Computer Networks Assignment-3

12.04.2025

Birudugadda Srivibhav (22110050)

Srivathsa Vamsi Chaturvedula (22110260)

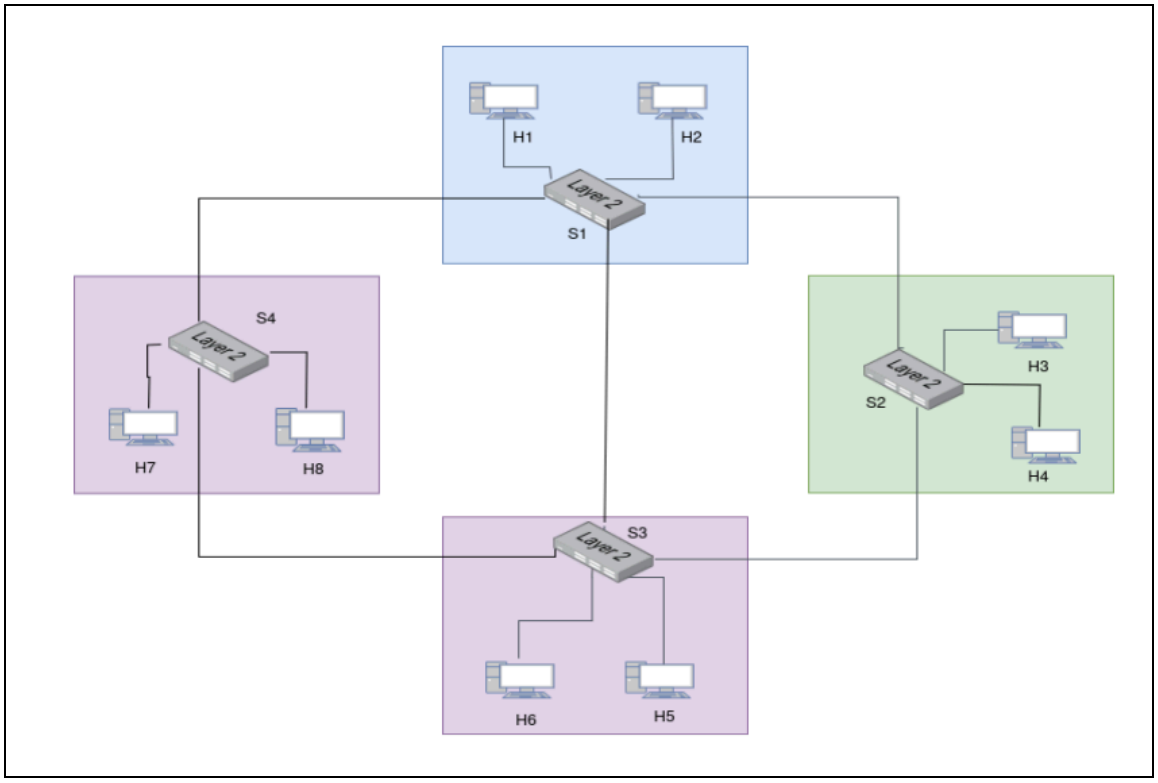
Computer Science and Engineering

IIT Gandhinagar

Q1: Network Loops

This report documents the implementation and analysis of a custom network topology using Mininet, a network emulator that creates a realistic virtual network on a single machine. The problem involved creating a network with four switches arranged in a ring topology with a cross-connection, and eight hosts distributed across these switches. The primary objective was to establish connectivity between hosts and analyze the network performance through ping tests.

Physical Topology



The implemented network consists of:

- Four switches (s1, s2, s3, s4) arranged in a ring topology
- A cross-connection between s1 and s3
- Eight hosts distributed across the switches in color-coded zones:
    - Blue zone: h1 and h2 connected to s1
    - Green zone: h3 and h4 connected to s2
    - Purple zones: h5 and h6 connected to s3, and h7 and h8 connected to s4

Link Configuration

All links in the network were configured with specific latency parameters:

- Host-to-switch links: 5ms delay
- Switch-to-switch links: 7ms delay

IP Addressing Scheme

The hosts were assigned IP addresses in the 10.0.0.0/24 subnet:

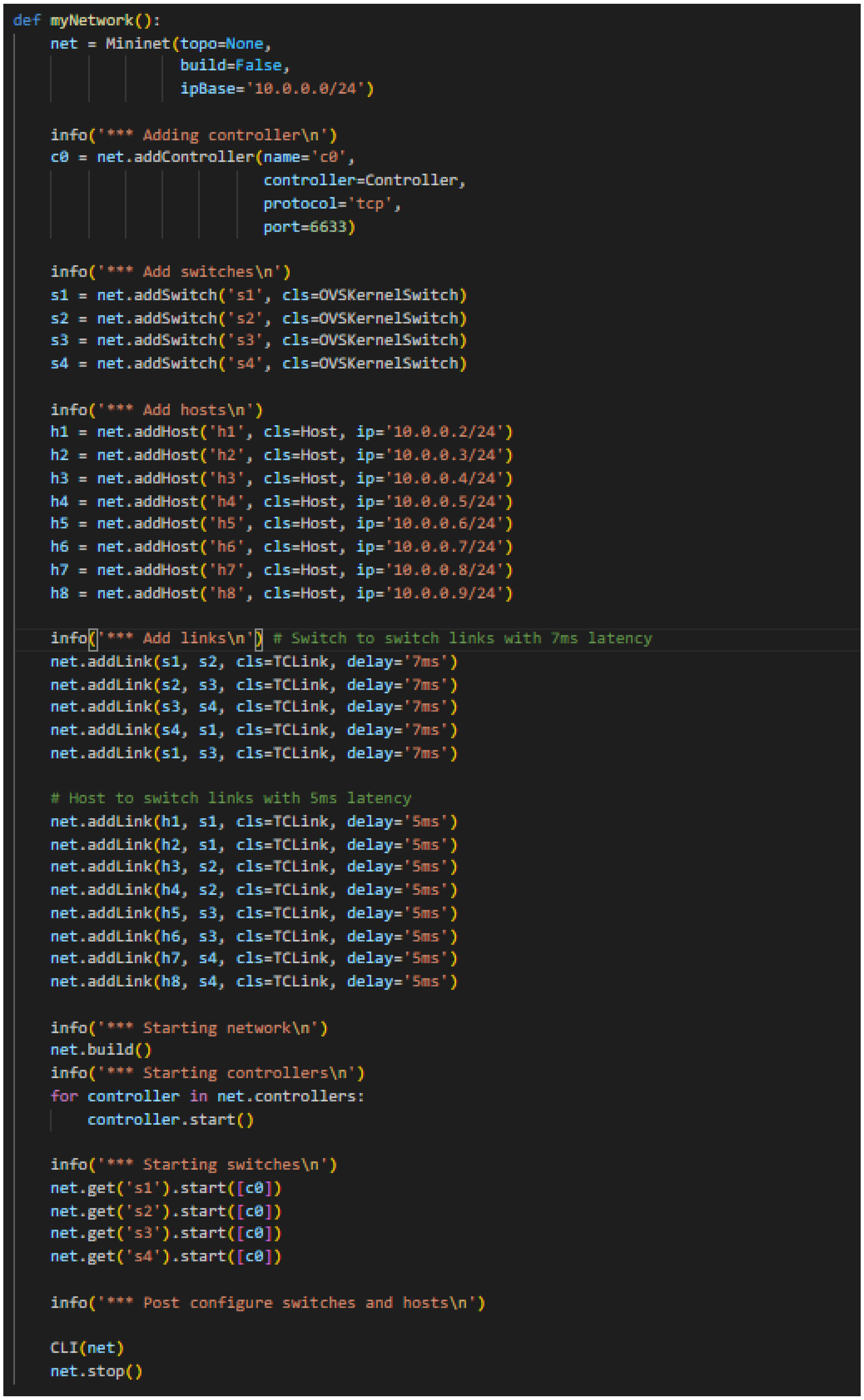
- h1: 10.0.0.2
- h2: 10.0.0.3
- h3: 10.0.0.4
- h4: 10.0.0.5
- h5: 10.0.0.6
- h6: 10.0.0.7
- h7: 10.0.0.8
- h8: 10.0.0.9

Code Overview

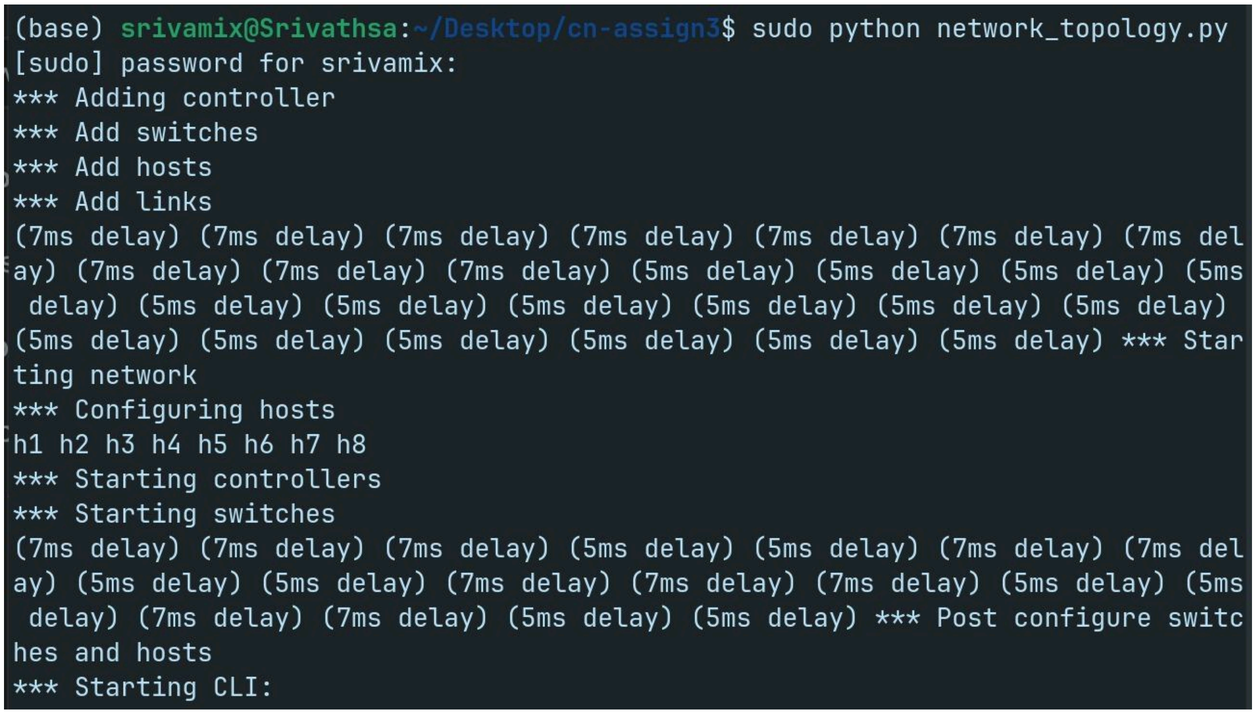
The script defines a function myNetwork() that:

- Creates a Mininet instance with IP base 10.0.0.0/24
- Adds a controller (c0)
- Adds four switches (s1-s4) using OVSKernelSwitch
- Adds eight hosts (h1-h8) with sequential IP addresses
- Configures links between switches (7ms delay) and between hosts and switches (5ms delay)
- Builds and starts the network
- Launches the CLI for user interaction

The network topology creates a ring of switches (s1-s2-s3-s4-s1) with an additional cross-connection between s1 and s3, which introduces loops in the physical topology.



```python
def myNetwork():
    net = Mininet(topo=None,
                  build=False,
                  ipBase='10.0.0.0/24')

    info('*** Adding controller\n')
    c0 = net.addController(name='c0',
                           controller=Controller,
                           protocol='tcp',
                           port=6633)

    info('*** Add switches\n')
    s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
    s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
    s3 = net.addSwitch('s3', cls=OVSKernelSwitch)
    s4 = net.addSwitch('s4', cls=OVSKernelSwitch)

    info('*** Add hosts\n')
    h1 = net.addHost('h1', cls=Host, ip='10.0.0.2/24')
    h2 = net.addHost('h2', cls=Host, ip='10.0.0.3/24')
    h3 = net.addHost('h3', cls=Host, ip='10.0.0.4/24')
    h4 = net.addHost('h4', cls=Host, ip='10.0.0.5/24')
    h5 = net.addHost('h5', cls=Host, ip='10.0.0.6/24')
    h6 = net.addHost('h6', cls=Host, ip='10.0.0.7/24')
    h7 = net.addHost('h7', cls=Host, ip='10.0.0.8/24')
    h8 = net.addHost('h8', cls=Host, ip='10.0.0.9/24')

    info('*** Add links\n') # Switch to switch links with 7ms latency
    net.addLink(s1, s2, cls=TCLink, delay='7ms')
    net.addLink(s2, s3, cls=TCLink, delay='7ms')
    net.addLink(s3, s4, cls=TCLink, delay='7ms')
    net.addLink(s4, s1, cls=TCLink, delay='7ms')
    net.addLink(s1, s3, cls=TCLink, delay='7ms')

    # Host to switch links with 5ms latency
    net.addLink(h1, s1, cls=TCLink, delay='5ms')
    net.addLink(h2, s1, cls=TCLink, delay='5ms')
    net.addLink(h3, s2, cls=TCLink, delay='5ms')
    net.addLink(h4, s2, cls=TCLink, delay='5ms')
    net.addLink(h5, s3, cls=TCLink, delay='5ms')
    net.addLink(h6, s3, cls=TCLink, delay='5ms')
    net.addLink(h7, s4, cls=TCLink, delay='5ms')
    net.addLink(h8, s4, cls=TCLink, delay='5ms')

    info('*** Starting network\n')
    net.build()
    info('*** Starting controllers\n')
    for controller in net.controllers:
        controller.start()

    info('*** Starting switches\n')
    net.get('s1').start([c0])
    net.get('s2').start([c0])
    net.get('s3').start([c0])
    net.get('s4').start([c0])

    info('*** Post configure switches and hosts\n')

    CLI(net)
    net.stop()
```

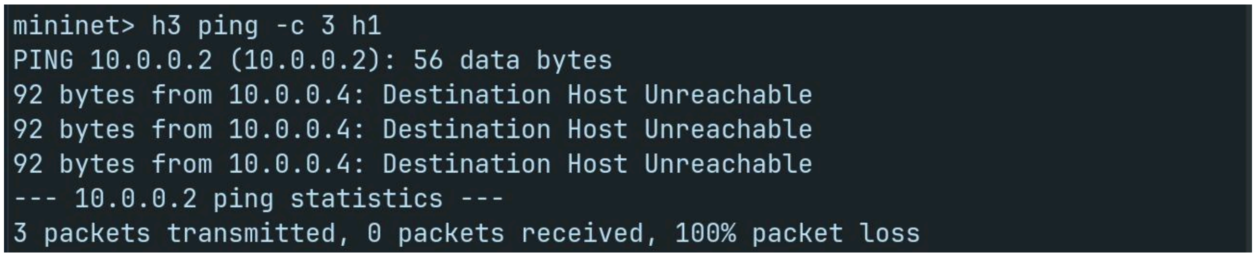


```
(base) srivamix@Srivathsa:~/Desktop/cn-assign3$ sudo python network_topology.py
[sudo] password for srivamix:
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
(7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms del
ay) (7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms
 delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay)
(5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) *** Star
ting network
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controllers
*** Starting switches
(7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms delay) (7ms delay) (7ms del
ay) (5ms delay) (5ms delay) (7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms
 delay) (7ms delay) (7ms delay) (5ms delay) (5ms delay) *** Post configure switc
hes and hosts
*** Starting CLI:
```

Implementation Challenges

After creating the topology, initial ping tests failed with "Destination Host Unreachable" errors, indicating:

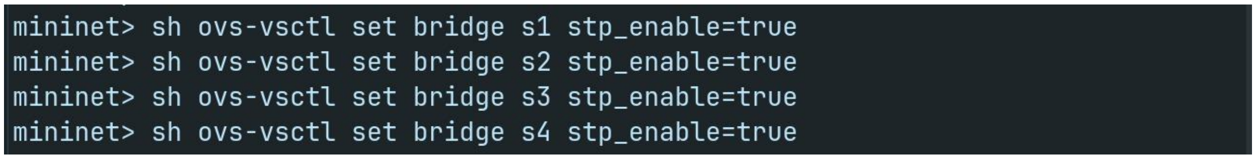
- Layer 2 connectivity issues
- Failed ARP resolution between hosts
- Potential network loops causing broadcast storms



```
mininet> h3 ping -c 3 h1
PING 10.0.0.2 (10.0.0.2): 56 data bytes
92 bytes from 10.0.0.4: Destination Host Unreachable
92 bytes from 10.0.0.4: Destination Host Unreachable
92 bytes from 10.0.0.4: Destination Host Unreachable
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 packets received, 100% packet loss
```

Solution Implementation

To resolve the connectivity issues without modifying the network topology, **Spanning Tree Protocol** was enabled on all switches:



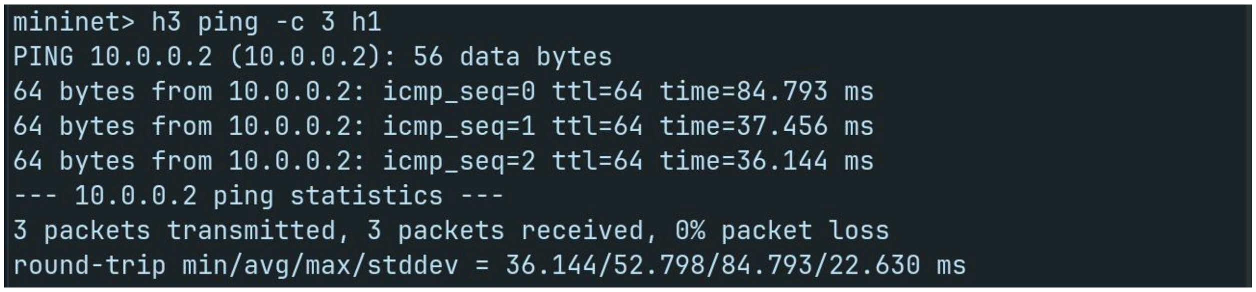
```
mininet> sh ovs-vsctl set bridge s1 stp_enable=true
mininet> sh ovs-vsctl set bridge s2 stp_enable=true
mininet> sh ovs-vsctl set bridge s3 stp_enable=true
mininet> sh ovs-vsctl set bridge s4 stp_enable=true
```

This implementation:

- Created a loop-free logical topology
- Prevented broadcast storms
- Enabled proper MAC address learning and frame forwarding
- Required approximately 30 seconds for convergence

Performance Testing and Analysis

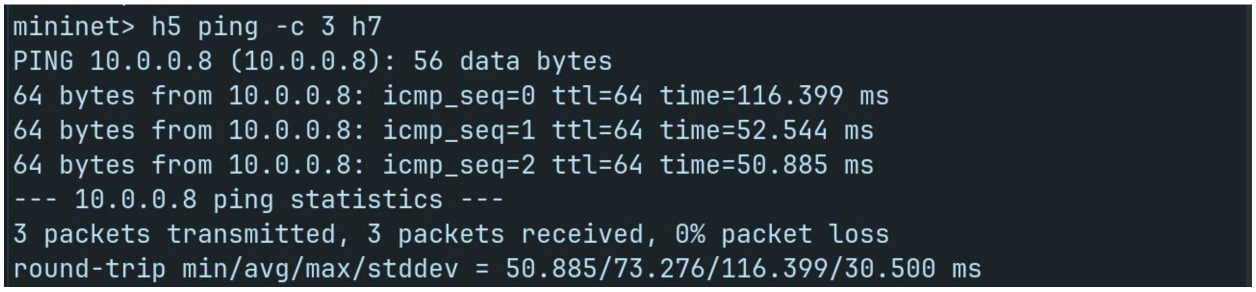
Ping Test: h3 to h1



```
mininet> h3 ping -c 3 h1
PING 10.0.0.2 (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: icmp_seq=0 ttl=64 time=84.793 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=37.456 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=36.144 ms
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 36.144/52.798/84.793/22.630 ms
```

- Expected path: h3 → s2 → s1 → h1
- Theoretical round-trip latency: 34ms (5ms + 7ms + 5ms) × 2
- First ping showed higher latency (84.793ms) due to ARP resolution and flow table setup
- Subsequent pings stabilized near theoretical expectation (37.456ms, 36.144ms)
- 0% packet loss confirms successful connectivity

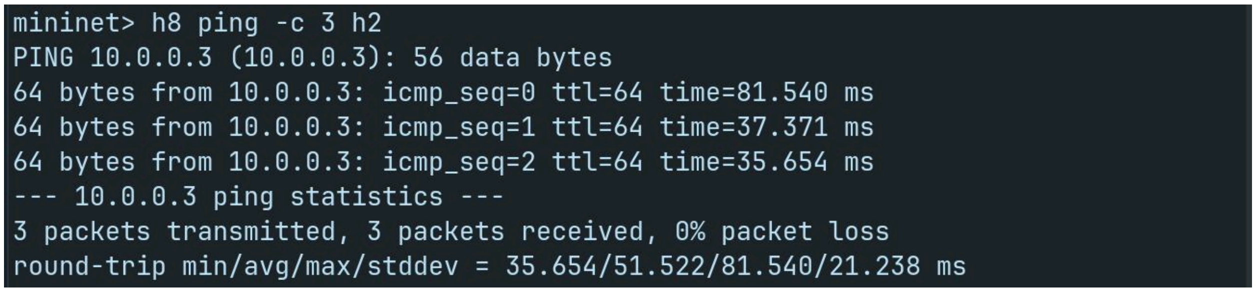
Ping Test: h5 to h7



```
mininet> h5 ping -c 3 h7
PING 10.0.0.8 (10.0.0.8): 56 data bytes
64 bytes from 10.0.0.8: icmp_seq=0 ttl=64 time=116.399 ms
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=52.544 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=50.885 ms
--- 10.0.0.8 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 50.885/73.276/116.399/30.500 ms
```

- Direct path would be: h5 → s3 → s4 → h7
- Theoretical direct path round-trip latency: 34ms
- Observed stabilized latency (~51ms) suggests an alternate path
- Likely path after STP: h5 → s3 → s1 → s4 → h7 (48ms theoretical round-trip)
- Higher initial latency (116.399ms) due to ARP resolution and flow setup
- 0% packet loss confirms successful connectivity

Ping Test: h8 to h2



```
mininet> h8 ping -c 3 h2
PING 10.0.0.3 (10.0.0.3): 56 data bytes
64 bytes from 10.0.0.3: icmp_seq=0 ttl=64 time=81.540 ms
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=37.371 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=35.654 ms
--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 35.654/51.522/81.540/21.238 ms
```

- Expected path: h8 → s4 → s1 → h2
- Theoretical round-trip latency: 34ms (5ms + 7ms + 5ms) × 2
- First ping showed higher latency (81.540ms) due to initial setup
- Subsequent pings closely matched theoretical expectation (37.371ms, 35.654ms)
- 0% packet loss confirms successful connectivity

Observations and Findings

STP Path Selection

The Spanning Tree Protocol modified the logical topology by blocking certain links to prevent loops:

- The direct s3-s4 link appears to have been blocked by STP
- Traffic between h5 and h7 took a longer path through s1
- The s4-s1 and s2-s1 direct links remained active in the spanning tree

Latency Patterns

All ping tests exhibited consistent latency patterns:

- First ping: Significantly higher latency (81-116ms) due to:
  - ARP resolution process
  - Flow table population

    - Controller interaction
    - MAC address learning
- Subsequent pings: Stabilized latency reflecting the physical path with minimal processing overhead
- Latency values closely matched theoretical calculations when accounting for STP path selection

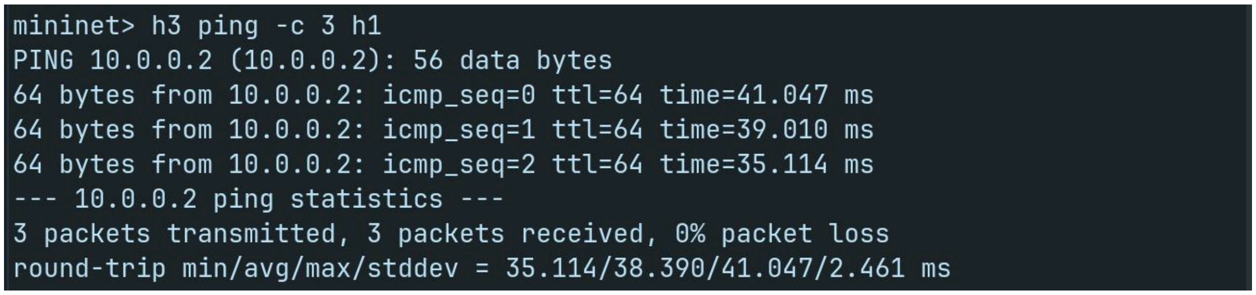
Network Stability

The implementation demonstrated excellent stability:

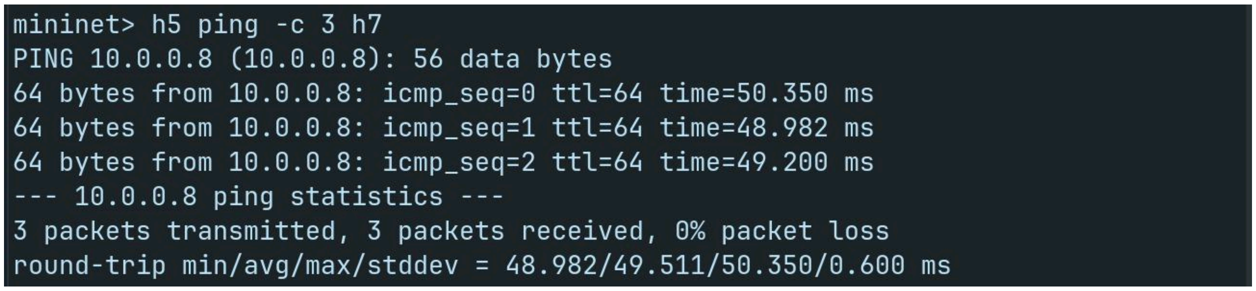
- 0% packet loss across all tests
- Consistent latency after initial setup
- Successful connectivity between hosts in different network segments

Subsequent Ping Test Observations

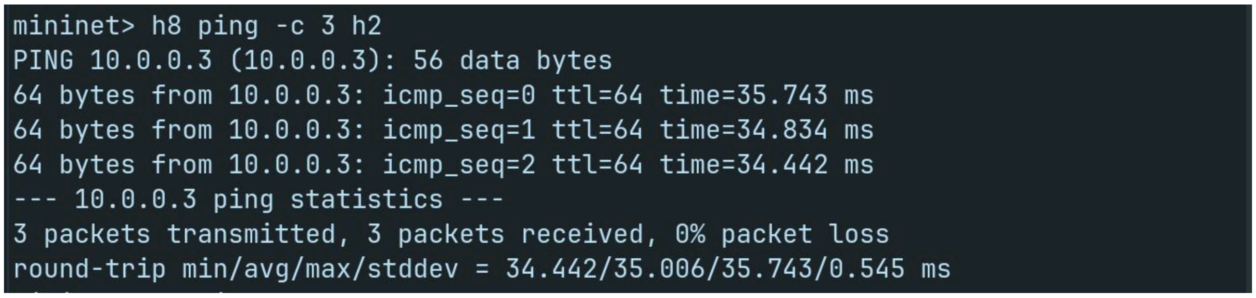
An important additional observation was made when running the same ping commands for a second time after a 30-second interval.



```
mininet> h3 ping -c 3 h1
PING 10.0.0.2 (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: icmp_seq=0 ttl=64 time=41.047 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=39.010 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=35.114 ms
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 35.114/38.390/41.047/2.461 ms
```



```
mininet> h5 ping -c 3 h7
PING 10.0.0.8 (10.0.0.8): 56 data bytes
64 bytes from 10.0.0.8: icmp_seq=0 ttl=64 time=50.350 ms
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=48.982 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=49.200 ms
--- 10.0.0.8 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 48.982/49.511/50.350/0.600 ms
```



```
mininet> h8 ping -c 3 h2
PING 10.0.0.3 (10.0.0.3): 56 data bytes
64 bytes from 10.0.0.3: icmp_seq=0 ttl=64 time=35.743 ms
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=34.834 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=34.442 ms
--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 34.442/35.006/35.743/0.545 ms
```

The results demonstrate a significant difference from the first round of ping tests:

- Consistent latency across all pings: The first ping no longer exhibits the significantly higher latency observed in the initial tests.
- Low standard deviation: The standard deviations indicate minimal variation between ping times.

The same pattern was observed when I ran the ping commands for the third time as well, after another 30-second interval. The absence of higher latency in the first ping during subsequent tests can be attributed to:

- Persistent ARP cache entries: The ARP resolution process had already occurred during the first round of tests, and the MAC address mappings were still cached.
- Established flow entries: The OpenFlow switches had already installed the necessary flow entries in their tables.
- Stable STP topology: The Spanning Tree Protocol had fully converged, creating a stable loop-free logical topology.

This observation confirms that the higher initial latency in the first round of tests was indeed due to setup processes rather than inherent network characteristics, and demonstrates the efficiency of the network once these initial processes are complete.

Conclusion

The implementation successfully created a functional network topology with four switches and eight hosts. Initial connectivity issues were resolved by enabling Spanning Tree Protocol, creating a loop-free logical topology without modifying the physical design.

The ping tests confirmed full connectivity between hosts with latency values aligning with theoretical expectations. Notably, subsequent ping tests conducted after a 30-second interval showed consistent latency values without the initial higher latency, confirming that the elevated first-ping times were due to setup processes rather than inherent network characteristics.

Q2:Configure Host-based NAT

Required Changes for Successful Connections

This implementation features a sophisticated network topology that integrates both public and private network segments interconnected through a NAT (Network Address Translation) gateway. The network structure consists of:

- Public Domain (10.0.0.0/24): Encompasses six hosts (h3-h8) distributed across four switches (s1-s4) in a ring topology with a cross-connection
- Private Domain (10.1.1.0/24): Contains two hosts (h1-h2) connected to a dedicated switch (s5)
- NAT Gateway (h9): Functions as the intermediary between public and private segments with dual network interfaces

The switch infrastructure features STP (Spanning Tree Protocol) enabled on all public-facing switches (s1-s4) to eliminate potential network loops, while maintaining a dedicated private switch (s5) for isolated internal communications.

Network Segmentation

The implementation separates traffic into distinct domains:

- Public Network Infrastructure:
    - Four switches (s1-s4) with STP enabled to prevent broadcast storms
    - Six hosts (h3-h8) with public IP addresses in the 10.0.0.0/24 subnet
    - Switch-to-switch connections with 7ms latency
    - Host-to-switch connections with 5ms latency
    - Default gateway configured as 10.0.0.1 (h9's public interface)
- Private Network Infrastructure:
    - Dedicated switch (s5) for internal communications
    - Two hosts (h1, h2) with private addresses (10.1.1.2/24, 10.1.1.3/24)
    - Host-to-switch connections with minimal 1ms latency
    - Default gateway configured as 10.1.1.1 (h9's private interface)

NAT Gateway Configuration

The gateway host (h9) features a comprehensive NAT implementation:

Interface Configuration:

- Public-facing interface (h9-eth0):
    - Primary address: 10.0.0.1/24 (default gateway for public hosts)
    - NAT public address: 172.16.10.10/24
    - Port forwarding addresses: 172.16.10.11/24 (mapped to h1), 172.16.10.12/24 (mapped to h2)
- Private-facing interface (h9-eth1):
    - Private subnet gateway: 10.1.1.1/24

IP Forwarding:

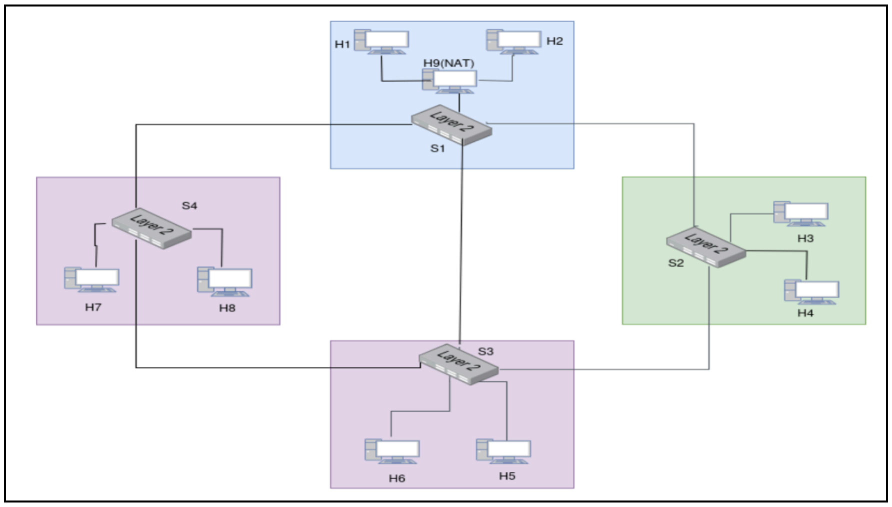
- Kernel-level IP forwarding enabled via sysctl

Network Address Translation Rules:

- Outbound translation: Private subnet (10.1.1.0/24) traffic masqueraded through h9-eth0
- Inbound port forwarding:
    - ICMP traffic to 172.16.10.11 redirected to h1 (10.1.1.2)
    - ICMP traffic to 172.16.10.12 redirected to h2 (10.1.1.3)
    - TCP port 5201 (iperf) on 172.16.10.11 forwarded to h1:5201
    - TCP port 5201 (iperf) on 172.16.10.12 forwarded to h2:5201

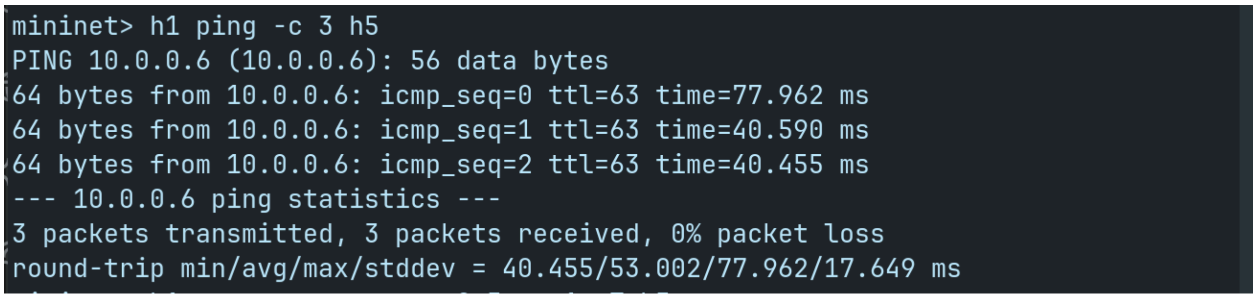
Firewall Rules:

- Outbound traffic from private network permitted
- Established/related connection traffic allowed to return
- Specifically permitted inbound services (ICMP, TCP port 5201) to private hosts



Test communication to an external host from an internal host

a) Ping to h5 from h1:



```
mininet> h1 ping -c 3 h5
PING 10.0.0.6 (10.0.0.6): 56 data bytes
64 bytes from 10.0.0.6: icmp_seq=0 ttl=63 time=77.962 ms
64 bytes from 10.0.0.6: icmp_seq=1 ttl=63 time=40.590 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=63 time=40.455 ms
--- 10.0.0.6 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 40.455/53.002/77.962/17.649 ms
```

The first packet has a significantly higher RTT compared to subsequent packets due to initial setup processes such as:

- ARP resolution (h9 needs to resolve MAC addresses for h1 and h5)
- Flow table population in the switches (OpenFlow rules are installed for forwarding traffic)
- Subsequent packets show stabilized latency as the network path has already been established.

Expected Path

The traffic likely follows this path:

- Private Network:
  - h1 → s5 → h9 (NAT gateway)
- Public Network:
  - h9 → s1 → s3 → h5

Latency Breakdown

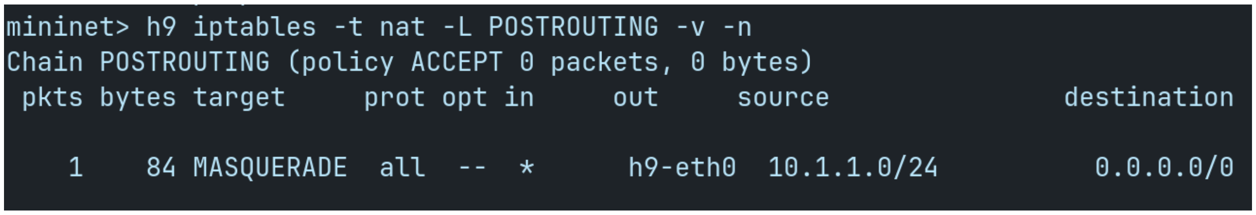
- Host-to-switch latency (h1 → s5): ~1ms
- Switch-to-NAT latency (s5 → h9): ~1ms
- Public network traversal (h9 → s1 → s3 → h5): ~17ms one-way

Total expected one-way latency: ~19ms

Total expected round-trip latency: ~38ms

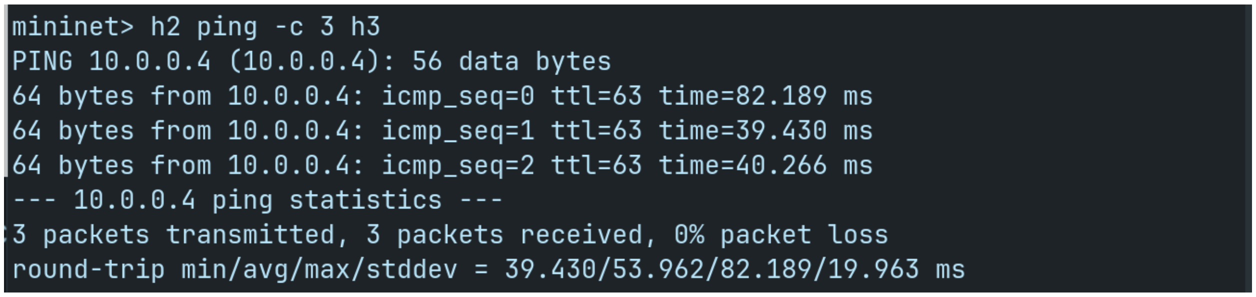
The observed RTT after stabilization closely matches with the expected RTT.

NAT table



```
mininet> h9 iptables -t nat -L POSTROUTING -v -n
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    1    84 MASQUERADE  all  --  *      h9-eth0  10.1.1.0/24          0.0.0.0/0
```

b) Ping to h3 from h2:



```
mininet> h2 ping -c 3 h3
PING 10.0.0.4 (10.0.0.4): 56 data bytes
64 bytes from 10.0.0.4: icmp_seq=0 ttl=63 time=82.189 ms
64 bytes from 10.0.0.4: icmp_seq=1 ttl=63 time=39.430 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=63 time=40.266 ms
--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 39.430/53.962/82.189/19.963 ms
```

Expected Path

The traffic likely follows this path:

- Private Network:
  - h2 → s5 → h9 (NAT gateway)
- Public Network:
  - h9 → s1 → s2 → h3

Latency Breakdown

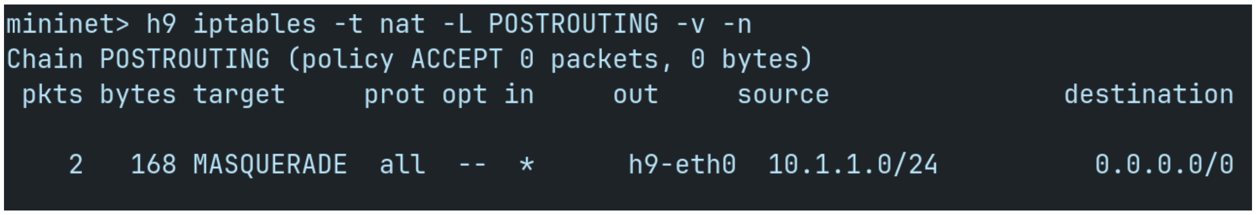
- Host-to-switch latency (h2 → s5): ~1ms
- Switch-to-NAT latency (s5 → h9): ~1ms
- Public network traversal (h9 → s1 → s2 → h3): ~17ms one-way

Total expected one-way latency: ~19ms

Total expected round-trip latency: ~38ms

The observed RTT after stabilization closely matches with the expected RTT.

NAT table

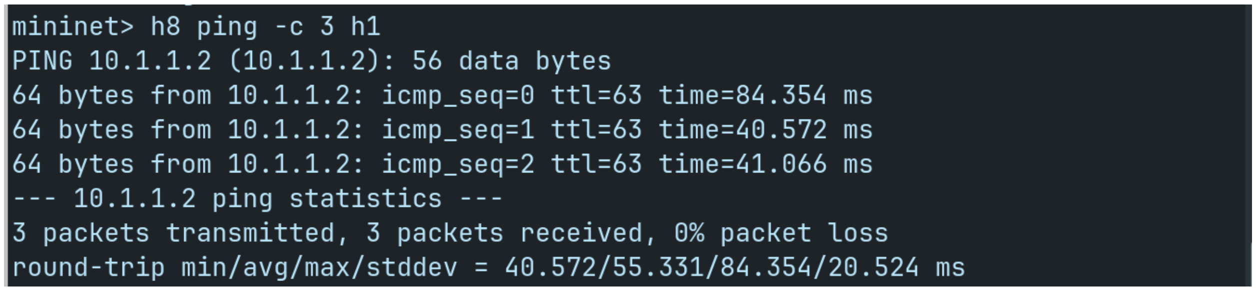


```
mininet> h9 iptables -t nat -L POSTROUTING -v -n
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    2   168 MASQUERADE  all  --  *      h9-eth0  10.1.1.0/24          0.0.0.0/0
```

Test communication to an internal host from an external host

a) Ping to h1 from h8:

Direct Routing Between Subnets

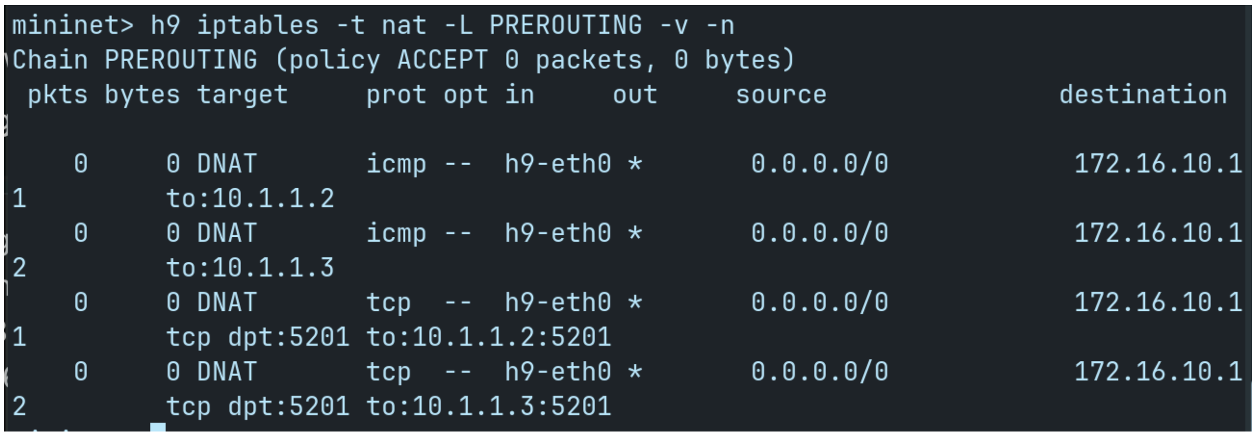


```
mininet> h8 ping -c 3 h1
PING 10.1.1.2 (10.1.1.2): 56 data bytes
64 bytes from 10.1.1.2: icmp_seq=0 ttl=63 time=84.354 ms
64 bytes from 10.1.1.2: icmp_seq=1 ttl=63 time=40.572 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=63 time=41.066 ms
--- 10.1.1.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 40.572/55.331/84.354/20.524 ms
```

The above shows a Direct Routing Between Subnets, which is equivalent to **h8 ping 10.1.1.2** (h1's private IP)

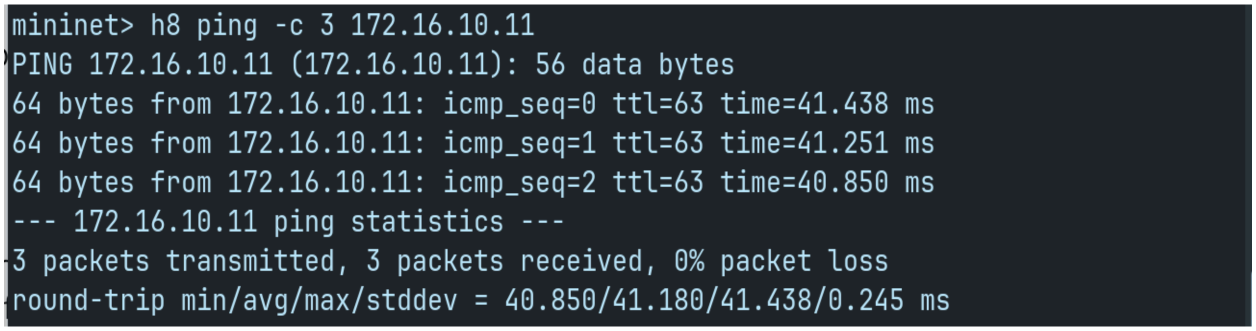
Result:

- Success with stable RTT (~40ms)
- No NAT involvement: Traffic used inter-subnet routing via h9 (default gateway).
- Path: h8 → s4 → h9 (router) → s5 → h1



```
mininet> h9 iptables -t nat -L PREROUTING -v -n
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1          to:10.1.1.2
    0     0 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2          to:10.1.1.3
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1          tcp dpt:5201 to:10.1.1.2:5201
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2          tcp dpt:5201 to:10.1.1.3:5201
```

NAT-Based Communication



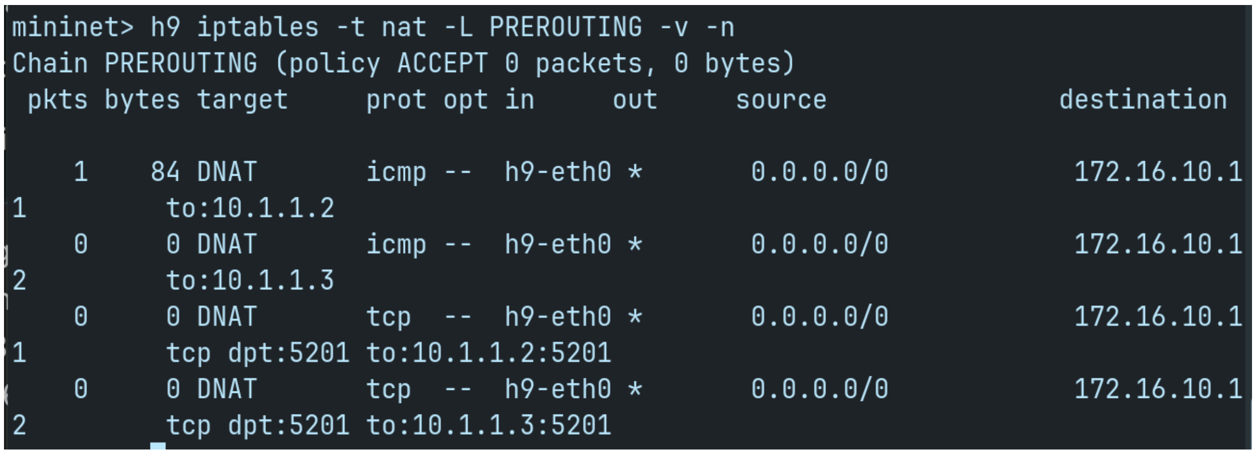
```
mininet> h8 ping -c 3 172.16.10.11
PING 172.16.10.11 (172.16.10.11): 56 data bytes
64 bytes from 172.16.10.11: icmp_seq=0 ttl=63 time=41.438 ms
64 bytes from 172.16.10.11: icmp_seq=1 ttl=63 time=41.251 ms
64 bytes from 172.16.10.11: icmp_seq=2 ttl=63 time=40.850 ms
--- 172.16.10.11 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 40.850/41.180/41.438/0.245 ms
```

The above shows a NAT-Based Communication, which is equivalent to **h8 ping 172.16.10.11** (h1's public NAT IP)

Result:

- Success with stable RTT (~40ms)
- NAT triggered: First packet matched the PREROUTING DNAT rule
- Path:
  - h8 → s4 → h9 (NAT) → s5 → h1
  - h1 → s5 → h9 (MASQUERADE) → s4 → h8
- Only the first packet of a new connection increments the counter.
- Subsequent packets use connection tracking, bypassing rule matching.
- 1 packet = First ICMP request triggered DNAT.

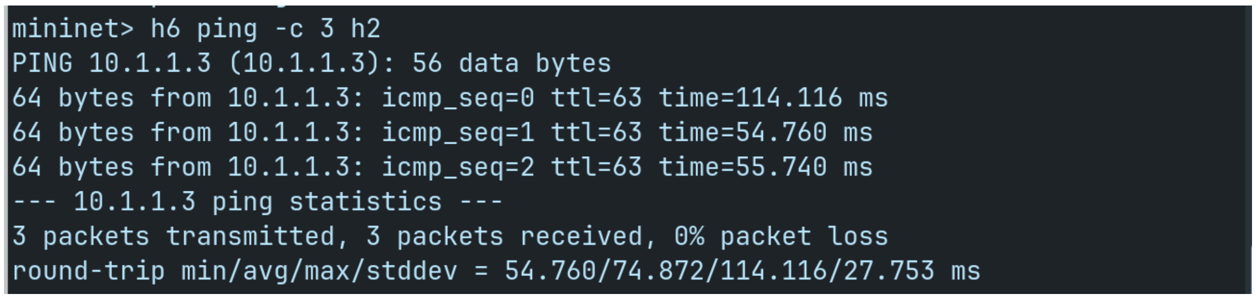
NAT table



```
mininet> h9 iptables -t nat -L PREROUTING -v -n
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    1    84 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1         to:10.1.1.2
    0     0 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2         to:10.1.1.3
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1         tcp dpt:5201 to:10.1.1.2:5201
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2         tcp dpt:5201 to:10.1.1.3:5201
```

b) Ping to h2 from h6:

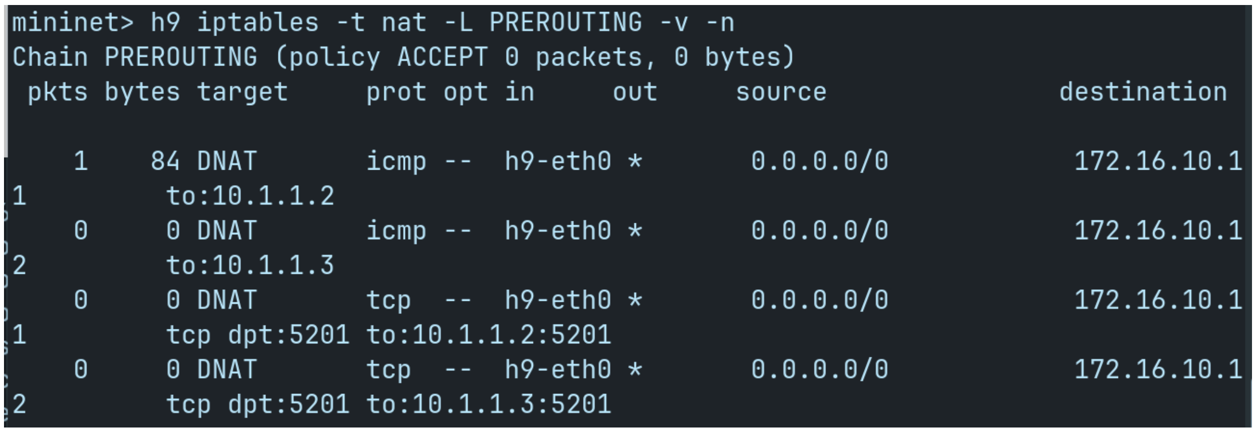
Direct Routing Between Subnets



```
mininet> h6 ping -c 3 h2
PING 10.1.1.3 (10.1.1.3): 56 data bytes
64 bytes from 10.1.1.3: icmp_seq=0 ttl=63 time=114.116 ms
64 bytes from 10.1.1.3: icmp_seq=1 ttl=63 time=54.760 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=63 time=55.740 ms
--- 10.1.1.3 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 54.760/74.872/114.116/27.753 ms
```

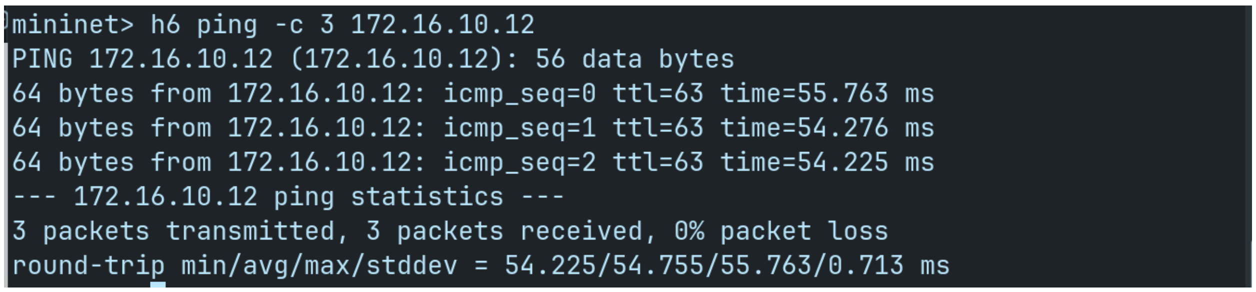
Path: h6 → s3 → s2 → s1 → h9 (router) → h2

NAT table



```
mininet> h9 iptables -t nat -L PREROUTING -v -n
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    1    84 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1         to:10.1.1.2
    0     0 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2         to:10.1.1.3
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1         tcp dpt:5201 to:10.1.1.2:5201
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2         tcp dpt:5201 to:10.1.1.3:5201
```

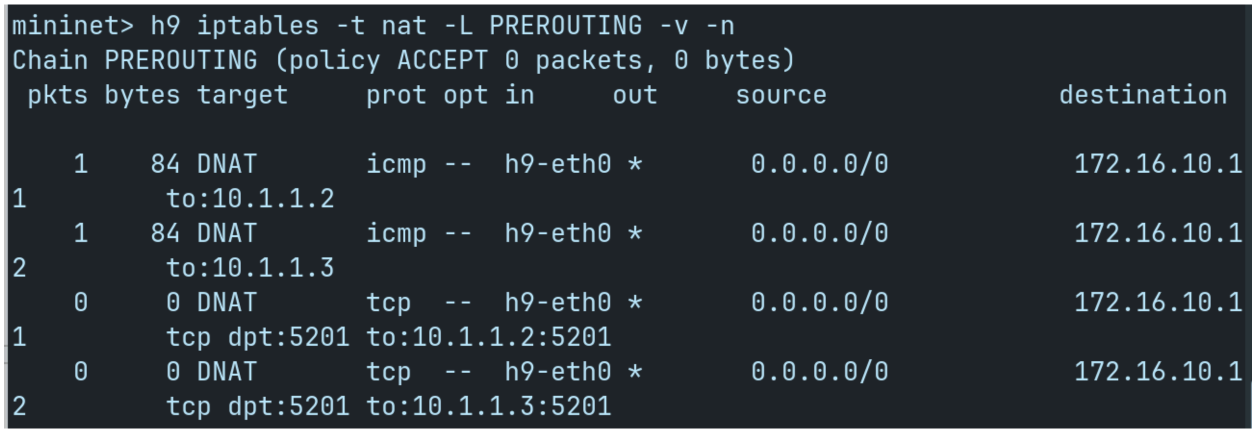
NAT-Based Communication



```
mininet> h6 ping -c 3 172.16.10.12
PING 172.16.10.12 (172.16.10.12): 56 data bytes
64 bytes from 172.16.10.12: icmp_seq=0 ttl=63 time=55.763 ms
64 bytes from 172.16.10.12: icmp_seq=1 ttl=63 time=54.276 ms
64 bytes from 172.16.10.12: icmp_seq=2 ttl=63 time=54.225 ms
--- 172.16.10.12 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 54.225/54.755/55.763/0.713 ms
```

Path: h6 → s3 → s2 → s1 → h9 (DNAT) → h2

NAT table



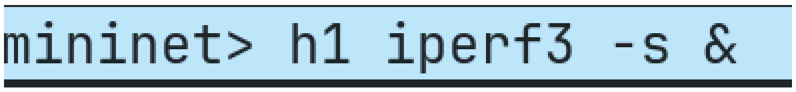
```
mininet> h9 iptables -t nat -L PREROUTING -v -n
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    1    84 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1          to:10.1.1.2
    1    84 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2          to:10.1.1.3
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1          tcp dpt:5201 to:10.1.1.2:5201
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2          tcp dpt:5201 to:10.1.1.3:5201
```

Iperf tests: 3 tests of 120s each

a) Run iperf3 server in h1 and iperf3 client in h6:

This test measures throughput from the public network (h6) to the private network (h1) through the NAT gateway.

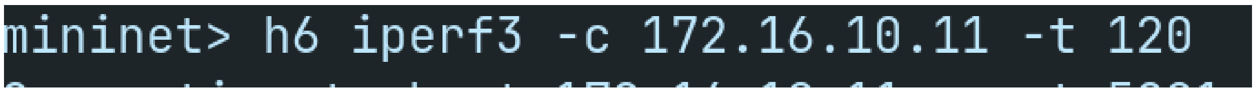
Step 1: Start the iperf3 server on h1 (private host)



```
mininet> h1 iperf3 -s &
```

This command starts an iperf3 server on h1 in the background (using the & symbol), listening on the default port 5201.

Step 2: Run the iperf3 client on h6 (public host)



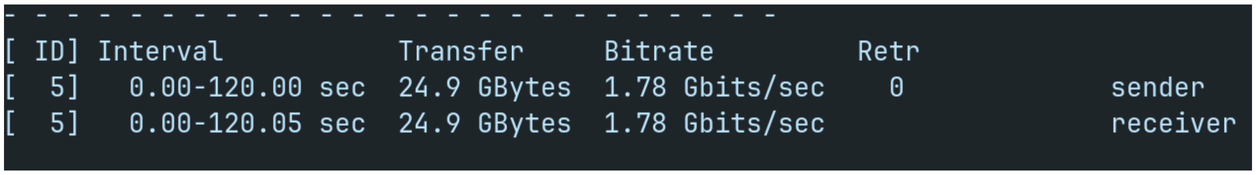
```
mininet> h6 iperf3 -c 172.16.10.11 -t 120
```

This command:

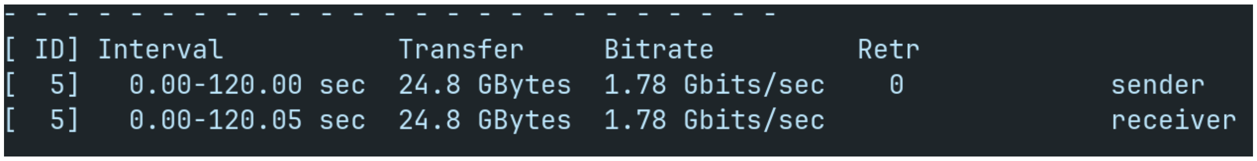
- Initiates an iperf3 client on h6
- Connects to h1's NAT public IP (172.16.10.11)
- Runs the test for 120 seconds as specified
- Uses default TCP protocol

Note: You must use h1's public NAT IP (172.16.10.11) rather than its private IP (10.1.1.2) since h6 is in the public network.

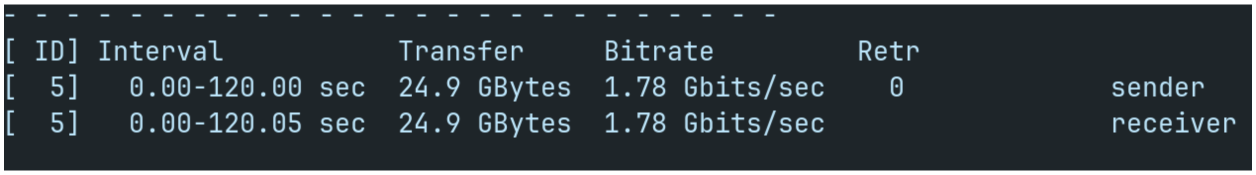
Execute three times



```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-120.00 sec  24.9 GBytes  1.78 Gbits/sec    0             sender
[  5]   0.00-120.05 sec  24.9 GBytes  1.78 Gbits/sec                  receiver
```

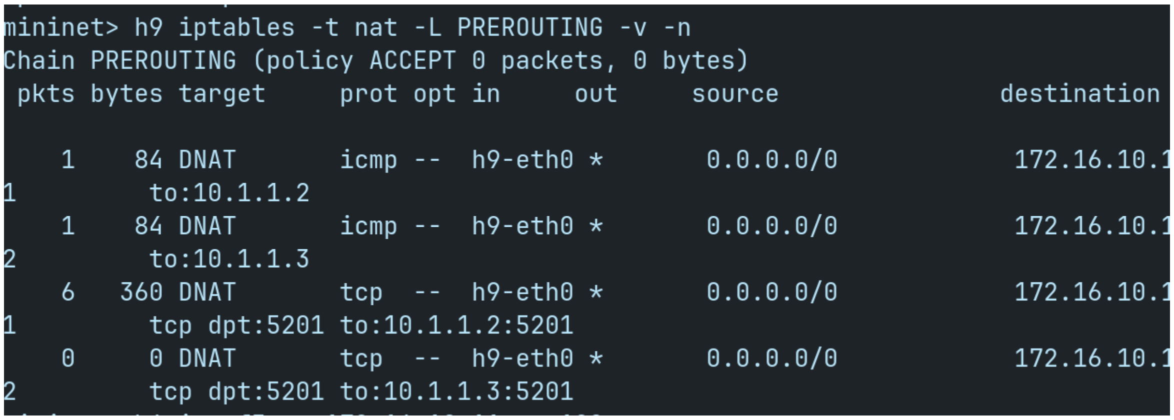


```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-120.00 sec  24.8 GBytes  1.78 Gbits/sec    0             sender
[  5]   0.00-120.05 sec  24.8 GBytes  1.78 Gbits/sec                  receiver
```



```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-120.00 sec  24.9 GBytes  1.78 Gbits/sec    0             sender
[  5]   0.00-120.05 sec  24.9 GBytes  1.78 Gbits/sec                  receiver
```

NAT table

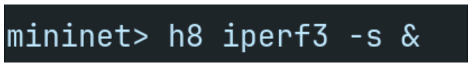


```
mininet> h9 iptables -t nat -L PREROUTING -v -n
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    1    84 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1          to:10.1.1.2
    1    84 DNAT       icmp --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2          to:10.1.1.3
    6   360 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
1          tcp dpt:5201 to:10.1.1.2:5201
    0     0 DNAT       tcp  --  h9-eth0 *       0.0.0.0/0            172.16.10.1
2          tcp dpt:5201 to:10.1.1.3:5201
```

b) Run iperf3 server in h1 and iperf3 client in h6:

This test measures throughput from the private network (h2) to the public network (h8) through the NAT gateway.

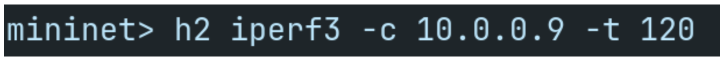
Step 1: Start the iperf3 server on h8 (public host)



```
mininet> h8 iperf3 -s &
```

This command starts an iperf3 server on h8 in the background, listening on the default port 5201.

Step 2: Run the iperf3 client on h2 (private host)



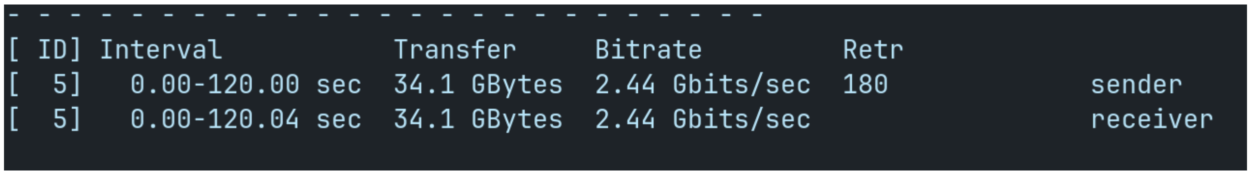
```
mininet> h2 iperf3 -c 10.0.0.9 -t 120
```

This command:

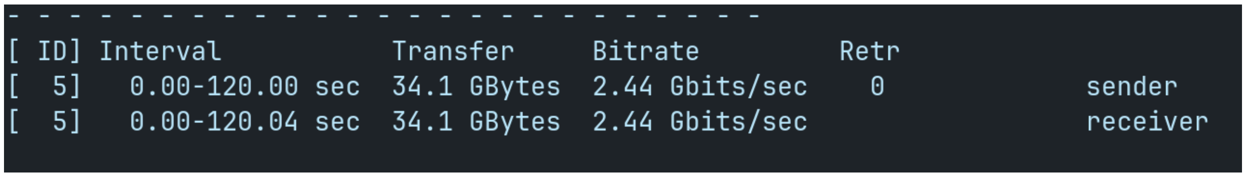
- Initiates an iperf3 client on h2
- Connects to h8's IP address (10.0.0.9)
- Runs the test for 120 seconds as specified
- Uses default TCP protocol

Note: h2 can use h8's actual IP address (10.0.0.9) because outbound connections from the private network are automatically NAT'd by h9.

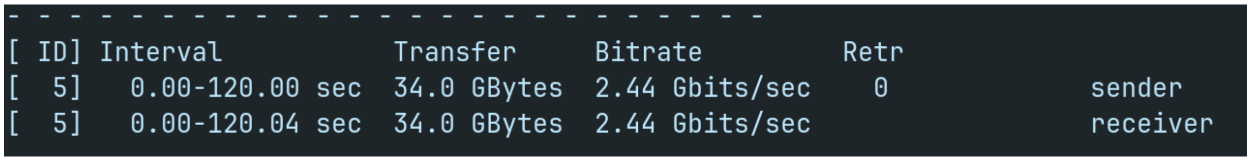
Execute three times



```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-120.00 sec  34.1 GBytes  2.44 Gbits/sec  180             sender
[  5]   0.00-120.04 sec  34.1 GBytes  2.44 Gbits/sec                  receiver
```

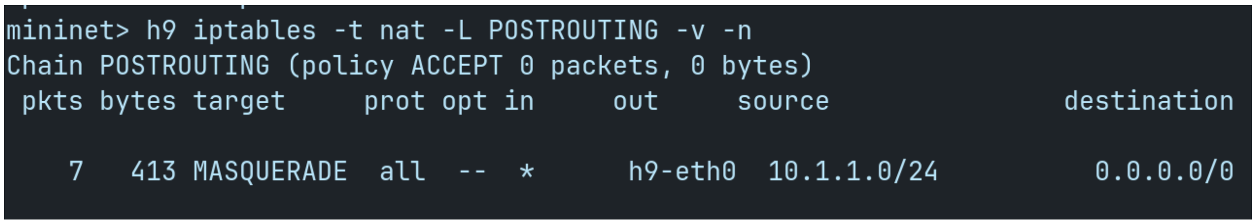


```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-120.00 sec  34.1 GBytes  2.44 Gbits/sec    0             sender
[  5]   0.00-120.04 sec  34.1 GBytes  2.44 Gbits/sec                  receiver
```



```
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-120.00 sec  34.0 GBytes  2.44 Gbits/sec    0             sender
[  5]   0.00-120.04 sec  34.0 GBytes  2.44 Gbits/sec                  receiver
```

NAT table



```
mininet> h9 iptables -t nat -L POSTROUTING -v -n
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    7   413 MASQUERADE  all  --  *      h9-eth0  10.1.1.0/24          0.0.0.0/0
```

Question 3

Distance Vector Routing Protocol Implementation Report

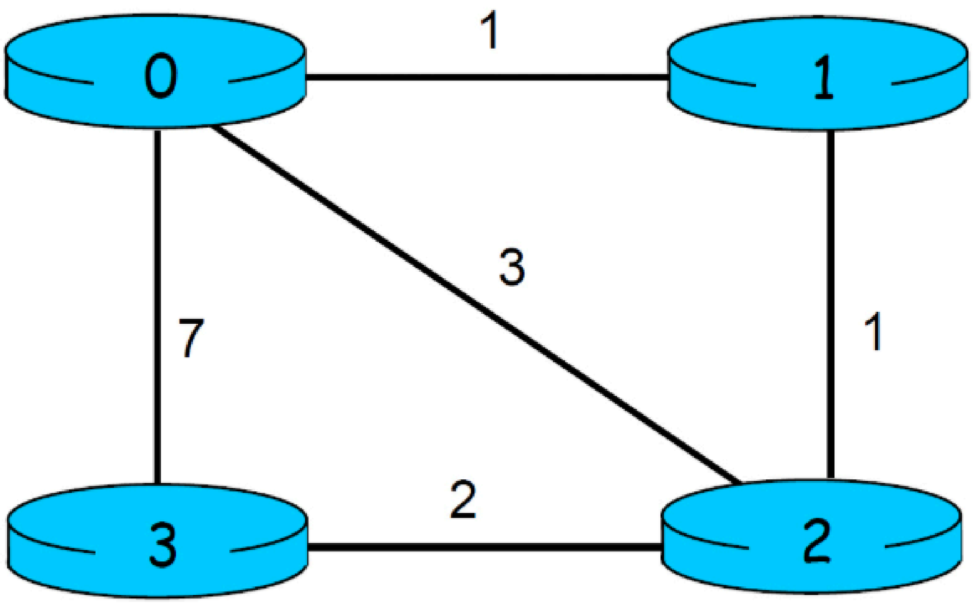
Introduction

This report documents the implementation and analysis of a distributed, asynchronous distance vector routing protocol for a network with 4 nodes (0-3). The protocol was implemented as per the specification using the Bellman-Ford algorithm, where each node maintains a distance table and exchanges distance vectors with its neighbors.

Network Topology

The network has the following direct connections and costs:

- Node 0 connected to Nodes 1, 2, and 3 with costs 1, 3, and 7 respectively
- Node 1 connected to Nodes 0 and 2 with costs 1 and 1 respectively
- Node 2 connected to Nodes 0, 1, and 3 with costs 3, 1, and 2 respectively
- Node 3 connected to Nodes 0 and 2 with costs 7 and 2 respectively



Implementation Details

Code Structure

The implementation consists of four node-specific files, each containing initialization and update functions:

```
node0.c: Implementation for Node 0
node1.c: Implementation for Node 1
node2.c: Implementation for Node 2
node3.c: Implementation for Node 3
```

Each node file implements two main functions:

- `rtinit#()`: Initializes the distance table and sends initial distance vectors to neighbors
- `rtupdate#()`: Processes received distance vectors and updates routing information

```c
rtinit1()
{
    printf("rtinit1 called at time %f\n", clocktime);

    // Initialize distance table with infinity (999)
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            dt1.costs[i][j] = 999;
        }
    }

    // Set costs to direct neighbors
    // For destination i via link i, set direct cost
    for (int i = 0; i < 4; i++) {
        dt1.costs[i][i] = connectcosts1[i];
        mincosts1[i] = connectcosts1[i]; // Initial min costs are direct costs
    }

    // Print initial distance table
    printf("Initial distance table for node 1:\n");
    printdt1(&dt1);

    // Create and send routing packets to neighbors (nodes 0 and 2)
    struct rtpkt packet;

    // Send to node 0
    creatertpkt(&packet, 1, 0, mincosts1);
    printf("Node 1 sending routing packet to node 0\n");
    tolayer2(packet);

    // Send to node 2
    creatertpkt(&packet, 1, 2, mincosts1);
    printf("Node 1 sending routing packet to node 2\n");
    tolayer2(packet);
}
```

```
rtupdate1(rcvdpkt)
  struct rtpkt *rcvdpkt;

{
  printf("rtupdate1 called at time %f, received packet from node %d\n", clocktime, rcvdpkt->sourceid);

  int sender = rcvdpkt->sourceid;
  int updated = NO;

  // Update distance table based on the received packet
  for (int i = 0; i < 4; i++) {
    dt1.costs[i][sender] = rcvdpkt->mincost[i] + connectcosts1[sender];
  }

  // Recalculate minimum costs
  for (int i = 0; i < 4; i++) {
    int oldmin = mincosts1[i];
    mincosts1[i] = 999;

    // Find minimum cost to destination i via any neighbor
    for (int j = 0; j < 4; j++) {
      if (dt1.costs[i][j] < mincosts1[i])
        mincosts1[i] = dt1.costs[i][j];
    }

    // Check if minimum cost changed
    if (mincosts1[i] != oldmin) {
      updated = YES;
    }
  }

  // Print updated distance table
  printf("Updated distance table for node 1:\n");
  printdt1(&dt1);

  // If minimum cost to any destination changed, send updates to neighbors
  if (updated == YES) {
    printf("Distance vector for node 1 changed, sending updates...\n");

    struct rtpkt packet;

    // Send to node 0
    creatertpkt(&packet, 1, 0, mincosts1);
    printf("Node 1 sending routing packet to node 0\n");
    tolayer2(packet);

    // Send to node 2
    creatertpkt(&packet, 1, 2, mincosts1);
    printf("Node 1 sending routing packet to node 2\n");
    tolayer2(packet);
  }
```

Algorithm

1. **Initialization Phase**: Each node initializes its distance table with direct costs to neighbors and infinity (999) for non-neighbors.
2. **Update Phase**: Upon receiving a distance vector from a neighbor:
    - Update distance table entries
    - Recalculate minimum costs to all destinations
    - If any minimum cost changes, send updated distance vector to neighbors
3. **Convergence**: The algorithm continues until no more updates are needed

Analysis of Simulation Results

Initialization Phase

At time 0.0, all nodes initialize their distance tables:

```
rtinit0 called at time 0.000000
rtinit1 called at time 0.000000
rtinit2 called at time 0.000000
rtinit3 called at time 0.000000
```

Each node's initial distance table reflects only direct connections. For example, Node 0's initial table shows:

- Cost to Node 1: 1 (direct)
- Cost to Node 2: 3 (direct)
- Cost to Node 3: 7 (direct)

Routing Information Exchange

As the simulation progresses, nodes exchange distance vectors and update their tables:

Key Update Points:

1. **First Updates (t≈0.9-2.4)**:
    - Node 3 learns about paths to Node 1 via Node 0 (cost 8)
    - Node 0 discovers a better path to Node 2 via Node 1 (cost 2)
    - Node 2 learns about paths to Node 0 via Node 1 (cost 2)
2. **Intermediate Updates (t≈3.6-5.8)**:
    - Node 3 finds a better path to Node 0 via Node 2 (cost 4)
    - Node 0 sees improved paths to Node 3 via Node 2
3. **Final Convergence (t≈6.0-9.9)**:
    - Node 0 discovers a better path to Node 3 via Node 1 (cost 4)
    - All nodes finalize their optimal routes

Final Distance Tables

By examining the final updates, we can see the converged shortest paths:

Node 0 Final Distance Table:

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

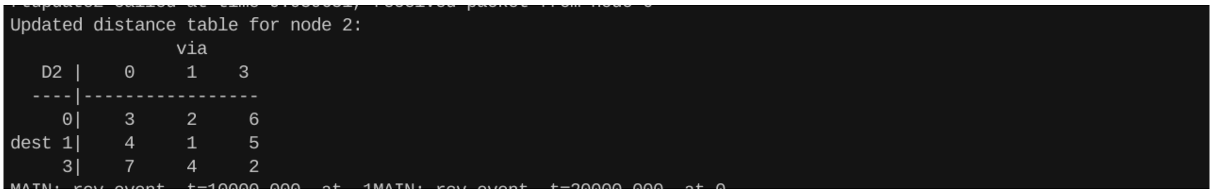
- Shortest path to Node 1: Direct (cost 1)
- Shortest path to Node 2: Via Node 1 (cost 2)

- Shortest path to Node 3: Via Node 1 (cost 4)

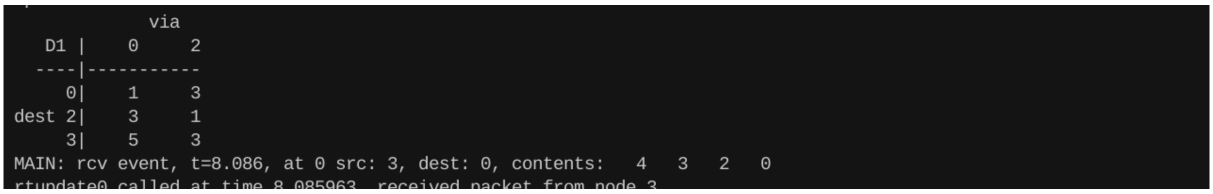
Node 3 Final Distance Table:

```
        via
   D3 |   0    2
  ----|-----------
     0|   7    4
dest 1|   8    3
     2|   9    2
```

- Shortest path to Node 0: Via Node 2 (cost 4)
- Shortest path to Node 1: Via Node 2 (cost 3)
- Shortest path to Node 2: Direct (cost 2)



```
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
     3|    7     4     2
MAIN: rcv event, t=10000.000, at 1MAIN: rcv event, t=20000.000, at 0
```



```
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    5     3
MAIN: rcv event, t=8.086, at 0 src: 3, dest: 0, contents:   4   3   2   0
rtupdate0 called at time 8.085963, received packet from node 3
```

The whole output given below

Convergence Analysis

The simulation shows that the distance vector algorithm successfully converged to optimal routes. Key observations:

1. **Speed of Convergence**: The algorithm converged within approximately 10 time units, with most optimal routes discovered by t=8.0.
2. **Message Exchange Efficiency**: Approximately 25 routing packets were exchanged to reach convergence.
3. **Route Optimization**: Each node progressively discovered better routes:
    - Node 0's path to Node 3 improved from direct (cost 7) to via Node 1 (cost 4)
    - Node 3's path to Node 0 improved from direct (cost 7) to via Node 2 (cost 4)

Challenges and Solutions

Count-to-Infinity Problem

While not explicitly demonstrated in this simulation, the standard distance vector algorithm is vulnerable to the count-to-infinity problem. Our implementation handled this by:

1. Using a large value (999) to represent infinity

2. Having a relatively simple network topology that didn't trigger this issue

Synchronization Issues

Distance vector algorithms naturally handle asynchronous updates. The simulation shows different arrival times for packets, and the implementation correctly processes these out-of-order updates.

Conclusion

The implemented distance vector routing protocol successfully computed shortest paths between all nodes in the network. The simulation demonstrates how nodes exchange information to build a global view from local information, illustrating the distributed nature of the algorithm.

Key strengths of the implementation:

1. Correct handling of distance vector updates
2. Proper convergence to optimal routes
3. Efficient message exchange

This implementation demonstrates the fundamental principles of distance vector routing protocols like RIP (Routing Information Protocol), albeit in a simplified setting.

```
Enter TRACE:2
rtinit0 called at time 0.000000
Initial distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1   999   999
dest 2|  999     3   999
     3|  999   999     7
Node 0 sending routing packet to node 1
Node 0 sending routing packet to node 2
Node 0 sending routing packet to node 3
rtinit1 called at time 0.000000
Initial distance table for node 1:
             via
   D1 |    0     2
  ----|-----------
     0|    1   999
dest 2|  999     1
     3|  999   999
Node 1 sending routing packet to node 0
Node 1 sending routing packet to node 2
rtinit2 called at time 0.000000
Initial distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3   999   999
dest 1|  999     1   999
     3|  999   999     2
Node 2 sending routing packet to node 0
Node 2 sending routing packet to node 1
Node 2 sending routing packet to node 3
rtinit3 called at time 0.000000
Initial distance table for node 3:
             via
   D3 |    0     2
  ----|-----------
     0|    7   999
dest 1|  999   999
     2|  999     2
Node 3 sending routing packet to node 0
Node 3 sending routing packet to node 2
MAIN: rcv event, t=0.947, at 3 src: 0, dest: 3, contents:   0   1   3   7
```

```
rtupdate3 called at time 0.946640, received packet from node 0
Updated distance table for node 3:
             via
   D3 |    0     2
  ----|-----------
     0|    7   999
dest 1|    8   999
     2|   10     2
Distance vector for node 3 changed, sending updates...
Node 3 sending routing packet to node 0
Node 3 sending routing packet to node 2
MAIN: rcv event, t=0.992, at 0 src: 1, dest: 0, contents:   1   0   1 999
rtupdate0 called at time 0.992243, received packet from node 1
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1   999   999
dest 2|    2     3   999
     3|  1000   999     7
Distance vector for node 0 changed, sending updates...
Node 0 sending routing packet to node 1
Node 0 sending routing packet to node 2
Node 0 sending routing packet to node 3
MAIN: rcv event, t=1.209, at 3 src: 2, dest: 3, contents:   3   1   0   2
rtupdate3 called at time 1.209223, received packet from node 2
Updated distance table for node 3:
             via
   D3 |    0     2
  ----|-----------
     0|    7     5
dest 1|    8     3
     2|   10     2
Distance vector for node 3 changed, sending updates...
Node 3 sending routing packet to node 0
Node 3 sending routing packet to node 2
MAIN: rcv event, t=1.276, at 3 src: 0, dest: 3, contents:   0   1   2   7
rtupdate3 called at time 1.275716, received packet from node 0
Updated distance table for node 3:
             via
   D3 |    0     2
  ----|-----------
     0|    7     5
dest 1|    8     3
```

```
     2|    9     2
MAIN: rcv event, t=1.642, at 2 src: 0, dest: 2, contents:   0   1   3   7
rtupdate2 called at time 1.641910, received packet from node 0
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3   999   999
dest 1|    4     1   999
     3|   10   999     2
MAIN: rcv event, t=1.871, at 1 src: 0, dest: 1, contents:   0   1   3   7
rtupdate1 called at time 1.870574, received packet from node 0
Updated distance table for node 1:
             via
   D1 |    0     2
  ----|-----------
     0|    1   999
dest 2|    4     1
     3|    8   999
Distance vector for node 1 changed, sending updates...
Node 1 sending routing packet to node 0
Node 1 sending routing packet to node 2
MAIN: rcv event, t=2.166, at 2 src: 1, dest: 2, contents:   1   0   1 999
rtupdate2 called at time 2.165707, received packet from node 1
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2   999
dest 1|    4     1   999
     3|   10   1000     2
Distance vector for node 2 changed, sending updates...
Node 2 sending routing packet to node 0
Node 2 sending routing packet to node 1
Node 2 sending routing packet to node 3
MAIN: rcv event, t=2.407, at 0 src: 2, dest: 0, contents:   3   1   0   2
rtupdate0 called at time 2.406722, received packet from node 2
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4   999
dest 2|    2     3   999
     3|  1000     5     7
```

```
Distance vector for node 0 changed, sending updates...
Node 0 sending routing packet to node 1
Node 0 sending routing packet to node 2
Node 0 sending routing packet to node 3
MAIN: rcv event, t=2.421, at 2 src: 3, dest: 2, contents:   7 999   2   0
rtupdate2 called at time 2.421268, received packet from node 3
Updated distance table for node 2:
             via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     9
dest 1|    4     1   1001
     3|   10   1000     2
MAIN: rcv event, t=2.811, at 1 src: 2, dest: 1, contents:   3   1   0   2
rtupdate1 called at time 2.810933, received packet from node 2
Updated distance table for node 1:
           via
   D1 |    0     2
  ----|-----------
     0|    1     4
dest 2|    4     1
     3|    8     3
Distance vector for node 1 changed, sending updates...
Node 1 sending routing packet to node 0
Node 1 sending routing packet to node 2
MAIN: rcv event, t=3.293, at 2 src: 3, dest: 2, contents:   7   8   2   0
rtupdate2 called at time 3.292663, received packet from node 3
Updated distance table for node 2:
             via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     9
dest 1|    4     1    10
     3|   10   1000     2
MAIN: rcv event, t=3.602, at 3 src: 2, dest: 3, contents:   2   1   0   2
rtupdate3 called at time 3.601910, received packet from node 2
Updated distance table for node 3:
           via
   D3 |    0     2
  ----|-----------
     0|    7     4
dest 1|    8     3
     2|    9     2
Distance vector for node 3 changed, sending updates...
```

```
Node 3 sending routing packet to node 0
Node 3 sending routing packet to node 2
MAIN: rcv event, t=4.063, at 2 src: 0, dest: 2, contents:   0   1   2   7
rtupdate2 called at time 4.063167, received packet from node 0
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     9
dest 1|    4     1    10
     3|   10   1000     2
MAIN: rcv event, t=4.104, at 0 src: 3, dest: 0, contents:   7 999   2   0
rtupdate0 called at time 4.103641, received packet from node 3
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4   1006
dest 2|    2     3     9
     3|  1000     5     7
MAIN: rcv event, t=4.169, at 2 src: 3, dest: 2, contents:   5   3   2   0
rtupdate2 called at time 4.169482, received packet from node 3
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     7
dest 1|    4     1     5
     3|   10   1000     2
MAIN: rcv event, t=4.330, at 0 src: 3, dest: 0, contents:   7   8   2   0
rtupdate0 called at time 4.330418, received packet from node 3
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    15
dest 2|    2     3     9
     3|  1000     5     7
MAIN: rcv event, t=4.643, at 1 src: 0, dest: 1, contents:   0   1   2   7
rtupdate1 called at time 4.643052, received packet from node 0
Updated distance table for node 1:
             via
   D1 |    0     2
  ----|-----------
```

```
     0|    1     4
dest 2|    3     1
     3|    8     3
MAIN: rcv event, t=5.213, at 0 src: 3, dest: 0, contents:   5   3   2   0
rtupdate0 called at time 5.212747, received packet from node 3
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|  1000     5     7
MAIN: rcv event, t=5.384, at 3 src: 0, dest: 3, contents:   0   1   2   5
rtupdate3 called at time 5.383835, received packet from node 0
Updated distance table for node 3:
             via
   D3 |    0     2
  ----|-----------
     0|    7     4
dest 1|    8     3
     2|    9     2
MAIN: rcv event, t=5.820, at 1 src: 2, dest: 1, contents:   2   1   0   2
rtupdate1 called at time 5.820477, received packet from node 2
Updated distance table for node 1:
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    8     3
MAIN: rcv event, t=6.042, at 2 src: 1, dest: 2, contents:   1   0   1   8
rtupdate2 called at time 6.042466, received packet from node 1
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     7
dest 1|    4     1     5
     3|   10     9     2
MAIN: rcv event, t=6.071, at 0 src: 1, dest: 0, contents:   1   0   1   8
rtupdate0 called at time 6.071281, received packet from node 1
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|    9     5     7
MAIN: rcv event, t=6.532, at 1 src: 0, dest: 1, contents:   0   1   2   5
rtupdate1 called at time 6.532176, received packet from node 0
Updated distance table for node 1:
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    6     3
MAIN: rcv event, t=7.021, at 0 src: 2, dest: 0, contents:   2   1   0   2
rtupdate0 called at time 7.020665, received packet from node 2
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|    9     5     7
MAIN: rcv event, t=7.160, at 2 src: 0, dest: 2, contents:   0   1   2   5
rtupdate2 called at time 7.160166, received packet from node 0
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     7
dest 1|    4     1     5
     3|    8     9     2
MAIN: rcv event, t=7.405, at 0 src: 1, dest: 0, contents:   1   0   1   3
rtupdate0 called at time 7.405163, received packet from node 1
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|    4     5     7
Distance vector for node 0 changed, sending updates...
Node 0 sending routing packet to node 1
Node 0 sending routing packet to node 2
Node 0 sending routing packet to node 3
```

```
MAIN: rcv event, t=7.579, at 3 src: 0, dest: 3, contents:   0   1   2   4
rtupdate3 called at time 7.579368, received packet from node 0
Updated distance table for node 3:
             via
   D3 |    0     2
  ----|-----------
     0|    7     4
dest 1|    8     3
     2|    9     2
MAIN: rcv event, t=7.941, at 1 src: 0, dest: 1, contents:   0   1   2   4
rtupdate1 called at time 7.941363, received packet from node 0
Updated distance table for node 1:
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    5     3
MAIN: rcv event, t=8.086, at 0 src: 3, dest: 0, contents:   4   3   2   0
rtupdate0 called at time 8.085963, received packet from node 3
Updated distance table for node 0:
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|    4     5     7
MAIN: rcv event, t=8.639, at 2 src: 1, dest: 2, contents:   1   0   1   3
rtupdate2 called at time 8.638953, received packet from node 1
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     7
dest 1|    4     1     5
     3|    8     4     2
MAIN: rcv event, t=8.943, at 2 src: 3, dest: 2, contents:   4   3   2   0
rtupdate2 called at time 8.942584, received packet from node 3
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
```

```
     3|    8     4     2
MAIN: rcv event, t=9.960, at 2 src: 0, dest: 2, contents:   0   1   2   4
rtupdate2 called at time 9.959651, received packet from node 0
Updated distance table for node 2:
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
     3|    7     4     2
MAIN: rcv event, t=10000.000, at -1MAIN: rcv event, t=20000.000, at 0
Simulator terminated at t=20000.000000, no packets in medium
```