CO323 - LAB SESSION 02

KANEWALA U.C.H.

E/13/175

SEMESTER 5

04/04/2017

Server IP Address: 10.40.18.102

Client IP Address: 10.40.18.60

A. Use iperf and generate tcp and udp traffic. Show the client’s and the server’s outputs at both occasions separately.

<u>TCP Traffic:</u>

TCP Port: 5001

Client’s Output:

```
e13175@ce-410:~$ iperf -c 10.40.18.102
------------------------------------------------------------
Client connecting to 10.40.18.102, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 10.40.18.60 port 48916 connected with 10.40.18.102 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   820 MBytes   688 Mbits/sec
e13175@ce-410:~$ iperf -c 10.40.18.102
------------------------------------------------------------
Client connecting to 10.40.18.102, TCP port 5001
TCP window size: 85.0 KByte (default)
------------------------------------------------------------
[  3] local 10.40.18.60 port 48917 connected with 10.40.18.102 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   973 MBytes   816 Mbits/sec
e13175@ce-410:~$
```

Figure 1: TCP Client's Output with TCP Traffic

Server’s Output:

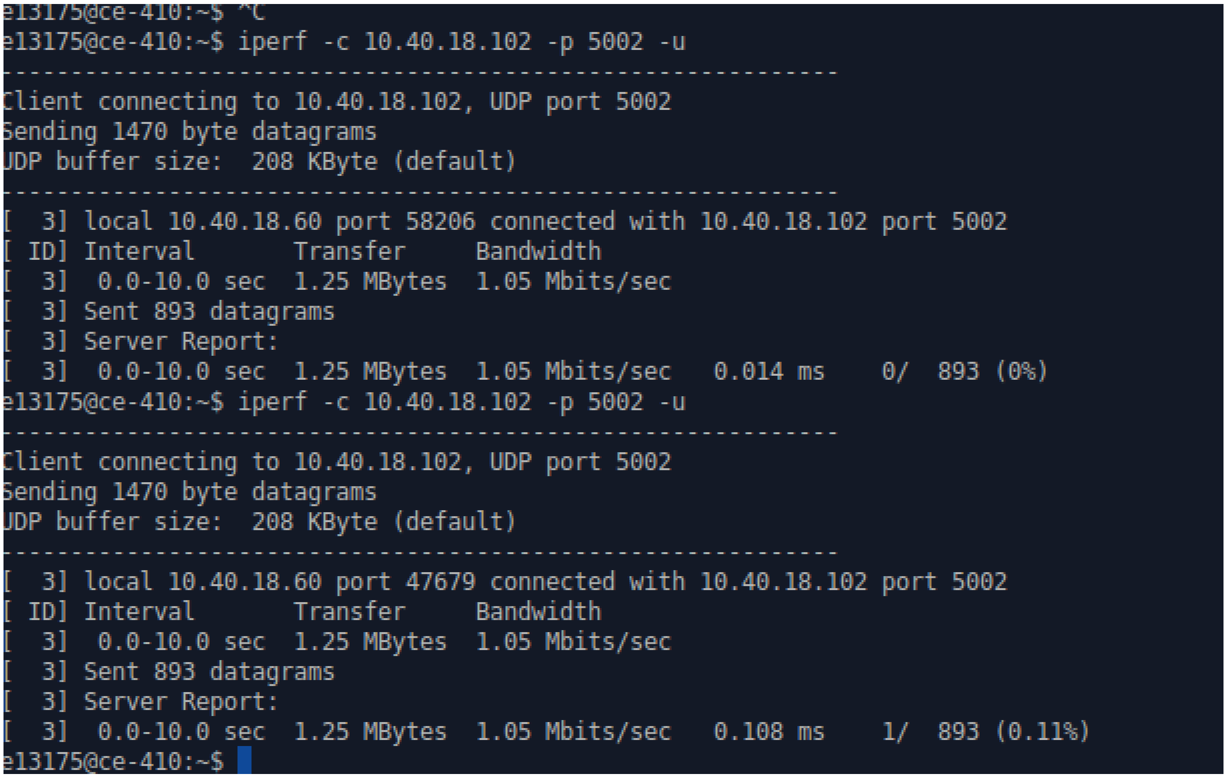
```
^Ce13175@ce-410:~$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 10.40.18.60 port 5001 connected with 10.40.18.102 port 35175
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  1004 MBytes   841 Mbits/sec
[  5] local 10.40.18.60 port 5001 connected with 10.40.18.102 port 35177
[  5]  0.0-10.0 sec  1009 MBytes   845 Mbits/sec
```

Figure 2: Server's Output with TCP Traffic

UDP Traffic:

UDP Port: 5002

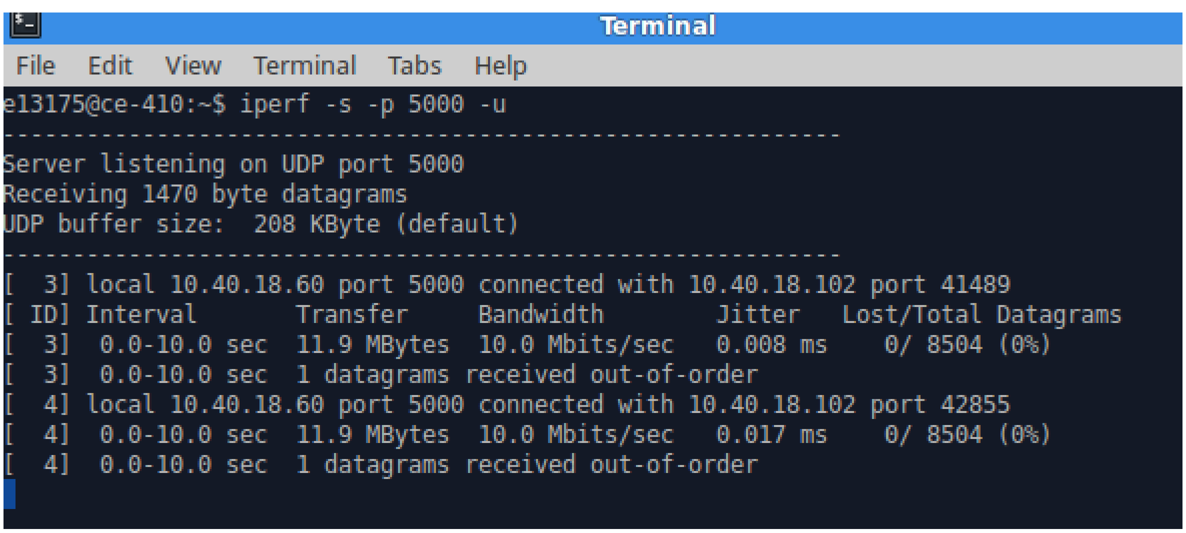
Client's Output:



```
e13175@ce-410:~$ ^C
e13175@ce-410:~$ iperf -c 10.40.18.102 -p 5002 -u
------------------------------------------------------------
Client connecting to 10.40.18.102, UDP port 5002
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.40.18.60 port 58206 connected with 10.40.18.102 port 5002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.014 ms    0/  893 (0%)
e13175@ce-410:~$ iperf -c 10.40.18.102 -p 5002 -u
------------------------------------------------------------
Client connecting to 10.40.18.102, UDP port 5002
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.40.18.60 port 47679 connected with 10.40.18.102 port 5002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[  3] Sent 893 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec   0.108 ms    1/  893 (0.11%)
e13175@ce-410:~$
```

Figure 3: Client's Output with UDP Traffic

Server's Output:



```
Terminal
File  Edit  View  Terminal  Tabs  Help
e13175@ce-410:~$ iperf -s -p 5000 -u
------------------------------------------------------------
Server listening on UDP port 5000
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.40.18.60 port 5000 connected with 10.40.18.102 port 41489
[ ID] Interval       Transfer     Bandwidth        Jitter   Lost/Total Datagrams
[  3]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec   0.008 ms    0/ 8504 (0%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
[  4] local 10.40.18.60 port 5000 connected with 10.40.18.102 port 42855
[  4]  0.0-10.0 sec  11.9 MBytes  10.0 Mbits/sec   0.017 ms    0/ 8504 (0%)
[  4]  0.0-10.0 sec  1 datagrams received out-of-order
```

Figure 4: Server's Output with UDP Traffic

B. Capture the TCP three way handshake using Wireshark

The TCP three way handshake can be identified by the first three TCP packets. TCP three way handshake is often known as “SYN, SYN-ACK, ACK” as there are three messages transmitted by TCP to establish and begin a TCP session.

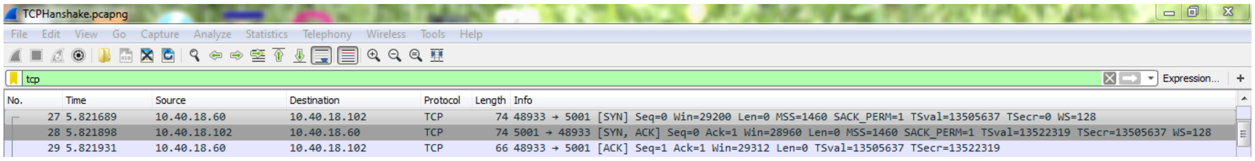


Figure 5: TCP Three Way Handshake

Initially, the client will send a TCP synchronize packet (SYN packet) with SYN flag set to 1.

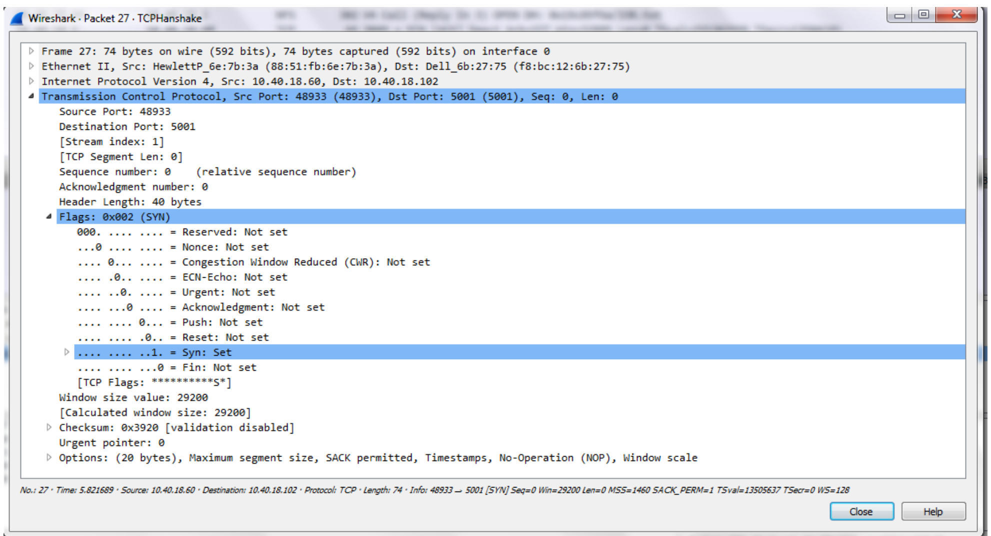


Figure 6: SYN Flag Set

Secondly, the server will send a Synchronize-Acknowledgment packet (SYN-ACK packet) indicating that the client’s SYN packet was received and the server wishes to establish a TCP connection with client by setting flags SYN and ACK to 1.

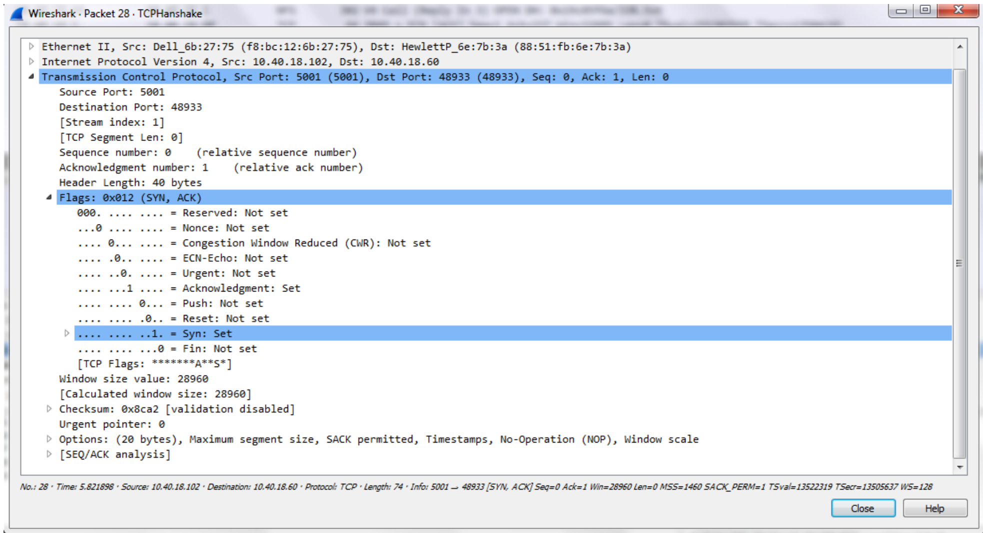


Figure 7: SYN and ACK Flags Set

Thirdly, the client will send an Acknowledgment packet (ACK packet) indicating the the server's SYN-ACK packet was received by setting the ACK flag to 1.

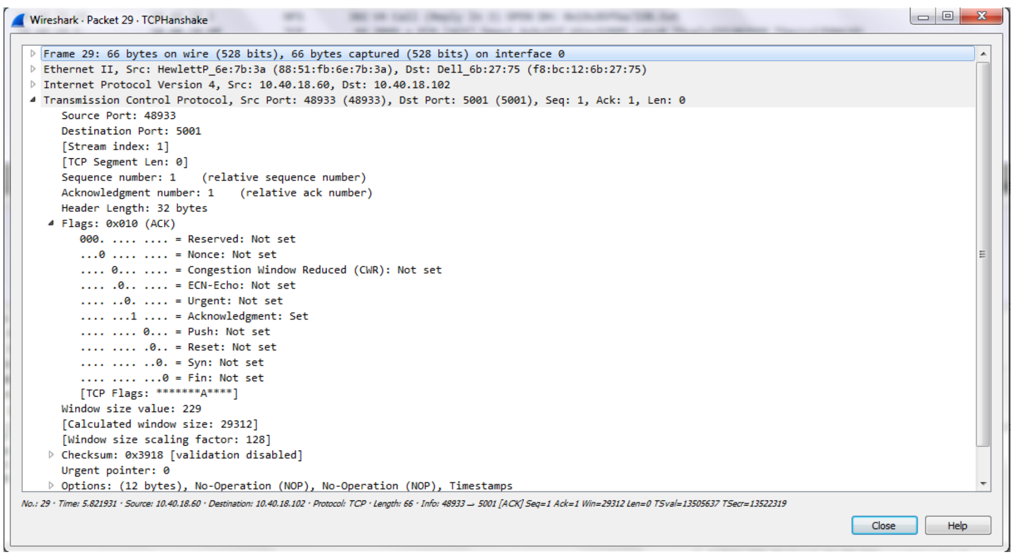


Figure 8: ACK Flag Set

C. Calculate the TCP connection establishment delay by using Wireshark.

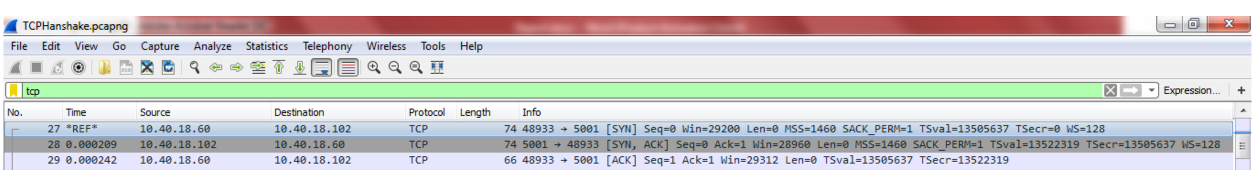


Figure 9: TCP Connection Establishment Delay

We can calculate the TCP connection establishment delay by setting the time of first TCP packet with SYN flag as a reference time. This will make that frame a new time zero origin. Therefore the TCP connection

establishment delay can be directly read by the time given at the last TCP packet of the three way handshake.
Therefore, TCP connection establishment delay = 0.000242 seconds = 0.242 milliseconds

D. In this communication, the initial sequence numbers are shown as zero in each direction. Clarify the reason behind that.

A 32 bit sequence number is maintained by the client on either side of a TCP session. This number is maintained in order to keep track of the amount of data the client has sent. The initial sequence number is a random number when the host initiates the TCP session. It can be a value between 0 – 2^{32}. Protocol analyzers like Wireshark typically display relative sequence numbers instead of actual sequence numbers. This is done as it is much easier to keep track of relatively small numbers than the actual numbers sent. By disabling the relative sequence number display on Wireshark we can observe the actual sequence numbers of these packets.

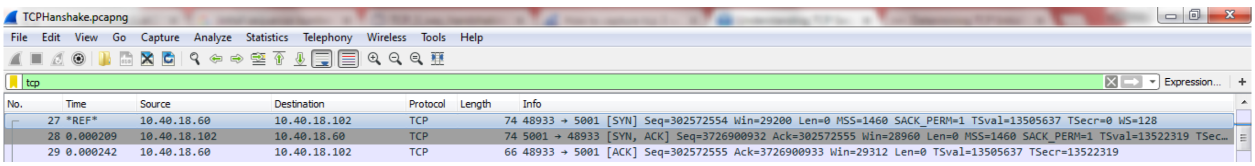


Figure 10: Displaying Actual Sequence Numbers

E. What TCP options are carried on the SYN packet on your trace?

TCP options are:

- Maximum segment size
- TCP SACK permitted option
- Timestamps
- No-Operation
- Window scale

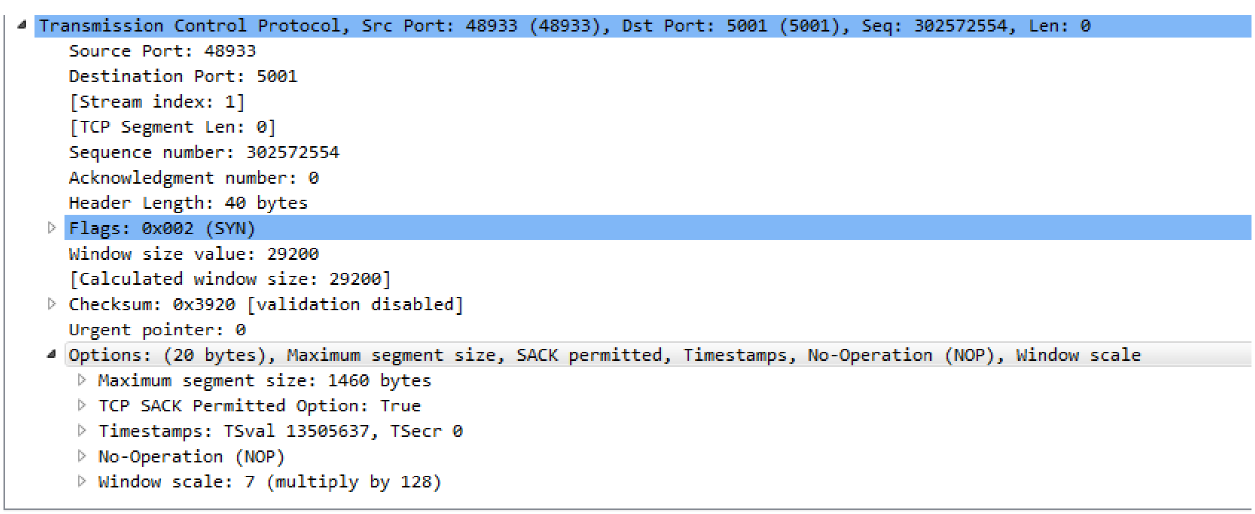
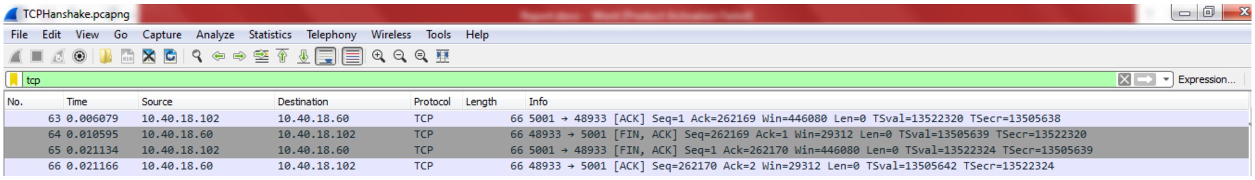


Figure 11: TCP Options

F. Identify the TCP connection teardown message sequence in the trace.



No.	Time	Source	Destination	Protocol	Length	Info
63	0.006079	10.40.18.102	10.40.18.60	TCP	66	5001 → 48933 [ACK] Seq=1 Ack=262169 Win=446080 Len=0 TSval=13522320 TSecr=13505638
64	0.010595	10.40.18.60	10.40.18.102	TCP	66	48933 → 5001 [FIN, ACK] Seq=262169 Ack=1 Win=29312 Len=0 TSval=13505639 TSecr=13522320
65	0.021134	10.40.18.102	10.40.18.60	TCP	66	5001 → 48933 [FIN, ACK] Seq=1 Ack=262170 Win=446080 Len=0 TSval=13522324 TSecr=13505639
66	0.021166	10.40.18.60	10.40.18.102	TCP	66	48933 → 5001 [ACK] Seq=262170 Ack=2 Win=29312 Len=0 TSval=13505642 TSecr=13522324

Figure 12: Tear Down Message Sequence

The highlighted packets indicate the tear down message sequence of the TCP connection. On tear down the client sends its final packet with the FIN flag set. The server accepts the termination of TCP connection by increasing the Acknowledgment number by one and setting the FIN flag. Therefore, the teardown message sequence could be identified by the FIN flag set packets.

G. Draw the traffic pattern for both TCP and UDP.

TCP Traffic:

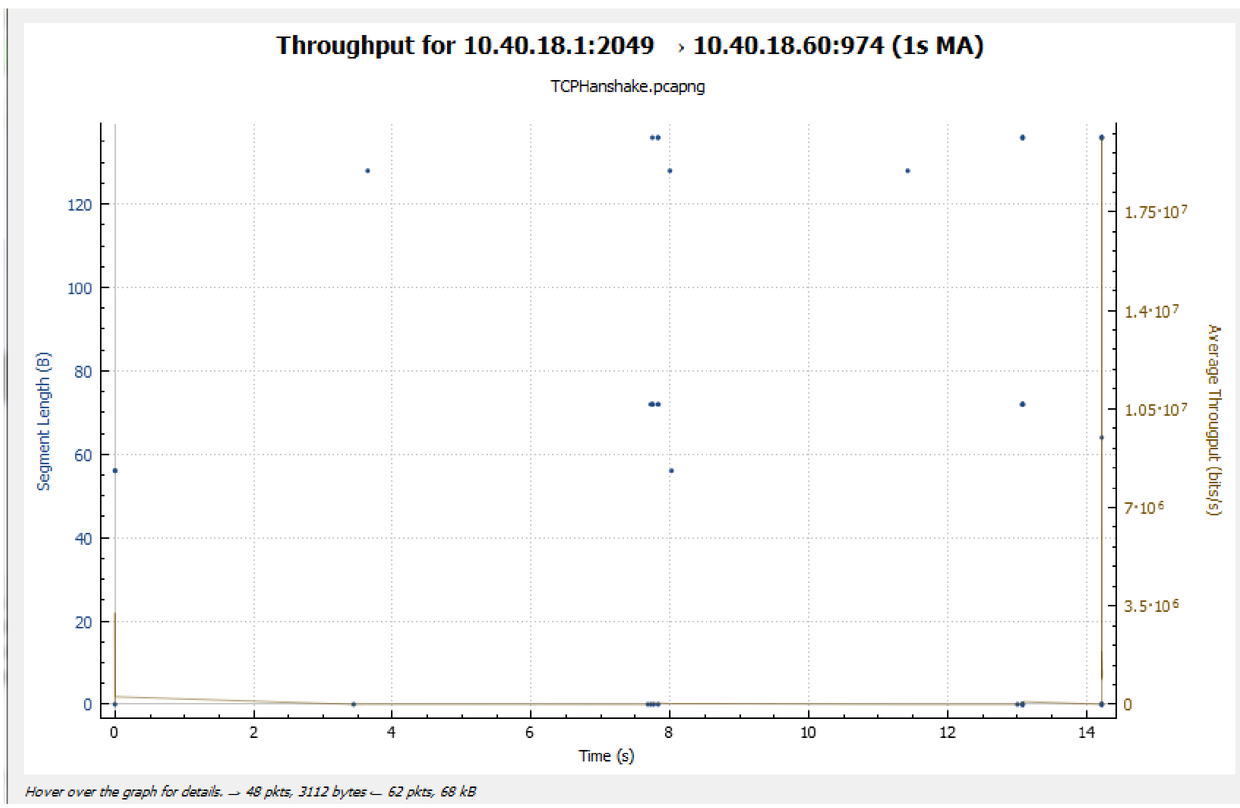


Figure 13: TCP Traffic Pattern

UDP Traffic:

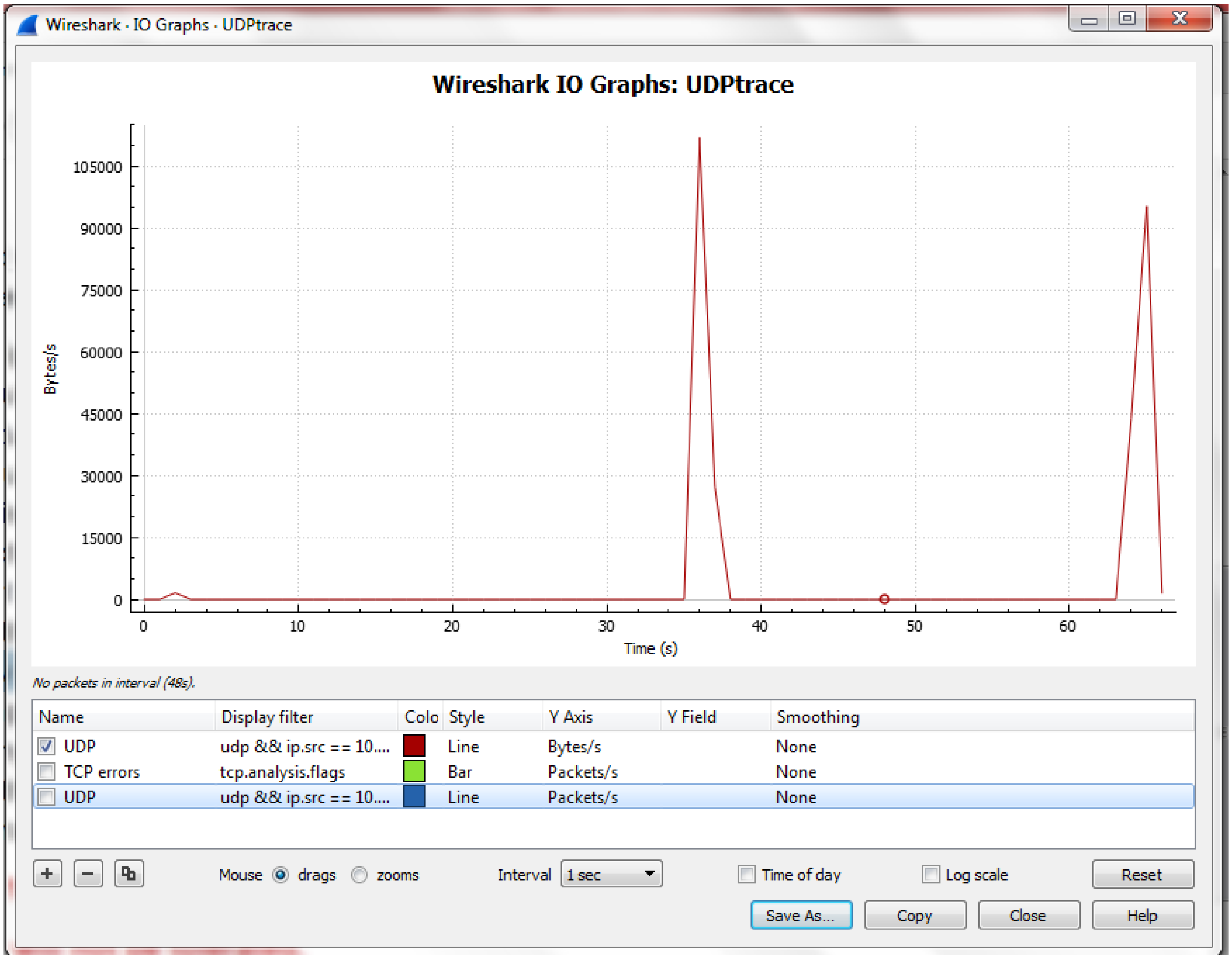


Figure 14: UDP Traffic Pattern

H. Compare the UDP vs TCP throughput and comment on it.

From the two graphs above we can determine the average throughput of TCP and UDP traffic. The same results are seen when we consider packet/sec or Bytes/sec. It can be seen that UDP has a higher throughput than TCP from the above two graphs. Also when considering transfer bandwidths UDP has lesser bandwidth consumption compared to TCP packets. Therefore more packets could be transmitted at one second. Therefore throughput of UDP is high.

I. Change the MTU size and redraw the TCP graph for MTU=500, 1000, 1500

MTU = 500:

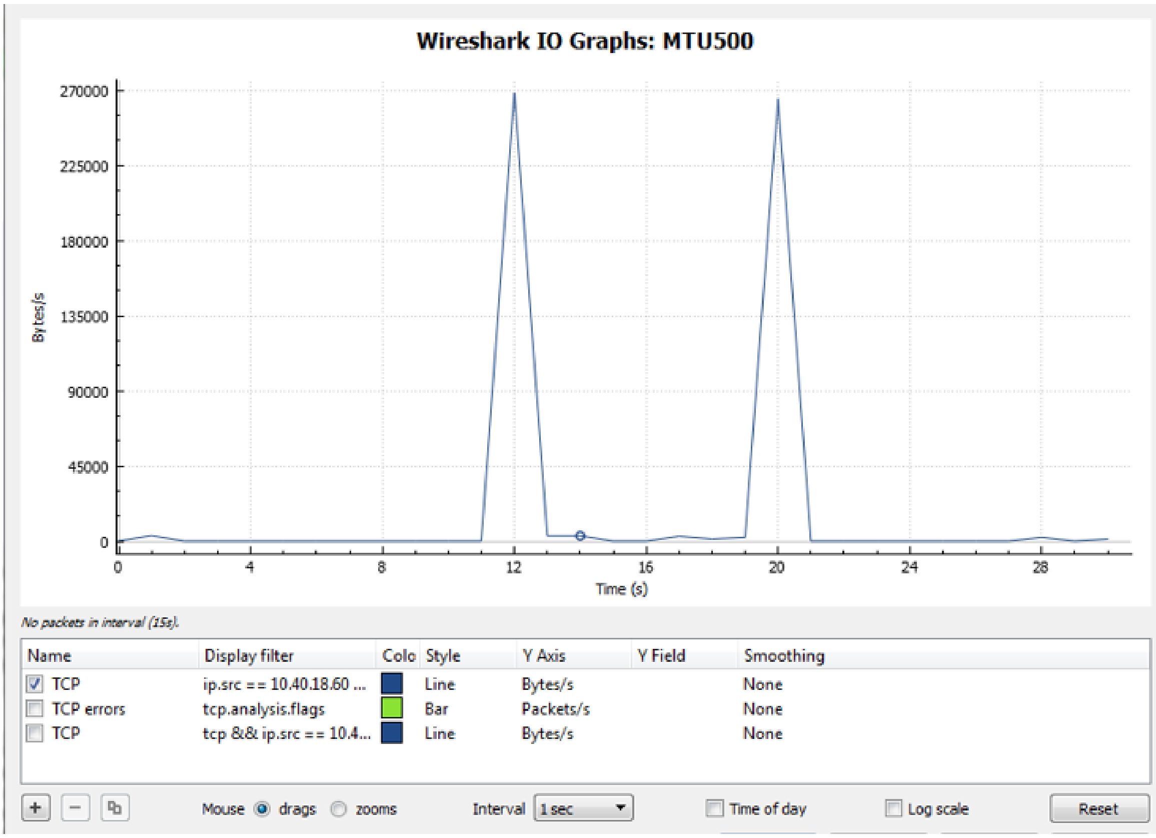


Figure 15: IO Graph for MTU = 500

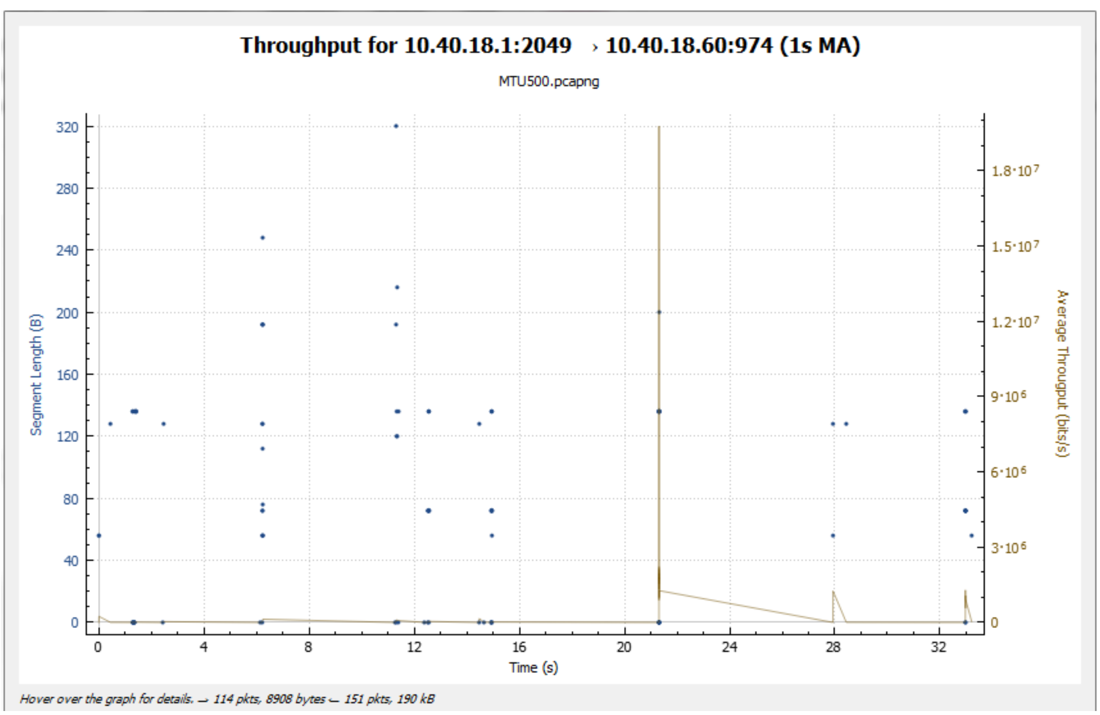


Figure 16: Throughput Graph for MTU = 500

MTU = 1000:

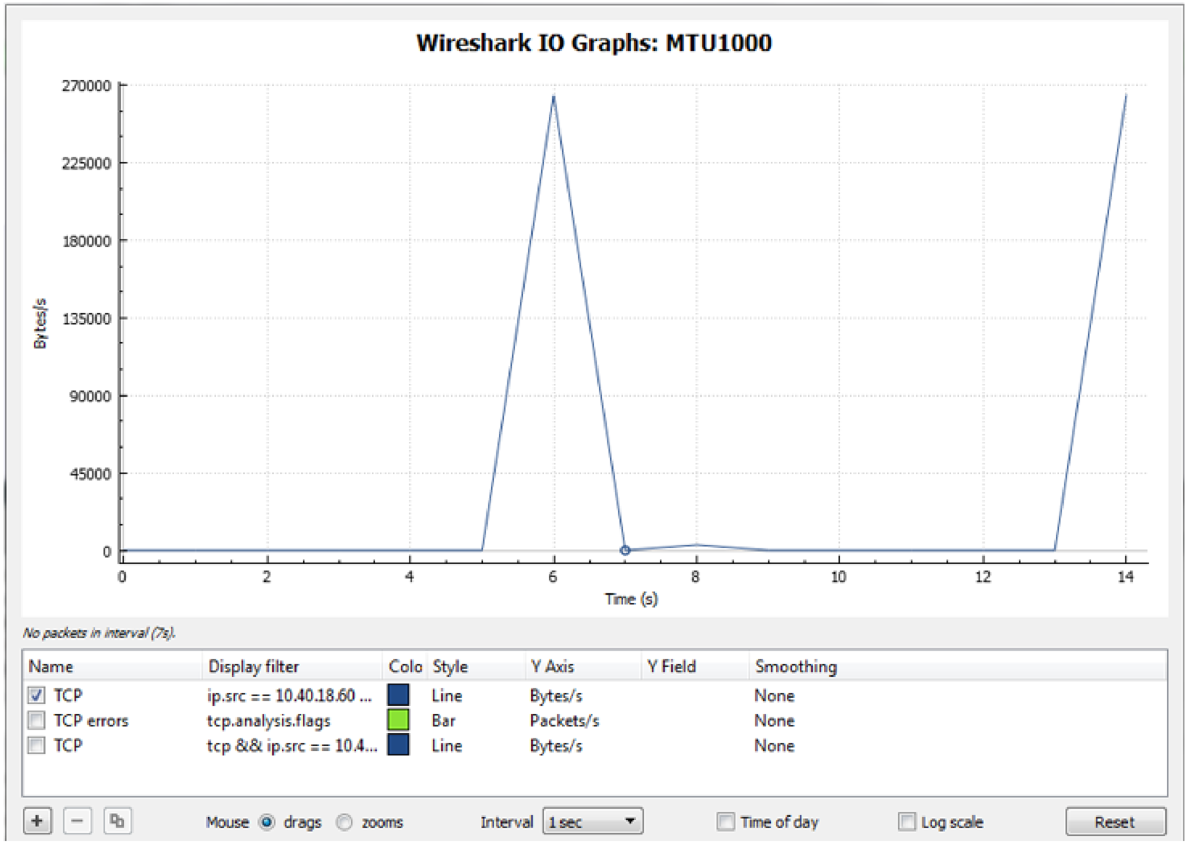


Figure 17: TCP Graph for MTU = 1000

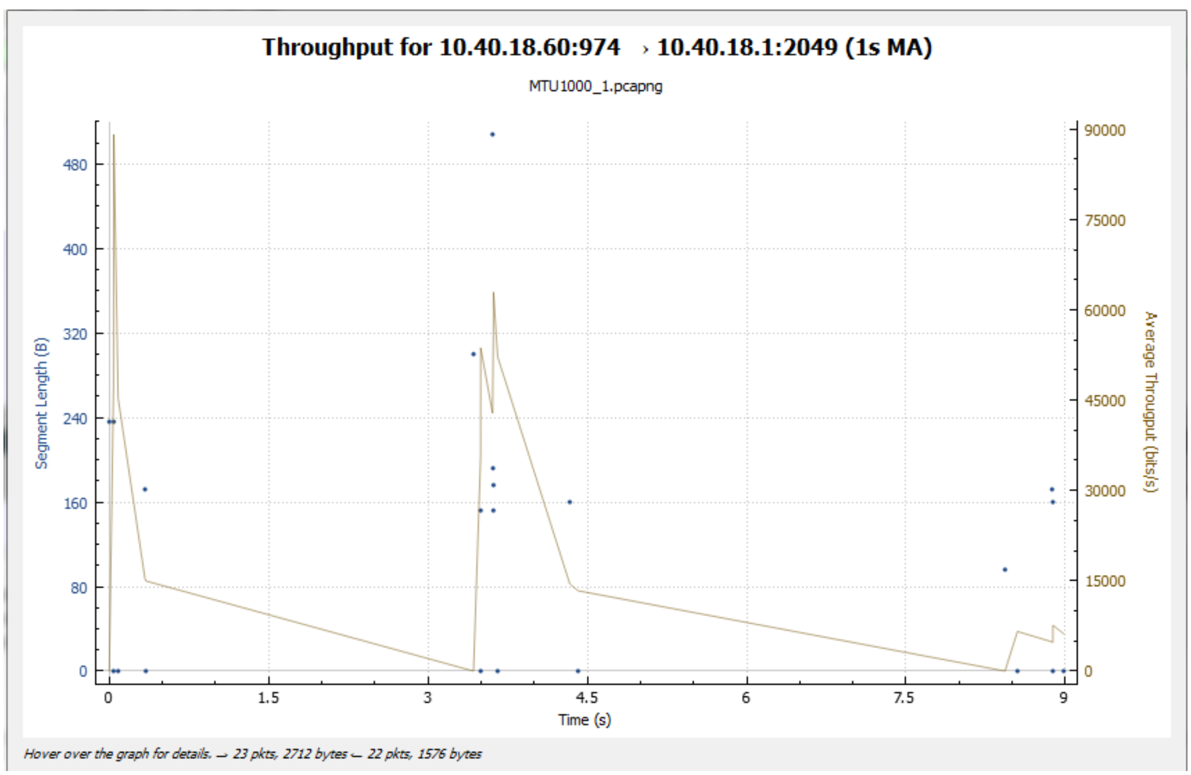


Figure 18: Throughput Graph for MTU = 1000

MTU = 1500:

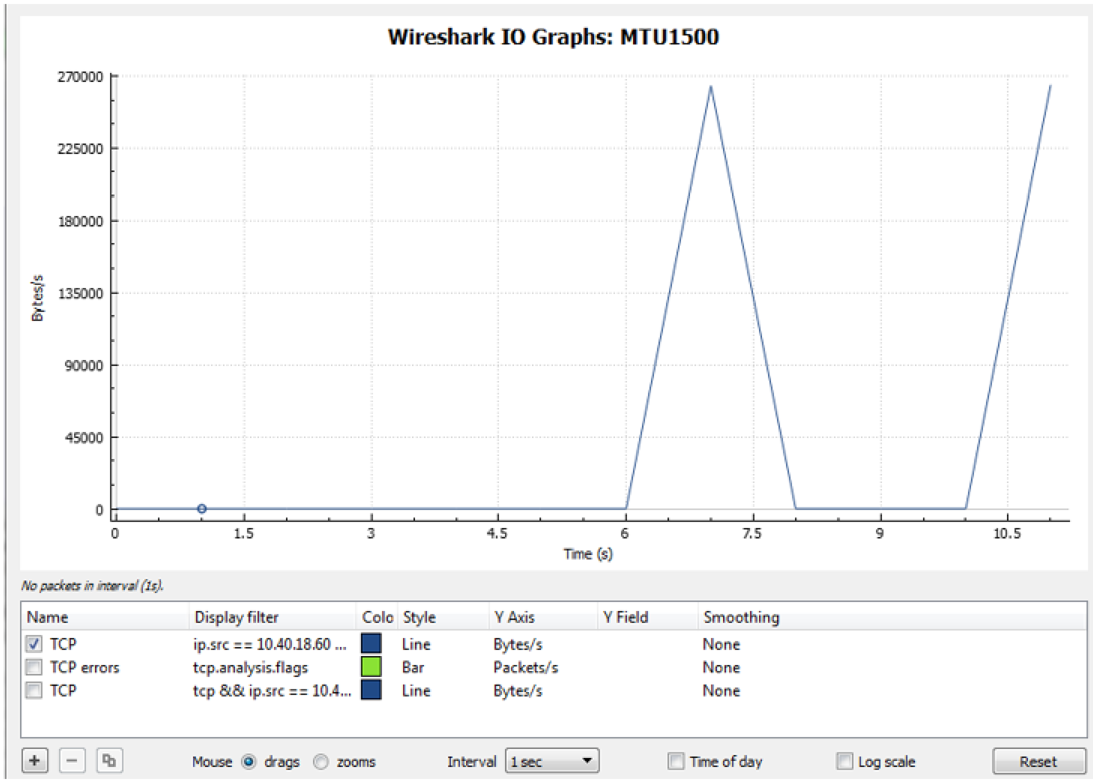


Figure 19: TCP Graph for MTU = 1500

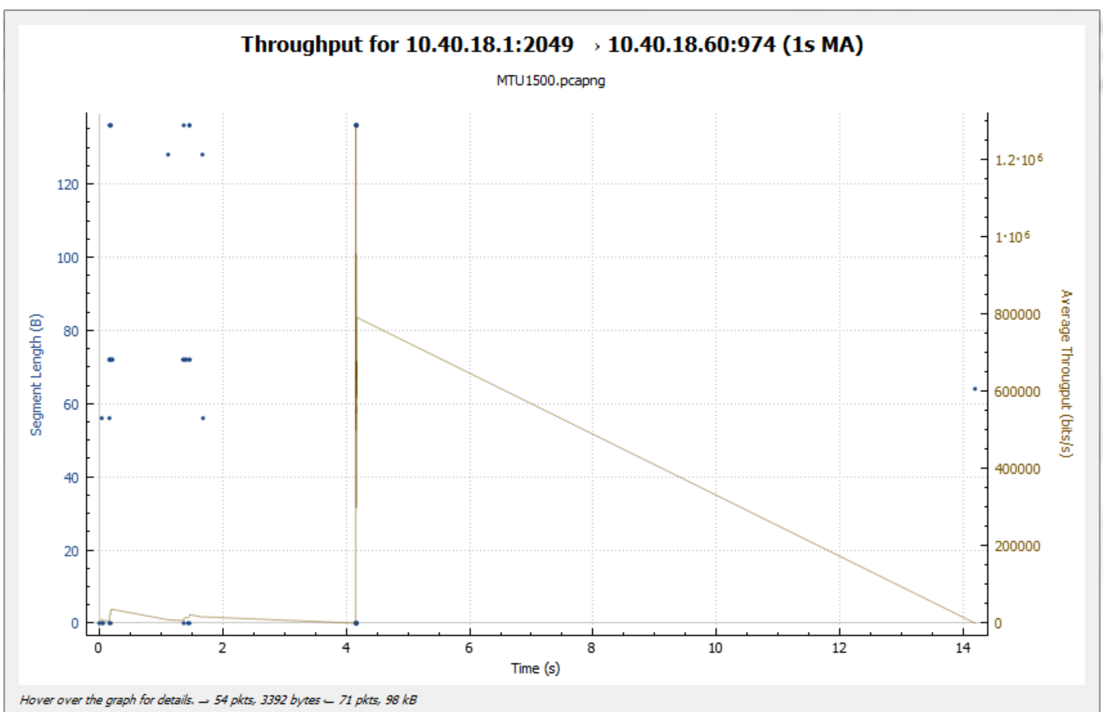


Figure 20: Throughput Graph for MTU = 1500

J. Identify the reason behind the shown traffic patterns (whether it comes to a saturation, if not why)?