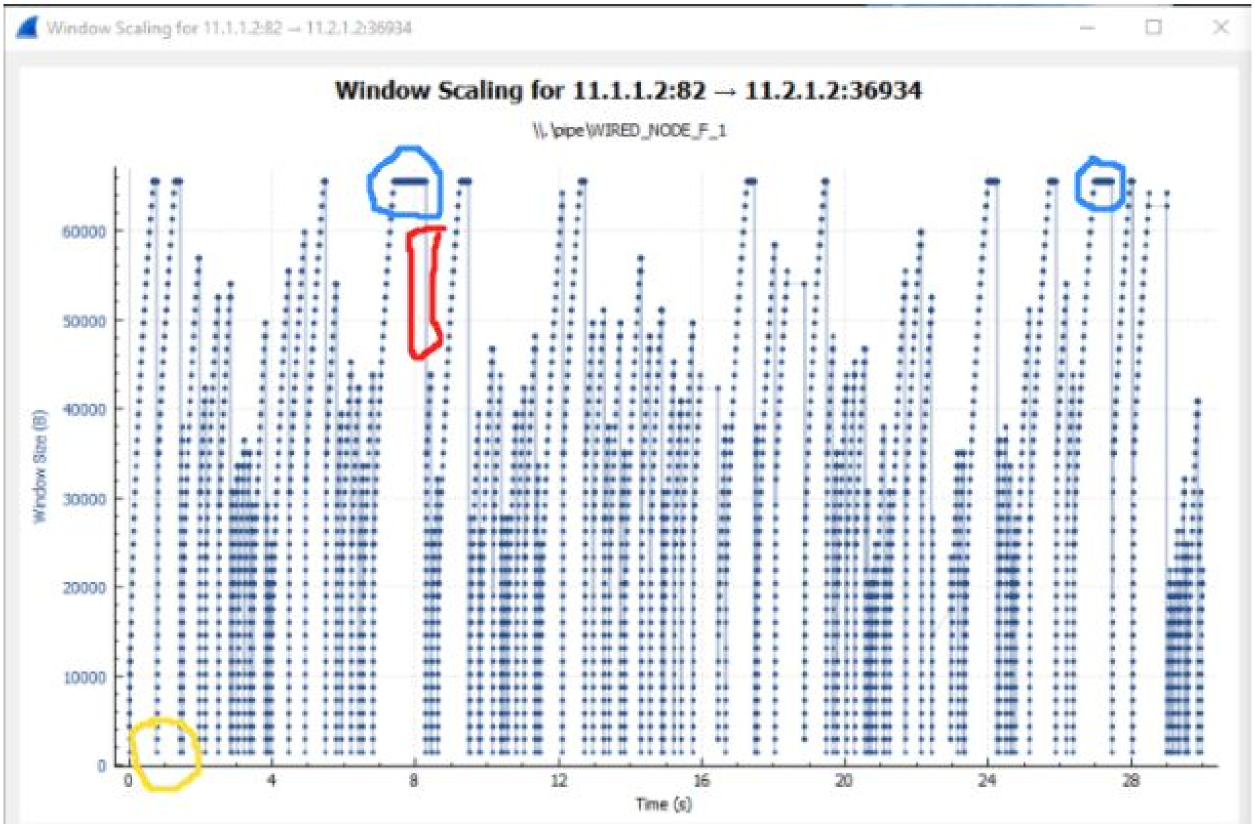
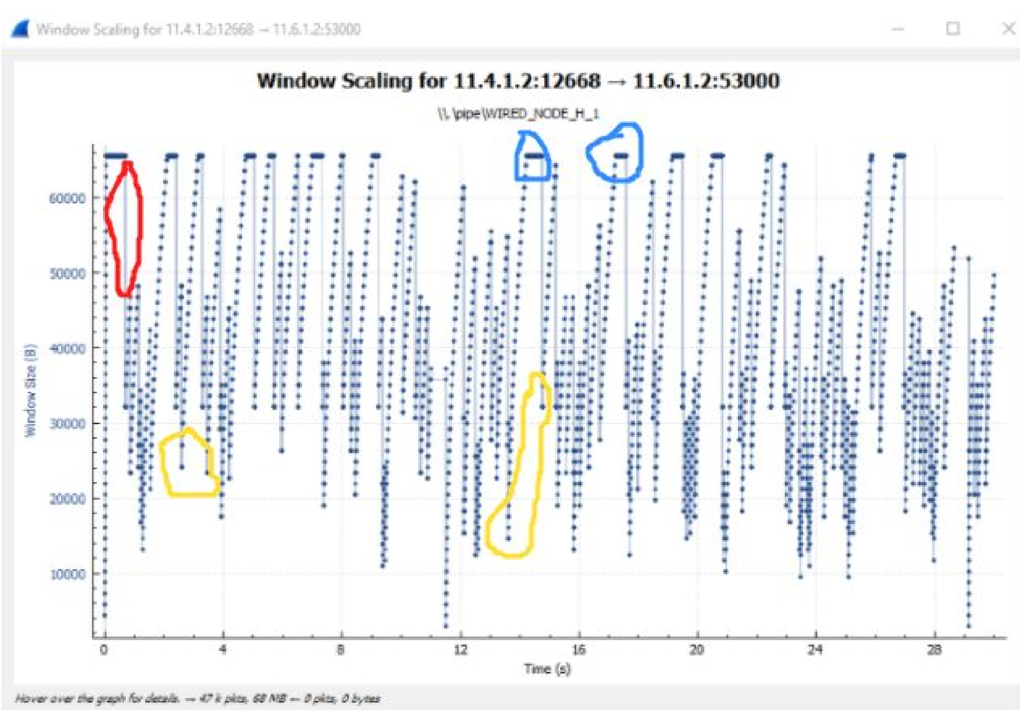
1.1)

Node f - Tahoe



Node H - Reno



- Slow Start are the points where the plot has fallen down and starts to rise again. (Yellow circle in the graph)
- Packet loss and timeout are present at the positions where the plot starts to fall. (Red shapes in the graph)
- TCP Congestion Avoidance are the positions of the flat peaks (Blue circles in the graph)

1.2)

- Tahoe immediately on facing a severe congestion it reduces the MSS to 1 and begins the whole process from there.However, this reduction of rate to 1 might not be needed.
- Reno on the other hand halves it every time until a suitable point is reached hence we often see more flat peaks (congestion control mechanism) in Reno and in Tahoe we see more steep peaks and an immediate fall from there.

2.1)

Application_metrics ☐ Detailed View

Application Id	Application Name	Packet transmitted	Packet received	Throughput (Mbps)	Delay(microsec)
1	APP1_CBR	249	249	0.581664	52877.576373
2	APP2_CBR	249	249	0.581664	52981.725472

- Since it is a symmetrical network and all the links and applications have similar properties , hence we observe the same throughput in both cases and a similar delay as well.
- The information is sent one by one via the router for both the sending nodes.

3-

Application_Metrics_Table

Application_metrics ☐ Detailed View

Application Id	Application Name	Packet transmitted	Packet received	Throughput (Mbps)	Delay(microsec)
1	APP1_VIDEO	499	499	0.262905	186.705892
2	APP2_CBR	499	499	0.199600	114.355271
3	APP3_CBR	499	499	0.199600	25292.954764
4	APP4_VIDEO	499	498	0.252925	179.010924

- Since CBR applications are TCP applications and the Video Applications are UDP ones hence the throughput of Video Applications is more than that of CBR. This is because UDP doesn't have control packets hence it sends data directly to the receiver, whereas TCP has a lot of control packets hence less data is sent for the same time period.

- Both the UDP apps have almost the same delay but one of the CBR apps has a very high delay. This is because the video apps consume almost the entire bandwidth and are creating a bottleneck for the cbr apps. Hence one of the CBR apps has a low average delay because it clears the bottleneck but the other faces a significant delay.

4.2-

1. The sequence numbers of the first 6 segments starting from HTTP POST as the first segment are as follows: 164041, 1, 1, 1, 164091.
The time stamps can be seen in ascending order in the second screenshot below.
The ACK numbers are 1, 162309, 164041, 164091, 164091, 731 respectively.
RTT for the first six segments are 160ms,189ms, 245ms,158ms,188ms,190ms respectively.

Formula for EstimatedRTT = 0.875 * PreviousEstimatedRTT + 0.125 * CurrentRTT
Estimated RTT for first segment=RTT for the first segment;
Estimated RTT for first segment=160ms
Estimated RTT for second segment=163ms
Estimated RTT for third segment=173ms
Estimated RTT for fourth segment=171ms
Estimated RTT for fifth segment=173ms
Estimated RTT for sixth segment=175ms

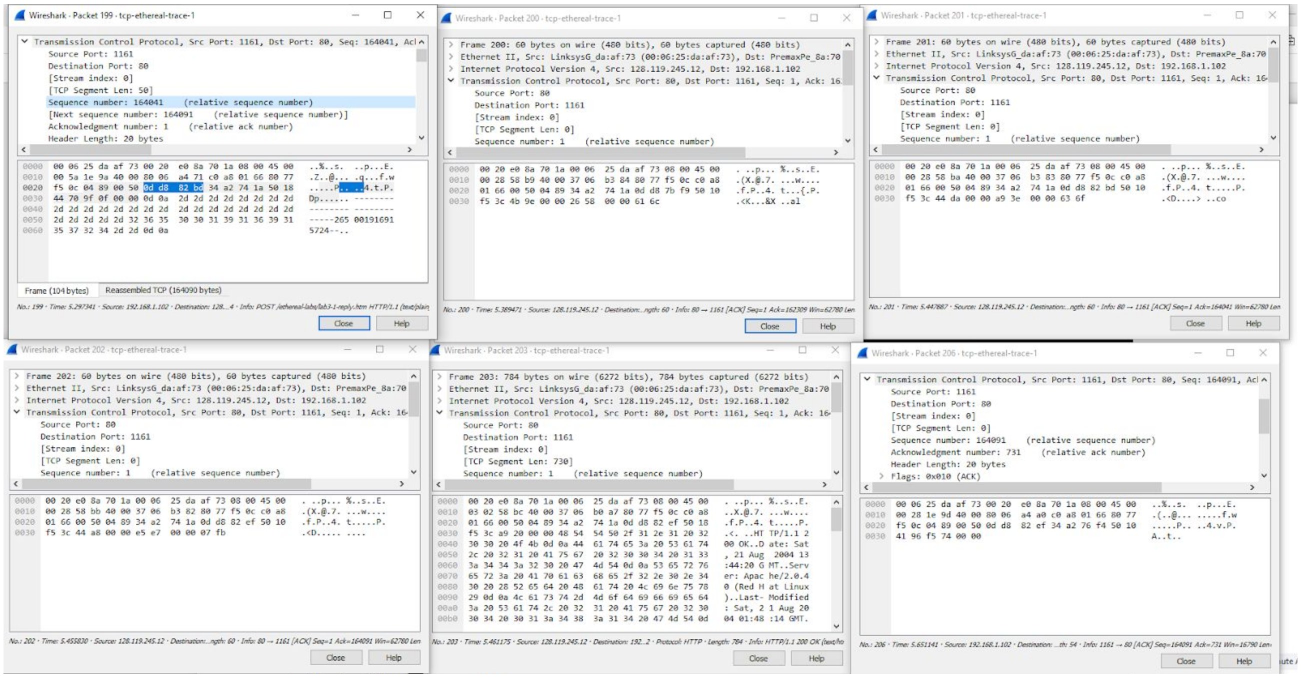
2. Segment lengths are 50,0,0,0,730,0 respectively.

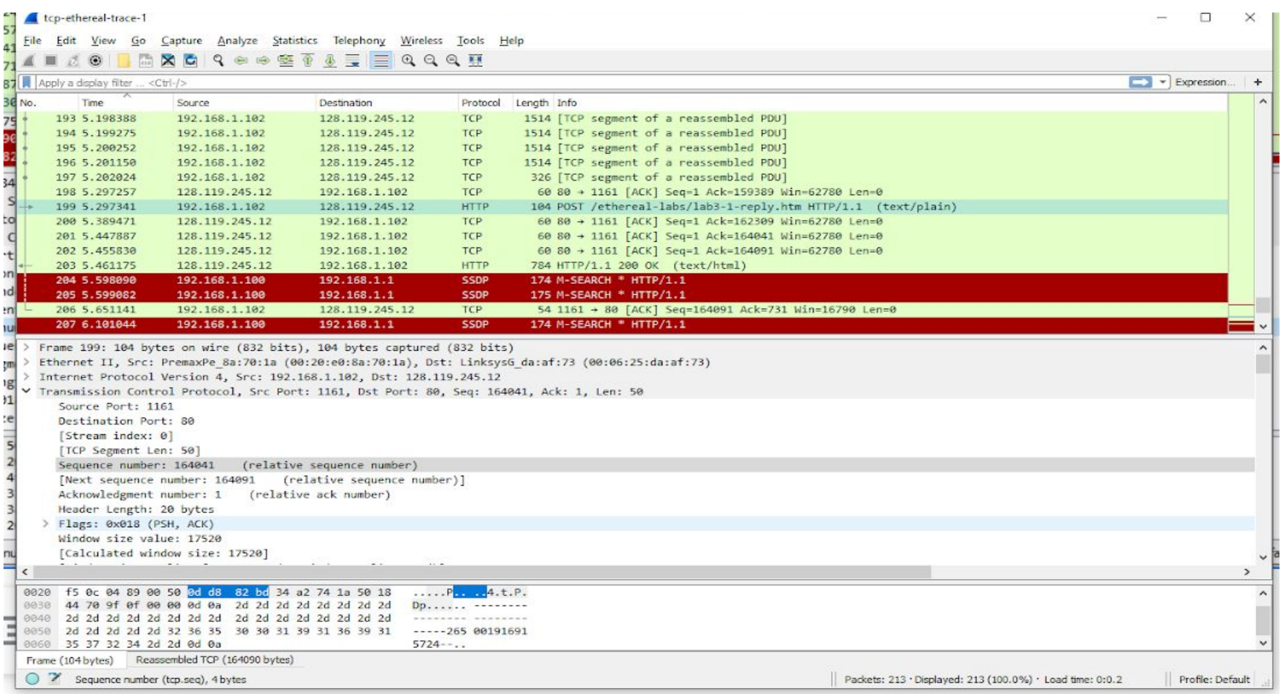
3. The minimum amount of available buffer space advertised at the received for the entire trace is indicated last ACK from the server, its value is 16790 bytes, For all of the remaining ACKs the window size is at maximum capacity of 62780 bytes as seen in the screenshot. The sender is throttled due to lack of receiver buffer space by inspecting this trace.

4. We checked the sequence numbers of ACKs in the trace file. These turned out to be the same. Since the sequence number did not increase, there are retransmitted segments in the trace file. OK status was received after these 3 ACKs and not right after the HTTP POST. This can be seen in the screenshot below.

5. The difference between the acknowledged sequence numbers of two consecutive ACKs indicates the data received by the server between these two ACKs.

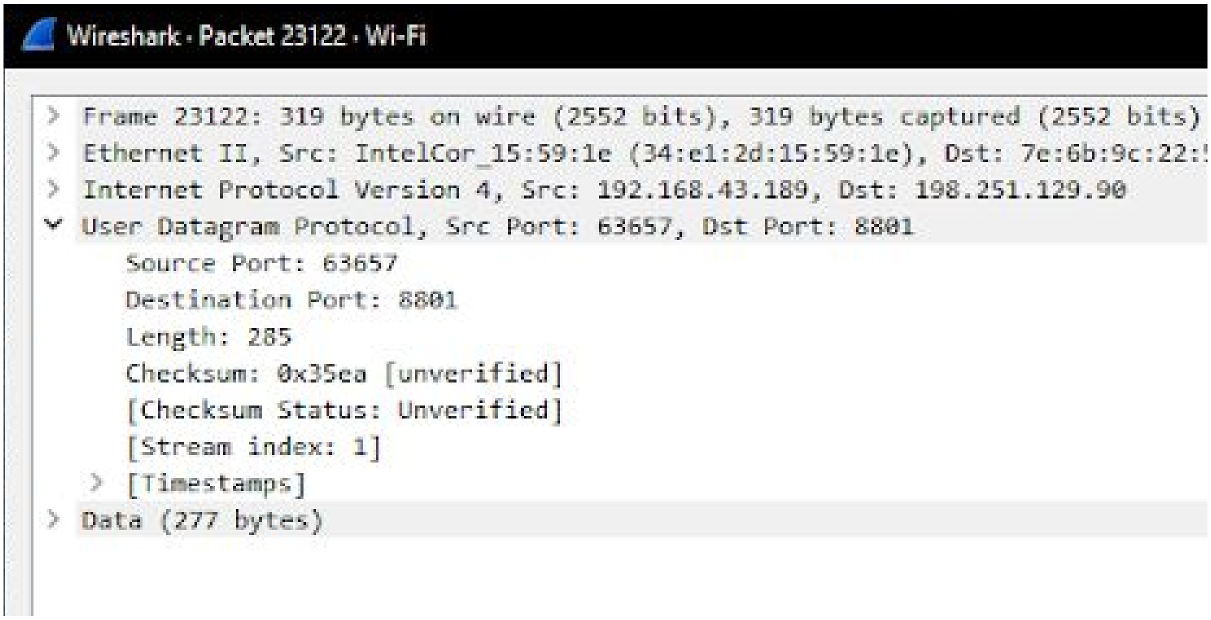
6. Throughput = file size/ (time difference b/w first TCP and last ACK segment)
= 178KB/ (7.595s)
= 23434.75 Bytes/ sec



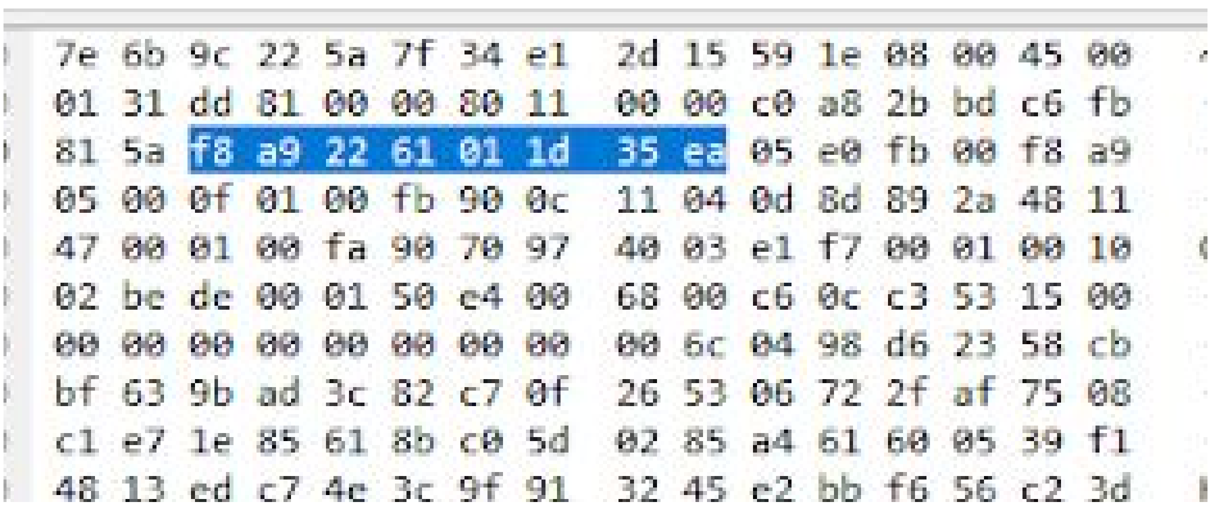


5-

1. UDP header contains 4 fields: source port, destination port, length, checksum as shown in the screenshot below.

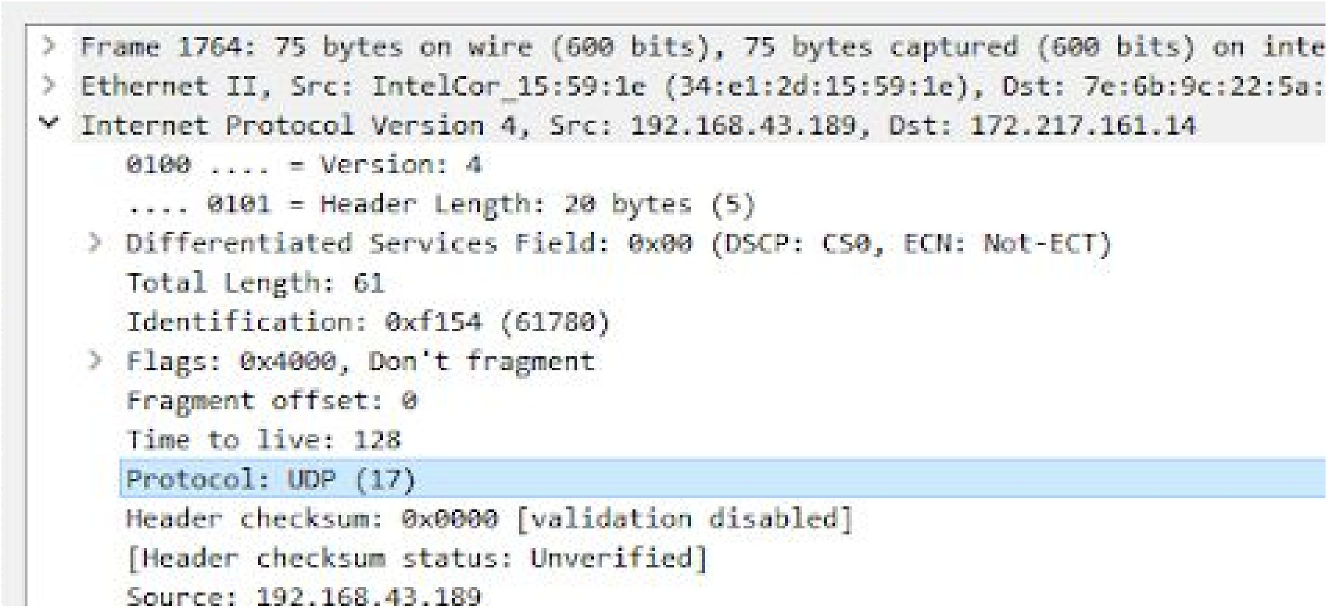


2. The length of UDP header is 8 bytes. There are 4 fields each having a length of 2 bytes.



3. The length field specifies the number of bytes in the UDP segment (header plus data).

4. The maximum number of bytes that can be included in a UDP payload is (2^16 – 1) bytes plus the header bytes. This gives 65535 bytes – 8 bytes = 65527 bytes.
5. A **port number** is a 16-bit unsigned integer, thus ranging from 0 to 65535.
6. The protocol number for UDP is 17 in decimal notation which in hexadecimal notation is 0x11.



7. DNS command has been used to capture UDP packets, because :
    a. UDP is much faster than TCP since it doesn't have control packets.
    b. Reduces load on DNS servers as they don't have to deal with multiple handshake requests.
    c. DNS requests are generally very small and fit well within UDP segments.
    d. UDP is not reliable, but reliability can be added on the application layer. An application can use UDP and can be reliable by using a timeout and resend at the application layer.

Following are some Application layer protocols/commands which can generate UDP traffic :

- NTP (Network Time Protocol)
- BOOTP, DHCP.
- NNP (Network News Protocol)
- Trace Route
- Record Route
- Time stamp
- TFTP, RTSP, RIP.