Computer Networks

Homework # 1

Submitted By: Neeha Hammad

Problem 1.1:

(a) By default, ping utility uses IPv4 and needs to be informed if IPv6 is to be used. However, we could also specify IPv4 by using:

```
ping -4 website
```

However, this gives a higher variation in minimum, maximum, and average times so I just used

```
ping website
```

I obtained my results in the following format:

```
--- amazon.com ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11015ms
rtt min/avg/max/mdev = 98.452/125.054/271.698/47.667 ms
```

Host	Min Time /ms	Average time /ms	Max Time /ms
amazon.com	98.452	125.054	271.698
www.amazon.com	8.467	11.305	19.046
www.jacobs-university.de	17.338	18.864	24.150
moodle.jacobs-university.de	1.499	2.020	3.288

Time of Measurement: Around 3:30 pm - 21.02.2020

Ping Version: ping utility, iputils-s20161105

- We observe that moodle is the fastest, followed www.amazon.com, then comes www.jacobs-university.de while amazon.com is the slowest.

- Moodle might be the fastest because we're using Eduroam and Moodle is probably hosted on university's own servers.

We see that when we run ping amazon.com, we get:

```
PING amazon.com (176.32.103.205) 56(84) bytes of data.
```

However, when we run ping www.amazon.com, we get:

```
PING e15316.e22.akamaiedge.net (104.85.252.108) 56(84) bytes of data.
```

Firstly, the "resolution" of DNS for amazon.com has a different list of servers hosting it, than for www.amazon.com. So, naturally, the timings are also different. If we use

```
dig www.amazon.com
```

we get:

```
www.amazon.com.                416    IN    CNAME    tp.47cf2c8c9-frontier.amazon.com.
tp.47cf2c8c9-frontier.amazon.com. 9 IN    CNAME    www.amazon.com.edgekey.net.
www.amazon.com.edgekey.net. 67    IN    CNAME    e15316.e22.akamaiedge.net.
e15316.e22.akamaiedge.net. 17      IN    A    104.85.252.108
```

However, if we dig amazon.com, we get:

```
amazon.com.         60    IN    A    176.32.103.205
amazon.com.         60    IN    A    205.251.242.103
amazon.com.         60    IN    A    176.32.98.166
```

- As we can see, the resolution for www points to Akamai, which has many points of presence (POPs) in metro areas, versus the main Amazon sites. This reduces latency, and ping response times for www.amazon.com. The explanation revolves around CDN (Cloud Delivery Network).

(b) I used the following command to see the routes:

```
mtr -z -c 15 amazon.com
```

and got the results in the following format:

```
                         Packets                Pings
Host                            Loss%  Snt   Last   Avg  Best  Wrst StDev
1. AS???       10.81.255.251          0.0%   15       3.4   3.0   1.6   6.8   1.3
2. AS???       192.168.242.3          0.0%   15       2.5   2.4   1.2   5.7   1.2
3. AS680       vkr-g2-5-1.x-win.uni-br  0.0% 15       2.1   2.7   1.9   5.5   1.0
4. AS680       cr-han2-be15.x-win.dfn.  0.0%          15       4.1   6.2   4.1  17.2   3.3
5. AS680       cr-fra2-be12.x-win.dfn.  0.0% 15   14.5  10.8   9.4  16.4   2.0
6. AS1299   ffm-b12-link.telia.net   0.0%      15       9.7  12.0   8.6  44.3   9.0
7. AS1299   ffm-bb2-link.telia.net   0.0%      15   96.0  96.6  95.9  98.2   0.6
8. AS1299   prs-bb4-link.telia.net   0.0%      15  107.4  98.7  96.0 107.4   3.6
9. AS1299   ash-bb3-link.telia.net  60.0%      15   95.9  97.2  95.9  99.8   1.5
10. AS1299   ash-b1-link.telia.net      0.0%   15  101.7 101.6 100.7 102.9   0.6
11. AS1299   vadata-ic-333118-ash-b1  0.0%         15  101.3 102.7 100.4 110.5   2.9
```

Host	Path	Number of AS???	Total Hops Including AS???	Total Hops Excluding AS???
amazon.com	AS??? (2 hops) AS680 (3 hops) -> AS1299 x 6 (hops)	2	11	9
www.amazon.com	AS??? (2 hops) -> AS680 (3 hops) -> AS??? (1 hop) -> AS16509 (1 hop)	3	7	4
www.jacobs-university.de	AS??? (2 hops) ->AS680 (3 hops) -> AS??? (1 hop) ->AS24940 (3 hops)	3	9	6
moodle.jacobs-university.de	AS??? (1 hop) ->AS680(1 hop)	1	2	1

mtr version: mtr 0.92

Time of Measurements: 7:15 pm - 21.02.2020

- AS680 is visited by all hosts. According to ultratools.com’s, ASN Lookup Tool, this Autonomous System belongs to Germany’s DFN Verein zur Foerderung eines Deutschen Forschungsnetzes e.V., DE.
- Moodle’s route is the shortest, probably because it is hosted by the university's own servers.
- amazon.com’s route is the longest.
- Routes for amazon.com and www.amazon.com are a bit different, which makes sense because they have a different list of servers hosting it.
- The first address accessed is always 10.81.255.251

Screenshots are attached. Note that there were times when the measurements varied.

Problem 1.2

(a)

AS Number	Owner	Registry
AS680	DFN - Verein zur Foerderung eines Deutschen Forschungsnetzes e.V.	RIPE
AS1299	TELIANET - Telia Company AB	RIPE
AS16509	AMAZON-02	ARIN
AS24940	HETZNER-AS - Hetzner Online GmbH	RIPE

(b) 2001:638:709::/48 is not globally announced. However, it is part of 2001:638::/32, which is publicly announced. It’s RPKI status says: DFN - Verein zur Foerderung eines Deutschen Forschungsnetzes e.V.

According to its description, the prefix is owned by the Campus Network of the International University Bremen. The netname is IUB-NET. It is also designated to RIPE NCC.

Problem 1.3:

(a) iperf is a tool for network performance measurement and tuning.

Example Command: client # iperf3 -i 10 -w 1M -t 60 -c [server hostname or ip address]

Here, **-i** is the interval to provide periodic bandwidth updates, **-s** is for listening as a server, and **-c** is to connect to a listening server. Upon running this:

`h2 iperf -s &` and this `h1 iperf -c h2 -i 10 -t 60`,

we get:

```
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 36554 connected with 10.0.0.2 port 5001
[ ID] Interval     Transfer         Bandwidth
[  3]  0.0-10.0 sec  12.1 MBytes  10.2 Mbits/sec
[  3] 10.0-20.0 sec  11.1 MBytes  9.33 Mbits/sec
[  3] 20.0-30.0 sec  11.5 MBytes  9.65 Mbits/sec
[  3] 30.0-40.0 sec  11.4 MBytes  9.54 Mbits/sec
[  3] 40.0-50.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 50.0-60.0 sec  11.4 MBytes  9.54 Mbits/sec
[  3]  0.0-60.1 sec  68.9 MBytes  9.62 Mbits/sec
```

The data is transferred at an approximate rate of 10 Mbits/sec, which makes sense because in the code, self.addLink(h1, h2, bw=10) is written and we have defined/limited the bandwidth as 10 ourselves. However, originally, at the interface level, the speed is 1000 Mb/sec. We have changed it ourselves. We could check this by the following command:

```
h1 ethtool h1-eth0
```

(b) `mininet> h1 ping h2`

```
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=0.061 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.055 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.059 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.061 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.054 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.061 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.067 ms
```

```
^C
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9003ms
rtt min/avg/max/mdev = 0.054/0.061/0.067/0.004 ms
```

However, when I ping while the iperf is measuring, my average time value is much greater.

```
mininet> h2 iperf -s &
mininet> h1 iperf -c h2 -i 10 -t 60 &
mininet> h1 ping h2
------------------------------------------------------------
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.1 port 59348 connected with 10.0.0.2 port 5001
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=22.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=21.0 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=19.3 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=36.0 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=26.2 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=20.1 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=24.8 ms
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  11.9 MBytes  9.96 Mbits/sec
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=15.4 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=26.7 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=31.9 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=28.1 ms
^C
--- 10.0.0.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10019ms
rtt min/avg/max/mdev = 15.448/24.760/36.008/5.658 ms
```

This is because of the queuing delay. Queuing refers to the time that a packet waits to be processed in the buffer of a switch. The delay is dependent on the arrival rate of the incoming packets, the transmission capacity of the outgoing link, and the nature of the network’s traffic.

Multiple applications may share the interface. Each link has a specific bandwidth. When we share the bandwidth on the link and use iperf, each application gets less bandwidth during the shared time. Frames are sent across one frame at a time. Thus, the packets have to wait in line to be sent (they are queued). A host can only serialize bits onto the wire at the bandwidth speed. Thus, the host will send an entire frame before sending the next frame.

Reference:
https://networkengineering.stackexchange.com/questions/65472/pinging-with-iperf-vs-without-iperf?noredirect=1#comment116789_65472

Problem 1.4:

(a) mininet> h3 ping h4

```
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=0.735 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.677 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.276 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.087 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.086 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.070 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.074 ms
^C
--- 10.0.0.4 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6004ms
rtt min/avg/max/mdev = 0.070/0.286/0.735/0.274 ms
```

```
mininet> h1 iperf -c h2 -i 10 -t 60 &
mininet> h3 ping h4

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=0.804 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=2.29 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.282 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.089 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.082 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=0.075 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=0.092 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=0.084 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=0.082 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=0.082 ms
^C
--- 10.0.0.4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9005ms
rtt min/avg/max/mdev = 0.075/0.397/2.298/0.669 ms
```

When running with iperf, the average is slightly higher but that's probably only because of 1 anomalous value. However, there doesn't seem to be a significant change so we could disregard the anomalous value and say that the iperf measurement doesn't really impact the round-trip times. I also tried h4 ping h3 but had the same observation.

(b) There doesn't seem to be much impact. This is what we get when we run them concurrently.

```
mininet> h4 iperf -s &
mininet> h1 iperf -c h2 -i 10 -t 60 &
mininet> h3 iperf -c h4 -i 10 -t 60
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  3] local 10.0.0.3 port 55868 connected with 10.0.0.4 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  12.0 MBytes  10.1 Mbits/sec
[  3] 10.0-20.0 sec  11.1 MBytes  9.33 Mbits/sec
[  3] 20.0-30.0 sec  11.5 MBytes  9.65 Mbits/sec
[  3] 30.0-40.0 sec  11.1 MBytes  9.33 Mbits/sec
[  3] 40.0-50.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 50.0-60.0 sec  11.4 MBytes  9.54 Mbits/sec
[  3]  0.0-60.2 sec  68.5 MBytes  9.55 Mbits/sec
```

When I run them individually, I observe that the transfer and bandwidth are approximately the same. The bandwidth lies in the range of 9.4-10 while the final transfer is 68.5 MBytes in both cases.

Problem 1.5:

(a) h1 to h4:

```
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  11.5 MBytes  9.65 Mbits/sec
[  3] 10.0-20.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 20.0-30.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 30.0-40.0 sec  11.1 MBytes  9.33 Mbits/sec
[  3] 40.0-50.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 50.0-60.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3]  0.0-60.2 sec  67.8 MBytes  9.44 Mbits/sec
```

h3 to h2:

```
[ ID] Interval    Transfer        Bandwidth
[  3]  0.0-10.0 sec  11.5 MBytes  9.65 Mbits/sec
[  3] 10.0-20.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 20.0-30.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 30.0-40.0 sec  11.1 MBytes  9.33 Mbits/sec
[  3] 40.0-50.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 50.0-60.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3]  0.0-60.2 sec  67.8 MBytes  9.44 Mbits/sec
```

Although I expected it to hit 10, the data rate in my case ranged from 9.44 to 9.65 Mbits/sec. However, I think it's safe to conclude that the average data rate is still 10 Mbits/sec. Note that in the network diagram in the code, we see that h2 - h3 and h1 - h4 are on the opposite ends.

However, when we run measurements for h1 to h3 and h2 to h4 (which are on same sides), we see that the data rate and transfer rate is far less. Thus, we can conclude that when 2 hosts lie on the same side, there's a delay when the clients have to send their packages. While on the other hand, when the link is between two opposite hosts, it's able to transfer data with its full capacity.

(b) h1 to h4:

```
[ ID] Interval    Transfer        Bandwidth
[  3]  0.0-10.0 sec  11.4 MBytes  9.54 Mbits/sec
[  3] 10.0-20.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 20.0-30.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 30.0-40.0 sec  11.1 MBytes  9.33 Mbits/sec
[  3] 40.0-50.0 sec  11.2 MBytes  9.44 Mbits/sec
[  3] 50.0-60.0 sec  11.1 MBytes  9.33 Mbits/sec
[  3]  0.0-60.1 sec  67.5 MBytes  9.42 Mbits/sec
```

This is just as it should be and has no connection with the second measurement.

h3 to h6:

```
[ ID] Interval    Transfer        Bandwidth
[  3]  0.0-10.0 sec  8.75 MBytes  7.34 Mbits/sec
[  3] 10.0-20.0 sec  7.62 MBytes  6.40 Mbits/sec
[  3] 20.0-30.0 sec  7.12 MBytes  5.98 Mbits/sec
[  3] 30.0-40.0 sec  9.12 MBytes  7.65 Mbits/sec
[  3] 40.0-50.0 sec  9.38 MBytes  7.86 Mbits/sec
[  3] 50.0-60.0 sec  8.75 MBytes  7.34 Mbits/sec
[  3]  0.0-60.1 sec  50.9 MBytes  7.10 Mbits/sec
```

As we can see, the data rate is less when we measure for h3 to h6. If we look at the diagram given, we see that a 5% loss is already mentioned between switches s2 and s6. We can also see this via the code:

self.addLink(s2, s3, bw=15, loss=5)

This makes our connection slow.