IaS - Exercise sheet 3

Task 1 - TCP Chat

The idea is to have a driver code *tcp_chat_launcher.py* that allows to start the application as a server or client, depending on the number of arguments given. For the server, only the port number is needed, for the client also the server's ip address is needed.

In the tcp_chat package (TestPyPi) we have two methods, one for initializing a server peer, which listens for a client initially, and the other for the client, which connects to the listening peer. If the *launch_client* method doesn't receive the correct parameters and the socket, trying to connect to the server, throws an error, it would be catched and a server would be launched instead. Once the connection is established the *run_chat* method is responsible for updating sockets (with `select()`), request data from socket and check if connection is still there (otherwise close connection), receive data and send data if something is typed in the standard input.

To close the connection, enter `:q` as a message.

Task 2/3 - UDP Chat

The idea is to have a driver code, like in the tcp-chat, *udp_chat_launcher.py* that allows to start either the central server unit, or a new client peer. If you run the launcher passing only the port, the server will be initialized, if you pass the server's ip address and port as arguments a new client will be connected to the server.

The udp_chat package (TestPyPi) contains two files, one for the client and one for the server side. Once you start a server, this will listen to all new peers and add them into the user dictionary, which acts as an adress book {username: client addr}. When you are connected to the server, you can request a userlist, a roomlist, you can connect to a peer/user, you can join a room or create one.

Table of client commands

Commands	Arguments	Meaning
poke	username	connect to that username (like private messages)
kick	-	kick all peers from room
roomlist	-	get a list with the avaiable rooms
userlist	-	get a list with the avaiable users
roomcreate	roomname	create a new room with the name roomname
roomjoin	roomname	join a room (like group chat)
roomleave	-	leave actual room you are in

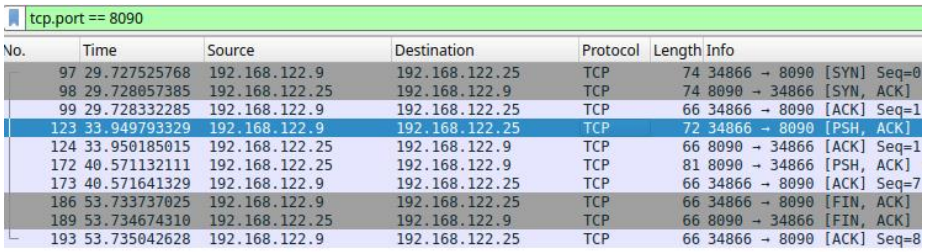
In order to run this commands, just prepend `$` to the command (ex. `$userlist`).

Task 3b)

With the command: `$userlist` you get the list of participants. It would be implemented in the "network" level, because you are routing some datagrams from a source to a destination.

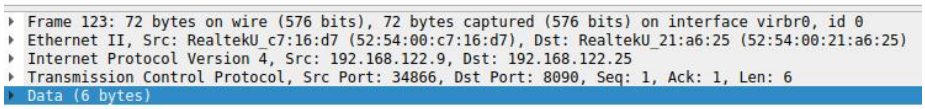
Task 4 - Network Analysis

a)

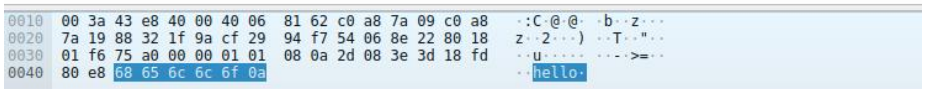


tcp.port == 8090

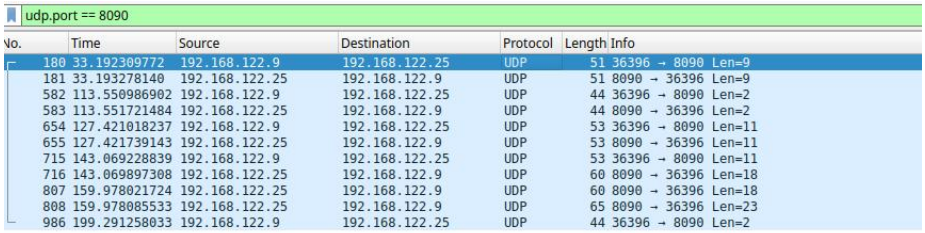
No.	Time	Source	Destination	Protocol	Length	Info
97	29.727525768	192.168.122.9	192.168.122.25	TCP	74	34866 → 8090 [SYN] Seq=0
98	29.728057385	192.168.122.25	192.168.122.9	TCP	74	8090 → 34866 [SYN, ACK]
99	29.728332285	192.168.122.9	192.168.122.25	TCP	66	34866 → 8090 [ACK] Seq=1
123	33.949793329	192.168.122.9	192.168.122.25	TCP	72	34866 → 8090 [PSH, ACK]
124	33.950185015	192.168.122.25	192.168.122.9	TCP	66	8090 → 34866 [ACK] Seq=1
172	40.571132111	192.168.122.25	192.168.122.9	TCP	81	8090 → 34866 [PSH, ACK]
173	40.571641329	192.168.122.9	192.168.122.25	TCP	66	34866 → 8090 [ACK] Seq=7
186	53.733737025	192.168.122.9	192.168.122.25	TCP	66	34866 → 8090 [FIN, ACK]
189	53.734674310	192.168.122.25	192.168.122.9	TCP	66	8090 → 34866 [FIN, ACK]
193	53.735042628	192.168.122.9	192.168.122.25	TCP	66	34866 → 8090 [ACK] Seq=8

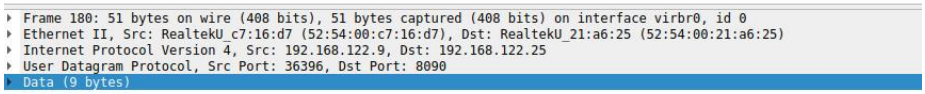


```
▸ Frame 123: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface virbr0, id 0
▸ Ethernet II, Src: RealtekU_c7:16:d7 (52:54:00:c7:16:d7), Dst: RealtekU_21:a6:25 (52:54:00:21:a6:25)
▸ Internet Protocol Version 4, Src: 192.168.122.9, Dst: 192.168.122.25
▸ Transmission Control Protocol, Src Port: 34866, Dst Port: 8090, Seq: 1, Ack: 1, Len: 6
▸ Data (6 bytes)
```



```
0010  00 3a 43 e8 40 00 40 06  81 62 c0 a8 7a 09 c0 a8   ·:C·@·@· ·b··z···
0020  7a 19 88 32 1f 9a cf 29  94 f7 54 06 8e 22 80 18   z··2···) ··T··"··
0030  01 f6 75 a0 00 00 01 01  08 0a 2d 08 3e 3d 18 fd   ··u····· ··-·>=··
0040  80 e8 68 65 6c 6c 6f 0a                           ··hello·
```





b)

TCP: Initialising the two peers you first see a *SYN* message, that initiates and establishes the connection, followed by an *ACK* message, that confirms to the other side that it has received the *SYN*. The *SYN, ACK* is a *SYN* from local device, and *ACK* of the earlier packet. In the end, when the peers terminate the communication, the *FIN* message is sent. All this messages belong to the transport/nework level. The data are processed, but not needed to be sent to the application level. You can see for the chat messages the flag *PSH*, that means "pushing". It refers to the fact that the messages are pushed directly from or to the buffer (depending if it's receiver or sender side) and than to the Application level, without waiting for additional data. This packets are sent to the application level, so that the different clients can read messages and send

back some others.

UDP: No handshaking between UDP sender-receiver! All the sent packeges are transmitted from the network, to the transport and the application level. The packages captured by wireshark with a datagram like "ul, rl, ..." are than handled by the application level with an intern protocol. The normal messages are displayed in the chat application as in the tcp-chat.

The packages begin with the header segment, similar in the style, but not always the same, and in the end is the data segment. You can also see how the datas are splitted (ex. "hi fro m alpine" sent in the tcp example). In addition, in tcp a line feed is appent to the message.