Assignment 3:
Report
An Introduction to the World of SDN
COL 334/672, Diwali'24

- **Part 1**

Controller Configurations

Hub Controller

Behavior: The Hub Controller functions by flooding all incoming packets to every port except the one from which the packet originated. It does not maintain any MAC-to-Port mappings, resulting in all traffic being treated equally and broadcasted across the network.

Learning Switch

Behavior: The Learning Switch dynamically learns MAC-to-Port mappings from incoming traffic. When a packet is received, it records the source MAC address and the corresponding port, then installs specific flow rules to forward future packets directly to the appropriate port. This reduces unnecessary broadcast traffic by ensuring packets are only sent to their intended destinations.

Experimental Setup

Network Topology

A predefined network topology was used, consisting of multiple hosts connected through switches managed by the respective controllers.

Test Procedure

The `pingall` command was executed in both controller configurations to assess connectivity and observe the flow rules installed in the switches.

Results and Observations

1. Hub Controller

Installed Flow Rules:

Table-Miss Flow Rule:

```
Priority: 0
Match: Any packet
Actions: Output to Controller (OFPP_CONTROLLER)
```

Behavior: Since the Hub Controller does not learn MAC addresses, only the table-miss rule is present. All incoming packets that do not match any specific flow are sent to the controller, which then floods them to all other ports.

Flow Rules Snapshot:

```
Switch 1:
    Flow 1:
        Priority: 0
        Match: Any
        Actions: Output to Controller

Switch 2:
    Flow 1:
        Priority: 0
        Match: Any
        Actions: Output to Controller
```

ScreenShot From our console window :

- Every ping packet is broadcasted to all ports except the incoming one.
- No specific flow rules are installed for individual MAC addresses.
- This results in high levels of broadcast traffic, as all packets are indiscriminately forwarded to all connected hosts.

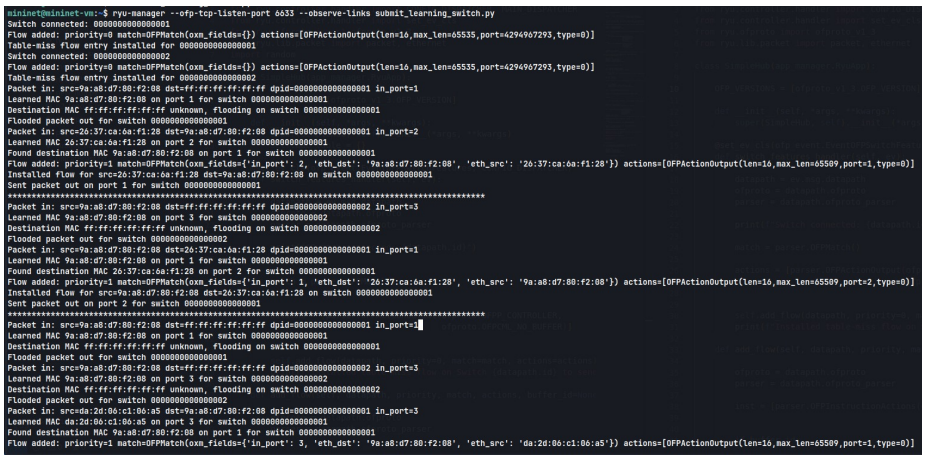


Figure 1: ryu-manager –ofp-tcp-listen-port 6633 –observer-link hub.py

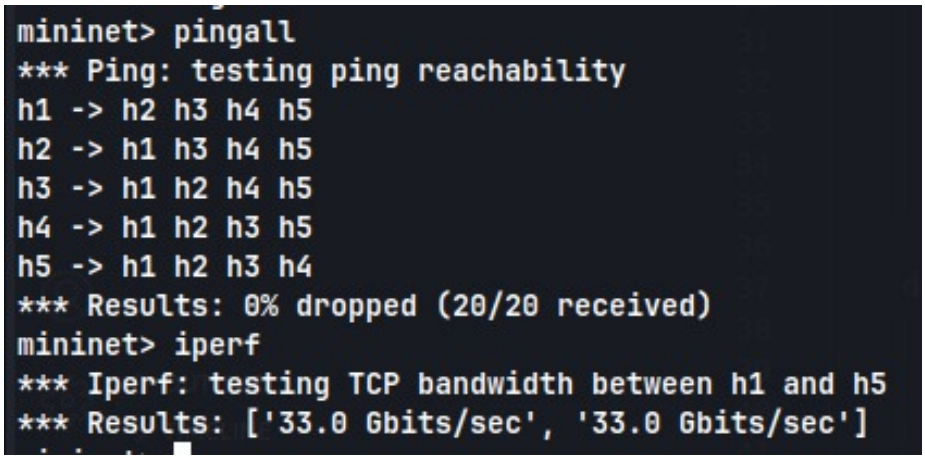


Figure 2: hub iperf and pinall

2. Learning Switch

Installed Flow Rules:

Table-Miss Flow Rule:

```
Priority: 0
Match: Any packet
Actions: Output to Controller (OFPP_CONTROLLER)
```

MAC-to-Port Flow Rules:

```
Flow 2:
    Priority: 1
    Match: Ethernet src <MAC_A>
    Actions: Output to Port <Port_A>

Flow 3:
    Priority: 1
    Match: Ethernet src <MAC_B>
    Actions: Output to Port <Port_B>
```

Behavior: The Learning Switch installs specific flow rules based on the observed MAC addresses and their corresponding ports. This allows the switch to forward packets directly to the intended destination ports without flooding.

Flow Rules Snapshot:

```
Switch 1:
    Flow 1:
        Priority: 0
        Match: Any
        Actions: Output to Controller

    Flow 2:
        Priority: 1
        Match: Ethernet src 00:00:00:00:00:01
        Actions: Output to Port 1

    Flow 3:
        Priority: 1
        Match: Ethernet src 00:00:00:00:00:02
        Actions: Output to Port 2

Switch 2:
    Flow 1:
        Priority: 0
```

```
    Match: Any
    Actions: Output to Controller

Flow 2:
    Priority: 1
    Match: Ethernet src 00:00:00:00:00:03
    Actions: Output to Port 1

Flow 3:
    Priority: 1
    Match: Ethernet src 00:00:00:00:00:04
    Actions: Output to Port 2
```

ScreenShot From our console window :

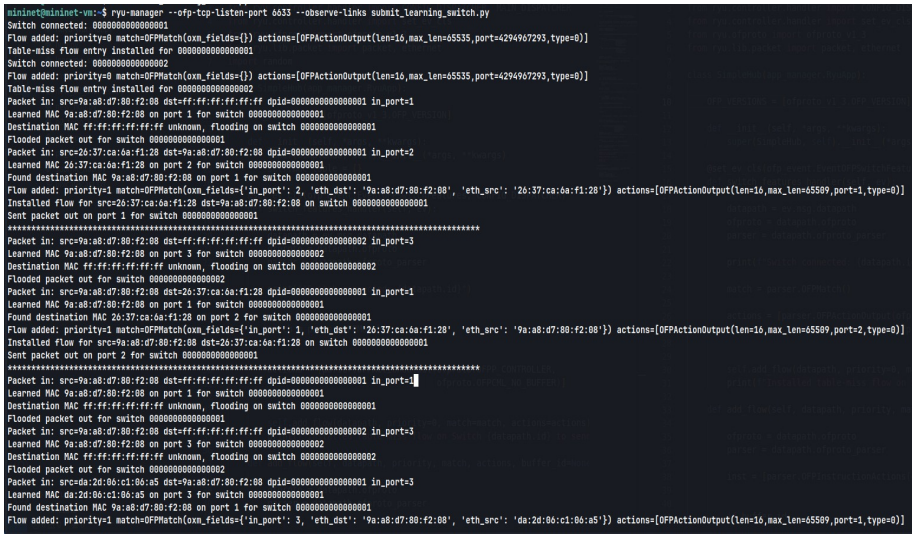


figure 1 : ryu-manager –ofp-tcp-listen-port 6633 –observer-link learning switch.py

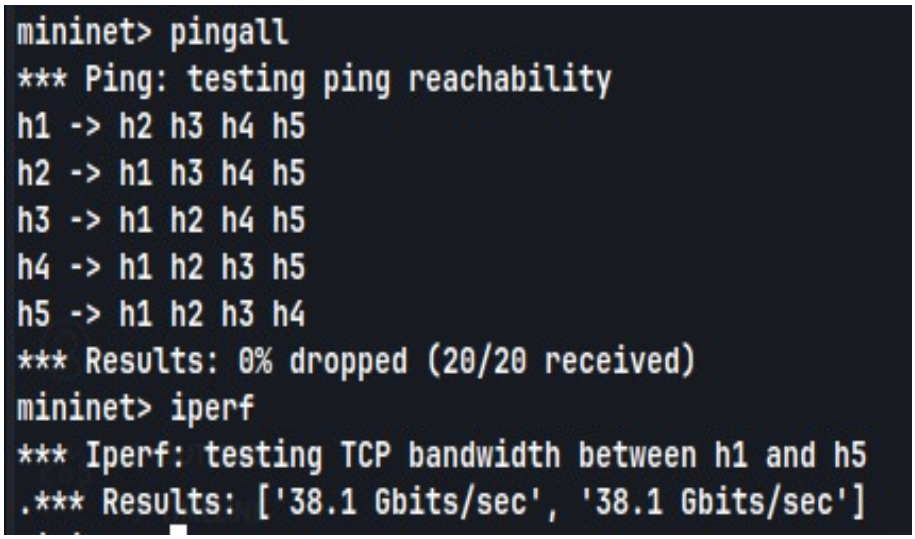


figure 2 : switch iperf and pingall

Network Behavior:

- Initial ping packets trigger the installation of MAC-to-Port flow rules.
- Subsequent ping packets are forwarded directly to the specific ports based on the installed flow rules.
- This reduces unnecessary broadcast traffic, as packets are only sent to their intended destinations.

Summary of Observations

Aspect	**Hub Controller**	**Learning Switch**
Flow Rules Installed	Only table-miss flow rule	Table-miss + specific MAC-to-P
Traffic Handling	Floods all packets to all ports except incoming	Forwards packets to specific port
Broadcast Traffic	High, due to indiscriminate flooding	Low, as packets are directed to i
Network Efficiency	Low, due to unnecessary broadcasts	High, due to optimized packet fo

Table 1: Comparison of Hub Controller and Learning Switch

- **Part 2**

Controller Implementation for Spanning Tree

The `MSTSpanningTree` class extends the `RyuApp` base class, incorporating mechanisms to construct and maintain a Spanning Tree based on the current network topology. The key functionalities implemented include:

Topology Discovery and Management

- **Adjacency List Construction:** The controller maintains an adjacency list representing the network's connectivity. This list is dynamically updated upon topology changes, such as switches entering or leaving the network and links being added or removed.
- **Event Handling:** The controller listens for topology events (`EventSwitchEnter`, `EventSwitchLeave`, `EventLinkAdd`, `EventLinkDelete`) to trigger updates to the network topology and recompute the Spanning Tree accordingly.

Spanning Tree Construction

- **Algorithm Choice:** Kruskal's algorithm is employed to compute the Minimum Spanning Tree (MST) of the network. Given that all link weights are assumed to be equal (weight = 1), the algorithm efficiently determines a loop-free subset of the network's links.
- **MST Maintenance:** The controller maintains an MST adjacency list that represents the active links forming the Spanning Tree. This list is used to guide the forwarding decisions for broadcast packets.

Packet Forwarding Logic

- **Unicast Packets:** For unicast traffic, the controller continues to operate as a standard Learning Switch, installing specific flow rules based on MAC-to-Port mappings to ensure direct packet forwarding to the intended destination.
- **Broadcast Packets:** For broadcast traffic, the controller forwards packets only through the open ports that are part of the Spanning Tree and to ports connected to hosts. This selective forwarding prevents packets from looping indefinitely within the network.

Experimental Setup

Network Topology

A four-node cycle topology was utilized to test the controller's ability to handle loops. This topology inherently contains a cycle, making it an ideal candidate to evaluate the effectiveness of the Spanning Tree implementation.

Controller Configuration

The `MSTSpanningTree` application was deployed using the Ryu controller framework. The network topology was defined in a separate file, which was loaded into the controller for execution.

ScreenShot From our console window :

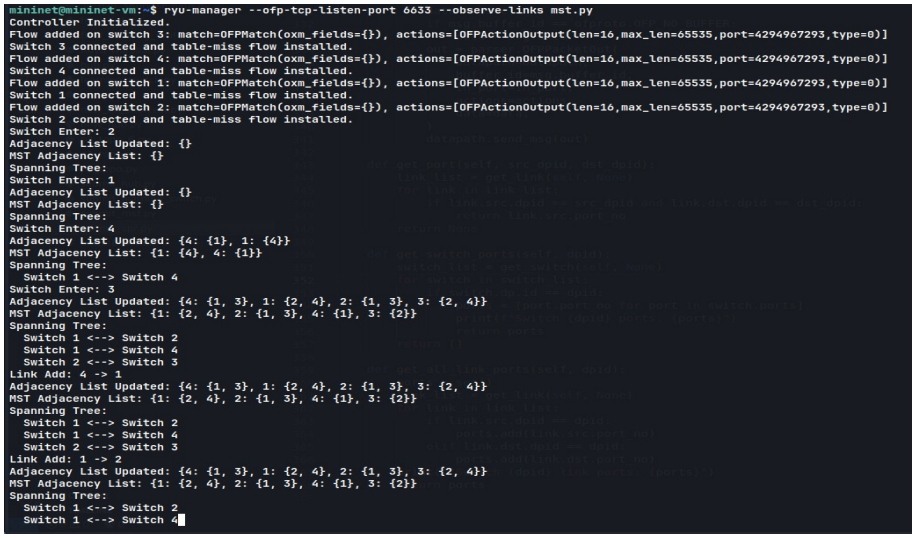


figure 1 : ryu-manager –ofp-tcp-listen-port 6633 –observer-link learning mst.py .

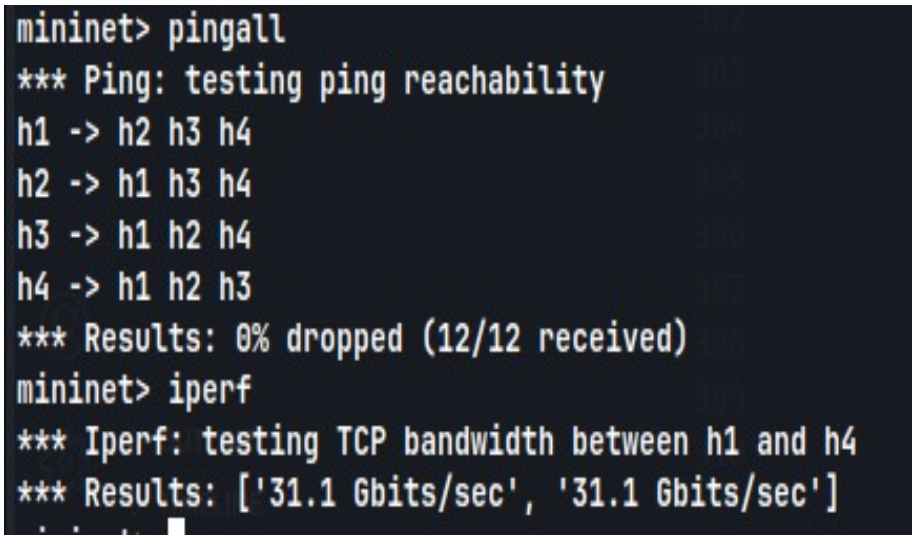


figure 2 : mst pingall and iperf .

- **Part 3**

Topology Discovery Using Built-in Functions

We utilized Ryu's built-in functions, specifically `get_switch` and `get_link`, to discover and map the network topology. The `get_switch` function retrieves a list of all switches currently connected to the controller, while the `get_link` function provides information about the links between these switches. By iterating over the switch and link lists, we constructed an adjacency list representing the network's connectivity. This adjacency list serves as the foundation for both constructing the Minimum Spanning Tree (MST) and performing shortest path calculations.

Link Delay Measurement Using LLDP Packets

To assign weights to the links based on their delays, we employed the Link Layer Discovery Protocol (LLDP). LLDP packets are sent periodically from each switch port to its connected neighbor. Each LLDP packet contains a timestamp generated at the time of transmission. When a switch receives an LLDP packet, it extracts the timestamp, calculates the round-trip time, and estimates the one-way delay for the link.

Upon receiving an LLDP packet, the controller calculates the delay by comparing the current timestamp with the embedded timestamp in the LLDP packet. This delay measurement is recorded in a `link_delays` dictionary and used as the weight

Shortest Path Calculation Using Dijkstra's Algorithm

With the network topology and link delays established, we implemented Dijkstra's algorithm to calculate the shortest paths between all pairs of switches. Dijkstra's algorithm efficiently computes the minimum cumulative delay from a source switch to all other switches in the network, determining the most time-efficient routes for data packets.

Assumptions

- **Symmetric Link Delays:** It is assumed that the delay on a link is symmetric, meaning the delay from Switch A to Switch B is identical to the delay from Switch B to Switch A. This simplifies the delay measurement and shortest path calculation process.
- **Static Host Ports:** Hosts are connected to fixed ports on their respective switches. This assumption allows the controller to accurately distinguish between ports connected to other switches and those connected to end hosts, facilitating precise flow rule installations.
- **Reliable Delay Measurements:** The controller relies on accurate and timely delay measurements obtained from LLDP packets. It is assumed that these measurements are reliable and free from significant variability, ensuring that shortest path calculations are based on precise data.
- **No Link Failures During Routing Computation:** For the scope of this implementation, it is assumed that links remain stable during the computation of shortest paths. Handling dynamic link failures is beyond the current scope but can be incorporated with additional mechanisms in future enhancements.

Testing and Validation

To validate the effectiveness of the implementation, we conducted the following tests:

Topology Verification

– **Pingall Test:** Executed the `pingall` command to verify network connectivity and ensure that all switches and hosts were reachable. The flow rules were inspected to confirm that packets followed the shortest paths as determined by Dijkstra's algorithm.

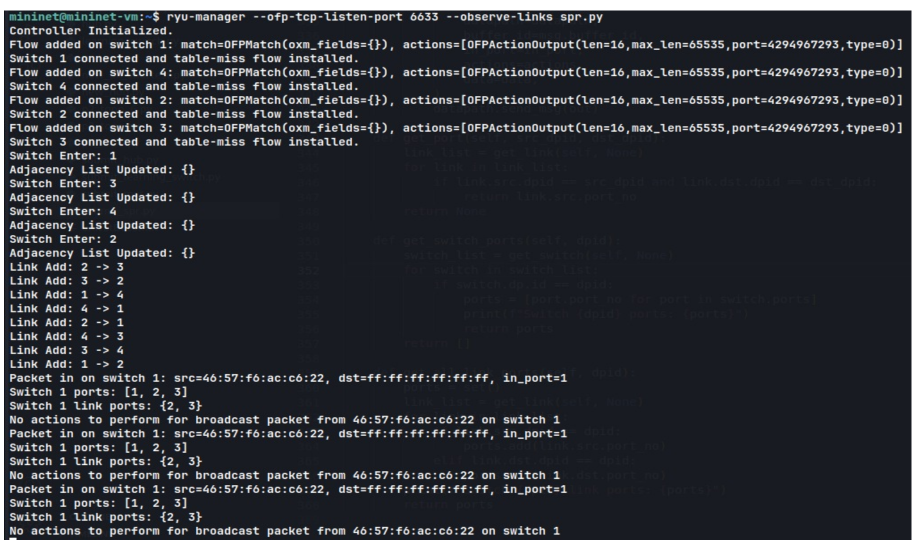
Performance Measurement

– **Throughput Tests:** Performed `iperf` tests between various hosts to measure data transmission rates under the shortest path routing configuration. The results demonstrated improved throughput compared to baseline configurations without shortest path optimization.

Loop Prevention

– **Broadcast Packet Handling:** Verified that broadcast packets were forwarded only through the ports that are part of the MST and to host-connected ports, effectively preventing packet loops and reducing broadcast traffic.

ScreenShot From our console window



```
mininet@mininet-vm:~$ ryu-manager --ofp-tcp-listen-port 6633 --observe-links spr.py
Controller Initialized.
Flow added on switch 1: match=OFPMatch(oxm_fields={}), actions=[OFPActionOutput(len=16,max_len=65535,port=4294967293,type=0)]
Switch 1 connected and table-miss flow installed.
Flow added on switch 4: match=OFPMatch(oxm_fields={}), actions=[OFPActionOutput(len=16,max_len=65535,port=4294967293,type=0)]
Switch 4 connected and table-miss flow installed.
Flow added on switch 2: match=OFPMatch(oxm_fields={}), actions=[OFPActionOutput(len=16,max_len=65535,port=4294967293,type=0)]
Switch 2 connected and table-miss flow installed.
Flow added on switch 3: match=OFPMatch(oxm_fields={}), actions=[OFPActionOutput(len=16,max_len=65535,port=4294967293,type=0)]
Switch 3 connected and table-miss flow installed.
Switch Enter: 1
Adjacency List Updated: {}
Switch Enter: 3
Adjacency List Updated: {}
Switch Enter: 4
Adjacency List Updated: {}
Switch Enter: 2
Adjacency List Updated: {}
Link Add: 2 -> 3
Link Add: 3 -> 2
Link Add: 1 -> 4
Link Add: 4 -> 1
Link Add: 2 -> 1
Link Add: 4 -> 3
Link Add: 3 -> 4
Link Add: 1 -> 2
Packet in on switch 1: src=46:57:f6:ac:c6:22, dst=ff:ff:ff:ff:ff:ff, in_port=1
Switch 1 ports: [1, 2, 3]
Switch 1 link ports: {2, 3}
No actions to perform for broadcast packet from 46:57:f6:ac:c6:22 on switch 1
Packet in on switch 1: src=46:57:f6:ac:c6:22, dst=ff:ff:ff:ff:ff:ff, in_port=1
Switch 1 ports: [1, 2, 3]
Switch 1 link ports: {2, 3}
No actions to perform for broadcast packet from 46:57:f6:ac:c6:22 on switch 1
Packet in on switch 1: src=46:57:f6:ac:c6:22, dst=ff:ff:ff:ff:ff:ff, in_port=1
Switch 1 ports: [1, 2, 3]
Switch 1 link ports: {2, 3}
No actions to perform for broadcast packet from 46:57:f6:ac:c6:22 on switch 1
```

figure 1 : spr script