

# CS-359 Assignment-11

Name: P. V. Sriram

Roll No.: 1801CS37

In this assignment, we experimented with various statistical values which related to the performance of Internet Protocols like Throughput, Round trip time, Packet Size, Number of packets lost, Number of TCP, UDP Packets, Number of responses with respect to requests etc.

For this purpose, we captured data packets from and to facebook.com at two different parts of the day. And on this data, we applied various functionalities of Wireshark to analyze performance.

## Capturing Packets

1. Firstly, we need the IP address of Facebook.com to apply capture filter.
2. We can do so by using the ping facebook.com command
3. We can also find out our own IP address using the ifconfig command
4. Facebook => 69.171.250.35
5. PC => 10.0.2.15
6. After we have this data, we can use capture filter as host 69.171.250.35 and then start capturing
7. Open a tab and go to facebook.com and perform actions until at least 2000 packets.

## Throughput

Throughput tells you how much data was transferred from a source at any given time and bandwidth tells you how much data could theoretically be transferred from a source at any given time.

We can measure average throughput and graphical plot of throughputs of various packets using Wireshark

For summary of transfers, got to “Statistics -> Capture File Properties”

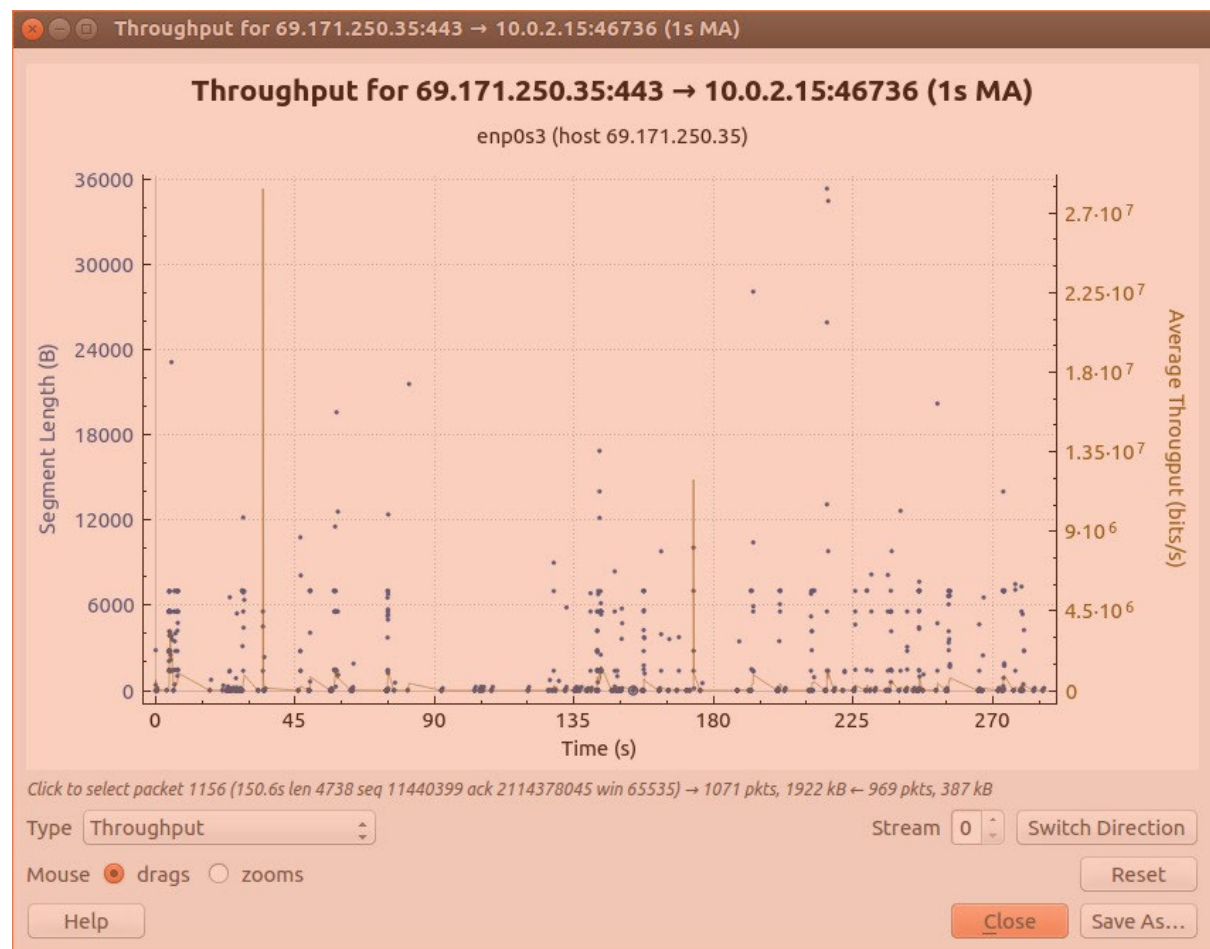
#### Statistics

Measurement	Captured	Displayed	Marked
Packets	2084	2084 (100.0%)	N/A
Time span, s	286.756	286.756	N/A
Average pps	7.3	7.3	N/A
Average packet size, B	1167.5	1167.5	N/A
Bytes	2433996	2433996 (100.0%)	0
Average bytes/s	8488	8488	N/A
Average bits/s	67 k	67 k	N/A

The above picture shows that Average rate of transfer in **bits/sec is 67k**. Which is average throughput

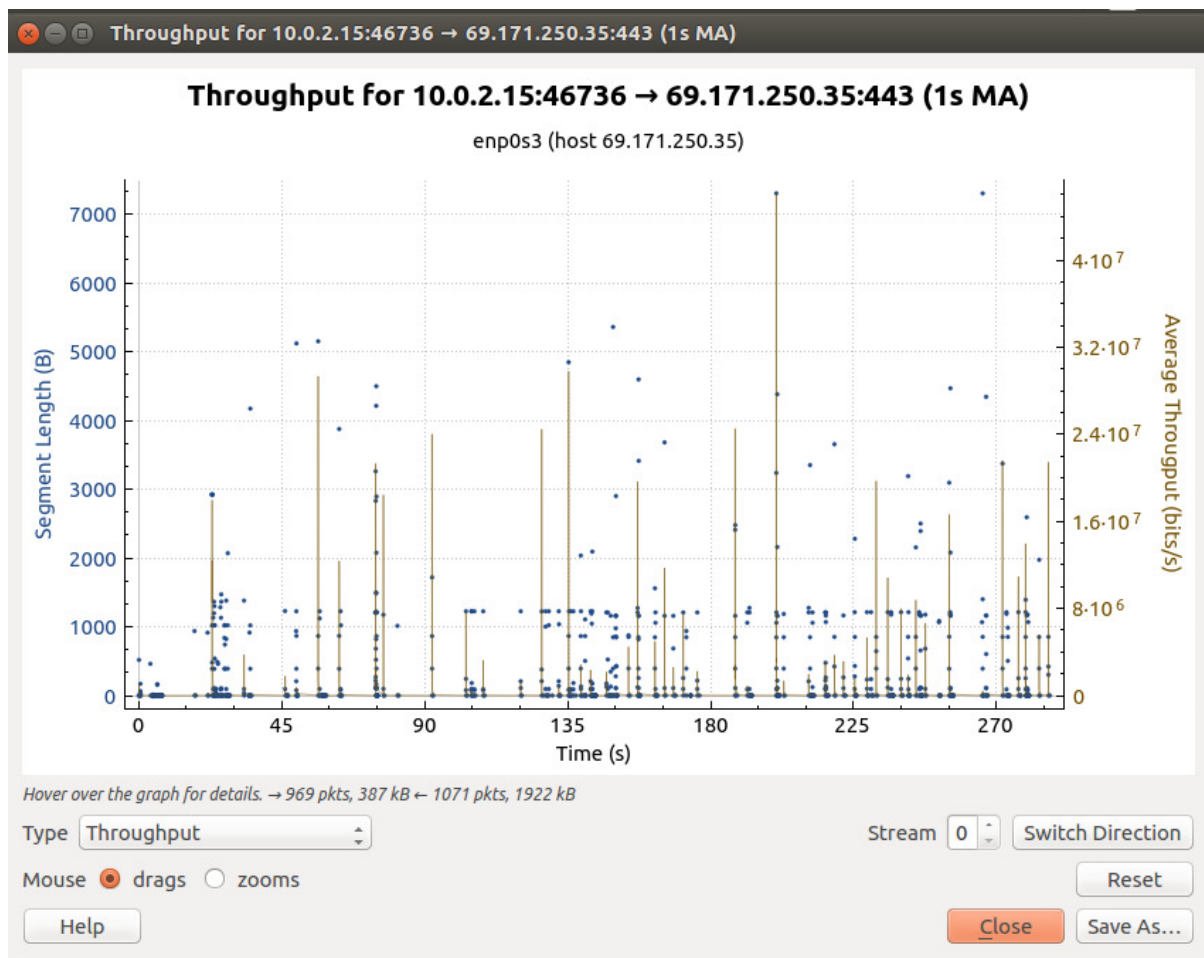
We can also view the distribution of Throughputs of various packets going coming from Facebook to the machine.

We can do this by “Statistics -> TCP Stream Graphs -> Throughput”



We can see in the above graph that there are a lot of spikes in speeds varying from  $10^6$  range to  $10^7$

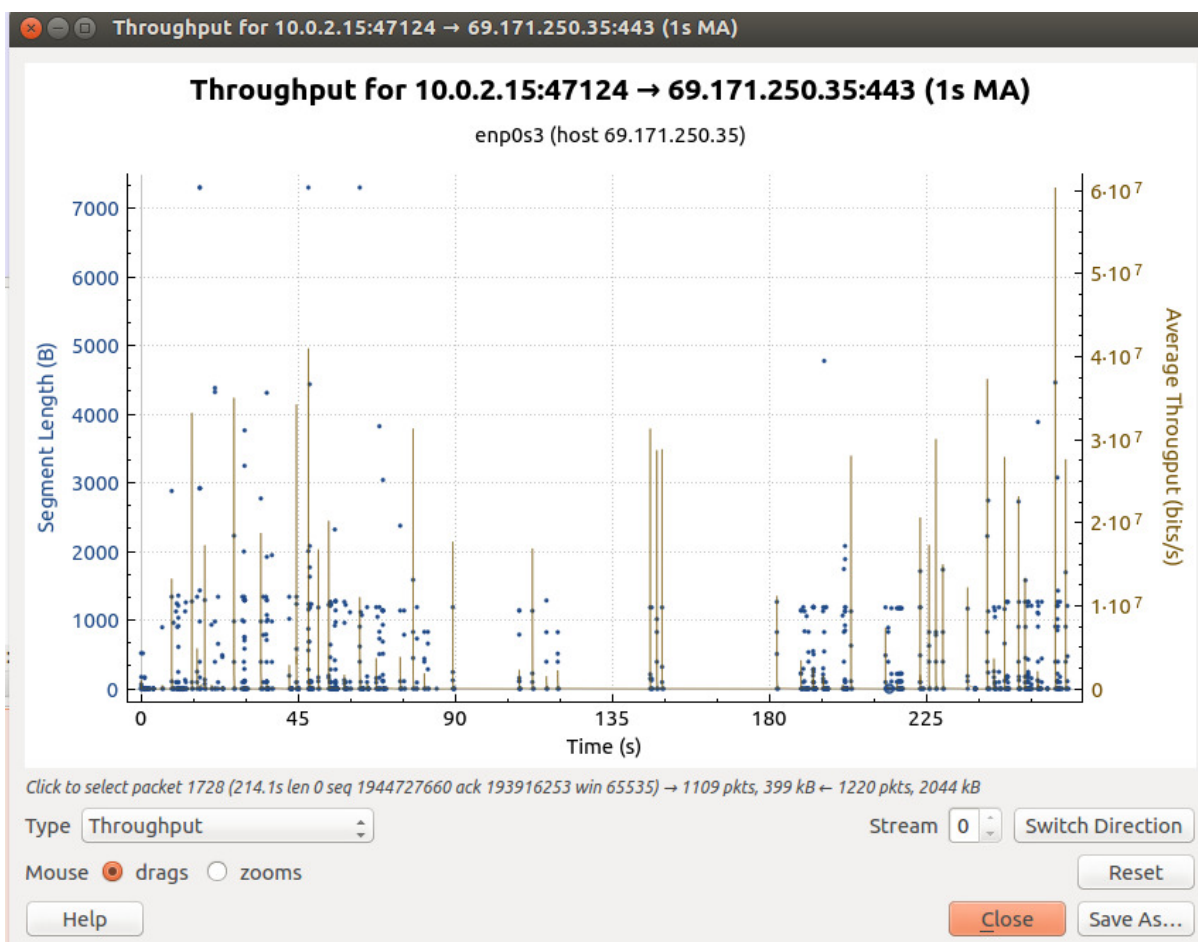
We can also view at packets going from machine to Facebook by switching directions

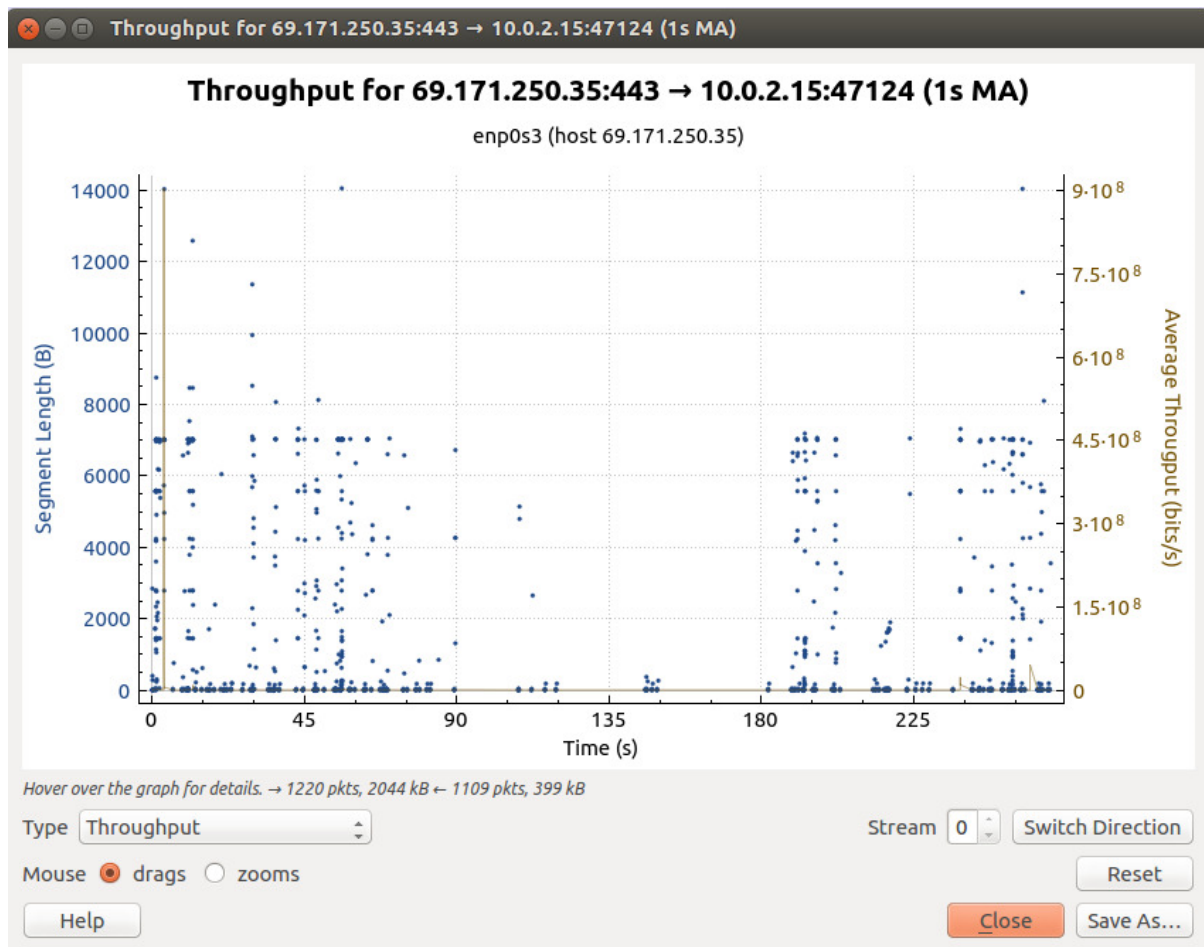


Similarly, we can repeat the experiment at another duration of the day. The following are the results.

## Statistics

Measurement	Captured	Displayed	Marked
Packets	2329	2329 (100.0%)	N/A
Time span, s	265.495	265.495	N/A
Average pps	8.8	8.8	N/A
Average packet size, B	1104.5	1104.5	N/A
Bytes	2572872	2572872 (100.0%)	0
Average bytes/s	9690	9690	N/A
Average bits/s	77 k	77 k	N/A





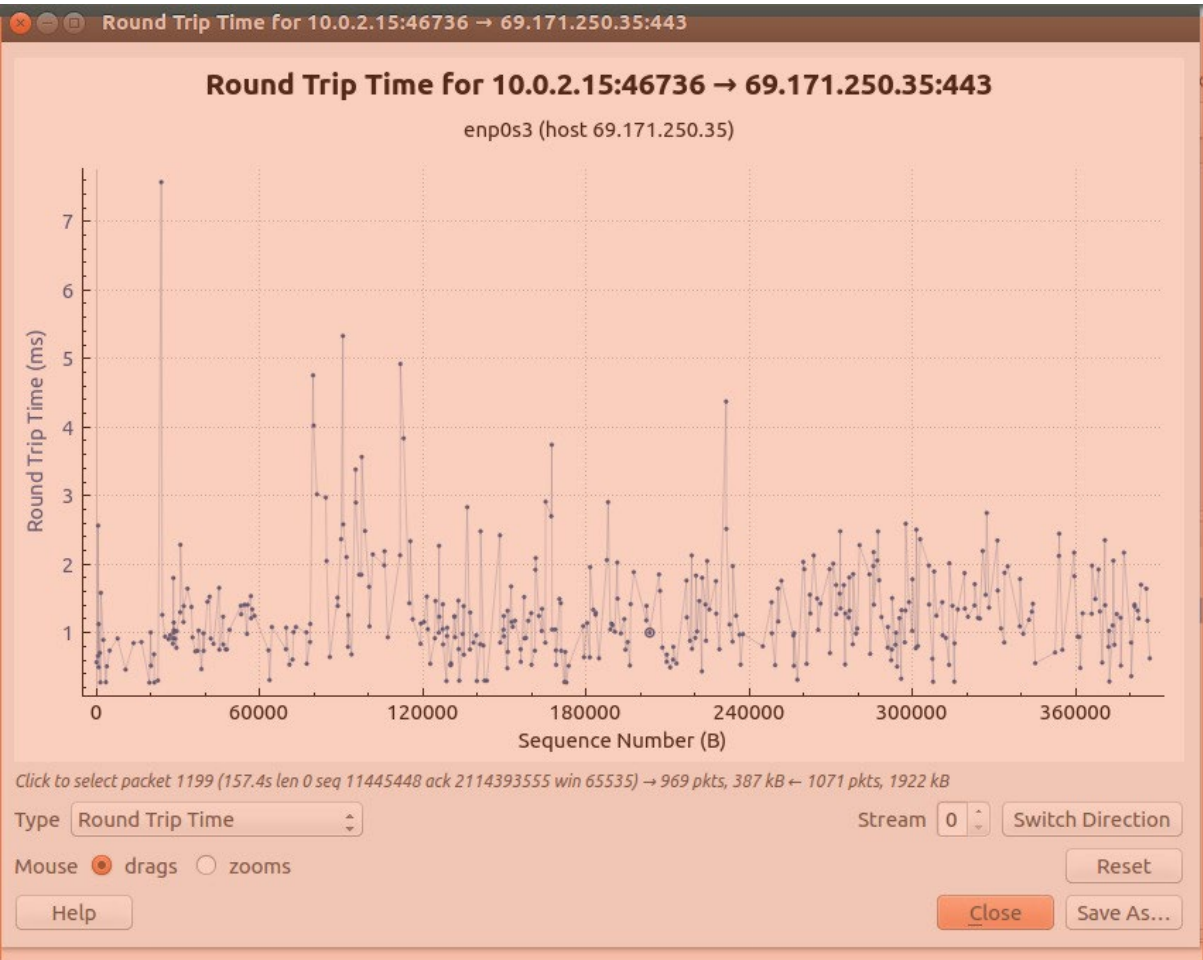
## Round Trip Time

Round Trip Time (RTT) is the length time it takes for a data packet to be sent to a destination plus the time it takes for an acknowledgment of that packet to be received back at the origin.

It is possible to plot the graphs similar to those of throughputs for to and fro packets in Wireshark.

We can do so by **“Statistics -> TCP Stream Graphs -> Round Trip Time”**

The following graph plots the RTT of packets going from machine to facebook. We can see they vary between 0.1ms to 7 ms with major chunk lying from 0 to 2.

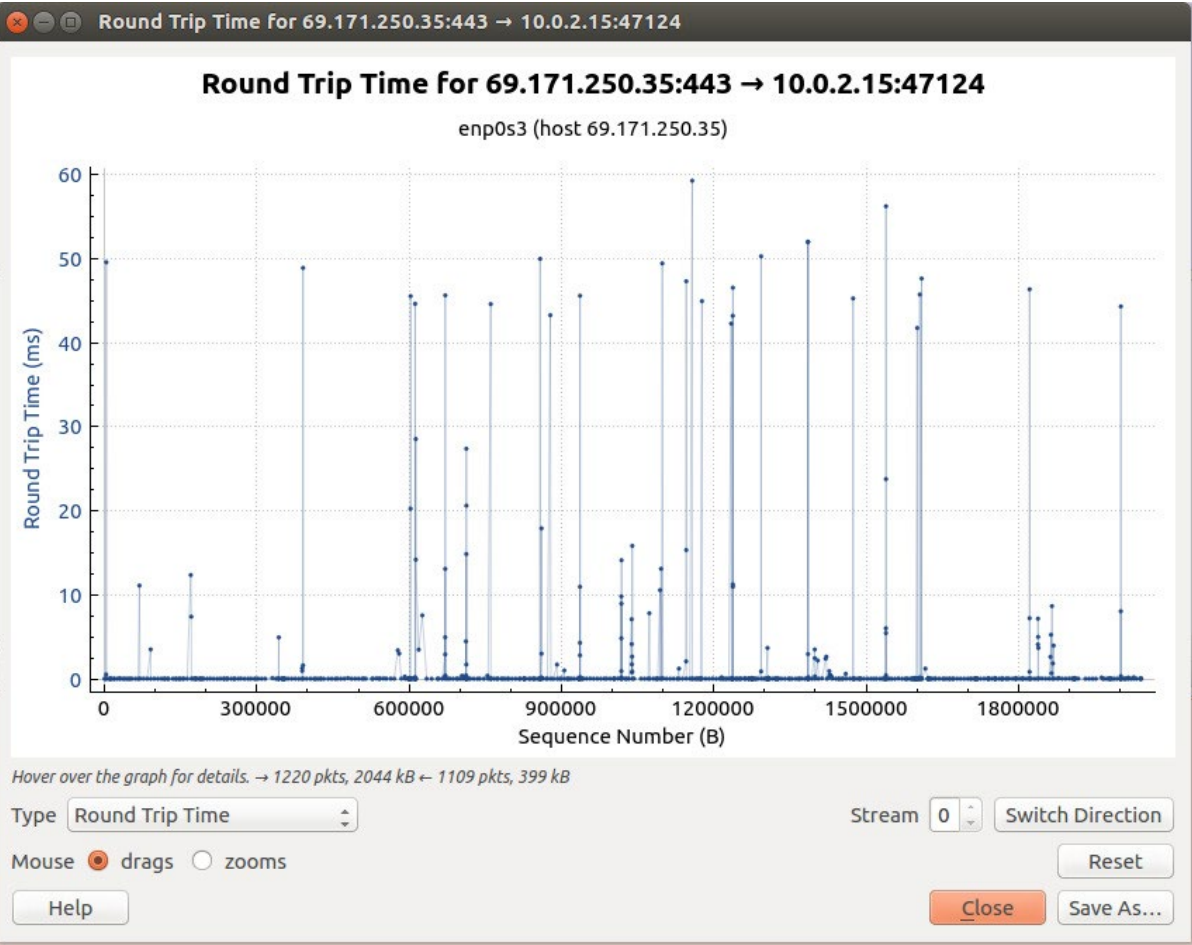


The packets coming from Facebook have very less RTT. Because of the high speed servers of Facebook and the acknowledgement is also sent immediately from our machine

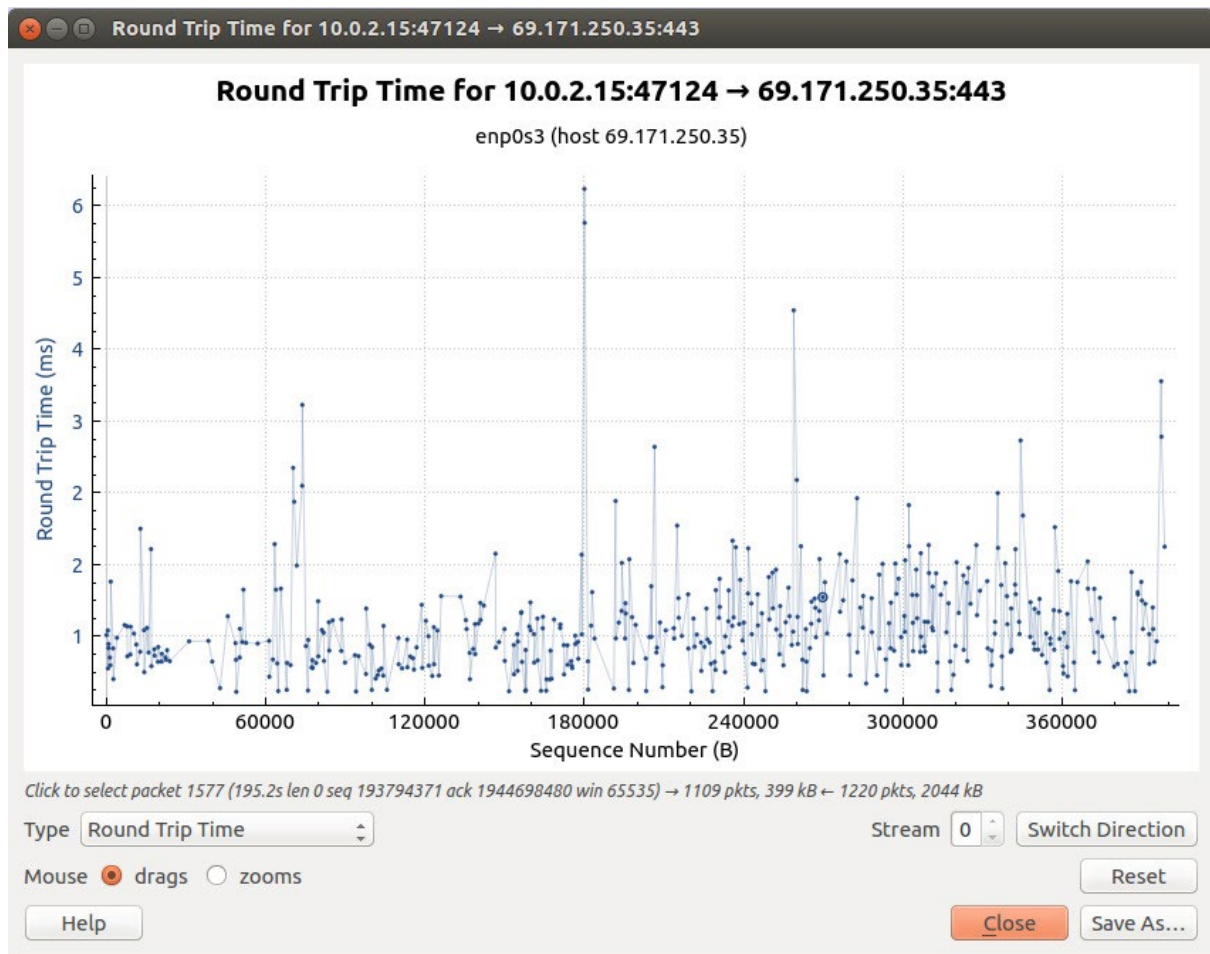


Similar experiment could be replicated at another time of the day by performing same steps.









## Packet Length

The summary of amount of data carried by packets can be displayed using **“Statistics -> Packet lengths”**

Following are the results for data collected at different times

Wireshark · Packet Lengths · wireshark\_enp0s3\_20210421082338\_WspMIS

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Packet Lengths	2084	1167.94	54	35423	0.0073	100%	0.7200	4.446
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	1083	56.89	54	78	0.0038	51.97%	0.3600	4.446
80-159	220	105.40	85	159	0.0008	10.56%	0.0600	24.287
160-319	114	228.48	160	317	0.0004	5.47%	0.0500	74.668
320-639	67	465.16	324	629	0.0002	3.21%	0.0300	3.678
640-1279	176	1103.94	646	1279	0.0006	8.45%	0.0700	74.715
1280-2559	119	1613.98	1282	2554	0.0004	5.71%	0.1100	143.695
2560-5119	105	3508.97	2573	5042	0.0004	5.04%	0.1800	4.439
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Display filter:  Apply

Copy Save as... Close

In the above data, we can see the average length of packet is 1167.94 bytes

Wireshark · Packet Lengths · wireshark\_enp0s3\_20210421100205\_5q1tFR

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Packet Lengths	2329	1104.71	54	14103	0.0088	100%	0.5700	29.546
0-19	0	-	-	-	0.0000	0.00%	-	-
20-39	0	-	-	-	0.0000	0.00%	-	-
40-79	1177	56.76	54	74	0.0044	50.54%	0.2900	29.558
80-159	245	106.13	85	159	0.0009	10.52%	0.1100	54.169
160-319	149	222.67	160	315	0.0006	6.40%	0.0600	29.582
320-639	79	467.23	320	636	0.0003	3.39%	0.0300	0.000
640-1279	197	1043.84	661	1279	0.0007	8.46%	0.0600	69.180
1280-2559	144	1647.12	1287	2532	0.0005	6.18%	0.0700	29.555
2560-5119	108	3697.33	2617	5038	0.0004	4.64%	0.0600	1.101
5120 and greater	0	-	-	-	0.0000	0.00%	-	-

Display filter:  Apply

Copy Save as... Close

In the above data, we can see the average length of packet is 1104.71 bytes

## Packets Lost

We can check the number of packets lost using the **“Statistics -> Capture File Properties”**

The summary shows many details including Dropped packets.

Luckily, due to the reliable Facebook servers, we get 0 dropped packets.

Interface	Dropped packets	Capture filter	Link type	Packet size limit
enp0s3	0 (0 %)	host 69.171.250.35	Ethernet	262144 bytes

## TCP Packets

Since Facebook Only uses TCP Protocol, there isn't much use of data from only Facebook. Therefore for the experiment of Capturing TCP Packets and UDP Packets, we remove the capture filter and collect general data.

We can use the Display filter **“tcp”** to view at only TCP Packets.

The image shows a Wireshark packet capture analysis. The top pane displays a list of captured packets, with the first 27 packets highlighted in green, indicating they are TCP. The second pane shows the details of the selected packet (No. 5), which is a TCP SYN packet from 10.0.2.15 to 34.107.221.82. The third pane shows the raw packet data in hexadecimal and ASCII format.

No.	Time	Source	Destination	Protocol	Length	Info
5	2021-04-21 11:43:10.4404741	10.0.2.15	34.107.221.82	TCP	74	49184 → 80 [SYN] Seq=9642834 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2960692 TSecr=0 WS=128
6	2021-04-21 11:43:10.4760219	34.107.221.82	10.0.2.15	TCP	60	80 → 49184 [ACK] Seq=533056001 Ack=9642835 Win=65535 Len=0 MSS=1460
7	2021-04-21 11:43:10.4760568	10.0.2.15	34.107.221.82	TCP	54	49184 → 80 [ACK] Seq=9642835 Ack=533056002 Win=29200 Len=0
8	2021-04-21 11:43:10.4767976	10.0.2.15	34.107.221.82	HTTP	348	GET /success.txt HTTP/1.1
9	2021-04-21 11:43:10.4767952	34.107.221.82	10.0.2.15	TCP	60	80 → 49184 [ACK] Seq=533056002 Ack=9643129 Win=65535 Len=0
10	2021-04-21 11:43:10.5117646	34.107.221.82	10.0.2.15	HTTP	274	HTTP/1.1 200 OK (text/plain)
11	2021-04-21 11:43:10.5117791	10.0.2.15	34.107.221.82	TCP	54	49184 → 80 [ACK] Seq=9643129 Ack=533056222 Win=30016 Len=0
16	2021-04-21 11:43:12.1307187	10.0.2.15	128.230.247.70	TCP	74	50080 → 80 [SYN] Seq=764247264 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2961117 TSecr=0 WS=128
17	2021-04-21 11:43:12.4353571	128.230.247.70	10.0.2.15	TCP	60	80 → 50080 [SYN, ACK] Seq=533312001 Ack=764247265 Win=65535 Len=0 MSS=1460
18	2021-04-21 11:43:12.4353840	10.0.2.15	128.230.247.70	TCP	54	50080 → 80 [ACK] Seq=764247265 Ack=533312002 Win=29200 Len=0
23	2021-04-21 11:43:12.7859698	10.0.2.15	34.120.5.221	TCP	74	50268 → 443 [SYN] Seq=1969373293 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2961278 TSecr=0 WS=128
24	2021-04-21 11:43:12.7861404	10.0.2.15	34.120.5.221	TCP	74	50268 → 443 [SYN] Seq=1101772324 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=2961278 TSecr=0 WS=128
25	2021-04-21 11:43:12.8056484	34.120.5.221	10.0.2.15	TCP	60	443 → 50268 [SYN, ACK] Seq=533440001 Ack=1969373294 Win=65535 Len=0 MSS=1460
26	2021-04-21 11:43:12.8056718	10.0.2.15	34.120.5.221	TCP	54	50268 → 443 [ACK] Seq=1969373294 Ack=533440002 Win=29200 Len=0
27	2021-04-21 11:43:12.8107753	10.0.2.15	34.196.5.221	TCP	571	Client Hello

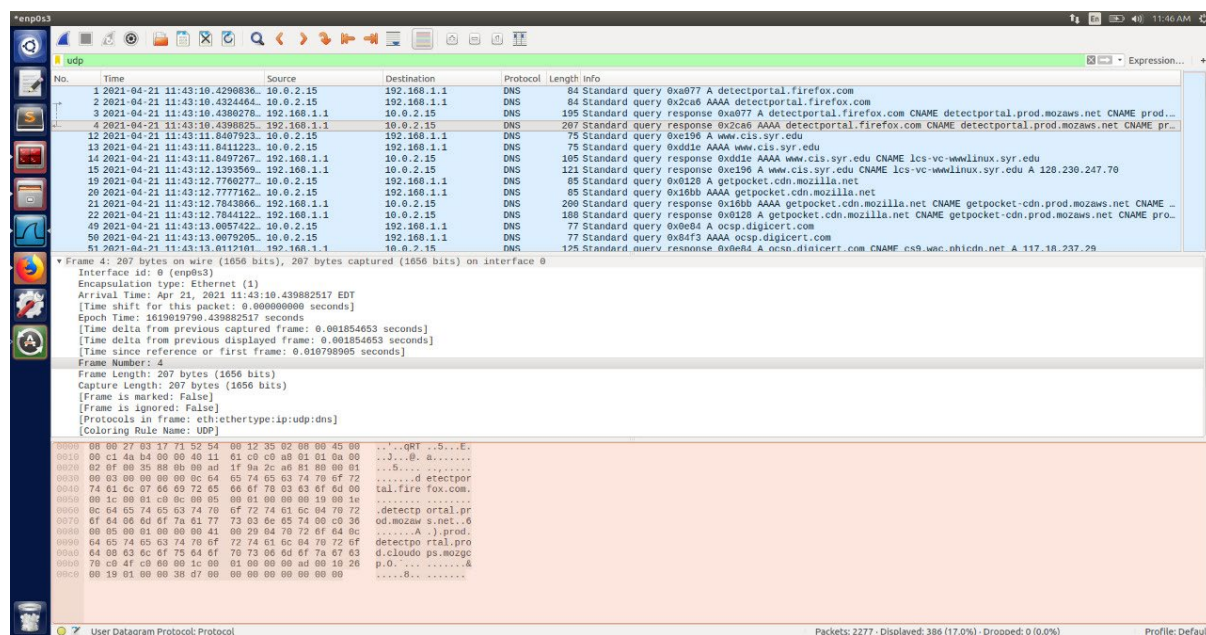
Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0  
Interface id: 0 (enp0s3)  
Encapsulation type: Ethernet (1)  
Arrival Time: Apr 21, 2021 11:43:10.440474161 EDT  
[Time shift for this packet: 0.000000000 seconds]  
Epoch Time: 1619619798.440474161 seconds  
[Time delta from previous captured frame: 0.000591644 seconds]  
[Time delta from previous displayed frame: 0.000000000 seconds]  
[Time since reference or first frame: 0.011390549 seconds]  
Frame Number: 5  
Frame Length: 74 bytes (592 bits)  
Capture Length: 74 bytes (592 bits)  
[Frame is marked: False]  
[Frame is ignored: False]  
[Protocols in frame: eth:ethertype:ip:tcp]  
[Coloring Rule Name: HTTP]  
0000 52 54 00 12 35 02 00 00 27 03 17 71 08 00 45 00 RT: S... ..q..E  
0010 00 3c 0e 00 00 00 00 20 0a 0a 00 02 0f 22 00 <.f.B. ....\*  
0020 dd 52 c0 20 00 50 00 93 23 52 00 00 00 00 a0 02 .R. .P. .HR.....  
0030 72 10 0b fb 00 00 02 04 05 04 04 02 08 0a 00 2d f.....  
0040 2d 34 00 00 00 00 93 03 03 07 ..d.....

We can see that 1891 packets out of 2277 captured packets are TCP protocol.

i.e. 83% of packets follow TCP protocol.

## UDP Packets

Similarly, we can capture the UDP Packets using the display filter “udp”



We can see that 386 packets (17% of packets) out of 2277 packets follow UDP Protocol.

## Request-Response

As a part of this experiment, we are also required to capture Request Response pairs. i.e. How many Responses would be there for one particular Request.

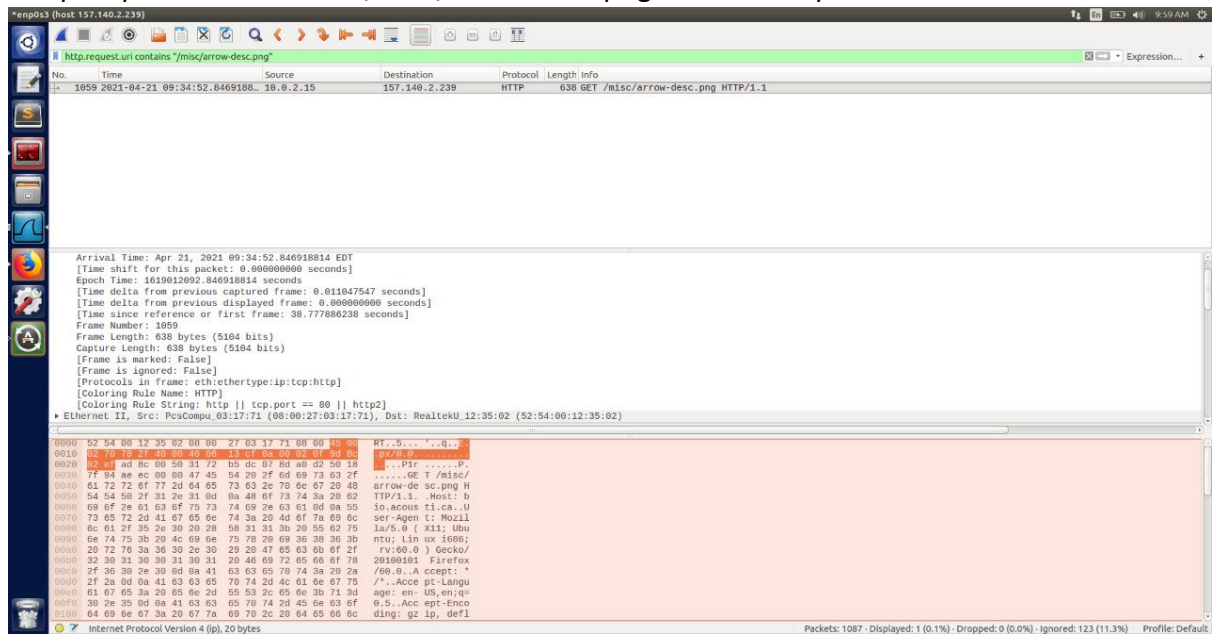
Therefore, we need HTTP GET data. Unfortunately, Facebook follows HTTPS protocol, and therefore we need to change our source for this experiment. A website which uses HTTP is essential.

For this experiment, we use a site bio.acousti.ca. IP address is 157.140.2.239. We can capture its data using “host 157.140.2.239” capture filter.

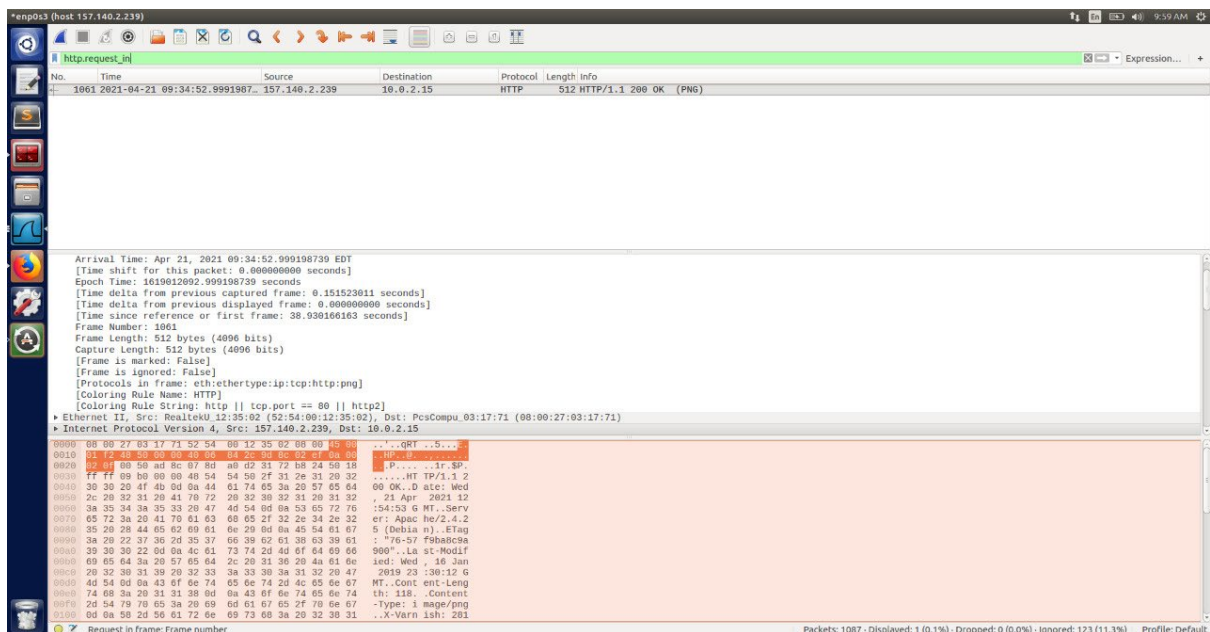


Now that we have the data, we need to perform the following steps to capture its response packets

- 1) From the data, display the request packets using **“http.request”** display filter
- 2) From the resulting list, choose one random request and display it. I have used **“http.request.uri contains \"/misc/arrow-desc.png”** for this experiment.



- 3) Next step involves removing all the other requests and their responses. This could be done using **“http.request && !http.request.uri contains \"/misc/arrow-desc.png”** display filter and remove that list using **“Edit -> Ignore all displayed”**
- 4) Now there is only the response for the request we are considering. We can display the response using **“http.request\_in”** display filter



We can now see the singular response for our request.