Student Name: Shachi Shah

Email: spshah22@asu.edu

Submission Date: 8 June 2025

Class Name and Term: CSE548 Summer 2025

SDN-Based Stateless Firewall Project

I. Project Overview

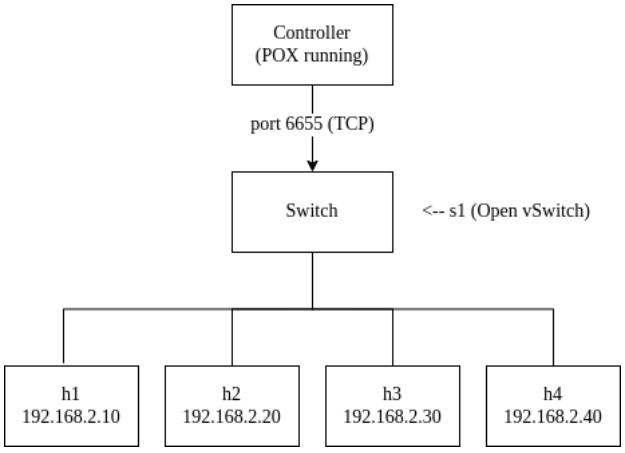
This project explores the design and implementation of a stateless firewall using Software Defined Networking (SDN) principles. In traditional networks, firewalls are often hardware-based and integrated into fixed network infrastructure. In contrast, this project demonstrates a software-defined approach where control logic is decoupled from the data plane. The SDN controller dynamically manages packet forwarding rules, allowing for centralized and programmable security policies. This approach provides greater flexibility, easier rule management, and a clearer separation of concerns between network logic and traffic forwarding.

The network topology is emulated using Mininet, which creates four virtual container hosts connected to a single Open vSwitch instance. A POX controller, running as a separate process, acts as the network's control plane and manages flow entries using the OpenFlow 1.0 protocol. The controller is extended with a custom firewall application that parses configuration files to enforce both Layer 2 (MAC-based) and Layer 3 (IP and port-based) firewall rules. These rules are pushed to the switch to block unwanted traffic, such as specific ICMP echo requests, TCP connections, or UDP datagrams, based on the defined criteria.

By simulating various traffic scenarios and observing how packets are filtered or forwarded, the project validates the correctness and functionality of the stateless firewall. Tools such as hping3 and ping are used to test traffic flows between hosts, while ovs-ofctl confirms the installation and behavior of flow rules. The results show that the firewall can effectively enforce access control policies without maintaining any connection state, demonstrating the power and simplicity of stateless SDN-based firewalls in programmable network environments.

II. Network Setup

Topology:



The network topology includes one Open vSwitch (s1), four hosts (h1–h4), and a remote POX controller. Each host is assigned the following IPs:

- h1: 192.168.2.10
- h2: 192.168.2.20
- h3: 192.168.2.30
- h4: 192.168.2.40

Initial reachability is confirmed via successful pings prior to applying the firewall.

III. Software

This project used several key tools to implement and test a stateless firewall within a software-defined networking (SDN) environment. Mininet was used to emulate a virtual network topology consisting of four hosts and a single Open vSwitch (OVS) switch, all controlled remotely by a POX controller. The topology was created and managed through Mininet's command-line interface, which allowed configuration of hosts, links, and interfaces.

The POX controller, a Python-based OpenFlow controller, was extended with a custom L3Firewall module that read rules from configuration files and installed matching OpenFlow 1.0 flow entries on the switch. These rules specified filtering behavior based on Layer 2 and Layer 3 fields, such as MAC and IP addresses, ports, and protocols.

Open vSwitch acted as the data plane switch, enforcing flow rules sent by the controller to drop or allow traffic as specified. Tools like ovs-ofctl were used to inspect flow tables and confirm that the correct rules had been installed and triggered.

To test firewall behavior, xterm was used to open individual host terminals, and hping3 was used to generate ICMP, TCP, and UDP packets for testing. This setup allowed precise validation of firewall rules and made it easy to observe which traffic was blocked or permitted.

IV. Project Description

My step-by-step procedure on how to demo using Mininet and the POX controller to implement a firewall using OpenFlow rules:

1. Start the POX controller with L3Firewall.py and the configuration files using:
    a. sudo ./pox.py openflow.of_01 --port=6655 pox.forwarding.L3Firewall --l2config="l2firewall.config" --l3config="l3firewall.config"
2. Launch Mininet with four hosts:
    a. sudo mn --topo=single,4 --controller=remote,port=6655 --switch=ovsk --mac
3. Assign static IPs to each host:
    a. h1 ifconfig h1-eth0 192.168.2.10
    b. h2 ifconfig h2-eth0 192.168.2.20
    c. h3 ifconfig h3-eth0 192.168.2.30
    d. h4 ifconfig h4-eth0 192.168.2.40
4. Open xterm windows:
    a. xterm h1
    b. xterm h2
5. Use ping and hping3 to simulate blocked and allowed traffic:
    a. ping 192.168.2.30 (blocked)
    b. hping3 -c 5 192.168.2.20 --tcp-timestamp (blocked)
    c. hping3 -c 5 192.168.2.20 --udp (blocked)
    d. ping 192.168.2.40 (allowed)
6. Verify OpenFlow rules installed with:
    a. sudo ovs-ofctl dump-flows s1
7. Observe that drop rules for specific traffic are active, based on IP, port, and protocol.

V. Conclusion

This project provided valuable hands-on experience with Software-Defined Networking (SDN) and the implementation of a stateless firewall using the POX controller and OpenFlow. One key takeaway was understanding how traffic can be programmatically controlled using flow rules based on L2 and L3 criteria. I found it especially interesting how specific packet types (like ICMP, TCP, and UDP) could be selectively blocked or allowed purely by modifying configuration files and without touching the underlying applications.

A helpful tip I discovered during testing was the usefulness of tools like hping3 and xterm to simulate various traffic types and isolate issues with rule enforcement. Additionally, ovs-ofctl dump-flows s1 proved to be a critical command for verifying whether the POX controller successfully installed the expected flow rules.

From a self-assessment perspective, I believe the project was implemented successfully. I was able to configure the network, apply the correct rules, and validate their effects through structured testing. Debugging initial connection and configuration issues taught me to pay close attention to controller port settings and flow priorities.

Overall, this project was well-designed and allowed me to connect theoretical knowledge about SDN with practical implementation skills. I appreciate how it emphasized problem-solving and exploration, and I now feel more confident working with network emulators like Mininet and controller frameworks like POX.

VI. Appendix B: Attached files

This configuration file defines L3 (network-layer) firewall rules for the POX controller. Each rule controls traffic based on IP addresses, transport-layer ports, and protocol type (e.g., ICMP, TCP, UDP).

- l2firewall.config

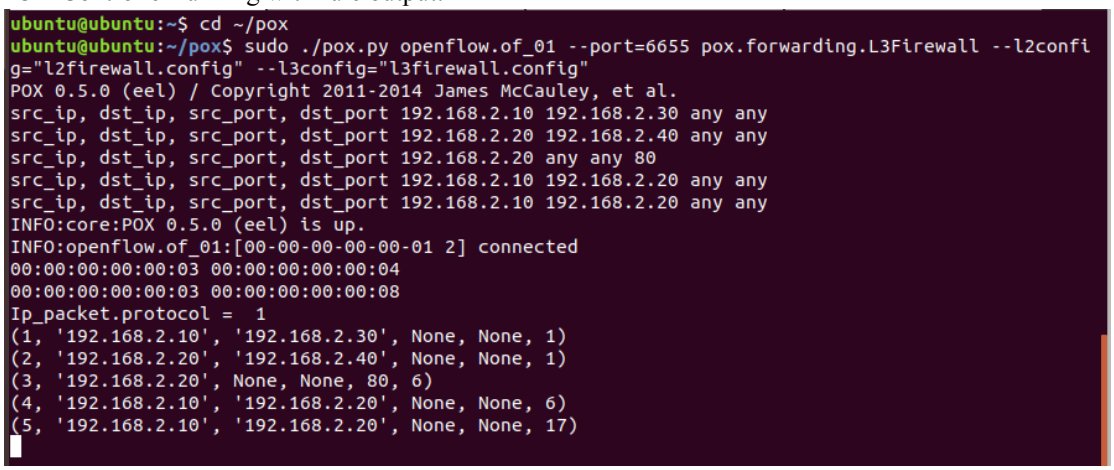
```
priority,src_mac,dst_mac
# Shachi Shah: Block MAC traffic from h2 to h4
1,00:00:00:00:00:02,00:00:00:00:00:04
```

- l3firewall.config

```
priority,src_mac,dst_mac,src_ip,dst_ip,src_port,dst_port,nw_proto
# Shachi Shah: Block ICMP from h1 to h3
1,any,any,192.168.2.10,192.168.2.30,any,any,icmp

# Shachi Shah: Block ICMP from h2 to h4
2,any,any,192.168.2.20,192.168.2.40,any,any,icmp

# Shachi Shah: Block HTTP (TCP port 80) from h2
3,any,any,192.168.2.20,any,any,80,tcp

# Shachi Shah: Block TCP from h1 to h2
4,any,any,192.168.2.10,192.168.2.20,any,any,tcp

# Shachi Shah: Block UDP from h1 to h2
5,any,any,192.168.2.10,192.168.2.20,any,any,udp
```

Screenshots:

- POX Controller running with rule output:



- Mininet CLI showing host configuration:

```
ubuntu@ubuntu:~$ sudo mn --topo=single,4 --controller=remote,port=6655 --switch=ovsk --mac
[sudo] password for ubuntu:
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet> h1 ifconfig h1-eth0 192.168.2.10
containernet> h2 ifconfig h2-eth0 192.168.2.20
containernet> h3 ifconfig h3-eth0 192.168.2.30
containernet> h4 ifconfig h4-eth0 192.168.2.40
containernet> xterm h1
containernet> xterm h2
containernet>
```

- xterm h1 tests showing blocked/allowed traffic:

"Node: h1"

```
root@ubuntu:~# ping -c 3 192.168.2.30
PING 192.168.2.30 (192.168.2.30) 56(84) bytes of data.

--- 192.168.2.30 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2031ms

root@ubuntu:~# hping3 -c 5 192.168.2.20 -V --tcp-timestamp
using h1-eth0, addr: 192.168.2.10, MTU: 1500
HPING 192.168.2.20 (h1-eth0 192.168.2.20): NO FLAGS are set, 40 headers + 0 data
 bytes

--- 192.168.2.20 hping statistic ---
5 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@ubuntu:~# hping3 -c 5 192.168.2.20 --udp
HPING 192.168.2.20 (h1-eth0 192.168.2.20): udp mode set, 28 headers + 0 data byt
es

--- 192.168.2.20 hping statistic ---
5 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@ubuntu:~# ping -c 3 192.168.2.40
PING 192.168.2.40 (192.168.2.40) 56(84) bytes of data.
64 bytes from 192.168.2.40: icmp_seq=1 ttl=64 time=0.623 ms
64 bytes from 192.168.2.40: icmp_seq=2 ttl=64 time=0.090 ms
64 bytes from 192.168.2.40: icmp_seq=3 ttl=64 time=0.092 ms

--- 192.168.2.40 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2017ms
rtt min/avg/max/mdev = 0.090/0.268/0.623/0.251 ms
root@ubuntu:~#
```

- sudo ovs-ofctl dump-flows s1 result:

```
ubuntu@ubuntu:~$ sudo ovs-ofctl dump-flows s1
 cookie=0x0, duration=355.506s, table=0, n_packets=0, n_bytes=0, priority=65535,dl_src=00:00:00:0
0:00:03,dl_dst=00:00:00:00:00:04 actions=drop
 cookie=0x0, duration=355.506s, table=0, n_packets=0, n_bytes=0, priority=65535,dl_src=00:00:00:0
0:00:03,dl_dst=00:00:00:00:00:08 actions=drop
 cookie=0x0, duration=187.800s, table=0, n_packets=2, n_bytes=196, idle_timeout=200, priority=500
01,icmp,nw_src=192.168.2.10,nw_dst=192.168.2.30 actions=drop
 cookie=0x0, duration=187.759s, table=0, n_packets=0, n_bytes=0, idle_timeout=200, priority=50002
,icmp,nw_src=192.168.2.20,nw_dst=192.168.2.40 actions=drop
 cookie=0x0, duration=187.760s, table=0, n_packets=5, n_bytes=330, idle_timeout=200, priority=500
04,tcp,nw_src=192.168.2.10,nw_dst=192.168.2.20 actions=drop
 cookie=0x0, duration=187.760s, table=0, n_packets=5, n_bytes=210, idle_timeout=200, priority=500
05,udp,nw_src=192.168.2.10,nw_dst=192.168.2.20 actions=drop
 cookie=0x0, duration=187.760s, table=0, n_packets=0, n_bytes=0, idle_timeout=200, priority=50003
,tcp,nw_src=192.168.2.20,tp_dst=80 actions=drop
 cookie=0x0, duration=187.760s, table=0, n_packets=14, n_bytes=924, actions=NORMAL
ubuntu@ubuntu:~$
```

VII. References

[1] OpenFlow Protocol, available at https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf, accessed by 06/07/2025.
[2] POX Controller Documentation: available at https://noxrepo.github.io/pox-doc/html/, accessed by 06/07/2025.
[3] Mininet Documentation: available at https://mininet.org/, accessed by 06/07/2025.