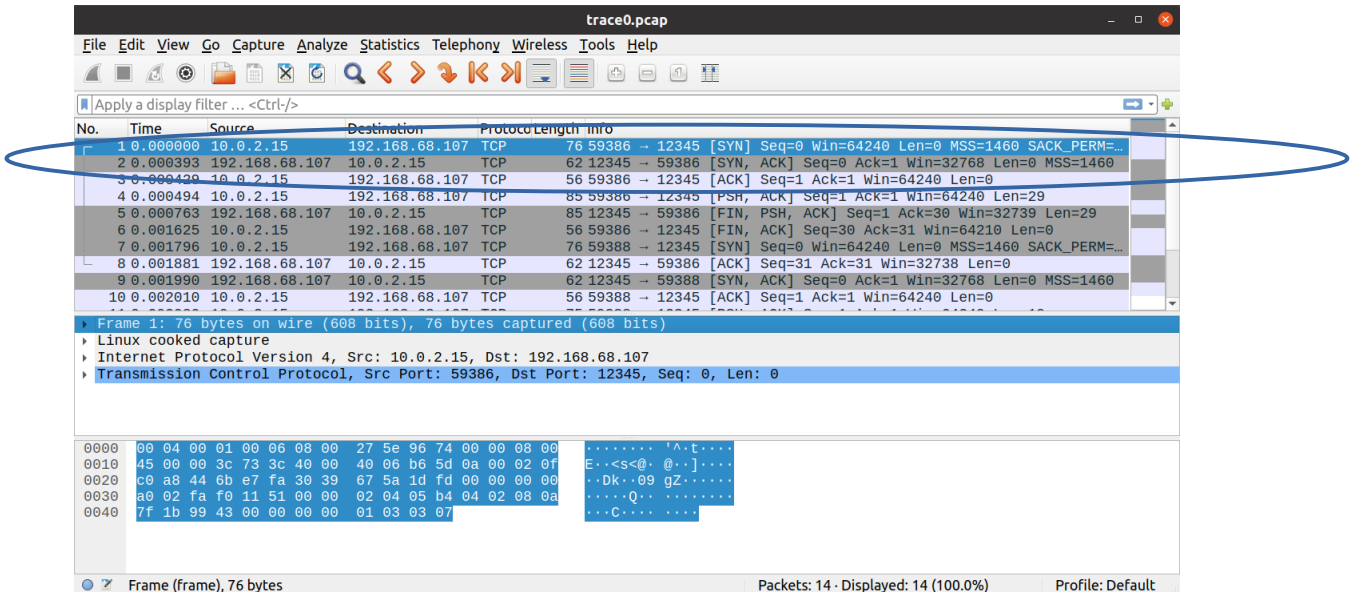
Part one

<u>Running the TCP client and server code</u>

The TCP handshake:

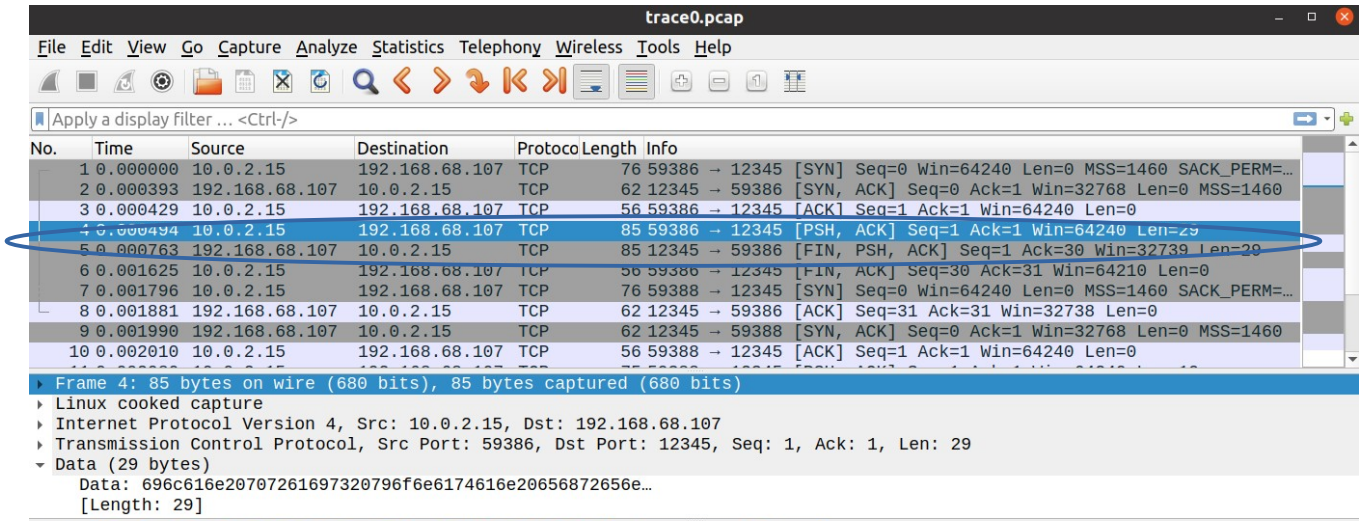
The client is the one who initiates the connection with the server:

First, he sends to the server a TCP packet with SYN flag (which means that this packet is for starting a connection) and with the initial sequence number of the client, which is 0 in that case. The server sends a packet back to the client, with SYN flag, and with the initial sequence number of the server, which is 0, and with ack number 1, which tells the client that the packet which he sent to the server accepted successfully. Then the client sends a packet to the server, with sequence number 1, and ack number 1, which tells the server that the packet which he sent to the client accepted successfully. This is the connection process.

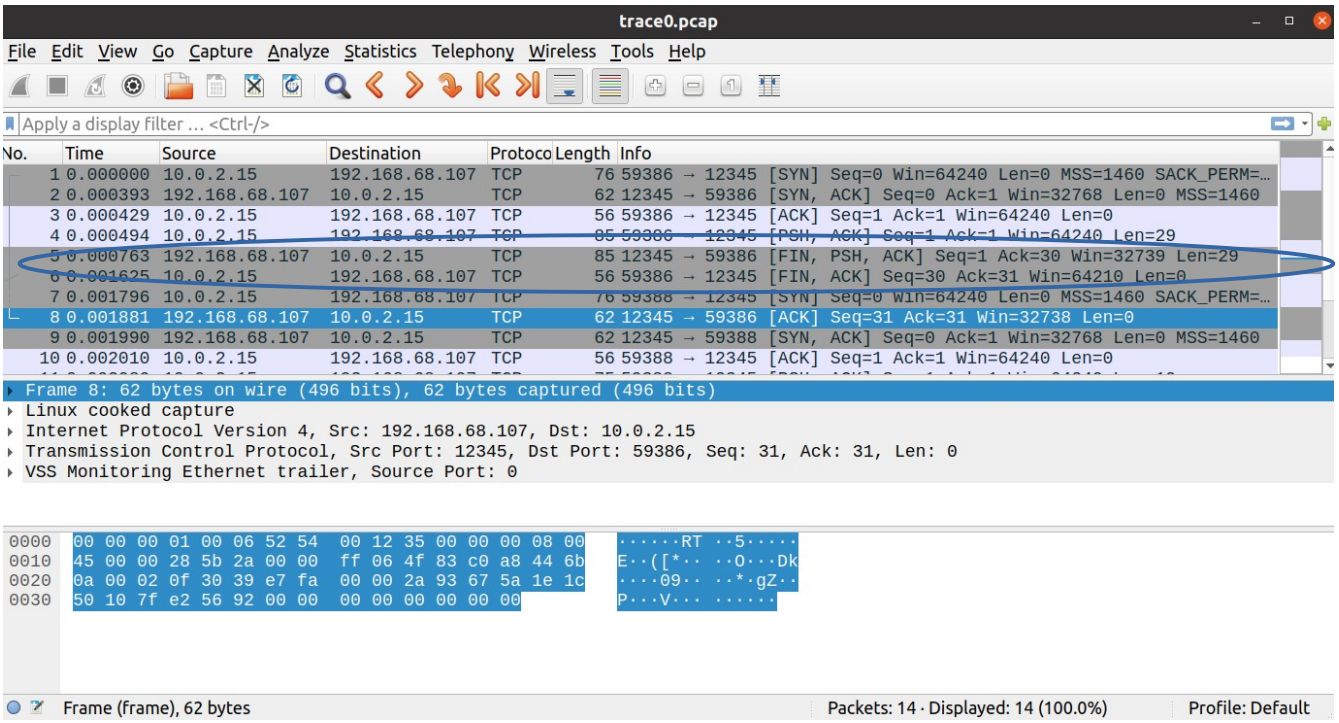


Sending a message to the server:

The client sends a message to the server, which contains our names. This packet is with sequence number 1, ack number 1 (because the server hasn't sent a packet since the SYN packet, so the ack number hasn't changed), and the data length is 29 bytes. The server sends a packet back to the client, which contatins our names in upper case. This packet is with FIN flag (will be explained later), sequence number 1, ack number 30 (because the previous ack number was 1 and the server got data with length 29 from the client, so the next sequence number that the server hopes to get is 30), and the data length is 29 bytes.



Closing the connection:
The server is the one who initiates the connection closing. In the last packet he sent to the client (the packet where the server sent back to the client our names in upper case), the FIN flag (which means that this packet is for closing the connection) was on. This packet is actually an example for how does the TCP protocol uses a single packet for many uses – in this packet, the server sent our names in upper case, sent ack for the packet that he received before from the server (the packet with our names), and also told the client that he is closing the connection (FIN flag). Next, the client sends a packet back to the server, with sequence number 30 (because the previous sequence number was 1 and the client sent before a packet with data with length 29 bytes), and ack number 31 (because the previous ack number was 1, and the server sent before a packet with data with length 29 bytes, and the another 1 byte is an ack for the FIN). Next, the server sends a packet back to the client, with sequence number 31, and ack number 31 (the 1 byte increase is because of the FIN). This is the connection closing process.



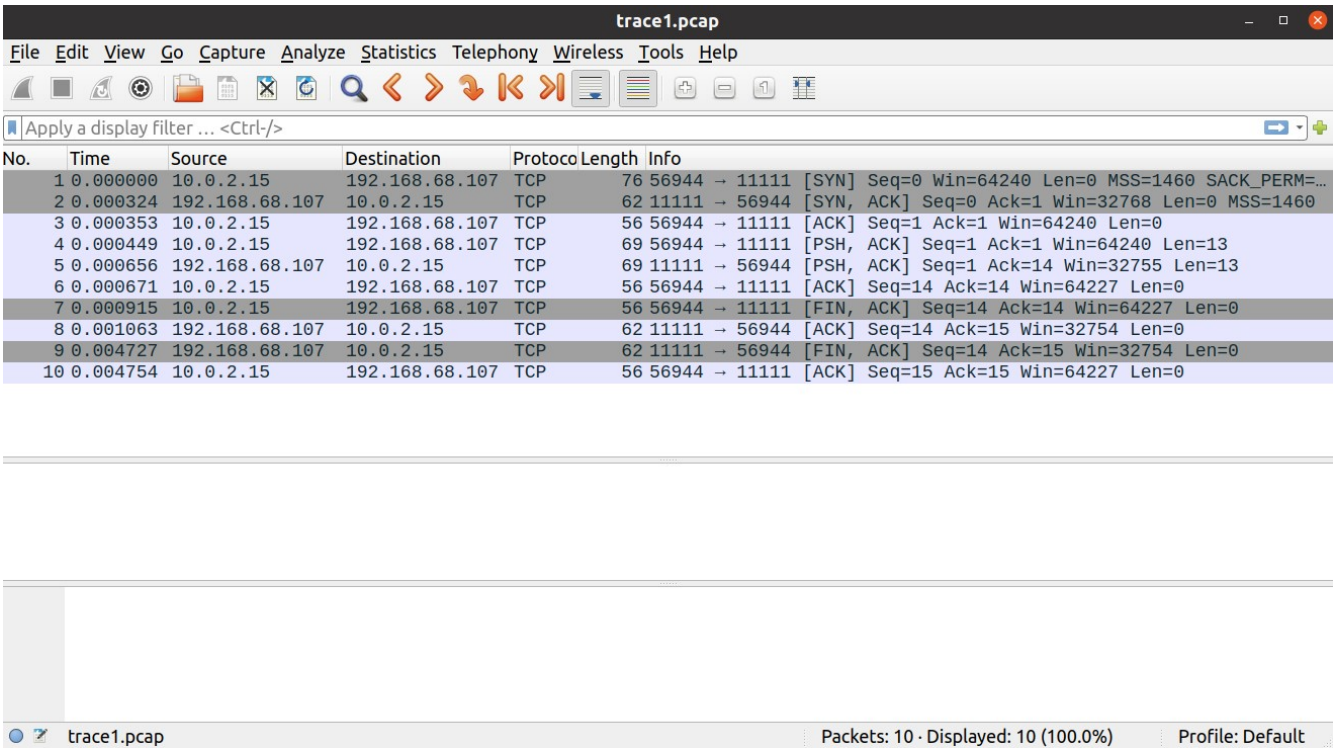
Running the different versions of the TCP client and server code

Version 1:

The server code creates a TCP socket, binds it to the given port, and sets the backlog to 1. The server socket keep getting connections from the different clients (single client every time), prints the ip address of the new client, and while the client socket is not empty (the messages that the client sent haven't been read completely), the server reads 1024 bytes from the client socket, prints the read content, and sends it back to the client in upper case using the client socket. When the client socket is empty, the server closes the connection.

The client code creates a TCP socket and connects to the given ip and port. The client sends "Hello world!" using the socket, reads 1024 bytes of the received content from socket, closes the connection, and then prints the message.

In the first packets, connection between the client to the server is established (SYN flag), and an ack for the connection is sent. The client is the one who initiates the connection. In the next packet, the client sends to the server "Hello world!", and then the server sends back to the client "Hello World!", in upper case, and also sends an ack for the previous packet. Then the client sends an ack back to the server. Then, the client closes the connection (FIN flag) as we see in the code, and the server sends an ack back to the client. In the next packet, the server closes the connection (FIN flag), because as we see in the code, the buffer is empty (because the server read the message from the socket completely), so the while-loop is broken, and the server is closing the connection. Next, the client sends an ack back to the server.



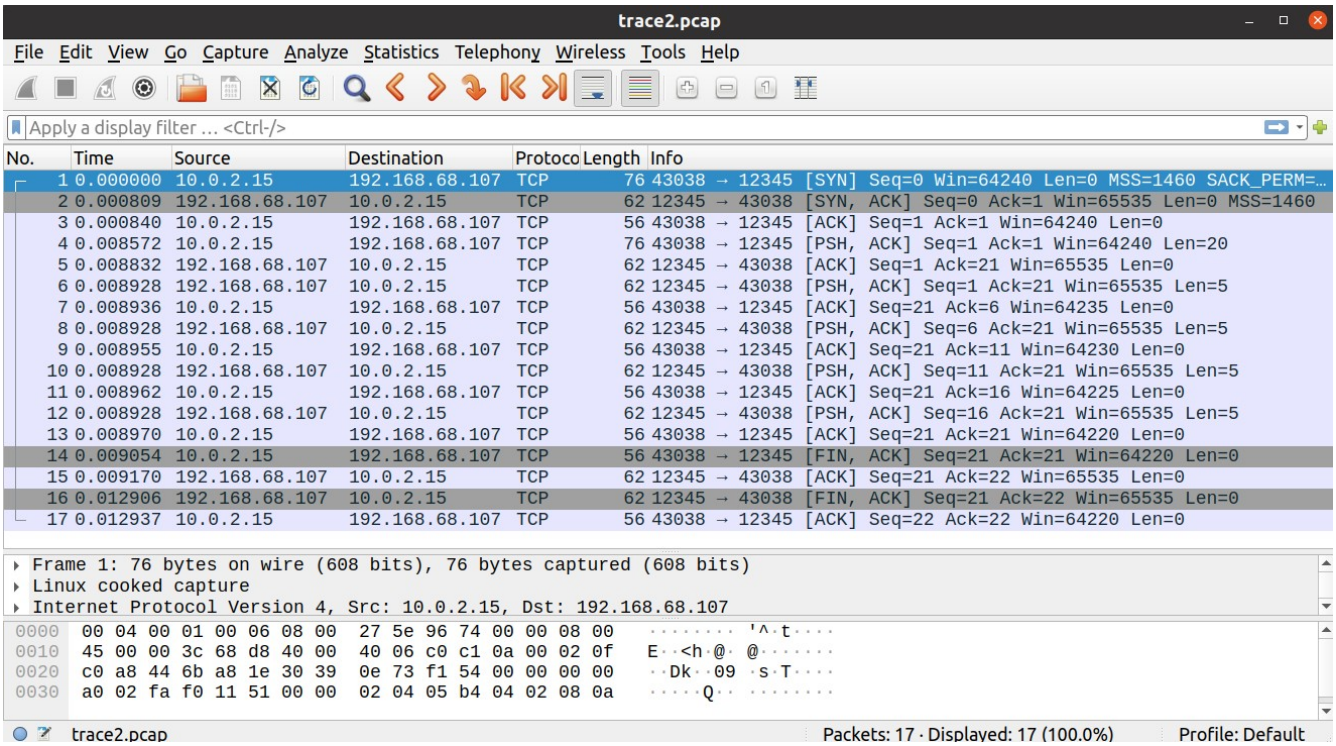
No.	Time	Source	Destination	Protoco	Length	Info
1	0.000000	10.0.2.15	192.168.68.107	TCP	76	56944 → 11111 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=…
2	0.000324	192.168.68.107	10.0.2.15	TCP	62	11111 → 56944 [SYN, ACK] Seq=0 Ack=1 Win=32768 Len=0 MSS=1460
3	0.000353	10.0.2.15	192.168.68.107	TCP	56	56944 → 11111 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.000449	10.0.2.15	192.168.68.107	TCP	69	56944 → 11111 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=13
5	0.000656	192.168.68.107	10.0.2.15	TCP	69	11111 → 56944 [PSH, ACK] Seq=1 Ack=14 Win=32755 Len=13
6	0.000671	10.0.2.15	192.168.68.107	TCP	56	56944 → 11111 [ACK] Seq=14 Ack=14 Win=64227 Len=0
7	0.000915	10.0.2.15	192.168.68.107	TCP	56	56944 → 11111 [FIN, ACK] Seq=14 Ack=14 Win=64227 Len=0
8	0.001063	192.168.68.107	10.0.2.15	TCP	62	11111 → 56944 [ACK] Seq=14 Ack=15 Win=32754 Len=0
9	0.004727	192.168.68.107	10.0.2.15	TCP	62	11111 → 56944 [FIN, ACK] Seq=14 Ack=15 Win=32754 Len=0
10	0.004754	10.0.2.15	192.168.68.107	TCP	56	56944 → 11111 [ACK] Seq=15 Ack=15 Win=64227 Len=0

Version 2:

The server code is almost same as version 1, except from that the server reads 5 bytes from the client socket in every iteration, instead of 1024 bytes every iteration in version 1.

The client code is the same as version 1, except from that the client sends “Hello! Hello, World!” instead of “Hello world!” in version 1.

In the first packets, connection between the client to the server is established (SYN flag), and an ack for the connection is sent. The client is the one who initiates the connection. In the next packet, the client sends to the server “World! Hello world!”, and then the server sends ack back to the client. Then the server sends to the client every time a packet with 5 bytes of “World! Hello world!” in upper case, and the client sends an ack back to the server. This happens four times, because the length of “World! Hello World!” is 20 bytes and in every iteration 5 bytes are sent (as we can see in the code, the server in every iteration reads 5 bytes from the buffer and sends them to the client, until the buffer is empty).
Then, the client closes the connection (FIN flag) as we see in the code, and the server sends an ack back to the client. In the next packet, the server closes the connection (FIN flag), because as we see in the code, the buffer is empty (because the server read the message from the socket completely), so the while-loop is broken, and the server is closing the connection. Next, the client sends an ack back to the server.



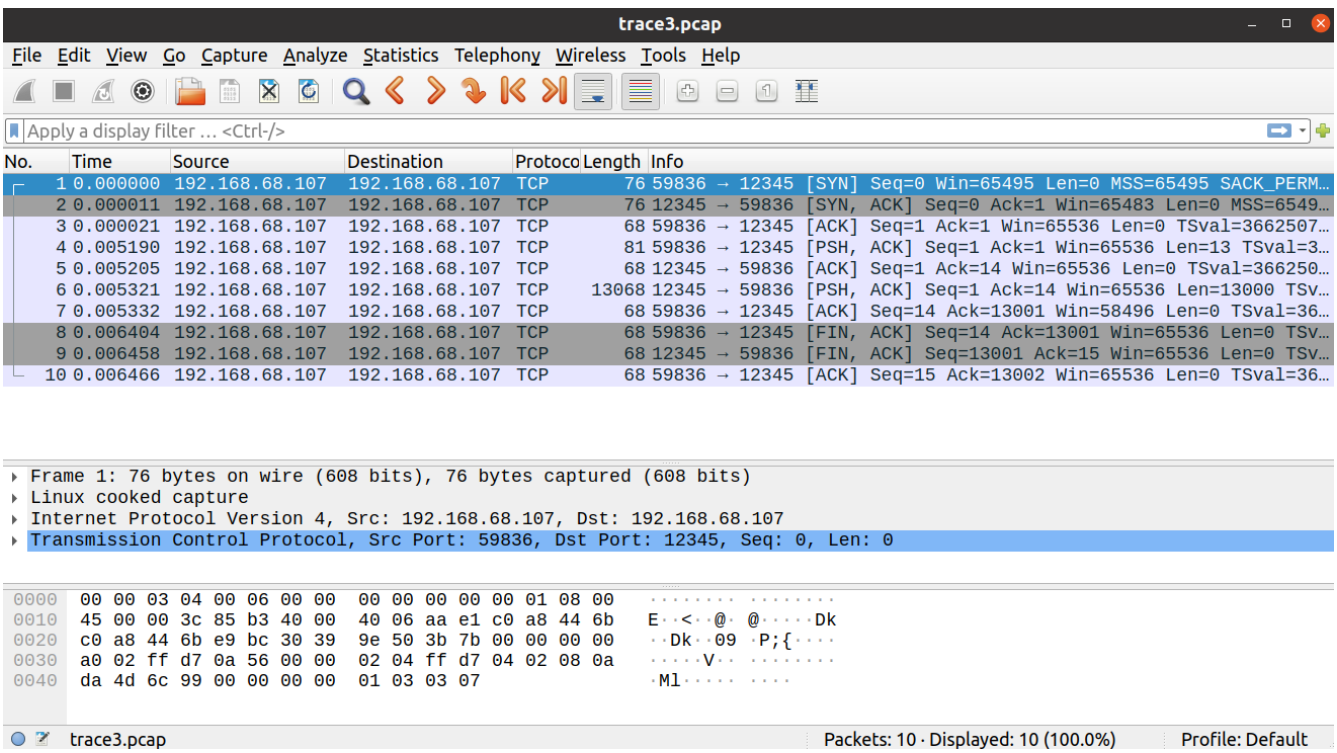
No.	Time	Source	Destination	Protoco	Length	Info
1	0.000000	10.0.2.15	192.168.68.107	TCP	76	43038 → 12345 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=…
2	0.000809	192.168.68.107	10.0.2.15	TCP	62	12345 → 43038 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
3	0.000840	10.0.2.15	192.168.68.107	TCP	56	43038 → 12345 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.008572	10.0.2.15	192.168.68.107	TCP	76	43038 → 12345 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=20
5	0.008832	192.168.68.107	10.0.2.15	TCP	62	12345 → 43038 [ACK] Seq=1 Ack=21 Win=65535 Len=0
6	0.008928	192.168.68.107	10.0.2.15	TCP	62	12345 → 43038 [PSH, ACK] Seq=1 Ack=21 Win=65535 Len=5
7	0.008936	10.0.2.15	192.168.68.107	TCP	56	43038 → 12345 [ACK] Seq=21 Ack=6 Win=64235 Len=0
8	0.008928	192.168.68.107	10.0.2.15	TCP	62	12345 → 43038 [PSH, ACK] Seq=6 Ack=21 Win=65535 Len=5
9	0.008955	10.0.2.15	192.168.68.107	TCP	56	43038 → 12345 [ACK] Seq=21 Ack=11 Win=64230 Len=0
10	0.008928	192.168.68.107	10.0.2.15	TCP	62	12345 → 43038 [PSH, ACK] Seq=11 Ack=21 Win=65535 Len=5
11	0.008962	10.0.2.15	192.168.68.107	TCP	56	43038 → 12345 [ACK] Seq=21 Ack=16 Win=64225 Len=0
12	0.008928	192.168.68.107	10.0.2.15	TCP	62	12345 → 43038 [PSH, ACK] Seq=16 Ack=21 Win=65535 Len=5
13	0.008970	10.0.2.15	192.168.68.107	TCP	56	43038 → 12345 [ACK] Seq=21 Ack=21 Win=64220 Len=0
14	0.009054	10.0.2.15	192.168.68.107	TCP	56	43038 → 12345 [FIN, ACK] Seq=21 Ack=21 Win=64220 Len=0
15	0.009170	192.168.68.107	10.0.2.15	TCP	62	12345 → 43038 [ACK] Seq=21 Ack=22 Win=65535 Len=0
16	0.012906	192.168.68.107	10.0.2.15	TCP	62	12345 → 43038 [FIN, ACK] Seq=21 Ack=22 Win=65535 Len=0
17	0.012937	10.0.2.15	192.168.68.107	TCP	56	43038 → 12345 [ACK] Seq=22 Ack=22 Win=64220 Len=0

Version 3:

The server code is almost same as version 1, except from that the server sends back to the client a 1000 times concatenation of the content that he read from the client socket in upper case, instead of sending just the content that he read from the client in upper case.

The client code is almost same as version 1, except from some things:
The client doesn't create the message in binary like in version 1, but he creates the message as a string and converts it to binary in the sending, using encode() method.
The client also reads 1024 bytes from the socket twice and prints it, instead of single read in version 1.

In the first packets, connection between the client to the server is established (SYN flag), and an ack for the connection is sent. The client is the one who initiates the connection. In the next packet, the client sends to the server "World! Hello world!", and then the server sends ack back to the client. Then the server sends to the client a 1000 times concatenation of "Hello, World!", in upper case, and then the client sends an ack back to the server. Then, the client closes the connection (FIN flag) as we see in the code, and the server sends an ack back to the client. In the next packet, the server closes the connection (FIN flag), because as we see in the code, the buffer is empty (because the server read the message from the socket completely), so the while-loop is broken, and the server is closing the connection. Next, the client sends an ack back to the server.

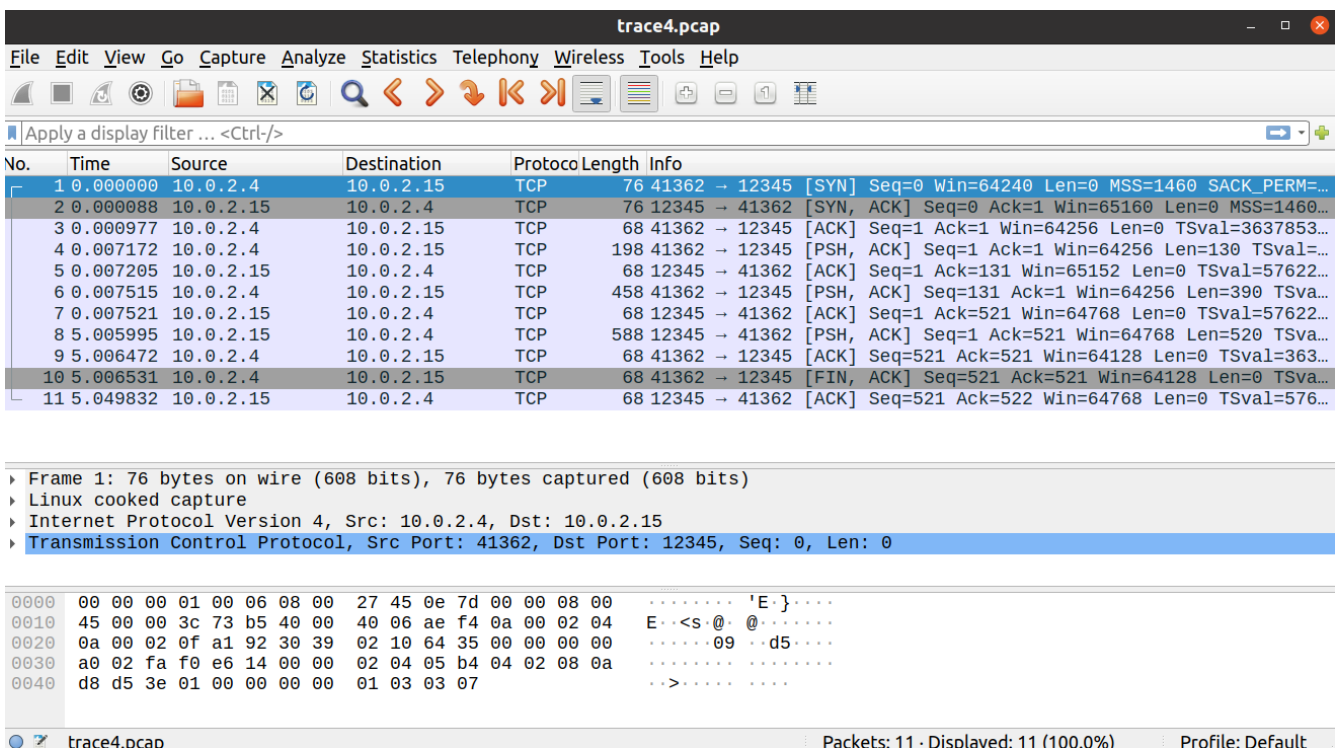


Version 4:

The server code is almost same as version 1, except from that the server sleeps for 5 seconds before every reading from the client socket.

The client code is the same as version 1, except from that the client sends a 10 times concatenation of “Hello, World!”, not in binary, and does that four times.

In the first packets, connection between the client to the server is established (SYN flag), and an ack for the connection is sent. The client is the one who initiates the connection. In the next packet, the client sends to the server a 10 times concatenation of “Hello, World!”, and then the server sends an ack back to the client. In the next packet, the client sends three times of the 10 times concatenation of “Hello, World!” (the client sent it actually 3 times after the first time, so it was all merged to one packet), and then the server sends an ack back to the client. In the next packet, the server sends to the client four times of the 10 times concatenation of “HELLO, WORLD!”, which is actually all of the data that the client sent before to the server, in upper case (all of that data was just sent now because the server has a 5 seconds timeout before he reads the data from the buffer and sends it back to the client, differently from the previous versions), and then the client sends an ack back to the server. Then, the client closes the connection (FIN flag) as we see in the code, and the server sends an ack back to the client.

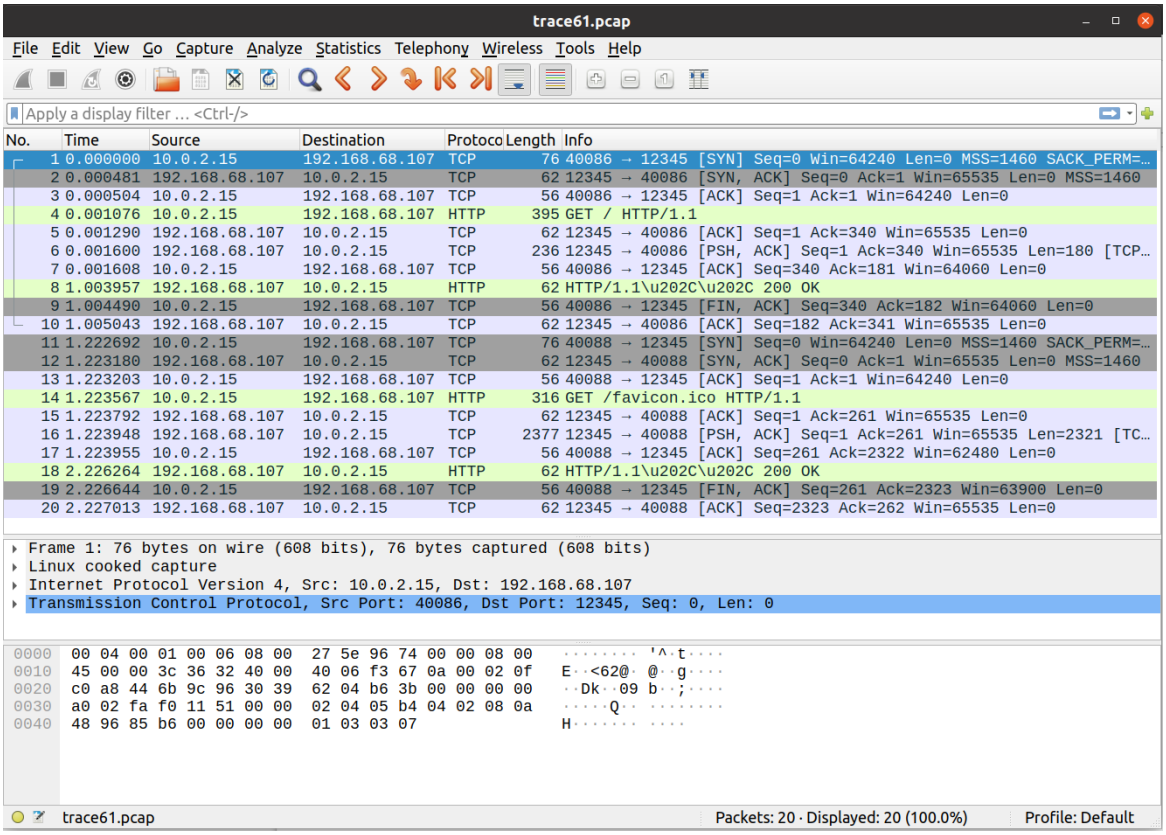


Part 2

<u>Running the server with the browser as a client</u>

First example – getting the file index.html

When the link localhost:12345 was typed in the browser, then the connection was established. Then, the browser sent to the server a request for index.html file (with HTTP protocol), and then the server found the file and sent its content to the browser, and acks were also sent. Then the server sent to the browser the status of the request (in HTTP protocol), and after that the connection was closed.
Then, another connection was established automatically, and the client sent to the server a request for favicon.ico file, and the server sent back the file content in the same procedure as the previous request, and then the connection was closed. The reason that the browser sent that request automatically is that the browser didn't find a favicon.ico file in the browser cache, so he requested one from the server.



Second example – getting the files index.html and back1.jpg

When the link localhost:12345 was typed in the browser, the connection was establised and the browser sent to the serer a request for index.html file, and received back the content of the file, as before, and then the connection was closed.
Next, when the link localhost:12345/back1.jpg was typed in the browser, another connection was establised and the browser sent to the server a request for back1.jpg file. Then the server sent back to the client the content of the file in 11 packets, because the size of back1.jpg is way bigger than the MSS (of course that the browser sent an ack to the server for every packet). Except of that the file content was not sent in a single packet, the procedure was same as before. Then the connection was closed.
In those requests, the browser didn't send an automatic request for favicon.ico file, and the reason for that is that a favicon.ico file has been already existed in the browser cache.