Computer Network Assignment 3

Yiping Deng

May 10, 2019

Problem 3

a)

Using topology defined in Listing 2, we can execute the ping and traceroute in mininet and verify the topology:

Listing 1: Mininet output for a)

```
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=0.125 ms
64 bytes from 2001:638:709:b::1: icmp_seq=2 ttl=62 time=0.060 ms
64 bytes from 2001:638:709:b::1: icmp_seq=3 ttl=62 time=0.060 ms
^C
--- 2001:638:709:b::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.060/0.081/0.125/0.032 ms


mininet> h2 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=62 time=0.058 ms
64 bytes from 2001:638:709:a::1: icmp_seq=2 ttl=62 time=0.062 ms
64 bytes from 2001:638:709:a::1: icmp_seq=3 ttl=62 time=0.073 ms
64 bytes from 2001:638:709:a::1: icmp_seq=4 ttl=62 time=0.064 ms
64 bytes from 2001:638:709:a::1: icmp_seq=5 ttl=62 time=0.060 ms
^C
--- 2001:638:709:a::1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4087ms
rtt min/avg/max/mdev = 0.058/0.063/0.073/0.008 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)  0.033 ms  0.018 ms  0.008 ms
 2  2001:638:709:f::2 (2001:638:709:f::2)  0.018 ms  0.009 ms  0.009 ms
 3  2001:638:709:b::1 (2001:638:709:b::1)  0.019 ms  0.010 ms  0.010 ms



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::f (2001:638:709:b::f)  0.030 ms  0.007 ms  0.007 ms
 2  2001:638:709:f::1 (2001:638:709:f::1)  0.019 ms  0.009 ms  0.008 ms
 3  2001:638:709:a::1 (2001:638:709:a::1)  0.019 ms  0.010 ms  0.010 ms
```

Listing 2: Basic topology network

```
#!/usr/bin/env python
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.nodelib import LinuxBridge
from mininet.log import setLogLevel

def set_ipv6(host, interface, ip):
    return host.cmd("ip -6 addr add " + ip + " dev " + interface)

def set_ipv6_gateway(host, gateway):
    return host.cmd("ip -6 route add default via " + gateway)

def set_ipv6_device_route(host, subnet, interface):
    return host.cmd("ip -6 route add " + subnet + " dev " + interface)

def set_ipv6_gateway_route(host, subnet, gateway, interface):
    # useful for the forward table
    return host.cmd("ip -6 route add " + subnet +
            " via " + gateway + " dev " + interface)



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

    net = Mininet(switch=LinuxBridge, controller=None)

    h1 = net.addHost('h1', ip=None)

    # create rest of the topology here
    s1 = net.addSwitch('s1')
    r1 = net.addHost('r1', ip = None)
    s0 = net.addSwitch('s0')
    r2 = net.addHost('r2', ip = None)
    s2 = net.addSwitch('s2')
    h2 = net.addHost('h2', ip = None)

    # addr info
    h1_eth0_addr = "2001:638:709:a::1/64"
    r1_eth0_addr = "2001:638:709:a::f/64"
    r1_eth1_addr = "2001:638:709:f::1/64"
    r2_eth0_addr = "2001:638:709:f::2/64"
    r2_eth1_addr = "2001:638:709:b::f/64"
    h2_eth0_addr = "2001:638:709:b::1/64"

    # subnet info
    subnet1_addr = "2001:638:709:a::/64"
    subnet2_addr = "2001:638:709:f::/64"
    subnet3_addr = "2001:638:709:b::/64"

    # link the router and switches
    net.addLink(h1, s1)
    net.addLink(s1, r1)
```

```
55      net.addLink(r1, s0)
56      net.addLink(s0, r2)
57      net.addLink(r2, s2)
58      net.addLink(s2, h2)
59
60      # configure IPv6 addresses and forwarding table entries here
61
62      # set ipv6 addr and gateway
63      print "setup ipv6"
64      print set_ipv6(h1, "h1-eth0", h1_eth0_addr)
65      print set_ipv6(r1, "r1-eth0", r1_eth0_addr)
66      print set_ipv6(r1, "r1-eth1", r1_eth1_addr)
67      print set_ipv6(r2, "r2-eth0", r2_eth0_addr)
68      print set_ipv6(r2, "r2-eth1", r2_eth1_addr)
69      print set_ipv6(h2, "h2-eth0", h2_eth0_addr)
70      print "done setup ipv6"
71
72      # setup the local route
73      print "setup forward rules"
74      print set_ipv6_device_route(h1, subnet1_addr, "h1-eth0")
75      print set_ipv6_device_route(r1, subnet1_addr, "r1-eth0")
76      print set_ipv6_device_route(r1, subnet2_addr, "r1-eth1")
77      print set_ipv6_device_route(r2, subnet2_addr, "r2-eth0")
78      print set_ipv6_device_route(r2, subnet3_addr, "r2-eth1")
79      print set_ipv6_device_route(h2, subnet3_addr, "h2-eth0")
80      print "done setup forward rules"
81
82      # setup the gateway
83      print "setup gateway"
84      print set_ipv6_gateway(h1, r1_eth0_addr[:-3])
85      print set_ipv6_gateway(h2, r2_eth1_addr[:-3])
86      print "done setup gateway"
87
88      # setup router forward table
89      print "setup router forward table"
90      print set_ipv6_gateway_route(r1, subnet3_addr, r2_eth0_addr[:-3], "r1-eth1")
91      print set_ipv6_gateway_route(r2, subnet1_addr, r1_eth1_addr[:-3], "r2-eth0")
92      print "done setup router forward table"
93
94      # router forwarding
95      print "enable forwarding for routers"
96      r1.cmd("sysctl -w net.ipv6.conf.all.forwarding=1")
97      r2.cmd("sysctl -w net.ipv6.conf.all.forwarding=1")
98      print "done enabling forwarding for routers"
99
100     print h1.cmd("ip -V")
101
102
103     net.start()
104     CLI(net)
105     net.stop()
```

b)

Using the modified Listing 4, we can use the traceroute to show the asymetric path:

Listing 3: Traceroute of b)

```
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)  0.042 ms  0.009 ms  0.006 ms
 2  2001:638:709:f::2 (2001:638:709:f::2)  0.052 ms  0.011 ms  0.008 ms
 3  2001:638:709:b::1 (2001:638:709:b::1)  0.117 ms  0.013 ms  0.010 ms


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)  0.033 ms  0.008 ms  0.007 ms
 2  2001:638:709:e::1 (2001:638:709:e::1)  0.054 ms  0.011 ms  0.009 ms
 3  2001:638:709:a::1 (2001:638:709:a::1)  0.025 ms  0.010 ms  0.010 ms
```

Listing 4: Asymetric route

```python
#!/usr/bin/env python
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.nodelib import LinuxBridge
from mininet.log import setLogLevel

def set_ipv6(host, interface, ip):
    return host.cmd("ip -6 addr add " + ip + " dev " + interface)

def set_ipv6_gateway(host, gateway):
    return host.cmd("ip -6 route add default via " + gateway)

def set_ipv6_device_route(host, subnet, interface):
    return host.cmd("ip -6 route add " + subnet + " dev " + interface)

def set_ipv6_gateway_route(host, subnet, gateway, interface):
    # useful for the forward table
    return host.cmd("ip -6 route add " + subnet +
            " via " + gateway + " dev " + interface)




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

    net = Mininet(switch=LinuxBridge, controller=None)

    h1 = net.addHost('h1', ip=None)

    # create rest of the topology here
    s1 = net.addSwitch('s1')
    r1 = net.addHost('r1', ip = None)
    s0 = net.addSwitch('s0')
    r2 = net.addHost('r2', ip = None)
    s2 = net.addSwitch('s2')
    h2 = net.addHost('h2', ip = None)
    r3 = net.addHost('r3', ip = None)
    s3 = net.addSwitch('s3')
    r4 = net.addHost('r4', ip = None)

    # addr info
```

```
43      h1_eth0_addr = "2001:638:709:a::1/64"
44      r1_eth0_addr = "2001:638:709:a::f/64"
45      r1_eth1_addr = "2001:638:709:f::1/64"
46      r2_eth0_addr = "2001:638:709:f::2/64"
47      r2_eth1_addr = "2001:638:709:b::f/64"
48      h2_eth0_addr = "2001:638:709:b::1/64"
49      r3_eth0_addr = "2001:638:709:a::e/64"
50      r3_eth1_addr = "2001:638:709:e::1/64"
51      r4_eth0_addr = "2001:638:709:e::2/64"
52      r4_eth1_addr = "2001:638:709:b::e/64"
53
54      # subnet info
55      subnet1_addr = "2001:638:709:a::/64"
56      subnet2_addr = "2001:638:709:f::/64"
57      subnet3_addr = "2001:638:709:b::/64"
58      subnet4_addr = " 2001:638:709:e::/64"
59
60      # link the router and switches
61      net.addLink(h1, s1)
62      net.addLink(s1, r1)
63      net.addLink(r1, s0)
64      net.addLink(s0, r2)
65      net.addLink(r2, s2)
66      net.addLink(s2, h2)
67      net.addLink(s1, r3)
68      net.addLink(r3, s3)
69      net.addLink(s3, r4)
70      net.addLink(r4, s2)
71
72      # configure IPv6 addresses and forwarding table entries here
73
74      # set ipv6 addr and gateway
75      print "setup ipv6"
76      print set_ipv6(h1, "h1-eth0", h1_eth0_addr)
77      print set_ipv6(r1, "r1-eth0", r1_eth0_addr)
78      print set_ipv6(r1, "r1-eth1", r1_eth1_addr)
79      print set_ipv6(r2, "r2-eth0", r2_eth0_addr)
80      print set_ipv6(r2, "r2-eth1", r2_eth1_addr)
81      print set_ipv6(h2, "h2-eth0", h2_eth0_addr)
82      print set_ipv6(r3, "r3-eth0", r3_eth0_addr)
83      print set_ipv6(r3, "r3-eth1", r3_eth1_addr)
84      print set_ipv6(r4, "r4-eth0", r4_eth0_addr)
85      print set_ipv6(r4, "r4-eth1", r4_eth1_addr)
86      print "done setup ipv6"
87
88      # setup the local route
89      print "setup forward rules"
90      print set_ipv6_device_route(h1, subnet1_addr, "h1-eth0")
91      print set_ipv6_device_route(r1, subnet1_addr, "r1-eth0")
92      print set_ipv6_device_route(r1, subnet2_addr, "r1-eth1")
93      print set_ipv6_device_route(r2, subnet2_addr, "r2-eth0")
94      print set_ipv6_device_route(r2, subnet3_addr, "r2-eth1")
95      print set_ipv6_device_route(h2, subnet3_addr, "h2-eth0")
96      print set_ipv6_device_route(r3, subnet1_addr, "r3-eth0")
97      print set_ipv6_device_route(r3, subnet4_addr, "r3-eth1")
98      print set_ipv6_device_route(r4, subnet4_addr, "r4-eth0")
99      print set_ipv6_device_route(r4, subnet3_addr, "r4-eth1")
100     print "done setup forward rules"
```

```
101
102     # setup the gateway
103     print "setup gateway"
104     print set_ipv6_gateway(h1, r1_eth0_addr[:-3])
105     print set_ipv6_gateway(h2, r4_eth1_addr[:-3])
106     print "done setup gateway"
107
108     # setup router forward table
109     print "setup router forward table"
110     print set_ipv6_gateway_route(r1, subnet3_addr, r2_eth0_addr[:-3], "r1-eth1")
111     print set_ipv6_gateway_route(r2, subnet1_addr, r1_eth1_addr[:-3], "r2-eth0")
112     print set_ipv6_gateway_route(r3, subnet3_addr, r4_eth0_addr[:-3], "r3-eth1")
113     print set_ipv6_gateway_route(r4, subnet1_addr, r3_eth1_addr[:-3], "r4-eth0")
114     print "done setup router forward table"
115
116     # router forwarding
117     print "enable forwarding for routers"
118     r1.cmd("sysctl -w net.ipv6.conf.all.forwarding=1")
119     r2.cmd("sysctl -w net.ipv6.conf.all.forwarding=1")
120     r3.cmd("sysctl -w net.ipv6.conf.all.forwarding=1")
121     r4.cmd("sysctl -w net.ipv6.conf.all.forwarding=1")
122     print "done enabling forwarding for routers"
123
124     print h1.cmd("ip -V")
125
126
127     net.start()
128     CLI(net)
129     net.stop()
```

c)

We continue to run the mininet of Listing 4, and we manually set the mtu of the interface on r1(a.k.a r1-eth1) and r2(a.k.a r2-eth0), and using ping6 to ping h2 from h1 with a package size of 1450.(default MTU is 1500 on h1)

Listing 5: mininet result of c)

```
1  mininet> r1 ifconfig r1-eth1 mtu 1400
2  mininet> r2 ifconfig r2-eth0 mtu 1400
3  mininet> h1 ifconfig
4  h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
5          inet6 fe80::f8fb:27ff:fe5a:ef33  prefixlen 64  scopeid 0x20<link>
6          inet6 2001:638:709:a::1  prefixlen 64  scopeid 0x0<global>
7          ether fa:fb:27:5a:ef:33  txqueuelen 1000  (Ethernet)
8          RX packets 71  bytes 7026 (7.0 KB)
9          RX errors 0  dropped 0  overruns 0  frame 0
10         TX packets 45  bytes 4282 (4.2 KB)
11         TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
12
13 lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
14         inet 127.0.0.1  netmask 255.0.0.0
15         inet6 ::1  prefixlen 128  scopeid 0x10<host>
16         loop  txqueuelen 1000  (Local Loopback)
17         RX packets 12  bytes 1716 (1.7 KB)
18         RX errors 0  dropped 0  overruns 0  frame 0
19         TX packets 12  bytes 1716 (1.7 KB)
20         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 fe80::c041:eeff:fe30:8074  prefixlen 64  scopeid 0x20<link>
        inet6 2001:638:709:b::1  prefixlen 64  scopeid 0x0<global>
        ether c2:41:ee:30:80:74  txqueuelen 1000  (Ethernet)
        RX packets 74  bytes 7284 (7.2 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 47  bytes 4454 (4.4 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 12  bytes 1716 (1.7 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 1716 (1.7 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

mininet> h1 tcpdump -i h1-eth0 -s 65535 -w ping1450mtu.dump &
tcpdump: h1 ping6 -M do -s 1450 2001:638:709:b::1
listening on h1-eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
PING 2001:638:709:b::1(2001:638:709:b::1) 1450 data bytes
From 2001:638:709:a::f icmp_seq=1 Packet too big: mtu=1400
1458 bytes from 2001:638:709:b::1: icmp_seq=2 ttl=62 time=0.208 ms
1458 bytes from 2001:638:709:b::1: icmp_seq=3 ttl=62 time=0.094 ms
1458 bytes from 2001:638:709:b::1: icmp_seq=4 ttl=62 time=0.083 ms
1458 bytes from 2001:638:709:b::1: icmp_seq=5 ttl=62 time=0.085 ms
1458 bytes from 2001:638:709:b::1: icmp_seq=6 ttl=62 time=0.081 ms
1458 bytes from 2001:638:709:b::1: icmp_seq=7 ttl=62 time=0.084 ms
1458 bytes from 2001:638:709:b::1: icmp_seq=8 ttl=62 time=0.083 ms
^C40 packets captured
40 packets received by filter
0 packets dropped by kernel
```

From the command line output, we can see that there is a message indicating that the package is too big.
We import the dump file 'ping1450mtu.dump' into the wireshark. We can clearly observe that all the ping6 packets are using ICMPv6 protocol. The first echo ICMPv6 packet has 1450 bytes of data and the first echo request didn't receive the echo response.
However, right after the first echo packet, h1(2001:638:709:a::1) receive a ICMPv6 packet from r1(2001:638:709:a::f) indicating that the packet is too big, with MTU of 1400.
Afterwards, the echo ICMPv6 packet is transfered in 2 IPv6 packets, and we can see that the ICMPv6 packet has been fragmented into 2 packets.

d)

We continue from previous problem, runing the tracepath:

Listing 6: Running tracepath

```
mininet> h1 tracepath -n 2001:638:709:b::1
 1?: [LOCALHOST]                      1.176ms pmtu 1500
 1:  2001:638:709:a::f                                     0.082ms
 1:  2001:638:709:a::f                                     0.016ms
 2:  2001:638:709:a::f                                     0.020ms pmtu 1400
 2:  2001:638:709:f::2                                     0.064ms
```

```
 3:  2001:638:709:b::1                                          0.150ms reached
     Resume: pmtu 1400 hops 3 back 3 1?: [LOCALHOST]                          0.552ms pmtu 1500
```

Ping can determine the MTU value because if the current MTU is bigger, the packet will be dropped in the route and an ICMPv6 will be forwarded back including the error message and the supported MTU value.
Tracepath, just like traceroute[1], using the TTL and the error ICMPv6 packet to determine the right MTU value. It set different TTL value so that the packet only perform limited hops. Using this technique and in combination of the error packet sent back when MTU has exceeded and dropped, we can determine the link that limit the MTU and the optimal MTU value.

e)

No, we can do a experiment in the mininet

Listing 7: Path MTU cache experiment

```
mininet> h1 tracepath -n 2001:638:709:b::1
 1?: [LOCALHOST]                         0.016ms pmtu 1500
 1:  2001:638:709:a::f                                          0.048ms
 1:  2001:638:709:a::f                                          0.015ms
 2:  2001:638:709:a::f                                          0.015ms pmtu 1400
 2:  2001:638:709:f::2                                          0.026ms
 3:  2001:638:709:b::1                                          0.037ms reached
     Resume: pmtu 1400 hops 3 back 3
mininet> h1 tracepath -n 2001:638:709:b::1
 1?: [LOCALHOST]                         0.016ms pmtu 1400
 1:  2001:638:709:a::f                                          0.042ms
 1:  2001:638:709:a::f                                          0.016ms
 2:  2001:638:709:f::2                                          0.025ms
 3:  2001:638:709:b::1                                          0.033ms reached
     Resume: pmtu 1400 hops 3 back 3
```

As we can see, in the second tracepath, the pmtu value starts from 1400. Linux has obviously cached the path MTU. Such a cache is stored in the routing table entries. The host will not have routing table entries for every destination, but it can cache per-host route for every active destination[2].

References

[1] How does traceroute work and example's of using traceroute command.

[2] Jeffrey Mogul and Steve Deering. Path mtu discovery. RFC 1191, RFC Editor, November 1990. http://www.rfc-editor.org/rfc/rfc1191.txt.