COL334: Assignment 3

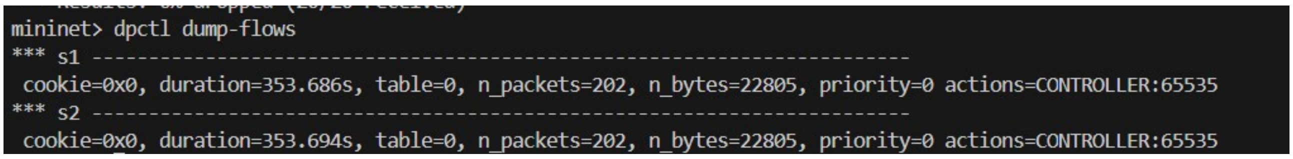
An Introduction to the World of SDN

Ujjwal Yadav(2021EE10669)
Rohan Das(2021EE10621)

Date of Submission: October 07, 2024

1 Part1: Controller Hub and Learning Switch

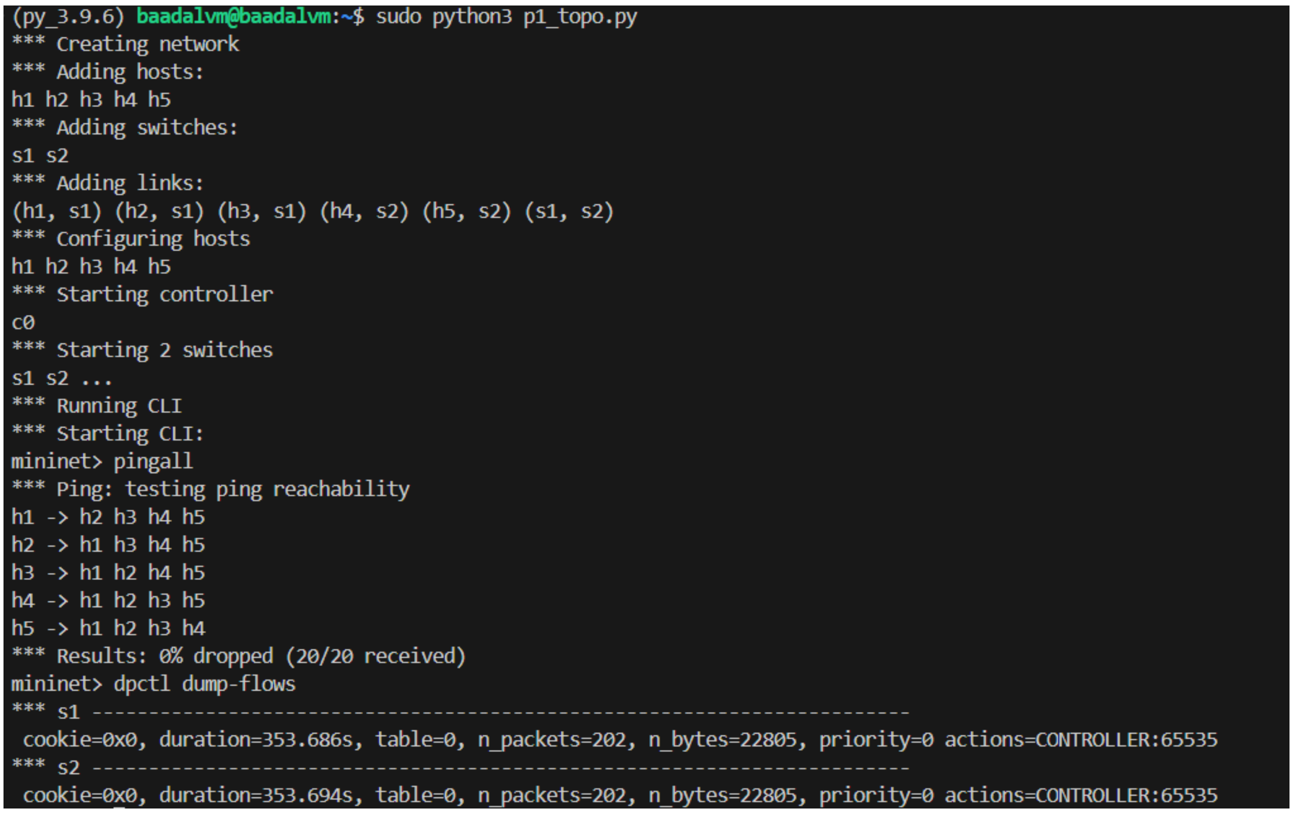
1.1 Hub Controller



```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=353.686s, table=0, n_packets=202, n_bytes=22805, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=353.694s, table=0, n_packets=202, n_bytes=22805, priority=0 actions=CONTROLLER:65535
```

Figure 1: Rules

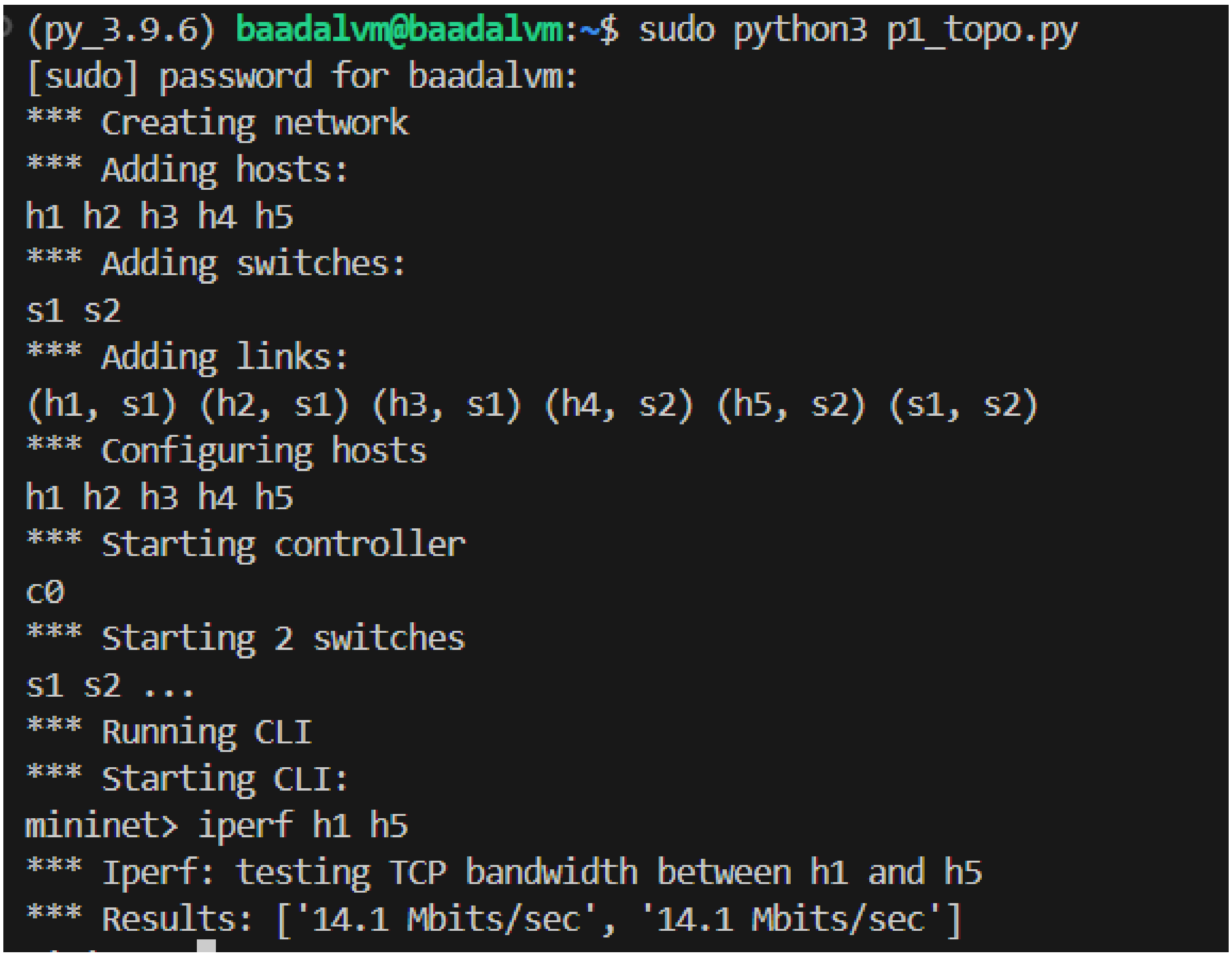
- The flow dump shows that the installed rules in both switches (s1 and s2) have the action CONTROLLER:65535. This means that all traffic is being directed to the controller, which then floods it to all other ports.
- There are no specific flow rules based on the learned MAC addresses, as the Hub Controller is not capable of learning or optimizing traffic paths.



```
(py_3.9.6) baadalvm@baadalvm:~$ sudo python3 p1_topo.py
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Running CLI
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=353.686s, table=0, n_packets=202, n_bytes=22805, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=353.694s, table=0, n_packets=202, n_bytes=22805, priority=0 actions=CONTROLLER:65535
```

Figure 2: Pingall Output

- The pingall test shows that all hosts are reachable, and there is 0% packet loss (20/20 received). This indicates that the Hub Controller ensures connectivity between all hosts.
- However, since the Hub Controller floods all traffic across every switch port (except the incoming port), the network suffers from **high unnecessary traffic**, which can increase latency and congestion.

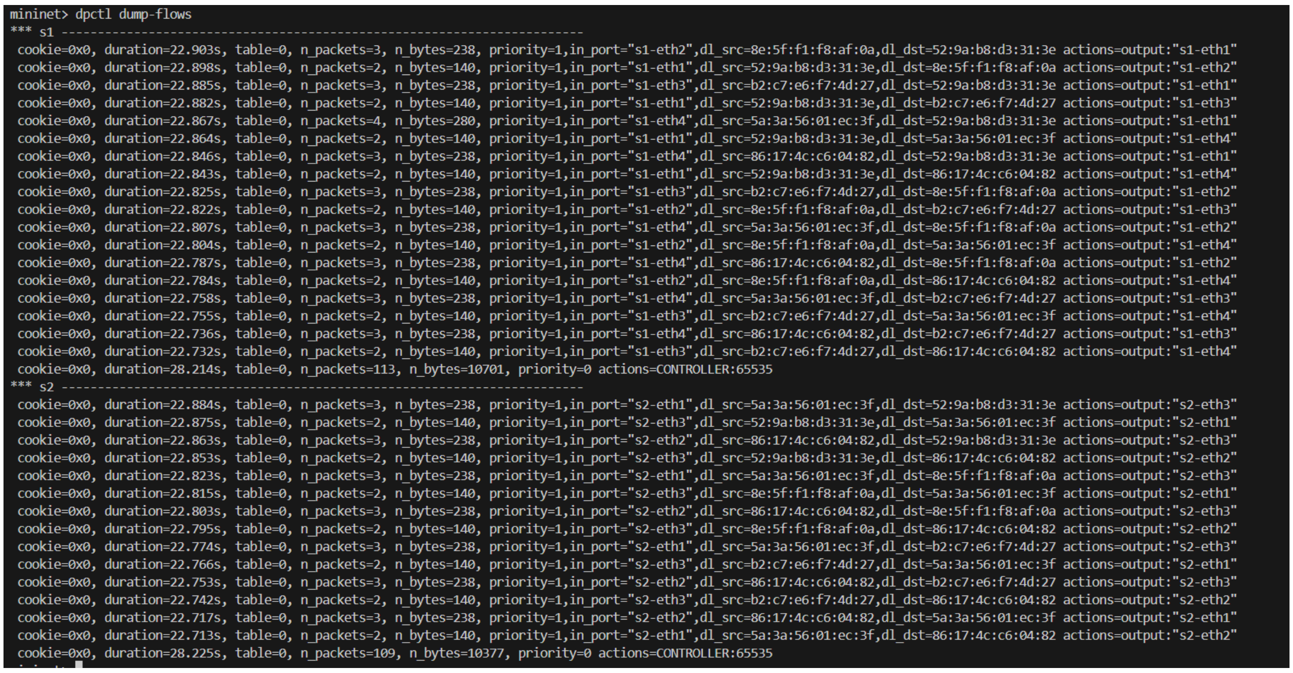


```
(py_3.9.6) baadalvm@baadalvm:~$ sudo python3 p1_topo.py
[sudo] password for baadalvm:
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Running CLI
*** Starting CLI:
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['14.1 Mbits/sec', '14.1 Mbits/sec']
```

Figure 3: TCP Bandwidth between Host 1 and Host 5

- The observed bandwidth between h1 and h5 is **14.1 Mbits/sec**.
- This relatively low throughput can be attributed to the **network flooding** nature of the Hub Controller, where all traffic is broadcasted across the network, leading to collisions, congestion, and inefficient use of bandwidth.

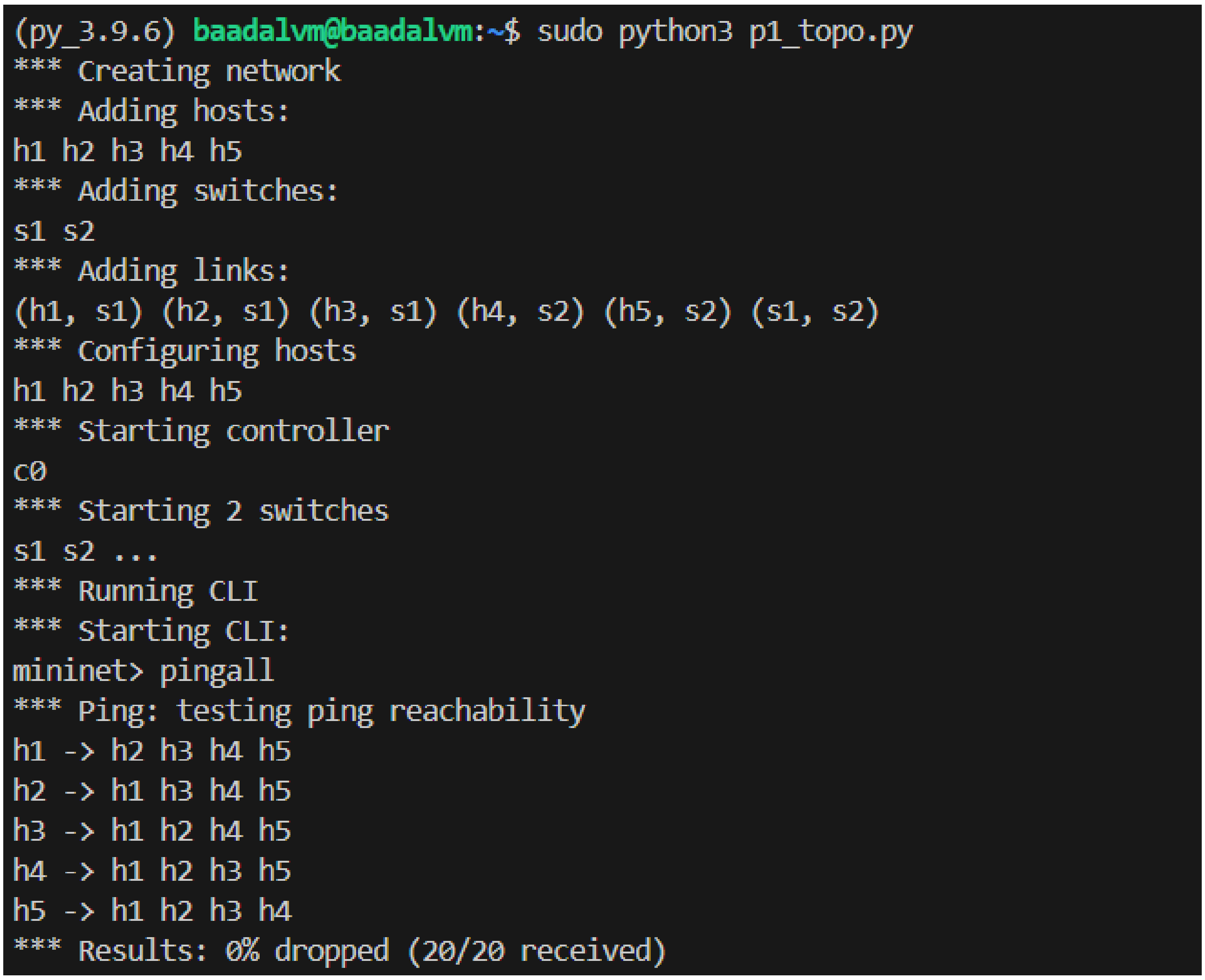
1.2 Learning Switch



```
mininet> dpctl dump-flows
*** s1 ----------------------------------------------------------------------
 cookie=0x0, duration=22.903s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_src=8e:5f:f1:f8:af:0a,dl_dst=52:9a:b8:d3:31:3e actions=output:"s1-eth1"
 cookie=0x0, duration=22.898s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=52:9a:b8:d3:31:3e,dl_dst=8e:5f:f1:f8:af:0a actions=output:"s1-eth2"
 cookie=0x0, duration=22.885s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=b2:c7:e6:f7:4d:27,dl_dst=52:9a:b8:d3:31:3e actions=output:"s1-eth1"
 cookie=0x0, duration=22.882s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=52:9a:b8:d3:31:3e,dl_dst=b2:c7:e6:f7:4d:27 actions=output:"s1-eth3"
 cookie=0x0, duration=22.867s, table=0, n_packets=4, n_bytes=280, priority=1,in_port="s1-eth4",dl_src=5a:3a:56:01:ec:3f,dl_dst=52:9a:b8:d3:31:3e actions=output:"s1-eth1"
 cookie=0x0, duration=22.864s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=52:9a:b8:d3:31:3e,dl_dst=5a:3a:56:01:ec:3f actions=output:"s1-eth4"
 cookie=0x0, duration=22.846s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=86:17:4c:c6:04:82,dl_dst=52:9a:b8:d3:31:3e actions=output:"s1-eth1"
 cookie=0x0, duration=22.843s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_src=52:9a:b8:d3:31:3e,dl_dst=86:17:4c:c6:04:82 actions=output:"s1-eth4"
 cookie=0x0, duration=22.825s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_src=b2:c7:e6:f7:4d:27,dl_dst=8e:5f:f1:f8:af:0a actions=output:"s1-eth2"
 cookie=0x0, duration=22.822s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_src=8e:5f:f1:f8:af:0a,dl_dst=b2:c7:e6:f7:4d:27 actions=output:"s1-eth3"
 cookie=0x0, duration=22.807s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=5a:3a:56:01:ec:3f,dl_dst=8e:5f:f1:f8:af:0a actions=output:"s1-eth2"
 cookie=0x0, duration=22.804s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_src=8e:5f:f1:f8:af:0a,dl_dst=5a:3a:56:01:ec:3f actions=output:"s1-eth4"
 cookie=0x0, duration=22.787s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=86:17:4c:c6:04:82,dl_dst=8e:5f:f1:f8:af:0a actions=output:"s1-eth2"
 cookie=0x0, duration=22.784s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_src=8e:5f:f1:f8:af:0a,dl_dst=86:17:4c:c6:04:82 actions=output:"s1-eth4"
 cookie=0x0, duration=22.758s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=5a:3a:56:01:ec:3f,dl_dst=b2:c7:e6:f7:4d:27 actions=output:"s1-eth3"
 cookie=0x0, duration=22.755s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth3",dl_src=b2:c7:e6:f7:4d:27,dl_dst=5a:3a:56:01:ec:3f actions=output:"s1-eth4"
 cookie=0x0, duration=22.736s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth4",dl_src=86:17:4c:c6:04:82,dl_dst=b2:c7:e6:f7:4d:27 actions=output:"s1-eth3"
 cookie=0x0, duration=22.732s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth3",dl_src=b2:c7:e6:f7:4d:27,dl_dst=86:17:4c:c6:04:82 actions=output:"s1-eth4"
 cookie=0x0, duration=28.214s, table=0, n_packets=113, n_bytes=10701, priority=0 actions=CONTROLLER:65535
*** s2 ----------------------------------------------------------------------
 cookie=0x0, duration=22.884s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth1",dl_src=5a:3a:56:01:ec:3f,dl_dst=52:9a:b8:d3:31:3e actions=output:"s2-eth3"
 cookie=0x0, duration=22.875s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth3",dl_src=52:9a:b8:d3:31:3e,dl_dst=5a:3a:56:01:ec:3f actions=output:"s2-eth1"
 cookie=0x0, duration=22.863s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_src=86:17:4c:c6:04:82,dl_dst=52:9a:b8:d3:31:3e actions=output:"s2-eth3"
 cookie=0x0, duration=22.853s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth3",dl_src=52:9a:b8:d3:31:3e,dl_dst=86:17:4c:c6:04:82 actions=output:"s2-eth2"
 cookie=0x0, duration=22.823s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth1",dl_src=5a:3a:56:01:ec:3f,dl_dst=8e:5f:f1:f8:af:0a actions=output:"s2-eth3"
 cookie=0x0, duration=22.815s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth3",dl_src=8e:5f:f1:f8:af:0a,dl_dst=5a:3a:56:01:ec:3f actions=output:"s2-eth1"
 cookie=0x0, duration=22.803s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_src=86:17:4c:c6:04:82,dl_dst=8e:5f:f1:f8:af:0a actions=output:"s2-eth3"
 cookie=0x0, duration=22.795s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth3",dl_src=8e:5f:f1:f8:af:0a,dl_dst=86:17:4c:c6:04:82 actions=output:"s2-eth2"
 cookie=0x0, duration=22.774s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth1",dl_src=5a:3a:56:01:ec:3f,dl_dst=b2:c7:e6:f7:4d:27 actions=output:"s2-eth3"
 cookie=0x0, duration=22.766s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth3",dl_src=b2:c7:e6:f7:4d:27,dl_dst=5a:3a:56:01:ec:3f actions=output:"s2-eth1"
 cookie=0x0, duration=22.753s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_src=86:17:4c:c6:04:82,dl_dst=b2:c7:e6:f7:4d:27 actions=output:"s2-eth3"
 cookie=0x0, duration=22.742s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth3",dl_src=b2:c7:e6:f7:4d:27,dl_dst=86:17:4c:c6:04:82 actions=output:"s2-eth2"
 cookie=0x0, duration=22.717s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_src=86:17:4c:c6:04:82,dl_dst=5a:3a:56:01:ec:3f actions=output:"s2-eth1"
 cookie=0x0, duration=22.713s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth1",dl_src=5a:3a:56:01:ec:3f,dl_dst=86:17:4c:c6:04:82 actions=output:"s2-eth2"
 cookie=0x0, duration=28.225s, table=0, n_packets=109, n_bytes=10377, priority=0 actions=CONTROLLER:65535
```

Figure 4: Rules

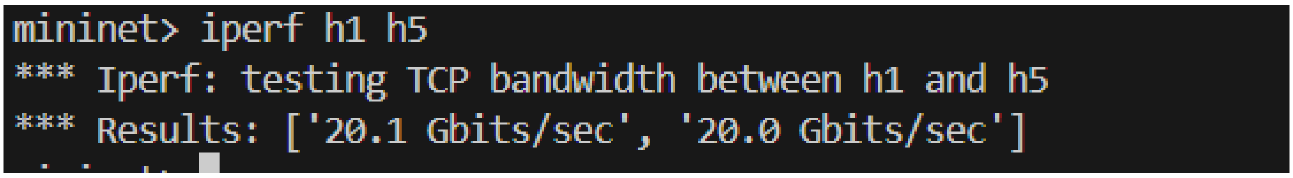
- The flow dump shows several flow rules installed in the switches. These rules are based on the MAC-to-port mappings that the Learning Switch learns from incoming traffic. For example:
  - in port= s1-eth2 ,dl src=...,dl dst=... actions=output:s1-eth1
- These rules allow the switch to forward packets directly to the correct destination port based on the learned MAC addresses, resulting in more efficient packet forwarding.



```
(py_3.9.6) baadalvm@baadalvm:~$ sudo python3 p1_topo.py
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s2) (h5, s2) (s1, s2)
*** Configuring hosts
h1 h2 h3 h4 h5
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Running CLI
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

Figure 5: Pingall Output

- Similar to the Hub Controller, the pingall test results show 0% packet loss (20/20 received). This indicates that the Learning Switch also ensures complete connectivity between the hosts.



```
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['20.1 Gbits/sec', '20.0 Gbits/sec']
```

Figure 6: TCP Bandwidth between Host 1 and Host 5

- The observed bandwidth between h1 and h5 is **20.1 Gbits/sec**, which is significantly higher than the Hub Controller's result.
- This improvement is due to the **optimized forwarding** performed by the Learning Switch, where packets are sent directly to their destination ports, reducing collisions, network congestion, and improving throughput.

1.3 Comparison

- **Traffic Handling:**
  - The Hub Controller floods all traffic, resulting in high unnecessary traffic, while the Learning Switch learns MAC addresses and forwards traffic directly to the appropriate port, reducing network congestion.
- **Performance:**
  - The Hub Controller suffers from lower throughput **(14.1 Mbits/sec)**, while the Learning Switch shows a much higher throughput **(20.1 Gbits/sec)**, indicating better performance in terms of bandwidth utilization.
- **Scalability:**
  - The Hub Controller does not scale well with network size, as flooding traffic across larger networks leads to higher congestion.

2 Part 2: Spanning Tree

2.1 Approach:

- **Topology Discovery:** The controller gathers switch and link information using LLDP (Link Layer Discovery Protocol). This helps it understand the physical network topology.
- **Merging Hosts and Switches:** This step merges the host-facing ports and spanning tree ports for each switch. After this step, the controller knows which ports to use for forwarding broadcast packets (host-facing ports + spanning tree ports) to avoid loops.
- **Spanning Tree Construction:** A spanning tree is built using a depth-first search (DFS) algorithm over the discovered topology. This tree ensures that loops in the network are avoided.
- **Broadcast Handling:** For broadcast packets, only ports part of the spanning tree are used for forwarding, preventing loops that would otherwise occur in a cyclic topology.
- **Flow Rule Management:** The controller installs flow rules to direct traffic based on learned MAC addresses, optimizing packet forwarding while ensuring broadcast packets are confined to non-looping paths.

2.2 Assumptions:

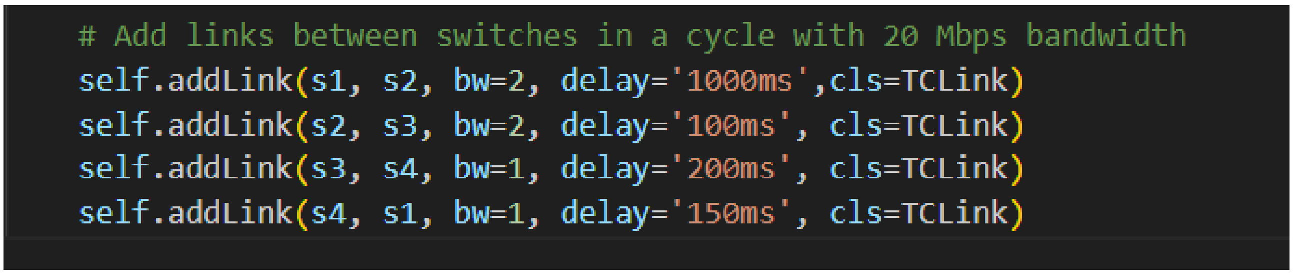
- **Switch and Link Discovery:** We have assumed that enough time has elapsed while calling the get switch and get link functions in the get topology function and that all the switches and link info have reached through the LLDP packets.
- **Spanning Tree Calculation:** The controller assumes that all link weights are equal (weight = 1) and constructs a spanning tree using this assumption. The calculate spanning tree method is a DFS-based algorithm that ensures all switches are part of the tree, while cycles are effectively broken by limiting forwarding to tree ports.
- **Network Topology:** The controller expects to work with any given network topology, as it dynamically discovers switches and links. However, the current implementation assumes the

network topology is stable once the spanning tree is constructed. Any changes in topology (e.g., link failures or additions) are not automatically handled and may require recalculating the spanning tree.

3 Part 3: Shortest Path Routing

3.1 Approach:

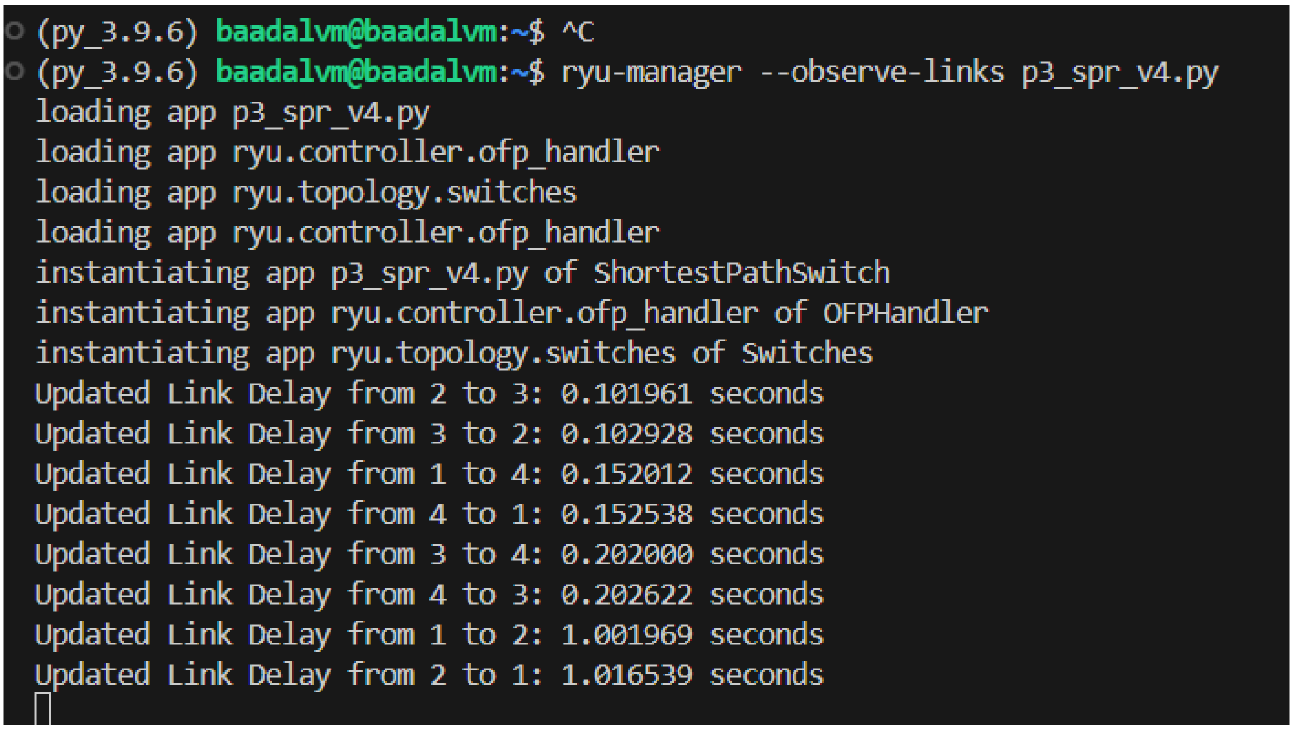
- **Spanning Tree Protocol:** This is the most crucial key in solving this part of the assignment, as it's again used for handling the broadcast message and avoiding looping in the network.



```
    # Add links between switches in a cycle with 20 Mbps bandwidth
    self.addLink(s1, s2, bw=2, delay='1000ms',cls=TCLink)
    self.addLink(s2, s3, bw=2, delay='100ms', cls=TCLink)
    self.addLink(s3, s4, bw=1, delay='200ms', cls=TCLink)
    self.addLink(s4, s1, bw=1, delay='150ms', cls=TCLink)
```

Figure 7: Actual Delay as per topology

- **Calculating Link Delays:** Link Delays are calculated using custom LLDP packets. These custom LLDP packets have unique identifier keywords and hence are segregated from the generic LLDP packets received in the packet in handler function.



```
(py_3.9.6) baadalvm@baadalvm:~$ ^C
(py_3.9.6) baadalvm@baadalvm:~$ ryu-manager --observe-links p3_spr_v4.py
loading app p3_spr_v4.py
loading app ryu.controller.ofp_handler
loading app ryu.topology.switches
loading app ryu.controller.ofp_handler
instantiating app p3_spr_v4.py of ShortestPathSwitch
instantiating app ryu.controller.ofp_handler of OFPHandler
instantiating app ryu.topology.switches of Switches
Updated Link Delay from 2 to 3: 0.101961 seconds
Updated Link Delay from 3 to 2: 0.102928 seconds
Updated Link Delay from 1 to 4: 0.152012 seconds
Updated Link Delay from 4 to 1: 0.152538 seconds
Updated Link Delay from 3 to 4: 0.202000 seconds
Updated Link Delay from 4 to 3: 0.202622 seconds
Updated Link Delay from 1 to 2: 1.001969 seconds
Updated Link Delay from 2 to 1: 1.016539 seconds
```

Figure 8: Calculated Delay with Custom LLDP Packets

- **Calculating Shortest Paths:** When all the link delays are obtained, a dictionary is maintained, which for each source switch and destination switch pair store paths (port no. and switches).
- **Nearest Switch to host**: For efficiently using routing algorithms which has no information about hosts, a dictionary for host-nearest switch pair is maintained.

3.2 Assumptions:

- All the assumptions are same as in 2nd Part.