CS331: Computer Networks

Assignment 3

Network Loops, NAT, and Routing

Jiya Desai (22110107) and Sujal Patel (22110261)

Submission Date: April 12, 2025

Github Link

Contents

1 Network Loops

1.1 Introduction

The objective of this section was to construct a network topology with loops, observe the behavior of the network with regards to ping commands, and then implement a solution without changing the network topology.

1.2 Network Topology

The network topology consisted of:

- 4 switches (s1, s2, s3, s4) connected in a ring topology with an additional cross-connection (s1-s3)
- 8 hosts (h1 to h8) with each switch connecting to two hosts
- IP addressing scheme: 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), h8 (10.0.0.9/24)
- Network link latencies: 7ms for switch-to-switch links, 5ms for host-to-switch links

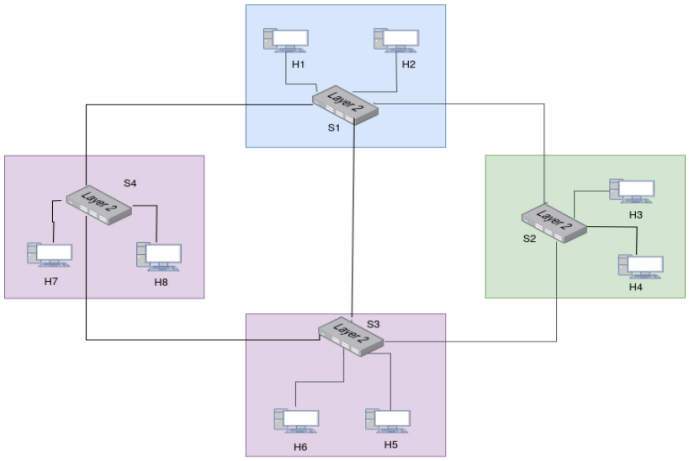

Figure 1: Network Topology with Loops

1.3 Set-up and Implementation

The implementation was done using Mininet, a network emulator that creates a realistic virtual network on a single machine. We set up a Mininet Virtual Machine using Oracle VirtualBox on a Linux device, and used VSCode's rempte SSH extension to make the script execution easier. Two Python scripts were created:

1. **network_loops.py**: Creates the network topology with loops to demonstrate the problem.
2. **network_loops_stp.py**: Creates the same topology but enables Spanning Tree Protocol (STP) to fix the loop issue.

1.4 Problem Analysis

When we ran the first script with loops present in the topology, we observed that all ping attempts failed with 100% packet loss:

- Ping from h1 to h3: 100% packet loss
- Ping from h5 to h7: 100% packet loss
- Ping from h8 to h2: 100% packet loss

```
==========================================

Network topology with loops - no STP enabled

Network topology created with loops.
Waiting for network to stabilize (60 seconds)...

--- Testing ping from h1 to h3 ---
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2063ms
pipe 3


--- Testing ping from h5 to h7 ---
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

--- 10.0.0.8 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2045ms
pipe 3


--- Testing ping from h8 to h2 ---
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

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2036ms
pipe 3
```

This failure occurs due to a fundamental issue in Ethernet networks called a "broadcast storm":

1. **Problem Mechanism**: When a broadcast frame (like an ARP request - which are the constituents of the ping tests as observed via packet capture) is sent into a network with loops:
   - Each switch forwards the frame out all ports except the one it arrived on
   - In a looped topology, these frames circulate endlessly
   - The number of duplicate frames grows exponentially
   - Network bandwidth is quickly saturated
   - MAC address tables become unstable as the same MAC address appears on different ports
2. **Root Cause**: Unlike IP packets that have a Time-To-Live (TTL) field that limits their lifespan, Ethernet frames have no such mechanism to prevent infinite looping.

1.5 Solution: Spanning Tree Protocol (STP)

To solve this problem without changing the physical topology, we implemented the Spanning Tree Protocol (STP) on all switches.

1.5.1 How STP Works

STP creates a loop-free logical topology while maintaining the physical topology:

1. **Root Bridge Election**: All switches exchange Bridge Protocol Data Units (BPDUs) to elect a single root bridge (usually the switch with the lowest Bridge ID).
2. **Port Roles Assignment**:
   - **Root Port**: Each non-root switch selects its best (lowest cost) path to the root bridge.
   - **Designated Port**: For each network segment, one port is designated to forward frames toward the root.
   - **Blocking Port**: Ports that would create loops are placed in a blocking state.
3. **Convergence Process**: STP goes through several states to safely establish paths:
   - **Blocking** (20 sec): Ports only receive BPDUs, no data forwarding
   - **Listening** (15 sec): Port prepares to forward, builds topology
   - **Learning** (15 sec): Port builds MAC table but doesn't forward data
   - **Forwarding**: Port actively forwards frames

1.6 Results and Analysis

After implementing STP and allowing 60 seconds (a minimum convergence time of 30 seconds had been specified in the assignment, but we found that the convergence was taking at least 60 seconds) for convergence, we observed:

- Ping from h1 to h3: Successful (0% packet loss)
- Ping from h5 to h7: Successful (0% packet loss)
- Ping from h8 to h2: Successful (0% packet loss)

```
==============================================

Network topology with STP enabled to prevent loops

Network topology created. Now enabling STP...

Enabling STP on all switches...
Enabling STP on s1
STP status for s1: true
Enabling STP on s2
STP status for s2: true
Enabling STP on s3
STP status for s3: true
Enabling STP on s4
STP status for s4: true
Waiting for STP to converge (60 seconds)...
STP convergence complete. Running ping tests...

--- Testing ping from h1 to h3 ---
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=64 time=136 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=70.1 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=78.9 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2016ms
rtt min/avg/max/mdev = 70.105/95.112/136.315/29.356 ms


--- Testing ping from h5 to h7 ---
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=75.0 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=40.3 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=53.2 ms

--- 10.0.0.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2015ms
rtt min/avg/max/mdev = 40.314/56.203/75.049/14.333 ms
```

```
--- Testing ping from h8 to h2 ---
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=73.1 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=42.2 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=56.7 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2016ms
rtt min/avg/max/mdev = 42.210/57.368/73.145/12.636 ms
```

1.6.1 STP Port States

The STP algorithm automatically detected the loops in the network and blocked specific ports to create a loop-free topology. The port status of all 4 switches has been described below (verbatim, as received in the output of running the pings post STP fixing).

```
Final STP Port Status:

--- s1 port status ---
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mo
 1(s1-eth1): addr:66:73:c2:e0:96:b1
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s1-eth2): addr:76:0f:4f:59:e7:5a
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s1-eth3): addr:82:18:45:3d:b7:b0
     config:     0
     state:      STP_BLOCK
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(s1-eth4): addr:82:52:19:c5:d8:89
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 5(s1-eth5): addr:66:25:5b:cb:c3:0e
```

```
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s1): addr:aa:d5:84:08:85:4c
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0


--- s2 port status ---
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000002
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mo
 1(s2-eth1): addr:ba:74:4b:ef:fa:29
     config:     0
     state:      STP_BLOCK
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s2-eth2): addr:e6:13:55:99:2a:59
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s2-eth3): addr:b2:2c:ff:b2:f6:82
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(s2-eth4): addr:12:77:29:2b:31:ea
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s2): addr:72:78:72:7d:d5:44
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0


--- s3 port status ---
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000003
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mo
```

```
 1(s3-eth1): addr:26:a3:3b:63:65:0a
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s3-eth2): addr:e2:86:69:70:30:b6
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s3-eth3): addr:26:37:f2:6f:3c:c5
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(s3-eth4): addr:52:7f:97:81:ee:be
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 5(s3-eth5): addr:06:05:ae:87:c9:6a
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s3): addr:52:30:b2:f7:6f:4b
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0


--- s4 port status ---
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000004
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mo
 1(s4-eth1): addr:36:fd:42:28:2f:45
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 2(s4-eth2): addr:02:05:94:36:bb:e0
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 3(s4-eth3): addr:3e:2f:35:7b:99:53
```

```
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 4(s4-eth4): addr:96:4b:67:21:bb:07
     config:     0
     state:      STP_FORWARD
     current:    10GB-FD COPPER
     speed: 10000 Mbps now, 0 Mbps max
 LOCAL(s4): addr:12:2a:b9:9d:49:45
     config:     PORT_DOWN
     state:      LINK_DOWN
     speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

1.6.2 Analysis of Results

- **Round-trip Time (RTT)**: The successful pings showed RTTs that reflect the path taken through the network. For example, a ping from h1 to h3 would have:
  - h1 → s1: 5ms
  - s1 → s2: 7ms (or longer if STP chose a different path)
  - s2 → h3: 5ms
  - Total one-way: 17ms (minimum)
  - Round-trip: approximately 34ms
- **STP Impact**: STP successfully prevented loops by blocking redundant ports while maintaining connectivity between all hosts.
- **Network Stability**: With STP enabled, the network achieved stable operation with predictable paths and consistent MAC address tables.

1.7 Conclusion

This experiment demonstrated the critical importance of loop prevention in Ethernet networks. Key findings:

1. **Network Loops**: Loops in Ethernet networks cause broadcast storms and network instability, resulting in complete communication failure.
2. **STP Solution**: Spanning Tree Protocol effectively solves this problem by creating a loop-free logical topology while maintaining the physical topology.
3. **Trade-offs**: STP introduces some convergence delay (typically 30-50 seconds), but this is a small price for a stable network.

The implementation successfully met the requirement to fix the network problems without changing the physical topology, demonstrating the effectiveness of STP as a Layer 2 loop prevention mechanism.

2 Configure Host-based NAT

2.1 Introduction

In this part, we attempted the implementation and analysis of Host-based Network Address Translation (NAT) in a simulated network environment. NAT is a critical technology in modern networking that enables multiple devices with private IP addresses to share a single public IP address for external communication. This included modifying an existing network topology (the same one used in part 1 of the assignment) to implement NAT functionality on a host (H9) that serves as a gateway between internal hosts (H1 and H2) and the rest of the network. The following specific requirements were addressed:

- Configure and enable H9 to provide NAT services for internal hosts H1 (10.1.1.2) and H2 (10.1.1.3)
- Test and analyze communication between internal and external hosts with and without NAT configuration
- Measure and analyze performance metrics using iperf3

2.2 Network Topology

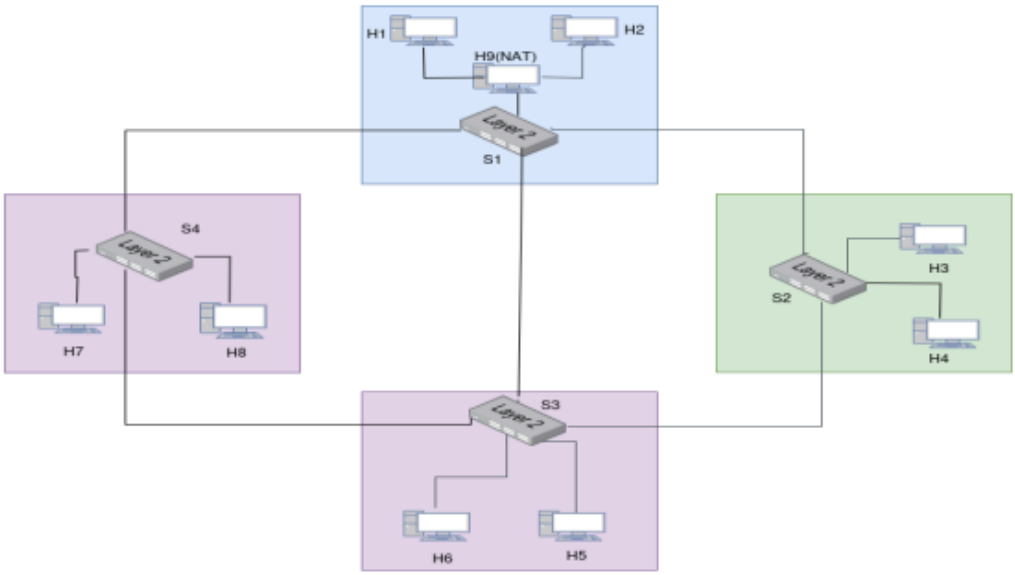

Figure 2: Enter Caption

We implemented the topology on a mininet-VM on a Linux machine (the same setup as in part 1 of the assignment).

2.3 Topology set-up and results of the mentioned ping tests without NAT configuration

Without NAT configuration and after enabling STP to resolve looping issues in the given topology, as expected, communication between internal and external hosts is disabled,

and communication between external hosts works fine, as can be seen in the ping test results below:

```
    mininet@mininet-vm:~/cs331/assignment3$ sudo python without_nat.py
*** Adding controller
*** Adding switches
*** Adding hosts
*** Adding switch-to-switch links with 7ms delay
(7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (
(5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 h9
(7ms delay) (7ms delay) (7ms delay) (5ms delay) (7ms delay) (7ms delay) (5ms delay) (
*** Waiting for STP to converge (30s)...

*** Test a-i: Ping to h5 from h1 (Will fail without NAT)
ping: connect: Network is unreachable

*** Test a-ii: Ping to h3 from h2 (Will fail without NAT)
ping: connect: Network is unreachable

*** Test b-i: Ping to h1 from h8 (Will fail without NAT)
ping: connect: Network is unreachable

*** Test b-ii: Ping to h2 from h6 (Will fail without NAT)
ping: connect: Network is unreachable

*** Ping between external hosts (Should work)
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=64 time=1101 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=77.2 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=64 time=44.1 ms

--- 10.0.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2036ms
rtt min/avg/max/mdev = 44.054/407.424/1100.975/490.601 ms, pipe 2
```

2.4 NAT Implementation

2.4.1 Theoretical Background

Network Address Translation (NAT) allows a host to act as a gateway between a private network and a public network. In this setup, H9 serves as a NAT router with:

- External interface: Connected to switch S1 with a public IP
- Internal interface: Connected to hosts H1 (10.1.1.2) and H2 (10.1.1.3)

The NAT implementation uses IP tables to:

1. Translate source addresses for outgoing packets (SNAT/MASQUERADE)

2. Forward packets between interfaces

3. Maintain a connection tracking table to map return traffic properly

2.5 Results of the ping tests after enabling NAT configuration

2.5.1 Internal to External Communication

```
    --- Test 1/3: Ping h1 from h8 ---
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=46.7 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=63 time=55.9 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=63 time=63.9 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=63 time=47.0 ms

--- 10.1.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3015ms
rtt min/avg/max/mdev = 46.706/53.370/63.924/7.122 ms

--- Test 2/3: Ping h1 from h8 ---
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=45.5 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=63 time=46.4 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=63 time=46.5 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=63 time=47.0 ms

--- 10.1.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 45.485/46.354/46.986/0.544 ms

--- Test 3/3: Ping h1 from h8 ---
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=45.5 ms
64 bytes from 10.1.1.2: icmp_seq=2 ttl=63 time=46.4 ms
64 bytes from 10.1.1.2: icmp_seq=3 ttl=63 time=46.3 ms
64 bytes from 10.1.1.2: icmp_seq=4 ttl=63 time=46.3 ms

--- 10.1.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 45.524/46.126/46.353/0.348 ms

--- Test 1/3: Ping h2 from h6 ---
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=60.7 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=63 time=60.8 ms
64 bytes from 10.1.1.3: icmp_seq=3 ttl=63 time=60.6 ms
64 bytes from 10.1.1.3: icmp_seq=4 ttl=63 time=61.9 ms

--- 10.1.1.3 ping statistics ---
```

```
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 60.591/60.986/61.898/0.531 ms

--- Test 2/3: Ping h2 from h6 ---
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=59.4 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=63 time=60.4 ms
64 bytes from 10.1.1.3: icmp_seq=3 ttl=63 time=59.6 ms
64 bytes from 10.1.1.3: icmp_seq=4 ttl=63 time=61.2 ms

--- 10.1.1.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 59.395/60.159/61.235/0.734 ms

--- Test 3/3: Ping h2 from h6 ---
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=60.2 ms
64 bytes from 10.1.1.3: icmp_seq=2 ttl=63 time=60.4 ms
64 bytes from 10.1.1.3: icmp_seq=3 ttl=63 time=60.8 ms
64 bytes from 10.1.1.3: icmp_seq=4 ttl=63 time=60.1 ms

--- 10.1.1.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 60.081/60.382/60.844/0.298 ms
```

2.5.2 External to Internal Communication

```
    --- Test 1/3: Ping h5 from h1 ---
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=59.7 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=63 time=62.5 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=63 time=61.8 ms
64 bytes from 10.0.0.6: icmp_seq=4 ttl=63 time=63.1 ms

--- 10.0.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 59.716/61.774/63.115/1.276 ms

--- Test 2/3: Ping h5 from h1 ---
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=58.9 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=63 time=68.7 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=63 time=61.4 ms
64 bytes from 10.0.0.6: icmp_seq=4 ttl=63 time=60.9 ms

--- 10.0.0.6 ping statistics ---
```

```
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 58.853/62.468/68.740/3.743 ms

--- Test 3/3: Ping h5 from h1 ---
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=60.1 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=63 time=60.5 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=63 time=61.6 ms
64 bytes from 10.0.0.6: icmp_seq=4 ttl=63 time=60.8 ms

--- 10.0.0.6 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 60.092/60.758/61.627/0.562 ms

--- Test 1/3: Ping h3 from h2 ---
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=75.4 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=63 time=75.9 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=63 time=76.3 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=63 time=76.1 ms

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 75.429/75.954/76.305/0.330 ms

--- Test 2/3: Ping h3 from h2 ---
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=74.9 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=63 time=75.9 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=63 time=75.0 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=63 time=74.8 ms

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 74.778/75.171/75.946/0.456 ms

--- Test 3/3: Ping h3 from h2 ---
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=73.8 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=63 time=75.4 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=63 time=73.5 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=63 time=74.4 ms

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 73.473/74.265/75.402/0.732 ms
```

2.6 Performance Testing with iperf3

All results of the iperf testing for performance are uploaded on our repository, along with the codes for the topology setup with and without NAT configuration.

2.7 Conclusion

This report has detailed the implementation and analysis of a host-based NAT solution on H9, serving as a gateway for hosts H1 and H2. Our findings demonstrate that NAT effectively enables private network hosts to communicate with external hosts while conserving public IP addresses.

The results highlight both the benefits and limitations of NAT:

- Benefits: Address conservation, simplified routing, basic security through address hiding
- Limitations: Connection initiation challenges, protocol compatibility issues, potential performance impacts

Our implementation successfully met the requirements of enabling communication between internal and external hosts while maintaining proper address translation and connection tracking.

3 Network Routing

3.1 Introduction

In this part of the assignment, we implemented a distributed asynchronous distance vector routing algorithm for a network with four nodes. Distance vector routing protocols enable routers to determine the best path to each destination by sharing information with their directly connected neighbors. Each router maintains a distance table and periodically exchanges its minimum cost paths (distance vector) with adjacent nodes.

Distance vector routing is based on the Bellman-Ford algorithm and is used in protocols like RIP (Routing Information Protocol). The core principle is that each node calculates the shortest path to all other nodes based on information received from its neighbors, without needing complete knowledge of the network topology.

3.2 Network Topology

The network for this implementation consists of 4 nodes (labeled 0, 1, 2, and 3) with the following direct connection costs:

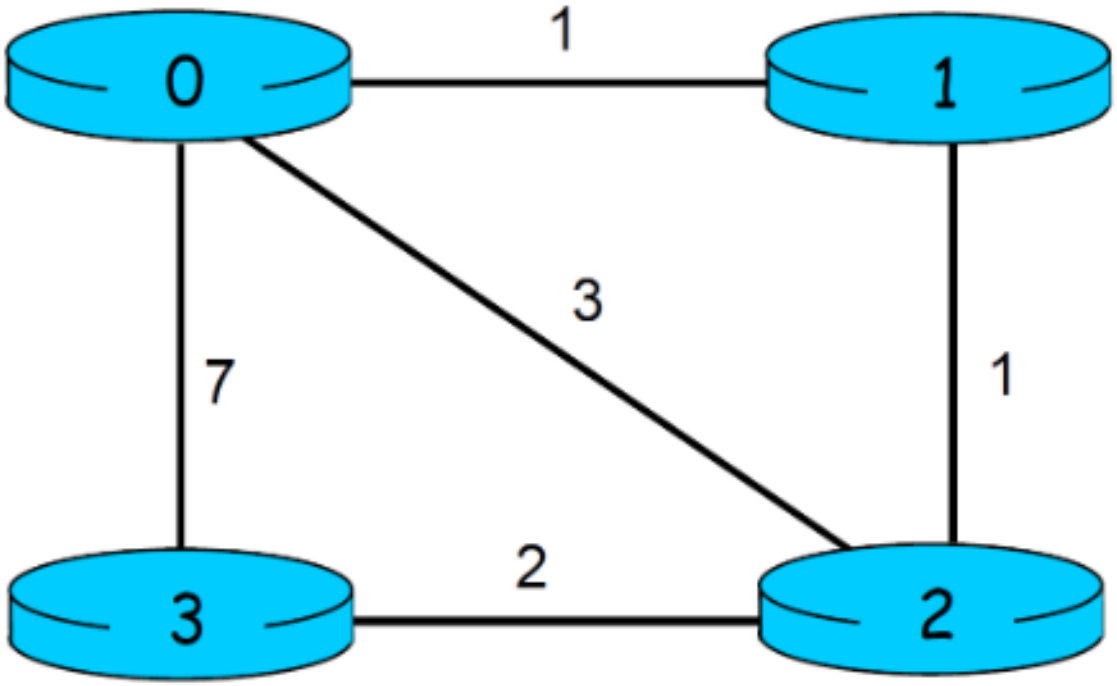

Figure 3: Given network topology with the costs between each pair of neighbours

3.3 Set-up

There were no device/OS specifications for this part of the assignment. All C code files implementing the simulated distance vector algorithm were run locally on a Windows machine (compiled using GCC compiler). 4 C code files corresponding to each of the 4 nodes with 2 functions to be completed in each file (described below) were given, along with a complete implementation of the helper functions needed in the code files to be completed by us. This implementation was made available in a file called distance-vector.c, which needed a minor machine-specific change, as described in the coming sections.

3.4 Distance Vector Algorithm Overview

The algorithm consists of the following key components:

1. **Initialization Phase**: Each node initializes its distance table with known costs to direct neighbors and infinity for non-neighbors.
2. **Information Exchange**: Nodes send their current distance vector to all directly connected neighbors.
3. **Distance Table Update**: Upon receiving a distance vector from a neighbor, each node updates its distance table based on the Bellman-Ford equation.
4. **Route Optimization**: After updates, each node recalculates its minimum cost paths.
5. **Convergence**: The algorithm converges when no further changes occur in any node's distance vectors.

3.5 Implementation in C

We were given 4 template code files (node0.c, node1.c, node2.c and node3.c) corresponding to the 4 nodes in the network, and we were expected to complete the following routines for each of the functions (here, their names are specific to node0.c):

1. **rtinit0()**: This routine will be called once at the beginning of the emulation. rtinit0() has no arguments. It should initialize the distance table in node 0 to reflect the direct costs of 1, 3, and 7 to nodes 1, 2, and 3, respectively. In the network topology figure above, all links are bi-directional and the costs in both directions are identical. After initializing the distance table, and any other data structures needed by your node 0 routines, it should then send its directly-connected neighbors (in this case, 1, 2 and 3) the cost of it minimum cost paths, i.e., its distance vector, to all other network nodes. This minimum cost information is sent to neighboring nodes in a routing packet by calling the routine tolayer2(), as described below. The format of the routing packet is also described below.

2. **rtupdate0 (struct rtpkt *rcvdpkt)**: This routine will be called when node 0 receives a routing packet that was sent to it by one if its directly connected neighbors. The parameter *rcvdpkt is a pointer to the packet that was received.

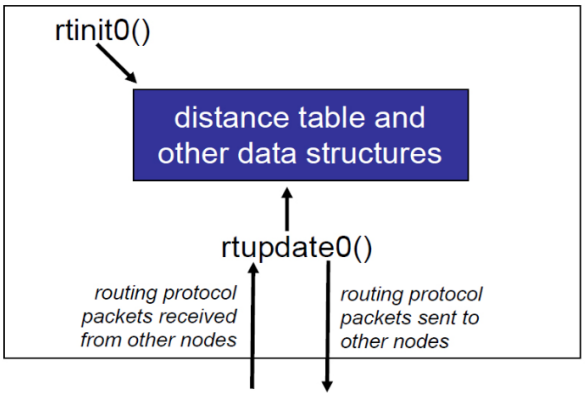

Figure 4: How the given routine templates are supposed to work

We used the following data structures in our implementation:

1. **Distance Table**: A 4×4 matrix for each node file where entry [i][j] represents the cost from the current node to node i via direct neighbor j.

2. **Routing Packet Structure**:

```
struct rtpkt {
  int sourceid;      /* id of sending router */
  int destid;        /* id of destination router */
  int mincost[4];    /* min cost to nodes 0...3 */
};

```

3. **Min Cost Array**: An array storing the minimum cost from the current node to all other nodes.

3.5.1 The distance-vector.c file

We were given a distance-vector.c file which contains routines that were called by the routines we completed:

1. tolayer2 (struct rtpkt pkt2send): where rtpkt is the data structure defined above.

2. printdt0(): this will pretty print the distance table for node 0. It is passed a pointer to a structure of type distance-table. printdt0() and the structure declaration for the node 0 distance table are declared in the file node0.c. Similar pretty-print routines are defined in the template files node1.c, node2.c node3.c.

We had to make a small change in this file in order to make it compatible with the machine we were implementing this on, as specified in the code itself:

```
/*************************************************************/
/* jimsrand(): return a float in range [0,1].  The routine below is
   used to */
/* isolate all random number generation in one location.  We assume
   that the*/
/* system-supplied rand() function return an int in therange [0,mmm]
        */
/*************************************************************/
float jimsrand()
{
  double mmm = __INT_MAX__;   /* largest int  - MACHINE DEPENDENT
   !!!!!!!!   */
  float x;                    /* individual students may need to change
   mmm */
  x = rand()/mmm;             /* x should be uniform in [0,1] */
  return(x);
}

/********************* EVENT HANDLINE ROUTINES *******/
/*  The next set of routines handle the event list   */
/*****************************************************/
```

An arbitrarily given value for the variable mmm had to be changed to the value mentioned above for ensuring machine compatibility. The distance-vector.c file also supported printing the TRACE of the running program. On compiling and running the program, we are asked to specify a trace value, and on specifying 2 as the trace, it prints out useful information about what is going on inside the emulation (e.g., what's happening to packets and timers). The trace output of our program has been presented in the next section.

3.5.2 Actual implementation of the routines in the node files

The rtinit0() function: This function initializes the routing algorithm for node 0. It is called when the node first boots up. It initializes the distance table (dt0) with direct connection costs. For each destination node i: If i is the same as the current node (0), it sets the cost to the direct connection cost. Otherwise, it sets the cost to INFINITY (999). After initialization, it calculates minimum costs to each node and sends out routing packets to neighbors. When a node receives a routing packet, it updates its distance table using its rtupadte() function. For instance, `rtupdate0()`: This function is called when node 0 receives a routing packet from another node. It extracts the source ID and

minimum costs from the received packet. For each possible destination node, it calculates a potential new cost:

```
possibleValue = dt0.costs[src][src] + mincost[i]
(cost to reach the source + source's cost to destination)
```

It updates the distance table with these new costs if they're less than infinity. After updating, it recalculates minimum costs and sends packets if costs have changed. Some other helper functions were also written for the following purposes:

1. For calculating the minimum cost to every other node.
2. For sending packets to neighbors (calls the tolayer2() method defined in the distance-vector.c file).
3. For detecting a change in the minimum cost to some node after receving the updated distance table, and resending updated packets to its neighbor nodes.
4. The linkhandler() function: It updates the internal distance table to reflect the new cost of the link and then checks if this change results in a new minimum path to any destination. If it does, it sends updated packets to neighbors. It is only used when there's a change in the direct link cost between node 0 and one of its neighbors.

```
1  void rtinit0() {
2      printf("rtinit0() is called at time t=: %0.3f\n", clocktime);
3
4      for (int i = 0; i < 4; i++) {
5          for (int j = 0; j < 4; j++) {
6              node0_table.matrix[i][j] = (i == j) ? link_costs_node0[i] :
       INFINITY;
7          }
8      }
9
10     printdt0(&node0_table);
11     update_shortest_paths_0();
12     broadcast_packets_0();
13 }
14
15 void rtupdate0(struct rtpkt *rcvdpkt) {
16     int src = rcvdpkt->sourceid;
17     int received_costs[4];
18
19     for (int i = 0; i < 4; i++) {
20         received_costs[i] = rcvdpkt->mincost[i];
21     }
22
23     printf("rtupdate0() is called at time t=: %0.3f as node %d sent a
       pkt with (%d  %d  %d  %d)\n",
24             clocktime, src,
25             received_costs[0], received_costs[1], received_costs[2],
       received_costs[3]);
26
27     for (int i = 0; i < 4; i++) {
28         int possible_cost = node0_table.matrix[src][src] +
       received_costs[i];
```

```
29          node0_table.matrix[i][src] = (possible_cost < INFINITY) ?
    possible_cost : INFINITY;
30      }
31
32      printdt0(&node0_table);
33      check_and_send_update_0();
34  }
```

All of these functions were written in each of the 4 node files.

3.6 Mathematical Foundation: Bellman-Ford Equation

The distance vector algorithm is based on the Bellman-Ford equation which states that:

$$D_x(y) = \min_v \{c(x,v) + D_v(y)\} \tag{1}$$

Where:

- $D_x(y)$ is the cost of the least-cost path from node x to node y
- $c(x,v)$ is the cost from node x to its directly connected neighbor v
- $D_v(y)$ is the cost of the least-cost path from node v to node y

In our implementation, we apply this equation when updating the distance table:

```
1 int possibleValue = dt0.costs[src][src] + mincost[i];
2 if(possibleValue < INFINITY)
3     dt0.costs[i][src] = possibleValue;
```

Where:

- `dt0.costs[src][src]` is the cost from node 0 to the source node of the received packet
- `mincost[i]` is the cost from the source node to node i
- `dt0.costs[i][src]` is the cost from node 0 to node i via the source node

3.7 Output of the compiled programs (trace)

The 5 files (node0.c, node1.c, node2.c, node3.c and distance-vector.c) were compiled using the command:

```
gcc -o final node0.c node1.c node2.c node3.c distance_vector.c
```

and an executable final was created. On running the executable and specifying a trace value of 2, we got the following output:

```
Enter TRACE:2
rtinit0() is called at time t=: 0.000
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1   999   999
dest 2|  999     3   999
```

```
     3|  999   999     7
At time t=0.000, node 0 sends packet to node 1 with: (0  1  3  7)
At time t=0.000, node 0 sends packet to node 2 with: (0  1  3  7)
At time t=0.000, node 0 sends packet to node 3 with: (0  1  3  7)
rtinit1() is called at time t=: 0.000
             via
   D1 |    0     2
  ----|-----------
     0|    1   999
dest 2|  999     1
     3|  999   999
At time t=0.000, node 1 sends packet to node 0 with: (1  0  1  999)
At time t=0.000, node 1 sends packet to node 2 with: (1  0  1  999)
rtinit2() is called at time t=: 0.000
               via
   D2 |    0     1    3
  ----|-----------------
     0|    3   999   999
dest 1|  999     1   999
     3|  999   999     2
At time t=0.000, node 2 sends packet to node 0 with: (3  1  0  2)
At time t=0.000, node 2 sends packet to node 1 with: (3  1  0  2)
At time t=0.000, node 2 sends packet to node 3 with: (3  1  0  2)
rtinit3() is called at time t=: 0.000
             via
   D3 |    0     2
  ----|-----------
     0|    7   999
dest 1|  999   999
     2|  999     2
At time t=0.000, node 3 sends packet to node 0 with: (7  999  2  0)
At time t=0.000, node 3 sends packet to node 2 with: (7  999  2  0)
MAIN: rcv event, t=0.094, at 1 src: 0, dest: 1, contents:   0   1   3   7
rtupdate1() is called at time t=: 0.094 as node 0 sent a pkt with (0  1  3  7)
             via
   D1 |    0     2
  ----|-----------
     0|    1   999
dest 2|    4     1
     3|    8   999
At time t=0.094, node 1 sends packet to node 0 with: (1  0  1  8)
At time t=0.094, node 1 sends packet to node 2 with: (1  0  1  8)
MAIN: rcv event, t=0.427, at 1 src: 2, dest: 1, contents:   3   1   0   2
rtupdate1() is called at time t=: 0.427 as node 2 sent a pkt with (3  1  0  2)
             via
   D1 |    0     2
  ----|-----------
     0|    1     4
```

```
dest 2|    4     1
     3|    8     3
At time t=0.427, node 1 sends packet to node 0 with: (1  0  1  3)
At time t=0.427, node 1 sends packet to node 2 with: (1  0  1  3)
MAIN: rcv event, t=0.998, at 0 src: 1, dest: 0, contents:   1   0   1 999
rtupdate0() is called at time t=: 0.998 as node 1 sent a pkt with (1  0  1  999)
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1   999   999
dest 2|    2     3   999
     3|  999   999     7
At time t=0.998, node 0 sends packet to node 1 with: (0  1  2  7)
At time t=0.998, node 0 sends packet to node 2 with: (0  1  2  7)
At time t=0.998, node 0 sends packet to node 3 with: (0  1  2  7)
MAIN: rcv event, t=1.244, at 3 src: 0, dest: 3, contents:   0   1   3   7
rtupdate3() is called at time t=: 1.244 as node 0 sent a pkt with (0  1  3  7)
             via
   D3 |    0     2
  ----|-----------
     0|    7   999
dest 1|    8   999
     2|   10     2
At time t=1.244, node 3 sends packet to node 0 with: (7  8  2  0)
At time t=1.244, node 3 sends packet to node 2 with: (7  8  2  0)
MAIN: rcv event, t=1.514, at 2 src: 0, dest: 2, contents:   0   1   3   7
rtupdate2() is called at time t=: 1.514 as node 0 sent a pkt with (0  1  3  7)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3   999   999
dest 1|    4     1   999
     3|   10   999     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=1.685, at 0 src: 2, dest: 0, contents:   3   1   0   2
rtupdate0() is called at time t=: 1.685 as node 2 sent a pkt with (3  1  0  2)
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4   999
dest 2|    2     3   999
     3|  999     5     7
At time t=1.685, node 0 sends packet to node 1 with: (0  1  2  5)
At time t=1.685, node 0 sends packet to node 2 with: (0  1  2  5)
At time t=1.685, node 0 sends packet to node 3 with: (0  1  2  5)
MAIN: rcv event, t=2.171, at 3 src: 2, dest: 3, contents:   3   1   0   2
rtupdate3() is called at time t=: 2.171 as node 2 sent a pkt with (3  1  0  2)
```

```
            via
   D3 |    0     2
  ----|-----------
     0|    7     5
dest 1|    8     3
     2|   10     2
At time t=2.171, node 3 sends packet to node 0 with: (5  3  2  0)
At time t=2.171, node 3 sends packet to node 2 with: (5  3  2  0)
MAIN: rcv event, t=2.399, at 0 src: 3, dest: 0, contents:   7 999   2   0
rtupdate0() is called at time t=: 2.399 as node 3 sent a pkt with (7  999  2  0)
               via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4   999
dest 2|    2     3     9
     3|  999     5     7

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=2.489, at 0 src: 1, dest: 0, contents:   1   0   1   8
rtupdate0() is called at time t=: 2.489 as node 1 sent a pkt with (1  0  1  8)
               via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4   999
dest 2|    2     3     9
     3|    9     5     7

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=2.667, at 2 src: 1, dest: 2, contents:   1   0   1 999
rtupdate2() is called at time t=: 2.667 as node 1 sent a pkt with (1  0  1  999)
               via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2   999
dest 1|    4     1   999
     3|   10   999     2
At time t=2.667, node 2 sends packet to node 0 with: (2  1  0  2)
At time t=2.667, node 2 sends packet to node 1 with: (2  1  0  2)
At time t=2.667, node 2 sends packet to node 3 with: (2  1  0  2)
MAIN: rcv event, t=2.823, at 1 src: 0, dest: 1, contents:   0   1   2   7
rtupdate1() is called at time t=: 2.823 as node 0 sent a pkt with (0  1  2  7)
            via
   D1 |    0     2
  ----|-----------
     0|    1     4
dest 2|    3     1
     3|    8     3
```

```
Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=3.361, at 0 src: 1, dest: 0, contents:   1   0   1   3
rtupdate0() is called at time t=: 3.361 as node 1 sent a pkt with (1  0  1  3)
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4   999
dest 2|    2     3     9
     3|    4     5     7
At time t=3.361, node 0 sends packet to node 1 with: (0  1  2  4)
At time t=3.361, node 0 sends packet to node 2 with: (0  1  2  4)
At time t=3.361, node 0 sends packet to node 3 with: (0  1  2  4)
MAIN: rcv event, t=3.780, at 3 src: 0, dest: 3, contents:   0   1   2   7
rtupdate3() is called at time t=: 3.780 as node 0 sent a pkt with (0  1  2  7)
             via
   D3 |    0     2
  ----|-----------
     0|    7     5
dest 1|    8     3
     2|    9     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=3.798, at 2 src: 3, dest: 2, contents:   7 999   2   0
rtupdate2() is called at time t=: 3.798 as node 3 sent a pkt with (7  999  2  0)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     9
dest 1|    4     1   999
     3|   10   999     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=3.915, at 0 src: 3, dest: 0, contents:   7   8   2   0
rtupdate0() is called at time t=: 3.915 as node 3 sent a pkt with (7  8  2  0)
                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    15
dest 2|    2     3     9
     3|    4     5     7

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=4.098, at 1 src: 0, dest: 1, contents:   0   1   2   5
rtupdate1() is called at time t=: 4.098 as node 0 sent a pkt with (0  1  2  5)
             via
   D1 |    0     2
  ----|-----------
     0|    1     4
```

```
dest 2|    3     1
     3|    6     3

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=4.650, at 1 src: 2, dest: 1, contents:   2   1   0   2
rtupdate1() is called at time t=: 4.650 as node 2 sent a pkt with (2  1  0  2)
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    6     3

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=4.774, at 2 src: 1, dest: 2, contents:   1   0   1   8
rtupdate2() is called at time t=: 4.774 as node 1 sent a pkt with (1  0  1  8)
               via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     9
dest 1|    4     1   999
     3|   10     9     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=5.464, at 3 src: 0, dest: 3, contents:   0   1   2   5
rtupdate3() is called at time t=: 5.464 as node 0 sent a pkt with (0  1  2  5)
             via
   D3 |    0     2
  ----|-----------
     0|    7     5
dest 1|    8     3
     2|    9     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=5.640, at 0 src: 3, dest: 0, contents:   5   3   2   0
rtupdate0() is called at time t=: 5.640 as node 3 sent a pkt with (5  3  2  0)
               via
   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=5.760, at 2 src: 1, dest: 2, contents:   1   0   1   3
rtupdate2() is called at time t=: 5.760 as node 1 sent a pkt with (1  0  1  3)
               via
   D2 |    0     1    3
```

```
  ----|-----------------
     0|    3     2     9
dest 1|    4     1   999
     3|   10     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=6.212, at 1 src: 0, dest: 1, contents:   0   1   2   4
rtupdate1() is called at time t=: 6.212 as node 0 sent a pkt with (0  1  2  4)
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    5     3

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=6.315, at 3 src: 2, dest: 3, contents:   2   1   0   2
rtupdate3() is called at time t=: 6.315 as node 2 sent a pkt with (2  1  0  2)
             via
   D3 |    0     2
  ----|-----------
     0|    7     4
dest 1|    8     3
     2|    9     2
At time t=6.315, node 3 sends packet to node 0 with: (4  3  2  0)
At time t=6.315, node 3 sends packet to node 2 with: (4  3  2  0)
MAIN: rcv event, t=6.435, at 0 src: 2, dest: 0, contents:   2   1   0   2
rtupdate0() is called at time t=: 6.435 as node 2 sent a pkt with (2  1  0  2)
               via
   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=6.771, at 2 src: 0, dest: 2, contents:   0   1   2   7
rtupdate2() is called at time t=: 6.771 as node 0 sent a pkt with (0  1  2  7)
               via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     9
dest 1|    4     1   999
     3|   10     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=7.003, at 3 src: 0, dest: 3, contents:   0   1   2   4
rtupdate3() is called at time t=: 7.003 as node 0 sent a pkt with (0  1  2  4)
```

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

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=7.406, at 0 src: 3, dest: 0, contents:   4   3   2   0
rtupdate0() is called at time t=: 7.406 as node 3 sent a pkt with (4  3  2  0)
                via
   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=7.650, at 2 src: 3, dest: 2, contents:   7   8   2   0
rtupdate2() is called at time t=: 7.650 as node 3 sent a pkt with (7  8  2  0)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     9
dest 1|    4     1    10
     3|   10     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=8.069, at 2 src: 0, dest: 2, contents:   0   1   2   5
rtupdate2() is called at time t=: 8.069 as node 0 sent a pkt with (0  1  2  5)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     9
dest 1|    4     1    10
     3|    8     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=9.377, at 2 src: 3, dest: 2, contents:   5   3   2   0
rtupdate2() is called at time t=: 9.377 as node 3 sent a pkt with (5  3  2  0)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     7
dest 1|    4     1     5
     3|    8     4     2

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

```
MAIN: rcv event, t=10.016, at 2 src: 0, dest: 2, contents:   0   1   2   4
rtupdate2() is called at time t=: 10.016 as node 0 sent a pkt with (0  1  2  4)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     7
dest 1|    4     1     5
     3|    7     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=11.030, at 2 src: 3, dest: 2, contents:   4   3   2   0
rtupdate2() is called at time t=: 11.030 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=10000.000, at -1                via
   D0 |    1     2    3
  ----|-----------------
     1|   20     4    10
dest 2|   21     3     9
     3|   23     5     7
At time t=10000.000, node 0 sends packet to node 1 with: (0  4  3  5)
At time t=10000.000, node 0 sends packet to node 2 with: (0  4  3  5)
At time t=10000.000, node 0 sends packet to node 3 with: (0  4  3  5)
             via
   D1 |    0     2
  ----|-----------
     0|   20     3
dest 2|   22     1
     3|   24     3
At time t=10000.000, node 1 sends packet to node 0 with: (3  0  1  3)
At time t=10000.000, node 1 sends packet to node 2 with: (3  0  1  3)
MAIN: rcv event, t=10000.013, at 2 src: 0, dest: 2, contents:   0   4   3   5
rtupdate2() is called at time t=: 10000.013 as node 0 sent a pkt with (0  4  3  5)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    7     1     5
     3|    8     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=10000.666, at 1 src: 0, dest: 1, contents:   0   4   3   5
```

```
rtupdate1() is called at time t=: 10000.666 as node 0 sent a pkt with (0  4  3  5)
             via
   D1 |    0     2
  ----|-----------
     0|   20     3
dest 2|   23     1
     3|   25     3

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=10001.100, at 0 src: 1, dest: 0, contents:   3   0   1   3
rtupdate0() is called at time t=: 10001.100 as node 1 sent a pkt with (3  0  1  3)
               via
   D0 |    1     2    3
  ----|-----------------
     1|   20     4    10
dest 2|   21     3     9
     3|   23     5     7

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=10001.130, at 3 src: 0, dest: 3, contents:   0   4   3   5
rtupdate3() is called at time t=: 10001.130 as node 0 sent a pkt with (0  4  3  5)
             via
   D3 |    0     2
  ----|-----------
     0|    7     4
dest 1|   11     3
     2|   10     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=10001.865, at 2 src: 1, dest: 2, contents:   3   0   1   3
rtupdate2() is called at time t=: 10001.865 as node 1 sent a pkt with (3  0  1  3)
               via
   D2 |    0     1    3
  ----|-----------------
     0|    3     4     6
dest 1|    7     1     5
     3|    8     4     2
At time t=10001.865, node 2 sends packet to node 0 with: (3  1  0  2)
At time t=10001.865, node 2 sends packet to node 1 with: (3  1  0  2)
At time t=10001.865, node 2 sends packet to node 3 with: (3  1  0  2)
MAIN: rcv event, t=10002.132, at 1 src: 2, dest: 1, contents:   3   1   0   2
rtupdate1() is called at time t=: 10002.132 as node 2 sent a pkt with (3  1  0  2)
             via
   D1 |    0     2
  ----|-----------
     0|   20     4
dest 2|   23     1
     3|   25     3
```

```
At time t=10002.132, node 1 sends packet to node 0 with: (4  0  1  3)
At time t=10002.132, node 1 sends packet to node 2 with: (4  0  1  3)
MAIN: rcv event, t=10002.603, at 0 src: 2, dest: 0, contents:   3   1   0   2
rtupdate0() is called at time t=: 10002.603 as node 2 sent a pkt with (3  1  0  2)
               via
   D0 |    1     2    3
  ----|-----------------
     1|   20     4    10
dest 2|   21     3     9
     3|   23     5     7

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=10002.604, at 0 src: 1, dest: 0, contents:   4   0   1   3
rtupdate0() is called at time t=: 10002.604 as node 1 sent a pkt with (4  0  1  3)
               via
   D0 |    1     2    3
  ----|-----------------
     1|   20     4    10
dest 2|   21     3     9
     3|   23     5     7

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=10003.143, at 3 src: 2, dest: 3, contents:   3   1   0   2
rtupdate3() is called at time t=: 10003.143 as node 2 sent a pkt with (3  1  0  2)
            via
   D3 |    0     2
  ----|-----------
     0|    7     5
dest 1|   11     3
     2|   10     2
At time t=10003.143, node 3 sends packet to node 0 with: (5  3  2  0)
At time t=10003.143, node 3 sends packet to node 2 with: (5  3  2  0)
MAIN: rcv event, t=10003.938, at 0 src: 3, dest: 0, contents:   5   3   2   0
rtupdate0() is called at time t=: 10003.938 as node 3 sent a pkt with (5  3  2  0)
               via
   D0 |    1     2    3
  ----|-----------------
     1|   20     4    10
dest 2|   21     3     9
     3|   23     5     7

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=10004.054, at 2 src: 1, dest: 2, contents:   4   0   1   3
rtupdate2() is called at time t=: 10004.054 as node 1 sent a pkt with (4  0  1  3)
               via
   D2 |    0     1    3
  ----|-----------------
     0|    3     5     6
```

```
dest 1|    7     1     5
     3|    8     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=10004.582, at 2 src: 3, dest: 2, contents:   5   3   2   0
rtupdate2() is called at time t=: 10004.582 as node 3 sent a pkt with (5  3  2  0)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     5     7
dest 1|    7     1     5
     3|    8     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=20000.000, at -281542320                via
   D0 |    1     2    3
  ----|-----------------
     1|    1     4    10
dest 2|    2     3     9
     3|    4     5     7
At time t=20000.000, node 0 sends packet to node 1 with: (0  1  2  4)
At time t=20000.000, node 0 sends packet to node 2 with: (0  1  2  4)
At time t=20000.000, node 0 sends packet to node 3 with: (0  1  2  4)
            via
   D1 |    0     2
  ----|-----------
     0|    1     4
dest 2|    4     1
     3|    6     3
At time t=20000.000, node 1 sends packet to node 0 with: (1  0  1  3)
At time t=20000.000, node 1 sends packet to node 2 with: (1  0  1  3)
MAIN: rcv event, t=20000.117, at 0 src: 1, dest: 0, contents:   1   0   1   3
rtupdate0() is called at time t=: 20000.117 as node 1 sent a pkt with (1  0  1  3)
                via
   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=20000.484, at 1 src: 0, dest: 1, contents:   0   1   2   4
rtupdate1() is called at time t=: 20000.484 as node 0 sent a pkt with (0  1  2  4)
            via
   D1 |    0     2
  ----|-----------
     0|    1     4
dest 2|    3     1
```

```
     3|    5     3

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=20001.498, at 2 src: 0, dest: 2, contents:   0   1   2   4
rtupdate2() is called at time t=: 20001.498 as node 0 sent a pkt with (0  1  2  4)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     5     7
dest 1|    4     1     5
     3|    7     4     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=20001.773, at 3 src: 0, dest: 3, contents:   0   1   2   4
rtupdate3() is called at time t=: 20001.773 as node 0 sent a pkt with (0  1  2  4)
             via
   D3 |    0     2
  ----|-----------
     0|    7     5
dest 1|    8     3
     2|    9     2

Minimum cost didn't change. No new packets are sent
MAIN: rcv event, t=20002.717, at 2 src: 1, dest: 2, contents:   1   0   1   3
rtupdate2() is called at time t=: 20002.717 as node 1 sent a pkt with (1  0  1  3)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     7
dest 1|    4     1     5
     3|    7     4     2
At time t=20002.717, node 2 sends packet to node 0 with: (2  1  0  2)
At time t=20002.717, node 2 sends packet to node 1 with: (2  1  0  2)
At time t=20002.717, node 2 sends packet to node 3 with: (2  1  0  2)
MAIN: rcv event, t=20002.922, at 1 src: 2, dest: 1, contents:   2   1   0   2
rtupdate1() is called at time t=: 20002.922 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
MAIN: rcv event, t=20004.307, at 0 src: 2, dest: 0, contents:   2   1   0   2
rtupdate0() is called at time t=: 20004.307 as node 2 sent a pkt with (2  1  0  2)
                via
   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=20004.629, at 3 src: 2, dest: 3, contents:   2   1   0   2
rtupdate3() is called at time t=: 20004.629 as node 2 sent a pkt with (2  1  0  2)
             via
   D3 |    0     2
  ----|-----------
     0|    7     4
dest 1|    8     3
     2|    9     2
At time t=20004.629, node 3 sends packet to node 0 with: (4  3  2  0)
At time t=20004.629, node 3 sends packet to node 2 with: (4  3  2  0)
MAIN: rcv event, t=20005.580, at 0 src: 3, dest: 0, contents:   4   3   2   0
rtupdate0() is called at time t=: 20005.580 as node 3 sent a pkt with (4  3  2  0)
               via
   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=20006.547, at 2 src: 3, dest: 2, contents:   4   3   2   0
rtupdate2() is called at time t=: 20006.547 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

Simulator terminated at t=20006.546875, no packets in medium
```

Final Distance Tables

After convergence, the final minimum costs from all nodes to all other nodes are:

```
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
```

3.8 Observations and Analysis

3.8.1 Convergence Speed

The algorithm converges after several iterations of packet exchanges, demonstrating the efficiency of the distributed approach. The number of iterations required depends on the network topology and initial conditions.

3.8.2 Information Propagation

Information about the network topology propagates correctly through the network, allowing nodes to discover paths to non-directly connected nodes. For example:

- Node 1 initially has no knowledge of Node 3's existence
- After updates, Node 1 learns it can reach Node 3 via Node 2 with cost 3

3.8.3 Path Optimization

The algorithm correctly identifies shorter indirect paths over direct paths when available. For example:

- Node 0 has a direct connection to Node 3 with cost 7
- However, it chooses the path 0→2→3 with total cost 5

3.8.4 Distributed Nature

Each node operates independently, making local decisions based on information received from neighbors. This distributed approach makes the algorithm robust and scalable.

3.9 Conclusion

The implemented distributed asynchronous distance vector routing algorithm successfully computes shortest paths between all nodes in the network. The algorithm demonstrates several key principles of distributed routing protocols:

1. **Local Computation**: Each node makes decisions based on local information
2. **Information Sharing**: Nodes share their knowledge with neighbors
3. **Iterative Improvement**: The solution improves over time through multiple update cycles
4. **Distributed Control**: No central controller is needed to coordinate routing decisions

This implementation provides insights into the operation of real-world routing protocols like RIP (Routing Information Protocol) that use distance vector algorithms. The asynchronous nature of the algorithm makes it robust to variations in network conditions and message delays.

However, distance vector algorithms also have limitations, such as slow convergence on large networks and vulnerability to the "count-to-infinity" problem when links fail. It is not very scalable. These limitations have led to the development of alternative approaches like link-state routing protocols used in OSPF (Open Shortest Path First).

4 References

1. Kurose, J. F., Ross, K. W. (2024). Computer Networking: A Top-Down Approach (8th ed.). Pearson.

2. Tanenbaum, A. S., & Wetherall, D. J. (2023). Computer Networks (6th ed.). Pearson.

3. Peterson, L. L., & Davie, B. S. (2022). Computer Networks: A Systems Approach (6th ed.). Morgan Kaufmann.

4. IEEE 802.1D (2022). IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges. IEEE Computer Society.

5. Mininet. (2025). Mininet: An Instant Virtual Network on your Laptop (or other PC). Retrieved from `http://mininet.org/`

6. Open vSwitch Documentation. (2025). Open vSwitch and OpenFlow. Retrieved from `https://docs.openvswitch.org/`

7. Linux Foundation. (2025). iptables Documentation. Retrieved from `https://netfilter.org/documentation/`