Computer Networks HW#3

Submitted By: Neeha Hammad

Problem 3.1

(a) Python file “partA.py”

```
mininet> h1 ping6 2001:638:709:a::1
PING 2001:638:709:a::1(2001:638:709:a::1) 56 data bytes
64 bytes from 2001:638:709:a::1: icmp_seq=1 ttl=64 time=0.084 ms
64 bytes from 2001:638:709:a::1: icmp_seq=2 ttl=64 time=0.072 ms
64 bytes from 2001:638:709:a::1: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 2001:638:709:a::1: icmp_seq=4 ttl=64 time=0.069 ms
^C
--- 2001:638:709:a::1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3073ms
rtt min/avg/max/mdev = 0.067/0.073/0.084/0.006 ms
```

```
mininet> h1 ping6 2001:638:709:a::f
PING 2001:638:709:a::f(2001:638:709:a::f) 56 data bytes
64 bytes from 2001:638:709:a::f: icmp_seq=1 ttl=64 time=0.126 ms
64 bytes from 2001:638:709:a::f: icmp_seq=2 ttl=64 time=0.128 ms
64 bytes from 2001:638:709:a::f: icmp_seq=3 ttl=64 time=0.171 ms
64 bytes from 2001:638:709:a::f: icmp_seq=4 ttl=64 time=0.111 ms
64 bytes from 2001:638:709:a::f: icmp_seq=5 ttl=64 time=0.110 ms
64 bytes from 2001:638:709:a::f: icmp_seq=6 ttl=64 time=0.113 ms
^C
--- 2001:638:709:a::f ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5117ms
rtt min/avg/max/mdev = 0.110/0.126/0.171/0.023 ms
```

```
mininet> h1 ping6 2001:638:709:f::1
PING 2001:638:709:f::1(2001:638:709:f::1) 56 data bytes
64 bytes from 2001:638:709:f::1: icmp_seq=1 ttl=64 time=0.113 ms
64 bytes from 2001:638:709:f::1: icmp_seq=2 ttl=64 time=0.109 ms
64 bytes from 2001:638:709:f::1: icmp_seq=3 ttl=64 time=0.114 ms
^C
--- 2001:638:709:f::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.109/0.112/0.114/0.002 ms
```

```
mininet> h1 ping6 2001:638:709:b::1
PING 2001:638:709:b::1(2001:638:709:b::1) 56 data bytes
64 bytes from 2001:638:709:b::1: icmp_seq=1 ttl=62 time=0.259 ms
64 bytes from 2001:638:709:b::1: icmp_seq=2 ttl=62 time=0.167 ms
64 bytes from 2001:638:709:b::1: icmp_seq=3 ttl=62 time=0.171 ms
64 bytes from 2001:638:709:b::1: icmp_seq=4 ttl=62 time=0.169 ms
64 bytes from 2001:638:709:b::1: icmp_seq=5 ttl=62 time=0.177 ms
64 bytes from 2001:638:709:b::1: icmp_seq=6 ttl=62 time=0.111 ms
^C
--- 2001:638:709:b::1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5118ms
rtt min/avg/max/mdev = 0.111/0.175/0.259/0.045 ms
```

```
mininet> r1 ping6 2001:638:709:a::1
PING 2001:638:709:a::1(2001:638:709:a::1) 56 data bytes
64 bytes from 2001:638:709:a::1: icmp_seq=1 ttl=64 time=0.114 ms
64 bytes from 2001:638:709:a::1: icmp_seq=2 ttl=64 time=0.111 ms
64 bytes from 2001:638:709:a::1: icmp_seq=3 ttl=64 time=0.114 ms
64 bytes from 2001:638:709:a::1: icmp_seq=4 ttl=64 time=0.110 ms
^C
--- 2001:638:709:a::1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3052ms
rtt min/avg/max/mdev = 0.110/0.112/0.114/0.007 ms
```

```
mininet> r2 ping6 2001:638:709:b::1
PING 2001:638:709:b::1(2001:638:709:b::1) 56 data bytes
64 bytes from 2001:638:709:b::1: icmp_seq=1 ttl=64 time=0.198 ms
64 bytes from 2001:638:709:b::1: icmp_seq=2 ttl=64 time=0.112 ms
64 bytes from 2001:638:709:b::1: icmp_seq=3 ttl=64 time=0.109 ms
^C
--- 2001:638:709:b::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.109/0.139/0.198/0.043 ms


mininet> r2 traceroute 2001:638:709:b::1
traceroute to 2001:638:709:b::1 (2001:638:709:b::1), 30 hops max, 80 byte packets
 1  2001:638:709:b::1 (2001:638:709:b::1)  0.250 ms  0.041 ms  0.033 ms


mininet> r2 traceroute 2001:638:709:a::1
traceroute to 2001:638:709:a::1 (2001:638:709:a::1), 30 hops max, 80 byte packets
 1  2001:638:709:f::1 (2001:638:709:f::1)  0.269 ms  0.062 ms  0.060 ms
 2  2001:638:709:a::1 (2001:638:709:a::1)  0.264 ms  0.080 ms  0.099 ms


traceroute to 2001:638:709:a::1 (2001:638:709:a::1), 30 hops max, 80 byte packets
 1  2001:638:709:b::f (2001:638:709:b::f)  0.048 ms  0.013 ms  0.011 ms
 2  2001:638:709:f::1 (2001:638:709:f::1)  0.027 ms  0.016 ms  0.015 ms
 3  2001:638:709:a::1 (2001:638:709:a::1)  0.028 ms  0.021 ms  0.021 ms

mininet> h1 traceroute 2001:638:709:f::1
traceroute to 2001:638:709:f::1 (2001:638:709:f::1), 30 hops max, 80 byte packets
 1  2001:638:709:f::1 (2001:638:709:f::1)  0.171 ms  0.048 ms  0.054 ms

mininet> h1 traceroute 2001:638:709:b::f
traceroute to 2001:638:709:b::f (2001:638:709:b::f), 30 hops max, 80 byte packets
 1  2001:638:709:a::f (2001:638:709:a::f)  0.116 ms  0.041 ms  0.034 ms
 2  2001:638:709:b::f (2001:638:709:b::f)  0.074 ms  0.050 ms  0.047 ms

mininet> h1 traceroute 2001:638:709:b::1
traceroute to 2001:638:709:b::1 (2001:638:709:b::1), 30 hops max, 80 byte packets
 1  2001:638:709:a::f (2001:638:709:a::f)  0.120 ms  0.040 ms  0.034 ms
 2  2001:638:709:f::2 (2001:638:709:f::2)  0.071 ms  0.053 ms  0.051 ms
 3  2001:638:709:b::1 (2001:638:709:b::1)  0.086 ms  0.067 ms  0.067 ms
```

Everything looks good!

(b) Python file “partB.py”

```
mininet> h1 traceroute 2001:638:709:b::1
traceroute to 2001:638:709:b::1 (2001:638:709:b::1), 30 hops max, 80 byte packets
 1  2001:638:709:a::f (2001:638:709:a::f)  0.337 ms  0.076 ms  0.065 ms
 2  2001:638:709:f::2 (2001:638:709:f::2)  0.193 ms  0.055 ms  0.049 ms
 3  2001:638:709:b::1 (2001:638:709:b::1)  0.186 ms  0.072 ms  0.064 ms
```

We know that 2001:638:709:a::f corresponds to r1-eth0 and 2001:638:709:f::2 corresponds to r2-eth0. Now, we know that traffic from h1 flows via r1 and r2 to h2. Similarly,

```
mininet> h2 traceroute 2001:638:709:a::1
traceroute to 2001:638:709:a::1 (2001:638:709:a::1), 30 hops max, 80 byte packets
 1  2001:638:709:b::e (2001:638:709:b::e)  0.311 ms  0.052 ms  0.034 ms
 2  2001:638:709:e::1 (2001:638:709:e::1)  0.125 ms  0.054 ms  0.073 ms
 3  2001:638:709:a::1 (2001:638:709:a::1)  0.102 ms  0.072 ms  0.088 ms
```

where (2001:638:709:b::e) corresponds to r4-eth1 and (2001:638:709:e::1) corresponds to r3-eth1. Now, we know that traffic from h2 flows via r4 and r3 to h1.

Asymmetric routing is the situation where packets from A to B follow a different path than packets from B to A. As we can see, the path from h1 to h2 is different from the path from h2 to h1. Thus, we can conclude that the traffic between h1 and h2 is using an asymmetric path.

(c) First, let’s check the default MTU on h1 and h2.

```
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::cc30:78ff:feca:b5c0  prefixlen 64  scopeid 0x20<link>
        inet6 2001:638:709:a::1  prefixlen 64  scopeid 0x0<global>
        ether ce:30:78:ca:b5:c0  txqueuelen 1000  (Ethernet)
        RX packets 61  bytes 7530 (7.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 11  bytes 982 (982.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
mininet> h2 ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 2001:638:709:b::1  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::a004:e2ff:fe75:4ae7  prefixlen 64  scopeid 0x20<link>
        ether a2:04:e2:75:4a:e7  txqueuelen 1000  (Ethernet)
        RX packets 70  bytes 8304 (8.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 1052 (1.0 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The link between r1 and r2 has r1-eth1 and e2-eth0. We manually change the maximum transmission unit (MTU), using these commands:

```
mininet> r1 ifconfig r1-eth1 mtu 1400
mininet> r2 ifconfig r2-eth0 mtu 1400
```

Now, let's ping on one terminal using:

```
h1 ping -s 65000 2001:638:709:b::1
```

After pinging, we can run the following command in a separate terminal (eg xterm):

```
h1 tcpdump -n -i h1-eth0
```

I am unable to copy the output from xterm, but it said that the "Packet too big: mtu=1400" and "capture size 65000 bytes" The length shown was 16 and it also said that ip6-allrouters: ICMP6 so we can see that the ICMP6 protocol is being used.

Only 3 packets were captured and 3 were received by filter (screenshot attached) in a very large time. So this also shows that the packet was indeed too big.

References:
https://support.purevpn.com/how-to-change-mtu-value-on-linux
https://www.hugeserver.com/kb/install-use-tcpdump-capture-packets/

(d)

```
mininet> h1 tracepath 2001:638:709:b::1
 1?: [LOCALHOST]                        0.035ms pmtu 1500
 1:  2001:638:709:a::f                          0.157ms
 1:  2001:638:709:a::f                          0.113ms
 2:  2001:638:709:f::2                          0.143ms
 3:  2001:638:709:b::1                          0.160ms reached
        Resume: pmtu 1500 hops 3 back 3
```

Path MTU discovery is something that the network stack does automatically on every connection. This is because sending full-sized frames is more efficient than using fragmentation.

This is done by sending large packets with the "Don't Fragment" bit set. In case any hop's router sends back the ICMP error message saying that "Destination Unreachable: Fragmentation Required but 'Don't Fragment' bit set", that too indicates what its MTU is. In simpler words, when we use ping, if the packet’s MTU is too big, it will be dropped. An error message is returned, along with the MTU value.

Tracepath traces the path to the *destination,* discovering MTU along this path using TTL (Time To Live) values to find the smallest MTU. It also uses ICMP6 and UDP port *port* or some random port. Similar to ping, in case of an error (eg when the MTU is too big), the packet is dropped. Once it’s dropped, we can see the MTU value.

References:
https://superuser.com/questions/1537024/how-can-ping-tracepath-derive-the-path-mtu-info/1537027#1537027
https://linux.die.net/man/8/tracepath
http://www.rfc-editor.org/rfc/rfc1191.txt

(e) As mentioned in the previous part, path discovery is done by sending large packets with the "Don't Fragment" bit set. Any device along the path whose MTU is smaller than the packet will drop it, and send back an error message that contains its MTU, allowing the source host to reduce its Path MTU appropriately. The process is repeated until the MTU is small enough to traverse the entire path without fragmentation. This suggests that it doesn't have to be repeated after the small one is found. Once it's found, it can be stored in the routing cache, also known as the forwarding information base (FIB).

Each node along the route has the responsibility to respond, with a "packet too large, please fragment", unless the "Don't Fragment" flag is set in the packet.

```
mininet> h1 tracepath -n 2001:638:709:b::1

 1?: [LOCALHOST]                      0.034ms pmtu 1500

 1:  2001:638:709:a::f                          0.236ms

 1:  2001:638:709:a::f                          0.067ms

 2:  2001:638:709:f::2                          0.188ms

 3:  2001:638:709:b::1                          0.317ms reached

         Resume: pmtu 1500 hops 3 back 3

mininet> h1 tracepath -n 2001:638:709:b::1

 1?: [LOCALHOST]                      0.038ms pmtu 1500

 1:  2001:638:709:a::f                          0.145ms

 1:  2001:638:709:a::f                          0.068ms

 2:  2001:638:709:f::2                          0.094ms

 3:  2001:638:709:b::1                          0.133ms reached

         Resume: pmtu 1500 hops 3 back 3
```

Reference:
https://en.wikipedia.org/wiki/Path_MTU_Discovery
https://unix.stackexchange.com/questions/576991/is-mtu-path-discovery-repeated-everytime-we-use-tracepath?noredirect=1#comment1073658_576991
http://linux-ip.net/html/routing-cache.html