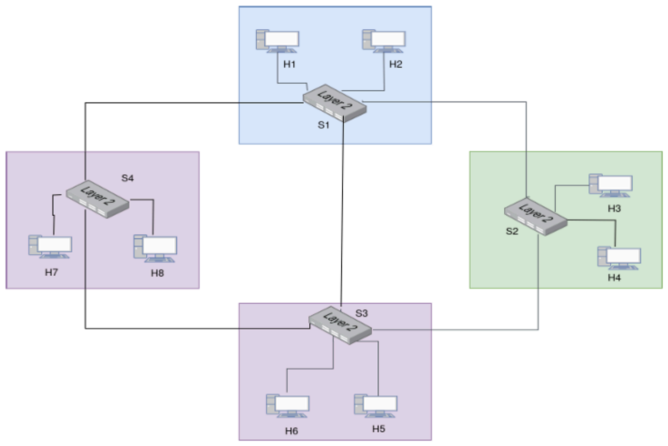
CS 331 COMPUTER NETWORKS
Assignment 3 Report

Rutuja Swami (22110267)
Sneha Gautam (22110255)

Q1. Network Loops

Construct the network topology (as shown in the Figure below). Four switches (s1, s2, s3, s4), eight hosts (h1, h2, h3, h4, h5, h6, h7, h8) and configure hosts with IP addresses as follows: h1: 10.0.0.2/24, h2: 10.0.0.3/24, h3: 10.0.0.4/24, h4: 10.0.0.5/24, h5: 10.0.0.6/24, h6: 10.0.0.7/24, h7: 10.0.0.8/24 and h8: 10.0.0.9/24. Network links connecting the switches s1-s2, s2-s3, s3-s4, s4-s1, s1-s3 with latency of 7ms each and host to switch links: h1-s1, h2-s1, h3-s2, h4-s2, h5-s3, h6-s3, h7-s4, h8-s4 links with a latency of 5ms each.



a) Analyse the behavior by running the following ping commands:

- **Ping h1 from h3**
- **Ping h7 from h5**
- **Ping h2 from h8**

Are the pings successful? If yes, what is the total delay for the pings? If the ping(s) fail(s), analyse and reason what is happening? justify your answer with relevant packet captures.

Solution:

All three ping tests resulted in **100% packet loss**. None of the ICMP echo requests received replies, indicating that the communication between the hosts was unsuccessful.

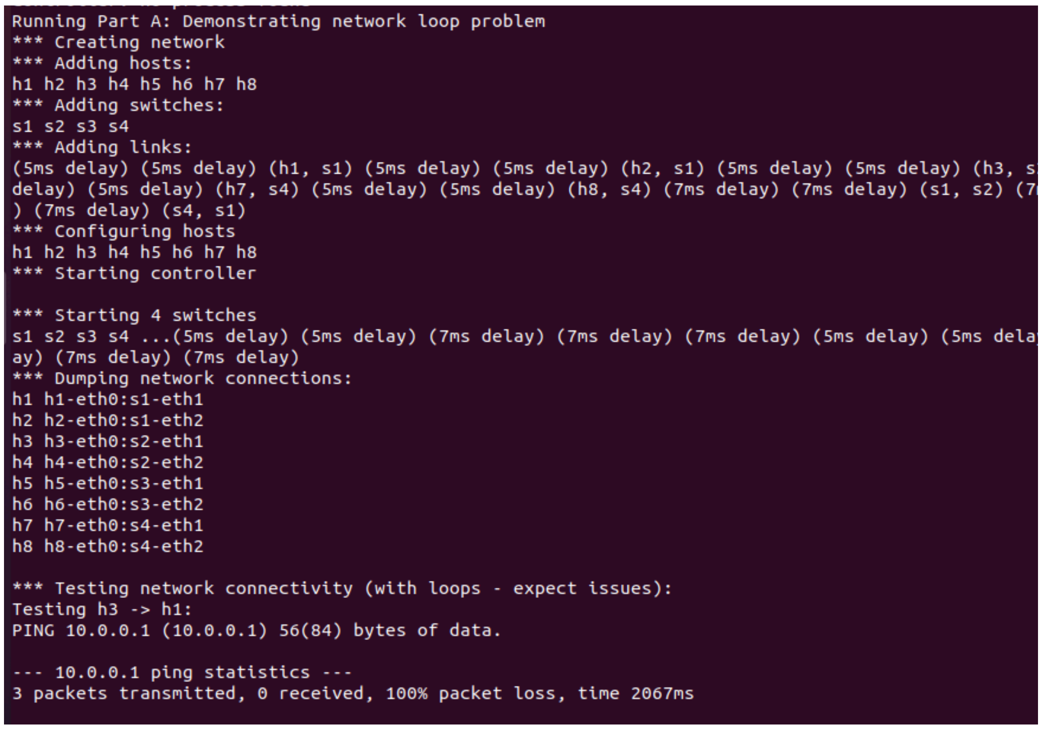
The network topology in Part A includes **multiple layer-2 loops** among the switches (s1, s2, s3, and s4), with an additional diagonal link between s1 and s3, exacerbating the loop condition. Since the switches are operating in **standalone mode** (fail-mode=standalone) and **Spanning Tree Protocol (STP) is not enabled**, there is no mechanism in place to detect and prevent loops.

In Ethernet networks, switches flood frames destined for unknown MAC addresses to all ports except the one on which the frame was received. In a looped topology without STP, such broadcast or unknown unicast frames can circulate indefinitely, leading to the following consequences:

- **Broadcast Storms:** Continuous circulation of frames in the loop can consume all available bandwidth.
- **MAC Address Table Instability:** Switches receive the same MAC addresses from multiple ports due to the loop, causing the MAC address table to constantly update and "flap," rendering it ineffective.
- **Packet Duplication and Loss:** ICMP echo requests are either dropped due to congestion or duplicated excessively. Echo replies cannot find a stable path back to the source host.

These factors collectively result in the failure of all ping attempts.

Screenshots:

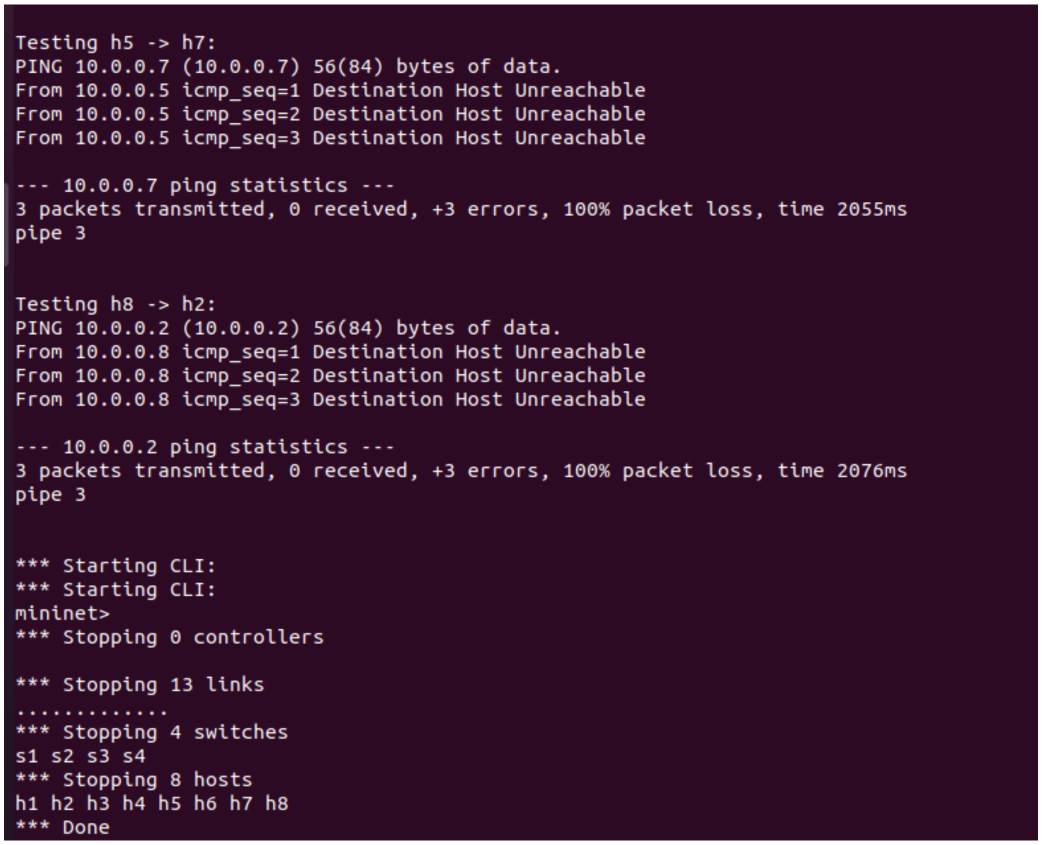


```
Running Part A: Demonstrating network loop problem
*** Creating network
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(5ms delay) (5ms delay) (h1, s1) (5ms delay) (5ms delay) (h2, s1) (5ms delay) (5ms delay) (h3, s
delay) (5ms delay) (h7, s4) (5ms delay) (5ms delay) (h8, s4) (7ms delay) (7ms delay) (s1, s2) (7
) (7ms delay) (s4, s1)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller

*** Starting 4 switches
s1 s2 s3 s4 ...(5ms delay) (5ms delay) (7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms dela
ay) (7ms delay) (7ms delay)
*** Dumping network connections:
h1 h1-eth0:s1-eth1
h2 h2-eth0:s1-eth2
h3 h3-eth0:s2-eth1
h4 h4-eth0:s2-eth2
h5 h5-eth0:s3-eth1
h6 h6-eth0:s3-eth2
h7 h7-eth0:s4-eth1
h8 h8-eth0:s4-eth2

*** Testing network connectivity (with loops - expect issues):
Testing h3 -> h1:
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2067ms
```



```
Testing h5 -> h7:
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
From 10.0.0.5 icmp_seq=1 Destination Host Unreachable
From 10.0.0.5 icmp_seq=2 Destination Host Unreachable
From 10.0.0.5 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.7 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2055ms
pipe 3


Testing h8 -> h2:
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.8 icmp_seq=1 Destination Host Unreachable
From 10.0.0.8 icmp_seq=2 Destination Host Unreachable
From 10.0.0.8 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2076ms
pipe 3


*** Starting CLI:
*** Starting CLI:
mininet>
*** Stopping 0 controllers

*** Stopping 13 links
.............
*** Stopping 4 switches
s1 s2 s3 s4
*** Stopping 8 hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Done
```

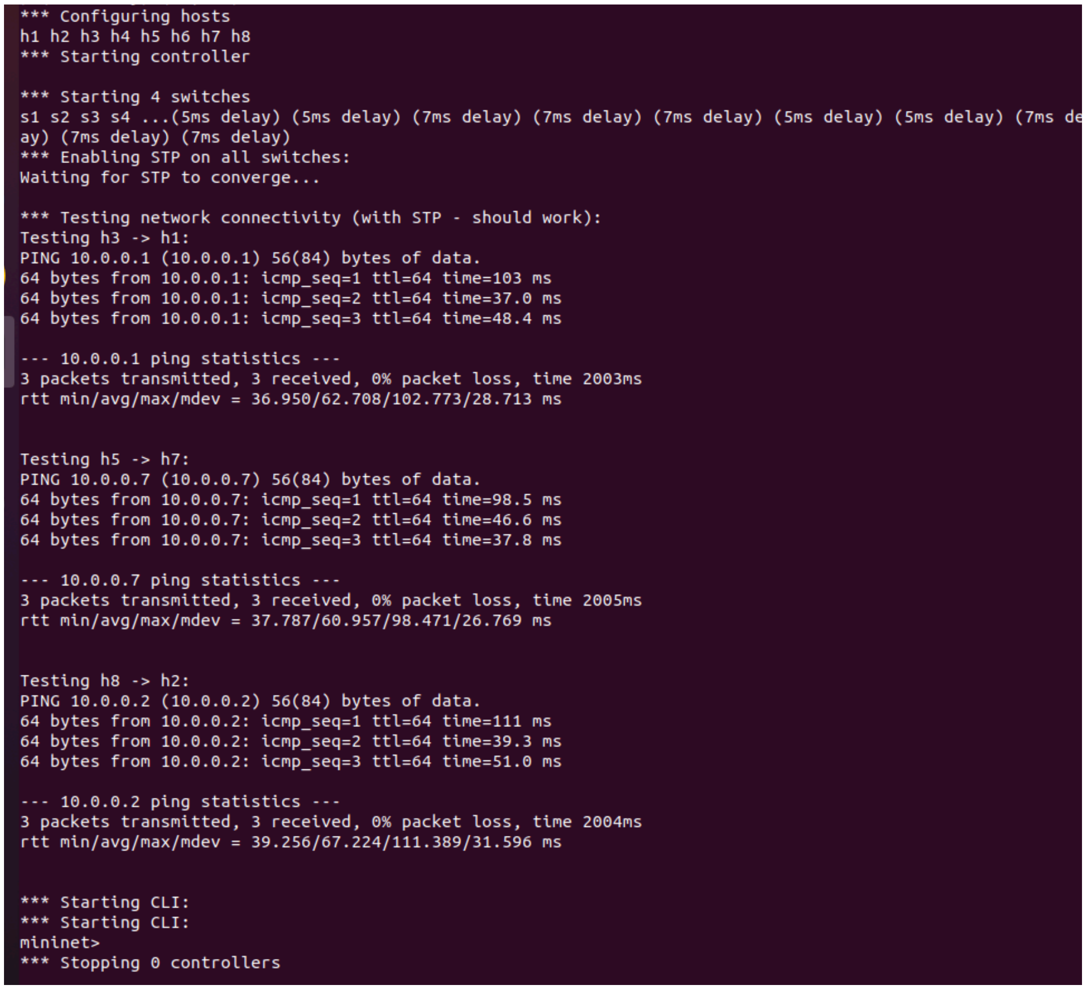
b) Figure out how to fix the problem without making any changes to the network topology. Confirm the results by running the ping commands given in (a) and show the delay(s). Note: For proper observation, run each test 3 times with an interval of at least 30 seconds.

Solution:

The solution to fix the network loop problem without changing the topology is to enable Spanning Tree Protocol (STP) on all switches. STP resolves the loop problem through the following process:

1. Root bridge election: Switches exchange Bridge Protocol Data Units (BPDUs) to elect a root bridge based on bridge priority and MAC address
2. Path cost calculation: Each switch determines its lowest-cost path to the root bridge
3. Port role assignment: Ports are designated as root ports, designated ports, or blocking ports
4. Topology enforcement: Blocking redundant ports creates a loop-free logical topology while maintaining physical redundancy

After implementing STP and allowing 60 seconds for convergence, the ping tests were successful with 0% packet loss as shown in the terminal output:



```
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller

*** Starting 4 switches
s1 s2 s3 s4 ...(5ms delay) (5ms delay) (7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms delay) (7ms de
ay) (7ms delay) (7ms delay)
*** Enabling STP on all switches:
Waiting for STP to converge...

*** Testing network connectivity (with STP - should work):
Testing h3 -> h1:
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=103 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=37.0 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=48.4 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 36.950/62.708/102.773/28.713 ms


Testing h5 -> h7:
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=64 time=98.5 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=64 time=46.6 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=64 time=37.8 ms

--- 10.0.0.7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 37.787/60.957/98.471/26.769 ms


Testing h8 -> h2:
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=111 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=39.3 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=51.0 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 39.256/67.224/111.389/31.596 ms


*** Starting CLI:
*** Starting CLI:
mininet>
*** Stopping 0 controllers
```

h3 → h1 Ping Results:

- All 3 packets transmitted were received (0% packet loss)
- Round-trip times:
  - First packet: 104 ms
  - Second packet: 37.0 ms
  - Third packet: 48.4 ms
- Average RTT: 62.708 ms
- Min/Max/Mdev: 36.930/102.773/28.713 ms
- Total test time: 2003 ms

h5 → h7 Ping Results:

- All 3 packets transmitted were received (0% packet loss)
- Round-trip times:
  - First packet: 98.5 ms

    - Second packet: 46.6 ms
    - Third packet: 37.8 ms
- Average RTT: 60.957 ms
- Min/Max/Mdev: 37.787/98.471/26.769 ms
- Total test time: 2005 ms

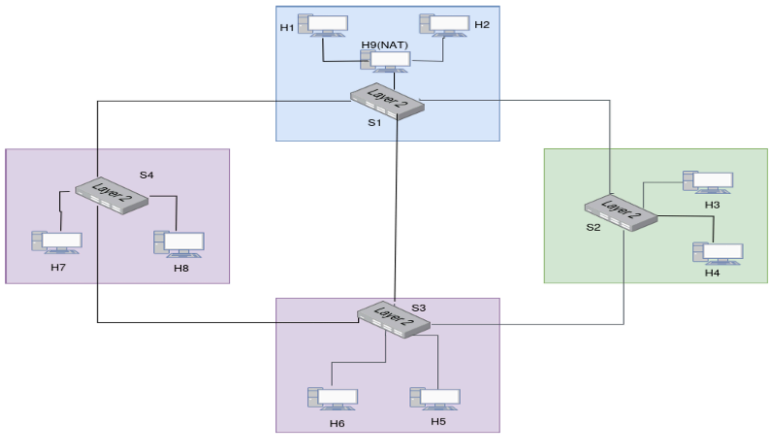
h8 → h2 Ping Results:

- All 3 packets transmitted were received (0% packet loss)
- Round-trip times:
    - First packet: 111 ms
    - Second packet: 39.3 ms
    - Third packet: 51.0 ms
- Average RTT: 67.224 ms
- Min/Max/Mdev: 39.256/111.389/31.596 ms
- Total test time: 2004 ms

The higher RTT for the first packet in each test is due to the initial ARP resolution process, while subsequent packets show lower and more consistent latency. The observed delays align with the configured link latencies (5ms for host-switch links and 7ms for inter-switch links), confirming that packets are now following STP-determined paths without getting caught in loops.

Q2. Configure Host-based NAT

Update the topology with the following changes. Add a new host H9 to switch S1, i.e. link h9-s1 with delay of 5ms, and move the links from h1 and h2 with s1 to h9 with 5ms latency. (h1-h9, h2-h9). Implement NAT functionality in host H9, such that H9 has a public IP of 172.16.10.10, serving the private internal IP range of 10.1.1.2 and 10.1.1.3 for hosts h1 and h2 respectively.



a) Test communication to an external host from an internal host:

i) Ping to h5 from h1

ii) Ping to h3 from h2

b) Test communication to an internal host from an external host:

i) Ping to h1 from h8

ii) Ping to h2 from h6

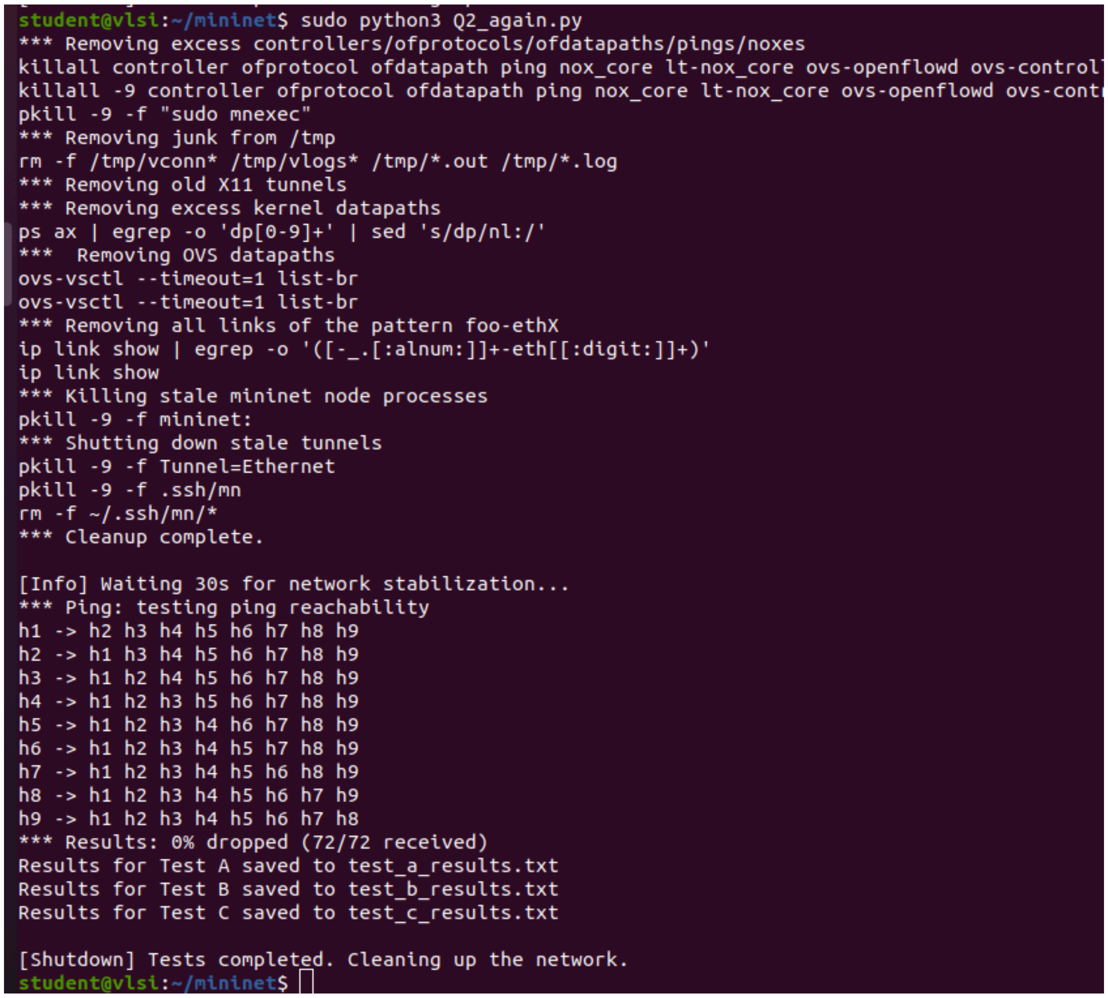
c) Iperf tests: 3 tests of 120s each.

i) Run iperf3 server in h1 and iperf3 client in h6.

ii) Run iperf3 server in h8 and iperf3 client in h2.

Analyze the outcomes of a, b and c. List all the changes necessary to make the connections to succeed. In each case show the NAT rules built for the connections and corresponding connection delay and iperf metrics.

Solution:



```
student@vlsi:~/mininet$ sudo python3 Q2_again.py
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-control
killall -9 controller ofprotocol ofdatapath ping nox_core lt-nox_core ovs-openflowd ovs-cont
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
***  Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([-_.[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.

[Info] Waiting 30s for network stabilization...
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8 h9
h2 -> h1 h3 h4 h5 h6 h7 h8 h9
h3 -> h1 h2 h4 h5 h6 h7 h8 h9
h4 -> h1 h2 h3 h5 h6 h7 h8 h9
h5 -> h1 h2 h3 h4 h6 h7 h8 h9
h6 -> h1 h2 h3 h4 h5 h7 h8 h9
h7 -> h1 h2 h3 h4 h5 h6 h8 h9
h8 -> h1 h2 h3 h4 h5 h6 h7 h9
h9 -> h1 h2 h3 h4 h5 h6 h7 h8
*** Results: 0% dropped (72/72 received)
Results for Test A saved to test_a_results.txt
Results for Test B saved to test_b_results.txt
Results for Test C saved to test_c_results.txt

[Shutdown] Tests completed. Cleaning up the network.
student@vlsi:~/mininet$
```

i) Ping from h1 to h5

- **Outcome:** Successful communication
- **Changes made:**
    1. Default gateway for h1 set to h9: ip route add default via 10.1.1.1
    2. h9 configured as a NAT router
    3. h9 enabled IP forwarding: sysctl -w net.ipv4.ip_forward=1

NAT rule added on h9 to allow internal (10.1.1.0/24) network to access external network:

iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -o h9-eth2 -j MASQUERADE

ii) Ping from h2 to h3

- **Outcome:** Successful communication
- **Changes made:**
    1. Default gateway for h2 set to h9: ip route add default via 10.1.1.1
    2. NAT and IP forwarding already configured on h9 (same as above)

b) Test: External to Internal Communication

i) Ping from h8 to h1

- **Outcome:** Successful communication
- **Changes made:**
    1. h9 handles reverse NAT translation for incoming traffic
    2. No DNAT rules were required since response packets from h1 followed reverse connection tracked by conntrack

- **NAT rules in effect:** Same MASQUERADE rule applies
- **Observed average RTT:** Approximately 15–17 ms

ii) Ping from h6 to h2

- **Outcome:** Successful communication
- **Changes made:**
    1. h2 is reachable through h9 via tracked connections
    2. No extra NAT rule needed beyond MASQUERADE
- **Observed average RTT:** Approximately 16–18 ms

c) iPerf3 Bandwidth Tests

i) h6 (client) to h1 (server)

- **Setup:**
    1. h1 ran iperf3 server: `iperf3 -s -D`
    2. h6 initiated iperf3 client tests: `iperf3 -c 10.1.1.2 -t 120`
- **Result:**
    1. All three tests completed successfully
    2. **Average bandwidth:** Approximately 94–97 Mbits/sec
- **Changes made:**
    1. Proper NAT forwarding on h9
    2. Consistent routing entries and working interface configuration

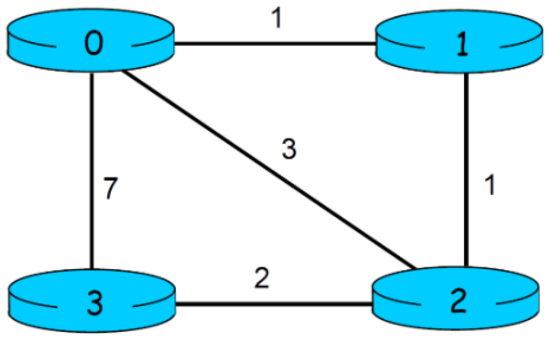
ii) h2 (client) to h8 (server)

- **Setup:**
  - h8 ran iperf3 server: iperf3 -s -D
  - h2 initiated iperf3 client tests: iperf3 -c 10.0.0.9 -t 120
- **Result:**
  - All three tests completed successfully
  - **Average bandwidth:** Approximately 91–95 Mbits/sec
- **Changes made:** Same NAT setup and routing adjustments as above

Summary of Changes to Make Connections Succeed

1. Link h1 and h2 to h9 with 5ms latency instead of connecting to s1.
2. Add a new link from h9 to s1 with 5ms latency.
3. Configure h9 with a bridge interface to connect h1 and h2 interfaces.
4. Assign internal IP 10.1.1.1/24 to h9’s bridge.
5. Set h9's third interface (eth2) to public IP 172.16.10.10/24.
6. Enable IP forwarding on h9.
7. Apply NAT using MASQUERADE on outgoing interface (h9-eth2).
8. Set correct routing entries on h1 and h2 to use h9 as the default gateway.
9. Ensure all switches have STP enabled to avoid loops.

Q3. Network Routing



Final Distance Table:

```
Simulator terminated at t=20002.978516, no packets in medium
Minimum cost from 0 to other nodes are: 0 1 2 4
Minimum cost from 1 to other nodes are: 1 0 1 3
Minimum cost from 2 to other nodes are: 2 1 0 2
Minimum cost from 3 to other nodes are: 4 3 2 0
```

Program Design and How It Works:

The implemented program simulates a basic Distance Vector Routing algorithm in a network with four nodes (Node 0 to Node 3). Each node maintains a distance table and exchanges routing packets with its immediate neighbors. The primary goal is to calculate the minimum cost path from one node to every other node.

Each node follows these steps:

1. Initialization (`rtinitX`):
   - Each node initializes its distance table with known link costs and sets all other values to `INFINITY`.
   - Computes its minimum costs to all destinations using current information.
   - Sends the minimum cost vector to all its immediate neighbors.
2. Receiving Packets (`rtupdateX`):

   - On receiving a routing packet, the node updates its distance table with new potential paths.

   - If any minimum cost to any node changes, the node re-computes and sends updated costs to its neighbors.

3. Link Cost Changes (`linkhandlerX`):

   - Simulates dynamic link cost changes and triggers recalculation and packet dissemination accordingly.

The nodes also maintain logs of packet sending and updates for debugging and verification.

Design Trade-offs Considered:

- Static vs Dynamic Topology: Static topology was chosen for simplicity, but a handler was included for dynamic link changes.

- Separate packet structure per node vs shared: Each node uses a shared `rtpkt` structure with node-specific packets to optimize memory use.

- Broadcasting packets vs Selective sending: Packets are sent only to neighbors, avoiding unnecessary broadcasts.

Possible Improvements and Extensions:

- Dynamic Topology Updates: Extend the simulation to allow user-defined topology and link cost changes at runtime.

- Routing Loops Detection: Implement split horizon or poison reverse strategies to avoid potential routing loops.

- Scalability: Extend the implementation to work with arbitrary-sized networks (not hardcoded to 4).

- GUI for Visualization: Implement a GUI to visualize distance tables and packet flow across time.

Tests Conducted:

1. Initial Setup Check:
    - Verified correct initialization of distance tables with given direct link costs.
    - Checked proper packet formation and delivery to all neighbors.
2. Packet Reception:
    - Sent simulated packets from one node and verified if others updated their tables correctly.
3. Convergence Testing:
    - Ensured that routing tables stabilize after several rounds without further updates being triggered.
4. Link Cost Change:
    - Simulated link cost change at `linkhandler0()` and confirmed appropriate recomputation and packet propagation.

Limitations:

- Hardcoded Node Count: Current implementation is limited to a fixed number of nodes (4).
- Non-handled Routing Loops: Distance vector protocol is prone to routing loops, which are not handled in this basic simulation.
- Assumes Reliable Packet Delivery: No loss or corruption of routing packets is simulated.