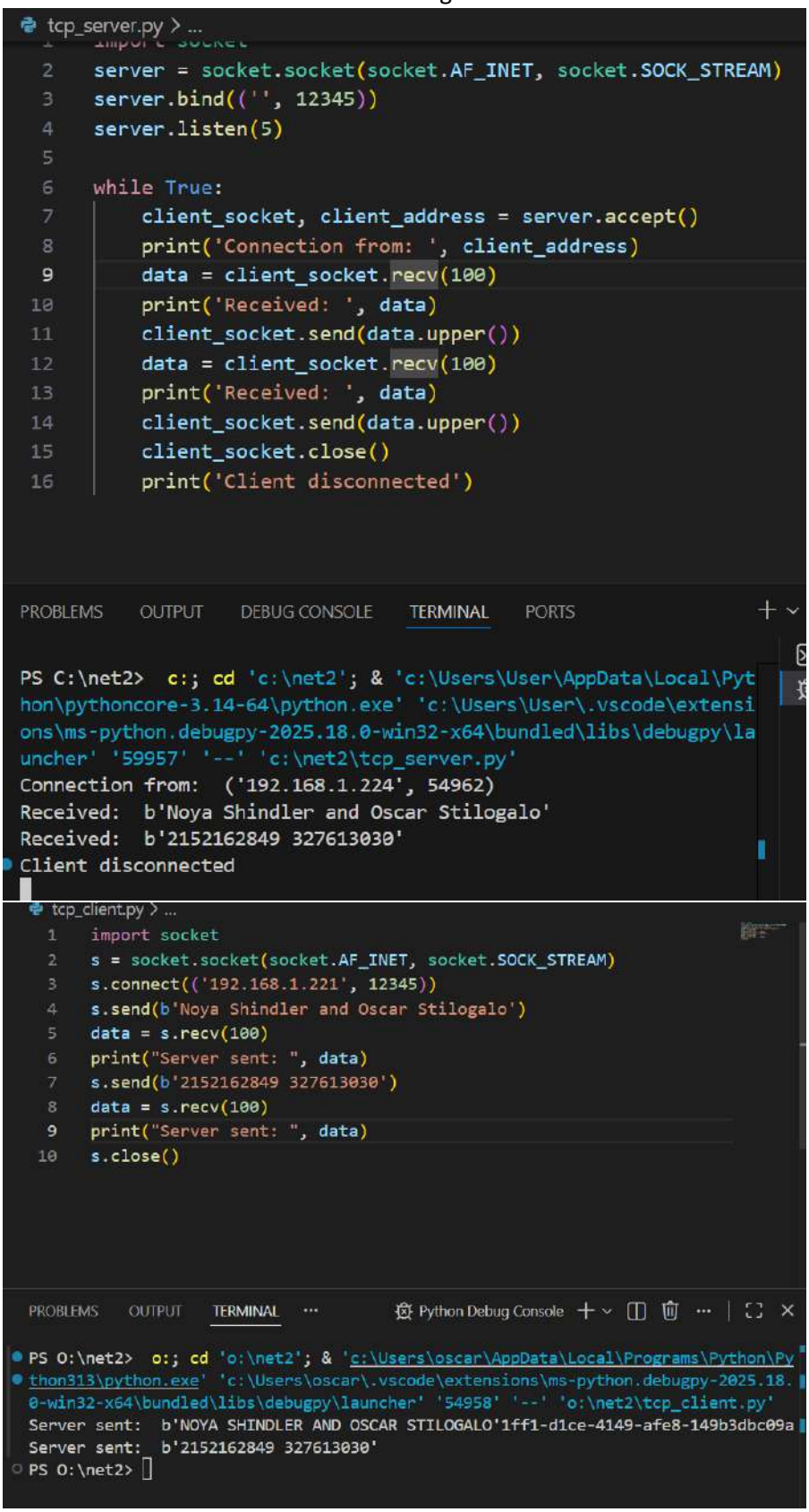
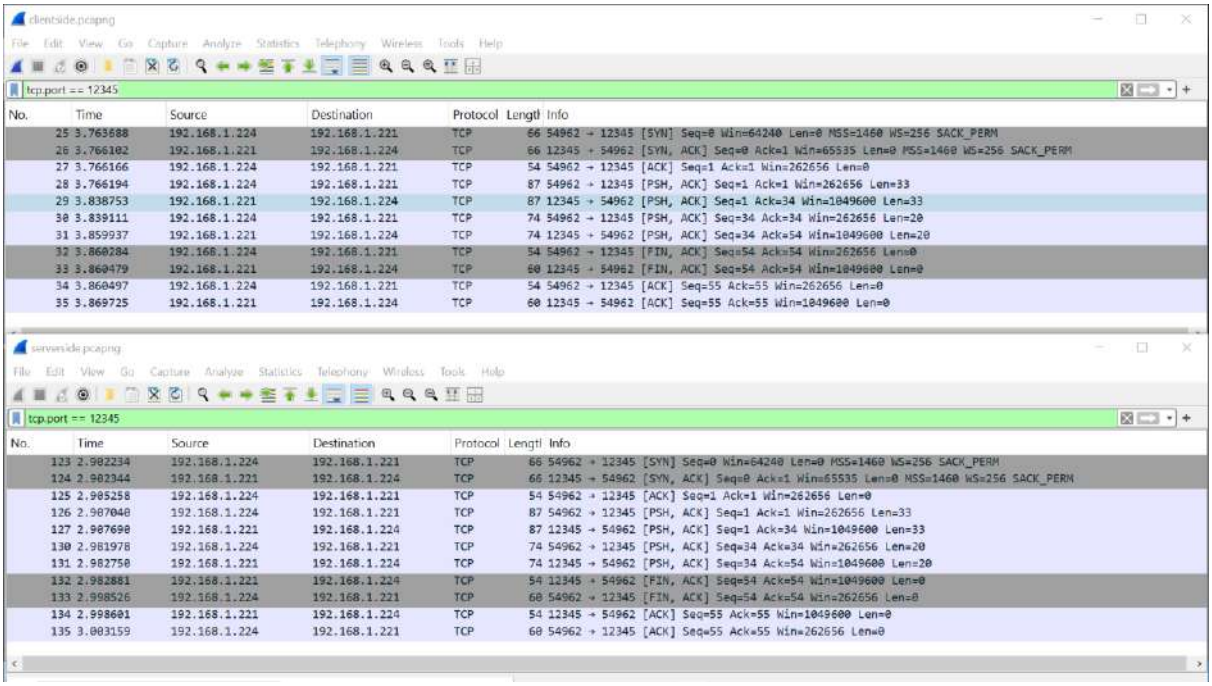
# TCP COMMUNICATION ANALYSIS

## Part 1:

1. The messages:

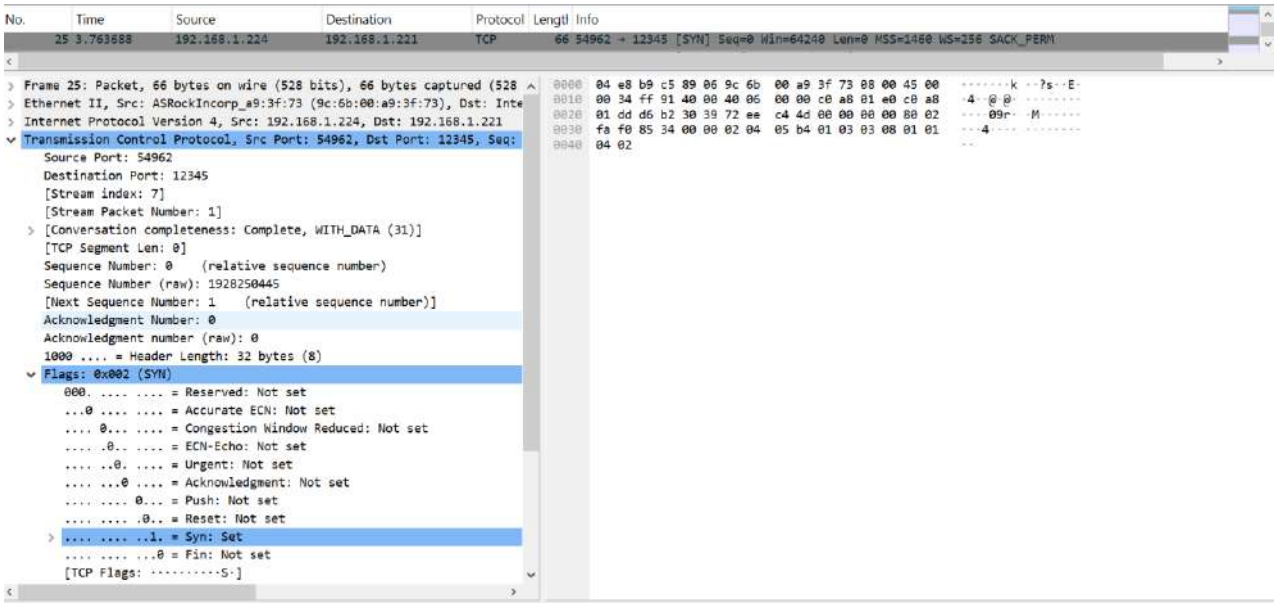


**2.** Wireshark




First message- client to server (224->221) only syn flag asks to make a connection

Sequence Number (raw): 1928250445 – where the stack starts

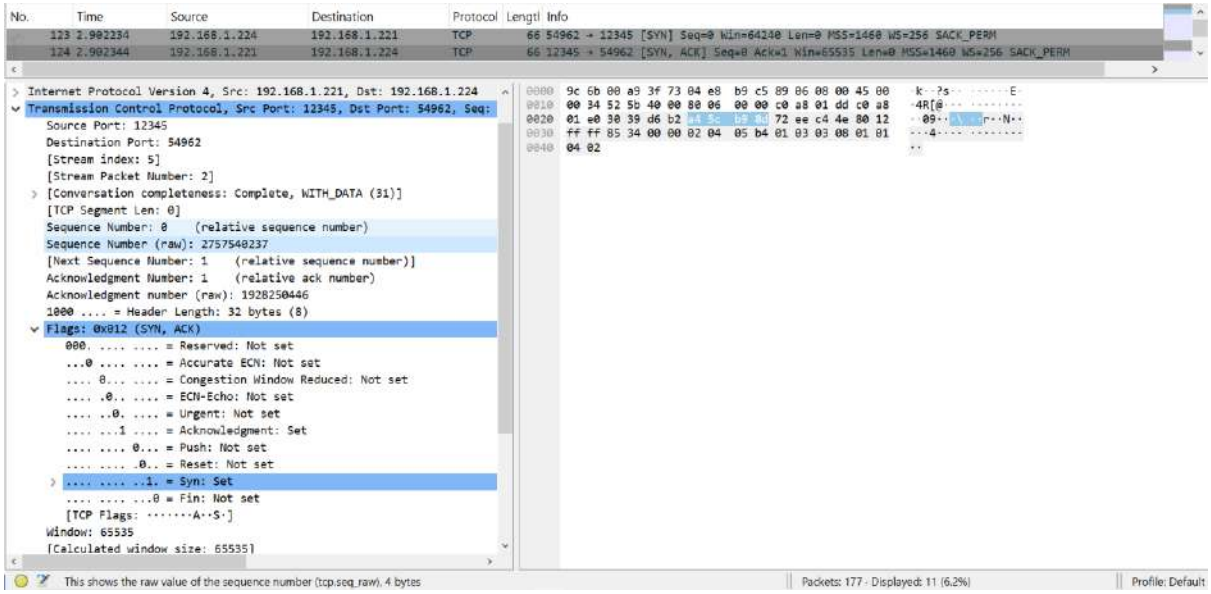


Second message- server to client (221->224), 2rd part of 3 part handshake,

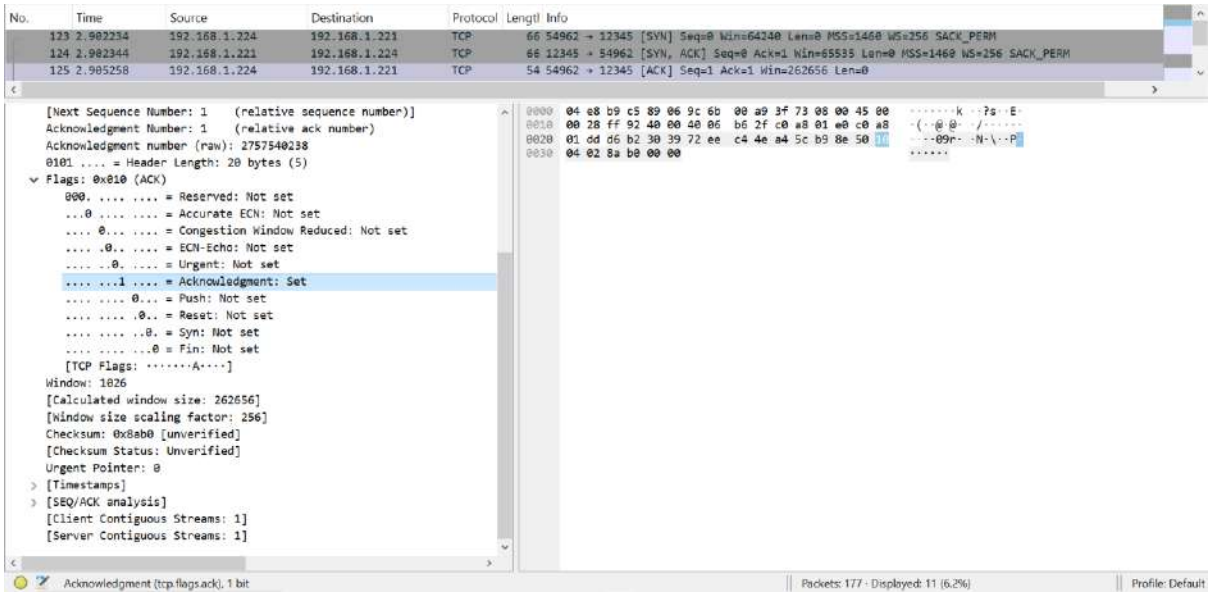
Acknowledgment number (raw): 1928250446 for one bit after the seq in the client

for syn the client sent.

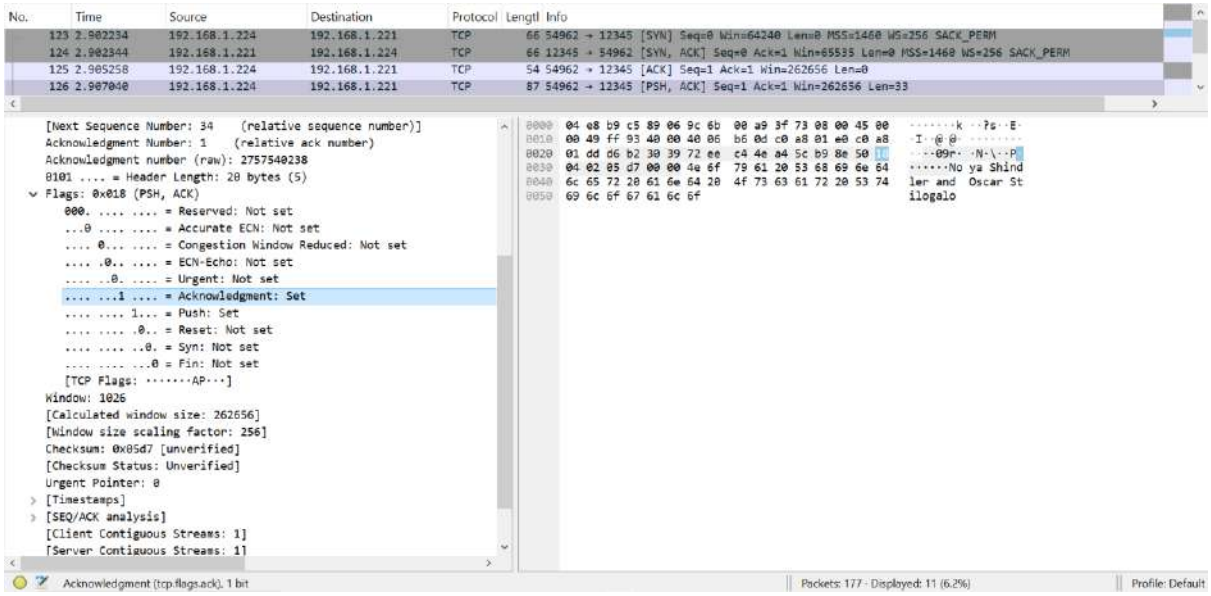
Sequence Number (raw): 2757540237: which is where the stack starts for the server




Third message- client to server (224->221),3rd part of 3part handshake only ack flag sets both server and Acknowledgment number (raw): 2757540238 for the 1 bit of syn

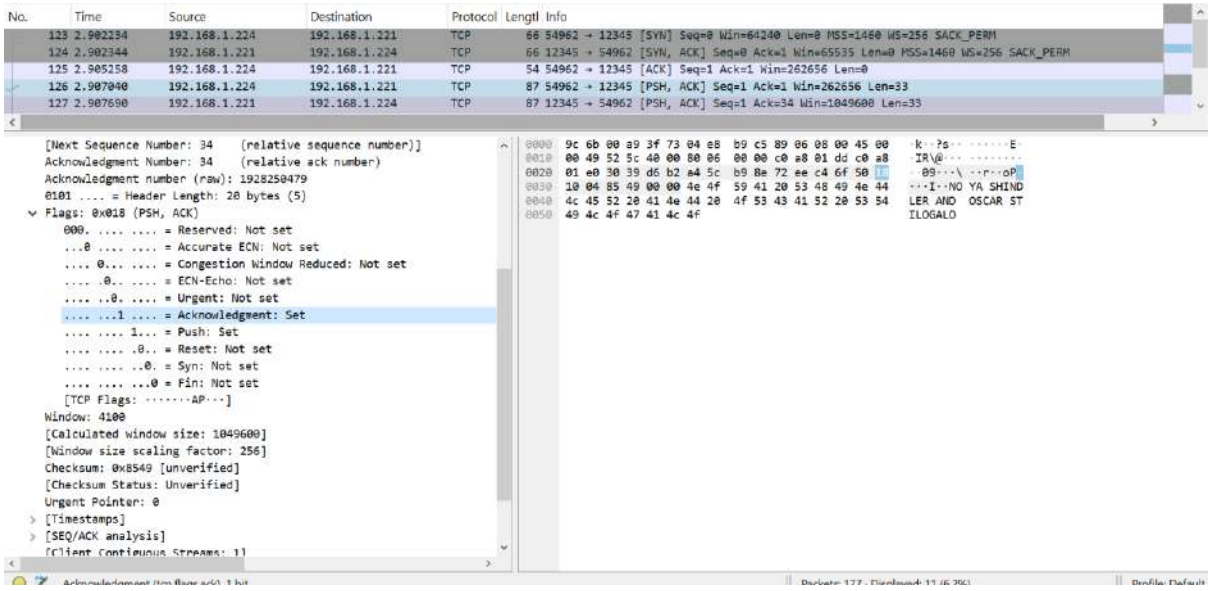


Forth message- client to server (224->221), sent the names with ack and psh flag Ack is at 1(dynamically) because that's where we start the ack list, note that msg len is 33



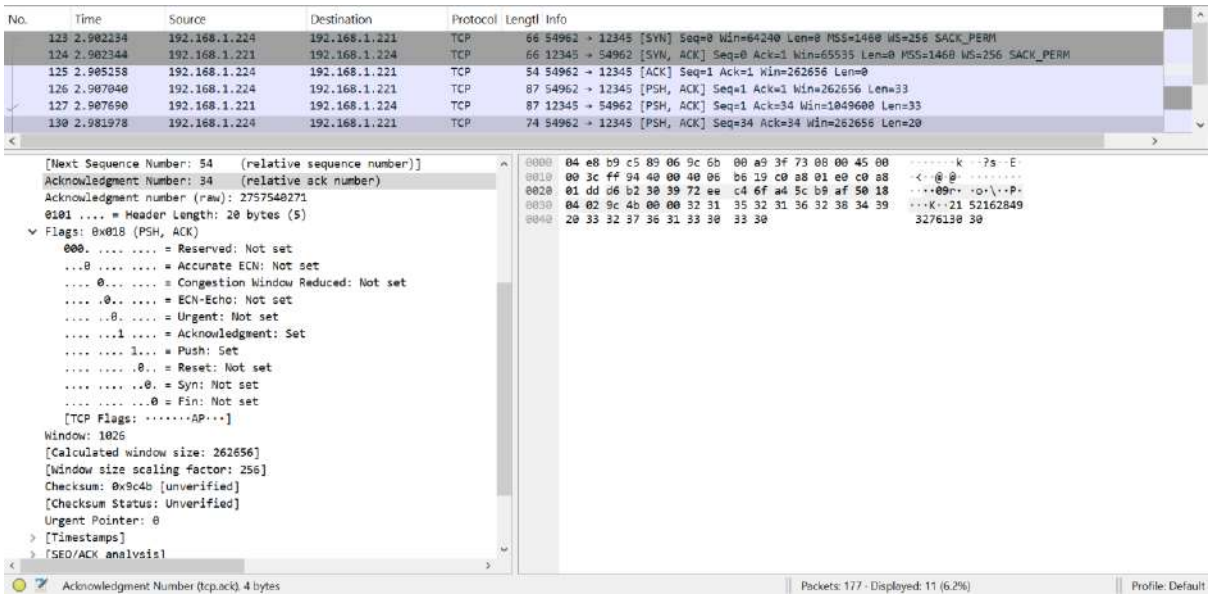

Fifth message- server to client (221->224), sent Acknowledgment number (raw): 1928250479 (meaning took all the 79-46 = 33 len of the msg) and sent together its response which as well has len 33

Sequence Number (raw): 2757540238 is still at the start

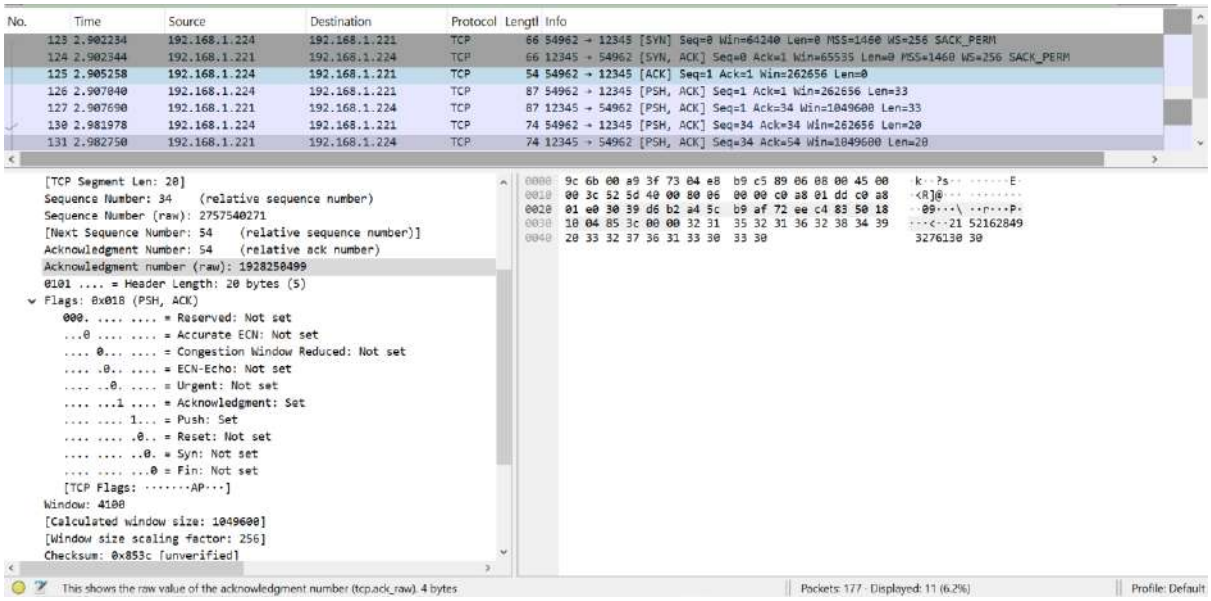


Sixth message- client to server (224->221), sent Acknowledgment number (raw): 2757540271 (meaning took all the 71 -38 = 33 len of the msg) and sent the ids which have len 20

Acknowledgment number (raw): 2757540271 still after the 33 + 1 bits

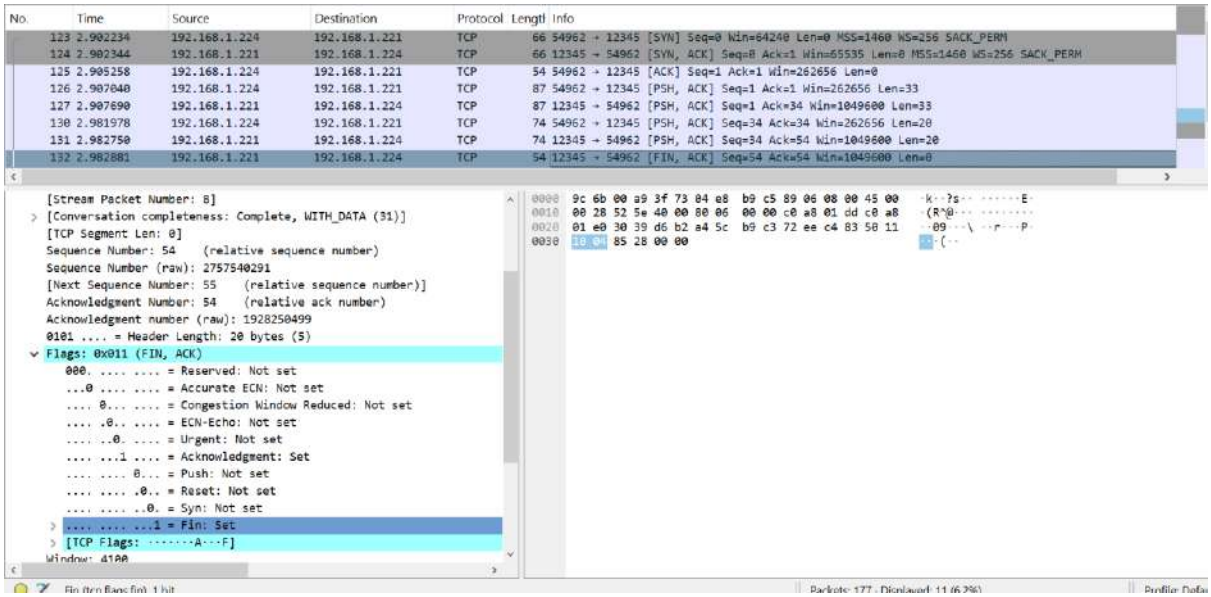



Seventh message- server to client (221->224), sent Acknowledgment number (raw): 1928250499 (1928250445+1+33+20) and sent the message back

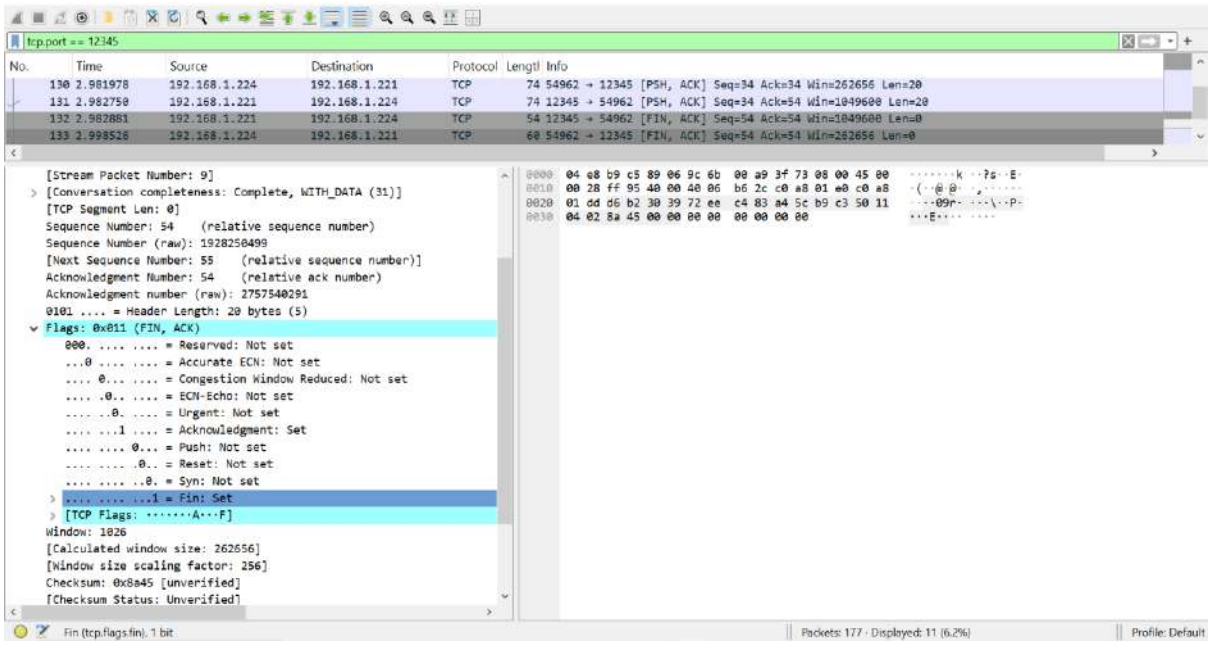


Eight message- server to client (221->224), asking to close connection Fin flag up part 1 of 4 part handshake, this happened before client could give ack for the server msg

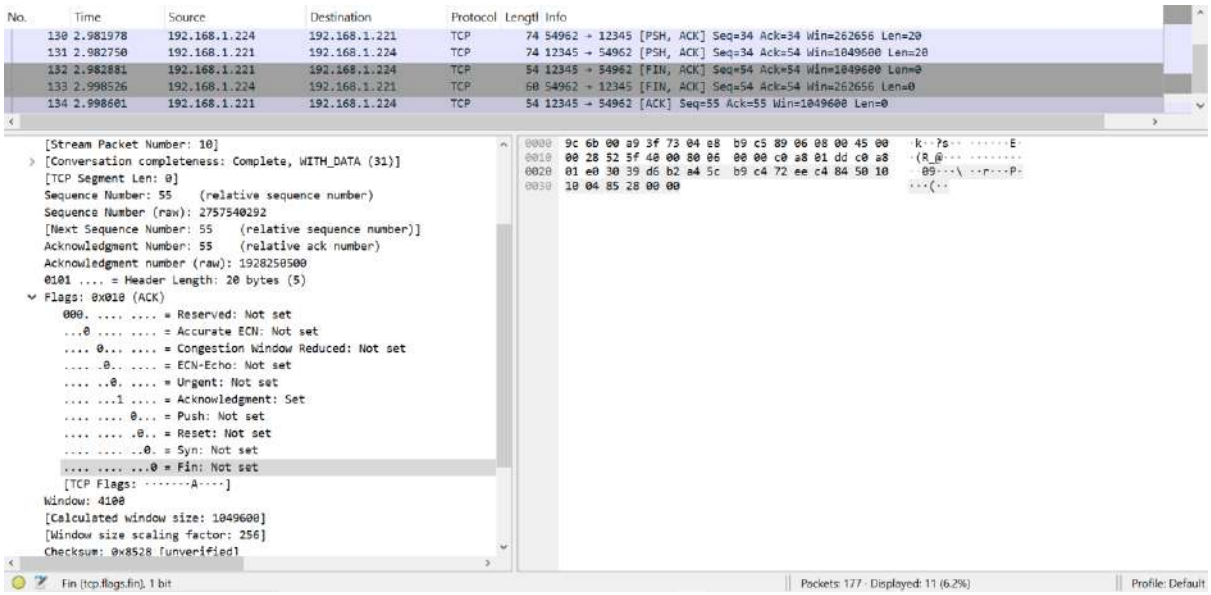
Sequence Number (raw): 2757540291 – even though we didn't get ack till here because client didn't send ack yet



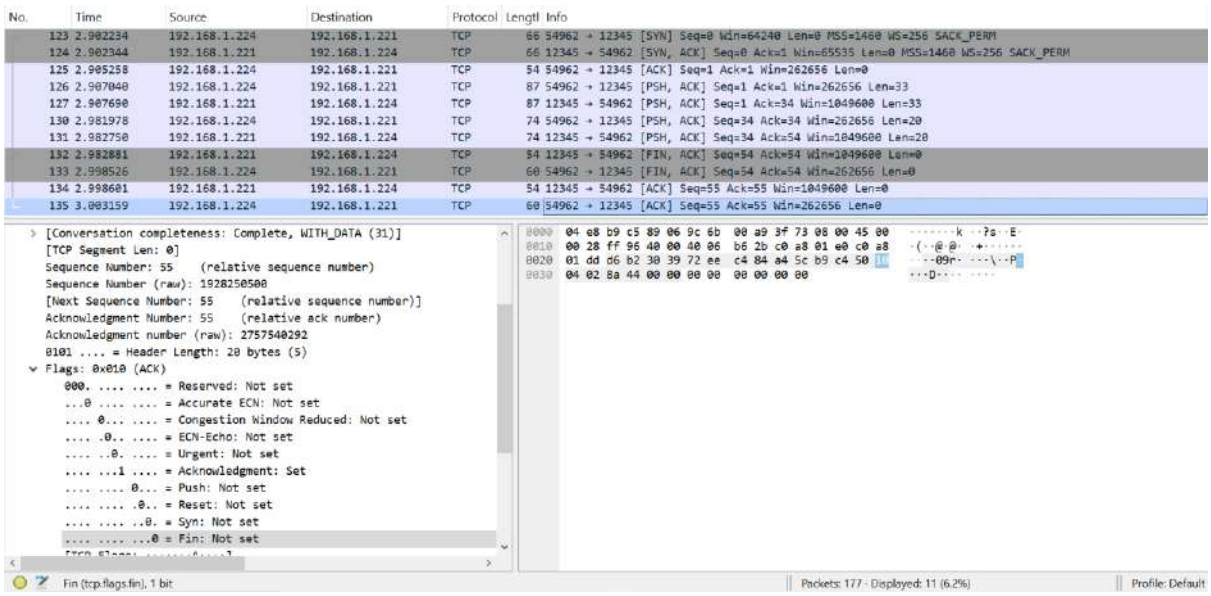

Ninth message- client to server (224->221), part 3 of 4 part handshake client didn't receive fin yet but it started to close the connection itself with Acknowledgment number (raw): 2757540291 (meaning it got the last 20)



Tenth message- server to client (221->224 ), part 4 of 4 part handshake client received server ack to close connection so it can close the connection now




Final message- client to server (224->221),part 2 of 4 part handshake server received ack for fin and now can close connection



## Part 2:

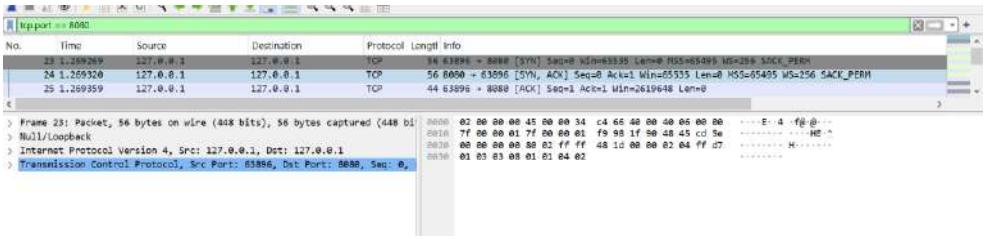
**1.** We will demonstrate the **keep alive** functionality using single_fin and double_fin files

**Single_fin-** here I opened the default linkage of index.html and quickly refreshed as to demonstrate the keep alive (in practice chrome opens 2 ports for every request so I actually had to make python client to do this with the same port)

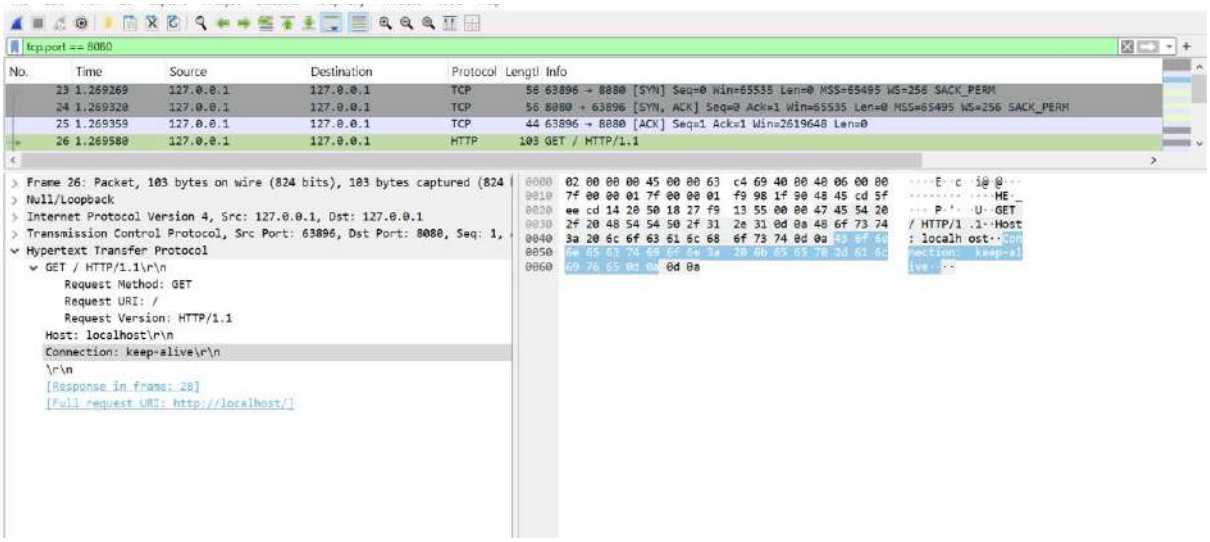


### lets analyze single_fin.pcapng:

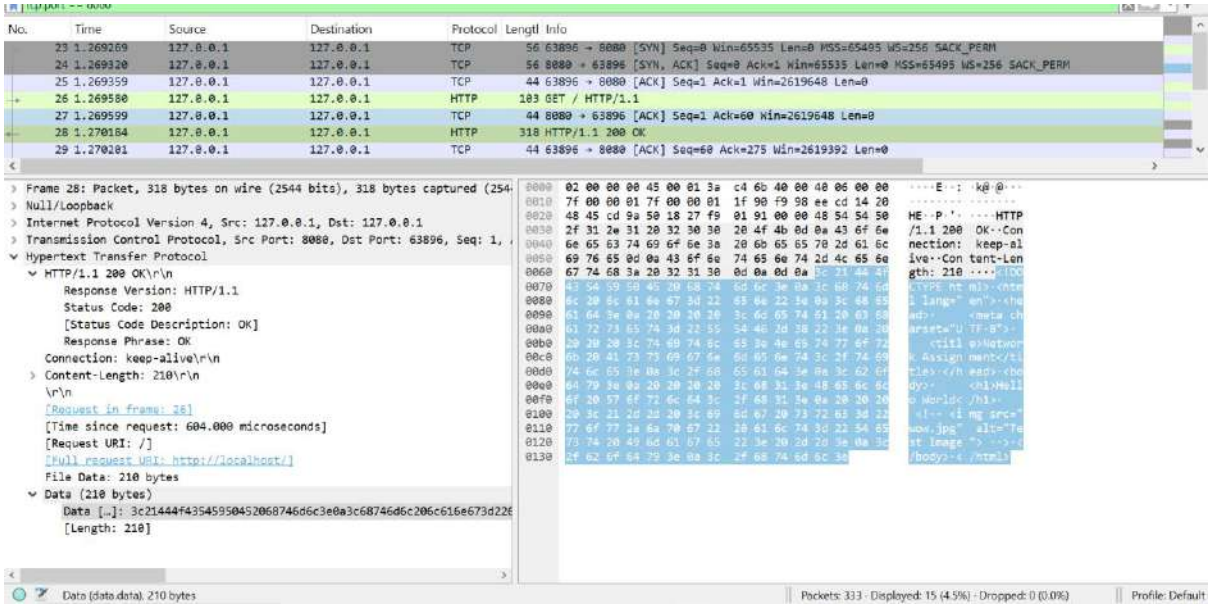
First 3 packets are syn handshake as explained in part 1



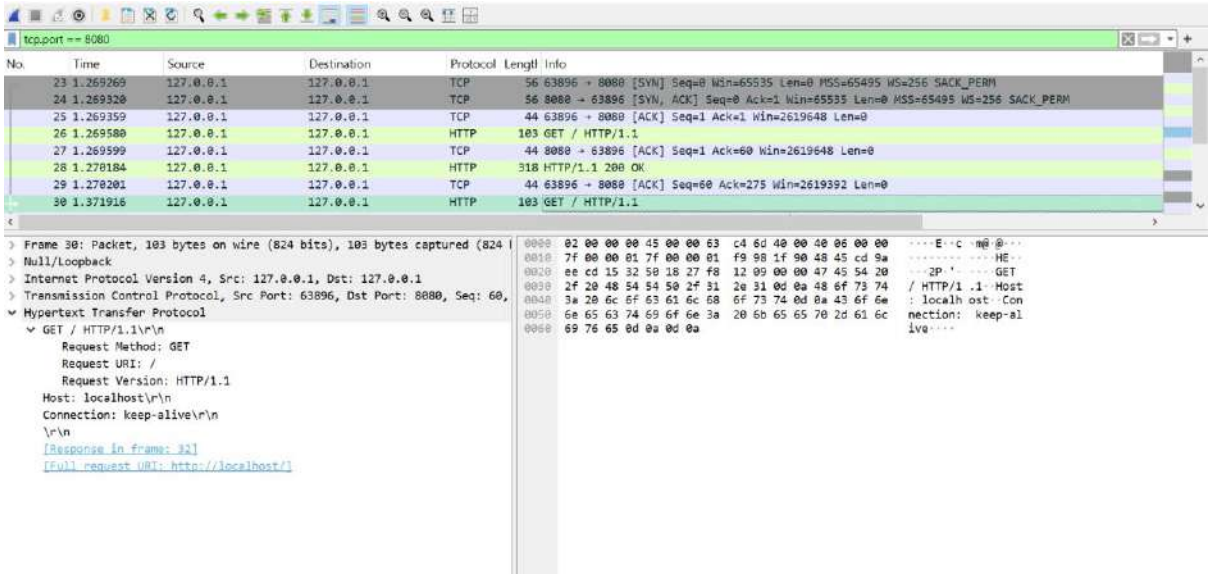

Next message is the get / HTTP/1.1 request for the page- do notice that the connection type in the http protocol is keep-alive



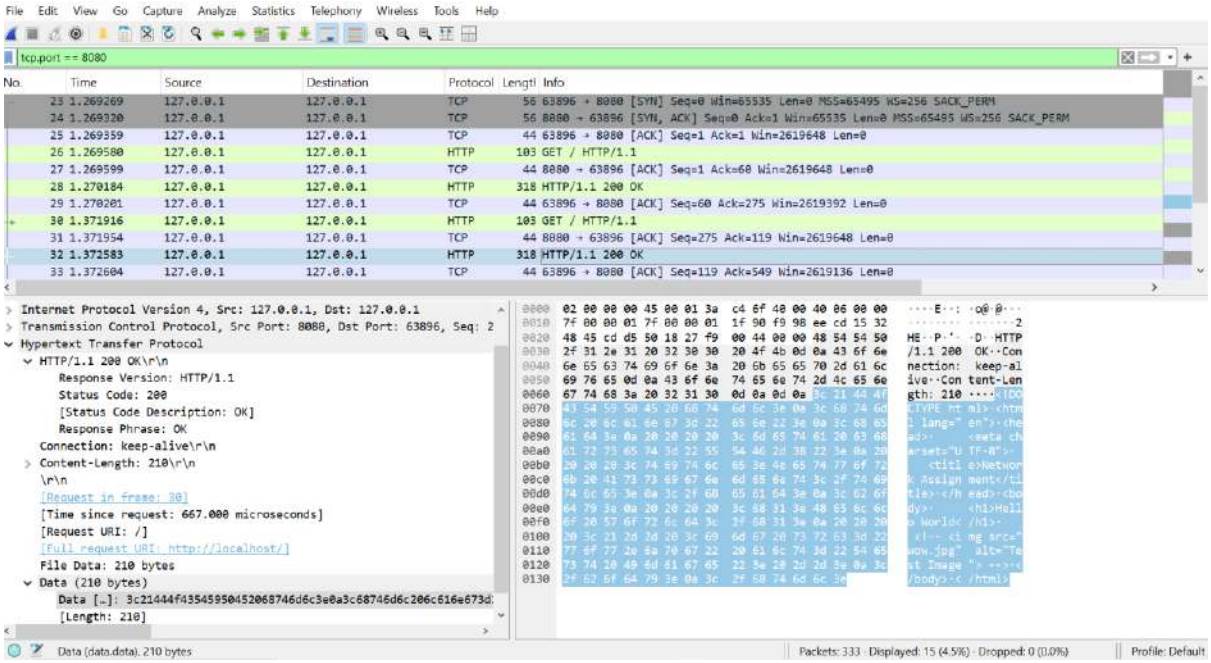
The next 3 messages go as follows: 1: server send ack on the get / request 2:server sent HTTP/1.1 message with code 200 meaning success and sent the data which contains the header hello word for example , notice connection type is still keep-alive 3: client sent ack on all the bits (274 presumably)




**The second message:** immediately after we receive and sent ack (this at 1.27) at time 1.37 we sent another http get/ request



As the connection is kept alive we don't need to connect to the server again (no syn and we didn't event get fin yet) we get immediately the ack and response as it was in the first message:




The last ack was sent at 1.37 and after a second at 2.38 we received the first fin to sever the connection and do the 4 stage handshake

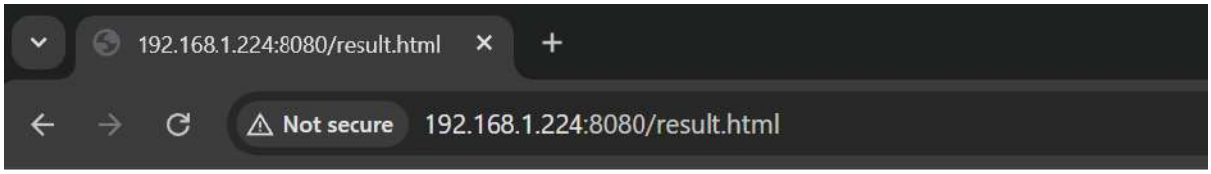
| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 23 | 1.269269 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 63896 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 24 | 1.269320 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 63896 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 25 | 1.269359 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 63896 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 26 | 1.269580 | 127.0.0.1 | 127.0.0.1 | HTTP | 103 | GET / HTTP/1.1 |
| 27 | 1.269599 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 63896 [ACK] Seq=1 Ack=60 Win=2619648 Len=0 |
| 28 | 1.270184 | 127.0.0.1 | 127.0.0.1 | HTTP | 318 | HTTP/1.1 200 OK |
| 29 | 1.270201 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 63896 → 8080 [ACK] Seq=60 Ack=275 Win=2619392 Len=0 |
| 30 | 1.371916 | 127.0.0.1 | 127.0.0.1 | HTTP | 103 | GET / HTTP/1.1 |
| 31 | 1.371954 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 63896 [ACK] Seq=275 Ack=119 Win=2619648 Len=0 |
| 32 | 1.372583 | 127.0.0.1 | 127.0.0.1 | HTTP | 318 | HTTP/1.1 200 OK |
| 33 | 1.372604 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 63896 → 8080 [ACK] Seq=119 Ack=549 Win=2619136 Len=0 |
| 60 | 2.386178 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 63896 [FIN, ACK] Seq=549 Ack=119 Win=2619648 Len=0 |
| 61 | 2.386208 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 63896 → 8080 [ACK] Seq=119 Ack=550 Win=2619136 Len=0 |
| 110 | 6.374259 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 63896 → 8080 [FIN, ACK] Seq=119 Ack=550 Win=2619136 Len=0 |
| 111 | 6.374288 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 63896 [ACK] Seq=550 Ack=120 Win=2619648 Len=0 |

All this shows that the server sent fin only after waiting a while for as the client sent the connection type as keep-alive

Now in file **double_fin** you can see where we waited to send the second message it sent the fin before we could send the second message and so we got syn again and 2 disconnects meaning 4 fins

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 33 | 1.153947 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 64393 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 34 | 1.154032 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 64393 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 35 | 1.154077 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 64393 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 36 | 1.154386 | 127.0.0.1 | 127.0.0.1 | HTTP | 103 | GET / HTTP/1.1 |
| 37 | 1.154407 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 64393 [ACK] Seq=1 Ack=60 Win=2619648 Len=0 |
| 38 | 1.155389 | 127.0.0.1 | 127.0.0.1 | HTTP | 318 | HTTP/1.1 200 OK |
| 39 | 1.155416 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 64393 → 8080 [ACK] Seq=60 Ack=275 Win=2619392 Len=0 |
| 40 | 2.165744 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 64393 [FIN, ACK] Seq=275 Ack=60 Win=2619648 Len=0 |
| 41 | 2.165767 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 64393 → 8080 [ACK] Seq=60 Ack=276 Win=2619392 Len=0 |
| 56 | 3.159318 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 64393 → 8080 [FIN, ACK] Seq=60 Ack=276 Win=2619392 Len=0 |
| 57 | 3.159340 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 64393 [ACK] Seq=276 Ack=61 Win=2619648 Len=0 |
| 58 | 3.159524 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 64394 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 59 | 3.159569 | 127.0.0.1 | 127.0.0.1 | TCP | 56 | 8080 → 64394 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM |
| 60 | 3.159595 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 64394 → 8080 [ACK] Seq=1 Ack=1 Win=2619648 Len=0 |
| 61 | 3.159785 | 127.0.0.1 | 127.0.0.1 | HTTP | 103 | GET / HTTP/1.1 |
| 62 | 3.159811 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 64394 [ACK] Seq=1 Ack=60 Win=2619648 Len=0 |
| 63 | 3.160674 | 127.0.0.1 | 127.0.0.1 | HTTP | 318 | HTTP/1.1 200 OK |
| 64 | 3.160697 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 64394 → 8080 [ACK] Seq=60 Ack=275 Win=2619392 Len=0 |
| 65 | 3.162168 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 64394 → 8080 [FIN, ACK] Seq=60 Ack=275 Win=2619392 Len=0 |
| 66 | 3.162184 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 64394 [ACK] Seq=275 Ack=61 Win=2619648 Len=0 |
| 67 | 3.162399 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 8080 → 64394 [FIN, ACK] Seq=275 Ack=61 Win=2619648 Len=0 |
| 68 | 3.162420 | 127.0.0.1 | 127.0.0.1 | TCP | 44 | 64394 → 8080 [ACK] Seq=61 Ack=276 Win=2619392 Len=0 |

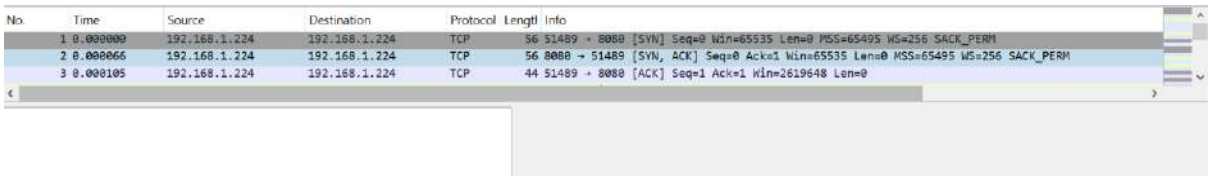
2. **Let analyze redirect.pcapng:**



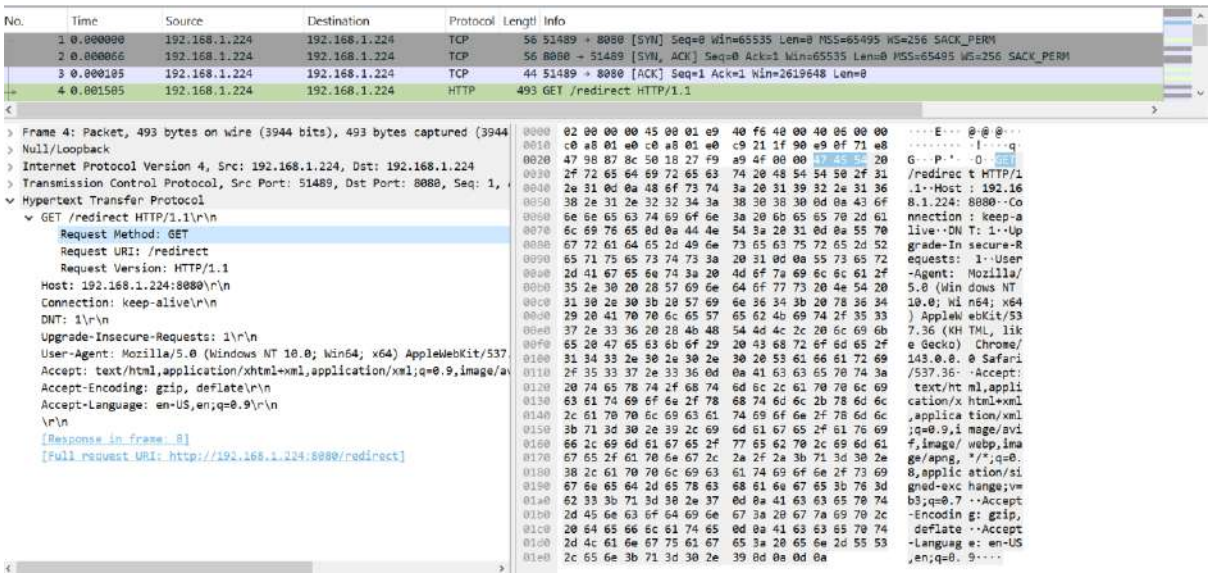

# Success!

You have been redirected to result.html

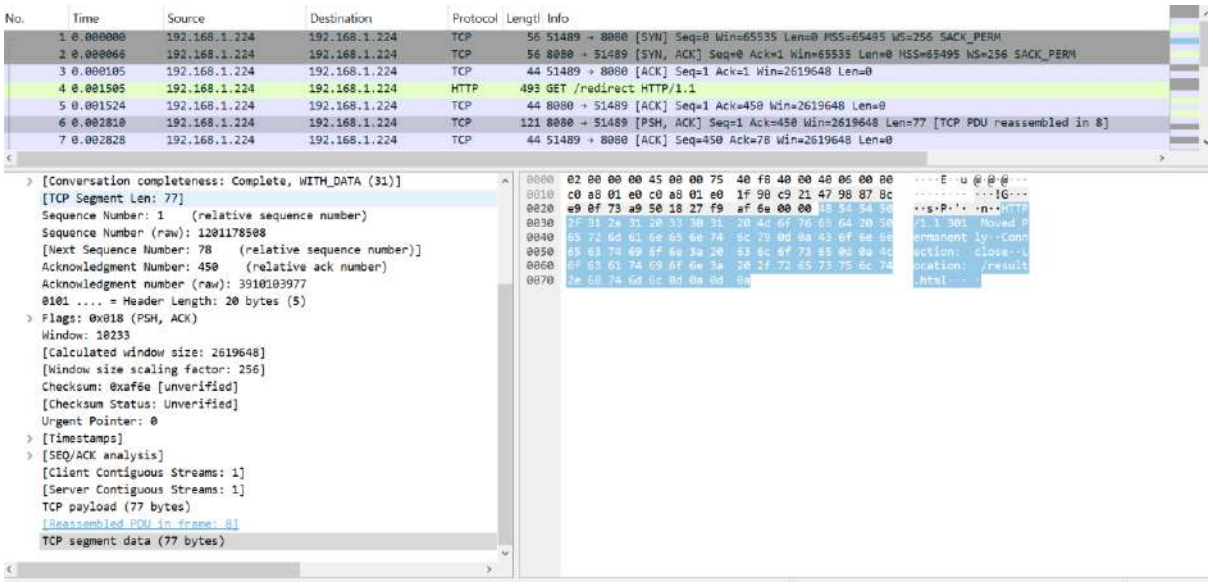
We as usual get the 3 part handshake for connection:



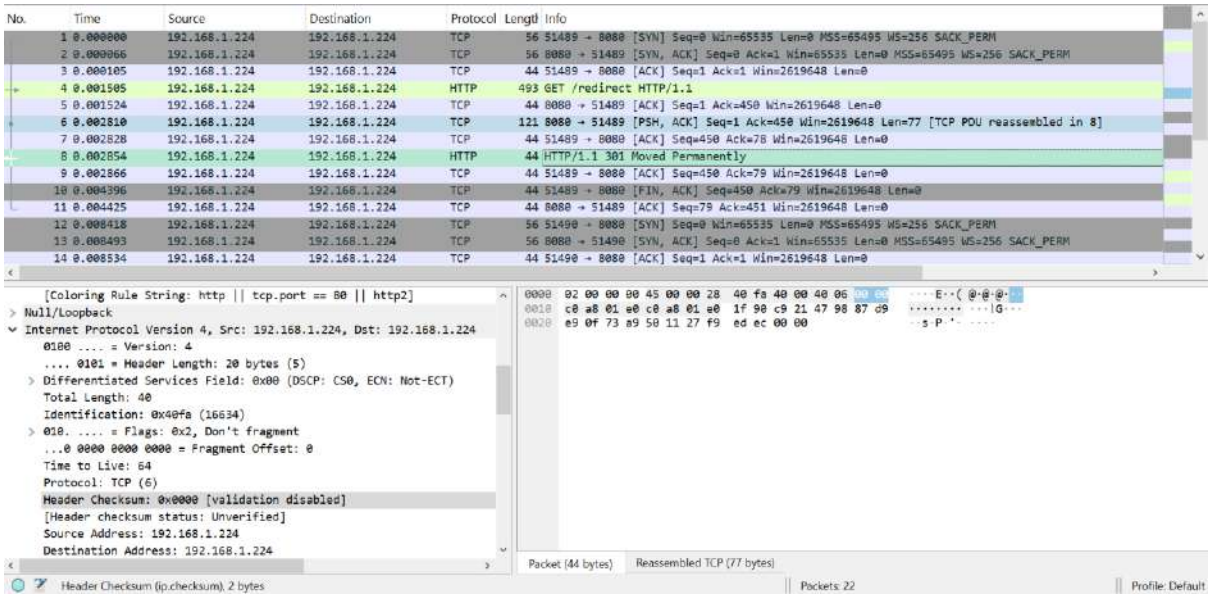

Now we send a http message get/redirect



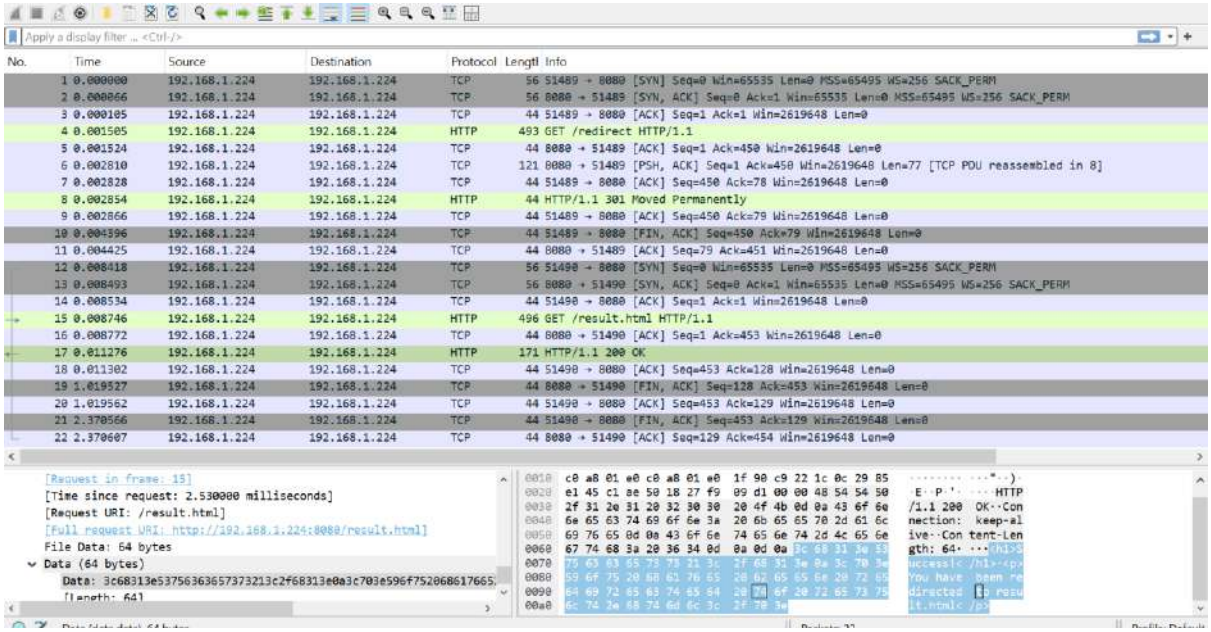
We receive an ack from the server and then the message to redirect us to /result and to close the connection, which we send ack as a response




Now we send a message that we moved permanently 301 close the connection from the client side (server side already knows to close the connection) then we open a new connection



From the new connection we send a new http get for /result we receive it from the server and after a second we close the connection




This shows the 301 redirect

3. **Now we will anlyze notfound.pcapng where we executed get/nope which doesn't exist**
   We can see that we sent the get /nope and received a 404 http code meaning the page was not found, everything else is similar to previous analysis.