Report

Prateek Bhaisora
(2023JCS2564)

November 2023

Commands Used:

- I have used `Ubuntu-18.04.6-LTS` for this assignment on which I installed `Mininet-2.3.1b4` and `Ryu-4.34`.
- Any `Ryu` app, that I made in this assignment, can be run using the following command:

```
ryu-manager file_name.py
```

- I have also created the custom topology mentioned in the given assignment in the file named `my_topo.py`. It can be used as a custom topology in `Mininet` CLI using the following command:

```
sudo mn --custom my_topo.py --topo mytopo --controller remote
```

- Apart from these, one can use the standard `Mininet` and `Ryu` commands to execute the various files of this assignment.

Part 1: Controller Hub and Learning Switch

- I conducted 3 pings between `Host2` and `Host5` using the following command:

```
Host2 ping -c 3 Host5
```

- The latency values are:

1. **Using Controller Hub:** The output is:

```
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5:  icmp_seq=1 ttl=64 time=16.2 ms
64 bytes from 10.0.0.5:  icmp_seq=2 ttl=64 time=6.62 ms
64 bytes from 10.0.0.5:  icmp_seq=3 ttl=64 time=16.7 ms
```

2. **Using Learning Switch:** The output is:

```
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5:  icmp_seq=1 ttl=64 time=19.0 ms
```

```
64 bytes from 10.0.0.5:  icmp_seq=2 ttl=64 time=0.301 ms
64 bytes from 10.0.0.5:  icmp_seq=3 ttl=64 time=0.048 ms
```

⇒ **Explanation:** The latency in the Hub is high as the Hub is a non-intelligent device. Any data it receives, it simply floods it to all the ports, except the port the data came from, thereby leading to random latencies in the test. On the other hand, a learning switch is an intelligent device. It maintains a MAC-to-Port mapping table and uses it to forward data. Initially, the Switch table is empty, hence, it broadcasts data and keeps filling its table dynamically. After the first ping, the entry is added to the lookup table of the switch, hence, the subsequent latencies are much lower compared to the latency of the first ping.

- Then I performed throughput test between `Host1` and `Host5` using the following command:

```
iperf
```

- The throughput test results are:

  1. **Using Controller Hub:** The output is:

     ```
     *** Iperf:  testing TCP bandwidth between Host1 and Host5 *** Results:  ['3.4 Mbits/sec', '4.4 Mbits/
     ```

  2. **Using Learning Switch:** The output is:

     ```
     *** Iperf:  testing TCP bandwidth between Host1 and Host5 *** Results:  ['28.5 Gbits/sec', '28.5
     Gbits/sec']
     ```

⇒ **Explanation:** In a network with a Hub Controller, all the traffic passes through the centralized controller. The controller makes decisions on how to handle each packet and then forwards it to the appropriate ports. The Hub Controller introduces a centralized point of control and processing, which can become a bottleneck, especially in large networks or networks with high traffic. Also, the flooding may lead to collisions, thereby decreasing throughput. On the other hand, a Learning Switch operates at the data plane and makes local forwarding decisions based on its learned MAC address table. It efficiently forwards packets without the need for centralized control, which reduces latency and ensures that packets can be forwarded at wire speed. This decentralized approach provides a better throughput.

- Finally I pinged all hosts with each other and then reported the installed rules on the switches using the following commands:

```
pingall
dpctl dump-flows
```

- The output for `pingall` are:

  1. **Using Controller Hub:** The output is:

     ```
     *** Ping:  testing ping reachability
     ```

```
Host1 -> Host2 Host3 Host4 Host5
Host2 -> Host1 Host3 Host4 Host5
Host3 -> Host1 Host2 Host4 Host5
Host4 -> Host1 Host2 Host3 Host5
Host5 -> Host1 Host2 Host3 Host4
*** Results:  0% dropped (20/20 received)
```

2. **Using Learning Switch:** The output is:

```
*** Ping:  testing ping reachability
Host1 -> Host2 Host3 Host4 Host5
Host2 -> Host1 Host3 Host4 Host5
Host3 -> Host1 Host2 Host4 Host5
Host4 -> Host1 Host2 Host3 Host5
Host5 -> Host1 Host2 Host3 Host4
*** Results:  0% dropped (20/20 received)
```

- The installed rules are:

1. **Using Controller Hub:** The output is:

```
*** Switch1 ----------------------------------------------
cookie=0x0, duration=1104.978s, table=0, n_packets=2158,
n_bytes=3029447, priority=0 actions=CONTROLLER:65535

*** Switch2 ----------------------------------------------
cookie=0x0, duration=1104.987s, table=0, n_packets=3114,
n_bytes=3092523, priority=0 actions=CONTROLLER:65535
```

2. **Using Learning Switch:** The output is:

```
*** Switch1 ----------------------------------------------------------
 cookie=0x0, duration=314.634s, table=0, n_packets=8,
 n_bytes=616, priority=1,in_port="Switch1-eth4",dl_src=ca:44:ca:eb:76:30,
 dl_dst=d2:d2:8c:7d:03:bd actions=output:"Switch1-eth2"
 cookie=0x0, duration=314.632s, table=0, n_packets=7,
 n_bytes=518, priority=1,in_port="Switch1-eth2",
 dl_src=d2:d2:8c:7d:03:bd,
 dl_dst=ca:44:ca:eb:76:30 actions=output:"Switch1-eth4"
 cookie=0x0, duration=225.488s, table=0, n_packets=165463,
 n_bytes=10920866, priority=1,in_port="Switch1-eth4",
 dl_src=ca:44:ca:eb:76:30,dl_dst=ee:ed:ba:36:21:5d actions=output:"Switch1-eth1"
 cookie=0x0, duration=225.483s, table=0, n_packets=363070,
 n_bytes=17856970348, priority=1,in_port="Switch1-eth1",
 dl_src=ee:ed:ba:36:21:5d,dl_dst=ca:44:ca:eb:76:30 actions=output:"Switch1-eth4"
```

⋮

```
*** Switch2 ------------------------------------------------------------
 cookie=0x0, duration=314.657s, table=0, n_packets=8,
 n_bytes=616, priority=1,in_port="Switch2-eth3",dl_src=ca:44:ca:eb:76:30,
 dl_dst=d2:d2:8c:7d:03:bd actions=output:"Switch2-eth1"
 cookie=0x0, duration=314.648s, table=0, n_packets=7,
 n_bytes=518, priority=1,in_port="Switch2-eth1",
 dl_src=d2:d2:8c:7d:03:bd,dl_dst=ca:44:ca:eb:76:30 actions=output:"Switch2-eth3"
 cookie=0x0, duration=225.518s, table=0, n_packets=165463,
 n_bytes=10920866, priority=1,in_port="Switch2-eth3",
 dl_src=ca:44:ca:eb:76:30,dl_dst=ee:ed:ba:36:21:5d actions=output:"Switch2-eth1"
 cookie=0x0, duration=225.497s, table=0, n_packets=363070,
 n_bytes=17856970348, priority=1,in_port="Switch2-eth1",
 dl_src=ee:ed:ba:36:21:5d,dl_dst=ca:44:ca:eb:76:30 actions=output:"Switch2-eth3"
```

⋮

Part 2: Firewall and Monitor

- Next I implemented a `firewall + monitor`, over the `Learning Switch`, that blocks communication between `Host2` and `Host5`, `Host3` and `Host5`, and `Host1` with `Host4`. Also, it counts all the packets coming from `Host3` on `Switch1` and displays the count. To test this, run the following command:

```
pingall
```

- The output is:

```
*** Ping: testing ping reachability
Host1 -> Host2 Host3 X Host5
Host2 -> Host1 Host3 Host4 X
Host3 -> Host1 Host2 Host4 X
Host4 -> X Host2 Host3 Host5
Host5 -> Host1 X X Host4
*** Results: 30% dropped (14/20 received)
```

- Then I reported the installed rules on both switches using the following command:

```
dpctl dump-flows
```

- The output *(logically-depicted)* is:

Switch1				
Cookie	Duration	Table	n_packets, n_bytes	...
0x0	361.458s	0	3, 238	...
0x0	361.451s	0	2, 140	...
0x0	361.418s	0	3, 238	...
0x0	361.414s	0	2, 140	...
0x0	361.393s	0	4, 224	...
0x0	351.374s	0	4, 280	...
0x0	351.368s	0	3, 182	...
0x0	351.328s	0	3, 238	...
0x0	351.321s	0	2, 140	...
0x0	351.292s	0	3, 238	...
0x0	351.277s	0	2, 140	...
0x0	351.242s	0	4, 224	...
0x0	341.166s	0	3, 238	...

⋮

Switch2				
Cookie	Duration	Table	n_packets, n_bytes	...
0x0	361.416s	0	4, 224	...
0x0	351.402s	0	4, 280	...
0x0	351.383s	0	3, 182	...
0x0	351.317s	0	3, 238	...
0x0	351.291s	0	2, 140	...
0x0	351.266s	0	4, 224	...
0x0	341.188s	0	3, 238	...
0x0	341.180s	0	2, 140	...
0x0	341.162s	0	4, 224	...
0x0	321.066s	0	3, 238	...
0x0	321.063s	0	2, 140	...
0x0	365.453s	0	102, 8481	...

⋮

- Also, the `firewall + monitor` printed the count of the number of packets coming from Host3 on Switch1 as:

```
anager firewall_monitor.py
loading app firewall_monitor.py
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app firewall_monitor.py of FirewallMonitor
instantiating app ryu.topology.switches of Switches
```

```
instantiating app ryu.controller.ofp_handler of OFPHandler
Packet count from Host3 on Switch1: 1
Packet count from Host3 on Switch1: 2
Packet count from Host3 on Switch1: 3
Packet count from Host3 on Switch1: 4
Packet count from Host3 on Switch1: 5
Packet count from Host3 on Switch1: 6
Packet count from Host3 on Switch1: 7
Packet count from Host3 on Switch1: 8
```

$\vdots$

- To minimize the number of firewall rules on the switch, one can use wildcard rules and OpenFlow groups. Wildcard rules can match multiple fields in a packet, reducing the need for explicit rules for each combination. OpenFlow groups allow us to group multiple rules into a single entry, which can be more efficient.
  For example, instead of having separate rules for each source-destination pair, one can use wildcard rules and group them by the action to take (e.g., block or allow).
- To implement firewall policies in real-time without interference from pre-existing rules, one can follow these steps:
  1. **Add rules with appropriate priorities:** When adding new firewall rules, assign higher priorities to the new rules. OpenFlow controllers generally process rules based on priority, with higher-priority rules taking precedence. New rules with higher priorities will override conflicting pre-existing rules.
  2. **Explicitly delete or modify conflicting pre-existing rules:** If she wants to replace or modify pre-existing rules, she can send OpenFlow messages to delete or modify those specific rules based on their match criteria.
  3. **Carefully manage rule conflicts:** Ensure that her new firewall policies are carefully designed to avoid conflicts with pre-existing rules. Test the new policies thoroughly in a controlled environment before deploying them in a production network.
  4. **Use FlowMod messages:** Use FlowMod messages to add, modify, or delete rules in real-time. This allows her to make dynamic changes to the firewall policies without affecting the entire switch configuration.

Part 3: Load Balancer

- Finally I implemented the load balancer, in which `Host4` and `Host5` acted as servers, that evenly distributed requests from the remaining hosts to these servers in a round-robin fashion. The hosts, `Host1`, `Host2`, and `Host3`, connected to the servers via a virtual IP address, specifically 10.0.0.42.
- Then I tested it by running the following command:

```
pingall
```

- The output is:

```
loading app load_balancer.py
loading app ryu.controller.ofp_handler
instantiating app load_balancer.py of LoadBalancer
instantiating app ryu.controller.ofp_handler of OFPHandler
**************************
---Handle TCP Packet---
TCP packet handled: False
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 1 00:00:00:00:00:03 00:00:00:00:00:01 3
packet in 1 00:00:00:00:00:01 00:00:00:00:00:03 1
**************************
---Handle TCP Packet---
TCP packet handled: False
packet in 1 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 2 00:00:00:00:00:01 ff:ff:ff:ff:ff:ff 1
packet in 2 00:00:00:00:00:04 00:00:00:00:00:01 2
packet in 1 00:00:00:00:00:04 00:00:00:00:00:01 4
packet in 1 00:00:00:00:00:01 00:00:00:00:00:04 1
```

⋮

- To implement a load balancing policy that considers the load on these servers, I would typically need to take the following additional steps:

  1. **Monitor Server Load:** I need to monitor the load on the servers (e.g., CPU utilization, memory usage, network traffic, etc.) using various monitoring tools and mechanisms to collect real-time data about server performance.
  2. **Define Load Balancing Metrics:** I will have to decide on the load balancing metrics or criteria that I'd like to consider, e.g., server CPU utilization, the number of active connections to determine the server's load, etc.
  3. **Implement a Load Balancing Algorithm:** Based on the load balancing metrics, I will have to implement a load balancing algorithm or policy. Common load balancing algorithms include Round Robin, Least Connections, Least Response Time, and Weighted Round Robin, among others.
  4. **Server Addition and Removal:** Implement a logic to add or remove servers from the load balancing pool dynamically. This allows me to scale my application horizontally by adding more servers to handle the increased loads or gracefully removing servers for maintenance or when they become unhealthy.
  5. **Integration with Load Balancer:** Integrate your load balancing policy with a dedicated load balancer or use your SDN controller to dynamically program flow rules based on the load balancing decisions.