 1 Firewall Topology

B. Firewall rules

To begin, we must determine the rules that will be enforced by our firewall. It should also be emphasised that it will be a bi-directional block, with the controller ensuring that no incoming or outgoing traffic from one host reaches the other if a rule exists. So, for our Firewall, we can have the following set of rules. The logic of the controller is broken into three parts.

The first part is if the packet is not an IP packets. This can be done by checking if the packet is of type IPV4 or not. In the case it is not IPV4, the controller simply floods all traffic using of.OFPP_FLOOD as the destination port since it is not an IP packet.

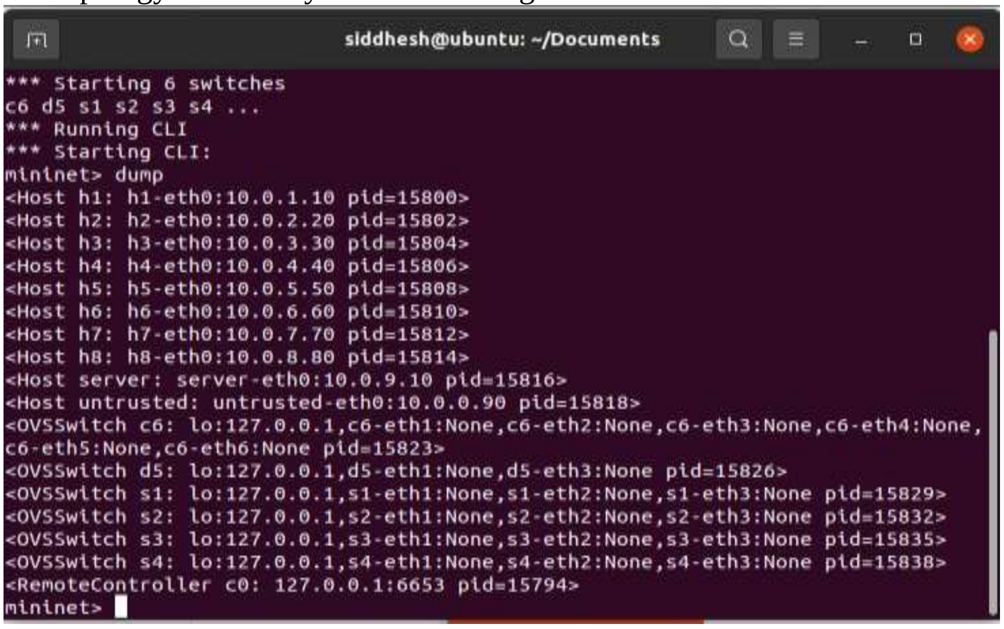
The second part is if the packet is an IP packet. In this case, we specify the ports instead of flooding. This was the topology and port numbers used when specifying. Knowing it is an IP packet, the second part now must first check if the packet is ok to send under the firewall rules, so there are two flags that must be checked before sending it to the destination port. The first flag checks if the packet is of type ICMP and the source is not from the untrusted host. This is the case since the untrusted host is not allowed to since ICMP packets whatsoever. The second flag is if the packet is of type TCP, and if (the source is the untrusted host while the destination is server) is False. This first flag exists because the untrusted host cannot send any IP traffic to the server; since the first flag covers ICMP packets being sent from the untrusted host to the server, the second flag has to check for the case of TCP packets. If either of these flags are true, then the packet is deemed either a safe ICMP or TCP packet respectively. Checking these two flags saves a lot of trouble from having to make special cases for the rest of the firewall code. The source and destination can be found in the IPV4 attributes.

Now that the packet is confirmed safe to send, it is time to send it to its destination IP. To do this, we first must check what switch the packet is currently in. After figuring that out, the packet is sent out to the correct port given the topology defined in the topology file until it finally reaches the destination IP.

The third part of the code is if the both flags were false, and the packet is not safe to send. In that case the packet is dropped by not adding the action, so it is not sent out any ports.

C. Results

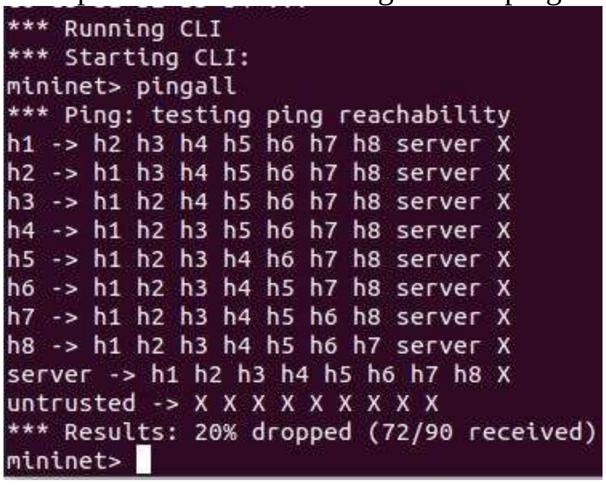
Once we invoke the mininet and implemented our topology with remote controller we can use various commands to see the network parameters and performance of the network. The command dump shows that all the devices were successfully created, and the IP addresses on all the hosts are correct. The dump command will give detailed information about every device connected in the network with IP address, ethernet port details and process id of the device. Similarly, the links command shows all links were successfully made, it also shows that the topology is correctly made also and gives status of the links.



```
*** Starting 6 switches
c6 d5 s1 s2 s3 s4 ...
*** Running CLI
*** Starting CLI:
mininet> dump
<Host h1: h1-eth0:10.0.1.10 pid=15800>
<Host h2: h2-eth0:10.0.2.20 pid=15802>
<Host h3: h3-eth0:10.0.3.30 pid=15804>
<Host h4: h4-eth0:10.0.4.40 pid=15806>
<Host h5: h5-eth0:10.0.5.50 pid=15808>
<Host h6: h6-eth0:10.0.6.60 pid=15810>
<Host h7: h7-eth0:10.0.7.70 pid=15812>
<Host h8: h8-eth0:10.0.8.80 pid=15814>
<Host server: server-eth0:10.0.9.10 pid=15816>
<Host untrusted: untrusted-eth0:10.0.0.90 pid=15818>
<OVSSwitch c6: lo:127.0.0.1,c6-eth1:None,c6-eth2:None,c6-eth3:None,c6-eth4:None,
c6-eth5:None,c6-eth6:None pid=15823>
<OVSSwitch d5: lo:127.0.0.1,d5-eth1:None,d5-eth3:None pid=15826>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=15829>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None,s2-eth3:None pid=15832>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None,s3-eth3:None pid=15835>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None,s4-eth3:None pid=15838>
<RemoteController c0: 127.0.0.1:6653 pid=15794>
mininet>
```

Figure 2 Dump Command output

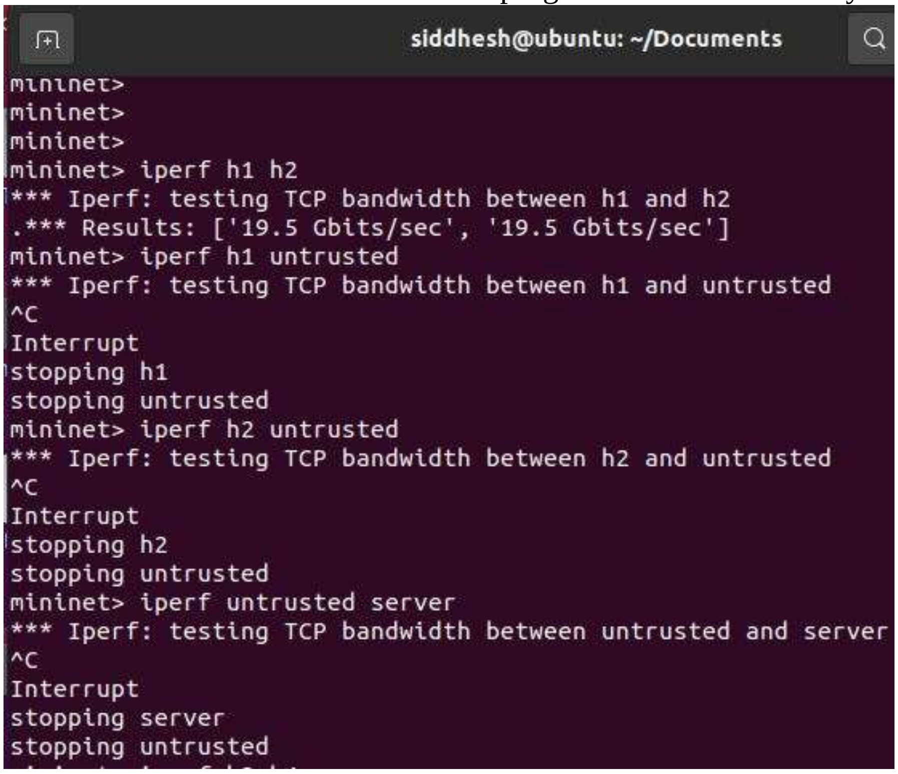
To check the connectivity of the network and test the operation of implemented firewall rules we use pingall command. The pingall command shows that all hosts and the server can ping one another with the exception of the untrusted host. Since the untrusted host is not trusted to send ICMP packets to anyone and the pingall command use ICMP packets, the output is correct in showing that the pings using the untrusted host fails.



```
*** Running CLI
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 server X
h2 -> h1 h3 h4 h5 h6 h7 h8 server X
h3 -> h1 h2 h4 h5 h6 h7 h8 server X
h4 -> h1 h2 h3 h5 h6 h7 h8 server X
h5 -> h1 h2 h3 h4 h6 h7 h8 server X
h6 -> h1 h2 h3 h4 h5 h7 h8 server X
h7 -> h1 h2 h3 h4 h5 h6 h8 server X
h8 -> h1 h2 h3 h4 h5 h6 h7 server X
server -> h1 h2 h3 h4 h5 h6 h7 h8 X
untrusted -> X X X X X X X X X
*** Results: 20% dropped (72/90 received)
mininet>
```

Figure 3 Pingall Command Output

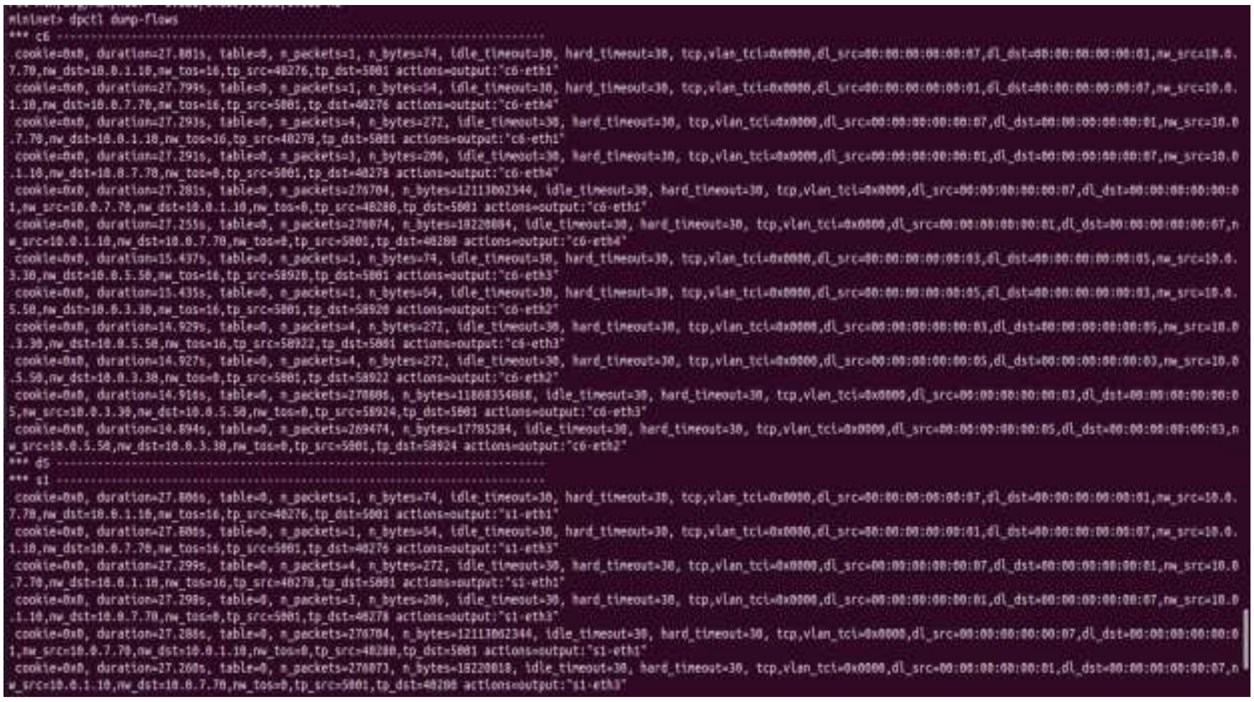
To determine TCP bandwidth between two hosts we used iperf command. Since The untrusted host is not allowed to send or receive any TCP packets the command is unsuccessful as shown and it needed to kill manually. Also, untrusted host can’t send any IP traffic to the server, so the packet is dropped. The command is stuck in a loop of retransmission due to timeout and the program had to be manually killed.



```
mininet>
mininet>
mininet>
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
.*** Results: ['19.5 Gbits/sec', '19.5 Gbits/sec']
mininet> iperf h1 untrusted
*** Iperf: testing TCP bandwidth between h1 and untrusted
^C
Interrupt
stopping h1
stopping untrusted
mininet> iperf h2 untrusted
*** Iperf: testing TCP bandwidth between h2 and untrusted
^C
Interrupt
stopping h2
stopping untrusted
mininet> iperf untrusted server
*** Iperf: testing TCP bandwidth between untrusted and server
^C
Interrupt
stopping server
stopping untrusted
```

Figure 4 TCP bandwidth between two hosts

The dpctl command line utility is used to monitor and administer OpenFlow data routes. It may display a Datapath's current status, including features, settings, and table entries. dpctl is used to add, delete, alter, and monitor data pathways when utilising the OpenFlow kernel module. The command dpctl dump-flows prints all of the flow table's entries.



```
mininet> dpctl dump-flows
*** c6 ------------------------------------------------------------------------
 cookie=0x0, duration=27.801s, table=0, n_packets=1, n_bytes=74, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:01,nw_src=10.0.
7.70,nw_dst=10.0.1.10,nw_tos=16,tp_src=40276,tp_dst=5001 actions=output:"c6-eth1"
 cookie=0x0, duration=27.799s, table=0, n_packets=1, n_bytes=54, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:07,nw_src=10.0.
1.10,nw_dst=10.0.7.70,nw_tos=16,tp_src=5001,tp_dst=40276 actions=output:"c6-eth4"
 cookie=0x0, duration=27.293s, table=0, n_packets=4, n_bytes=272, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:01,nw_src=10.0
.7.70,nw_dst=10.0.1.10,nw_tos=16,tp_src=40278,tp_dst=5001 actions=output:"c6-eth1"
 cookie=0x0, duration=27.291s, table=0, n_packets=3, n_bytes=206, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:07,nw_src=10.0
.1.10,nw_dst=10.0.7.70,nw_tos=0,tp_src=5001,tp_dst=40278 actions=output:"c6-eth4"
 cookie=0x0, duration=27.281s, table=0, n_packets=276704, n_bytes=12113002344, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:0
1,nw_src=10.0.7.70,nw_dst=10.0.1.10,nw_tos=0,tp_src=40280,tp_dst=5001 actions=output:"c6-eth1"
 cookie=0x0, duration=27.255s, table=0, n_packets=270074, n_bytes=18220804, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:07,n
w_src=10.0.1.10,nw_dst=10.0.7.70,nw_tos=0,tp_src=5001,tp_dst=40280 actions=output:"c6-eth4"
 cookie=0x0, duration=15.437s, table=0, n_packets=1, n_bytes=74, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05,nw_src=10.0.
3.30,nw_dst=10.0.5.50,nw_tos=16,tp_src=58920,tp_dst=5001 actions=output:"c6-eth3"
 cookie=0x0, duration=15.435s, table=0, n_packets=1, n_bytes=54, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03,nw_src=10.0.
5.50,nw_dst=10.0.3.30,nw_tos=16,tp_src=5001,tp_dst=58920 actions=output:"c6-eth2"
 cookie=0x0, duration=14.929s, table=0, n_packets=4, n_bytes=272, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:05,nw_src=10.0
.3.30,nw_dst=10.0.5.50,nw_tos=16,tp_src=58922,tp_dst=5001 actions=output:"c6-eth3"
 cookie=0x0, duration=14.927s, table=0, n_packets=4, n_bytes=272, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03,nw_src=10.0
.5.50,nw_dst=10.0.3.30,nw_tos=0,tp_src=5001,tp_dst=58922 actions=output:"c6-eth2"
 cookie=0x0, duration=14.916s, table=0, n_packets=270806, n_bytes=11808354088, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:03,dl_dst=00:00:00:00:00:0
5,nw_src=10.0.3.30,nw_dst=10.0.5.50,nw_tos=0,tp_src=58924,tp_dst=5001 actions=output:"c6-eth3"
 cookie=0x0, duration=14.894s, table=0, n_packets=269474, n_bytes=17785284, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:05,dl_dst=00:00:00:00:00:03,n
w_src=10.0.5.50,nw_dst=10.0.3.30,nw_tos=0,tp_src=5001,tp_dst=58924 actions=output:"c6-eth2"
*** d5 ------------------------------------------------------------------------
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=27.806s, table=0, n_packets=1, n_bytes=74, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:01,nw_src=10.0.
7.70,nw_dst=10.0.1.10,nw_tos=16,tp_src=40276,tp_dst=5001 actions=output:"s1-eth1"
 cookie=0x0, duration=27.800s, table=0, n_packets=1, n_bytes=54, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:07,nw_src=10.0.
1.10,nw_dst=10.0.7.70,nw_tos=16,tp_src=5001,tp_dst=40276 actions=output:"s1-eth3"
 cookie=0x0, duration=27.299s, table=0, n_packets=4, n_bytes=272, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:01,nw_src=10.0
.7.70,nw_dst=10.0.1.10,nw_tos=16,tp_src=40278,tp_dst=5001 actions=output:"s1-eth1"
 cookie=0x0, duration=27.298s, table=0, n_packets=3, n_bytes=206, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:07,nw_src=10.0
.1.10,nw_dst=10.0.7.70,nw_tos=0,tp_src=5001,tp_dst=40278 actions=output:"s1-eth3"
 cookie=0x0, duration=27.286s, table=0, n_packets=276704, n_bytes=12113002344, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:07,dl_dst=00:00:00:00:00:0
1,nw_src=10.0.7.70,nw_dst=10.0.1.10,nw_tos=0,tp_src=40280,tp_dst=5001 actions=output:"s1-eth1"
 cookie=0x0, duration=27.260s, table=0, n_packets=270073, n_bytes=18220818, idle_timeout=30, hard_timeout=30, tcp,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:07,n
w_src=10.0.1.10,nw_dst=10.0.7.70,nw_tos=0,tp_src=5001,tp_dst=40280 actions=output:"s1-eth3"
```

Figure 5 Flow Table Entries

CONCLUSION

Using the POX Controller, we built a Layer-2 Firewall in this project. We were able to manage the switches using POX utilizing rules describing Layer-2 attributes to either permit or prohibit communication between hosts. A firewall is a network component that monitors and regulates all network traffic as well as having the ability to identify and block unwanted activity. Installing physical firewalls can be costly in terms of hardware and device upgrades. When a firewall is physically removed, reconfiguring each firewall-related system for troubleshooting becomes a big challenge. SDN is a game-changer not just in terms of making control more flexible and controllable, but also in terms of achieving programmability in firewalls by isolating the firewall hardware from the control software. POX is a deployment and evaluation tool for SDN applications. Its main advantage is that it may be used with real hardware, test beds, or the Mininet emulator.

Our work is limited by the fact that our firewall does not maintain track of the status of the connection, making it stateless. The limitation is due to the use of OpenFlow version 1.0, which does not maintain track of packet status and only supports a few header fields. Implementing a stateful firewall might be a future project. Also, in this project we implemented firewall only on software implementing that on hardware can be future project. Currently, the firewall design only looks at the packet header fields to determine the action. Future developers may be able to enhance this logic by adding SDN capabilities to increase security by watching the complete network flow and efficiently blocking network assaults early on without the need for deep packet inspection.

REFERENCES

1) Karamjeet Kaur, Krishan Kumar, Japinder Singh, Navtej Singh Ghumman, “Programmable firewall using Software Defined Networking” 2nd IEEE International Conference on Computing for Sustainable Global Development (INDIACom) 11-13 March 2015.
2) Sukhveer Kaur, Karamjeet Kaur, Vipin Gupta, “Implementing Openflow Based Firewall”, IEEE International Conference on Information Technology (InCITe) - The Next Generation IT Summit on the Theme - Internet of Things: Connect your Worlds 6-7 Oct. 2016.
3) Michelle Suh, Sae Hyong Park, Byungjoon Lee, Sunhee Yang, “Building firewall over the software-defined network controller”, 16th IEEE International Conference on Advanced Communication Technology 16-19 Feb. 2014.
4) Sukhveer Kaur, Japinder Singh and Navtej Singh Ghumman, “Network Programmability Using POX Controller”, International Conference on Communication, Computing & Systems (ICCCS) August 2014.
5) Dhaval Satasiya, Raviya Rupal, “Analysis of Software Defined Network Firewall (SDF)”, IEEE WiSPNET 2016 conference March 2016.