

NAMES: Larry Chiem, Ian Rowe, Raymond Shum, Nicholas Stankovich
DUE DATE: June 19, 2021
ASSIGNMENT: Team Programming Assignment #4
DESCRIPTION: Answers to Steps 1 to 5.

Programming Assignment 4

Note: Please feel free to use the included .pdf bookmarks to navigate between headers quickly.

1.0 – Network Diagram

Please refer to Appendix 1.0 (page 7) for the network diagram. We moved the image due to its size.

2.0 – Screen Capture – Program running without errors

```
mininet@mininet-vm:~/CST311/TeamProjects/P4$ ls
legacy_routers.mn  legacy_routers.py  linuxrouter.py
mininet@mininet-vm:~/CST311/TeamProjects/P4$ sudo python legacy_routers.py
*** Adding controller
*** Add switches
*** Add hosts
*** Add links
*** Starting network
*** Configuring hosts
r5 r4 r3 h1 h2
*** Starting controllers
*** Starting switches
*** Post configure switches and hosts
*** Adding static routes for R3, R4, R5
*** Starting CLI:
mininet>
```

3.0 – Screen Capture – Ping Test: h1 ping h2 and h2 ping h1

```
mininet> h1 ping h2 -c 5
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
64 bytes from 10.0.2.100: icmp_seq=1 ttl=61 time=0.448 ms
64 bytes from 10.0.2.100: icmp_seq=2 ttl=61 time=0.091 ms
64 bytes from 10.0.2.100: icmp_seq=3 ttl=61 time=0.089 ms
64 bytes from 10.0.2.100: icmp_seq=4 ttl=61 time=0.091 ms
64 bytes from 10.0.2.100: icmp_seq=5 ttl=61 time=0.089 ms

--- 10.0.2.100 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4019ms
rtt min/avg/max/mdev = 0.089/0.161/0.448/0.143 ms
```

```
mininet> h2 ping h1 -c 5
PING 10.0.1.100 (10.0.1.100) 56(84) bytes of data.
64 bytes from 10.0.1.100: icmp_seq=1 ttl=61 time=6.17 ms
64 bytes from 10.0.1.100: icmp_seq=2 ttl=61 time=0.915 ms
64 bytes from 10.0.1.100: icmp_seq=3 ttl=61 time=0.264 ms
64 bytes from 10.0.1.100: icmp_seq=4 ttl=61 time=0.089 ms
64 bytes from 10.0.1.100: icmp_seq=5 ttl=61 time=0.091 ms

--- 10.0.1.100 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4017ms
rtt min/avg/max/mdev = 0.089/1.507/6.176/2.354 ms
mininet>
```

4.0 – List of Changed Lines & Why

In refactoring the provided code, several lines have been changed due to implementation of the network, commenting, adding the header and readability. Due to the large number of changes, we have included screenshots of the sections that we've altered. Comments have been included for further clarity.

4.1 – Add switches

```
50      info('*** Add switches\n')
51      # [Change] Moved switches ahead of routers as per instructions
52      s2 = net.addSwitch('s2', cls=OVSKernelSwitch)
53      s1 = net.addSwitch('s1', cls=OVSKernelSwitch)
54
55      # [Change] Set IP address and mask of r5-eth0
56      r5 = net.addHost('r5', cls=Node, ip='10.0.2.1/24')
57      r5.cmd('sysctl -w net.ipv4.ip_forward=1')
58
59      # [Change] Set IP address and mask of r4-eth0
60      r4 = net.addHost('r4', cls=Node, ip='192.168.1.2/30')
61      r4.cmd('sysctl -w net.ipv4.ip_forward=1')
62
63      # [Change] Set IP address and mask of r3-eth0
64      r3 = net.addHost('r3', cls=Node, ip='10.0.1.1/24')
65      r3.cmd('sysctl -w net.ipv4.ip_forward=1')
```

Lines 52 and 53: As per instructions, moved switch instantiations ahead of router instantiations.

Line 56: Set r5-eth0 IP address and subnet mask (changed from 0.0.0.0)

Line 60: Set r4-eth0 IP address and subnet mask (changed from 0.0.0.0)

Line 64: Set r3-eth0 IP address and subnet mask (changed from 0.0.0.0)

4.2 – Add hosts

```
67      info('*** Add hosts\n')
68
69      # [Change] Set IP address, mask and default route of h1 and h2
70      h1 = net.addHost('h1', cls=Host, ip='10.0.1.100/24',
71                      defaultRoute='via 10.0.1.1')
72      h2 = net.addHost('h2', cls=Host, ip='10.0.2.100/24',
73                      defaultRoute='via 10.0.2.1')
```

Lines 70 and 71: Set IP address, subnet mask and default route for h1 (changed from 10.0.0.1, none)

Lines 72 and 73: Set IP address, subnet mask and default route for h2 (changed from 10.0.0.2, none)

4.3 – Add links

```
75     info('*** Add links\n')
76     net.addLink(h1, s1)
77     net.addLink(h2, s2)
78
79     net.addLink(s2, r5)
80     net.addLink(s1, r3)
81
82     # [Change] Set IP address and mask of r3-eth1
83     net.addLink(r3, r4, intfName1='r3-eth1',
84                 params1={'ip': '192.168.1.1/30'})
85
86     # [Change] Set IP address and mask of r4-eth1 and r5-eth1
87     net.addLink(r4, r5, intfName1='r4-eth1',
88                 params1={'ip': '192.168.1.5/30'},
89                 intfName2='r5-eth1',
90                 params2={'ip': '192.168.1.6/30'})
```

Lines 83 and 84: Set IP address and subnet mask for r3-eth1

Lines 87 to 90: Set IP address and subnet mask for r4-eth1 and r5-eth1

4.4 – Adding static routes

```
103     info('*** Post configure switches and hosts\n')
104
105     # [Change] Added section to configure routing tables of r3, r4, r5
106     info('*** Adding static routes for R3, R4, R5\n')
107
108     # Configured routing table of r3
109     # Added route to 10.0.2.0/24 (subnet of r5-eth0 and h2-eth0)
110     # Added route to 192.168.1.4/30 (subnet of r4-eth1 and r5-eth1)
111     r3.cmd('ip route add 10.0.2.0/24 via 192.168.1.2 dev r3-eth1')
112     r3.cmd('ip route add 192.168.1.4/30 via 192.168.1.2 dev r3-eth1')
113
114     # Configured routing table of r4
115     # Added route to 10.0.2.0/24 (subnet of r5-eth0 and h2-eth0)
116     # Added route to 10.0.1.0/24 (subnet of h1-eth0 and r3-eth0)
117     r4.cmd('ip route add 10.0.2.0/24 via 192.168.1.6 dev r4-eth1')
118     r4.cmd('ip route add 10.0.1.0/24 via 192.168.1.1 dev r4-eth0')
119
120     # Configured routing table of r5
121     # Added route to 10.0.1.0/24 (subnet of r5-eth0 and h2-eth0)
122     # Added route to 192.168.1.0/30 (subnet of r3-eth1 and r4-eth0)
123     r5.cmd('ip route add 10.0.1.0/24 via 192.168.1.5 dev r5-eth1')
124     r5.cmd('ip route add 192.168.1.0/30 via 192.168.1.5 dev r5-eth1')
```

We added this section to consolidation the configuration of static routes for r3, r4 and r5.

Lines 111 and 112: For r3, added routes to 10.0.2.0/24 and 192.168.1.4/30 via interface r3-eth1.

Lines 117 and 118: For r4, added routes to 10.0.2.0/24 via interface r4-eth1 and 10.0.1.0/24 via interface r4-eth0.

Lines 123 and 124: For r5, added routes to 10.0.1.0/24 and 192.168.1.0/30 via interface r5-eth1.

5.0 – Four-Part Question

5.1 – Part A – What were any interesting findings and lessons learned?

Though not limited to this project, we learned the importance of clearly defining roles and regular communication between members. Clearly defined responsibilities allow for work to be divided equitably among members. Communication allows for immediate resolution of blockers.

Specific to this project, designing the network prior to implementation was critical for success. This was due to our unfamiliarity with the Mininet API. Correctly designing the network prior to implementation allowed us to test for errors in implementation without having to second guess the design.

An interesting finding relates to a discussion that one member had with Dr. Satyavolu regarding the Mininet API. If the commands to configure routing tables is issued immediately after router instantiation, these changes do not appear to persist after the topology is built (during the interactive Mininet CLI session).

5.2 – Part B – Why didn't the original program forward packets between hosts?

The original program did not forward packets between hosts because of several reasons relating to the initial configuration of the topology.

First, h1 and h2 did not have subnet masks or default routes configured. Neither host would know whether the other was on the same subnet. Even if this was not so, neither host would know where to forward packets if the destination host was on a different subnet.

Second, r3-eth0 and r5-eth0 (the default gateways for h1 and h2, respectively) did not have IP addresses and subnet masks configured to place the interfaces on the same subnet as h1 and h2. This meant that IP datagrams could not be sent to them from either host.

Third, none of the router interfaces on the path between r3-eth0 and r5-eth0 were configured with IP addresses or subnet masks. These include the interfaces r3-eth1, r4-eth0, r4-eth1 and r5-eth1. Routers operate at Layer 3, and the absence of Layer 3 addresses (IP) meant that routers would not be able to send or receive IP datagrams on these interfaces.

Fourth, none of the routing tables for r3, r4 and r5 were configured. Generalizing the end goal of the assignment (h1 must ping h2 and vice versa), r3 must know that there is a path to h2's subnet through r4 and r4 must know that there is a path to h2's subnet through r5. On the reverse path, r5 must know that there is a path to h1's subnet through r4 and r4 must know there is a path to h1's subnet through r3.

5.3 – Part C – Is the line ' r3.cmd('sysctl -w net.ipv4.ip_forward=1') ' required?

Yes, this line is required. It allows r3 to act as a router by enabling the forwarding of IPv4 packets.

5.4 – Part D – Intentionally break your working program. Explain why your change caused the network to break.

For this question, we've broken the network by removing a routing table entry for r4, the path to 10.0.2.0/24:

```
# Configured routing table of r4
# Added route to 10.0.2.0/24 (subnet of r5-eth0 and h2-eth0)
# Added route to 10.0.1.0/24 (subnet of h1-eth0 and r3-eth0)
# r4.cmd('ip route add 10.0.2.0/24 via 192.168.1.6 dev r4-eth1')
r4.cmd('ip route add 10.0.1.0/24 via 192.168.1.1 dev r4-eth0')
```

After building the topology, h1 can no longer ping h2:

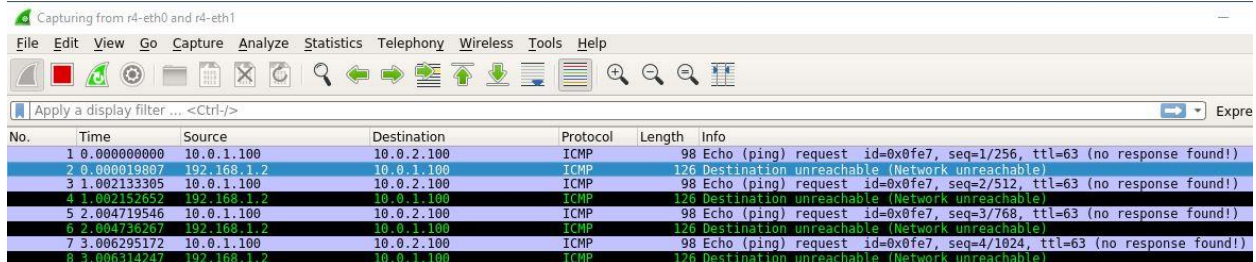
```
mininet> h1 ping h2
PING 10.0.2.100 (10.0.2.100) 56(84) bytes of data.
From 192.168.1.2 icmp_seq=1 Destination Net Unreachable
From 192.168.1.2 icmp_seq=2 Destination Net Unreachable
From 192.168.1.2 icmp_seq=3 Destination Net Unreachable
^C
--- 10.0.2.100 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2002ms
```

Reviewing r4's routing table, we see that the route to 10.0.2.0/24 no longer exists:

```
mininet> r4 route
QXcbConnection: Could not connect to display localhost:10.0
Kernel IP routing table
Destination        Gateway         Genmask         Flags Metric Ref    Use Iface
10.0.1.0           192.168.1.1    255.255.255.0   UG    0      0      0 r4-eth0
192.168.1.0        *              255.255.255.252 U      0      0      0 r4-eth0
192.168.1.4        *              255.255.255.252 U      0      0      0 r4-eth1
```

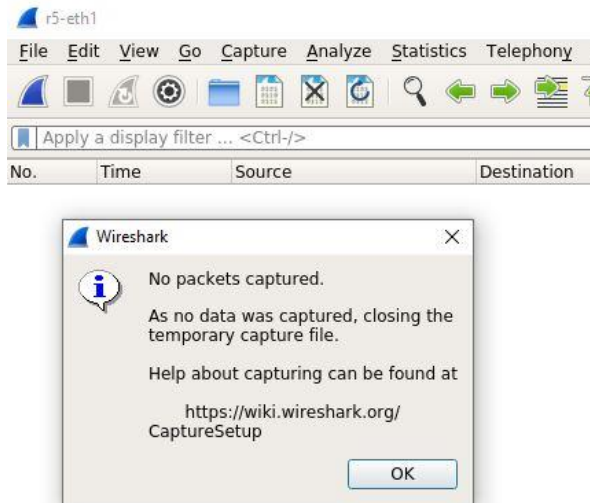
Why?: h1 is no longer able to ping h2 because r4 no longer has a path to the 10.0.2.0/24 subnet. It does not know that it exists through the interface r4-eth1. The packet with the destination address 10.0.2.100 arrives at router r4, which finds that it has no matching entries in the routing table. It drops the packet and returns an ICMP message to the source stating that the destination network is unreachable.

We can see this through the Wireshark capture of the r4-eth0 and r4-eth1 interfaces:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.1.100	10.0.2.100	ICMP	98	Echo (ping) request id=0x0fe7, seq=1/256, ttl=63 (no response found!)
2	0.000019807	192.168.1.2	10.0.1.100	ICMP	126	Destination unreachable (Network unreachable)
3	1.002133305	10.0.1.100	10.0.2.100	ICMP	98	Echo (ping) request id=0x0fe7, seq=2/512, ttl=63 (no response found!)
4	1.002152652	192.168.1.2	10.0.1.100	ICMP	126	Destination unreachable (Network unreachable)
5	2.004719546	10.0.1.100	10.0.2.100	ICMP	98	Echo (ping) request id=0x0fe7, seq=3/768, ttl=63 (no response found!)
6	2.004729257	192.168.1.2	10.0.1.100	ICMP	126	Destination unreachable (Network unreachable)
7	3.006295172	10.0.1.100	10.0.2.100	ICMP	98	Echo (ping) request id=0x0fe7, seq=4/1024, ttl=63 (no response found!)
8	3.006314247	192.168.1.2	10.0.1.100	ICMP	126	Destination unreachable (Network unreachable)

The ICMP message from h1 arrives successfully at r4. However, r4 is unable to forward the packet to r5 due to the missing routing table entry. We can confirm that r5 has not received any ICMP messages through a capture on r5-eth1:



Thus, we confirm that by removing the route to 10.0.2.0/24 on r4, r4 is then unable to forward a packet to r5 through its directly connected interface, r5-eth1. This breaks the network and h1 is no longer able to ping h2.

Appendix 1.0 – Network Diagram

