COL334 - Computer Networks
Assignment 3: SDN

Nikhil Zawar - 2022CS11106
Yash Bansal - 2022CS51133

Contents

1 Controller Hub and Learning Switch

1.1 Implementation Details

Hub Controller: A hub is a simple network device that receives data on one of its ports and forwards it to all other ports, without considering which device the data is meant for. The core behavior here is flooding, which means the controller tells the switch to send the packet to all ports except the one it came from.
Learning Switch Controller: The learning controller uses a mechanism to learn the MAC addresses of devices connected to the switch. It dynamically builds a mapping of which MAC addresses are reachable on which ports. This approach helps the controller to avoid unnecessary flooding.

- When a packet arrives, the controller checks whether it already knows the destination MAC address (i.e., if it's in the MAC-to-port table).
- If the destination MAC is known, it sends the packet directly to the port associated with that MAC address.
- If the destination MAC is unknown (meaning the controller hasn't seen this address before), it floods the packet to all ports, except the one from which it received the packet, similar to a hub.

1.2 Pingall and Throughput between

```
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>
```

Figure 1: Pingall for hub controller

```
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>
```

Figure 2: Pingall for learning controller

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
*** s2 ------------------------------------------------------------------------
mininet>
```

Figure 3: Flow rules for hub controller

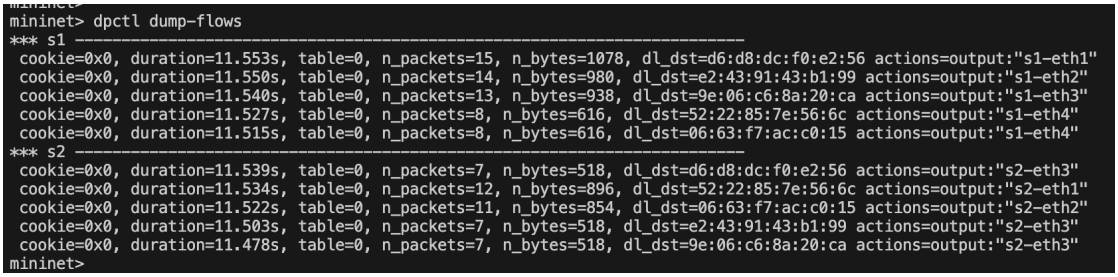

Figure 4: Flow rules for learning controller

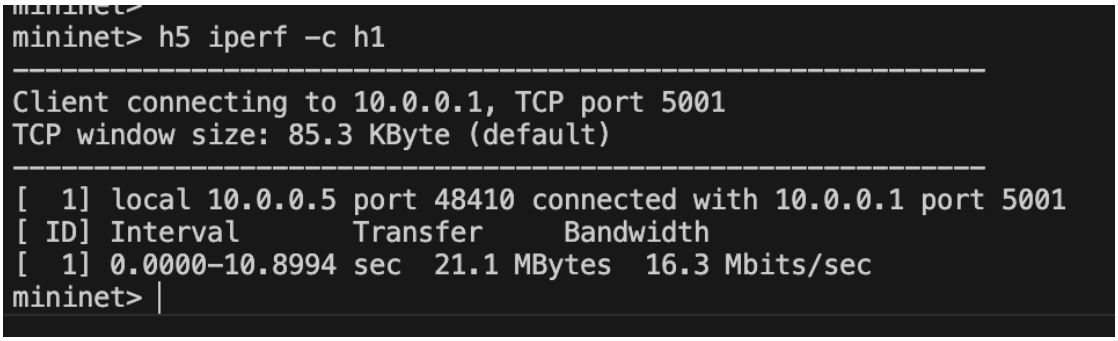

Figure 5: Throughput results for hub controller

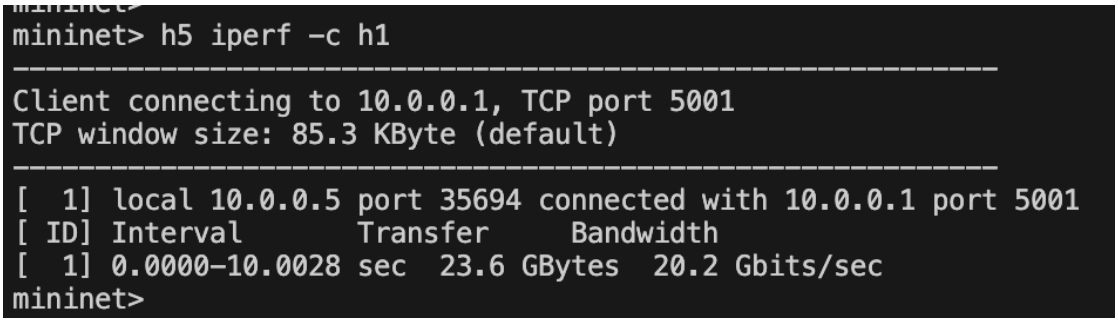

Figure 6: Throughput results for learning controller

1.3 Performance of Hub and Learning Switch

- As seen from the attached dump-flows images, the learning controller installs flow rules in the switches for immediate forwarding if the destination is already learned. In contrast, the hub controller installs no flow rules in the switches.
- As for the hub controller, every time a switch receives a packet, it has to forward it to the controller to know where the packet needs to be forwarded. In the case of a learning switch, once the destination is learned, the packet does not need to be transferred to the controller.

- Thus, the throughput of the learning switch is very high compared to that of a hub controller.
- Hub controller throughput:- 16 Mbps.
- Learning switch throughput:- 20.2 Gbps.

1.4 Use of Loopback address

Ryu and Mininet use the loopback address (127.0.0.1) by default for communication because both the controller (Ryu) and the network emulation (Mininet) typically run on the same machine during development and testing.

- Local Communication: The loopback interface is used for internal communication on the local machine. Since both Ryu and Mininet are running locally, the loopback address is an ideal way to facilitate communication without requiring a real external network.
- Network Isolation: Using the loopback interface isolates the traffic between Ryu and Mininet from the rest of the network. This is important during testing, as it ensures that any experimentation doesn't interfere with or depend on external network conditions.
- Efficiency: Communication via the loopback address is faster and more efficient than using an external IP address because it avoids the overhead of going through the machine's physical networking hardware.

2 Spanning Tree

The dictionary self.mac_to_port behaves by maintaining a MAC-to-port mapping table for each switch. When a packet arrives, if its destination MAC is known, the packet is forwarded through the associated port. If the destination is unknown, the switch floods the packet across all ports except the input port.

2.1 Implementation Details

2.1.1 Spanning Tree Construction

To prevent packets from circulating in loops, we use a spanning tree to control how packets are forwarded. The spanning tree is constructed using Depth-First Search (DFS), starting from any switch.
Once the spanning tree is established, broadcast packets (those whose destination MAC is unknown or those meant for multiple recipients) are forwarded only through the open ports in the spanning tree. Any ports not part of the spanning tree are pruned, ensuring that loops are avoided.

The network topology is represented as a collection of switches interconnected by bidirectional links. We assume the network topology is static for each run of the controller (dynamic changes such as switches leaving or joining are handled by topology update events). Broadcasts should only traverse ports that are part of the spanning tree.

2.1.2 New Data structures:

- switch_graph: Represents the topology, storing connections between switches and their corresponding ports.
- spanning_tree: Contains the spanning tree created from the switch_graph.

2.1.3 Algorithmic Design - Flow Table

Each time the spanning tree or topology changes, flow tables are reset to ensure that outdated rules are not applied. New flow rules are installed to forward packets based on the MAC address mappings and spanning tree structure.

2.2 Pingall and Throughput between

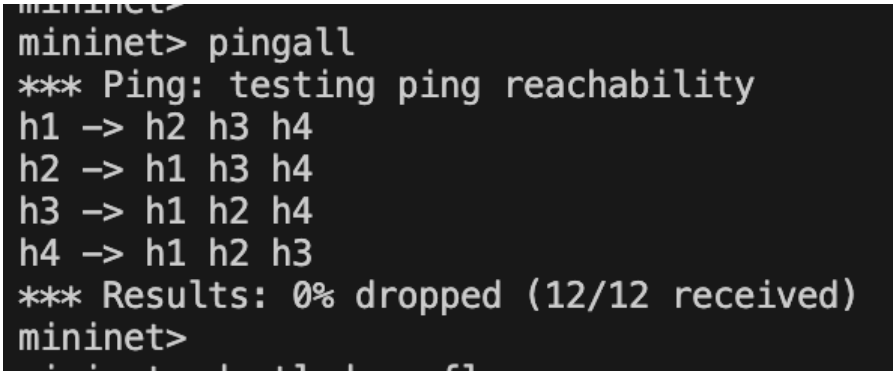

Figure 7: Pingall for spanning tree controller

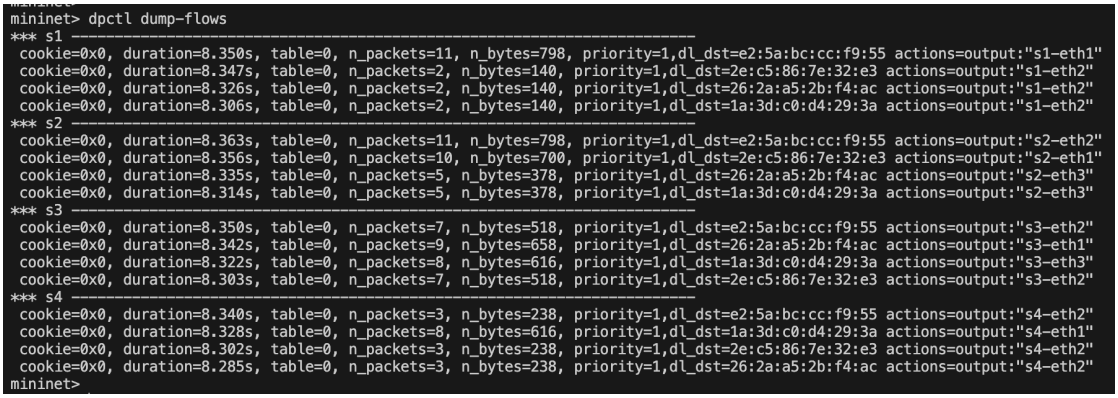

Figure 8: Flow rules for spanning tree controller

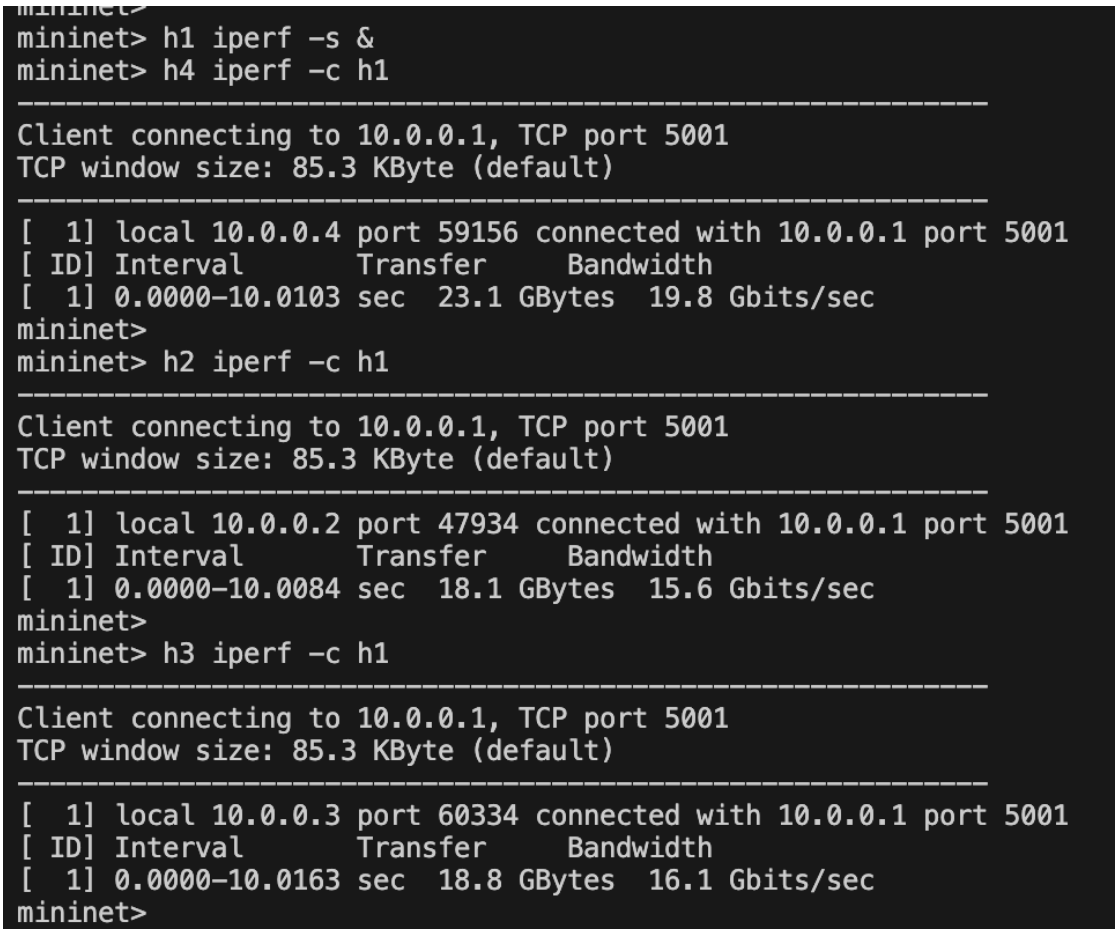

Figure 9: Throughput results for spanning tree controller

- The throughput results of spanning tree controller is similar to that of learning switch, as in this case also, the switch learns the destination port for mac address and forward the packets as soon as possible. The only difference is to forward the packets just to the spanning tree ports to prevent the packets from going to a possibly infinite cycle.
- Average throughput results:- 17.1 Gbps.

3 Shortest Path Routing

Main objective of the code - compute shortest path trees (SPTs), handle LLDP messages, measure Round-Trip Time (RTT), and manage flow rules dynamically.

3.1 Implementation Details:

3.1.1 New Data Structures

- 'link_delays_dict': Stores link delays calculated from LLDP packets.
- 'rtt_dict': Tracks Round-Trip Times for each switch.
- 'spt_tree', 'flow_rule_tree': Store Shortest Path Trees and flow rules respectively.
- 'switches_list', 'links_list': Lists of switches and links in the network.

3.1.2 LLDP Message Handling:

- 'send_lldp_packet()' & 'build_lldp_packet()': Constructs and sends LLDP packets containing organizational-specific TLVs (Time-Stamps). The LLDP packets include an organizational-specific TLV containing a timestamp. This timestamp is used to calculate link delays. Upon receiving an LLDP packet, the controller compares this timestamp with the current time to calculate the link delay.

3.1.3 RTT Measurement for switch-controller delay:

- send_rtt_packets()': Sends Echo Request packets to measure Round-Trip Times (RTT) between switches.
- '_echo_reply_handler()': Handles Echo Reply packets to calculate and store RTT values.

3.2 Algorithmic Flow

1. As soon as the topology is updated, which includes add link, delete link, add switch, delete switch, then the existing data structures are all reset to empty, only preserving the updated graph. Next step is to send LLDP packets across all links between the switches.

2. Build the LLDP packets, with storing the custom timestamp for each packet and storing it in the Organizationally specifc TLV. Then send these packets across links. When these packets are caught by the packet_in_handler function find the time it took to reach to the controller from the time when the packet was made. From this time difference, subtract the time it takes for the packet to travel between switch and controller. The final time got is the link delay for that link.

3. Once such packets are read and all the link delays are found, the next step is to compute the shortest path trees. Here note that the shortest path trees will be different for starting from each node. Compute these trees with link delays as edge weights and switches as vertex, according to the dijkstra's algorithm.

4. Once the shortest path trees are calculated, then for every packet transfer see the route that is shown by the shortest path tree to reach the destination node starting from the start node. If there is not path in the shortest path tree, only then flood the packet on all ports except for the input port.

5. Call add_flows function for every packet transfer to fill the flow tables of the switches appropriately.

3.3 Pingall and Throughput between

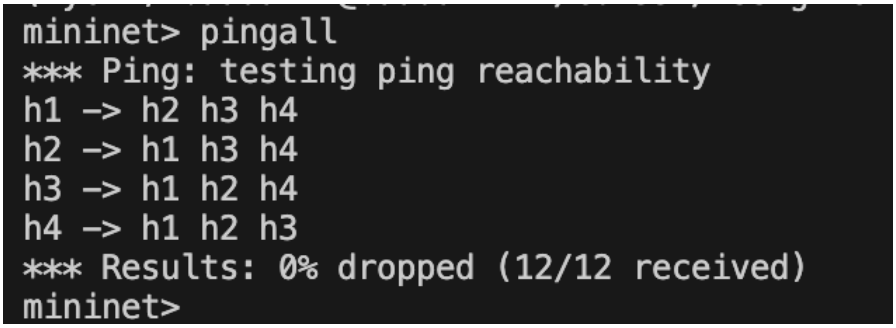

Figure 10: Pingall for shortest path routing

Figure 11: Flow rules for shortest path routing

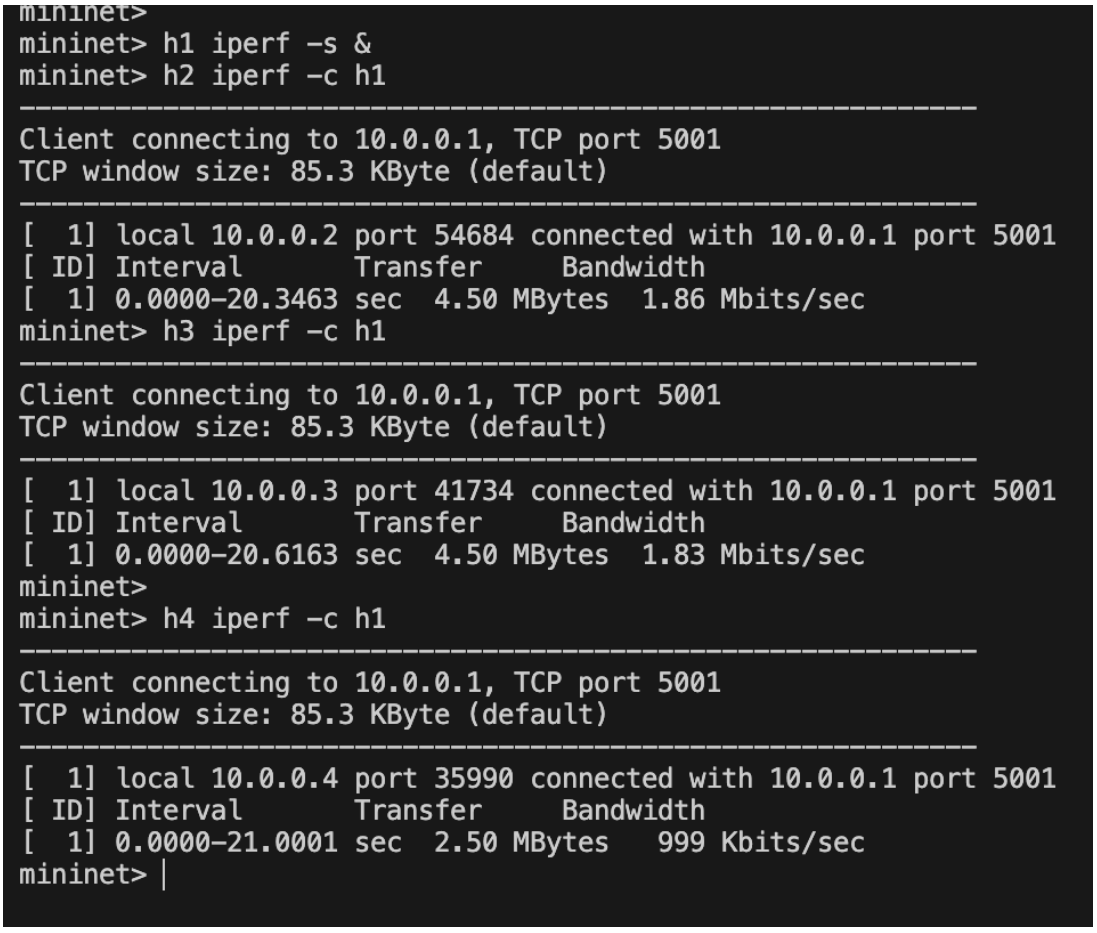

Figure 12: Throughput results for shortest path routing

4 Congestion-aware Shortest Path Routing

For congestion-aware shortest path routing, the controller would periodically send the LLDP packets every 10 seconds and compute the shortest path tree for each switch. RTT would also be calculated in each cycle.