TCP and OSPF Assignment
ELC 3080

Name	ID
كلارا عيسى اسحاق عبدالمسيح	9203081
مينا صبحي كامل السيد	9203591

Submitted To: Prof. Dr. Khaled Fouad
3rd Year – Spring 2023

Contents

1. TCP Lab

Each of the following sections describes an experiment based on the network topology provided by the **Project_OSPF_TCP.imn** file. In your report, divide the answers to correspond to one of these sections in exactly the same sequence and same name, for example use Question 2.1, Question 3.2 and so on. The Introductory Lab explained and introduced most of the needed tools like iperf3, vtysh, wireshark, etc. If you have not gone through it, do not start working on the assignment below. It is better to start the network emulation once and run Wireshark captures. You can reset Wireshark captures from experiment to another.

1.1. Effect of TCP Window Size

- Set wireshark filter to display TCP packets only.
- Start an iperf3 server on node n11.
- Start an iperf3 client on node n7 connecting to server on node n11 for a duration of 40 seconds and reporting interval of 10. **Note that in iperf, the client is the node sending the traffic and the server simply receives and sends an ACK**.

1.1.1. Vary iperf3 window size from 1Kbytes to 6 Kbytes in increments of 1 Kbyte, then set it to 12, 16, 24, 32 Kbytes. Plot the average throughput and the average number of retransmissions as function of window size. Comment on the results and explain the zigzag behaviour noticed for larger window sizes. Note that number of retransmissions is the 5th column in iperf3 default output & throughput is the 4th. You can plot data using matlab/python/any online graphing tool (Ex. https://chart-studio.plotly.com/create/#/). The window size is the last option in the iperf command (return to INTRO Lab).

<table>
<tr><th>Window size
(Kbytes)</th><th>Average number of transmissions</th><th>Average throughput (Mbytes/sec)</th></tr>
<tr><td>1k
<pre>
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 1k
Connecting to host 10.0.13.20, port 5201
[4] local 10.0.12.20 port 40779 connected to 10.0.13.20 port 5201
[ID] Interval   Transfer   Bandwidth   Retr Cwnd
[4]   0.00-10.00 sec   977 KBytes   800 Kbits/sec   0   5.45 KBytes
[4] 10.00-20.00 sec 1.34 MBytes 1.12 Mbits/sec   0   5.45 KBytes
[4] 20.00-30.00 sec 1.32 MBytes 1.11 Mbits/sec   0   5.45 KBytes
[4] 30.00-40.00 sec 1.32 MBytes 1.11 Mbits/sec   0   5.45 KBytes
- -
[ID] Interval   Transfer   Bandwidth   Retr
[4]   0.00-40.00 sec 4.94 MBytes 1.03 Mbits/sec   0   sender
[4]   0.00-40.00 sec 4.93 MBytes 1.03 Mbits/sec   receiver

iperf Done.
</pre></td><td>0</td><td>0.123375</td></tr>
<tr><td>2k
<pre>
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 2k
Connecting to host 10.0.13.20, port 5201
[4] local 10.0.12.20 port 40781 connected to 10.0.13.20 port 5201
[ID] Interval   Transfer   Bandwidth   Retr Cwnd
[4]   0.00-10.00 sec 2.44 MBytes 2.05 Mbits/sec   0   7.07 KBytes
[4] 10.00-20.00 sec 3.43 MBytes 2.87 Mbits/sec   0   7.07 KBytes
[4] 20.00-30.00 sec 3.43 MBytes 2.87 Mbits/sec   0   7.07 KBytes
[4] 30.00-40.00 sec 3.44 MBytes 2.88 Mbits/sec   0   7.07 KBytes
- -
[ID] Interval   Transfer   Bandwidth   Retr
[4]   0.00-40.00 sec 12.7 MBytes 2.67 Mbits/sec   0   sender
[4]   0.00-40.00 sec 12.7 MBytes 2.67 Mbits/sec   receiver

iperf Done.
</pre></td><td>0</td><td>0.3175</td></tr>
<tr><td>3k
<pre>
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 3k
Connecting to host 10.0.13.20, port 5201
[4] local 10.0.12.20 port 40783 connected to 10.0.13.20 port 5201
[ID] Interval   Transfer   Bandwidth   Retr Cwnd
[4]   0.00-10.00 sec 5.25 MBytes 4.41 Mbits/sec   0   14.1 KBytes
[4] 10.00-20.00 sec 6.85 MBytes 5.74 Mbits/sec   0   14.1 KBytes
[4] 20.00-30.00 sec 6.52 MBytes 5.47 Mbits/sec   0   14.1 KBytes
[4] 30.00-40.00 sec 6.85 MBytes 5.75 Mbits/sec   0   14.1 KBytes
- -
[ID] Interval   Transfer   Bandwidth   Retr
[4]   0.00-40.00 sec 25.5 MBytes 5.34 Mbits/sec   0   sender
[4]   0.00-40.00 sec 25.5 MBytes 5.34 Mbits/sec   receiver

iperf Done.
</pre></td><td>0</td><td>0.6375</td></tr>
<tr><td>4k
<pre>
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 4k
Connecting to host 10.0.13.20, port 5201
[4] local 10.0.12.20 port 40785 connected to 10.0.13.20 port 5201
[ID] Interval   Transfer   Bandwidth   Retr Cwnd
[4]   0.00-10.00 sec 5.10 MBytes 4.28 Mbits/sec   0   14.1 KBytes
[4] 10.00-20.00 sec 6.87 MBytes 5.76 Mbits/sec   0   14.1 KBytes
[4] 20.00-30.00 sec 6.87 MBytes 5.76 Mbits/sec   0   14.1 KBytes
[4] 30.00-40.00 sec 6.77 MBytes 5.68 Mbits/sec   0   14.1 KBytes
- -
[ID] Interval   Transfer   Bandwidth   Retr
[4]   0.00-40.00 sec 25.6 MBytes 5.37 Mbits/sec   0   sender
[4]   0.00-40.00 sec 25.6 MBytes 5.37 Mbits/sec   receiver

iperf Done.
</pre></td><td>0</td><td>0.64</td></tr>
</table>

<table>
<tr>
<td>

5k

<pre>
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 5k
Connecting to host 10.0.13.20, port 5201
[4] local 10.0.12.20 port 40787 connected to 10.0.13.20 port 5201
[ID] Interval   Transfer   Bandwidth   Retr Cwnd
[4]   0.00-10.00 sec 5.99 MBytes 5.03 Mbits/sec   0   14.1 KBytes
[4] 10.00-20.00 sec 6.92 MBytes 5.80 Mbits/sec   0   14.1 KBytes
[4] 20.00-30.00 sec 6.81 MBytes 5.72 Mbits/sec   0   14.1 KBytes
[4] 30.00-40.00 sec 6.91 MBytes 5.80 Mbits/sec   0   14.1 KBytes
- -
[ID] Interval   Transfer   Bandwidth   Retr
[4]   0.00-40.00 sec 26.6 MBytes 5.59 Mbits/sec   0   sender
[4]   0.00-40.00 sec 26.6 MBytes 5.58 Mbits/sec   receiver

iperf Done.
</pre>
</td>
<td>0</td>
<td>0.665</td>
</tr>
<tr>
<td>

6k

<pre>
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 6k
Connecting to host 10.0.13.20, port 5201
[4] local 10.0.12.20 port 40789 connected to 10.0.13.20 port 5201
[ID] Interval   Transfer   Bandwidth   Retr Cwnd
[4]   0.00-10.00 sec   154 KBytes   126 Kbits/sec   86   4.24 KBytes
[4] 10.00-20.00 sec   143 KBytes   117 Kbits/sec   88   4.24 KBytes
[4] 20.00-30.00 sec   144 KBytes   118 Kbits/sec   86   4.24 KBytes
[4] 30.00-40.00 sec   139 KBytes   114 Kbits/sec   85   4.24 KBytes
- -
[ID] Interval   Transfer   Bandwidth   Retr
[4]   0.00-40.00 sec   580 KBytes   119 Kbits/sec 345   sender
[4]   0.00-40.00 sec   570 KBytes   117 Kbits/sec   receiver

iperf Done.
</pre>
</td>
<td>345</td>
<td>0.0148038</td>
</tr>
<tr>
<td>

12k

<pre>
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 12k
Connecting to host 10.0.13.20, port 5201
[4] local 10.0.12.20 port 40791 connected to 10.0.13.20 port 5201
[ID] Interval   Transfer   Bandwidth   Retr Cwnd
[4]   0.00-10.00 sec 1.56 MBytes 1.31 Mbits/sec 268   2.83 KBytes
[4] 10.00-20.00 sec 2.09 MBytes 1.75 Mbits/sec 378   2.83 KBytes
[4] 20.00-30.00 sec 2.17 MBytes 1.82 Mbits/sec 392   2.83 KBytes
[4] 30.00-40.00 sec 2.03 MBytes 1.71 Mbits/sec 367   4.24 KBytes
- -
[ID] Interval   Transfer   Bandwidth   Retr
[4]   0.00-40.00 sec 7.85 MBytes 1.65 Mbits/sec 1405   sender
[4]   0.00-40.00 sec 7.83 MBytes 1.64 Mbits/sec   receiver

iperf Done.
</pre>
</td>
<td>1405</td>
<td>0.196</td>
</tr>
<tr>
<td>

16k

<pre>
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 16k
Connecting to host 10.0.13.20, port 5201
[4] local 10.0.12.20 port 40793 connected to 10.0.13.20 port 5201
[ID] Interval   Transfer   Bandwidth   Retr Cwnd
[4]   0.00-10.00 sec 1.47 MBytes 1.23 Mbits/sec 224   4.24 KBytes
[4] 10.00-20.00 sec 1.48 MBytes 1.24 Mbits/sec 214   4.24 KBytes
[4] 20.00-30.00 sec   686 KBytes   562 Kbits/sec 145   4.24 KBytes
[4] 30.00-40.00 sec   445 KBytes   365 Kbits/sec 125   7.07 KBytes
- -
[ID] Interval   Transfer   Bandwidth   Retr
[4]   0.00-40.00 sec 4.05 MBytes   849 Kbits/sec 708   sender
[4]   0.00-40.00 sec 4.02 MBytes   844 Kbits/sec   receiver

iperf Done.
</pre>
</td>
<td>708</td>
<td>0.100875</td>
</tr>
</table>

24k root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 24k Connecting to host 10.0.13.20, port 5201 [4] local 10.0.12.20 port 40795 connected to 10.0.13.20 port 5201 [ID] Interval Transfer Bandwidth Retr Cwnd [4] 0.00-10.00 sec 310 KBytes 254 Kbits/sec 113 7.07 KBytes [4] 10.00-20.00 sec 277 KBytes 227 Kbits/sec 111 5.66 KBytes [4] 20.00-30.00 sec 257 KBytes 211 Kbits/sec 106 8.48 KBytes [4] 30.00-40.00 sec 277 KBytes 227 Kbits/sec 111 7.07 KBytes - [ID] Interval Transfer Bandwidth Retr [4] 0.00-40.00 sec 1.10 MBytes 230 Kbits/sec 441 sender [4] 0.00-40.00 sec 1.06 MBytes 221 Kbits/sec receiver iperf Done.	441	0.027
32k root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 32k Connecting to host 10.0.13.20, port 5201 [4] local 10.0.12.20 port 40797 connected to 10.0.13.20 port 5201 [ID] Interval Transfer Bandwidth Retr Cwnd [4] 0.00-10.00 sec 246 KBytes 202 Kbits/sec 105 7.07 KBytes [4] 10.00-20.00 sec 189 KBytes 155 Kbits/sec 96 12.7 KBytes [4] 20.00-30.00 sec 161 KBytes 132 Kbits/sec 93 22.6 KBytes [4] 30.00-40.00 sec 214 KBytes 175 Kbits/sec 112 7.07 KBytes - [ID] Interval Transfer Bandwidth Retr [4] 0.00-40.00 sec 810 KBytes 166 Kbits/sec 406 sender [4] 0.00-40.00 sec 759 KBytes 156 Kbits/sec receiver iperf Done.	406	0.01915

From the above data the following code has written to plot the average number of retransmission and throughput vs the window size

```
%setting the window size values
window_size = [1 2 3 4 5 6 12 16 24 32];
%setting the average retransmissions values
avg_retr = [0 0 0 0 0 345 1405 708 441 406];
%setting the average throughput values
avg_throughput = [1.03 2.67 5.34 5.37 5.585 0.118 1.645 ...
    0.8465 0.2255 0.161];

%plotting average retransmissions vs window size
figure
plot(window_size, avg_retr)
xlabel("Window size Kbytes");
ylabel("Average number of retransmissions");
title("Average number of transmissions vs Window size");
%plotting average throughput vs window size
figure
plot(window_size, avg_throughput)
xlabel("Window size in Kbytes");
ylabel("Average throughput in Mbits/ sec");
title("Average throughput vs Window size");
```

This code generates the following two figures:

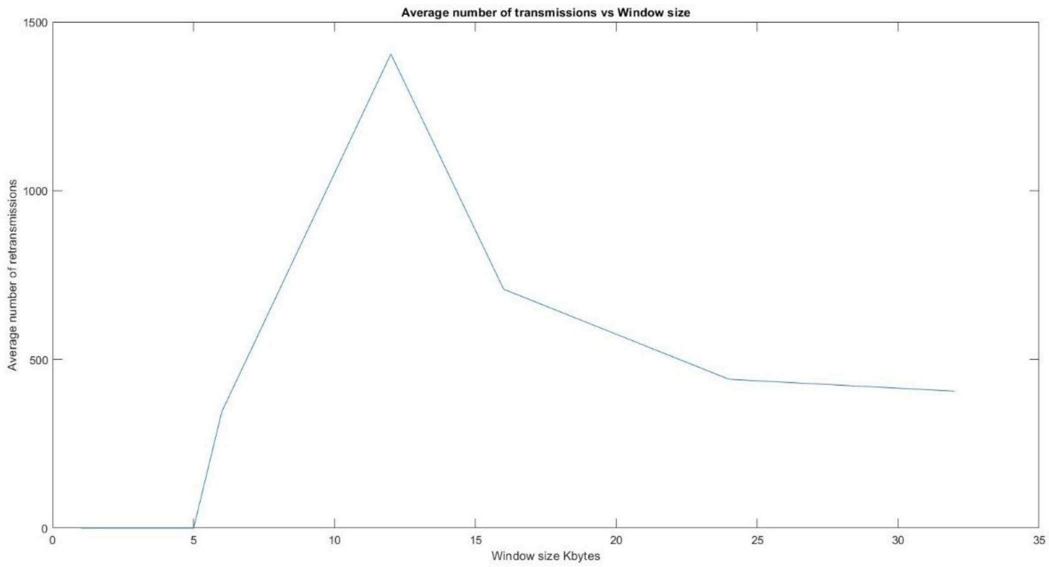


Figure 1-1 average number of transmission vs window size.

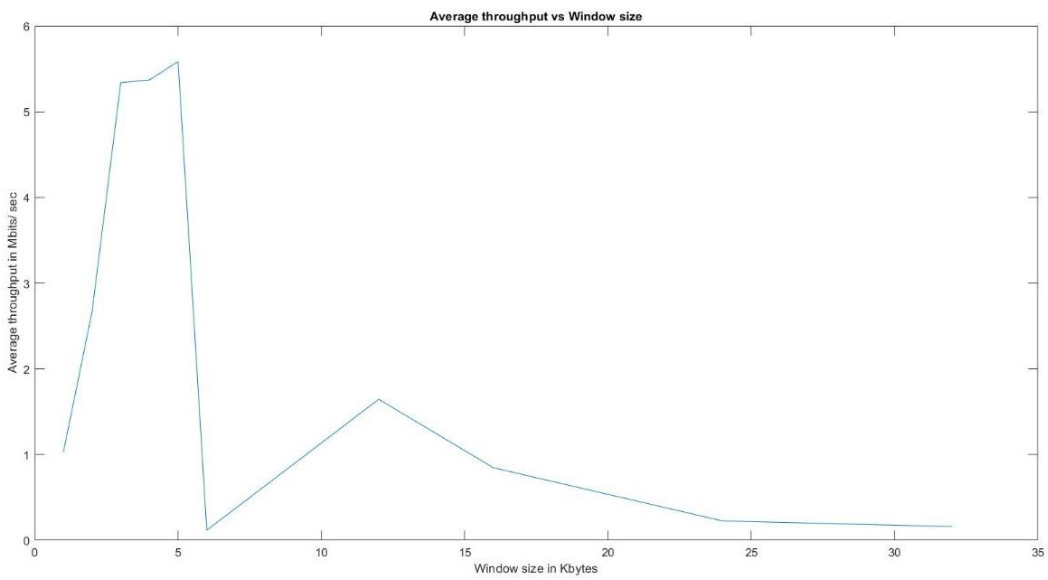


Figure 1-2 average throughput vs window size.

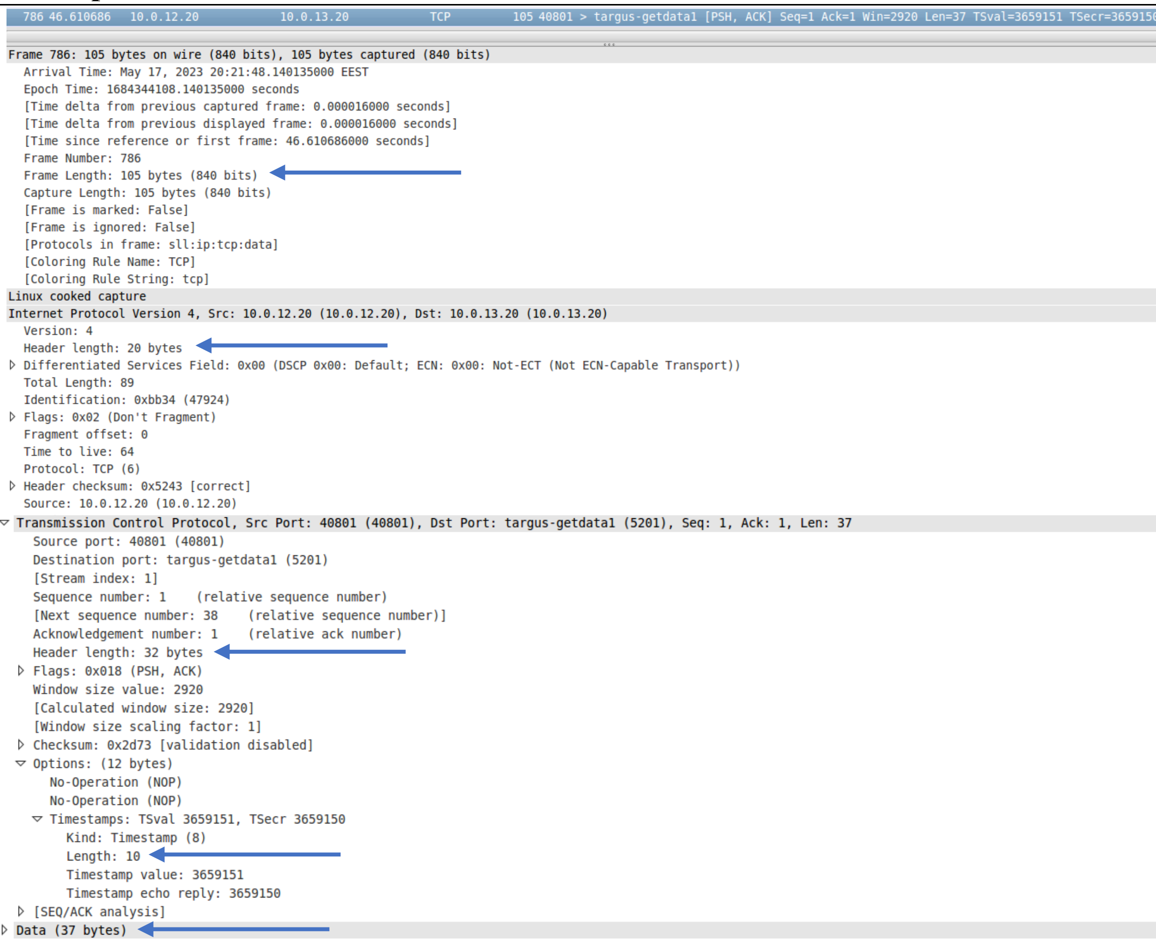
Comment on your findings:

Firstly, when the window size is from 1 Kbytes to 6 Kbytes, the retransmission packets are equal to zero, meaning that the receiver buffer is not filled yet, therefore by increasing the window size the throughput increases until reaching 6 Kbytes window.

At 6 Kbytes window, the retransmission packets increase as the buffer is filled and the packets are lost as the TCP uses Go Back N protocol, therefore it retransmits the whole window which increases the congestion and decreases the throughput, then the receiver must upgrade the buffer size to handle the new window size.

By increasing the window size to 12 Kbytes, the retransmission packets reduce, and the throughput increases until the buffer is filled so the retransmission increases, and the throughput decreases again which form the zigzag shape.
As the window size increases, the chance to retransmit a packet increases as the window size has more packets, then the chance to retransmit the window again will increase which make the throughput decreases at the end of the graph comparing to the initial values.

1.1.2. Click on any of the TCP a data segment whose source is node n7, dissect the segment by following different protocol layer headers from TCP->IP->Ethernet identifying how many header bytes are added by each layer, identify the TCP options used.



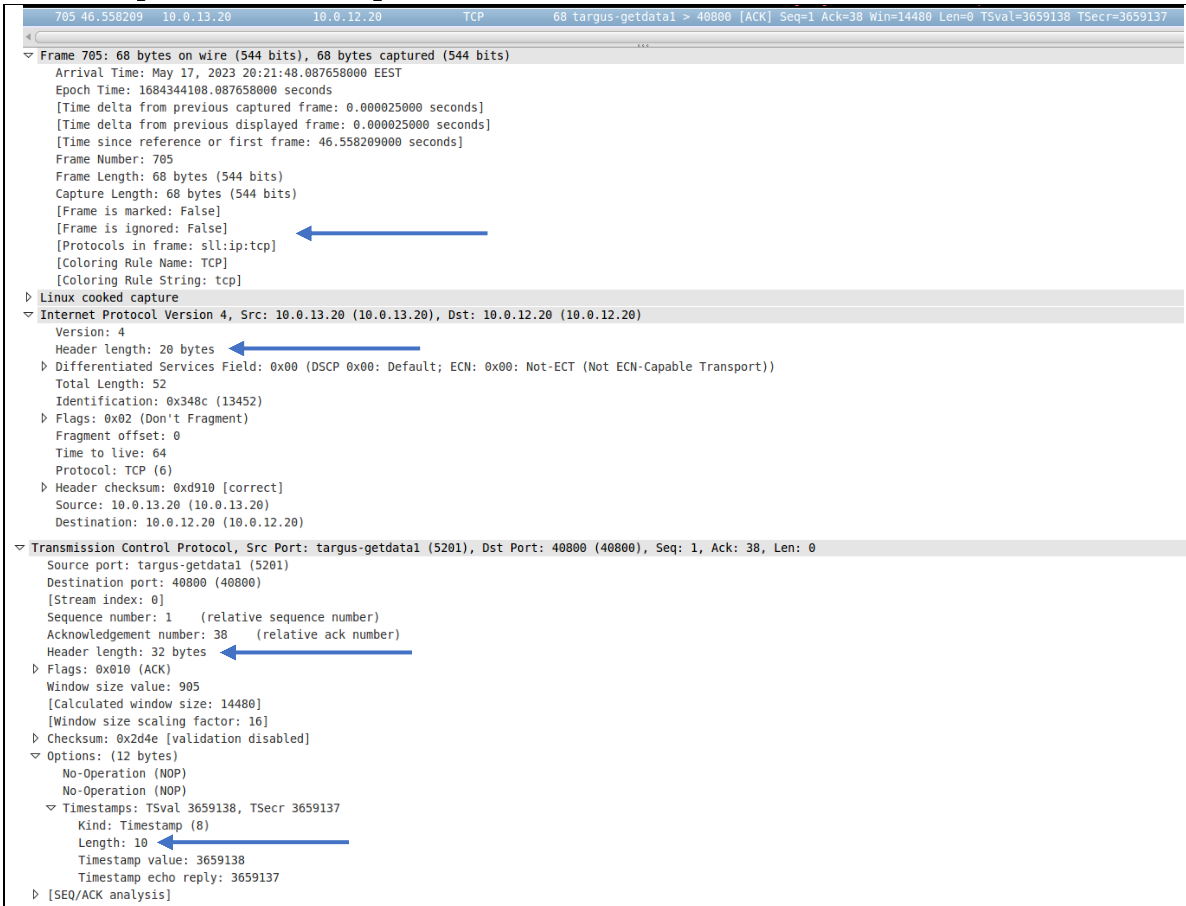
```
786 46.610686   10.0.12.20          10.0.13.20          TCP       105 40801 > targus-getdata1 [PSH, ACK] Seq=1 Ack=1 Win=2920 Len=37 TSval=3659151 TSecr=3659150
Frame 786: 105 bytes on wire (840 bits), 105 bytes captured (840 bits)
  Arrival Time: May 17, 2023 20:21:48.140135000 EEST
  Epoch Time: 1684344108.140135000 seconds
  [Time delta from previous captured frame: 0.000016000 seconds]
  [Time delta from previous displayed frame: 0.000016000 seconds]
  [Time since reference or first frame: 46.610686000 seconds]
  Frame Number: 786
  Frame Length: 105 bytes (840 bits)
  Capture Length: 105 bytes (840 bits)
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: sll:ip:tcp:data]
  [Coloring Rule Name: TCP]
  [Coloring Rule String: tcp]
Linux cooked capture
Internet Protocol Version 4, Src: 10.0.12.20 (10.0.12.20), Dst: 10.0.13.20 (10.0.13.20)
  Version: 4
  Header length: 20 bytes
▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
  Total Length: 89
  Identification: 0xbb34 (47924)
▷ Flags: 0x02 (Don't Fragment)
  Fragment offset: 0
  Time to live: 64
  Protocol: TCP (6)
▷ Header checksum: 0x5243 [correct]
  Source: 10.0.12.20 (10.0.12.20)
▽ Transmission Control Protocol, Src Port: 40801 (40801), Dst Port: targus-getdata1 (5201), Seq: 1, Ack: 1, Len: 37
    Source port: 40801 (40801)
    Destination port: targus-getdata1 (5201)
    [Stream index: 1]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 38    (relative sequence number)]
    Acknowledgement number: 1    (relative ack number)
    Header length: 32 bytes
  ▷ Flags: 0x018 (PSH, ACK)
    Window size value: 2920
    [Calculated window size: 2920]
    [Window size scaling factor: 1]
  ▷ Checksum: 0x2d73 [validation disabled]
  ▽ Options: (12 bytes)
      No-Operation (NOP)
      No-Operation (NOP)
    ▽ Timestamps: TSval 3659151, TSecr 3659150
        Kind: Timestamp (8)
        Length: 10
        Timestamp value: 3659151
        Timestamp echo reply: 3659150
  ▷ [SEQ/ACK analysis]
▷ Data (37 bytes)
```

Figure 1-3 Wireshark response of TCP packet sent from n7 to n11.

Comment on Your findings:

- TCP header length = 32 bytes (including options: timestamp (10 bytes) + 2 bytes No Operation for padding the options and making the total header multiple of 32 bits)
- IP header length = 20 bytes
- Data length = 37 bytes
- Ethernet length = Frame length – TCP header length – IP header length - Data = 105 – 32 – 20 – 37 = 16 bytes

1.1.3. Repeat for an ACK packet sent from node n11.



```
705 46.558209   10.0.13.20   10.0.12.20   TCP   68 targus-getdata1 > 40800 [ACK] Seq=1 Ack=38 Win=14480 Len=0 TSval=3659138 TSecr=3659137
▽ Frame 705: 68 bytes on wire (544 bits), 68 bytes captured (544 bits)
    Arrival Time: May 17, 2023 20:21:48.087658000 EEST
    Epoch Time: 1684344108.087658000 seconds
    [Time delta from previous captured frame: 0.000025000 seconds]
    [Time delta from previous displayed frame: 0.000025000 seconds]
    [Time since reference or first frame: 46.558209000 seconds]
    Frame Number: 705
    Frame Length: 68 bytes (544 bits)
    Capture Length: 68 bytes (544 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: sll:ip:tcp]
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]
▷ Linux cooked capture
▽ Internet Protocol Version 4, Src: 10.0.13.20 (10.0.13.20), Dst: 10.0.12.20 (10.0.12.20)
    Version: 4
    Header length: 20 bytes
  ▷ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
    Total Length: 52
    Identification: 0x348c (13452)
  ▷ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
  ▷ Header checksum: 0xd910 [correct]
    Source: 10.0.13.20 (10.0.13.20)
    Destination: 10.0.12.20 (10.0.12.20)
▽ Transmission Control Protocol, Src Port: targus-getdata1 (5201), Dst Port: 40800 (40800), Seq: 1, Ack: 38, Len: 0
    Source port: targus-getdata1 (5201)
    Destination port: 40800 (40800)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    Acknowledgement number: 38    (relative ack number)
    Header length: 32 bytes
  ▷ Flags: 0x010 (ACK)
    Window size value: 905
    [Calculated window size: 14480]
    [Window size scaling factor: 16]
  ▷ Checksum: 0x2d4e [validation disabled]
  ▽ Options: (12 bytes)
      No-Operation (NOP)
      No-Operation (NOP)
    ▽ Timestamps: TSval 3659138, TSecr 3659137
        Kind: Timestamp (8)
        Length: 10
        Timestamp value: 3659138
        Timestamp echo reply: 3659137
  ▷ [SEQ/ACK analysis]
```

Figure 1-4 Wireshark response of TCP packet sent from n11 to n7.

Comment on your findings here:

- TCP header length = 32 bytes (including options: timestamp (10 bytes) + 2 bytes No Operation for padding the options and making the total header multiple of 32 bits)
- IP header length = 20 bytes
- Data length = 0 bytes (just an ACK)
- Ethernet length = Frame length – TCP header length – IP header length = 68 – 32 – 20 = 16 bytes

1.2. TCP short VS long paths

- Run an iperf3 server on node n8.
- Start an iperf3 client on node n7 connecting to server on node n8 for a duration of 40 seconds and reporting interval of 10 with window size 4K.

1.2.1. Compare the result of throughput with the case when connection was made to node n11. Why throughput drops when connecting to n8 although capacities on the two paths are the same.

```
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.11.20 -t 40 -i 10 -w 4k
Connecting to host 10.0.11.20, port 5201
[  4] local 10.0.12.20 port 45063 connected to 10.0.11.20 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-10.00  sec  3.02 MBytes  2.53 Mbits/sec    0   14.1 KBytes
[  4]  10.00-20.01  sec  1.58 MBytes  1.33 Mbits/sec    3   8.48 KBytes
[  4]  20.01-30.00  sec  2.51 MBytes  2.11 Mbits/sec    0   8.48 KBytes
[  4]  30.00-40.00  sec  3.27 MBytes  2.74 Mbits/sec    1   8.48 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-40.00  sec  10.4 MBytes  2.18 Mbits/sec    4             sender
[  4]   0.00-40.00  sec  10.4 MBytes  2.18 Mbits/sec                  receiver

iperf Done.
```

```
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 4k
Connecting to host 10.0.13.20, port 5201
[  4] local 10.0.12.20 port 40785 connected to 10.0.13.20 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-10.00  sec  5.10 MBytes  4.28 Mbits/sec    0   14.1 KBytes
[  4]  10.00-20.00  sec  6.87 MBytes  5.76 Mbits/sec    0   14.1 KBytes
[  4]  20.00-30.00  sec  6.87 MBytes  5.76 Mbits/sec    0   14.1 KBytes
[  4]  30.00-40.00  sec  6.77 MBytes  5.68 Mbits/sec    0   14.1 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-40.00  sec  25.6 MBytes  5.37 Mbits/sec    0             sender
[  4]   0.00-40.00  sec  25.6 MBytes  5.37 Mbits/sec                  receiver

iperf Done.
```

Figure 1-5 left picture is sending from n7 to n8 & right picture is sending from n7 to n11

Comment in your findings here:

The throughput of sending from n7 to n8 (2.18 Mbits/sec) is much smaller than the throughput of sending from n7 to n11 (5.37 Mbits/sec). The path from n7 to n8 is larger than the path from n7 to n11, hence there is more propagation delays with larger RTT and more cost on the path which leads to less throughput.

1.3. Higher link Capacity with Drops VS Reliable Lower Capacity

This questions for this part are based on the path between n7 and n11.

For each case of the following, run iperf3 client from n7 to n11 with window size 4K.

A Select link between n4 and n5, configure it to have capacity of 10 Mbps with zero loss in both directions.

B Select link between n4 and n5, configure it to have capacity of 3 Mbps with zero loss in both directions.

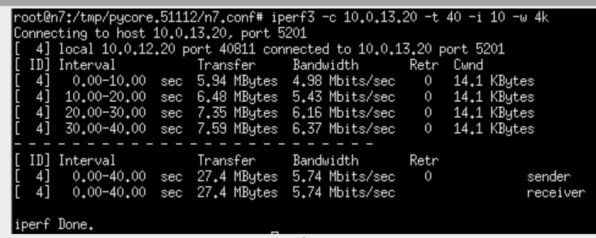
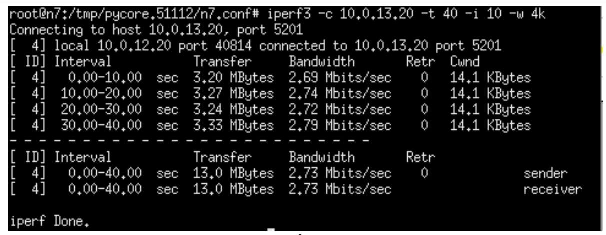
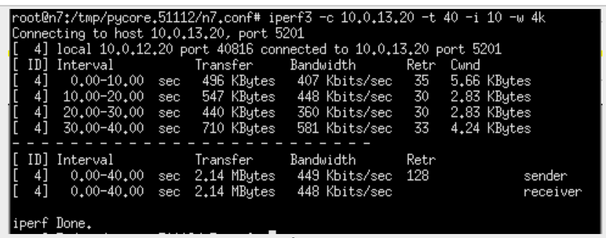
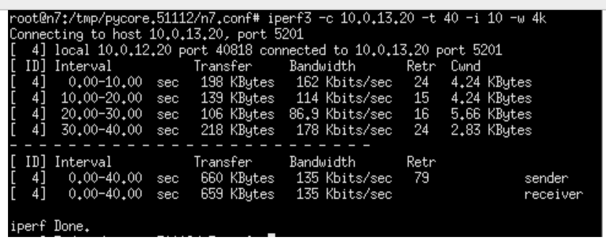
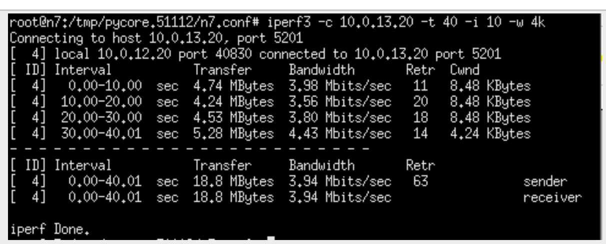
C Select link between n4 and n5, configure it to have capacity of 10 Mbps with 5% loss in both directions.

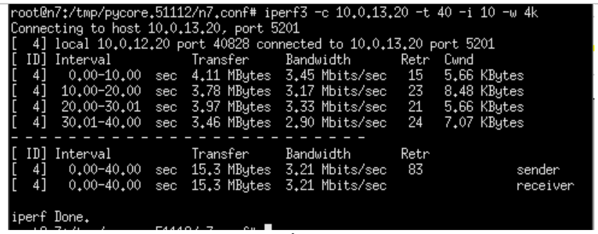
D Select link between n4 and n5, configure it to have capacity of 100 Mbps with 10% loss in both directions.

E Select link between n4 and n5, configure it to have capacity of 10 Mbps with 1% loss in direction from n4 to n5 and 0% loss in the other direction.

F Select link between n4 and n5, configure it to have capacity of 10 Mbps with 0% loss in direction from n4 to n5 and 1% loss in the other direction.

Please fill the following table with the AVG throughput of each case:

Case	Throughput
A	 root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 4k Connecting to host 10.0.13.20, port 5201 [4] local 10.0.12.20 port 40811 connected to 10.0.13.20 port 5201 [ID] Interval Transfer Bandwidth Retr Cwnd [4] 0.00-10.00 sec 5.94 MBytes 4.98 Mbits/sec 0 14.1 KBytes [4] 10.00-20.00 sec 6.48 MBytes 5.43 Mbits/sec 0 14.1 KBytes [4] 20.00-30.00 sec 7.35 MBytes 6.16 Mbits/sec 0 14.1 KBytes [4] 30.00-40.00 sec 7.59 MBytes 6.37 Mbits/sec 0 14.1 KBytes - [ID] Interval Transfer Bandwidth Retr [4] 0.00-40.00 sec 27.4 MBytes 5.74 Mbits/sec 0 sender [4] 0.00-40.00 sec 27.4 MBytes 5.74 Mbits/sec receiver iperf Done. 5.74 Mbits/sec
B	 root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 4k Connecting to host 10.0.13.20, port 5201 [4] local 10.0.12.20 port 40814 connected to 10.0.13.20 port 5201 [ID] Interval Transfer Bandwidth Retr Cwnd [4] 0.00-10.00 sec 3.20 MBytes 2.69 Mbits/sec 0 14.1 KBytes [4] 10.00-20.00 sec 3.27 MBytes 2.74 Mbits/sec 0 14.1 KBytes [4] 20.00-30.00 sec 3.24 MBytes 2.72 Mbits/sec 0 14.1 KBytes [4] 30.00-40.00 sec 3.33 MBytes 2.79 Mbits/sec 0 14.1 KBytes - [ID] Interval Transfer Bandwidth Retr [4] 0.00-40.00 sec 13.0 MBytes 2.73 Mbits/sec 0 sender [4] 0.00-40.00 sec 13.0 MBytes 2.73 Mbits/sec receiver iperf Done. 2.73 Mbits/sec
C	 root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 4k Connecting to host 10.0.13.20, port 5201 [4] local 10.0.12.20 port 40816 connected to 10.0.13.20 port 5201 [ID] Interval Transfer Bandwidth Retr Cwnd [4] 0.00-10.00 sec 496 KBytes 407 Kbits/sec 35 5.66 KBytes [4] 10.00-20.00 sec 547 KBytes 448 Kbits/sec 30 2.83 KBytes [4] 20.00-30.00 sec 440 KBytes 360 Kbits/sec 30 2.83 KBytes [4] 30.00-40.00 sec 710 KBytes 581 Kbits/sec 33 4.24 KBytes - [ID] Interval Transfer Bandwidth Retr [4] 0.00-40.00 sec 2.14 MBytes 449 Kbits/sec 128 sender [4] 0.00-40.00 sec 2.14 MBytes 448 Kbits/sec receiver iperf Done. 449 Kbits/sec
D	 root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 4k Connecting to host 10.0.13.20, port 5201 [4] local 10.0.12.20 port 40818 connected to 10.0.13.20 port 5201 [ID] Interval Transfer Bandwidth Retr Cwnd [4] 0.00-10.00 sec 198 KBytes 162 Kbits/sec 24 4.24 KBytes [4] 10.00-20.00 sec 139 KBytes 114 Kbits/sec 15 4.24 KBytes [4] 20.00-30.00 sec 106 KBytes 86.9 Kbits/sec 16 5.66 KBytes [4] 30.00-40.00 sec 218 KBytes 178 Kbits/sec 24 2.83 KBytes - [ID] Interval Transfer Bandwidth Retr [4] 0.00-40.00 sec 660 KBytes 135 Kbits/sec 79 sender [4] 0.00-40.00 sec 659 KBytes 135 Kbits/sec receiver iperf Done. 135 Kbits/sec
E	 root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 4k Connecting to host 10.0.13.20, port 5201 [4] local 10.0.12.20 port 40830 connected to 10.0.13.20 port 5201 [ID] Interval Transfer Bandwidth Retr Cwnd [4] 0.00-10.00 sec 4.74 MBytes 3.98 Mbits/sec 11 8.48 KBytes [4] 10.00-20.00 sec 4.24 MBytes 3.56 Mbits/sec 20 8.48 KBytes [4] 20.00-30.00 sec 4.53 MBytes 3.80 Mbits/sec 18 8.48 KBytes [4] 30.00-40.01 sec 5.28 MBytes 4.43 Mbits/sec 14 4.24 KBytes - [ID] Interval Transfer Bandwidth Retr [4] 0.00-40.01 sec 18.8 MBytes 3.94 Mbits/sec 63 sender [4] 0.00-40.01 sec 18.8 MBytes 3.94 Mbits/sec receiver iperf Done. 3.94 Mbits/sec

F	 3.21 Mbits/sec

```
root@n7:/tmp/pycore.51112/n7.conf# iperf3 -c 10.0.13.20 -t 40 -i 10 -w 4k
Connecting to host 10.0.13.20, port 5201
[  4] local 10.0.12.20 port 40828 connected to 10.0.13.20 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-10.00  sec  4.11 MBytes  3.45 Mbits/sec   15   5.66 KBytes
[  4]  10.00-20.00  sec  3.78 MBytes  3.17 Mbits/sec   23   8.48 KBytes
[  4]  20.00-30.01  sec  3.97 MBytes  3.33 Mbits/sec   21   5.66 KBytes
[  4]  30.01-40.00  sec  3.46 MBytes  2.90 Mbits/sec   24   7.07 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-40.00  sec  15.3 MBytes  3.21 Mbits/sec   83             sender
[  4]   0.00-40.00  sec  15.3 MBytes  3.21 Mbits/sec                  receiver

iperf Done.
```

1.3.1. Compare throughputs in cases a, b, c. Why b is better than c?

	A	B	C
Throughput	5.74 Mbits/sec	2.73 Mbits/sec	449 Kbits/sec

Comment in your findings here:

- The throughput of case A is better than case B as the capacity of case A is 10 Mbps while case B is 3 Mbps.
- Although, the capacity of C is better than the capacity of B (10 Mbps to 3 Mbps) but case C has losses which will make retransmission in our system and decreases its throughput making the throughput of B is better than the throughput of C.

1.3.2. Compare throughputs in cases b, c and d. Which is better? Why?

	B	C	D
Throughput	2.73 Mbits/sec	449 Kbits/sec	135 Kbits/sec

Comment in your findings here:

- Comparing the throughput of the 3 cases, we can observe that case B is better as it has no losses so there is no retransmission, while case C and D have losses which will make their throughput affected by the retransmission.
- Although, the capacity of D is better than the capacity of C (100 Mbps to 10 Mbps) but case D has more losses which will make retransmission in our system much worse and decreases its throughput more making the throughput of C is better than the throughput of D.
  Overall, $B > C > D$.

1.3.3. Compare throughputs in cases e and f? Which is better? Why?

	E	F
Throughput	3.94 Mbits/sec	3.21 Mbits/sec

Comment in your findings here:

The transmission is from n7 to n11, using the data path from n5 to n4.

- In case E, the losses in the direction of n4 to n5, so the losses will be only in the ACK sent from n11 to n7 to retransmit the data if doesn't arrive to n7 and as TCP using Go Back N protocol with accumulative ACK, it won't affect the losses as last ACK received at transmitter means that all the packets with sequence number less than the one received with ACK has been received at receiver while all the transmitted and retransmitted data use path from n5 to n4 which has no losses.
- In case F, the losses in the direction of n5 to n4, so the losses will affect all the transmitted and retransmitted data sent from n7 to n11 while the ACK use path from n4 to n5 which has no losses.

From case E and F, we can conclude that the throughput of E will be better as all the data sent doesn't affect by losses.

2. OSPF Lab

Each of the following sections describes an experiment based on the network topology provided by the **Project_OSPF_TCP.imn** file.

The Introductory Lab explained and introduced most of the needed tools like iperf3, vtysh, wireshark, etc. If you have not gone through it, do not start working on the assignment below.

It is better to start the network emulation once and run Wireshark captures. You can reset Wireshark captures from experiment to another.

2.1. OSPF Link Cost Change

1. Stop any running iperf3 clients.

2. Set all links to have zero loss in the two directions with 100 Mbps speed.

3. Run iperf3 between n7 and n11 for a duration of 500 seconds or longer. Identify the path between n7 and n11.

4. Open vtysh on node n5 by opening a bash terminal and typing vtysh. Now we can configure router and link costs.

5. Type the following in vtysh: show ip ospf interface eth1 (To know the ethernet number of each interface you can choose toolbar->view->show->interface names) This displays information about interface eth1. Note its ospf cost.
6. Type the following configure terminal
   - interface eth1 (the interface for the link between n5 and n4)
   - ospf cost 40 (set cost to 40)

2.1.1. Check what happens to the path between n7 and n11 (as seen after steps 3 and 6)? Explain what happens. To help visualize the path choose toolbar->widgets-> Throughput.

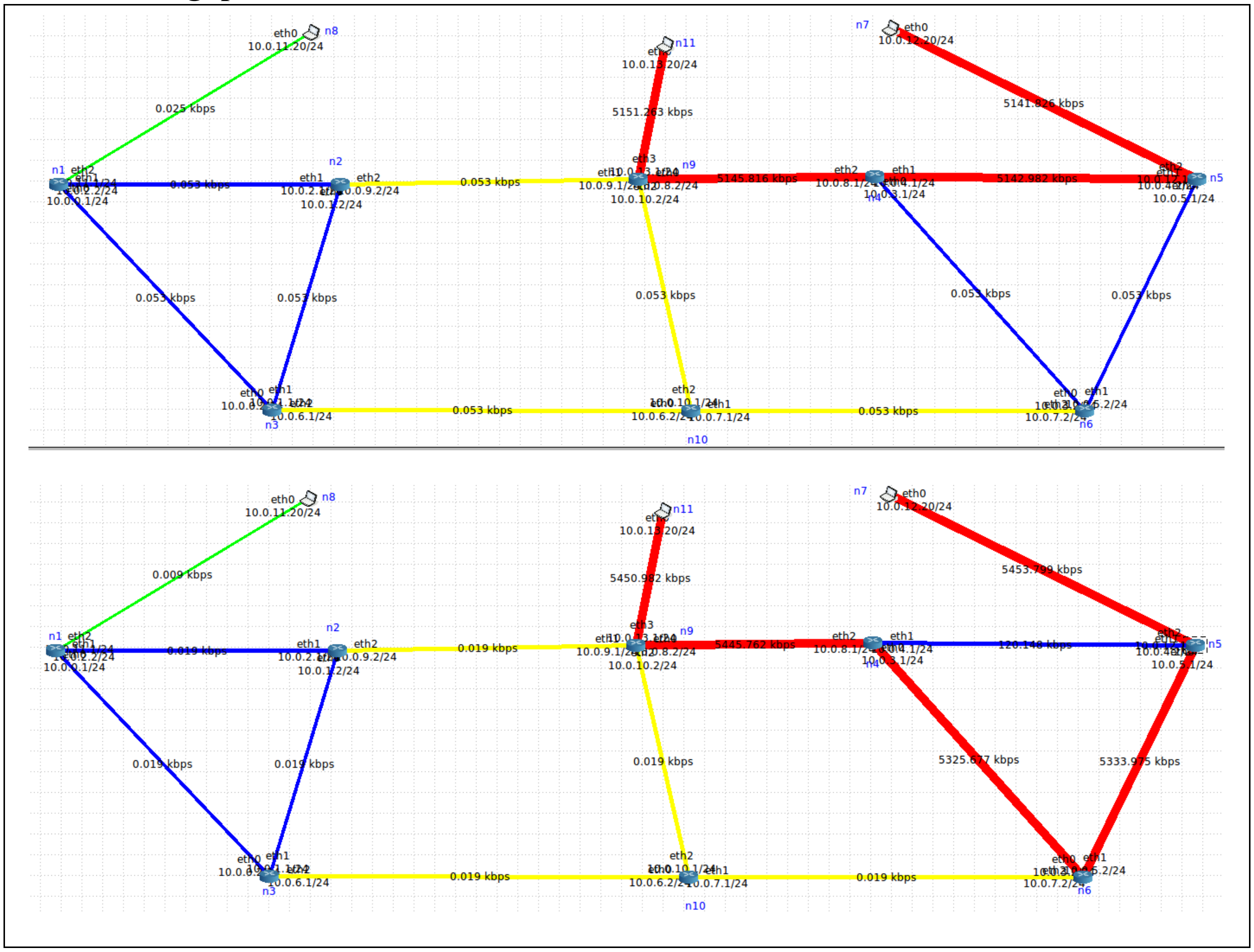


Figure 2-1 before increasing the cost on the left picture and after increasing the cost on right picture.

Comment on your findings:

Before increasing the cost of ethernet 1, OSPF protocol choose the shortest path according to the cost where it moves from n5 to n4 through eth1 directly which has a cost = 10 that represents the minimum path cost, while after increasing the cost of this link to 40, now, the cost of moving from n5 to n6 through ethernet 0 has cost = 10 and then from n6 to n4 has cost = 10 which has a total cost from n5 to n4 equal to 20 instead of moving directly by ethernet 1 from n5 to n4 with cost 40. So, most of the data will move in the new shortest path.

2.1.2 Set the cost of eth1 at node n5 back to 10. Establish two iperf3 connections: one from n7 to n11 and the second from n11 to n7 both for duration of 500 seconds. Now go to node n4 and set interface cost for interface connecting n4 with n5 to 40. What happens in the paths of the two connections? Explain what happens. What do you conclude?

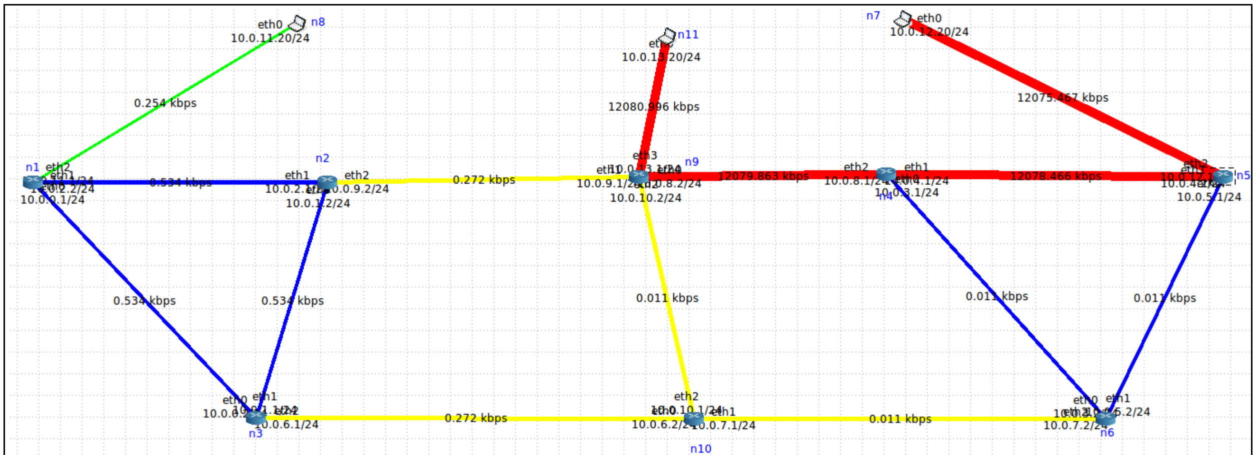


Figure 2-2 sending from n11 to n7 and from n7 to n11 before increasing the cost.

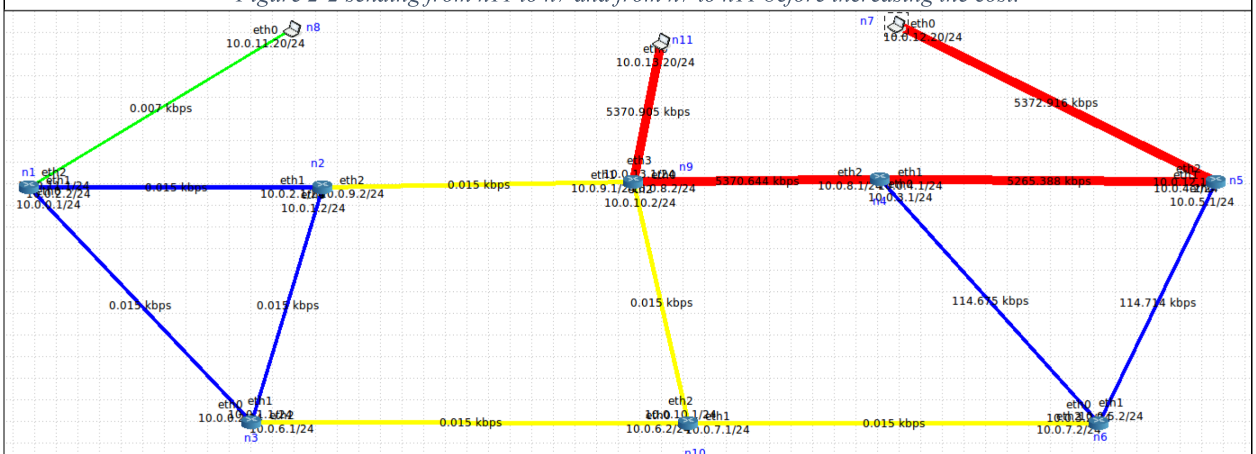


Figure 2-3 sending from n7 to n11 after increasing the cost.

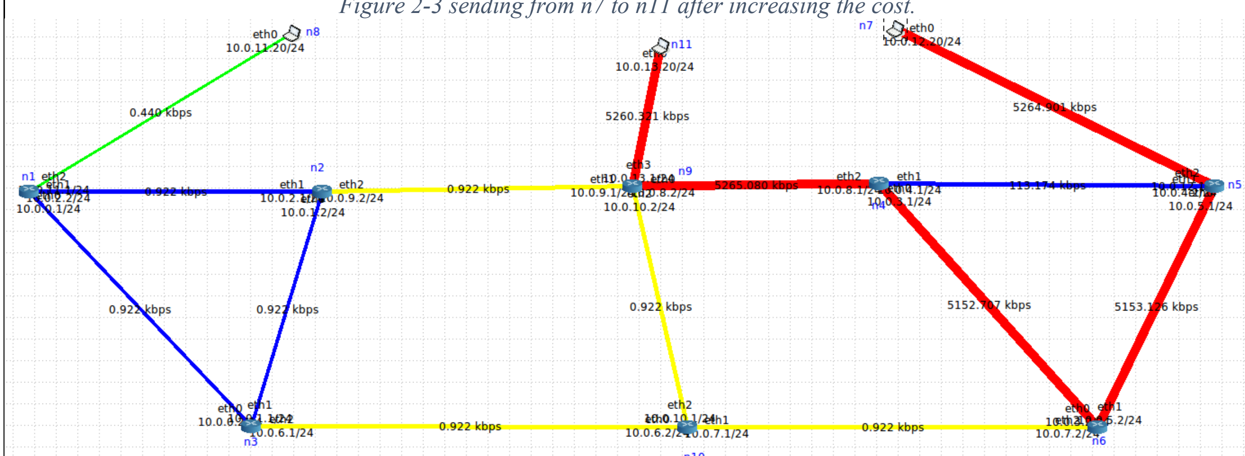


Figure 2-4 sending from n11 to n7 after increasing the cost.

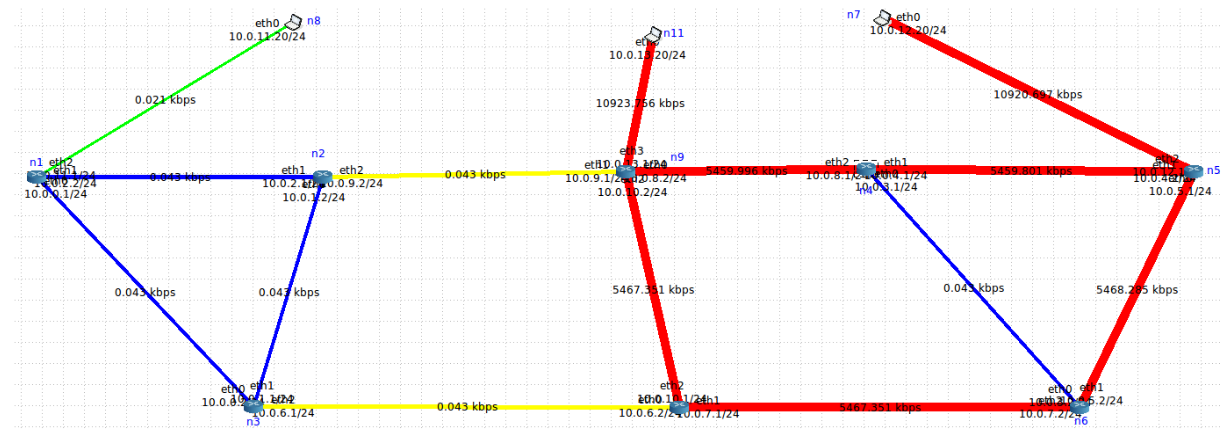


Figure 2-5 sending from n7 to n11 and from n11 to n7 after increasing the cost.

Comment on your findings:

Before increasing the cost, sending in both direction is the same as shown in figure 2-2 as it is the lowest cost path for both directions, while after increasing the cost, sending from n7 to n11 will not be affected as the connection from n5 to n4 (eth1) didn't change as shown in figure 2-3, but sending from n11 to n7 will go from n4 to n6 then n5 instead of n4 to n5 directly as the cost of n4 has been increased to 40 as shown in figure 2-4. If both directions are active at the same time, sending from n7 to n11 will be the same as illustrated before, while sending from n11 to n7 will use the node n10 to avoid using the same path of sending from n7 to n11 as these two paths have the same cost as shown in figure 2-5.

2.2 OSPF Database Updates

1. Start wireshark and have its filter to capture only OSPF related packets.

2. Go to another router say router n2, open vtysh and issue the commands:
   - config terminal
   - show ip ospf database (check this link to understand more)
   - show ip ospf route

3. On router n4, set link cost of eth1 to 20. Capture the link state packets advertised in Wireshark.

4. Go to router n4, go out of vtysh, or open new bash terminal and issue the following commands to disconnect router n4 from the network:

   - ifconfig eth0 down
   - ifconfig eth1 down
   - ifconfig eth2 down

2.2.1 Capture and explain the outputs due to execution of step 2. Why some destinations have more than route in the routing tables?

```
       OSPF Router with ID (10.0.1.2)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.0.0.1        10.0.0.1        1372 0x8000000b 0xf0be 3
10.0.0.2        10.0.0.2        1372 0x8000000d 0x940b 3
10.0.1.2        10.0.1.2        1381 0x8000000b 0x8b0b 3
10.0.3.1        10.0.3.1        1223 0x8000000c 0x432c 3
10.0.3.2        10.0.3.2        1364 0x8000000d 0xa7e1 3
10.0.5.1        10.0.5.1        1155 0x8000000b 0x1662 3
10.0.6.2        10.0.6.2        1382 0x8000000b 0xc8ad 3
10.0.8.2        10.0.8.2        1352 0x8000000d 0x2016 4

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.0.0.2        10.0.0.2         651 0x80000003 0x5bcd
10.0.1.2        10.0.1.2        1691 0x80000003 0x60c4
10.0.2.1        10.0.1.2         220 0x80000004 0x4bd9
10.0.3.2        10.0.3.2         643 0x80000003 0x67b5
10.0.4.2        10.0.5.1         224 0x80000004 0x5eba
10.0.5.1        10.0.5.1        1694 0x80000003 0x65b3
10.0.6.2        10.0.6.2         222 0x80000004 0x45cf
10.0.7.1        10.0.6.2        1693 0x80000003 0x55bd
10.0.8.2        10.0.8.2         221 0x80000004 0x4cc0
10.0.9.1        10.0.8.2         241 0x80000004 0x27e6
```

```
n2# show ip ospf route
============ OSPF network routing table ============
N    10.0.0.0/24           [20] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.2.2, eth1
N    10.0.1.0/24           [10] area: 0.0.0.0
                           directly attached to eth0
N    10.0.2.0/24           [10] area: 0.0.0.0
                           directly attached to eth1
N    10.0.3.0/24           [30] area: 0.0.0.0
                           via 10.0.9.1, eth2
N    10.0.4.0/24           [60] area: 0.0.0.0
                           via 10.0.9.1, eth2
N    10.0.5.0/24           [40] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.9.1, eth2
N    10.0.6.0/24           [20] area: 0.0.0.0
                           via 10.0.1.1, eth0
N    10.0.7.0/24           [30] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.9.1, eth2
N    10.0.8.0/24           [20] area: 0.0.0.0
                           via 10.0.9.1, eth2
N    10.0.9.0/24           [10] area: 0.0.0.0
                           directly attached to eth2
N    10.0.10.0/24          [20] area: 0.0.0.0
                           via 10.0.9.1, eth2
N    10.0.11.0/24          [20] area: 0.0.0.0
                           via 10.0.2.2, eth1
N    10.0.12.0/24          [50] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.9.1, eth2
N    10.0.13.0/24          [20] area: 0.0.0.0
                           via 10.0.9.1, eth2

============ OSPF router routing table =============

============ OSPF external routing table ===========
```

Figure 2-6 left picture represents database and right picture represents routing table.

Comment on your findings:

From the data shown in the database and routing table as shown in figure 2-6, we can observe that the routing table gets the shortest path to the destinations. Knowing that the cost of all links is the same which is equal to 10 except eth1 of node 4 has cost of 40, hence some destinations can have more than one path which have the same total cost, so the router can send the packets in either path of them.

2.2.2 After executing step 3, determine how long it took the network to exchange link state packets and adjust routing tables. (Hint: you can calculate the required time by observing the time of first OSPF update message and the last ACK from Wireshark).

683764	48.600680	10.0.12.1	224.0.0.5	OSPF	80	Hello Packet
683765	48.600683	10.0.12.1	224.0.0.5	OSPF	80	Hello Packet
683768	48.601342	10.0.5.1	224.0.0.5	OSPF	84	Hello Packet
683769	48.601344	10.0.4.2	224.0.0.5	OSPF	84	Hello Packet
683770	48.601346	10.0.12.1	224.0.0.5	OSPF	80	Hello Packet
704896	50.112122	10.0.3.1	224.0.0.5	OSPF	124	LS Update
704897	50.112125	10.0.3.1	224.0.0.5	OSPF	124	LS Update
704898	50.112135	10.0.4.1	224.0.0.5	OSPF	124	LS Update
704899	50.112138	10.0.4.1	224.0.0.5	OSPF	124	LS Update
704900	50.112145	10.0.8.1	224.0.0.5	OSPF	124	LS Update
704901	50.112146	10.0.8.1	224.0.0.5	OSPF	124	LS Update
704903	50.112279	10.0.8.1	224.0.0.5	OSPF	124	LS Update
704909	50.112338	10.0.9.1	224.0.0.5	OSPF	124	LS Update
704910	50.112341	10.0.9.1	224.0.0.5	OSPF	124	LS Update
704911	50.112348	10.0.10.2	224.0.0.5	OSPF	124	LS Update
704912	50.112350	10.0.10.2	224.0.0.5	OSPF	124	LS Update
709143	50.460007	10.0.10.1	224.0.0.5	OSPF	80	LS Acknowledge
709145	50.460293	10.0.7.2	224.0.0.5	OSPF	80	LS Acknowledge
709146	50.460294	10.0.3.2	224.0.0.5	OSPF	80	LS Acknowledge
709147	50.460295	10.0.5.2	224.0.0.5	OSPF	80	LS Acknowledge
709149	50.460323	10.0.4.2	224.0.0.5	OSPF	80	LS Acknowledge
709190	50.464107	10.0.8.2	224.0.0.5	OSPF	80	LS Acknowledge
709191	50.464118	10.0.8.2	224.0.0.5	OSPF	80	LS Acknowledge
709198	50.464292	10.0.8.2	224.0.0.5	OSPF	80	LS Acknowledge
758796	53.882216	fe80::200:ff:feaa:d	ff02::5	OSPF	96	Hello Packet
758797	53.882221	fe80::200:ff:feaa:d	ff02::5	OSPF	96	Hello Packet
758798	53.882233	fe80::200:ff:feaa:e	ff02::5	OSPF	96	Hello Packet
758799	53.882236	fe80::200:ff:feaa:e	ff02::5	OSPF	96	Hello Packet

Figure 2-7 Wireshark response of OSPF packets after a changing the cost in the network.

Comment on your findings:

Initially, the cost of eth1 of node 4 equals to 40, the routers only send hello packets until any change happens. After changing the cost to 20, the routers must send the new link state update to let the network know that there is some change happened and adjust its routing table according to the new shortest path for every destination. The time taken to spread the update = time of last ACK – time of first update = 50.464292 – 50.112122 = 0.35217 sec.

2.2.3 After execution of step 4, identify the new routing table and router database at router n2. Explain the updates in the new routing table and the new database.

```
n2# show ip ospf database

       OSPF Router with ID (10.0.1.2)

                Router Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum  Link count
10.0.0.1        10.0.0.1        1080 0x8000000d 0xecc0 3
10.0.0.2        10.0.0.2        1070 0x8000000f 0x900d 3
10.0.1.2        10.0.1.2        1109 0x8000000d 0x870d 3
10.0.3.1        10.0.3.1         118 0x80000018 0xa649 1
10.0.3.2        10.0.3.2         102 0x80000010 0xeaab 3
10.0.5.1        10.0.5.1          82 0x8000000e 0x7f07 3
10.0.6.2        10.0.6.2        1111 0x8000000d 0xc4af 3
10.0.8.2        10.0.8.2          72 0x80000010 0x2226 4

                Net Link States (Area 0.0.0.0)

Link ID         ADV Router      Age  Seq#       CkSum
10.0.0.2        10.0.0.2         310 0x80000005 0x57cf
10.0.1.2        10.0.1.2        1339 0x80000005 0x5cc6
10.0.2.1        10.0.1.2        1729 0x80000005 0x49da
10.0.5.1        10.0.5.1        1342 0x80000005 0x61b5
10.0.6.2        10.0.6.2        1731 0x80000005 0x43d0
10.0.7.1        10.0.6.2        1341 0x80000005 0x51bf
10.0.9.1        10.0.8.2        1740 0x80000005 0x25e7
10.0.10.2       10.0.8.2         360 0x80000005 0x59ac
```

```
n2# show ip ospf route
============ OSPF network routing table ============
N    10.0.0.0/24           [20] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.2.2, eth1
N    10.0.1.0/24           [10] area: 0.0.0.0
                           directly attached to eth0
N    10.0.2.0/24           [10] area: 0.0.0.0
                           directly attached to eth1
N    10.0.3.0/24           [40] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.9.1, eth2
N    10.0.4.0/24           [80] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.9.1, eth2
N    10.0.5.0/24           [40] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.9.1, eth2
N    10.0.6.0/24           [20] area: 0.0.0.0
                           via 10.0.1.1, eth0
N    10.0.7.0/24           [30] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.9.1, eth2
N    10.0.8.0/24           [20] area: 0.0.0.0
                           via 10.0.9.1, eth2
N    10.0.9.0/24           [10] area: 0.0.0.0
                           directly attached to eth2
N    10.0.10.0/24          [20] area: 0.0.0.0
                           via 10.0.9.1, eth2
N    10.0.11.0/24          [20] area: 0.0.0.0
                           via 10.0.2.2, eth1
N    10.0.12.0/24          [50] area: 0.0.0.0
                           via 10.0.1.1, eth0
                           via 10.0.9.1, eth2
N    10.0.13.0/24          [20] area: 0.0.0.0
                           via 10.0.9.1, eth2

============ OSPF router routing table =============

============ OSPF external routing table ===========
```

Figure 2-8 left picture represents database and right picture represents routing table.

Comment on your findings:

- In the database table, the number of links connected to 10.0.3.1 reduced from 3 to 1 as n4 is disconnected, therefore 10.0.3.1 only has link to 10.0.3.2 instead of 10.0.3.2 & 10.0.4.1 & 10.0.8.1.
- In the routing table, the cost of 10.0.3.0 increased from 30 to 40 & the cost of 10.0.4.0 increased from 40 to 60, as after disconnecting n4 some links will get another shortest path which will be with higher cost than with connecting n4.

2.3 OSPF Link State Advertisement Periodicity

This part is on your own. No instructions are given. Bring router n4 ethernet interfaces back up.

Stop any previous iperf3 connections. Let Wireshark capture focus on Ospfv2 only (IPv4).

Apply the filter (ospf.msg.lsupdate || ospf.msg.lsack) && ip.addr == 224.0.0.5

2.3.1 Bring router n4 down and then up again and determine how long it took the network to recognize the router is down/up.

Filter: msg.lsupdate || ospf.msg.lsack) && ip.addr == 224.0.0.5 Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
2888	100.460902	10.0.8.1	224.0.0.5	OSPF	172	LS Update
2889	100.460911	10.0.8.1	224.0.0.5	OSPF	172	LS Update
2898	100.461842	10.0.9.1	224.0.0.5	OSPF	100	LS Update
2899	100.461844	10.0.10.2	224.0.0.5	OSPF	100	LS Update
2900	100.461926	10.0.1.2	224.0.0.5	OSPF	100	LS Update
2902	100.461932	10.0.1.2	224.0.0.5	OSPF	100	LS Update
2904	100.461945	10.0.2.1	224.0.0.5	OSPF	100	LS Update
2905	100.461948	10.0.2.1	224.0.0.5	OSPF	100	LS Update
2906	100.462008	10.0.6.2	224.0.0.5	OSPF	100	LS Update
3736	139.886829	10.0.0.1	224.0.0.5	OSPF	120	LS Acknowledge
3737	139.886916	10.0.1.2	224.0.0.5	OSPF	100	LS Acknowledge
3738	139.886920	10.0.2.2	224.0.0.5	OSPF	100	LS Acknowledge
3739	139.886922	10.0.1.1	224.0.0.5	OSPF	80	LS Acknowledge
3740	139.886926	10.0.6.1	224.0.0.5	OSPF	120	LS Acknowledge
3741	139.887094	10.0.9.2	224.0.0.5	OSPF	100	LS Acknowledge
3742	139.887108	10.0.9.2	224.0.0.5	OSPF	100	LS Acknowledge
3743	139.887269	10.0.1.2	224.0.0.5	OSPF	100	LS Acknowledge
3744	139.887280	10.0.9.2	224.0.0.5	OSPF	100	LS Acknowledge

Figure 2-9 Wireshark response of OSPF packets after changing the state of n4 from up to down.

Filter: msg.lsupdate || ospf.msg.lsack) && ip.addr == 224.0.0.5 Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Length	Info
259	15.828157	10.0.8.1	224.0.0.5	OSPF	112	LS Update
260	15.828164	10.0.8.1	224.0.0.5	OSPF	112	LS Update
262	15.828387	10.0.8.1	224.0.0.5	OSPF	112	LS Update
265	15.828569	10.0.9.1	224.0.0.5	OSPF	112	LS Update
266	15.828578	10.0.9.1	224.0.0.5	OSPF	112	LS Update
267	15.828593	10.0.10.2	224.0.0.5	OSPF	112	LS Update
268	15.828597	10.0.10.2	224.0.0.5	OSPF	112	LS Update
270	15.828917	10.0.9.1	224.0.0.5	OSPF	112	LS Update
272	15.828920	10.0.10.2	224.0.0.5	OSPF	112	LS Update
273	15.829051	10.0.1.2	224.0.0.5	OSPF	112	LS Update
274	15.829060	10.0.1.2	224.0.0.5	OSPF	112	LS Update
275	15.829076	10.0.2.1	224.0.0.5	OSPF	112	LS Update
276	15.829082	10.0.2.1	224.0.0.5	OSPF	112	LS Update
821	26.066780	10.0.7.2	224.0.0.5	OSPF	80	LS Acknowledge
822	26.100484	10.0.8.2	224.0.0.5	OSPF	80	LS Acknowledge
823	26.100530	10.0.8.2	224.0.0.5	OSPF	80	LS Acknowledge
824	26.100793	10.0.8.2	224.0.0.5	OSPF	80	LS Acknowledge
825	26.160029	10.0.5.1	224.0.0.5	OSPF	80	LS Acknowledge
826	26.160085	10.0.5.1	224.0.0.5	OSPF	80	LS Acknowledge
827	26.160338	10.0.5.1	224.0.0.5	OSPF	80	LS Acknowledge
828	26.269086	10.0.0.1	224.0.0.5	OSPF	80	LS Acknowledge
829	26.269116	10.0.0.1	224.0.0.5	OSPF	80	LS Acknowledge
830	26.269729	10.0.0.1	224.0.0.5	OSPF	80	LS Acknowledge

Figure 2-10 Wireshark response of OSPF packets after changing the state of n4 from down to up.

Comment on your findings:

Initially router n4 is up, then after disconnected all its interfaces and bring them down, the network sends the new link state packets to update the routing table, the time taken to spread the update = 139.887280 – 100.460902 = 39.426378 as shown in figure 2-9.
Bringing back router n4 by reconnecting all its interface, the time taken to spread the update = 26.269729 – 15.828157 = 10.441572 sec.

2.3.2 Now, the following is a bonus question (extra marks). Explain the noticed behavior of the link state update packet storms (flooding) where it generally takes longer times for the LSP exchange to die out (i.e. LSP exchange stops) when a router is down as compared with the case when the router is up again.

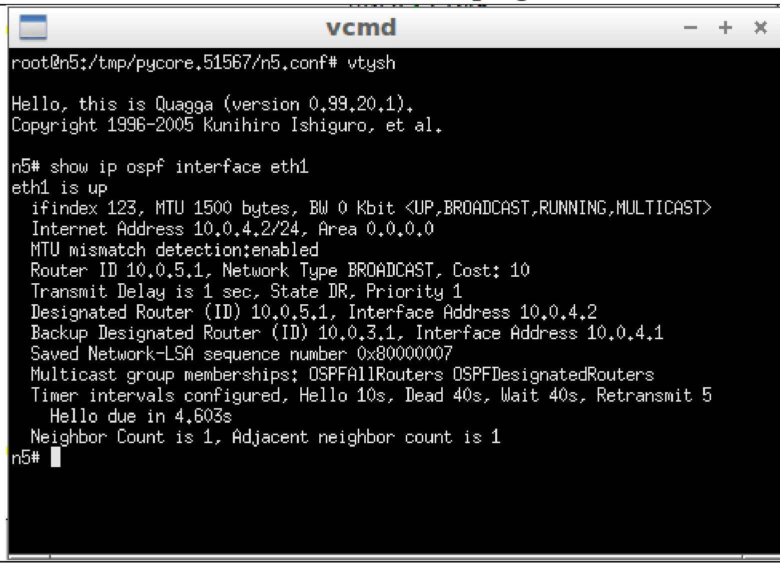


Figure 2-11 eth1 interface of router n5.

Comment on your findings:

From figure 2-11, the dead time equals to 40 sec, means that to know that the link is disconnected, the node sends packets and wait for the ACK from the other node for 40 sec, if it didn't get anything from that link in this interval of time, the node understands that this link is disconnected, and this can be proved when disconnecting router n4 from the previous question.
Also, the hello time equals to 10 sec, means that when router enters the network, it will send Hello packets for 10 sec to let the network knows its existence and update its routing table, and this can be proved when reconnecting router n4 and bringing it back to network from the previous question.