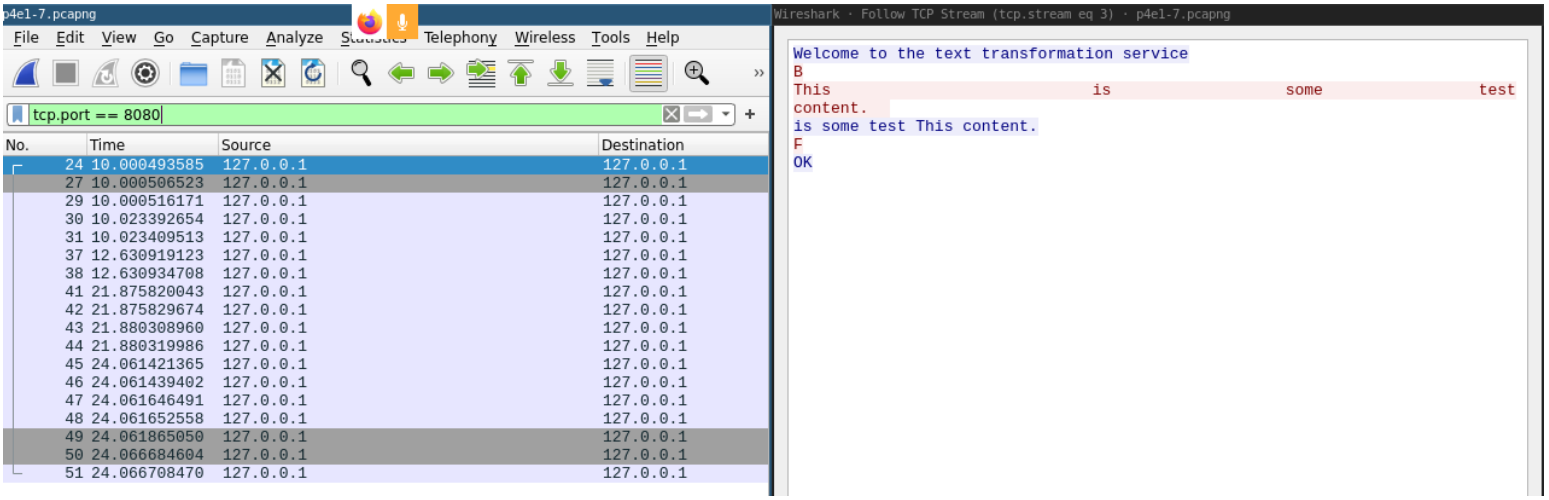
Network and Distributed Systems
Laboratory Work

Practical Exercises 4

Name, Surname 1: Jakub Hejhal **Name, Surname 2**: **Degree** (Ing. del Software/Ing. Informática/Ing. Computadores): Erasmus **Group** (A/B/C): B **PC Label in the lab** (e.g. 012): own laptop

Trace 1 (**p4e1-7.pcapng**):

Exercise 1. Use the “Analize -> Follow -> TCP stream” option of the Wireshark to see the information interchange between the client and the server. Show a screenshot with this information.



Exercise 2. Does the client send the messages (letter and text) in the same TCP segment? Why?

No, it does not, because I explicitly flush the socket after writing the command letter, such that especially in case of “F”, the command is sent immediately to the server and doesn’t wait around in the buffer.

.

Exercise 3. What is the port used by the client? What is the one of the server? What are the fields of the TCP header where they are stored?
Client uses port 55394, which is chosen by the operating system and we have no control over it, it's more or less random. Server binds to port 8080. It's stored in the Source/Destination Port TCP header field.

Exercise 4. What is the initial sequence number used by the client? What is the initial sequence number used by the server?

Client seq_n: 1754341604
Server seq_n: 2757352421

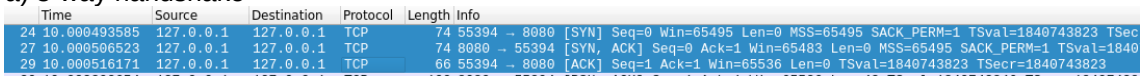
Although Wireshark is nice enough to substract this initial sequence number from every segment in ongoing TCP connection and shows us the relative sequence number.

Network and Distributed Systems
Laboratory Work

Exercise 5. Take screenshots of the TCP segments responsible of the following activities.

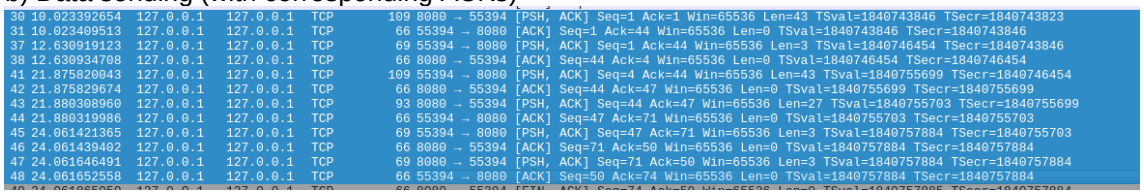
a) Connection initialization
b) Data sending
c) Finish a connection.

a) 3-way handshake



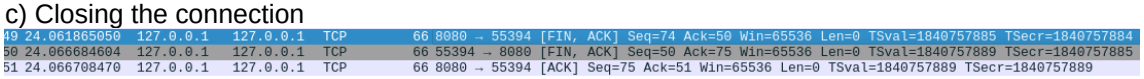
	Time	Source	Destination	Protocol	Length	Info
24	10.000493585	127.0.0.1	127.0.0.1	TCP	74	55394 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM=1 TSval=1840743823 TSec
27	10.000506523	127.0.0.1	127.0.0.1	TCP	74	8080 → 55394 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM=1 TSval=1840
29	10.000516171	127.0.0.1	127.0.0.1	TCP	66	55394 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=1840743823 TSecr=1840743823

b) Data sending (with corresponding ACKs)



30	10.023392654	127.0.0.1	127.0.0.1	TCP	109	8080 → 55394 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=43 TSval=1840743846 TSecr=1840743823
31	10.023409513	127.0.0.1	127.0.0.1	TCP	66	55394 → 8080 [ACK] Seq=1 Ack=44 Win=65536 Len=0 TSval=1840743846 TSecr=1840743846
37	12.630919123	127.0.0.1	127.0.0.1	TCP	69	55394 → 8080 [PSH, ACK] Seq=1 Ack=44 Win=65536 Len=3 TSval=1840746454 TSecr=1840743846
38	12.630934708	127.0.0.1	127.0.0.1	TCP	66	8080 → 55394 [ACK] Seq=44 Ack=4 Win=65536 Len=0 TSval=1840746454 TSecr=1840746454
41	21.875820043	127.0.0.1	127.0.0.1	TCP	109	55394 → 8080 [PSH, ACK] Seq=4 Ack=44 Win=65536 Len=43 TSval=1840755699 TSecr=1840746454
42	21.875829674	127.0.0.1	127.0.0.1	TCP	66	8080 → 55394 [ACK] Seq=44 Ack=47 Win=65536 Len=0 TSval=1840755699 TSecr=1840755699
43	21.880308960	127.0.0.1	127.0.0.1	TCP	93	8080 → 55394 [PSH, ACK] Seq=44 Ack=47 Win=65536 Len=27 TSval=1840755703 TSecr=1840755699
44	21.880319986	127.0.0.1	127.0.0.1	TCP	66	55394 → 8080 [ACK] Seq=47 Ack=71 Win=65536 Len=0 TSval=1840755703 TSecr=1840755703
45	24.061421365	127.0.0.1	127.0.0.1	TCP	69	55394 → 8080 [PSH, ACK] Seq=47 Ack=71 Win=65536 Len=3 TSval=1840757884 TSecr=1840755703
46	24.061439402	127.0.0.1	127.0.0.1	TCP	66	8080 → 55394 [ACK] Seq=71 Ack=50 Win=65536 Len=0 TSval=1840757884 TSecr=1840757884
47	24.061646491	127.0.0.1	127.0.0.1	TCP	69	8080 → 55394 [PSH, ACK] Seq=71 Ack=50 Win=65536 Len=3 TSval=1840757884 TSecr=1840757884
48	24.061652558	127.0.0.1	127.0.0.1	TCP	66	55394 → 8080 [ACK] Seq=50 Ack=74 Win=65536 Len=0 TSval=1840757884 TSecr=1840757884

c) Closing the connection



49	24.061865050	127.0.0.1	127.0.0.1	TCP	66	8080 → 55394 [FIN, ACK] Seq=74 Ack=50 Win=65536 Len=0 TSval=1840757885 TSecr=1840757884
50	24.066684604	127.0.0.1	127.0.0.1	TCP	66	55394 → 8080 [FIN, ACK] Seq=50 Ack=75 Win=65536 Len=0 TSval=1840757889 TSecr=1840757885
51	24.066708470	127.0.0.1	127.0.0.1	TCP	66	8080 → 55394 [ACK] Seq=75 Ack=51 Win=65536 Len=0 TSval=1840757889 TSecr=1840757889

Exercise 6. How many sequence numbers are in each communication end (client and server) consumed during the initialization and closing of the connection?

One sequence number is consumed for initializing and one for closing the simplex connection, so 4 sequence numbers are consumed by the full-duplex init and close process.

Exercise 7. Observe the size of the sliding window that is transmitted in the segments that travel in both directions (client and server). Does the sliding window size change? What are its values in the client and in the server?

Only first 2 packets have a sliding window size different from the rest of the communication – 65495 and 65483 for the client and server respectively. In the rest of the communication, the sliding window=65536.

Trace 2 (**p4e8.pcapng**):

Exercise 8. Does the client receive any response after trying to connect to the server? If the answer is positive then, what are the special characteristics of this response?

If there is no service running at port 8080 to which the client is trying to connect to, the kernel responds with TCP RST, by which it tells the clients to kill the connection instantly.

Trace 3 (**p4e9-10.pcapng**):

Exercise 9. Did the 3 client manage to connect? If one has not been able to connect, does it receive a notification about full queue?

Yes, all 3 clients manage to connect, which is not what I would expect. I create the server socket with backlog=1, as instructed, which is according to Oracle documentation - “requested maximum length of the queue of incoming connections”

As I understand it, the following should happen:

- First client connects successfully and server starts handling it’s request.

- Second client tries to create the Socket to server, and is successful as well, because there is no-one in the waiting queue, but trying to read data from socket blocks, because server hasn’t yet accepted the connection and hasn’t sent anything.

- Third client tries to create the Socket to server, but is unsuccessul, because the waiting queue is full.

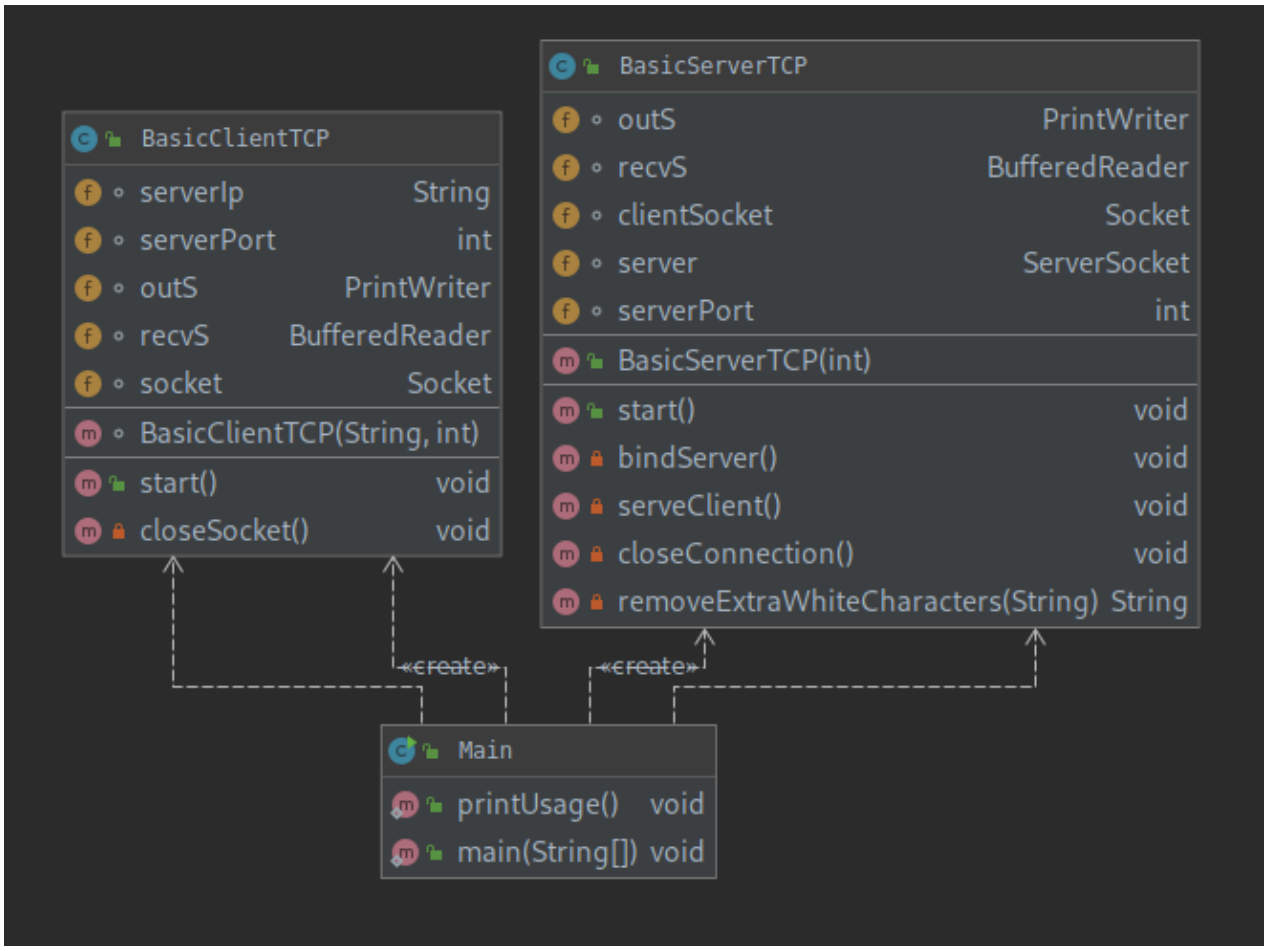
But that is not what happens. Looking at the communication in wireshark, we can see that all three connections are created – we can see 3-way handshake for every client, even though server hasn’t yet accepted the second and third client connection and even though the third client shouldn’t connect at all.

When I try to connect 4-th client, another interesting thing happens. The creation of a Socket blocks on the client side, so it looks like backlog=1 actually means backlog=2, or I don’t know.

Exercise 10. Do the waiting clients (those queued ones) have the connection initialised? Or the connection initialization is performed after the clients are pulled from the queue (in the `accept` method)?

As I said, all 3-way handshakes happen and all connections are initialized even though they are not pulled from the queue by the accept method.

The code UML:


BasicClientTCP
serverIp String
serverPort int
outS PrintWriter
recvS BufferedReader
socket Socket
BasicClientTCP(String, int)
start() void
closeSocket() void
BasicServerTCP
outS PrintWriter
recvS BufferedReader
clientSocket Socket
server ServerSocket
serverPort int
BasicServerTCP(int)
start() void
bindServer() void
serveClient() void
closeConnection() void
removeExtraWhiteCharacters(String) String
«create»
«create»
Main
printUsage() void
main(String[]) void