COL334 Assignment 3

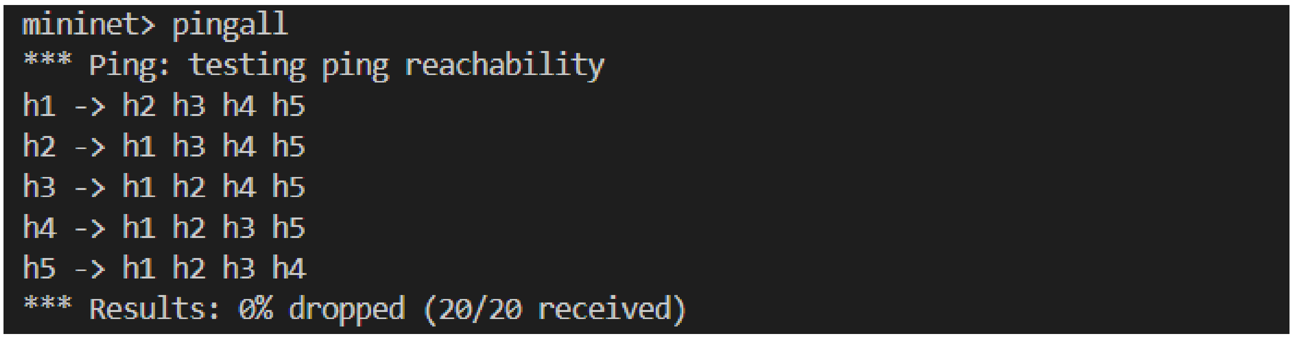
Aniket Abhiraj 2021EE10676

Sarthak kumar singh 2021EE10673

§1 Part 1

Ping Reachability

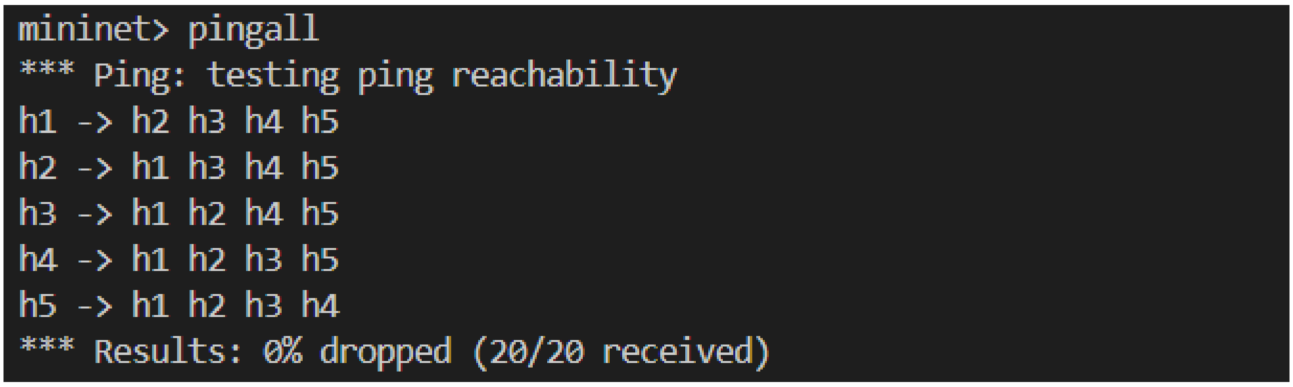
- **Hub Controller**: 0% packet loss, confirming all hosts could reach each other.



```
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 1: pingall: Hub Controller

- **Learning Switch**: 0% packet loss, confirming reachability between all hosts.

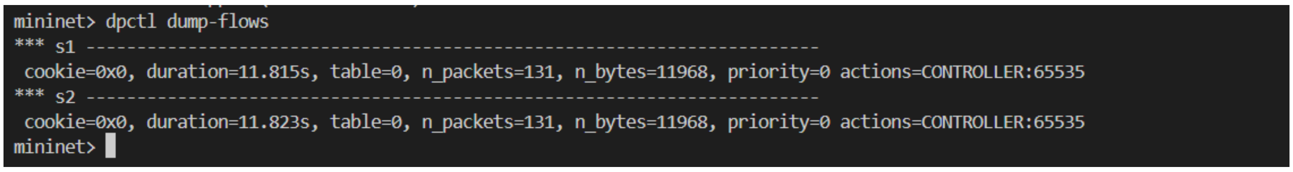


```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5
h2 -> h1 h3 h4 h5
h3 -> h1 h2 h4 h5
h4 -> h1 h2 h3 h5
h5 -> h1 h2 h3 h4
*** Results: 0% dropped (20/20 received)
```

Figure 2: pingall: Learning Switch

Flow Table Analysis

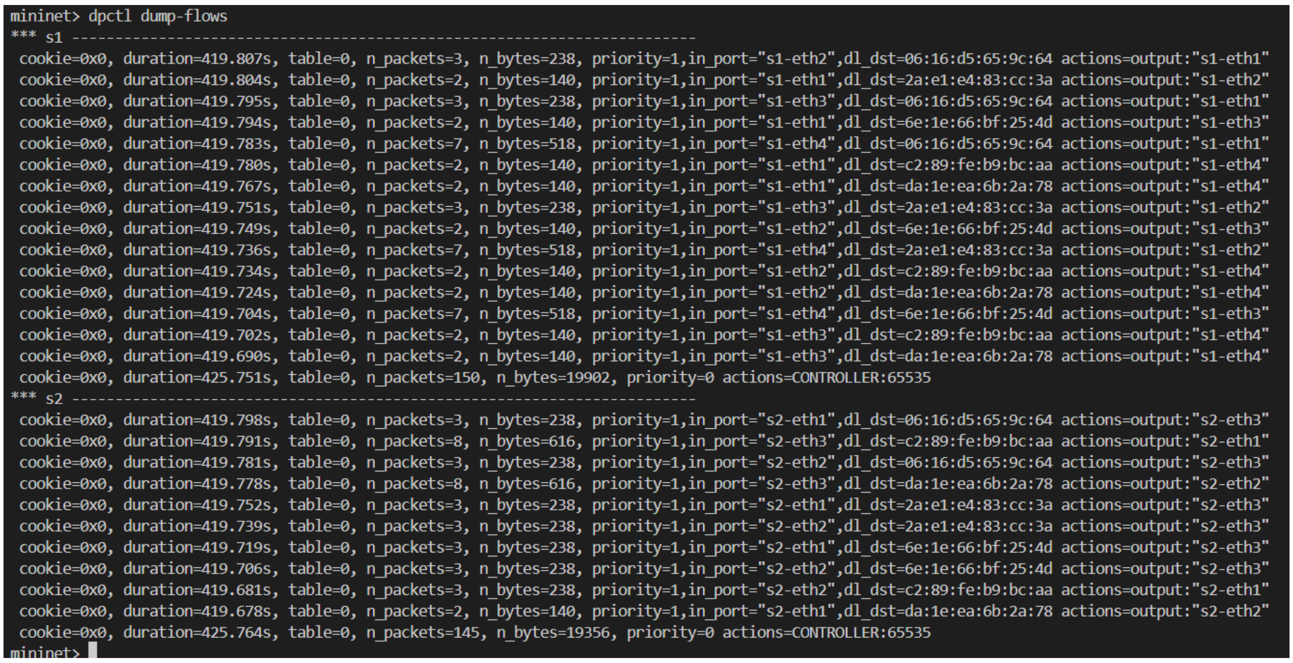
- **Hub Controller**: Flow tables only had a default rule, forwarding all packets to the controller (`priority=0 actions=CONTROLLER:65535`). This is expected behavior for a hub, as it does not perform any intelligent packet forwarding.



```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=11.815s, table=0, n_packets=131, n_bytes=11968, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=11.823s, table=0, n_packets=131, n_bytes=11968, priority=0 actions=CONTROLLER:65535
mininet> 
```

Figure 3: Flows: Hub Controller

- **Learning Switch**:
  - **Dynamic MAC Learning (Priority 1)**: The switch learns MAC addresses and installs flow rules to forward packets directly to the appropriate port, reducing the need for controller involvement and optimizing traffic.
  - **Default Controller Forwarding (Priority 0)**: A basic flow rule with priority 0 ensures unmatched packets are sent to the controller for MAC learning and new rule installation.

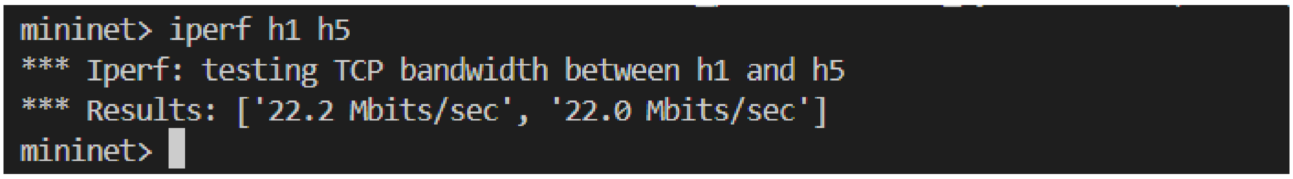


```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
 cookie=0x0, duration=419.807s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth2",dl_dst=06:16:d5:65:9c:64 actions=output:"s1-eth1"
 cookie=0x0, duration=419.804s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_dst=2a:e1:e4:83:cc:3a actions=output:"s1-eth2"
 cookie=0x0, duration=419.795s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_dst=06:16:d5:65:9c:64 actions=output:"s1-eth1"
 cookie=0x0, duration=419.794s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_dst=6e:1e:66:bf:25:4d actions=output:"s1-eth3"
 cookie=0x0, duration=419.783s, table=0, n_packets=7, n_bytes=518, priority=1,in_port="s1-eth4",dl_dst=06:16:d5:65:9c:64 actions=output:"s1-eth1"
 cookie=0x0, duration=419.780s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_dst=c2:89:fe:b9:bc:aa actions=output:"s1-eth4"
 cookie=0x0, duration=419.767s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth1",dl_dst=da:1e:ea:6b:2a:78 actions=output:"s1-eth4"
 cookie=0x0, duration=419.751s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_dst=2a:e1:e4:83:cc:3a actions=output:"s1-eth2"
 cookie=0x0, duration=419.749s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_dst=6e:1e:66:bf:25:4d actions=output:"s1-eth3"
 cookie=0x0, duration=419.736s, table=0, n_packets=7, n_bytes=518, priority=1,in_port="s1-eth4",dl_dst=2a:e1:e4:83:cc:3a actions=output:"s1-eth2"
 cookie=0x0, duration=419.734s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_dst=c2:89:fe:b9:bc:aa actions=output:"s1-eth4"
 cookie=0x0, duration=419.724s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth2",dl_dst=da:1e:ea:6b:2a:78 actions=output:"s1-eth4"
 cookie=0x0, duration=419.704s, table=0, n_packets=7, n_bytes=518, priority=1,in_port="s1-eth4",dl_dst=6e:1e:66:bf:25:4d actions=output:"s1-eth3"
 cookie=0x0, duration=419.702s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth3",dl_dst=c2:89:fe:b9:bc:aa actions=output:"s1-eth4"
 cookie=0x0, duration=419.690s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s1-eth3",dl_dst=da:1e:ea:6b:2a:78 actions=output:"s1-eth4"
 cookie=0x0, duration=425.751s, table=0, n_packets=150, n_bytes=19902, priority=0 actions=CONTROLLER:65535
*** s2 ------------------------------------------------------------------------
 cookie=0x0, duration=419.798s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth1",dl_dst=06:16:d5:65:9c:64 actions=output:"s2-eth3"
 cookie=0x0, duration=419.791s, table=0, n_packets=8, n_bytes=616, priority=1,in_port="s2-eth3",dl_dst=c2:89:fe:b9:bc:aa actions=output:"s2-eth1"
 cookie=0x0, duration=419.781s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_dst=06:16:d5:65:9c:64 actions=output:"s2-eth3"
 cookie=0x0, duration=419.778s, table=0, n_packets=8, n_bytes=616, priority=1,in_port="s2-eth3",dl_dst=da:1e:ea:6b:2a:78 actions=output:"s2-eth2"
 cookie=0x0, duration=419.752s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth1",dl_dst=2a:e1:e4:83:cc:3a actions=output:"s2-eth3"
 cookie=0x0, duration=419.739s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_dst=2a:e1:e4:83:cc:3a actions=output:"s2-eth3"
 cookie=0x0, duration=419.719s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth1",dl_dst=6e:1e:66:bf:25:4d actions=output:"s2-eth3"
 cookie=0x0, duration=419.706s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_dst=6e:1e:66:bf:25:4d actions=output:"s2-eth3"
 cookie=0x0, duration=419.681s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s2-eth2",dl_dst=c2:89:fe:b9:bc:aa actions=output:"s2-eth1"
 cookie=0x0, duration=419.678s, table=0, n_packets=2, n_bytes=140, priority=1,in_port="s2-eth1",dl_dst=da:1e:ea:6b:2a:78 actions=output:"s2-eth2"
 cookie=0x0, duration=425.764s, table=0, n_packets=145, n_bytes=19356, priority=0 actions=CONTROLLER:65535
mininet>
```

Figure 4: Flows: Learning Switch

Throughput (Iperf Results)

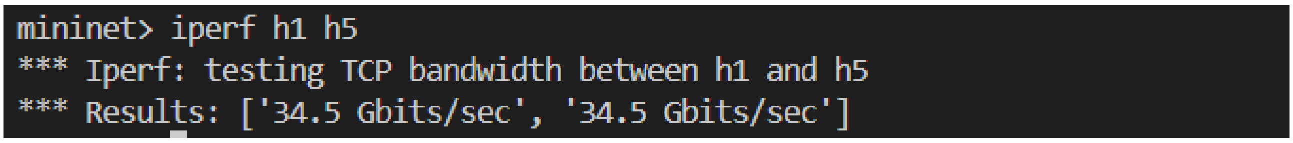
- **Hub Controller**: TCP bandwidth between `h1` and `h5` was **22.1 Mbits/sec**.



```
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['22.2 Mbits/sec', '22.0 Mbits/sec']
mininet>
```

Figure 5: iperf: Hub Controller

- **Learning Switch**: TCP bandwidth improved significantly to **34.5 Gbits/sec**. The Learning Switch optimizes bandwidth by reducing unnecessary traffic to the controller, unlike the Hub Controller which floods packets.



```
mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['34.5 Gbits/sec', '34.5 Gbits/sec']
```

Figure 6: iperf: Learning Switch

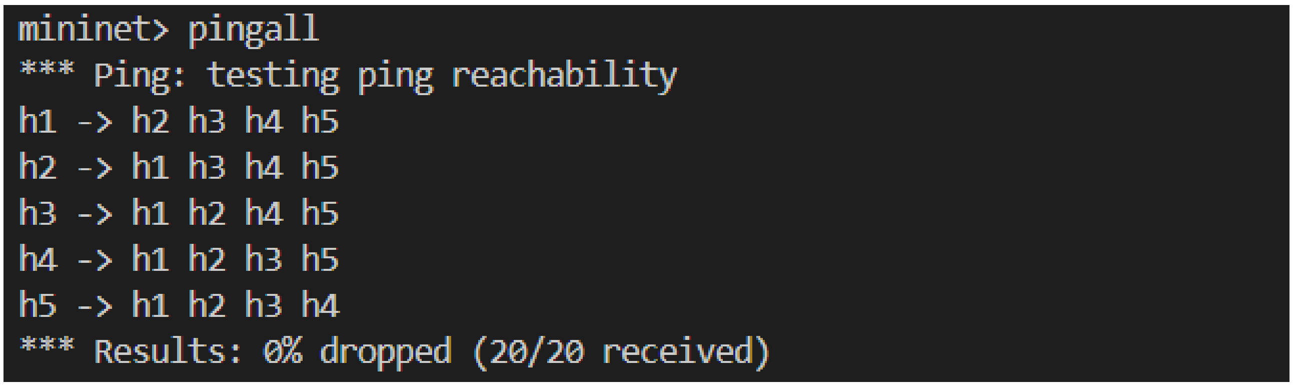
Conclusion

The **Hub Controller** acts as a simple forwarder, resulting in lower performance due to the flooding of traffic and constant involvement of the controller. On the other hand, the **Learning Switch** dynamically learns MAC addresses and sets up efficient flow rules, leading to better bandwidth and overall network efficiency.

§2 Part 2

Approach

- **Topology Discovery**:
  - Captures switches and links using Ryu's topology API (`get_all_switch`, `get_all_link`).
  - Updates the network graph whenever a new link is added.
- **Spanning Tree Calculation**:
  - Each time a link is added, the spanning tree is recalculated using Kruskal's algorithm.
  - The algorithm ensures that loops are avoided by maintaining the minimum spanning tree.
- **Port State Management**:
  - Ports not part of the spanning tree are disabled.
  - Only ports corresponding to links in the spanning tree are marked as "working."
- **Packet Handling**:
  - Packets arriving at "disabled" ports are dropped.
  - Packets are only routed through the "working" ports to avoid looping.
  - If a port is in a "working" state, MAC learning is performed, and packets are forwarded accordingly.
- **Flow Installation**:
  - Flow rules are installed for known destinations, preventing further controller intervention for those flows.
  - If the destination is unknown, the packet is flooded, but only through "working" ports.



```
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 7: pingall: Spanning Tree

Assumptions

- All link weights are equal.
- The topology changes are processed sequentially.

§3 Part 3

Approach

1. Spanning Tree Construction (Kruskal's Algorithm)

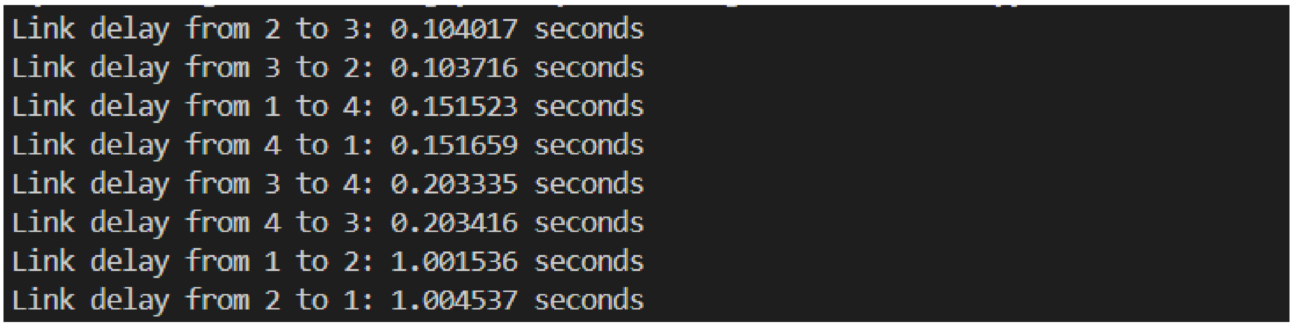
We use the Spanning Tree Protocol (STP) constructed in Part 2 for multicast/broadcast packet forwarding to avoid looping within the network. By only allowing packets to be forwarded through the ports that are part of the spanning tree, we prevent broadcast storms and ensure that no loops occur, leading to a more efficient multicast packet distribution across the network.

2. LLDP for Link Delay Measurement

LLDP Packets: To measure link delays, custom LLDP (Link Layer Discovery Protocol) packets are used. These packets contain a unique identifier (in the form of a counter value) with each transmission. When a new link is detected, an LLDP packet is sent between the two switches, and the time is measured to calculate the delay.

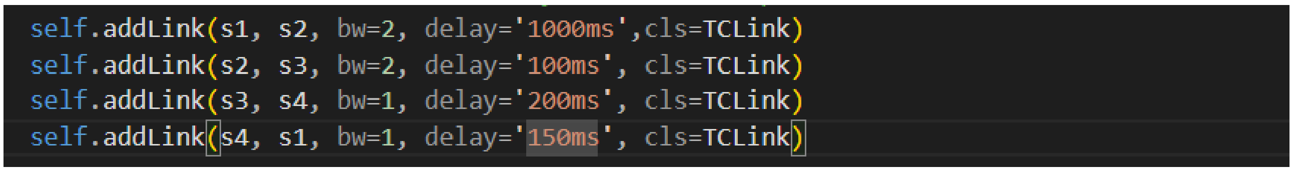
Delay Calculation: The LLDP packet includes a custom Type-Length-Value (TLV) field, and upon receiving the packet, the delay is calculated and stored as:

$$\text{Delay} = \text{End Time} - \text{Start Time}$$



```
Link delay from 2 to 3: 0.104017 seconds
Link delay from 3 to 2: 0.103716 seconds
Link delay from 1 to 4: 0.151523 seconds
Link delay from 4 to 1: 0.151659 seconds
Link delay from 3 to 4: 0.203335 seconds
Link delay from 4 to 3: 0.203416 seconds
Link delay from 1 to 2: 1.001536 seconds
Link delay from 2 to 1: 1.004537 seconds
```

Figure 8: Measured Link Delay



```
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 9: Actual Link Delay

3. Topology Graph with Weights

Once all the link delays are measured, the network graph is updated with the delay values as weights. These weights are later used to compute the shortest path between nodes, optimizing the packet routing based on real-time link conditions.

4. Routing Table and Shortest Path Algorithm

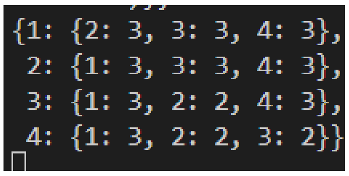
To calculate the optimal routes, we implemented Dijkstra's algorithm, which computes the shortest path from each switch to every other switch based on link delays as edge weights. The routing table generated by this algorithm maps each destination switch to the corresponding outgoing port on the current switch.

Additionally, we maintain a mapping of MAC addresses to destination switch IDs (DPIDs) in the routing table. This mapping allows the system to efficiently route packets to their intended destinations based on the destination MAC address. The routing table is structured as follows:

$$\text{Routing Table}[i][j] = \text{Port from switch } i \text{ to switch } j$$

where:

- **DPID** represents the unique identifier of a switch.
- **MAC Address** maps to the corresponding DPID for each destination.



```
{1: {2: 3, 3: 3, 4: 3},
 2: {1: 3, 3: 3, 4: 3},
 3: {1: 3, 2: 2, 4: 3},
 4: {1: 3, 2: 2, 3: 2}}
```

Figure 10: Routing Table

Assumptions

- LLDP packets are assumed to travel without interference, and the measured delays are reflective of the actual link performance.
- Switches are assumed to respond accurately to LLDP packets, and there is no loss of LLDP packets during transmission.
- Delay calculations assume minimal processing overhead at the switches.

Conclusion

This implementation ensures efficient routing through the network by dynamically adjusting to changes in the topology. The system uses real-time link delay measurements and computes the shortest paths, ensuring that packets are forwarded through the network with minimal delay. The use of Kruskal's algorithm for spanning tree construction and Dijkstra's algorithm for optimal path calculation provides a robust solution for network traffic management.