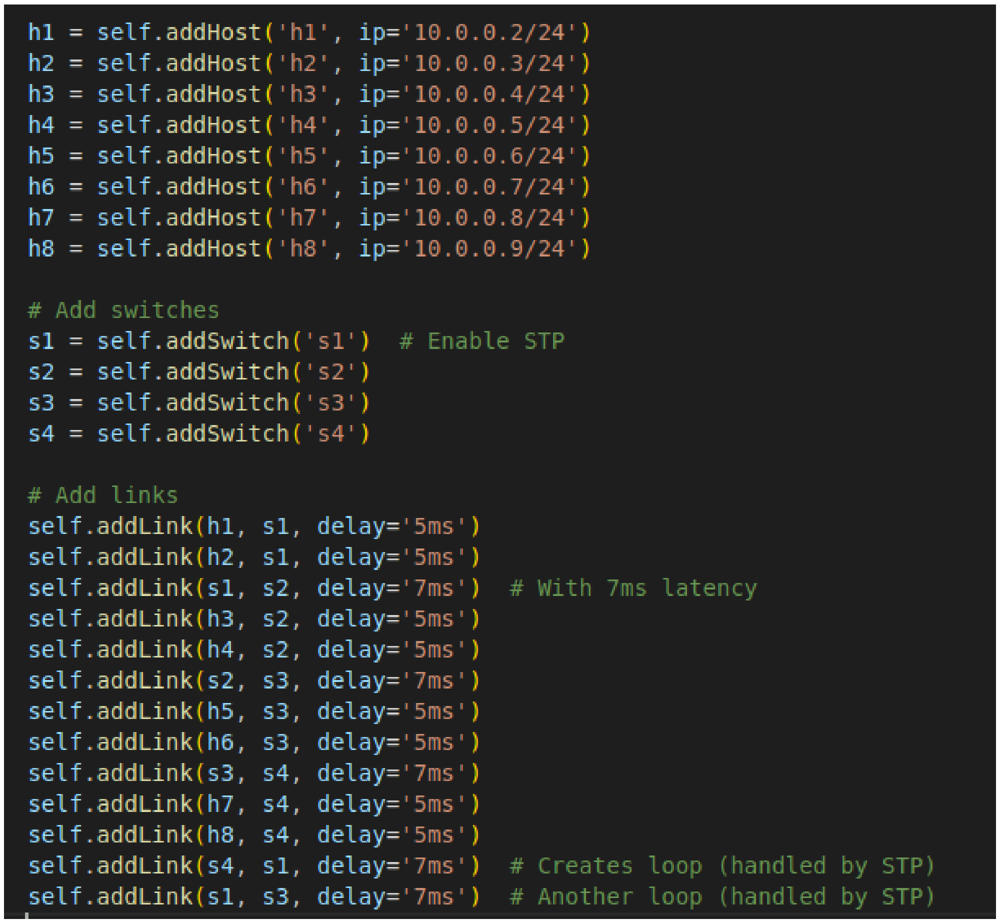
CS 331 Computer Networks Assignment 3

Aayush Parmar 22110181 Bhoumik Patidar 22110049

TASK 1:

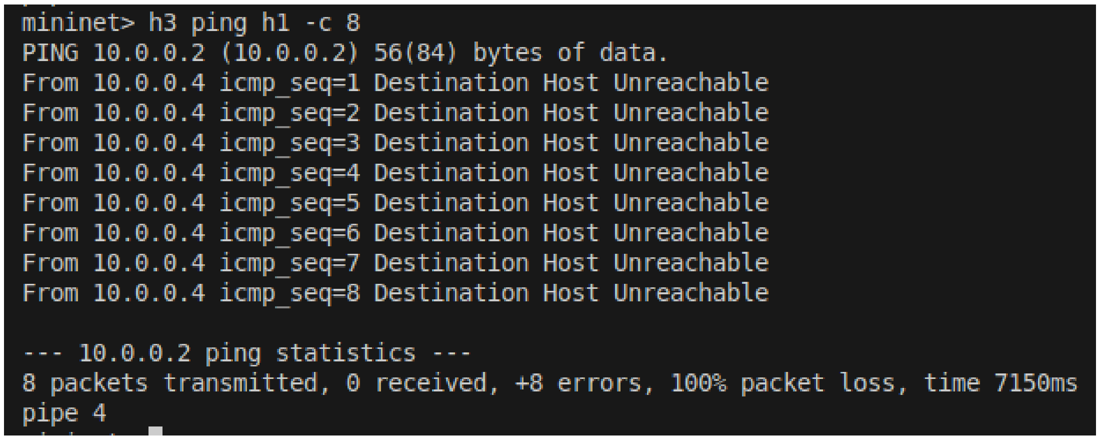
Creating the topology as directed in the question:



```
h1 = self.addHost('h1', ip='10.0.0.2/24')
h2 = self.addHost('h2', ip='10.0.0.3/24')
h3 = self.addHost('h3', ip='10.0.0.4/24')
h4 = self.addHost('h4', ip='10.0.0.5/24')
h5 = self.addHost('h5', ip='10.0.0.6/24')
h6 = self.addHost('h6', ip='10.0.0.7/24')
h7 = self.addHost('h7', ip='10.0.0.8/24')
h8 = self.addHost('h8', ip='10.0.0.9/24')

# Add switches
s1 = self.addSwitch('s1')  # Enable STP
s2 = self.addSwitch('s2')
s3 = self.addSwitch('s3')
s4 = self.addSwitch('s4')

# Add links
self.addLink(h1, s1, delay='5ms')
self.addLink(h2, s1, delay='5ms')
self.addLink(s1, s2, delay='7ms')  # With 7ms latency
self.addLink(h3, s2, delay='5ms')
self.addLink(h4, s2, delay='5ms')
self.addLink(s2, s3, delay='7ms')
self.addLink(h5, s3, delay='5ms')
self.addLink(h6, s3, delay='5ms')
self.addLink(s3, s4, delay='7ms')
self.addLink(h7, s4, delay='5ms')
self.addLink(h8, s4, delay='5ms')
self.addLink(s4, s1, delay='7ms')  # Creates loop (handled by STP)
self.addLink(s1, s3, delay='7ms')  # Another loop (handled by STP)
```

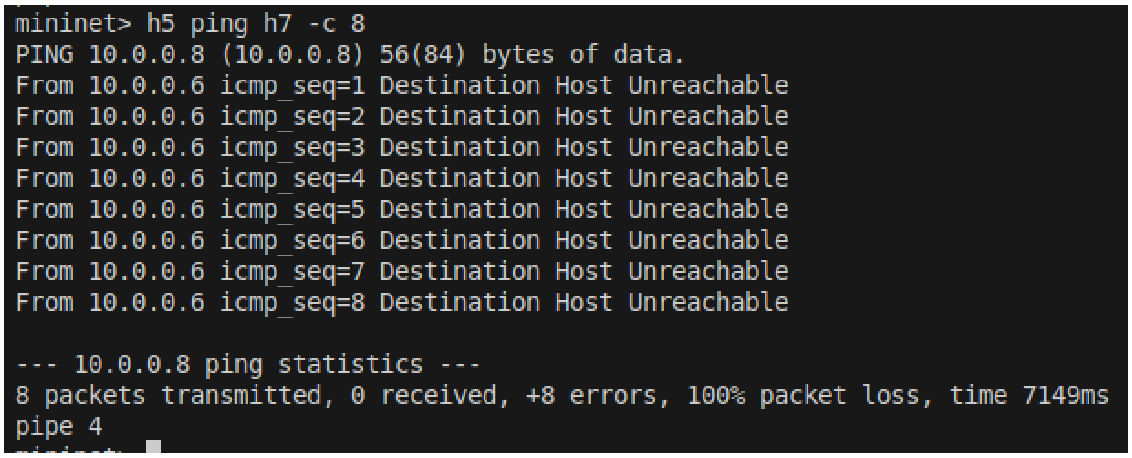
a) **Analysing Behaviour**

Ping h1 from h3



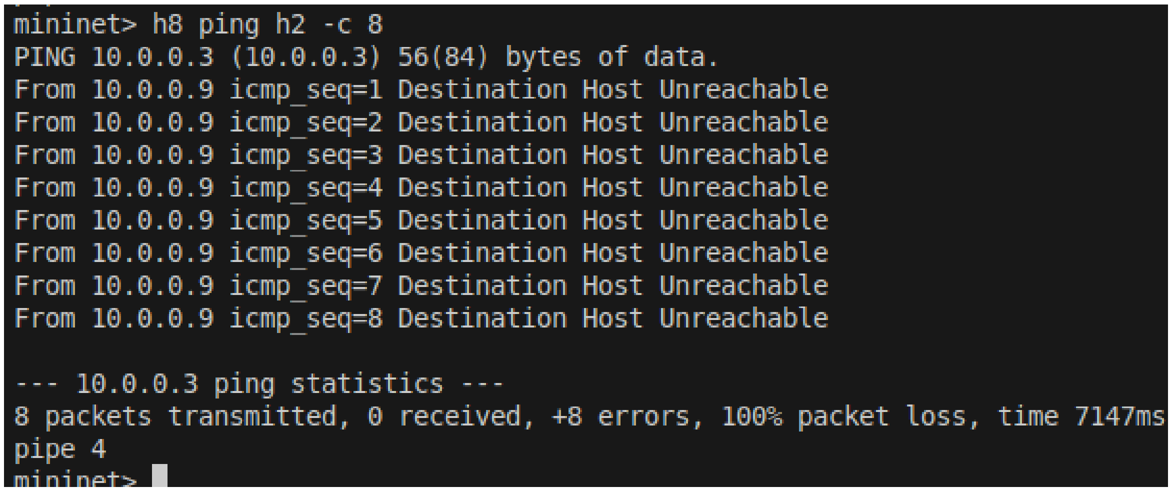
```
mininet> h3 ping h1 -c 8
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=1 Destination Host Unreachable
From 10.0.0.4 icmp_seq=2 Destination Host Unreachable
From 10.0.0.4 icmp_seq=3 Destination Host Unreachable
From 10.0.0.4 icmp_seq=4 Destination Host Unreachable
From 10.0.0.4 icmp_seq=5 Destination Host Unreachable
From 10.0.0.4 icmp_seq=6 Destination Host Unreachable
From 10.0.0.4 icmp_seq=7 Destination Host Unreachable
From 10.0.0.4 icmp_seq=8 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
8 packets transmitted, 0 received, +8 errors, 100% packet loss, time 7150ms
pipe 4
```

Ping h7 from h5



```
mininet> h5 ping h7 -c 8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
From 10.0.0.6 icmp_seq=1 Destination Host Unreachable
From 10.0.0.6 icmp_seq=2 Destination Host Unreachable
From 10.0.0.6 icmp_seq=3 Destination Host Unreachable
From 10.0.0.6 icmp_seq=4 Destination Host Unreachable
From 10.0.0.6 icmp_seq=5 Destination Host Unreachable
From 10.0.0.6 icmp_seq=6 Destination Host Unreachable
From 10.0.0.6 icmp_seq=7 Destination Host Unreachable
From 10.0.0.6 icmp_seq=8 Destination Host Unreachable

--- 10.0.0.8 ping statistics ---
8 packets transmitted, 0 received, +8 errors, 100% packet loss, time 7149ms
pipe 4
```

Ping h2 from h8

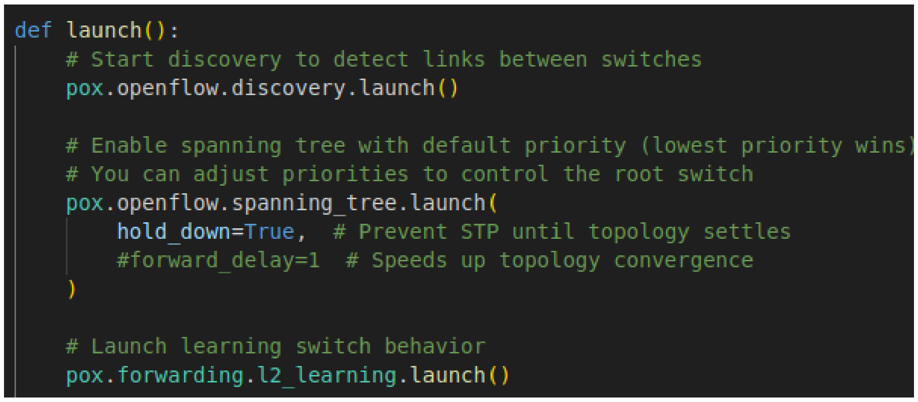


```
mininet> h8 ping h2 -c 8
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
From 10.0.0.9 icmp_seq=1 Destination Host Unreachable
From 10.0.0.9 icmp_seq=2 Destination Host Unreachable
From 10.0.0.9 icmp_seq=3 Destination Host Unreachable
From 10.0.0.9 icmp_seq=4 Destination Host Unreachable
From 10.0.0.9 icmp_seq=5 Destination Host Unreachable
From 10.0.0.9 icmp_seq=6 Destination Host Unreachable
From 10.0.0.9 icmp_seq=7 Destination Host Unreachable
From 10.0.0.9 icmp_seq=8 Destination Host Unreachable

--- 10.0.0.3 ping statistics ---
8 packets transmitted, 0 received, +8 errors, 100% packet loss, time 7147ms
pipe 4
mininet>
```

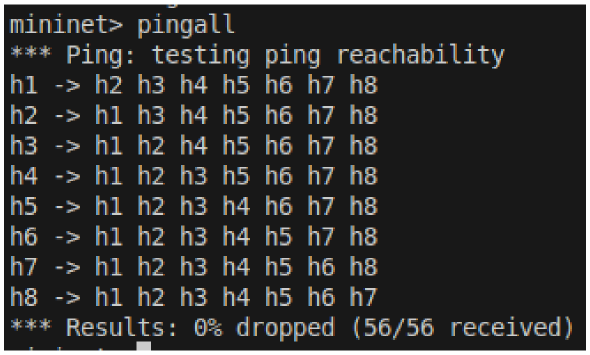
The pings are not successful when using inbuilt OVS-controller.
Reason:

- Open vSwitch (OVS) do not inherently support loops in the sense of actively facilitating their beneficial operation without any mitigating mechanisms.
- **Broadcast Storms:** When a broadcast, multicast, or unknown unicast frame enters a looped topology, it can be endlessly forwarded around the loop. Each switch replicates the frame out of all other ports, leading to an exponential increase in traffic that can quickly overwhelm network devices, consuming bandwidth and processing resources.
- **MAC Address Table Instability:** Switches learn MAC addresses by associating the source MAC address of incoming frames with the port they arrived on. In a looped topology, a frame originating from a host can arrive at a switch via multiple paths. This causes the switch to repeatedly update its MAC address table with the same MAC address associated with different ports, leading to instability and incorrect forwarding decisions.
- **Multiple Frame Copies:** Hosts can receive multiple copies of the same frame, leading to processing overhead and potential application errors.
- **Oscillating Packets:** The packets are oscillating inside the loop instead of reaching the defined host.

b) Enabling STP (Spanning Tree Protocol) using POX controller



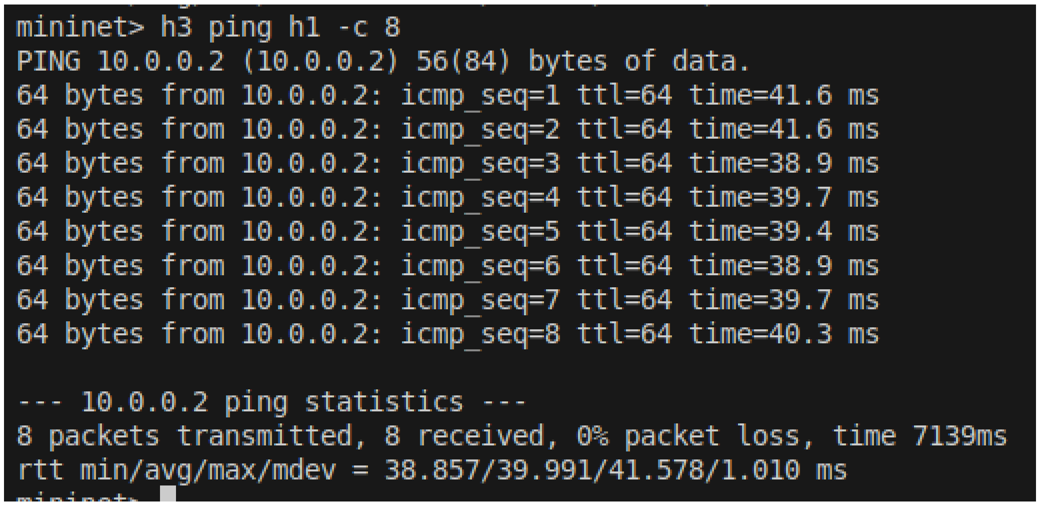
```
def launch():
    # Start discovery to detect links between switches
    pox.openflow.discovery.launch()

    # Enable spanning tree with default priority (lowest priority wins)
    # You can adjust priorities to control the root switch
    pox.openflow.spanning_tree.launch(
        hold_down=True,  # Prevent STP until topology settles
        #forward_delay=1  # Speeds up topology convergence
    )

    # Launch learning switch behavior
    pox.forwarding.l2_learning.launch()
```



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
```

All the hosts are connected with other.

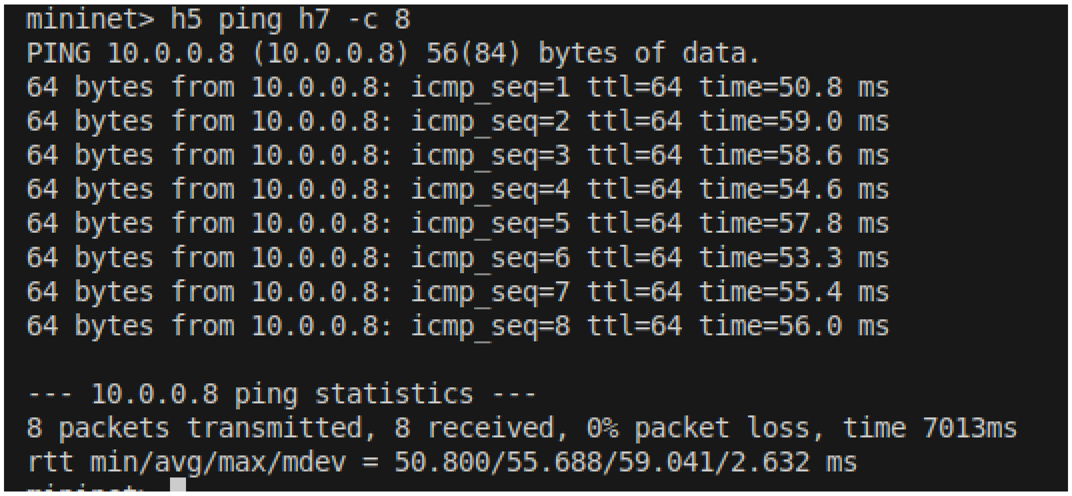
Ping 1 from h3



```
mininet> h3 ping h1 -c 8
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=41.6 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=41.6 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=38.9 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=39.7 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=39.4 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=38.9 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=39.7 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=40.3 ms

--- 10.0.0.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7139ms
rtt min/avg/max/mdev = 38.857/39.991/41.578/1.010 ms
```

Avg Over Three Runs : 40.075 ms

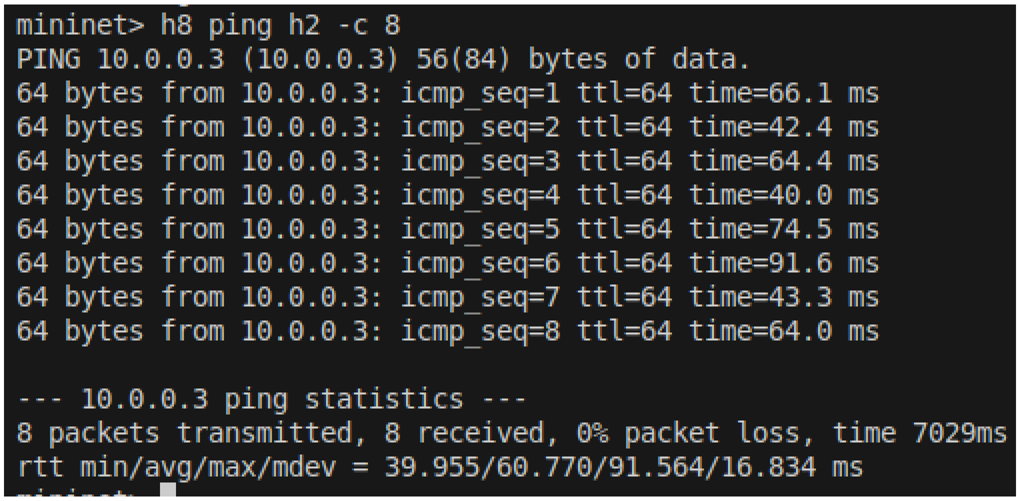
Ping h7 from h5



```
mininet> h5 ping h7 -c 8
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=50.8 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=59.0 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=58.6 ms
64 bytes from 10.0.0.8: icmp_seq=4 ttl=64 time=54.6 ms
64 bytes from 10.0.0.8: icmp_seq=5 ttl=64 time=57.8 ms
64 bytes from 10.0.0.8: icmp_seq=6 ttl=64 time=53.3 ms
64 bytes from 10.0.0.8: icmp_seq=7 ttl=64 time=55.4 ms
64 bytes from 10.0.0.8: icmp_seq=8 ttl=64 time=56.0 ms

--- 10.0.0.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7013ms
rtt min/avg/max/mdev = 50.800/55.688/59.041/2.632 ms
```

Avg Over Three Runs : 55.254 ms

Ping h2 from h8



```
mininet> h8 ping h2 -c 8
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=66.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=42.4 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=64.4 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=40.0 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=74.5 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=91.6 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=43.3 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=64.0 ms

--- 10.0.0.3 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7029ms
rtt min/avg/max/mdev = 39.955/60.770/91.564/16.834 ms
```

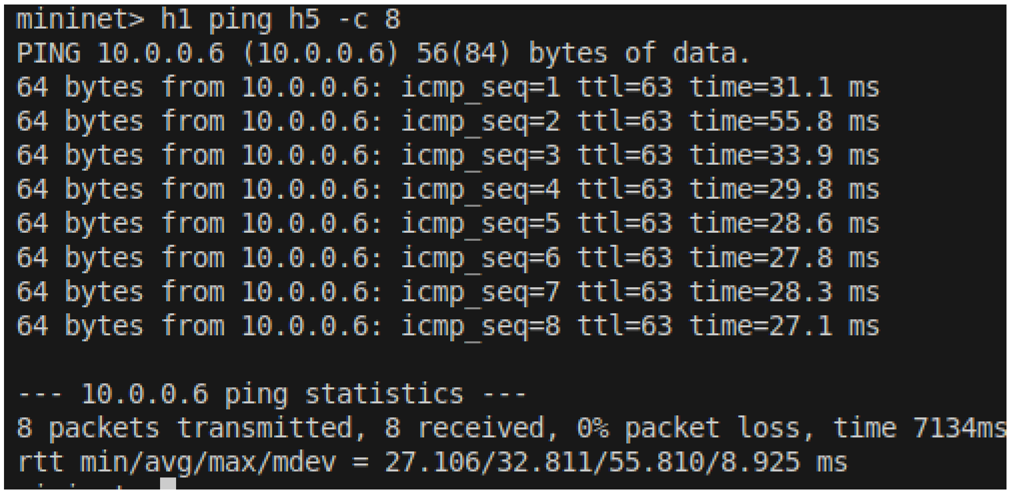
Average Over Three Runs : 61.856 ms

Reason :

- The `discovery.spanning_tree` module in Pox is a direct implementation of the Spanning Tree Protocol.
- STP's fundamental purpose is to detect and prevent Layer 2 loops in a network topology. It achieves this by electing a root bridge and then calculating the shortest path to the root from all other bridges.
- By exchanging Bridge Protocol Data Units (BPDUs), switches running STP collaboratively identify redundant paths and logically block them, creating a loop-free logical topology.
- Spanning Tree Algorithm : The algorithm begins with an election process to designate a single root bridge, the central reference point for the spanning tree. Subsequently, each non-root bridge identifies its root port, representing the most cost-effective path to reach the elected root. On every network segment, a designated port is selected, which is the port on the bridge offering the lowest cost path towards the root bridge for that particular segment. Critically, all ports that are neither root ports nor designated ports are transitioned into a blocking state, effectively disabling redundant links and preventing the creation of forwarding loops. This selective blocking ensures that only one active path exists between any two devices, thus avoiding broadcast storms and MAC address table instability.

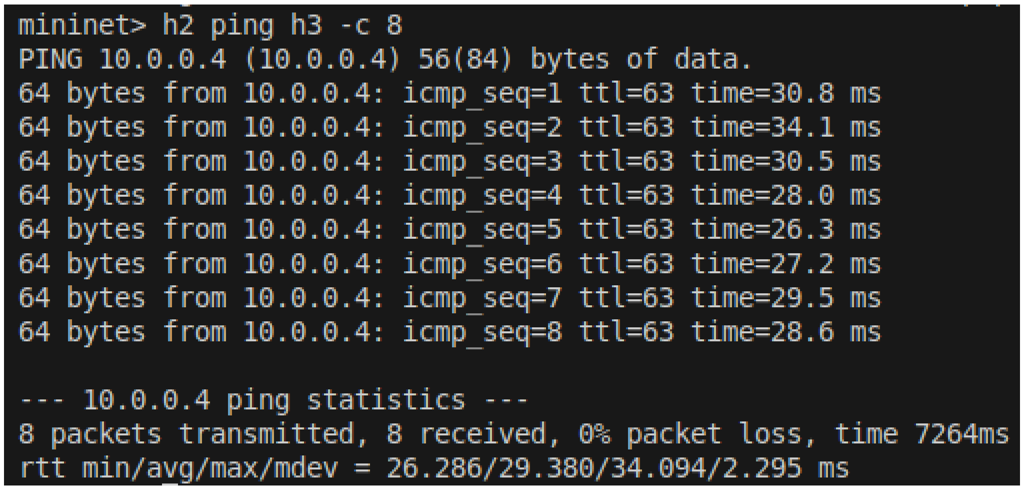
TASK 2:

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



```
mininet> h1 ping h5 -c 8
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=63 time=31.1 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=63 time=55.8 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=63 time=33.9 ms
64 bytes from 10.0.0.6: icmp_seq=4 ttl=63 time=29.8 ms
64 bytes from 10.0.0.6: icmp_seq=5 ttl=63 time=28.6 ms
64 bytes from 10.0.0.6: icmp_seq=6 ttl=63 time=27.8 ms
64 bytes from 10.0.0.6: icmp_seq=7 ttl=63 time=28.3 ms
64 bytes from 10.0.0.6: icmp_seq=8 ttl=63 time=27.1 ms

--- 10.0.0.6 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7134ms
rtt min/avg/max/mdev = 27.106/32.811/55.810/8.925 ms
```

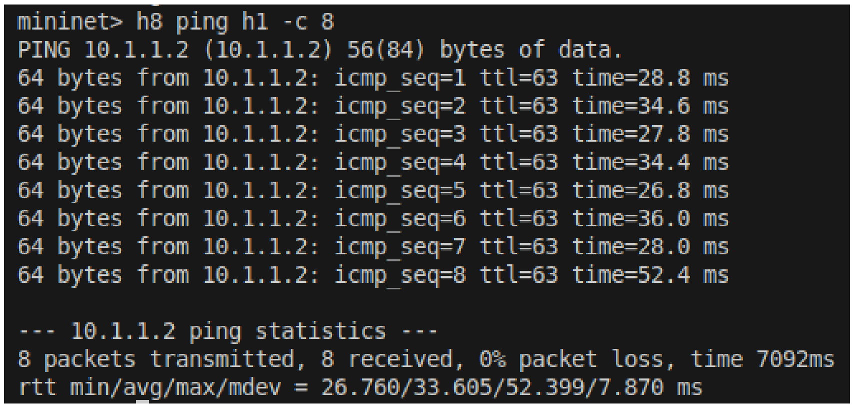
Avg RTT : 31.256 ms



```
mininet> h2 ping h3 -c 8
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=63 time=30.8 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=63 time=34.1 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=63 time=30.5 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=63 time=28.0 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=63 time=26.3 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=63 time=27.2 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=63 time=29.5 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=63 time=28.6 ms

--- 10.0.0.4 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7264ms
rtt min/avg/max/mdev = 26.286/29.380/34.094/2.295 ms
```

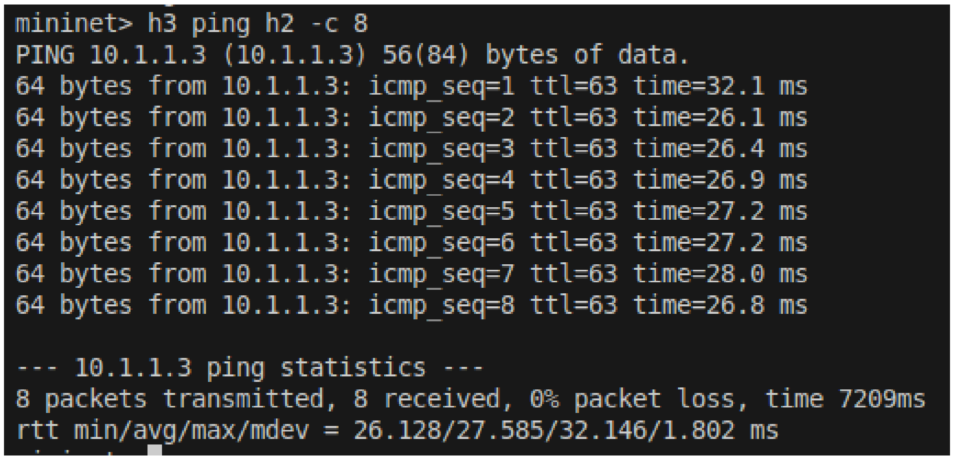
Avg RTT : 29.759 ms

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



```
mininet> h8 ping h1 -c 8
PING 10.1.1.2 (10.1.1.2) 56(84) bytes of data.
64 bytes from 10.1.1.2: icmp_seq=1 ttl=63 time=28.8 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=63 time=34.6 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=63 time=27.8 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=63 time=34.4 ms
64 bytes from 10.1.1.2: icmp_seq=5 ttl=63 time=26.8 ms
64 bytes from 10.1.1.2: icmp_seq=6 ttl=63 time=36.0 ms
64 bytes from 10.1.1.2: icmp_seq=7 ttl=63 time=28.0 ms
64 bytes from 10.1.1.2: icmp_seq=8 ttl=63 time=52.4 ms

--- 10.1.1.2 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7092ms
rtt min/avg/max/mdev = 26.760/33.605/52.399/7.870 ms
```

Avg RTT : 33.052 ms

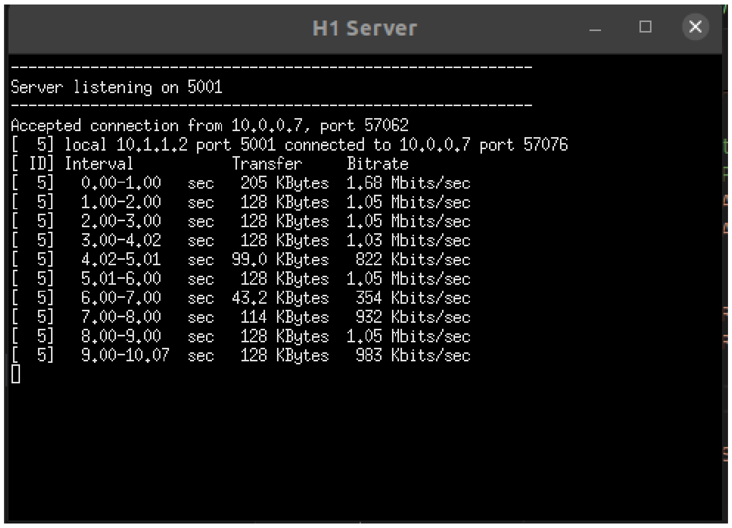


```
mininet> h3 ping h2 -c 8
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.
64 bytes from 10.1.1.3: icmp_seq=1 ttl=63 time=32.1 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=63 time=26.1 ms
64 bytes from 10.1.1.3: icmp_seq=3 ttl=63 time=26.4 ms
64 bytes from 10.1.1.3: icmp_seq=4 ttl=63 time=26.9 ms
64 bytes from 10.1.1.3: icmp_seq=5 ttl=63 time=27.2 ms
64 bytes from 10.1.1.3: icmp_seq=6 ttl=63 time=27.2 ms
64 bytes from 10.1.1.3: icmp_seq=7 ttl=63 time=28.0 ms
64 bytes from 10.1.1.3: icmp_seq=8 ttl=63 time=26.8 ms

--- 10.1.1.3 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7209ms
rtt min/avg/max/mdev = 26.128/27.585/32.146/1.802 ms
```

Avg RTT : 26.367 ms

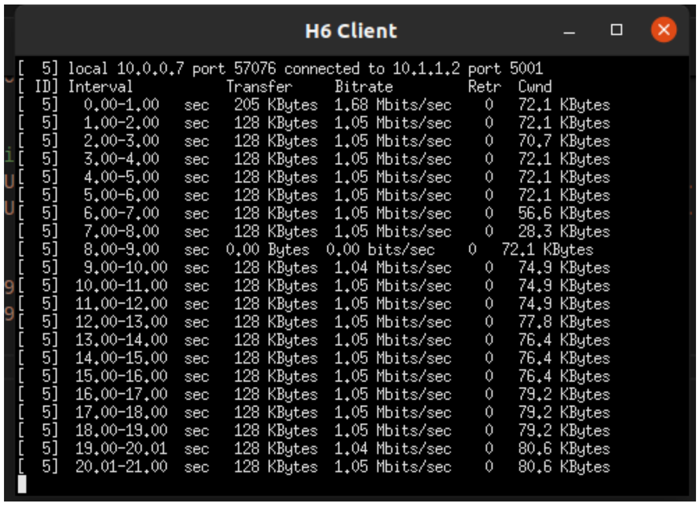
c) Iperf tests: 3 tests of 120s each

Server on h1 and Client on h6



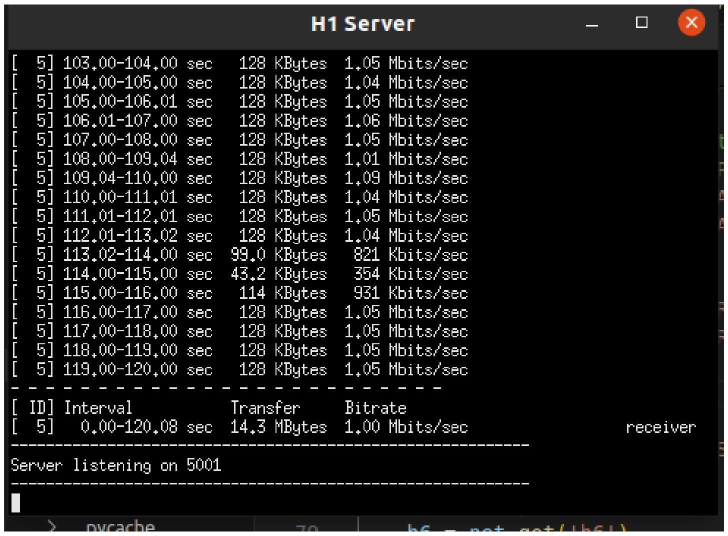
H1 Server

```
------------------------------------------------------------
Server listening on 5001
------------------------------------------------------------
Accepted connection from 10.0.0.7, port 57062
[  5] local 10.1.1.2 port 5001 connected to 10.0.0.7 port 57076
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec   205 KBytes  1.68 Mbits/sec
[  5]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   3.00-4.02   sec   128 KBytes  1.03 Mbits/sec
[  5]   4.02-5.01   sec  99.0 KBytes   822 Kbits/sec
[  5]   5.01-6.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   6.00-7.00   sec  43.2 KBytes   354 Kbits/sec
[  5]   7.00-8.00   sec   114 KBytes   932 Kbits/sec
[  5]   8.00-9.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   9.00-10.07  sec   128 KBytes   983 Kbits/sec
```



H6 Client

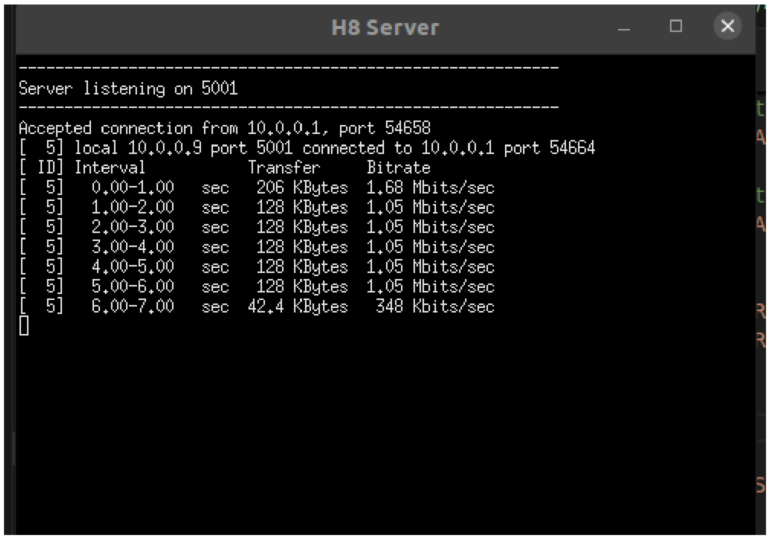
```
[  5] local 10.0.0.7 port 57076 connected to 10.1.1.2 port 5001
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   205 KBytes  1.68 Mbits/sec    0   72.1 KBytes
[  5]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec    0   72.1 KBytes
[  5]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec    0   70.7 KBytes
[  5]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec    0   72.1 KBytes
[  5]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec    0   72.1 KBytes
[  5]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec    0   72.1 KBytes
[  5]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec    0   56.6 KBytes
[  5]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec    0   28.3 KBytes
[  5]   8.00-9.00   sec  0.00 Bytes  0.00 bits/sec    0   72.1 KBytes
[  5]   9.00-10.00  sec   128 KBytes  1.04 Mbits/sec    0   74.9 KBytes
[  5]  10.00-11.00  sec   128 KBytes  1.05 Mbits/sec    0   74.9 KBytes
[  5]  11.00-12.00  sec   128 KBytes  1.05 Mbits/sec    0   74.9 KBytes
[  5]  12.00-13.00  sec   128 KBytes  1.05 Mbits/sec    0   77.8 KBytes
[  5]  13.00-14.00  sec   128 KBytes  1.05 Mbits/sec    0   76.4 KBytes
[  5]  14.00-15.00  sec   128 KBytes  1.05 Mbits/sec    0   76.4 KBytes
[  5]  15.00-16.00  sec   128 KBytes  1.05 Mbits/sec    0   76.4 KBytes
[  5]  16.00-17.00  sec   128 KBytes  1.05 Mbits/sec    0   79.2 KBytes
[  5]  17.00-18.00  sec   128 KBytes  1.05 Mbits/sec    0   79.2 KBytes
[  5]  18.00-19.00  sec   128 KBytes  1.05 Mbits/sec    0   79.2 KBytes
[  5]  19.00-20.01  sec   128 KBytes  1.04 Mbits/sec    0   80.6 KBytes
[  5]  20.01-21.00  sec   128 KBytes  1.05 Mbits/sec    0   80.6 KBytes
```



H1 Server

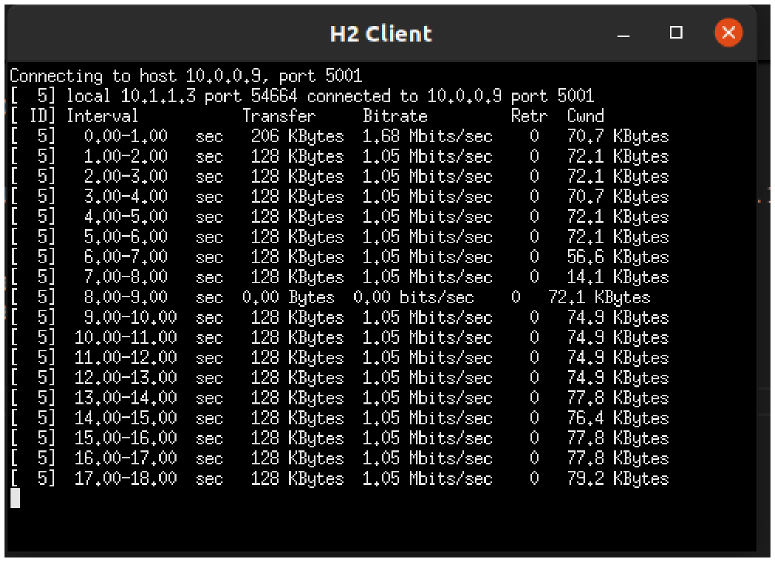
```
[  5] 103.00-104.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 104.00-105.00 sec   128 KBytes  1.04 Mbits/sec
[  5] 105.00-106.01 sec   128 KBytes  1.05 Mbits/sec
[  5] 106.01-107.00 sec   128 KBytes  1.06 Mbits/sec
[  5] 107.00-108.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 108.00-109.04 sec   128 KBytes  1.01 Mbits/sec
[  5] 109.04-110.00 sec   128 KBytes  1.09 Mbits/sec
[  5] 110.00-111.01 sec   128 KBytes  1.04 Mbits/sec
[  5] 111.01-112.01 sec   128 KBytes  1.05 Mbits/sec
[  5] 112.01-113.02 sec   128 KBytes  1.04 Mbits/sec
[  5] 113.02-114.00 sec  99.0 KBytes   821 Kbits/sec
[  5] 114.00-115.00 sec  43.2 KBytes   354 Kbits/sec
[  5] 115.00-116.00 sec   114 KBytes   931 Kbits/sec
[  5] 116.00-117.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 117.00-118.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 118.00-119.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 119.00-120.00 sec   128 KBytes  1.05 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-120.08 sec  14.3 MBytes  1.00 Mbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5001
-----------------------------------------------------------
```

Server on h8 and client on h2

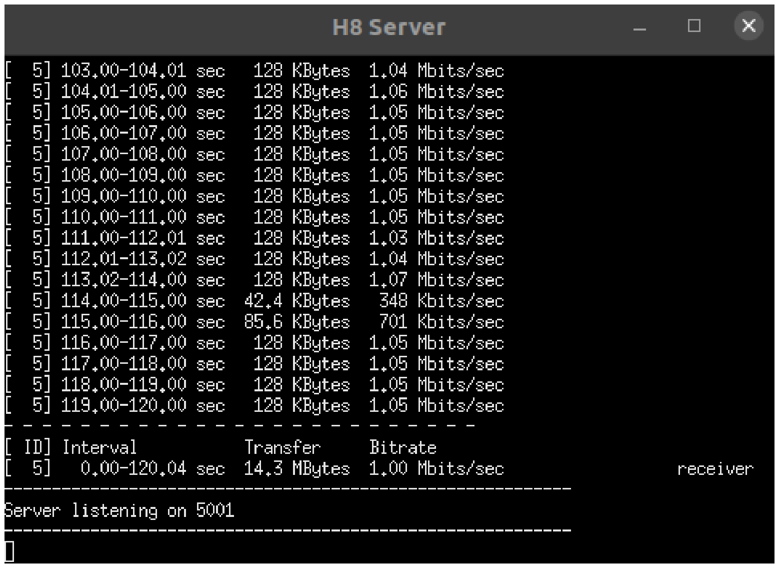


H8 Server

```
-----------------------------------------------------------
Server listening on 5001
-----------------------------------------------------------
Accepted connection from 10.0.0.1, port 54658
[  5] local 10.0.0.9 port 5001 connected to 10.0.0.1 port 54664
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-1.00   sec   206 KBytes  1.68 Mbits/sec
[  5]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec
[  5]   6.00-7.00   sec  42.4 KBytes   348 Kbits/sec
```



```
Connecting to host 10.0.0.9, port 5001
[  5] local 10.1.1.3 port 54664 connected to 10.0.0.9 port 5001
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec   206 KBytes  1.68 Mbits/sec    0   70.7 KBytes
[  5]   1.00-2.00   sec   128 KBytes  1.05 Mbits/sec    0   72.1 KBytes
[  5]   2.00-3.00   sec   128 KBytes  1.05 Mbits/sec    0   72.1 KBytes
[  5]   3.00-4.00   sec   128 KBytes  1.05 Mbits/sec    0   70.7 KBytes
[  5]   4.00-5.00   sec   128 KBytes  1.05 Mbits/sec    0   72.1 KBytes
[  5]   5.00-6.00   sec   128 KBytes  1.05 Mbits/sec    0   72.1 KBytes
[  5]   6.00-7.00   sec   128 KBytes  1.05 Mbits/sec    0   56.6 KBytes
[  5]   7.00-8.00   sec   128 KBytes  1.05 Mbits/sec    0   14.1 KBytes
[  5]   8.00-9.00   sec  0.00 Bytes  0.00 bits/sec    0   72.1 KBytes
[  5]   9.00-10.00  sec   128 KBytes  1.05 Mbits/sec    0   74.9 KBytes
[  5]  10.00-11.00  sec   128 KBytes  1.05 Mbits/sec    0   74.9 KBytes
[  5]  11.00-12.00  sec   128 KBytes  1.05 Mbits/sec    0   74.9 KBytes
[  5]  12.00-13.00  sec   128 KBytes  1.05 Mbits/sec    0   74.9 KBytes
[  5]  13.00-14.00  sec   128 KBytes  1.05 Mbits/sec    0   77.8 KBytes
[  5]  14.00-15.00  sec   128 KBytes  1.05 Mbits/sec    0   76.4 KBytes
[  5]  15.00-16.00  sec   128 KBytes  1.05 Mbits/sec    0   77.8 KBytes
[  5]  16.00-17.00  sec   128 KBytes  1.05 Mbits/sec    0   77.8 KBytes
[  5]  17.00-18.00  sec   128 KBytes  1.05 Mbits/sec    0   79.2 KBytes
```

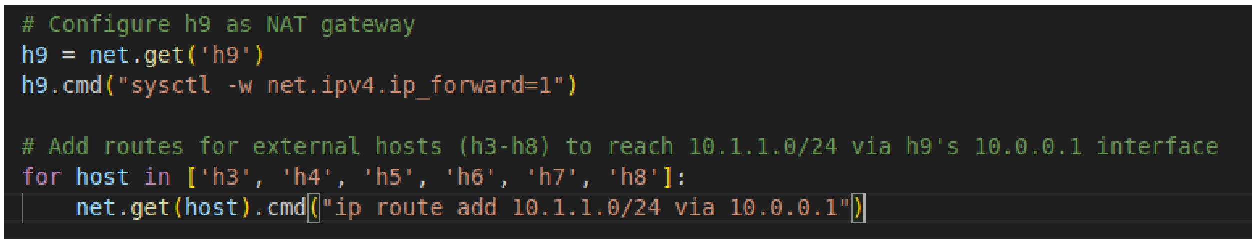


```
[  5] 103.00-104.01 sec   128 KBytes  1.04 Mbits/sec
[  5] 104.01-105.00 sec   128 KBytes  1.06 Mbits/sec
[  5] 105.00-106.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 106.00-107.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 107.00-108.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 108.00-109.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 109.00-110.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 110.00-111.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 111.00-112.01 sec   128 KBytes  1.03 Mbits/sec
[  5] 112.01-113.02 sec   128 KBytes  1.04 Mbits/sec
[  5] 113.02-114.00 sec   128 KBytes  1.07 Mbits/sec
[  5] 114.00-115.00 sec  42.4 KBytes   348 Kbits/sec
[  5] 115.00-116.00 sec  85.6 KBytes   701 Kbits/sec
[  5] 116.00-117.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 117.00-118.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 118.00-119.00 sec   128 KBytes  1.05 Mbits/sec
[  5] 119.00-120.00 sec   128 KBytes  1.05 Mbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate
[  5]   0.00-120.04 sec  14.3 MBytes  1.00 Mbits/sec                  receiver
-----------------------------------------------------------
Server listening on 5001
-----------------------------------------------------------
```

Observation:

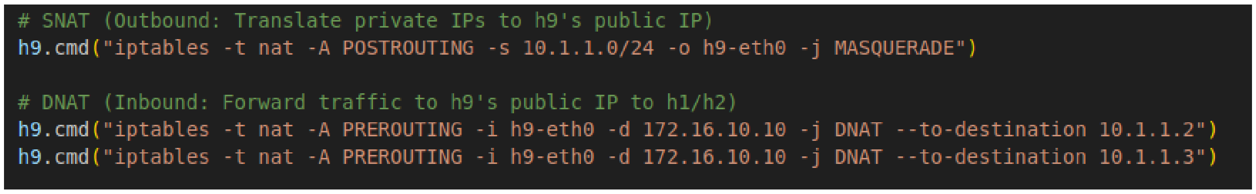
- An average of 5.2 ms of delay was added when connecting hosts inside the local network to the outside when compared with NAT and without NAT.
- The 5 ms delay is due to the latency between h9 and s1.
- The 0.2 ms delay is due to the processing of forwarding packets to the corresponding interface.

Changes Performed to implement NAT on h9:



```
# Configure h9 as NAT gateway
h9 = net.get('h9')
h9.cmd("sysctl -w net.ipv4.ip_forward=1")

# Add routes for external hosts (h3-h8) to reach 10.1.1.0/24 via h9's 10.0.0.1 interface
for host in ['h3', 'h4', 'h5', 'h6', 'h7', 'h8']:
    net.get(host).cmd("ip route add 10.1.1.0/24 via 10.0.0.1")
```

1. **Enable IP Forwarding on h9:**
   This allows `h9` to act as a router, forwarding IP packets between its interfaces. Without this, traffic would not be routed from one subnet to another.
2. **Routing Setup on h3–h8:**
   The path to `10.1.1.0/24` is explicitly routed through `h9`.



```
# SNAT (Outbound: Translate private IPs to h9's public IP)
h9.cmd("iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -o h9-eth0 -j MASQUERADE")

# DNAT (Inbound: Forward traffic to h9's public IP to h1/h2)
h9.cmd("iptables -t nat -A PREROUTING -i h9-eth0 -d 172.16.10.10 -j DNAT --to-destination 10.1.1.2")
h9.cmd("iptables -t nat -A PREROUTING -i h9-eth0 -d 172.16.10.10 -j DNAT --to-destination 10.1.1.3")
```

3. **SNAT (Source NAT)**
   When packets go from the internal network (`10.1.1.0/24`) to the external side (through `h9-eth0`, which has IP `172.16.10.10`), it replaces the source IP of those packets with `172.16.10.10`, So that return traffic comes back to `h9`, which can reverse the translation and forward it internally.

4. **DNAT (Destination NAT)**
   Incoming traffic sent to `172.16.10.10` is forwarded to either `10.1.1.2` (h1) or `10.1.1.3` (h2) based on the port specified and in address.

Summary: NAT Implementation Flow

1. Hosts h1 and h2 send packets to the outside world via `h9` (default gateway).
2. `h9` replaces their source IP with its external IP `172.16.10.10` using SNAT.
3. External hosts (e.g., h3–h8) send traffic to `172.16.10.10`, which gets mapped to internal IPs (`10.1.1.2` or `10.1.1.3`) via DNAT.
4. `h9` forwards traffic between interfaces as allowed by iptables rules and IP forwarding.

TASK 3:

- Distance Table (`dt0/dt1`): Stores the cost to each destination node via each possible neighbor.
- mincost: The smallest cost known to reach each destination node.
- `connectcostsX[]`: Direct costs from node X to other nodes.
- `rtinitX()`: Initializes node X's distance table and sends initial packets to neighbors.
- `rtupdateX()`: Called when node X receives a distance vector from a neighbor.
- `sendpktX()`: Sends updated distance vectors to neighbors.
- `calc_min_cost_nodeX()`: Computes the current best costs from X to all other nodes.
- `calc_send_pktX()`: Decides if distance vector changed and triggers send.
- `printdtX()`: Prints the distance table in a formatted view.
- `linkhandlerX()`: Handles link cost changes dynamically.

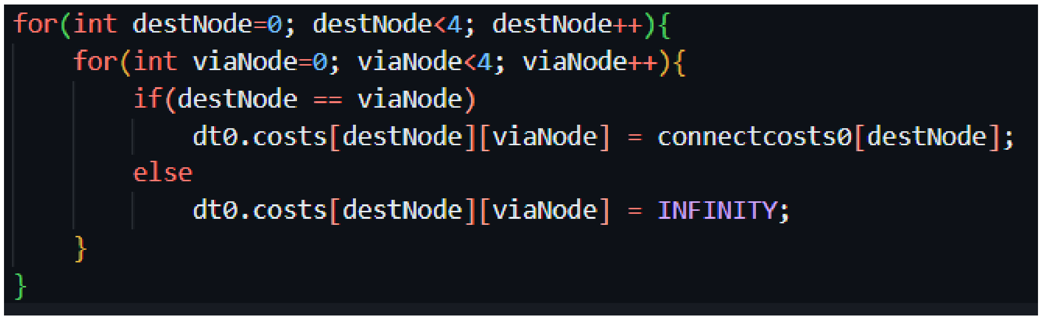
Core Algorithm: Distance Vector Routing

Each node maintains a distance table `dt0.costs[dest][via]`

- `dest` = destination node
- `via` = neighbor used to reach `dest`
- `costs[dest][via]` = cost to reach `dest` via `via`

The algorithm follows **Bellman-Ford** logic:
For every destination D and for each neighbor N, Cost to D via N = cost to N + cost from N to D. Pick the path with the **minimum** cost.

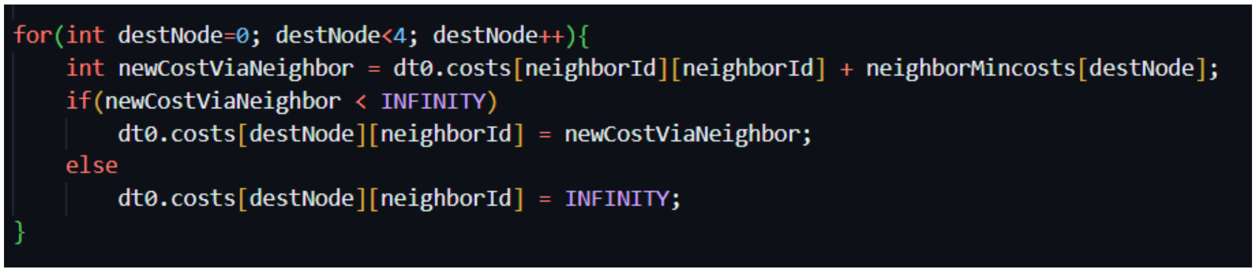
`rtinit0()`: Initialization



```
for(int destNode=0; destNode<4; destNode++){
    for(int viaNode=0; viaNode<4; viaNode++){
        if(destNode == viaNode)
            dt0.costs[destNode][viaNode] = connectcosts0[destNode];
        else
            dt0.costs[destNode][viaNode] = INFINITY;
    }
}
```

1. Set up initial costs:
   - `dt0.costs[i][i] = connectcosts0[i]`: direct neighbors
   - All other entries = `INFINITY`
2. Compute the minimum cost from node 0 to all other nodes and store in `min_cost_node0[4]`.
3. Send initial distance vector to neighbors using `sendpkt0()`.

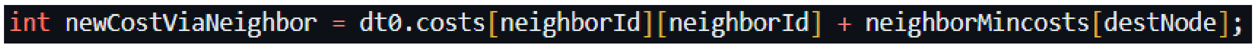
`rtupdate0(struct rtpkt *receivedPacket)`: When Node 0 Receives an Update



```
for(int destNode=0; destNode<4; destNode++){
    int newCostViaNeighbor = dt0.costs[neighborId][neighborId] + neighborMincosts[destNode];
    if(newCostViaNeighbor < INFINITY)
        dt0.costs[destNode][neighborId] = newCostViaNeighbor;
    else
        dt0.costs[destNode][neighborId] = INFINITY;
}
```

1. Extract:
   - `neighborId = receivedPacket->sourceid`
   - `neighborMincosts[] = receivedPacket->mincost[]` (neighbor's best known cost to each node)

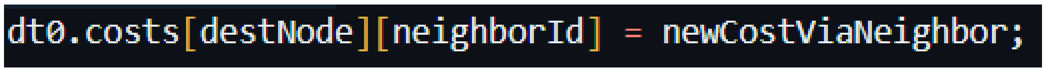
For each possible **destination node `i`**:



```
int newCostViaNeighbor = dt0.costs[neighborId][neighborId] + neighborMincosts[destNode];
```

2. `dt0.costs[neighborId][neighborId]`: Cost from node 0 to neighbor
3. `neighborMincosts[i]`: Neighbor's cost to destination `i`

If the sum is less than `INFINITY`, update:



```
dt0.costs[destNode][neighborId] = newCostViaNeighbor;
```

4. After updating all `dt0.costs[i][neighborId]`, recalculate the **minimum cost to all nodes** using `calc_min_cost_node0()`.
5. If the **minimum cost has changed** for any destination, send updated packets to all neighbors.

`sendpkt0()`: Send Current Min Costs to Neighbors

- Prepares and sends packets to all directly connected neighbors (except itself).
- Each packet contains the current minimum costs (`min_cost_node0[]`).
- Sends via `tolayer2()` and prints debug output.

`calc_send_pkt0()`: Check for Updates

This checks if the newly calculated `min_cost_node0[]` differs from the old one:

- If yes → send updated packet using `sendpkt0()`
- If no → do nothing

`linkhandler0(int linkid, int newcost)`: Handle Link Cost Changes (Dynamic Topology)

- This function handles **dynamic link changes** (e.g., a cost changes from 3 → 5).
- Adjusts the distance table for all destinations routed via the changed link.
- Triggers a recalculation and packet send if needed.

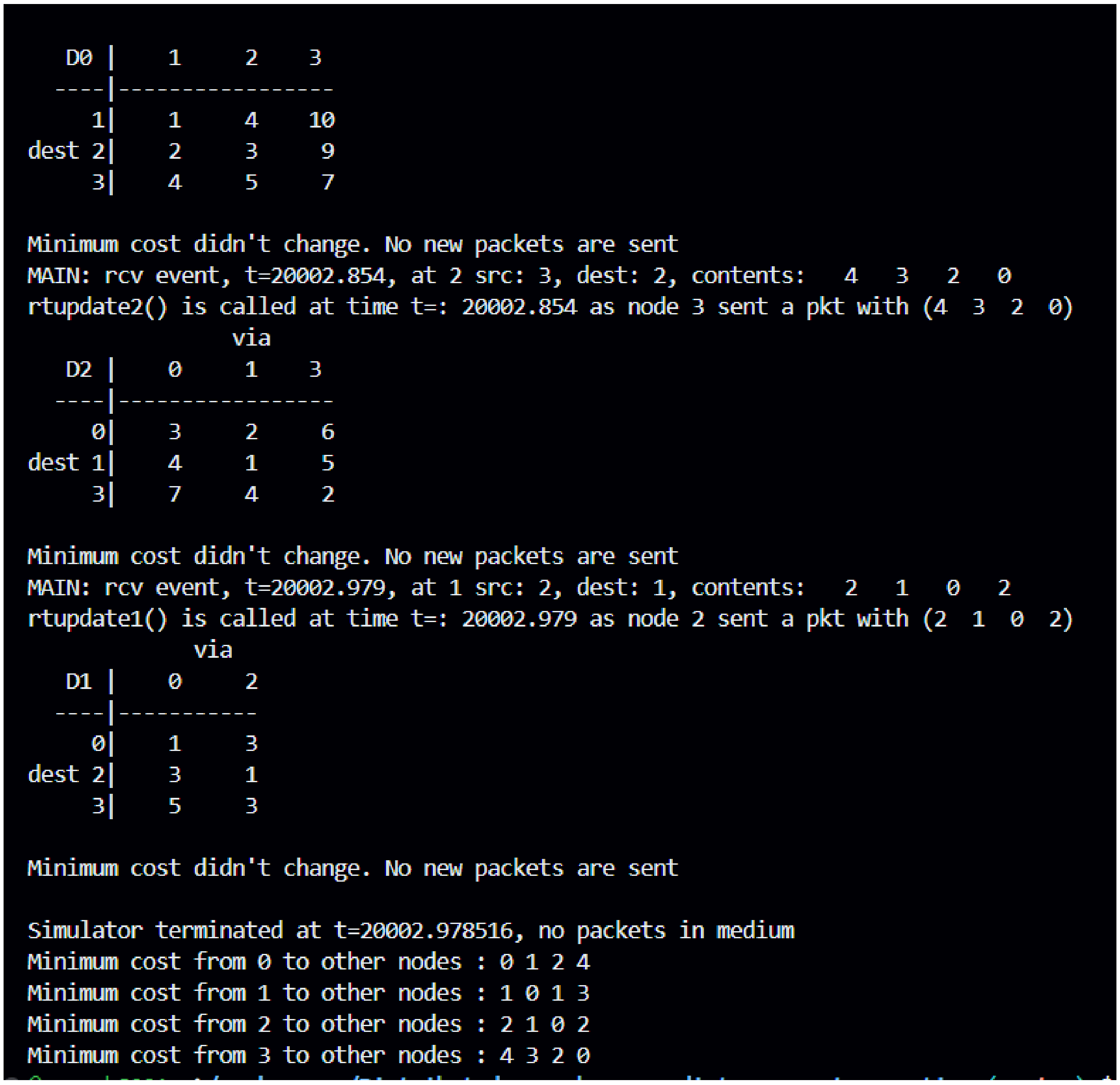
Summary

Each node maintains:

- A table of distances via each neighbor.
- A record of the best known cost to each destination.
- Communication via `sendpktX()` when there's a change.
- Reactions to changes in network topology through `rtupdateX()` and `linkhandlerX()`.

These Functions have to be written for all the for node file i.e node0.c, node1.c, node2.c, node3.c using appropriate initial values.

The files can be compiled using cc distance_vector.c node0.c node1.c node2.c node3.c



```
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|    4     5     7

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=20002.854, at 2 src: 3, dest: 2, contents:   4   3   2   0
rtupdate2() is called at time t=: 20002.854 as node 3 sent a pkt with (4  3  2  0)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
     3|    7     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=20002.979, at 1 src: 2, dest: 1, contents:   2   1   0   2
rtupdate1() is called at time t=: 20002.979 as node 2 sent a pkt with (2  1  0  2)
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    5     3

Minimum cost didn't change. No new packets are sent

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

Running the ./a.out executable.