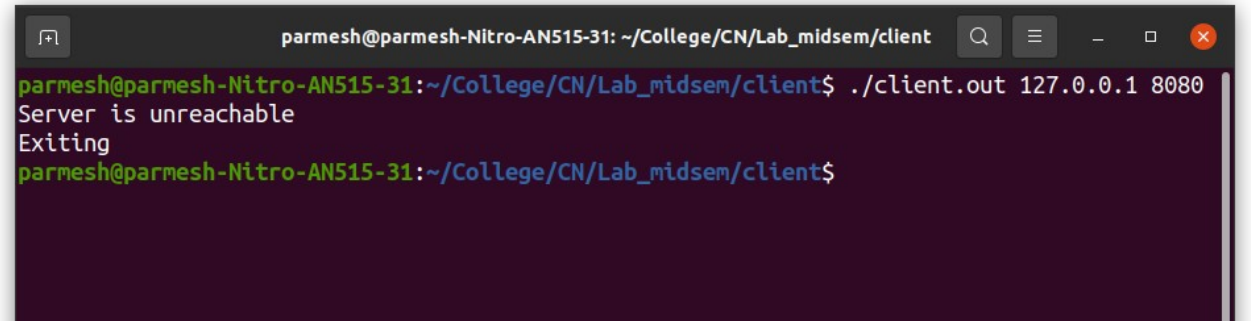
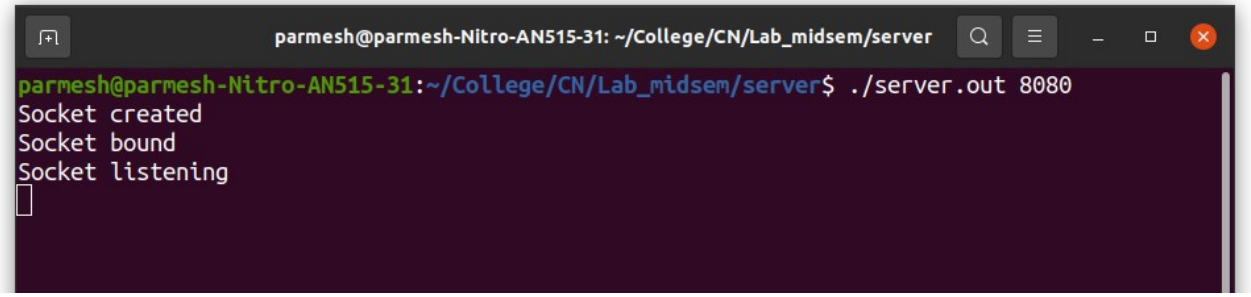
Lab Midsem

Parmesh Mathur **2018A7PS0133G**

When the server is not running, the client does not connect to it and exits immediately.

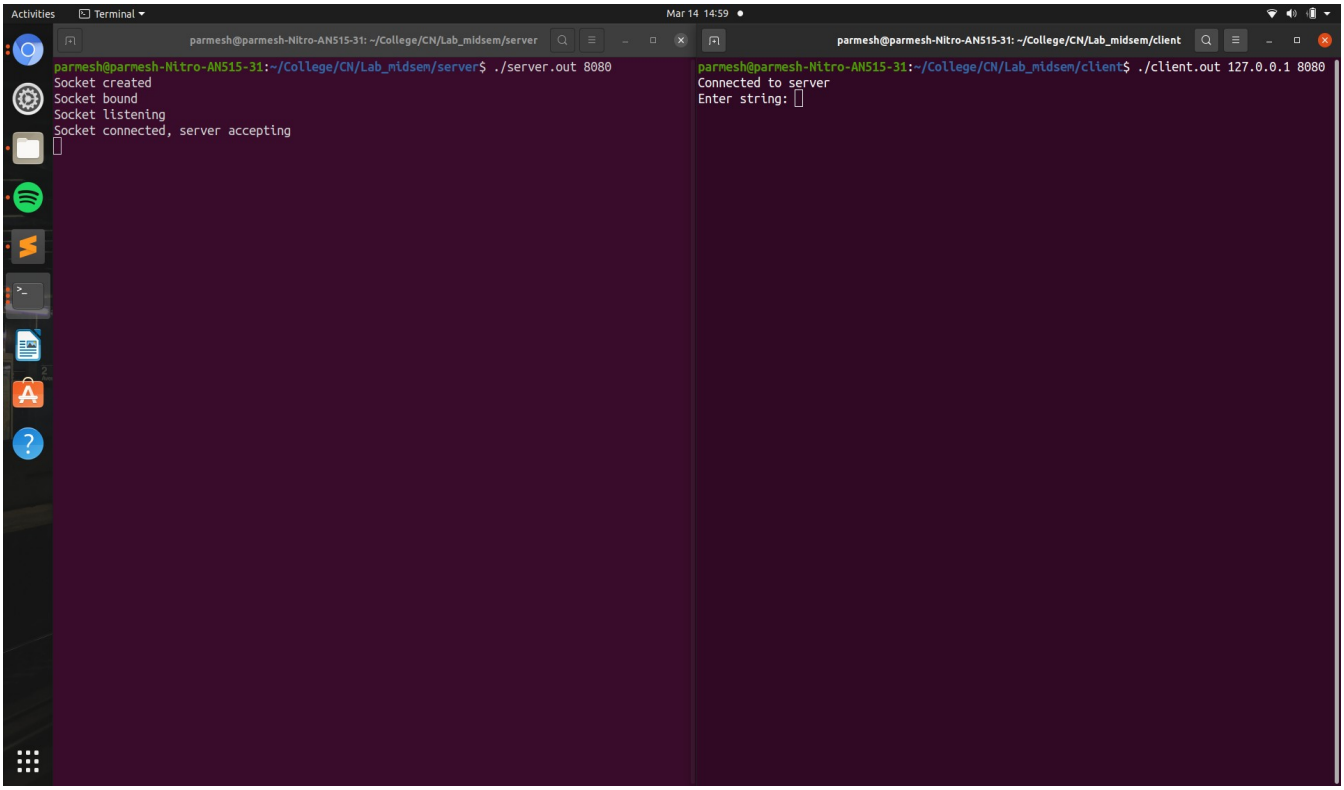


The server will create a socket and listens to it when it is executed. It is then ready to communicate with an incoming client.



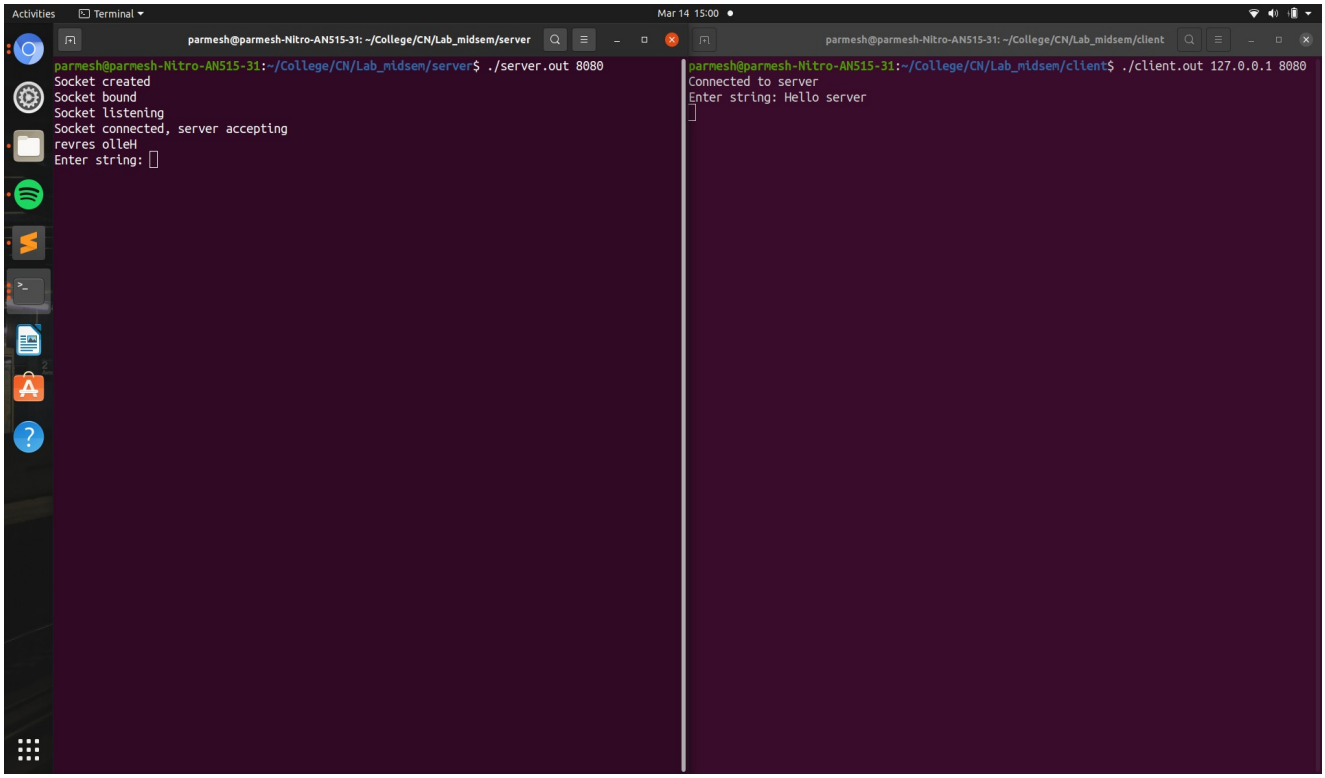
P.T.O

When the client is executed while the server is running, the socket is connected and it is ready to receive a string from the user (stdin).



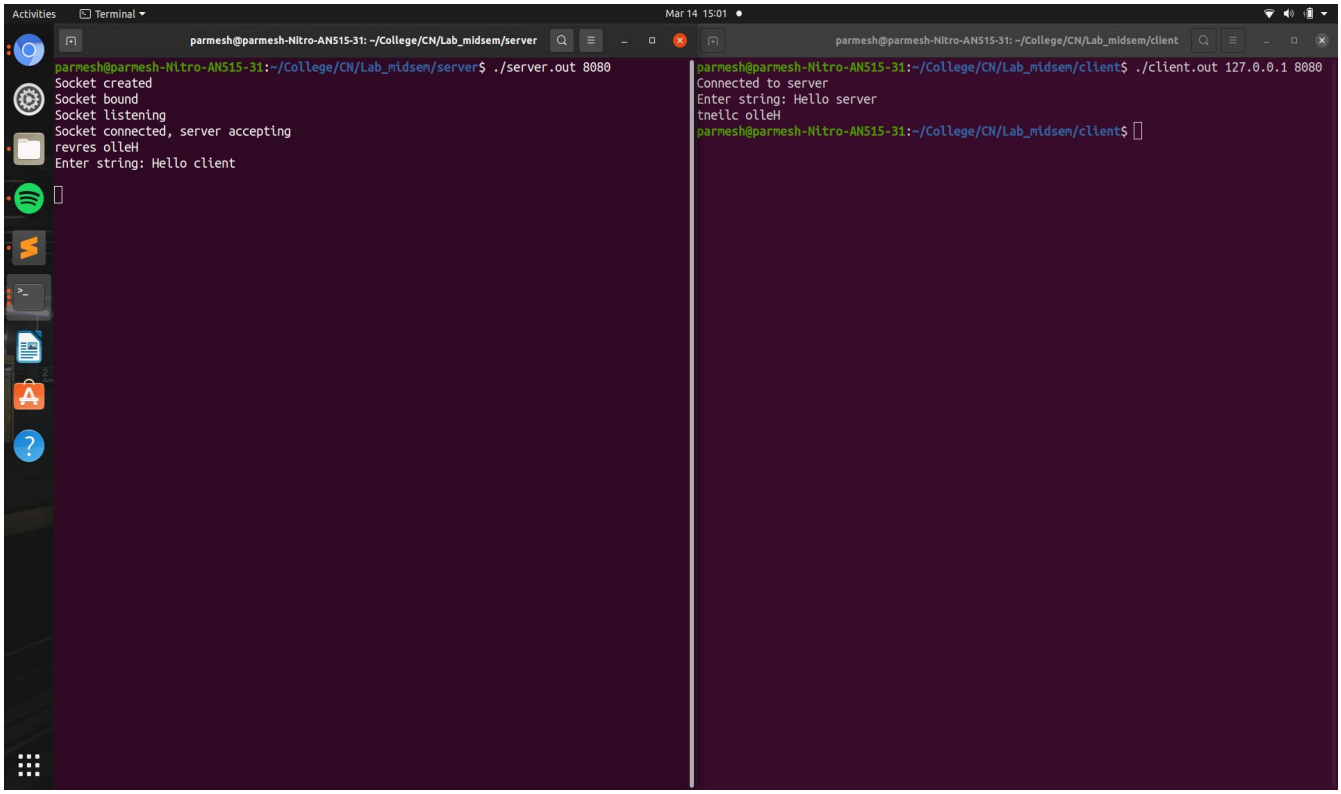
P.T.O.

Once the string is sent to the server, it displays it in reverse order to the user (stdout). The server is now ready to receive its string input from stdin.



P.T.O.

Once the server receives the string from the user, it is sent to the client. The client then reverses the string to display it and then exits.

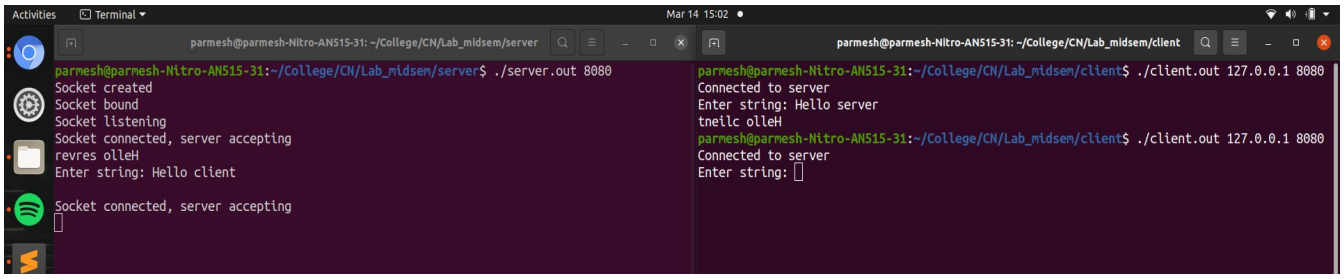


Meanwhile, the server is still running and is now ready to accept another client process.

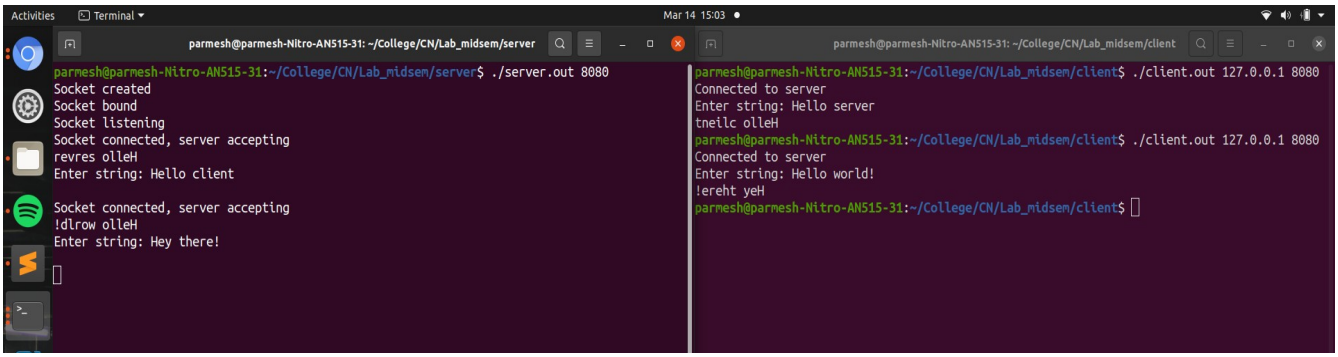
P.T.O.

When another client comes in consecutively, the whole process of sending and receiving messages repeats untill the client exits again.

The client connects to the running server process.

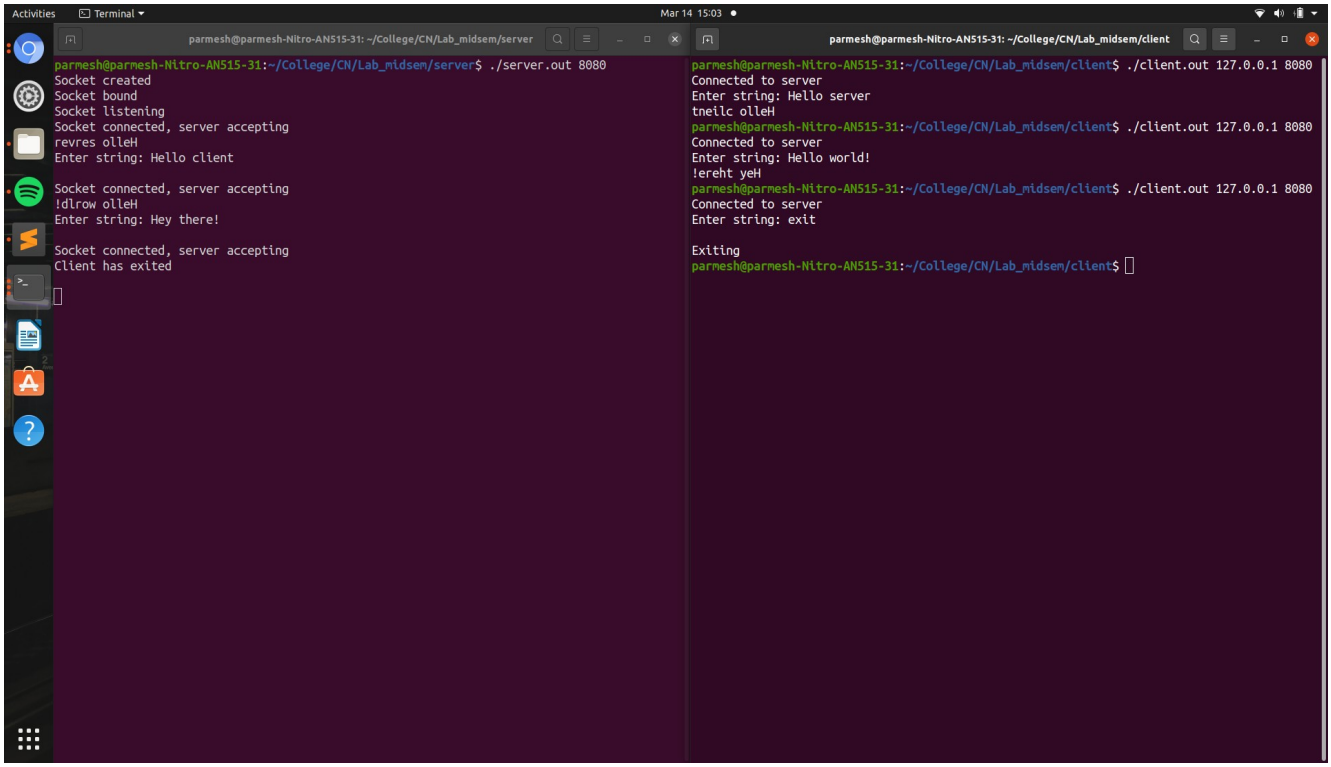


The client and server exchange messages, and then the client exits again.



The server is again ready to accept another client.

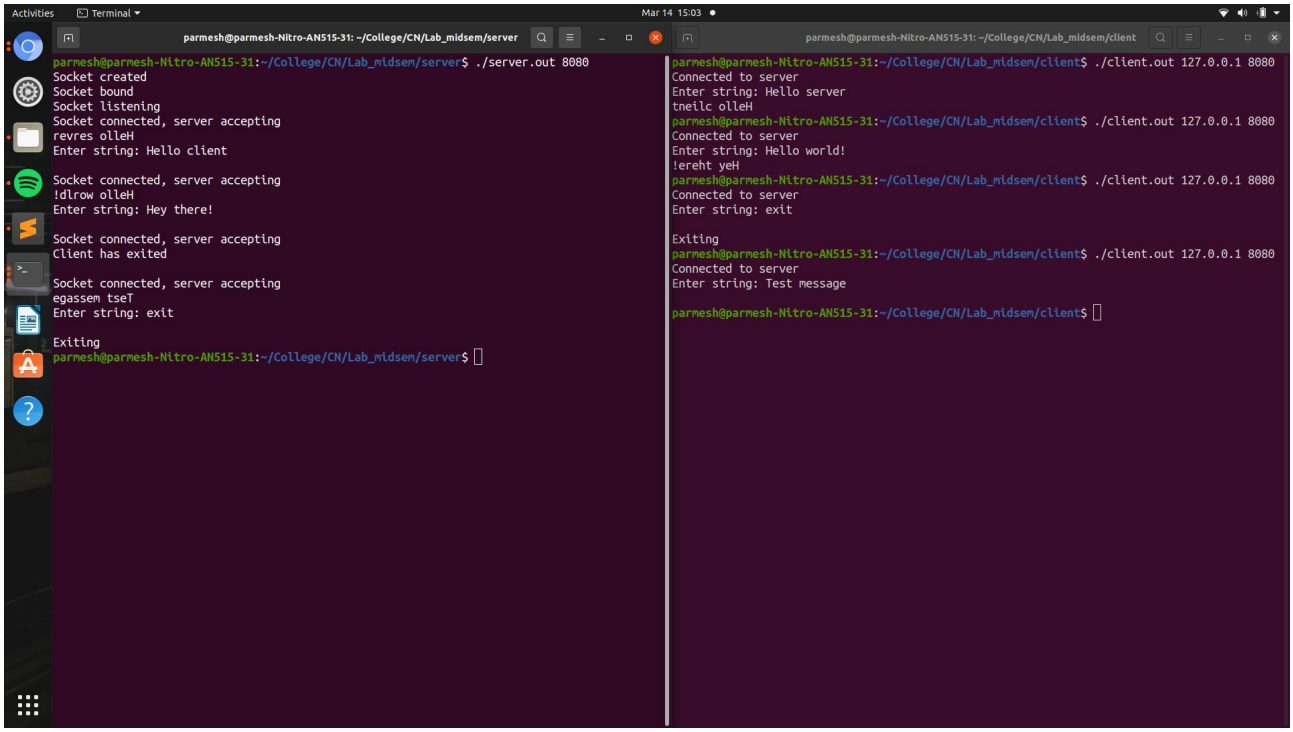
After the client connects, if the user enters the exit string (“exit”), the client will exit immediately.



The server continues running waiting for the next client.

P.T.O.

When it is the server's turn to send the message, if the user inputs the exit string, the server exits immediately. Consequently the client also exits as it can't run without a server.



P.T.O.

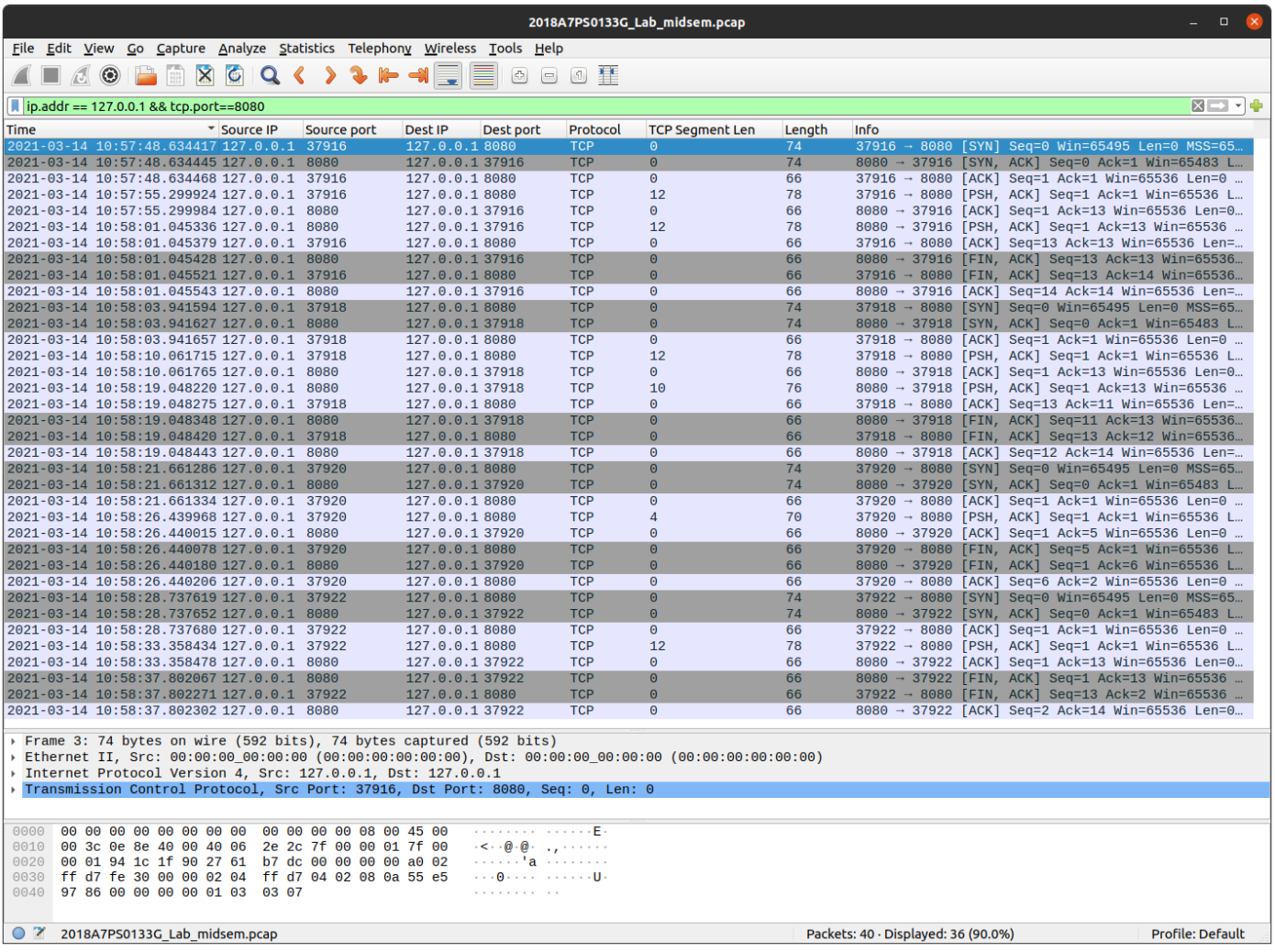
Using Wireshark

All the packets between the two processes were captured on Wireshark on the

As the file might contain some non-tcp packets, the filter used to obtain only the packets transactions between the two processes was:

```
ip.addr==127.0.0.1 && tcp.port==8080
```

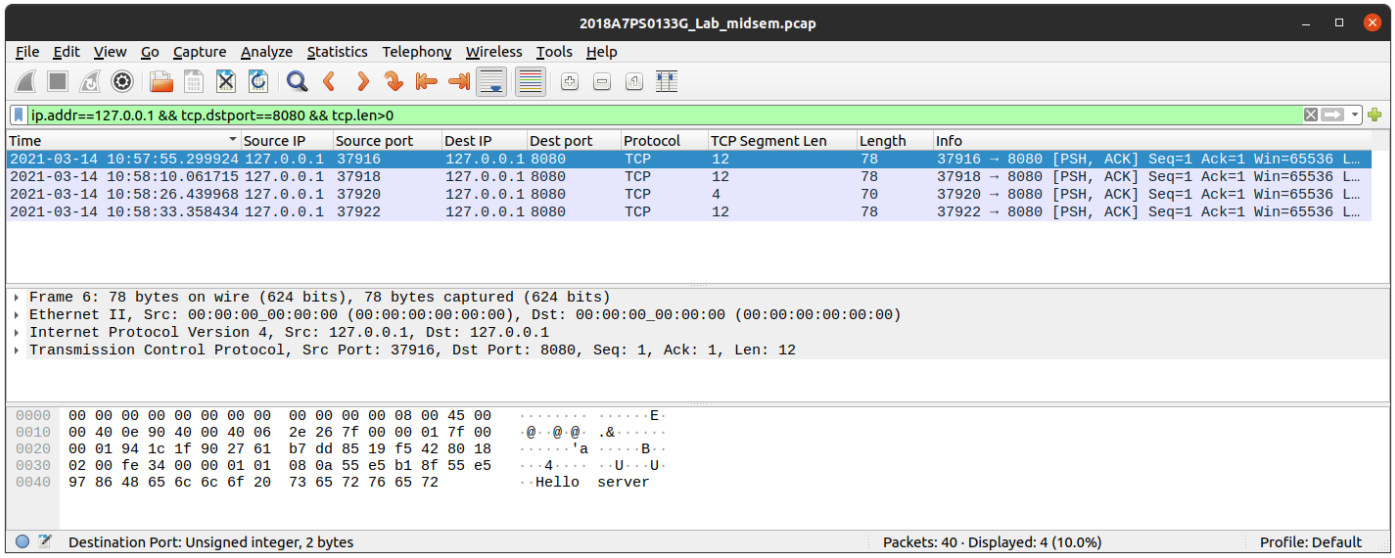
This filter is used to make sure that only the messages on the specified IP address and port number are shown. It also ensures that packets are only shown if the protocal being used is TCP.



The columns show all necessary information about each packet like source and destination addresses and the length of each packet and each TCP segment.

To further filter out packets that were sent specifically from the client to the server and contained messages, we use the previous filter and extend it to:

```
ip.addr==127.0.0.1 && tcp.dstport==8080 && tcp.len>0
```



This filters out messages that do not contain any messages, as they have TCP Segent lengths of zero. We also observe that the TCP segment length is the same as the length of the message being sent (as shown with example above where message being sent is 12 charcters [Hello server] and the TCP segment Length is also 12 bytes).

The difference in size of the TCP segment length and the packet length (dsiplayed under the column ‘length’ in the window) of 66 bytes is due to the headers used by each protocol while sending packets, regardless of whether the message contains any data in it at the transpot layer level.

Here 32 bytes is contributed by the TCP layer header. The reamaining 34 bytes are contributed by the layers below it:

- 20 bytes in the network layer (by the IP protocol).
- 14 bytes in the data link and physical layers (by the Ethernet protocol).

Refer to the README file to view instructions on how to run the processes.