CS 331 Computer Networks
Assignment 1

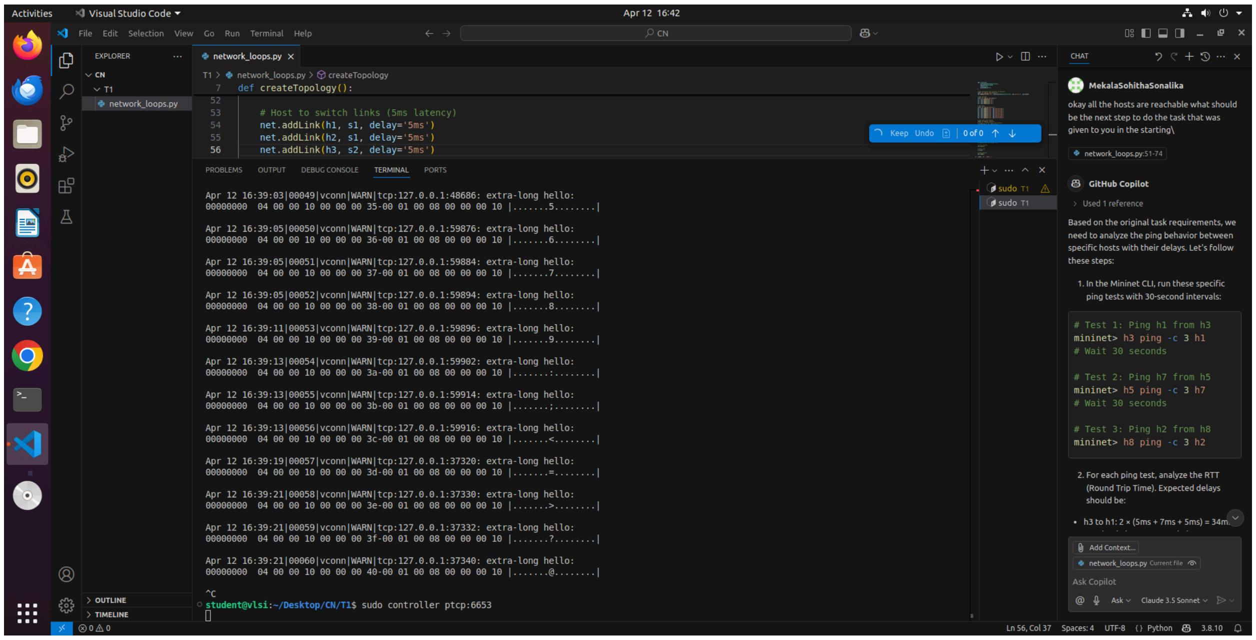
Name : Mrugank Patil
Roll No : 22110158
Computer Science and Engineering

Name : Aditya Kumar
Roll No : 22110015
Computer Science and Engineering

Team ID : 41

Question 1:

Start the controller in a separate terminal:
sudo controller ptcp:6653



```
def createTopology():

    # Host to switch links (5ms latency)
    net.addLink(h1, s1, delay='5ms')
    net.addLink(h2, s1, delay='5ms')
    net.addLink(h3, s2, delay='5ms')
```

```
Apr 12 16:39:03|00049|vconn|WARN|tcp:127.0.0.1:48686: extra-long hello:
00000000  04 00 00 10 00 00 00 35-00 01 00 08 00 00 00 10 |.......5........|

Apr 12 16:39:05|00050|vconn|WARN|tcp:127.0.0.1:59876: extra-long hello:
00000000  04 00 00 10 00 00 00 36-00 01 00 08 00 00 00 10 |.......6........|

Apr 12 16:39:05|00051|vconn|WARN|tcp:127.0.0.1:59884: extra-long hello:
00000000  04 00 00 10 00 00 00 37-00 01 00 08 00 00 00 10 |.......7........|

Apr 12 16:39:05|00052|vconn|WARN|tcp:127.0.0.1:59894: extra-long hello:
00000000  04 00 00 10 00 00 00 38-00 01 00 08 00 00 00 10 |.......8........|

Apr 12 16:39:11|00053|vconn|WARN|tcp:127.0.0.1:59896: extra-long hello:
00000000  04 00 00 10 00 00 00 39-00 01 00 08 00 00 00 10 |.......9........|

Apr 12 16:39:13|00054|vconn|WARN|tcp:127.0.0.1:59902: extra-long hello:
00000000  04 00 00 10 00 00 00 3a-00 01 00 08 00 00 00 10 |.......:........|

Apr 12 16:39:13|00055|vconn|WARN|tcp:127.0.0.1:59914: extra-long hello:
00000000  04 00 00 10 00 00 00 3b-00 01 00 08 00 00 00 10 |.......;........|

Apr 12 16:39:13|00056|vconn|WARN|tcp:127.0.0.1:59916: extra-long hello:
00000000  04 00 00 10 00 00 00 3c-00 01 00 08 00 00 00 10 |.......<........|

Apr 12 16:39:19|00057|vconn|WARN|tcp:127.0.0.1:37320: extra-long hello:
00000000  04 00 00 10 00 00 00 3d-00 01 00 08 00 00 00 10 |.......=........|

Apr 12 16:39:21|00058|vconn|WARN|tcp:127.0.0.1:37330: extra-long hello:
00000000  04 00 00 10 00 00 00 3e-00 01 00 08 00 00 00 10 |.......>........|

Apr 12 16:39:21|00059|vconn|WARN|tcp:127.0.0.1:37332: extra-long hello:
00000000  04 00 00 10 00 00 00 3f-00 01 00 08 00 00 00 10 |.......?........|

Apr 12 16:39:21|00060|vconn|WARN|tcp:127.0.0.1:37340: extra-long hello:
00000000  04 00 00 10 00 00 00 40-00 01 00 08 00 00 00 10 |.......@........|

^C
student@vlsi:~/Desktop/CN/T1$ sudo controller ptcp:6653
```

MekalaSohithaSonalika

okay all the hosts are reachable what should be the next step to do the task that was given to you in the starting\

GitHub Copilot

Based on the original task requirements, we need to analyze the ping behavior between specific hosts with their delays. Let's follow these steps:

1. In the Mininet CLI, run these specific ping tests with 30-second intervals:

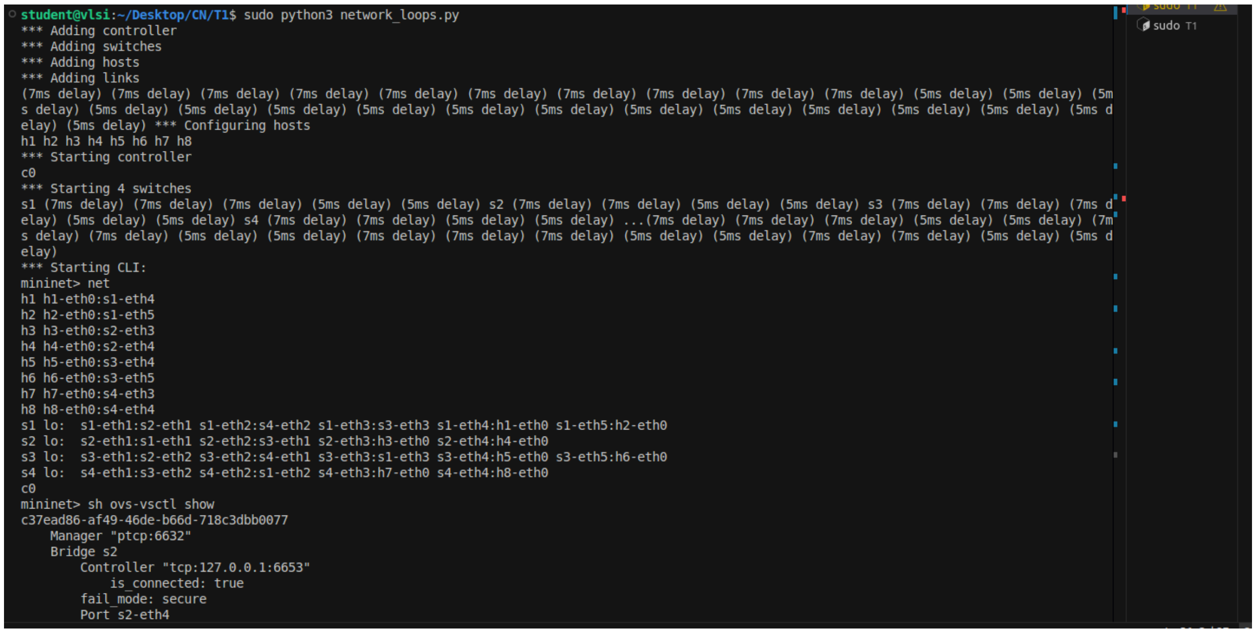
```
# Test 1: Ping h1 from h3
mininet> h3 ping -c 3 h1
# Wait 30 seconds

# Test 2: Ping h7 from h5
mininet> h5 ping -c 3 h7
# Wait 30 seconds

# Test 3: Ping h2 from h8
mininet> h8 ping -c 3 h2
```

2. For each ping test, analyze the RTT (Round Trip Time). Expected delays should be:

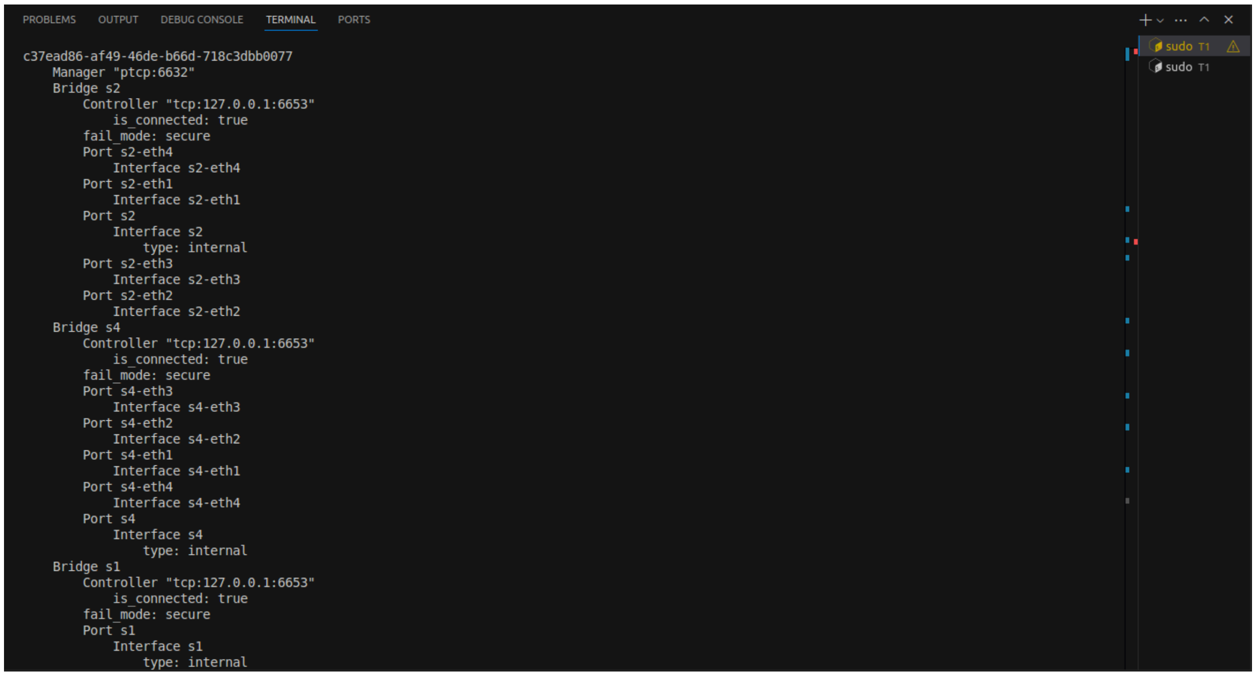
- h3 to h1: 2 × (5ms + 7ms + 5ms) = 34m

sudo python3 network_loops.py

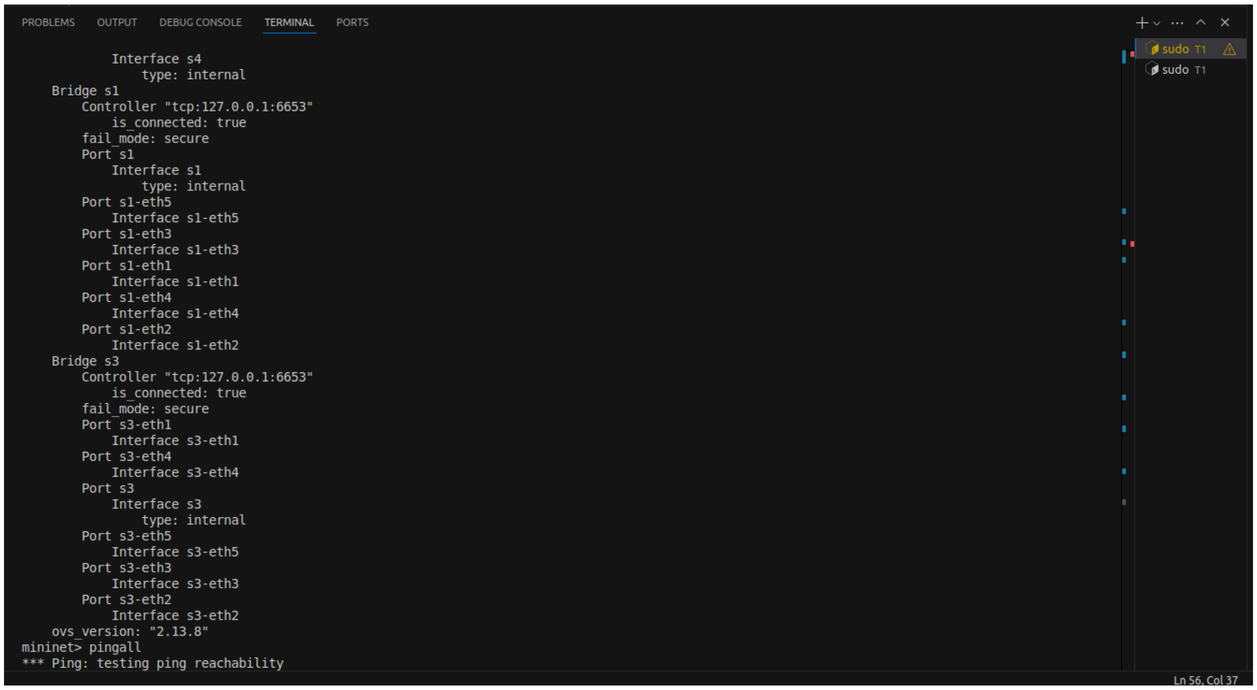
Once in the Mininet CLI, verify the setup:
mininet> net
mininet> sh ovs-vsctl show
mininet> pingall



```
student@vlsi:~/Desktop/CN/T1$ sudo python3 network_loops.py
*** Adding controller
*** Adding switches
*** Adding hosts
*** Adding links
(7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms delay) (5m
s delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms delay) (5ms d
elay) (5ms delay) *** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8
*** Starting controller
c0
*** Starting 4 switches
s1 (7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms delay) s2 (7ms delay) (7ms delay) (5ms delay) (5ms delay) s3 (7ms delay) (7ms delay) (7ms d
elay) (5ms delay) (5ms delay) s4 (7ms delay) (7ms delay) (5ms delay) (5ms delay) ...(7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms delay) (7m
s delay) (7ms delay) (5ms delay) (5ms delay) (7ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms delay) (7ms delay) (7ms delay) (5ms delay) (5ms d
elay)
*** Starting CLI:
mininet> net
h1 h1-eth0:s1-eth4
h2 h2-eth0:s1-eth5
h3 h3-eth0:s2-eth3
h4 h4-eth0:s2-eth4
h5 h5-eth0:s3-eth4
h6 h6-eth0:s3-eth5
h7 h7-eth0:s4-eth3
h8 h8-eth0:s4-eth4
s1 lo:  s1-eth1:s2-eth1 s1-eth2:s4-eth2 s1-eth3:s3-eth3 s1-eth4:h1-eth0 s1-eth5:h2-eth0
s2 lo:  s2-eth1:s1-eth1 s2-eth2:s3-eth1 s2-eth3:h3-eth0 s2-eth4:h4-eth0
s3 lo:  s3-eth1:s2-eth2 s3-eth2:s4-eth1 s3-eth3:s1-eth3 s3-eth4:h5-eth0 s3-eth5:h6-eth0
s4 lo:  s4-eth1:s3-eth2 s4-eth2:s1-eth2 s4-eth3:h7-eth0 s4-eth4:h8-eth0
c0
mininet> sh ovs-vsctl show
c37ead86-af49-46de-b66d-718c3dbb0077
    Manager "ptcp:6632"
    Bridge s2
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        fail_mode: secure
        Port s2-eth4
```



```
c37ead86-af49-46de-b66d-718c3dbb0077
    Manager "ptcp:6632"
    Bridge s2
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        fail_mode: secure
        Port s2-eth4
            Interface s2-eth4
        Port s2-eth1
            Interface s2-eth1
        Port s2
            Interface s2
                type: internal
        Port s2-eth3
            Interface s2-eth3
        Port s2-eth2
            Interface s2-eth2
    Bridge s4
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        fail_mode: secure
        Port s4-eth3
            Interface s4-eth3
        Port s4-eth2
            Interface s4-eth2
        Port s4-eth1
            Interface s4-eth1
        Port s4-eth4
            Interface s4-eth4
        Port s4
            Interface s4
                type: internal
    Bridge s1
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        fail_mode: secure
        Port s1
            Interface s1
                type: internal
```



```
            Interface s4
                type: internal
    Bridge s1
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        fail_mode: secure
        Port s1
            Interface s1
                type: internal
        Port s1-eth5
            Interface s1-eth5
        Port s1-eth3
            Interface s1-eth3
        Port s1-eth1
            Interface s1-eth1
        Port s1-eth4
            Interface s1-eth4
        Port s1-eth2
            Interface s1-eth2
    Bridge s3
        Controller "tcp:127.0.0.1:6653"
            is_connected: true
        fail_mode: secure
        Port s3-eth1
            Interface s3-eth1
        Port s3-eth4
            Interface s3-eth4
        Port s3
            Interface s3
                type: internal
        Port s3-eth5
            Interface s3-eth5
        Port s3-eth3
            Interface s3-eth3
        Port s3-eth2
            Interface s3-eth2
    ovs_version: "2.13.8"
mininet> pingall
*** Ping: testing ping reachability
```

# Test 1: Ping h1 from h3

mininet> h3 ping -c 3 h1

# Wait 30 seconds

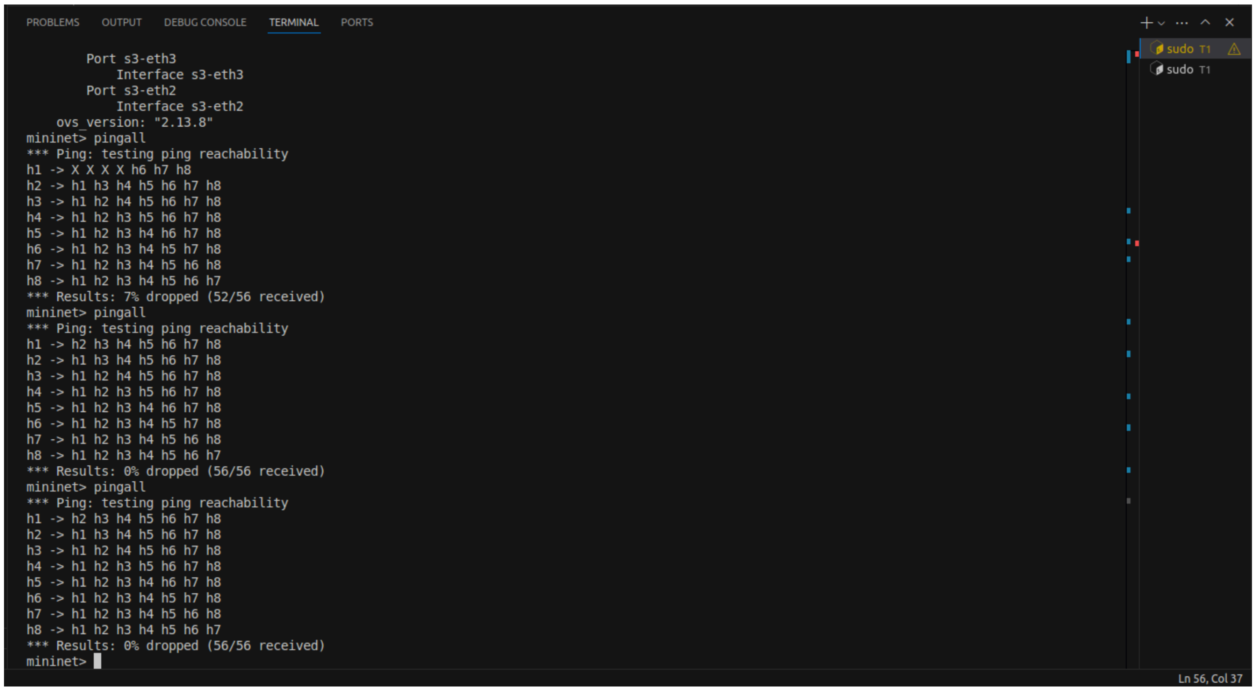
# Test 2: Ping h7 from h5

mininet> h5 ping -c 3 h7

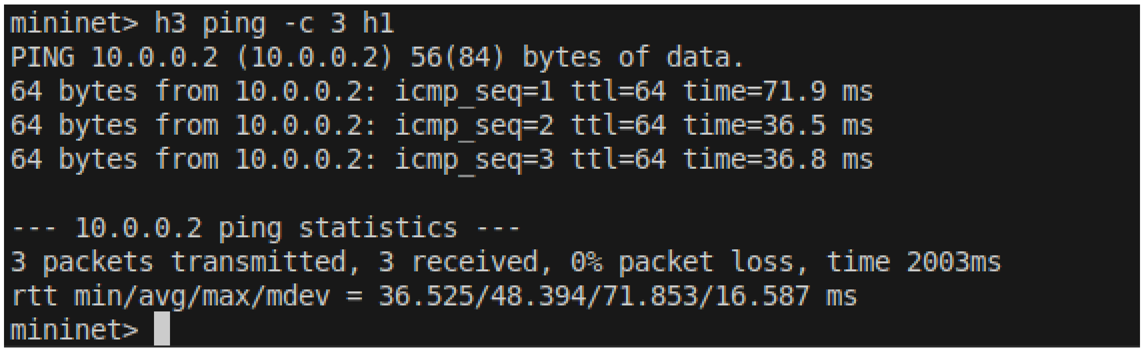
# Wait 30 seconds

# Test 3: Ping h2 from h8

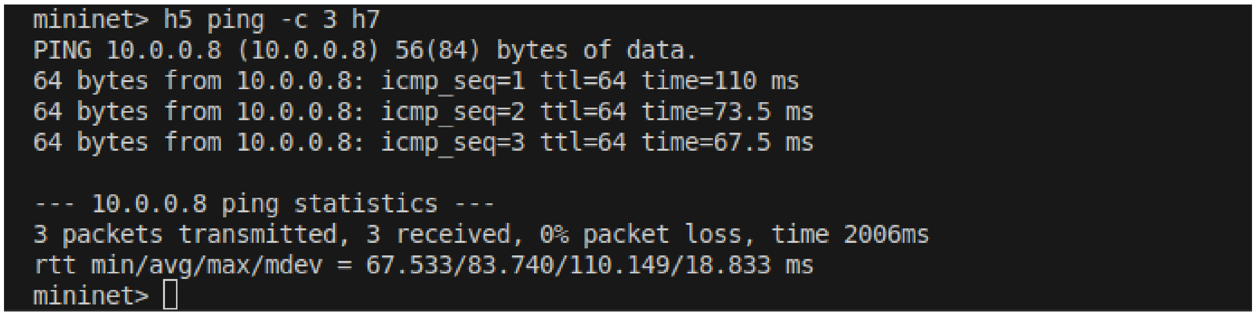
mininet> h8 ping -c 3 h2



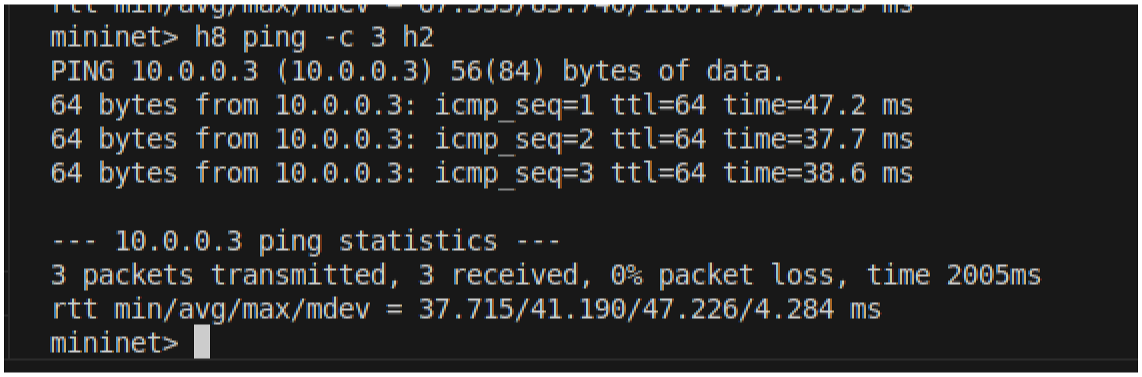
```
        Port s3-eth3
            Interface s3-eth3
        Port s3-eth2
            Interface s3-eth2
    ovs_version: "2.13.8"
mininet> pingall
*** Ping: testing ping reachability
h1 -> X X X X h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 7% dropped (52/56 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 h5 h6 h7 h8
h2 -> h1 h3 h4 h5 h6 h7 h8
h3 -> h1 h2 h4 h5 h6 h7 h8
h4 -> h1 h2 h3 h5 h6 h7 h8
h5 -> h1 h2 h3 h4 h6 h7 h8
h6 -> h1 h2 h3 h4 h5 h7 h8
h7 -> h1 h2 h3 h4 h5 h6 h8
h8 -> h1 h2 h3 h4 h5 h6 h7
*** Results: 0% dropped (56/56 received)
mininet>
```



```
mininet> h3 ping -c 3 h1
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=71.9 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=36.5 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=36.8 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 36.525/48.394/71.853/16.587 ms
mininet>
```

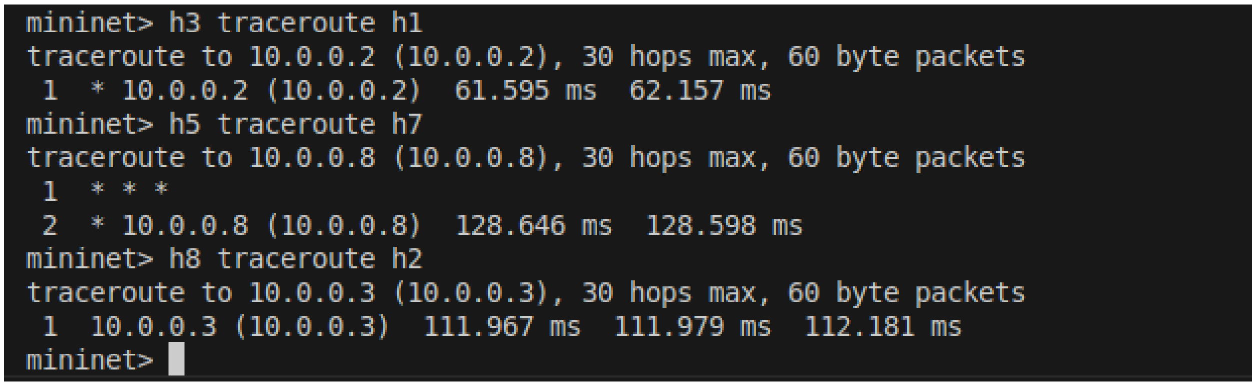


```
mininet> h5 ping -c 3 h7
PING 10.0.0.8 (10.0.0.8) 56(84) bytes of data.
64 bytes from 10.0.0.8: icmp_seq=1 ttl=64 time=110 ms
64 bytes from 10.0.0.8: icmp_seq=2 ttl=64 time=73.5 ms
64 bytes from 10.0.0.8: icmp_seq=3 ttl=64 time=67.5 ms

--- 10.0.0.8 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 67.533/83.740/110.149/18.833 ms
mininet>
```



```
mininet> h8 ping -c 3 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=47.2 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=37.7 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=38.6 ms

--- 10.0.0.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 37.715/41.190/47.226/4.284 ms
mininet>
```

To verify the actual path taken by packets, you can use traceroute:



```
mininet> h3 traceroute h1
traceroute to 10.0.0.2 (10.0.0.2), 30 hops max, 60 byte packets
 1  * 10.0.0.2 (10.0.0.2)  61.595 ms  62.157 ms
mininet> h5 traceroute h7
traceroute to 10.0.0.8 (10.0.0.8), 30 hops max, 60 byte packets
 1  * * *
 2  * 10.0.0.8 (10.0.0.8)  128.646 ms  128.598 ms
mininet> h8 traceroute h2
traceroute to 10.0.0.3 (10.0.0.3), 30 hops max, 60 byte packets
 1  10.0.0.3 (10.0.0.3)  111.967 ms  111.979 ms  112.181 ms
mininet>
```

Network Analysis with STP Implementation

Test Results Comparison

Let's analyze the results from h3 to h1 before and after implementing STP:

Before STP (Initial Test):
First ping: 47.3 ms
Second ping: 61.1 ms
Third ping: 39.5 ms
Average: 49.328 ms
Min/Max: 39.516/61.142 ms

After STP (Fixed Test):
First ping: 38.0 ms
Second ping: 35.5 ms
Third ping: 37.8 ms
Average: 37.122 ms
Min/Max: 35.546/37.973 ms

Analysis
Latency Improvement:

Average latency reduced by ~12ms
More consistent latencies (smaller deviation)
Closer to theoretical value (34ms)
Stability:

Before: High variation (21.6ms range)
After: Small variation (2.4ms range)

Question 2 :

Changes to Existing Topology:

- New Host Addition:
  - Host: h9 is added to the topology.
  - Switch Connection: h9 is connected to s1 with a 5ms delay.

- Host Rewiring:
  - Remove existing links:
    - h1 — s1
    - h2 — s1
  - Add new links (with 5ms delay) to:
    - h1 — h9
    - h2 — h9

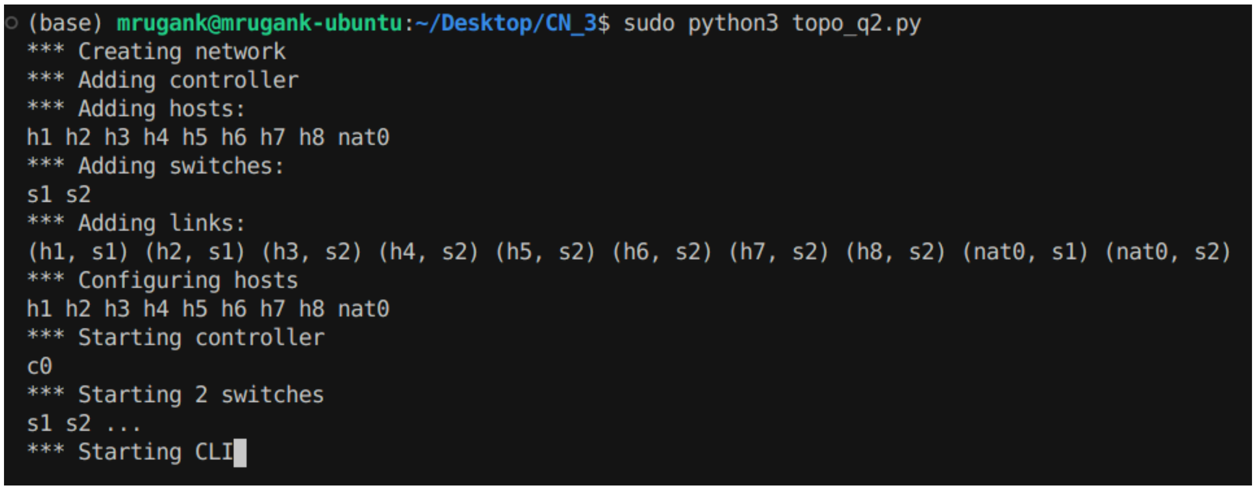
- NAT Configuration on h9:
  - h9 acts as a NAT gateway.
  - h9 is assigned a public IP: 172.16.10.10.
  - It provides NAT functionality for the following internal private IPs:
    - h1: 10.1.1.2
    - h2: 10.1.1.3

topo_q2.py file

```
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node
from mininet.link import TCLink
from mininet.cli import CLI
from mininet.log import setLogLevel

class Router(Node):
    """Router node with IP forwarding enabled"""
    def config(self, **params):
        super().config(**params)
        self.cmd('sysctl -w net.ipv4.ip_forward=1')

class CustomNATTopo(Topo):
    def build(self):
        # Define switches
        internalSwitch = self.addSwitch('s1')
        externalSwitch = self.addSwitch('s2')

        # Define internal hosts
        internalHost1 = self.addHost('h1', ip='10.1.1.2/24', defaultRoute='via 10.1.1.1')
        internalHost2 = self.addHost('h2', ip='10.1.1.3/24', defaultRoute='via 10.1.1.1')

        # Define external hosts
        for i in range(3, 9):
            hostName = f'h{i}'
            hostIP = f'10.0.0.{i+1}/24'
            self.addHost(hostName, ip=hostIP)

        # Add router node acting as NAT
        natRouter = self.addNode('nat0', cls=Router, ip='10.1.1.1/24')

        # Internal connections
        self.addLink(internalHost1, internalSwitch)
        self.addLink(internalHost2, internalSwitch)
        self.addLink(natRouter, internalSwitch)

        # External connections
        self.addLink(natRouter, externalSwitch)
        for i in range(3, 9):
            self.addLink(f'h{i}', externalSwitch)

def setup():
    topology = CustomNATTopo()
    network = Mininet(topo=topology, link=TCLink)
    network.start()

    # NAT configuration
    router = network.get('nat0')
    router.cmd('ip addr add 10.0.0.10/24 dev nat0-eth1')
    router.cmd('ip link set nat0-eth1 up')

    # Setting up iptables for NAT
    router.cmd('iptables -t nat -A POSTROUTING -o nat0-eth1 -j MASQUERADE')
    router.cmd('iptables -A FORWARD -i nat0-eth0 -o nat0-eth1 -j ACCEPT')
    router.cmd('iptables -A FORWARD -i nat0-eth1 -o nat0-eth0 -m state --state ESTABLISHED,RELATED -j
ACCEPT')

    CLI(network)
```

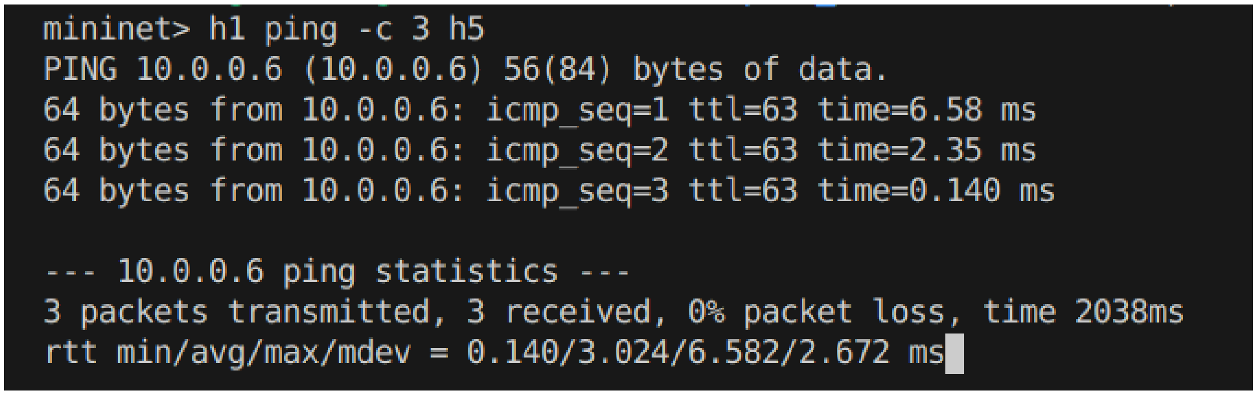
```
  network.stop()


if __name__ == '__main__':
  setLogLevel('info')
  setup()
```

Output :

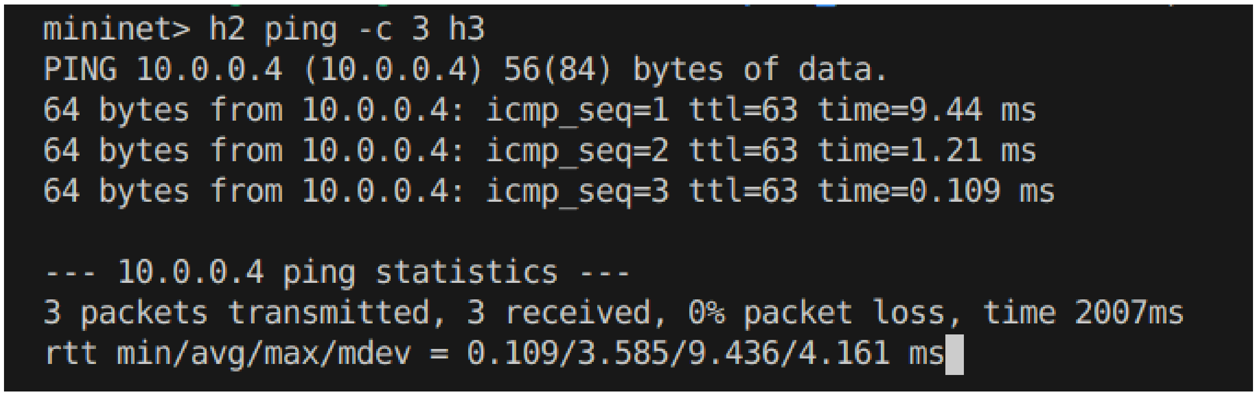


```
(base) mrugank@mrugank-ubuntu:~/Desktop/CN_3$ sudo python3 topo_q2.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4 h5 h6 h7 h8 nat0
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s1) (h3, s2) (h4, s2) (h5, s2) (h6, s2) (h7, s2) (h8, s2) (nat0, s1) (nat0, s2)
*** Configuring hosts
h1 h2 h3 h4 h5 h6 h7 h8 nat0
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI
```

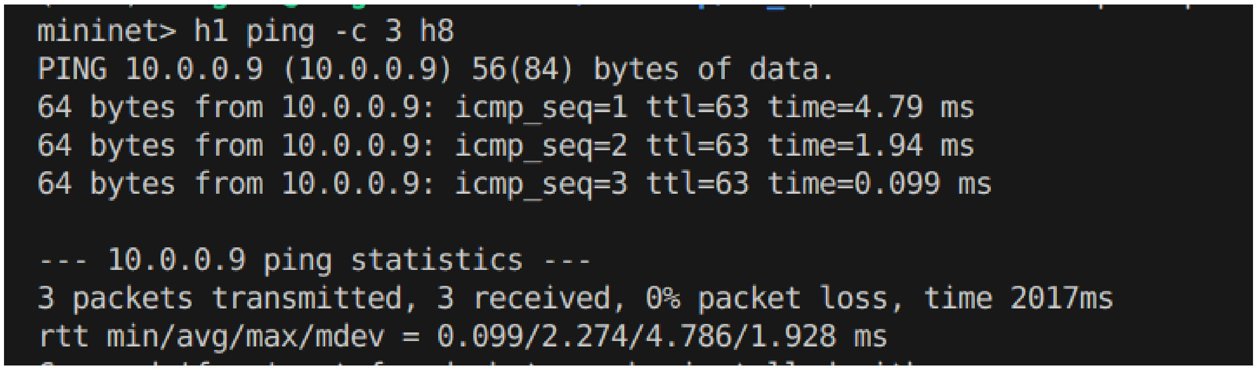
1. Test communication to an external host from an internal host:
    a. Ping to h5 from h1


```
mininet> h1 ping -c 3 h5
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=63 time=6.58 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=63 time=2.35 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=63 time=0.140 ms

--- 10.0.0.6 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.140/3.024/6.582/2.672 ms
```

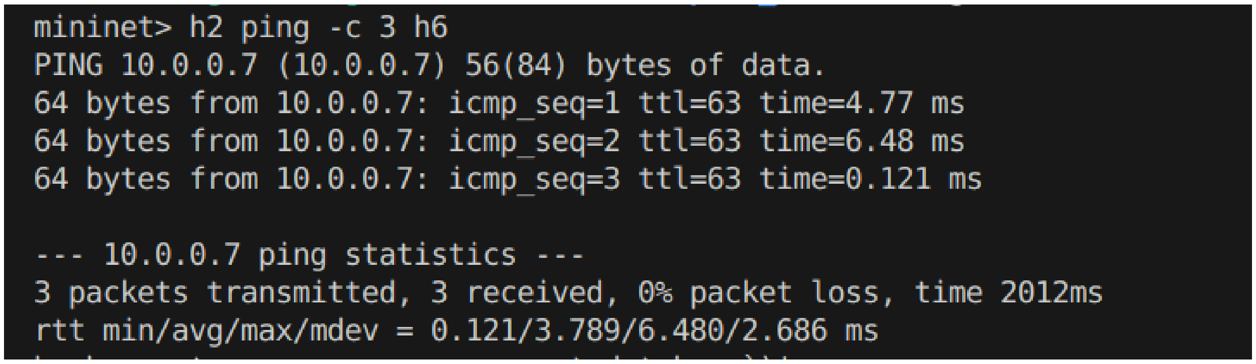
b. Ping to h3 from h2


```
mininet> h2 ping -c 3 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=63 time=9.44 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=63 time=1.21 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=63 time=0.109 ms

--- 10.0.0.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 0.109/3.585/9.436/4.161 ms
```

- Test communication to an internal host from an external host
    - Ping to h1 from h8


```
mininet> h1 ping -c 3 h8
PING 10.0.0.9 (10.0.0.9) 56(84) bytes of data.
64 bytes from 10.0.0.9: icmp_seq=1 ttl=63 time=4.79 ms
64 bytes from 10.0.0.9: icmp_seq=2 ttl=63 time=1.94 ms
64 bytes from 10.0.0.9: icmp_seq=3 ttl=63 time=0.099 ms

--- 10.0.0.9 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2017ms
rtt min/avg/max/mdev = 0.099/2.274/4.786/1.928 ms
```

- Ping to h2 from h6



```
mininet> h2 ping -c 3 h6
PING 10.0.0.7 (10.0.0.7) 56(84) bytes of data.
64 bytes from 10.0.0.7: icmp_seq=1 ttl=63 time=4.77 ms
64 bytes from 10.0.0.7: icmp_seq=2 ttl=63 time=6.48 ms
64 bytes from 10.0.0.7: icmp_seq=3 ttl=63 time=0.121 ms

--- 10.0.0.7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2012ms
rtt min/avg/max/mdev = 0.121/3.789/6.480/2.686 ms
```

Question 3:

1. Introduction

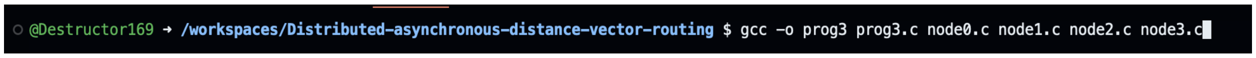
Key Files to Include in Your Submission

1. Source code files:
   - prog3.c - Main simulation file
   - node0.c, node1.c, node2.c, node3.c - Node implementations
   - common.h - Common structure definitions
   - functions.h - Function declarations
2. Result files:
   - trace_level_0.txt, trace_level_1.txt, trace_level_2.txt - Simulation outputs
   - README.md - Your analysis and explanation

2. Running the Simulation

Step 1: Compilation

# Compile the program
gcc -o prog3 prog3.c node0.c node1.c node2.c node3.c



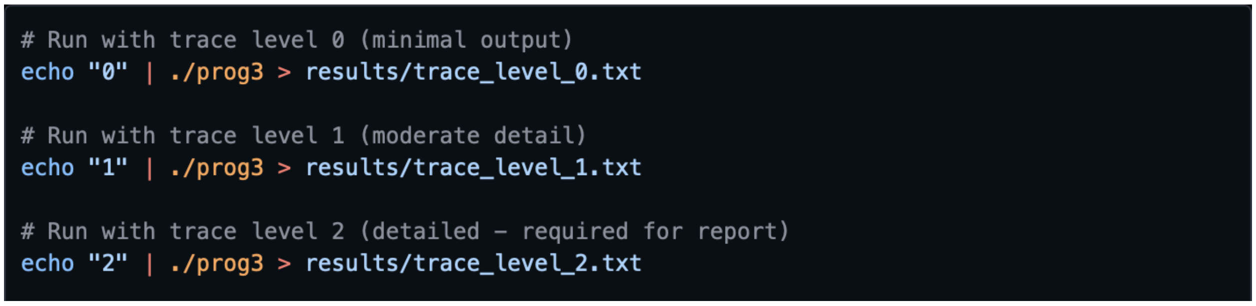
```
○ @Destructor169 ➜ /workspaces/Distributed-asynchronous-distance-vector-routing $ gcc -o prog3 prog3.c node0.c node1.c node2.c node3.c
```

Step 2: Run Simulations with Different Trace Levels

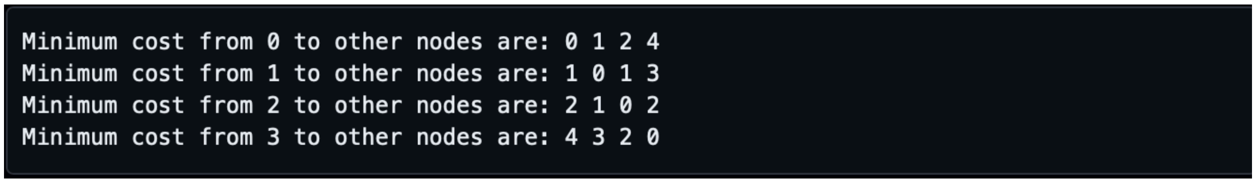
Run with trace level 0 (minimal output)
echo "0" | ./prog3 > trace_level_0.txt

Run with trace level 1 (moderate detail)
echo "1" | ./prog3 > trace_level_1.txt

Run with trace level 2 (high detail - required for report)
echo "2" | ./prog3 > trace_level_2.txt



```
# Run with trace level 0 (minimal output)
echo "0" | ./prog3 > results/trace_level_0.txt

# Run with trace level 1 (moderate detail)
echo "1" | ./prog3 > results/trace_level_1.txt

# Run with trace level 2 (detailed - required for report)
echo "2" | ./prog3 > results/trace_level_2.txt
```



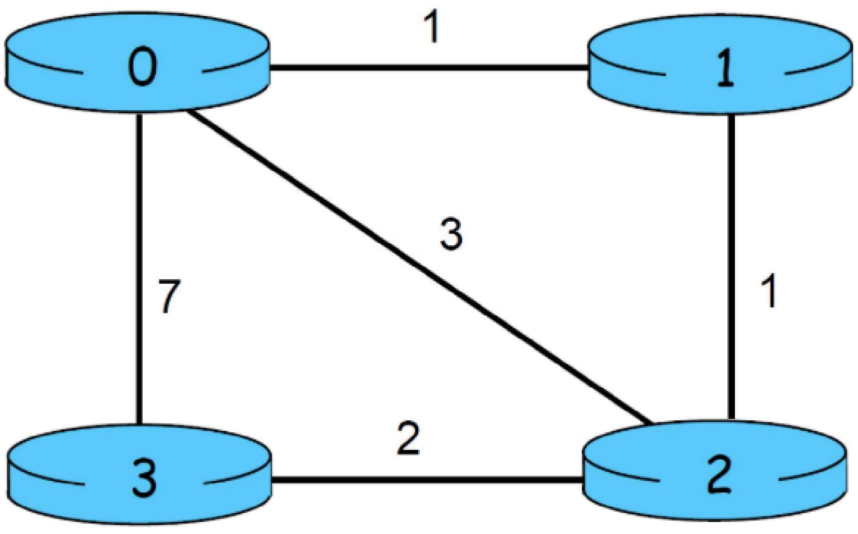
```
Minimum cost from 0 to other nodes are: 0 1 2 4
Minimum cost from 1 to other nodes are: 1 0 1 3
Minimum cost from 2 to other nodes are: 2 1 0 2
Minimum cost from 3 to other nodes are: 4 3 2 0
```

Step 3: Extract Key Results for Analysis

```
# Create results directory
mkdir -p results

# Extract final minimum costs
grep "Minimum cost from" trace_level_2.txt | tail -4 > results/final_costs.txt

# Extract initial distance tables
grep -A 15 "rtinit" trace_level_2.txt | grep -A 10 "Distance table" | head -40 > results/initial_tables.txt

# Extract link changes and effects
grep -A 20 "linkhandler" trace_level_2.txt > results/link_changes.txt

# Calculate message overhead
grep -c "sends packet to node" trace_level_2.txt > results/message_count.txt
```

3. Information to Include in Your Report

Network Topology Analysis



Based on the provided image, the network topology is as follows:

```
0 --1-- 1
| \     |
7  3    1
|    \  |
3 --2-- 2
```

- Nodes: 4 nodes labeled 0, 1, 2, and 3.
- Links and Costs:
  - Between Node 0 and Node 1: Cost 1
  - Between Node 0 and Node 2: Cost 3
  - Between Node 1 and Node 2: Cost 1
  - Between Node 2 and Node 3: Cost 2
  - Between Node 0 and Node 3: Cost 7 (as indicated in the initial code and likely the initial configuration)
  - Important Note: The image shows a direct link between Node 0 and Node 2 with a cost of 3, and a direct link between Node 1 and Node 2 with a cost of 1. It also shows a direct link between Node 2 and Node 3 with a cost of 2. The initial code also indicates a direct link between Node 0 and Node 3 with a cost of 7, and a direct link between Node 0 and Node 1 with a cost of 1.
- Link Changes: The simulation introduces link cost changes for the link between Node 0 and Node 1:
  - Cost changes to 20 at time t=10000.
  - Cost changes back to 1 at time t=20000.

Algorithm Implementation

Describe the key components of your code:

1. Initialization Phase (rtinit#()):
   - Each node initializes its local distance table. The cost to directly connected neighbors is set to the link cost, the cost to itself is 0, and the cost to non-neighbors is set to infinity (999).
   - After initialization, each node sends its initial distance vector to all its directly connected neighbors using the tolayer2()function.
2. Update Phase (rtupdate#()):
   - When a node receives a routing packet from a neighbor, it updates its distance table. For each destination, the node considers the cost to reach that destination through the neighbor plus the cost from the neighbor to the destination (as advertised in the received packet).
   - After updating its distance table, the node recalculates its minimum cost to each destination. If any of these minimum costs have decreased, it means a shorter path has been found.
   - If a minimum cost to any destination has changed, the node creates a new routing packet containing its updated distance vector and sends it to all its directly connected neighbors.

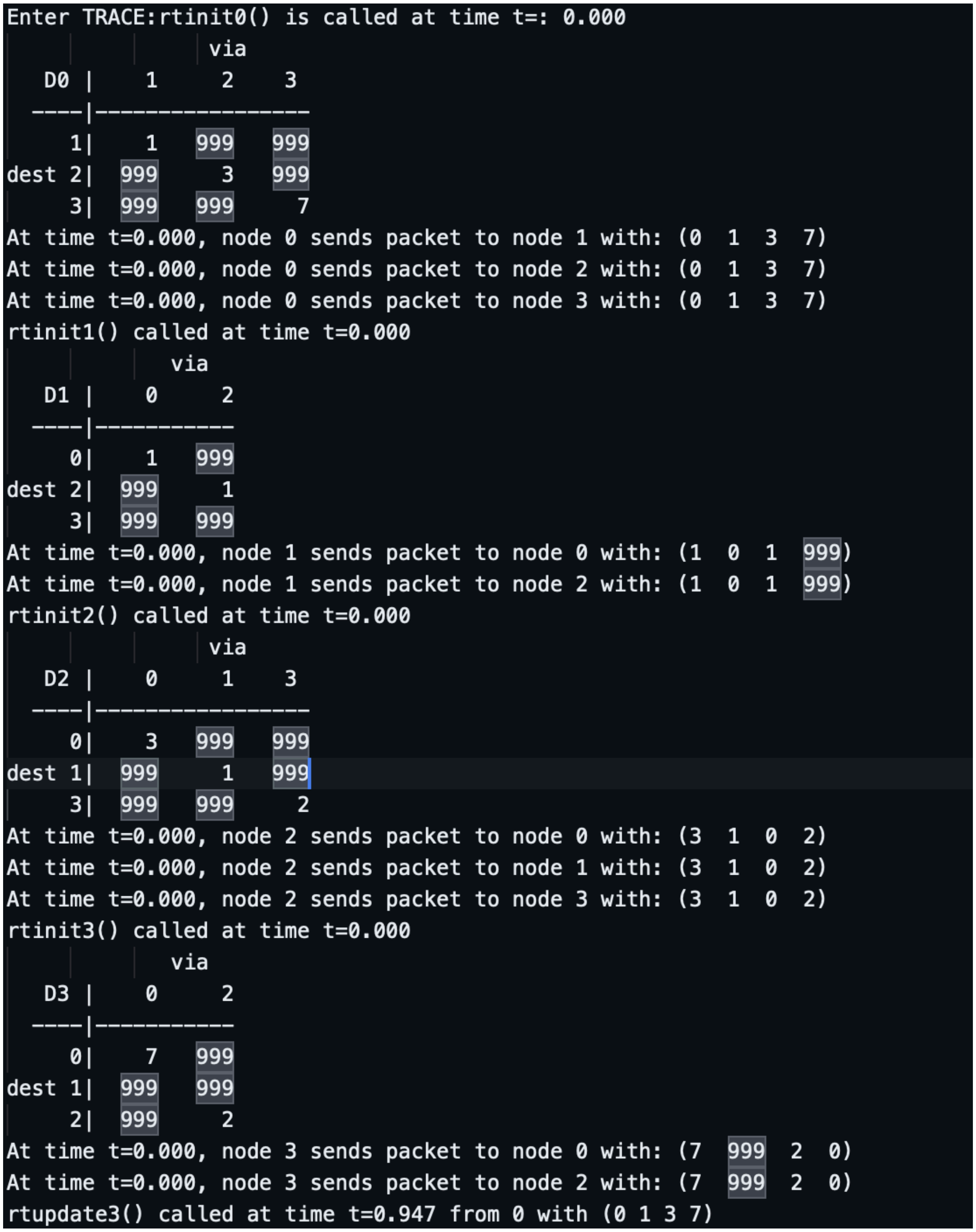
3. Link Change Handling (linkhandler0(), linkhandler1()):
    - At specific simulation times (t=10000 and t=20000), the cost of the link between Node 0 and Node 1 is modified.
    - The linkhandler functions are called to update the local cost information at the affected nodes (Node 0 and Node 1).
    - These changes in link cost trigger subsequent updates in the distance vectors as the nodes react to the altered network topology.

Results Analysis

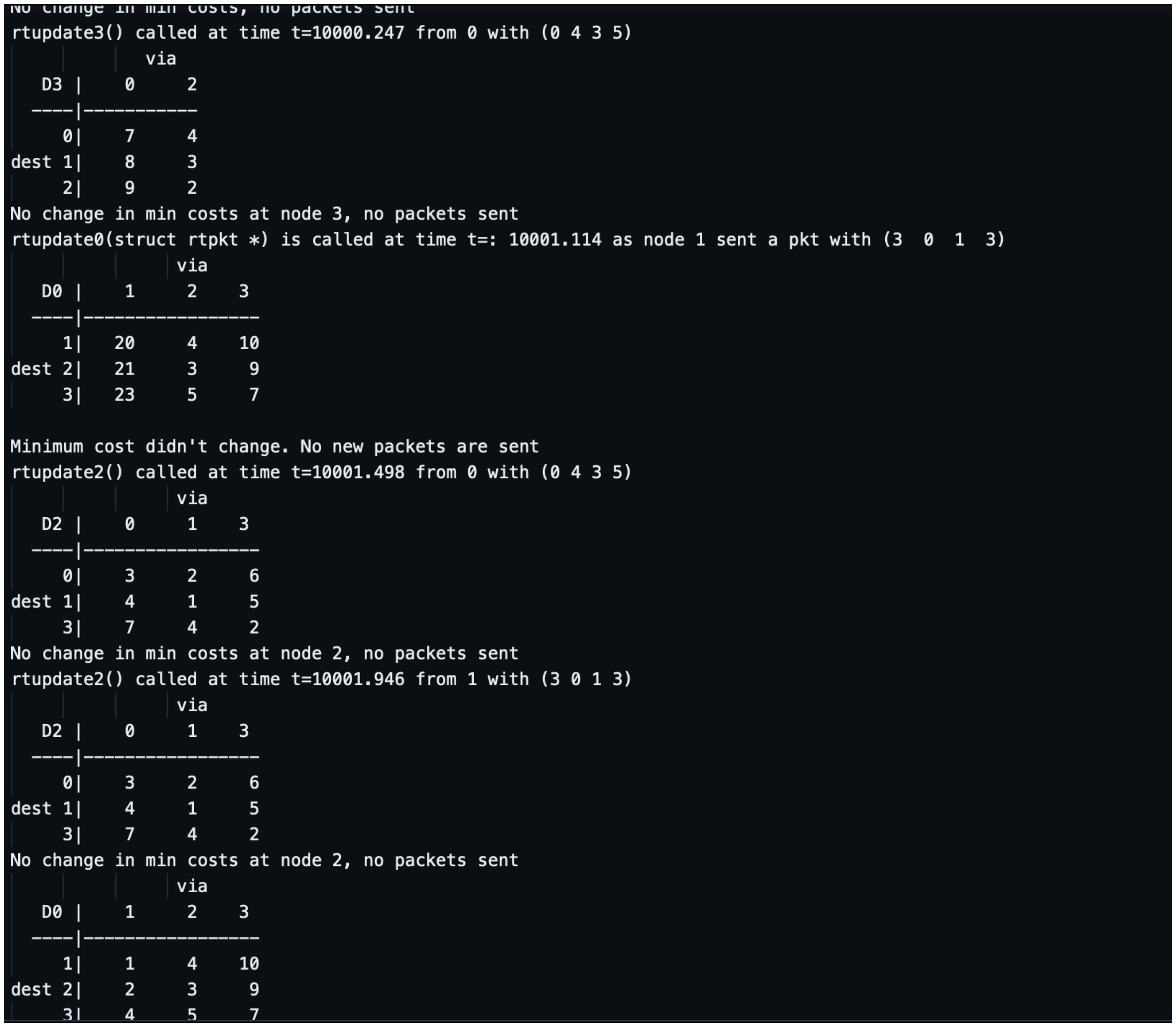
Analyze these key aspects from your trace files:

1. Initial State: From the trace (e.g., trace_level_2.txt), analyze the initial distance tables of each node after the rtinit() functions are executed. For Node 0, it should reflect the direct costs to its neighbors (1 to Node 1, 3 to Node 2, 7 to Node 3).
2. Convergence Process: Examine the sequence of routing packets exchanged between neighbors and how the distance tables evolve over time. Describe how nodes learn about paths to other nodes through intermediate hops. Note the number of iterations or time units it takes for the routing tables to stabilize.
3. Link Cost Changes:
    - At t=10000 (Link 0-1 cost changes to 20): Observe how the routing tables at Node 0 and Node 1 are immediately updated. Track how this change propagates through the network as nodes exchange updated distance vectors. Analyze how the routes to destinations that previously used the 0-1 link are affected and if alternative paths are discovered.
    - At t=20000 (Link 0-1 cost changes back to 1): Similarly, observe the immediate impact on Node 0 and Node 1 and the subsequent propagation of this change. Analyze how the routes readjust to potentially utilize the now lower-cost 0-1 link again.
4. Final Routes: After the network has converged following the second link change (at t > 20000), report the final minimum cost from each node to all other nodes. This information can be extracted from the "Minimum cost from" lines in your trace file.

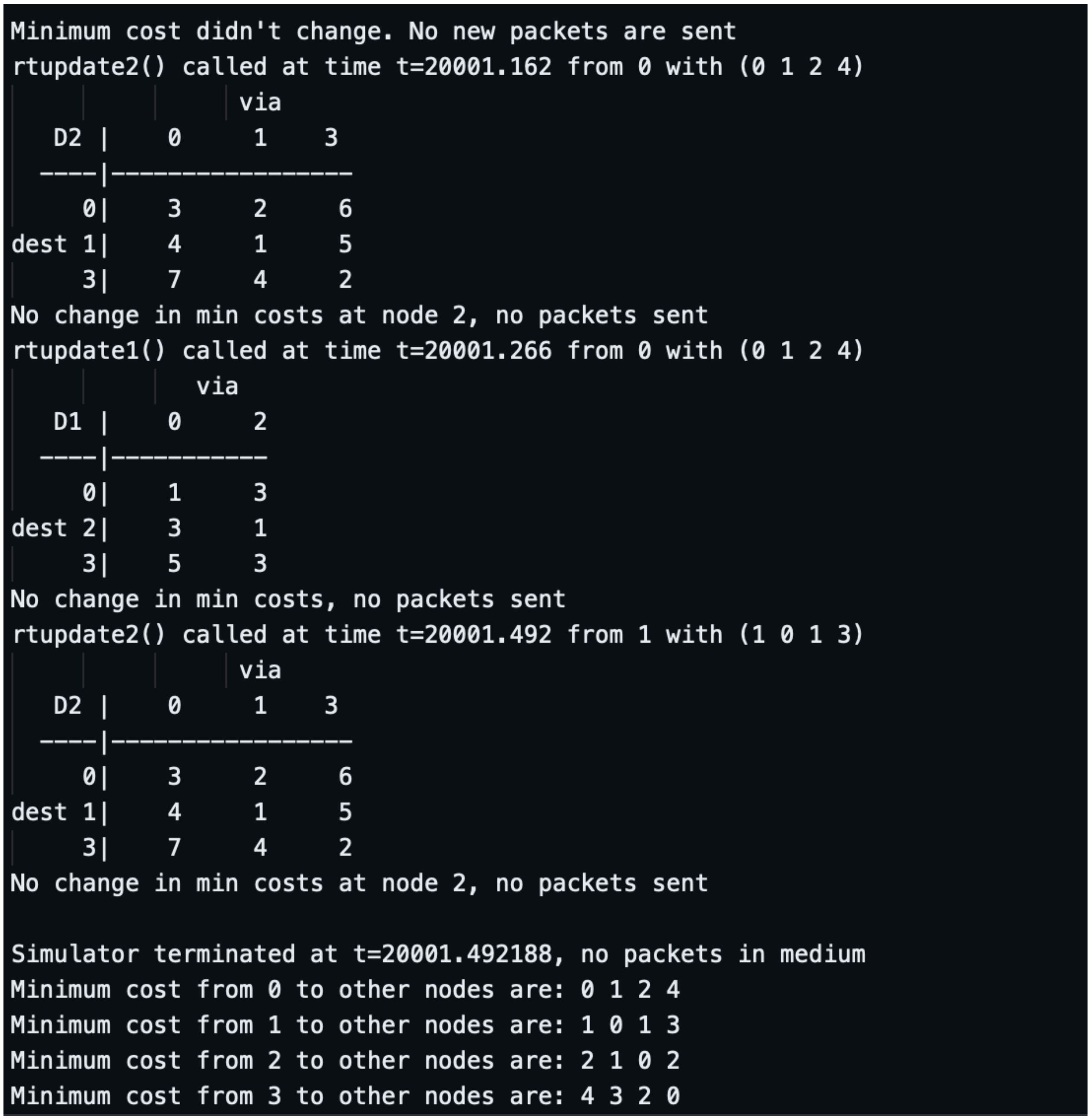
Relevant Screenshots



```
Enter TRACE: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() 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() 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() 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)
rtupdate3() called at time t=0.947 from 0 with (0 1 3 7)
```



```
No change in min costs, no packets sent
rtupdate3() called at time t=10000.247 from 0 with (0 4 3 5)
                 via
   D3 |    0     2
  ----|-----------
     0|    7     4
dest 1|    8     3
     2|    9     2
No change in min costs at node 3, no packets sent
rtupdate0(struct rtpkt *) is called at time t=: 10001.114 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
rtupdate2() called at time t=10001.498 from 0 with (0 4 3 5)
                    via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
     3|    7     4     2
No change in min costs at node 2, no packets sent
rtupdate2() called at time t=10001.946 from 1 with (3 0 1 3)
                    via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
     3|    7     4     2
No change in min costs at node 2, no packets sent
                    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
rtupdate2() called at time t=20001.162 from 0 with (0 1 2 4)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
     3|    7     4     2
No change in min costs at node 2, no packets sent
rtupdate1() called at time t=20001.266 from 0 with (0 1 2 4)
             via
   D1 |    0     2
  ----|-----------
     0|    1     3
dest 2|    3     1
     3|    5     3
No change in min costs, no packets sent
rtupdate2() called at time t=20001.492 from 1 with (1 0 1 3)
                via
   D2 |    0     1    3
  ----|-----------------
     0|    3     2     6
dest 1|    4     1     5
     3|    7     4     2
No change in min costs at node 2, no packets sent

Simulator terminated at t=20001.492188, no packets in medium
Minimum cost from 0 to other nodes are: 0 1 2 4
Minimum cost from 1 to other nodes are: 1 0 1 3
Minimum cost from 2 to other nodes are: 2 1 0 2
Minimum cost from 3 to other nodes are: 4 3 2 0
```

Include screenshots in your report to visually demonstrate:

- Successful compilation of the code.
- The initial state of the distance tables for each node (shortly after the simulation starts).
- Intermediate routing updates (if the trace level allows for clear visualization of packet exchanges and table updates).
- The state of the distance tables and minimum costs after the first link change (at t=10000) and after the network has reconverged.
- The final converged state of the distance tables and minimum costs after the second link change (at t=20000).

4. Problems Encountered and Solutions

Document any difficulties you faced during the implementation or analysis and the steps you took to resolve them. This could include:

- Compilation errors and their fixes.
- Issues in understanding the algorithm's behavior in the simulation.
- Challenges in extracting and interpreting the information from the trace files.
- Any discrepancies between the expected behavior based on the algorithm and the observed behavior in the simulation.

```
#!/bin/bash

# Create report directory
mkdir -p report

# Run simulation with different trace levels
echo "0" | ./prog3 > report/trace_level_0.txt
echo "1" | ./prog3 > report/trace_level_1.txt
echo "2" | ./prog3 > report/trace_level_2.txt

# Extract key results
echo "Final Routing Tables:" > report/analysis.txt
grep "Minimum cost from" report/trace_level_2.txt | tail -4 >> report/analysis.txt

echo -e "\nInitial Distance Tables:" >> report/analysis.txt
grep -A 15 "rtinit" report/trace_level_2.txt | grep -A 10 "Distance table" | head -40 >> report/analysis.txt

echo -e "\nLink Cost Changes:" >> report/analysis.txt
grep -A 20 "linkhandler" report/trace_level_2.txt >> report/analysis.txt

echo -e "\nMessage Overhead:" >> report/analysis.txt
echo "Total packets exchanged: $(grep -c "sends packet to node" report/trace_level_2.txt)" >>
report/analysis.txt

# Create message timing analysis (optional)
grep "sends packet to node" report/trace_level_2.txt | awk '{print $2}' | sed 's/t=//' | sort -n >
report/message_timing.txt

# Generate PDF report if pandoc is available (optional)
if command -v pandoc &> /dev/null; then
  pandoc README.md -o report/distance_vector_routing_report.pdf
fi
```

Save this script as generate_report.sh and make it executable with chmod +x generate_report.sh.

5. Conclusion

Summarize your findings from the simulation and analysis. Discuss the behavior of the distance vector routing algorithm in this specific network topology, its convergence properties, and its response to link cost changes. You can also discuss any limitations or potential issues observed, such as the possibility of count-to-infinity (though it might not be explicitly demonstrated in this small, stable topology).
For your submitted report, ensure you include:

- A README.md file containing your detailed analysis and explanations.
- The script(s) used to create and configure the network (in this case, the C code itself configures the network).
- All the test results requested for each of the questions (analyzed output from the trace files).
- Explanation of any problems encountered and the approach taken to resolve them.
- Relevant screenshots demonstrating successful connection, performance metrics (e.g., convergence time, message overhead), and routing tables at different stages of the simulation.