1. Network configuration

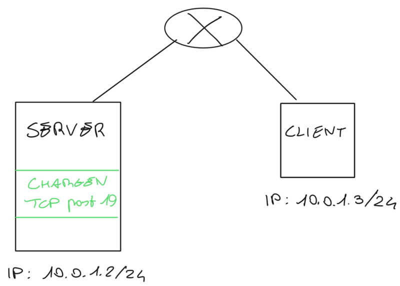


Figure 1. Simplified network configuration

2. Performed steps

Following the instruction we connected two devices using the provided switch, one as a client (with IP address 10.0.1.3/24) and the other one acting as a server (IP 10.0.1.2/24); the server machine had the Chargen service active on TCP port 19. We started Chargen request from client terminal using telnet command and started the capture on Wireshark; during the capture we performed the subsequent steps, roughly every 5 seconds of analysis:

- Maximise the terminal window: initial condition, transmission rate in less than the maximum possible one due to workload on the client side
- Minimize the terminal window: the transmission rate increases due to less workload on the client
- Press CTRL-C on terminal: the transmission rate reaches maximum thanks to no workload on the client
- Enter telnet command mode: no data exchanged
- Press return on command mode: return to transmission
- Enter again command mode
- Close the connection

3. Plots and comments

3.1. Sequence number

Every sequence number reported here is a relative one, starting from 0, i.e. the relative sequence number in Wireshark.

3.1.1 Client

Figure: 2
Since the client didn't send any data to server after SYN packet, for the first 15 seconds the sequence numbers remain constant; after pressing CTRL-C we sent to the server a packet with a 5 bytes payload, the sequence number increased to 6. The capture finished to sequence number 8 due to the other commands sent and the FIN packet.

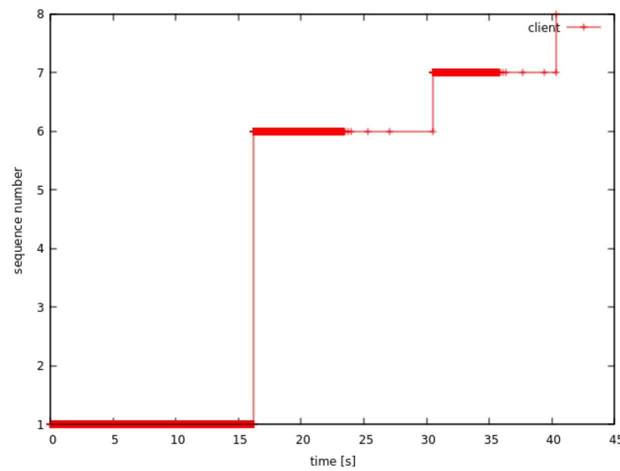


Figure 2. Client sequence number evolution

3.1.2 Server

Figure: 3
On the other side, server sent lots of data to the client and due to this the sequence number was increasing very fast

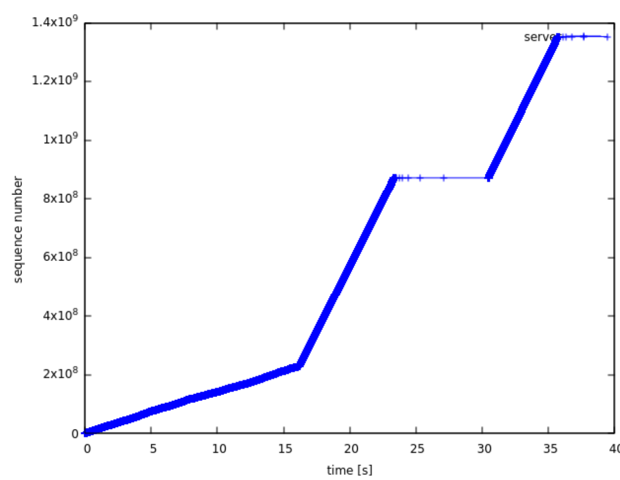
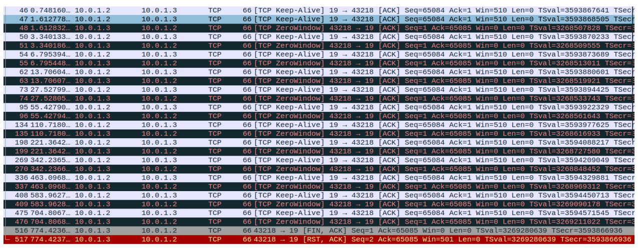


Figure 3. Server sequence number evolution

3.1.3 Stopping the server

Figure: 4
When the client hits CTRL+], the server immediately receives a TCP segment announcing that the window size is 0. In TCP, ”stop” is coded with a packet that reports the value 0 in the Window Size field. At this point, the connection is suspended. On the other hand, the server schedules the sending of messages at an increasingly greater distance (exponential back-off), called Keep Alive. These messages solicit the client to respond by providing, among other things, the current window size. If this is greater than zero, the server will be able to continue sending data.



46	0.748160…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3593867641 TSec
47	1.612778…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3593868505 TSec
48	1.612832…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268507828 TSecr=3
50	3.340133…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3593870233 TSec
51	3.340186…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268509555 TSecr=3
54	6.795394…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3593873689 TSec
55	6.795448…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268513011 TSecr=3
62	13.70604…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3593880601 TSec
63	13.70607…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268519921 TSecr=3
73	27.52799…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3593894425 TSec
74	27.52805…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268533743 TSecr=3
95	55.42790…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3593922329 TSec
96	55.42794…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268561643 TSecr=3
134	110.7180…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3593977625 TSec
135	110.7180…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268616933 TSecr=3
198	221.3642…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3594088217 TSec
199	221.3642…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268727580 TSecr=3
269	342.2365…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3594209049 TSecr
270	342.2366…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268848452 TSecr=3
336	463.0968…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3594329881 TSecr
337	463.0968…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3268969312 TSecr=3
408	583.9627…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3594450713 TSecr
409	583.9628…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3269090178 TSecr=3
475	704.8067…	10.0.1.2	10.0.1.3	TCP	66	[TCP Keep-Alive] 19 → 43218 [ACK] Seq=65084 Ack=1 Win=510 Len=0 TSval=3594571545 TSecr
476	704.8068…	10.0.1.3	10.0.1.2	TCP	66	[TCP ZeroWindow] 43218 → 19 [ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3269211022 TSecr=3
516	774.4236…	10.0.1.3	10.0.1.2	TCP	66	43218 → 19 [FIN, ACK] Seq=1 Ack=65085 Win=0 Len=0 TSval=3269280639 TSecr=3593866936
517	774.4237…	10.0.1.3	10.0.1.2	TCP	66	43218 → 19 [RST, ACK] Seq=2 Ack=65085 Win=501 Len=0 TSval=3269280639 TSecr=3593866936

Figure 4. Back-off

Note that: for the client’s TCP module to respond, the simplest mechanism for the server is to send it TCP messages with already used sequence numbers.

3.2. ACK number

Every acknowledgment number reported here is a relative one, starting from 0, i.e. the relative acknowledgment number in Wireshark.

3.2.1 Client

Figure: 5
As we expected the Client ack numbers evolution is the same of the Server sequence numbers one; that’s because for definition of TCP a client send the ack with the sequence number of next expected byte.

3.2.2 Server

Figure: 6
For the same reason this plot is associated to the Client sequence numbers evolution

3.3. Packet size

Packet size The size of the packets sent by the two TCP devices differ significantly since only one side of the connection is genuinely loaded with data. In actuality, the server (**Figure** 9) sends packets that range in size from 1514 bytes to little over 64 bytes, with an average of 68 bytes

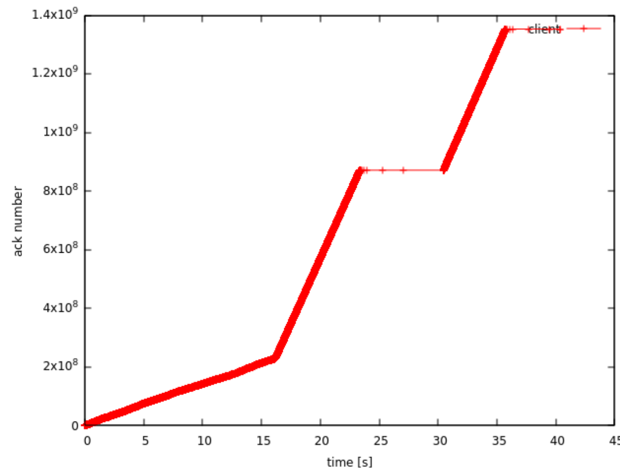


Figure 5. Client ack number evolution

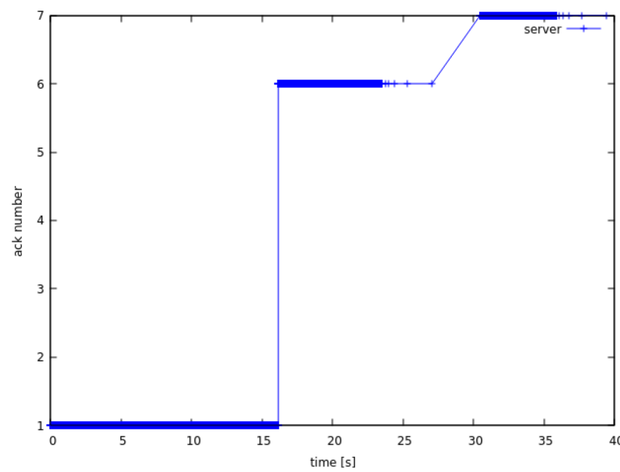


Figure 6. Server ack number evolution

for those delivered that are solely acknowledgment message by the client. Notice that the experiment was carried out without denying the TCP any optimizations (such the ability to do a selective acknowledgment, or SACK), and in fact, the client (**Figure** 8) transmits a lot of TCP packets with precisely selected acknowledgments in the options field (**Figure** 7); therefore, the graph shows the presence of messages that are larger than a traditional TCP acknowledgment packet.

3.4. Window size

3.4.1 Client

Figure: 10
The client reception window exhibits very fluctuating behavior during the period of time in which the data generated by Chargen is displayed on the screen, i.e first 15 seconds, with multiple instances in which the value drops to zero,

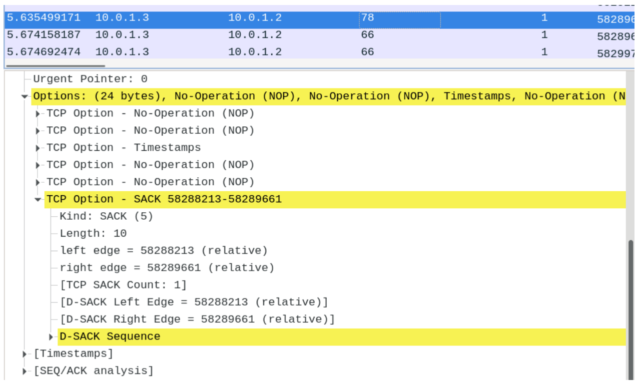


Figure 7. TCP options dissection with Wireshark

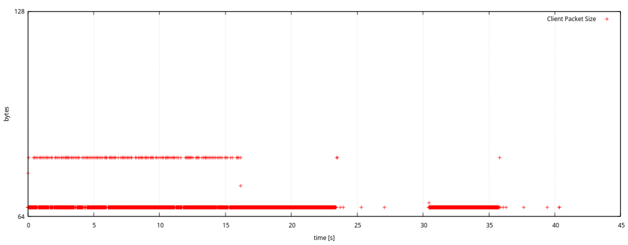


Figure 8. Client bytes sent

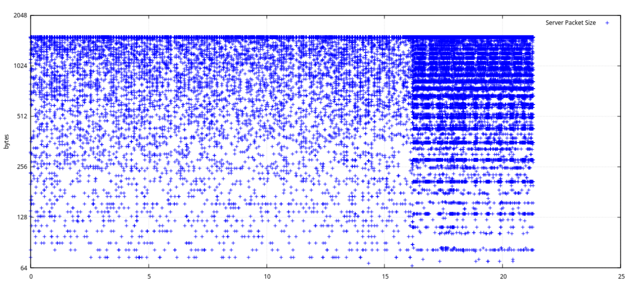


Figure 9. Server bytes sent

i.e. encoded as TCP ZeroWindow by Wireshark. In contrast, in time windows the output is suppressed, i.e. between 15 and 25 seconds as well as between 30 and 35 seconds, the receiver window remains mostly constant over the entire timespan.

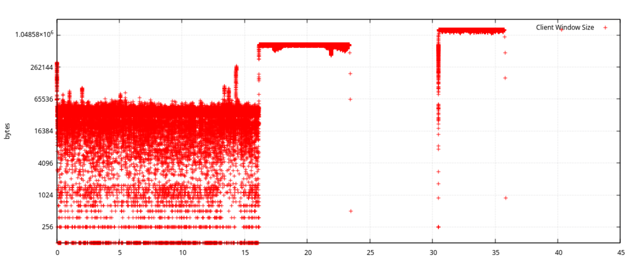


Figure 10. Client receiver window evolution

3.4.2 Client with Zero Window Size

In a side experiment, we investigate the Zero Window Size, (**Figure** 11): we ask Wireshark to calculate the quantity of bytes sent by the server that have not yet been acknowledged, i.e. the difference between the last sequence number sent by the server and the last acknowledgment received from the client, or bytes on flight. You can clearly see that the TCP ZeroWindow event occurs as soon as the amount of bytes that have been sent to the network by the server corresponds precisely to the last reception window size announced by the client. If the client is still connected, it sends a TCP message, i.e. coded TCP Window Update by Wireshark, which tells the server the new client's window size, i.e. **Figure** 11. After that, the sender starts transmitting its buffered data and traffic should flow normally.

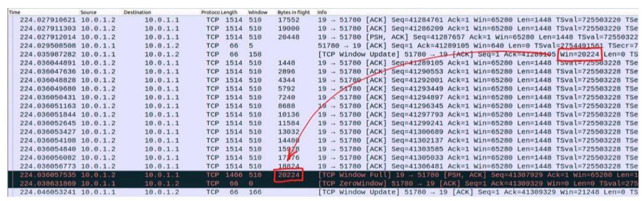


Figure 11. Client's buffer gets full but it is still connected

Conversely, if the client gets disconnected, the overall connection enter a kind of stall: the client wants more data, but the server had to stop. To exit the stall, the server must be notified with the new receiver window, but since TCP is designed to receive responses only in response to requests, the server is forced to prompt the client in some way. And in fact, the server starts sending TCP packets with sequence numbers already previously sent: in particular, if the last acknowledgment number received is X, the server will prompt the client with a sequence number X - 1, i.e. **Figure** 12.

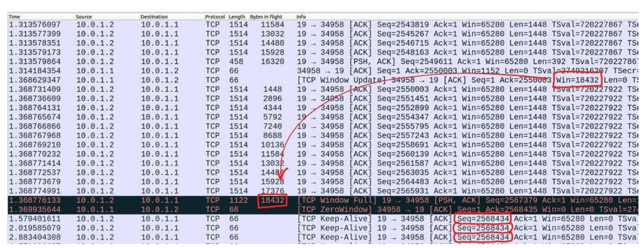


Figure 12. Client's buffer gets full and disconnected

The server sends solicitation messages with ever-increasing temporal spacing and we have already seen the back-off technique. Since the client and the server share the same network, as soon as the server forgets the MAC address of the client, each TCP message sent by the server gets an ICMP error message, as depicted in **Figure** 13. After a long-term awaiting and attempts, the connection is closed at server side. Therefore, if the client connects the server, the server resets the connection.

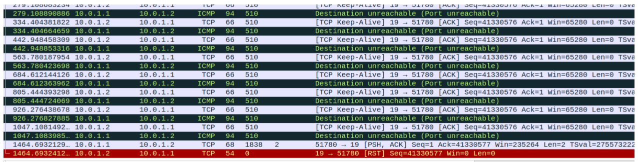


Figure 13. ICMP Port Unreachable

3.4.3 Server

Figure: 14
Since there is not a substantial transfer occurring on the client end of the experiment, the server reception window stays almost constant.

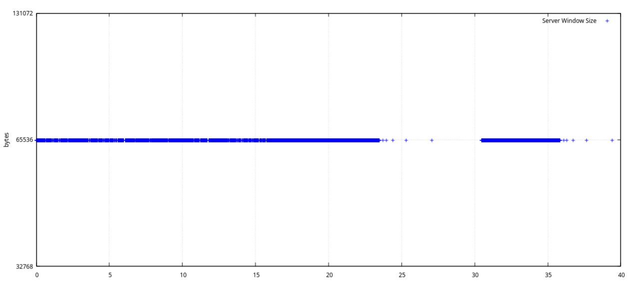


Figure 14. Server receiver window evolution

4. Two connections analysis

In order to provide further analysis we exploited another test in which a new TCP connection to port 19 from the client machine to the server one is opened. During our new test after some seconds of normal analysis we started another connection from the same host using a second terminal and we provided the following operation, every 5 seconds starting from second 20 of timeline:

- start a new TCP connection
- press CTRL-C on first terminal
- press CTRL-C on second terminal
- enter Telnet command mode on first terminal
- enter Telent command mode on second terminal
- close connections

As we expected the second parallel connection had an impact on the transmission speed rate of the first one, as it will be explained in the next sections. Other plots are omitted due to the strong similiarity with the previous ones.

4.1. Differences in output handling

Figure 15 shows us how the terminal size and the output printing in it have a massive effects on the transmission rate; we can see in the time axis how the behaviour is after the output is suppressed on first terminal (CTRL-C pressed, second 25) and the big difference compared with the other terminal, where the output is still printed.

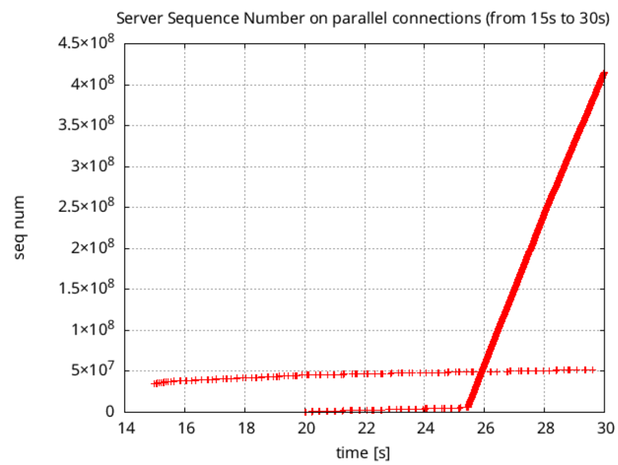


Figure 15. Sequence number evolution of the two connections

4.2. Concurrency handling

In **Figure** 16 is shown how the TCP connection is managed in order to maximise the possible speed rate (shown as the inclination of the Sequence number evolution curve). In the first part of the graph both connection are configured to sending data at the maximum possible speed rate (CTRL-C pressed in both terminals), so in that moment a single transmission had the half possible speed rate, assuming ideal conditions in our small network. Reached the 35 seconds time the first terminal entered in Telnet configuration mode, so the transmission was stopped in the first TCP connection; as we can notice the inclination of the second curve increases immediately because the second connection reached the maximum transmission rate, having the channel totally free.

5. Appendix: useful bash commands

5.1. Extracting data from Wireshark

After we obtained the csv files (**Figure** 17) with the complete capture we run some bash commands in order to get only the data needed to plot the different study cases; that's because gnuplot needs a cleaner file with only the useful columns. In every file we obtained in that manner we had the first column representing time in seconds (x axis) and

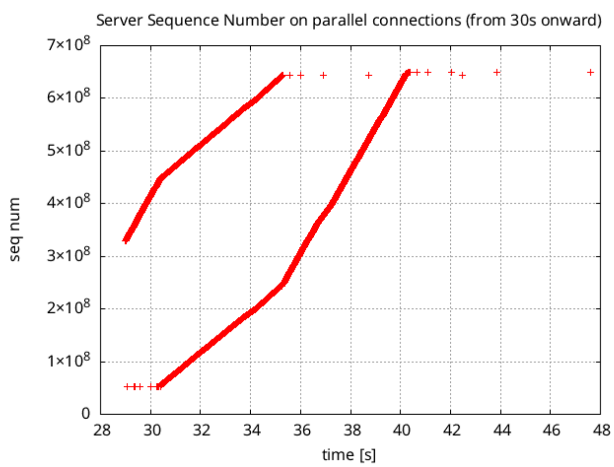


Figure 16. Evolution in terms of channel occupation

Time	Source	Destination	Protocol	Length	Sequence number	Acknowledgment number
0.00000000	10.0.1.3	10.0.1.2	TCP	74	0	0
0.000524988	10.0.1.3	10.0.1.2	TCP	66	1	1
0.001865429	10.0.1.3	10.0.1.2	TCP	66	1	75
0.001874324	10.0.1.3	10.0.1.2	TCP	66	1	1523
0.002004159	10.0.1.3	10.0.1.2	TCP	66	1	2971

Figure 17. Example of csv file exported from Wireshark

the other ones the data we wanted to represent in our plot (y axis)

```
# Run this for extracting Time and Sequence Number fields
cat file.csv | tail -n +2 | cut -d "," -f1,6 | tr -d '"' | tr ',' '\t' > output_seq.txt
# Run this for extracting Time and Acknowledgment Number fields
cat file.csv | tail -n +2 | cut -d "," -f1,7 | tr -d '"' | tr ',' '\t' > output_ack.txt
```

Figure 18. How to extract Sequence and ACK numbers from a csv

5.2. Plotting data on gnuplot

Using the txt files we obtained via the previous bash commands (**Figure** 18) it was much easier to create the graphs for this report. A sample of an our plot creation is provided in **Figure** 19 (the result is shown in **Figure** 3)

```
gnuplot > set xlabel "time (s)"
gnuplot > set ylabel "Sequence number"
gnuplot > plot "output_seq.txt" using 1:2 title "Server sequence number" with linespoint
```

Figure 19. How to create Sequence number graph with gnuplot