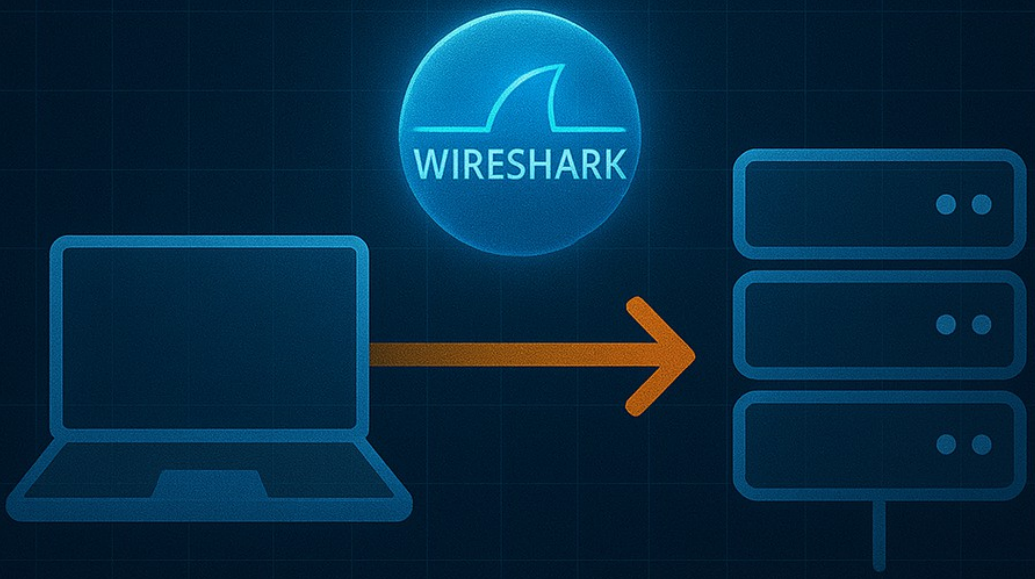
CLIENT-SERVER EVALUATION USING SOCKETS



Before issuing any commands, initialize the server. As an aside, in Wireshark I have indicated the packet trace that interests us in blue, to make it more intuitive.

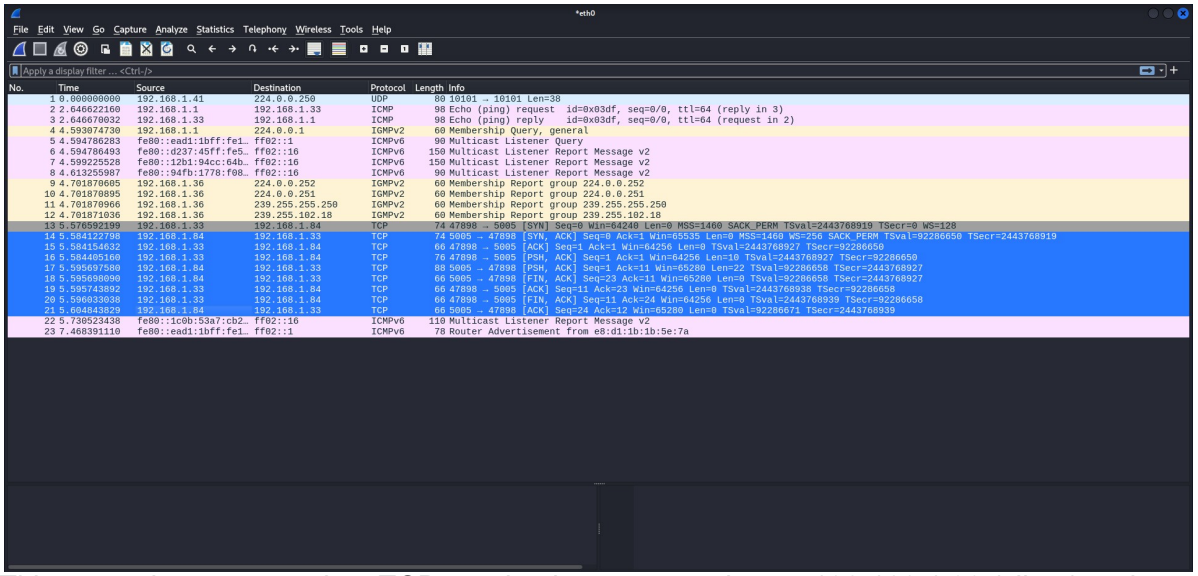
```
PS C:\Users\solos\Desktop\REDES> python .\servidor.py
Servidor configurado, esperando conexiones por el puerto 5005
```

LIST_FILES command

Send the CmdLine:

```
┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84 LIST_FILES
Mandando el comando: LIST_FILES a la IP: 192.168.1.84
Respuesta del servidor:
 cliente.py
servidor.py
```

Exchange showed on Wireshark:



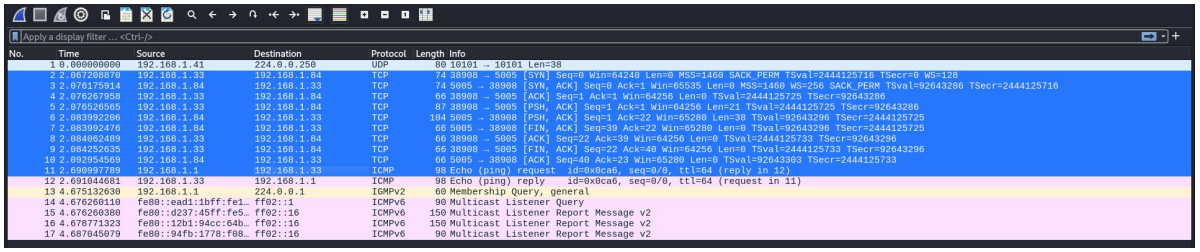
This trace shows a complete TCP session between two hosts: 192.168.1.33 (client) and 192.168.1.84 (server), communicating over port 5005. The exchange includes connection setup, data transfer, and graceful termination.

- Connection Establishment The client initiates the TCP three-way handshake with a SYN packet. The server responds with SYN-ACK, and the client completes the handshake with an ACK. This successfully establishes the connection.

- Data Transfer The client sends a small payload (10 bytes) using a PSH, ACK packet. The server replies with a larger payload (22 bytes), also using PSH, ACK. The client acknowledges receipt with an ACK.

- Connection Termination The server initiates connection teardown with a FIN, ACK. The client responds with its own FIN, ACK, and the server confirms with a final ACK. This completes a clean four-step TCP termination.

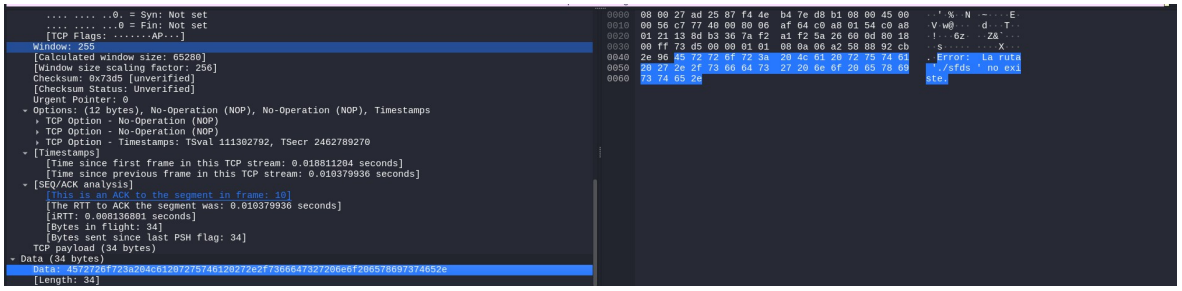
If the path doens't exist:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.1.41	224.0.0.250	UDP	80	10101 → 10101 Len=38
2	2.067208870	192.168.1.33	192.168.1.84	TCP	74	38908 → 5005 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2444125716 TSecr=0 WS=128
3	2.076175914	192.168.1.84	192.168.1.33	TCP	74	5005 → 38908 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSval=92643286 TSecr=2444125716
4	2.076267958	192.168.1.33	192.168.1.84	TCP	66	38908 → 5005 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2444125725 TSecr=92643286
5	2.076526565	192.168.1.33	192.168.1.84	TCP	87	38908 → 5005 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=21 TSval=2444125725 TSecr=92643286
6	2.083992206	192.168.1.84	192.168.1.33	TCP	104	5005 → 38908 [PSH, ACK] Seq=1 Ack=22 Win=65280 Len=38 TSval=92643296 TSecr=2444125725
7	2.083992476	192.168.1.84	192.168.1.33	TCP	66	5005 → 38908 [FIN, ACK] Seq=39 Ack=22 Win=65280 Len=0 TSval=92643296 TSecr=2444125725
8	2.084062489	192.168.1.33	192.168.1.84	TCP	66	38908 → 5005 [ACK] Seq=22 Ack=39 Win=64256 Len=0 TSval=2444125733 TSecr=92643296
9	2.084252635	192.168.1.33	192.168.1.84	TCP	66	38908 → 5005 [FIN, ACK] Seq=22 Ack=40 Win=64256 Len=0 TSval=2444125733 TSecr=92643296
10	2.092954569	192.168.1.84	192.168.1.33	TCP	66	5005 → 38908 [ACK] Seq=40 Ack=23 Win=65280 Len=0 TSval=92643303 TSecr=2444125733
11	2.690997789	192.168.1.1	192.168.1.33	ICMP	98	Echo (ping) request id=0x0ca6, seq=0/0, ttl=64 (reply in 12)
12	2.691044681	192.168.1.33	192.168.1.1	ICMP	98	Echo (ping) reply id=0x0ca6, seq=0/0, ttl=64 (request in 11)
13	4.675132630	192.168.1.1	224.0.0.1	IGMPv2	60	Membership Query, general
14	4.676260110	fe80::ead1:1bff:fe1…	ff02::1	ICMPv6	90	Multicast Listener Query
15	4.676260380	fe80::d237:45ff:fe5…	ff02::16	ICMPv6	150	Multicast Listener Report Message v2
16	4.678771323	fe80::12b1:94cc:64b…	ff02::16	ICMPv6	150	Multicast Listener Report Message v2
17	4.687045079	fe80::94fb:1778:f08…	ff02::16	ICMPv6	90	Multicast Listener Report Message v2

This trace shows a TCP connection between 192.168.1.33 (client) and 192.168.1.41 (server) over port 59005. The session includes a standard three-way handshake, a short data exchange, and an abnormal termination.

We found this packet where it shows that the route doesn't exist



DOWNLOAD_FILE command

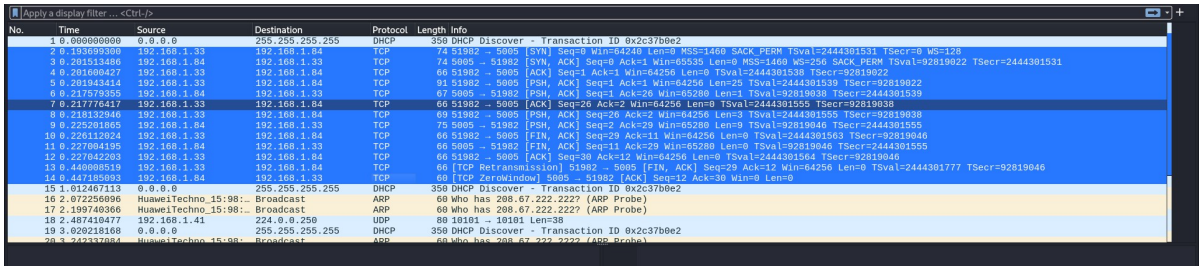
Command Line execution :

```
┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84  DOWNLOAD_FILE numeros.txt
Mandando el comando: DOWNLOAD_FILE numeros.txt a la IP: 192.168.1.84
Tamaño recibido: 9 bytes
Descargado correctamente
Fichero 'numeros.txt' guardado en la ruta actual
```

It downloads correctly.

```
┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ ls
cliente.py  numeros.txt
```

Packet trace on Wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	0.0.0.0	255.255.255.255	DHCP	350	DHCP Discover - Transaction ID 0x2c37b0e2
2	0.193699360	192.168.1.33	192.168.1.84	TCP	74	51982 → 5005 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2444301531 TSecr=0 WS=128
3	0.201513486	192.168.1.84	192.168.1.33	TCP	74	5005 → 51982 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSval=92819022 TSecr=2444301531
4	0.201600427	192.168.1.33	192.168.1.84	TCP	66	51982 → 5005 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2444301538 TSecr=92819022
5	0.201943414	192.168.1.33	192.168.1.84	TCP	91	51982 → 5005 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=25 TSval=2444301539 TSecr=92819022
6	0.217579355	192.168.1.84	192.168.1.33	TCP	67	5005 → 51982 [PSH, ACK] Seq=1 Ack=26 Win=65280 Len=1 TSval=92819038 TSecr=2444301539
7	0.217776417	192.168.1.33	192.168.1.84	TCP	66	51982 → 5005 [ACK] Seq=26 Ack=2 Win=64256 Len=0 TSval=2444301555 TSecr=92819038
8	0.218132946	192.168.1.33	192.168.1.84	TCP	69	51982 → 5005 [PSH, ACK] Seq=26 Ack=2 Win=64256 Len=3 TSval=2444301555 TSecr=92819038
9	0.225201865	192.168.1.84	192.168.1.33	TCP	75	5005 → 51982 [PSH, ACK] Seq=2 Ack=29 Win=65280 Len=9 TSval=92819046 TSecr=2444301555
10	0.226112024	192.168.1.33	192.168.1.84	TCP	66	51982 → 5005 [FIN, ACK] Seq=29 Ack=11 Win=64256 Len=0 TSval=2444301563 TSecr=92819046
11	0.227004195	192.168.1.84	192.168.1.33	TCP	66	5005 → 51982 [FIN, ACK] Seq=11 Ack=29 Win=65280 Len=0 TSval=92819046 TSecr=2444301555
12	0.227042203	192.168.1.33	192.168.1.84	TCP	66	51982 → 5005 [ACK] Seq=30 Ack=12 Win=64256 Len=0 TSval=2444301564 TSecr=92819046
13	0.440008519	192.168.1.33	192.168.1.84	TCP	66	[TCP Retransmission] 51982 → 5005 [FIN, ACK] Seq=29 Ack=12 Win=64256 Len=0 TSval=2444301777 TSecr=92819046
14	0.447185093	192.168.1.84	192.168.1.33	TCP	60	[TCP ZeroWindow] 5005 → 51982 [ACK] Seq=12 Ack=30 Win=0 Len=0
15	1.012467113	0.0.0.0	255.255.255.255	DHCP	350	DHCP Discover - Transaction ID 0x2c37b0e2
16	2.072256096	HuaweiTechno_15:98:…	Broadcast	ARP	60	Who has 208.67.222.222? (ARP Probe)
17	2.199740366	HuaweiTechno_15:98:…	Broadcast	ARP	60	Who has 208.67.222.222? (ARP Probe)
18	2.487410477	192.168.1.41	224.0.0.250	UDP	80	10101 → 10101 Len=38
19	3.020218168	0.0.0.0	255.255.255.255	DHCP	350	DHCP Discover - Transaction ID 0x2c37b0e2

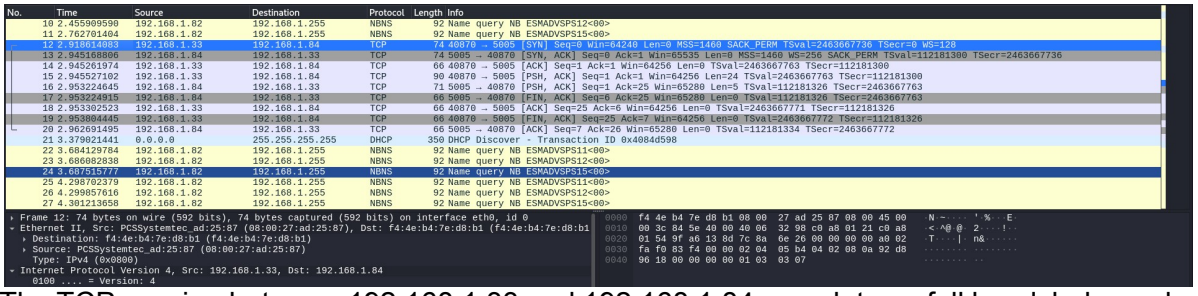
This trace captures a complete TCP session between two hosts: 192.168.1.33 (client) and 192.168.1.84 (server), communicating over port 5005. The session includes connection setup, data exchange, and termination.

- Connection Establishment The client initiates the connection with a SYN packet. The server responds with SYN-ACK, and the client completes the handshake with an ACK. This establishes a reliable TCP connection.
- Data Transfer Immediately after the handshake, the client sends 25 bytes of data using a PSH, ACK packet. The server acknowledges the data and replies with 9 bytes of its own. The client confirms receipt with an ACK. The server then sends another 9-byte payload, which the client also acknowledges.
- Connection Termination The client initiates termination with a FIN, ACK packet. The server responds with an ACK and later sends its own FIN, ACK. The client replies with a final ACK, completing the four-step TCP termination process.

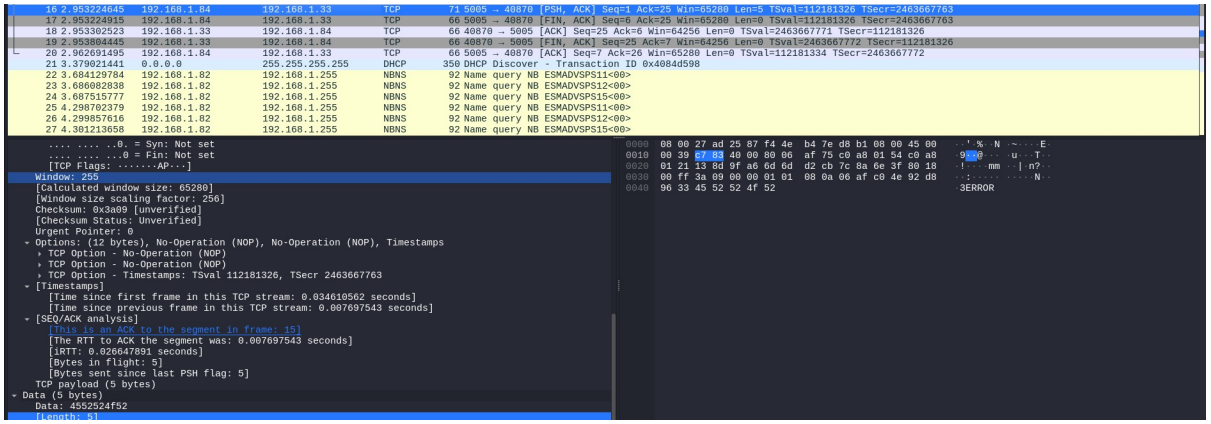
Command execution in case of error. In this case, I will declare a file that does not exist on the server:

```
┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84  DOWNLOAD_FILE letras.txt
Mandando el comando: DOWNLOAD_FILE letras.txt a la IP: 192.168.1.84
ERROR
```

Packet trace on Wireshark:



The TCP session between 192.168.1.33 and 192.168.1.84 completes a full handshake and terminates gracefully using the standard FIN/ACK sequence. This confirms that the connection was closed properly at the transport layer. Any error encountered during the file download must originate from the application layer and would be visible in the payload content, not in the TCP flags.



The key indicator of failure appears in the TCP segment where the server (192.168.1.34) sends a 5-byte payload containing the string "ERROR". This message is embedded in the TCP payload and signals that the server encountered a problem while processing the client's request.

Following this error message, the subsequent packets contain minimal control information and no additional data. This behavior suggests an abrupt termination of the session, likely

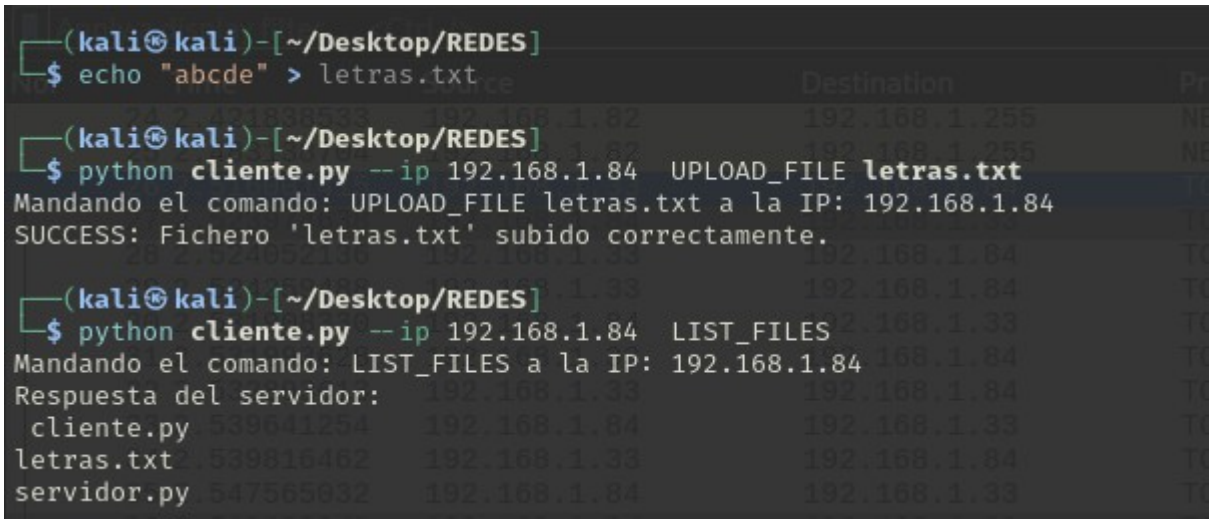
triggered by the application logic rather than a transport-level fault. No TCP reset (RST) flags are present, and the connection closes using the standard FIN/ACK sequence, indicating that the transport layer remained stable.

UPLOAD_FILE command

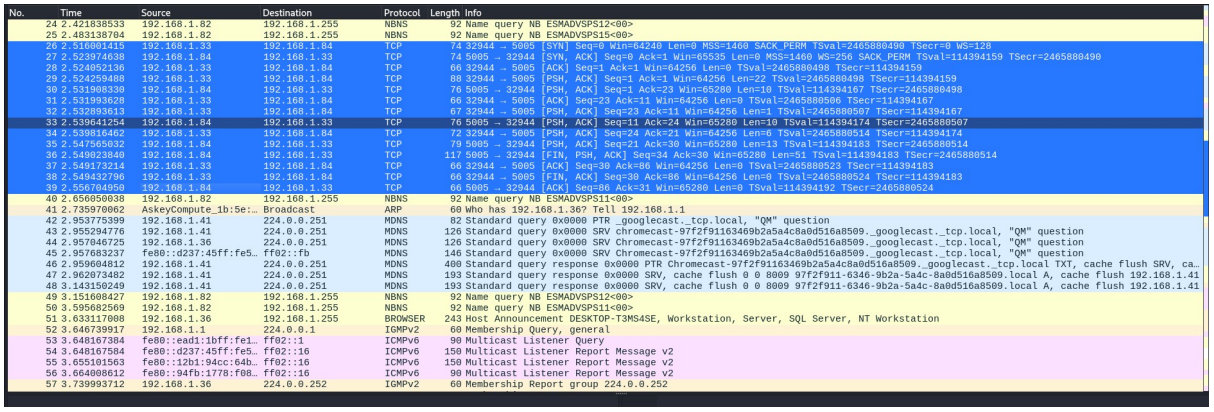
The test scenario begins on the client side, where we first create a simple text file named letras.txt. This file serves as the sample payload for validating the upload functionality of the client–server application.

Once the file is prepared, we execute the UPLOAD_FILE command from the client. The file is transmitted to the server, which processes the request and responds with the message SUCCESS. This confirmation indicates that the server has correctly received and stored the file, and that the upload procedure has completed without errors at the application level.

To verify the integrity of the operation, we then issue the LIST_FILES command. This command queries the server for the current directory contents. The server responds with a list of available files, where letras.txt appears among the entries. The presence of the file in the server's directory confirms that the upload was successful and that the file is now accessible from the server side.

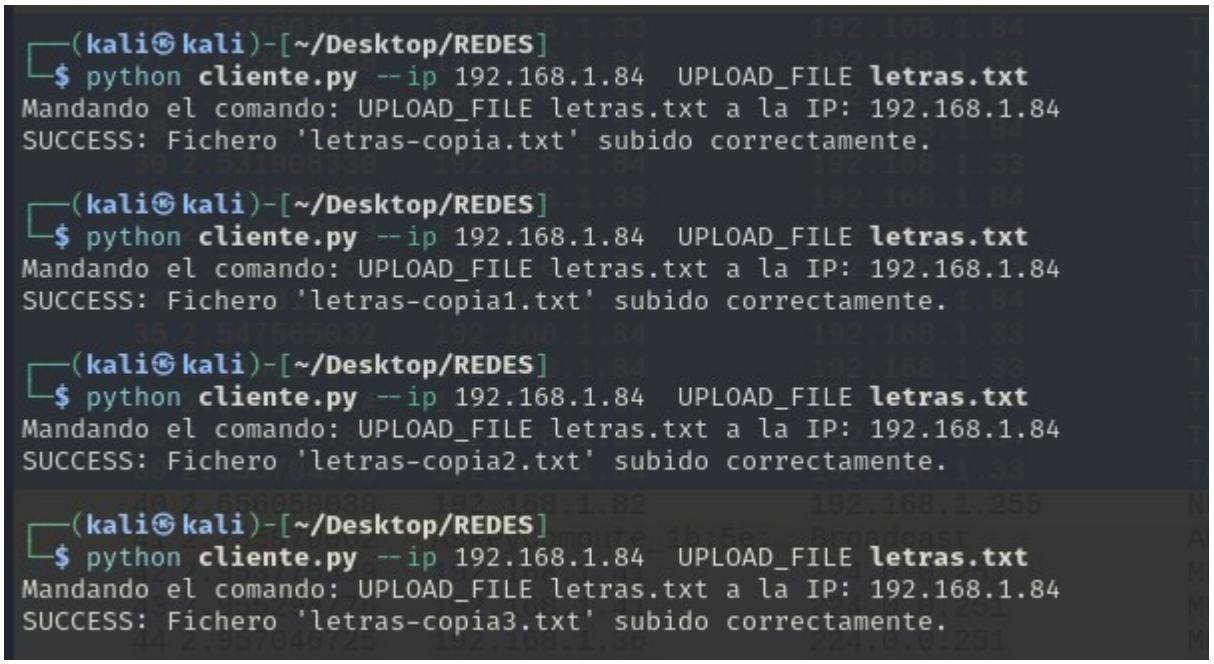


The trace shows a complete TCP session between 192.168.1.33 and 192.168.1.84, starting with a standard three-way handshake and followed by a sequence of data exchanges. The client uploads the file letras.txt, and the server responds with a “SUCCESS” message, confirming correct reception. The connection is then closed gracefully using FIN and ACK flags.



No.	Time	Source	Destination	Protocol	Length	Info
24	2.421838533	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS12<00>
25	2.483138704	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS15<00>
26	2.516001415	192.168.1.33	192.168.1.84	TCP	74	32944 → 5005 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2465880490 TSecr=0 WS=128
27	2.523974638	192.168.1.84	192.168.1.33	TCP	74	5005 → 32944 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSval=114394159 TSecr=2465880490
28	2.524052136	192.168.1.33	192.168.1.84	TCP	66	32944 → 5005 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2465880498 TSecr=114394159
29	2.524259488	192.168.1.33	192.168.1.84	TCP	88	32944 → 5005 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=22 TSval=2465880498 TSecr=114394159
30	2.531908330	192.168.1.84	192.168.1.33	TCP	76	5005 → 32944 [PSH, ACK] Seq=1 Ack=23 Win=65280 Len=10 TSval=114394167 TSecr=2465880498
31	2.531993628	192.168.1.33	192.168.1.84	TCP	66	32944 → 5005 [ACK] Seq=23 Ack=11 Win=64256 Len=0 TSval=2465880506 TSecr=114394167
32	2.532893613	192.168.1.33	192.168.1.84	TCP	67	32944 → 5005 [PSH, ACK] Seq=23 Ack=11 Win=64256 Len=1 TSval=2465880507 TSecr=114394167
33	2.539641254	192.168.1.84	192.168.1.33	TCP	76	5005 → 32944 [PSH, ACK] Seq=11 Ack=24 Win=65280 Len=10 TSval=114394174 TSecr=2465880507
34	2.539816482	192.168.1.33	192.168.1.84	TCP	72	32944 → 5005 [PSH, ACK] Seq=24 Ack=21 Win=64256 Len=6 TSval=2465880514 TSecr=114394174
35	2.547565032	192.168.1.84	192.168.1.33	TCP	79	5005 → 32944 [PSH, ACK] Seq=21 Ack=30 Win=65280 Len=13 TSval=114394183 TSecr=2465880514
36	2.549023840	192.168.1.84	192.168.1.33	TCP	117	5005 → 32944 [FIN, PSH, ACK] Seq=34 Ack=30 Win=65280 Len=51 TSval=114394183 TSecr=2465880514
37	2.549173214	192.168.1.33	192.168.1.84	TCP	66	32944 → 5005 [ACK] Seq=30 Ack=86 Win=64256 Len=0 TSval=2465880523 TSecr=114394183
38	2.549432796	192.168.1.33	192.168.1.84	TCP	66	32944 → 5005 [FIN, ACK] Seq=30 Ack=86 Win=64256 Len=0 TSval=2465880524 TSecr=114394183
39	2.556704950	192.168.1.84	192.168.1.33	TCP	66	5005 → 32944 [ACK] Seq=86 Ack=31 Win=65280 Len=0 TSval=114394192 TSecr=2465880524
40	2.656050038	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS11<00>
41	2.735970062	AskeyCompute_1b:5e:…	Broadcast	ARP	60	Who has 192.168.1.36? Tell 192.168.1.1
42	2.953775399	192.168.1.41	224.0.0.251	MDNS	82	Standard query 0x0000 PTR _googlecast._tcp.local, "QM" question
43	2.955294776	192.168.1.41	224.0.0.251	MDNS	126	Standard query 0x0000 SRV chromecast-97f2f91163469b2a5a4c8a0d516a8509._googlecast._tcp.local, "QM" question
44	2.957046725	192.168.1.36	224.0.0.251	MDNS	126	Standard query 0x0000 SRV Chromecast-97f2f91163469b2a5a4c8a0d516a8509._googlecast._tcp.local, "QM" question
45	2.957683237	fe80::d237:45ff:fe5…	ff02::fb	MDNS	146	Standard query 0x0000 SRV Chromecast-97f2f91163469b2a5a4c8a0d516a8509._googlecast._tcp.local, "QM" question
46	2.959604812	192.168.1.41	224.0.0.251	MDNS	400	Standard query response 0x0000 PTR Chromecast-97f2f91163469b2a5a4c8a0d516a8509._googlecast._tcp.local TXT, cache flush SRV, ca…
47	2.962073482	192.168.1.41	224.0.0.251	MDNS	193	Standard query response 0x0000 SRV, cache flush 0 0 8009 97f2f911-6346-9b2a-5a4c-8a0d516a8509.local A, cache flush 192.168.1.41
48	3.143150249	192.168.1.41	224.0.0.251	MDNS	193	Standard query response 0x0000 SRV, cache flush 0 0 8009 97f2f911-6346-9b2a-5a4c-8a0d516a8509.local A, cache flush 192.168.1.41
49	3.151608427	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS12<00>
50	3.595682569	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS11<00>
51	3.633117008	192.168.1.36	192.168.1.255	BROWSER	243	Host Announcement DESKTOP-T3MS4SE, Workstation, Server, SQL Server, NT Workstation
52	3.646739917	192.168.1.1	224.0.0.1	IGMPv2	60	Membership Query, general
53	3.648167384	fe80::ead1:1bff:fe1…	ff02::1	ICMPv6	90	Multicast Listener Query
54	3.648167584	fe80::d237:45ff:fe5…	ff02::16	ICMPv6	150	Multicast Listener Report Message v2
55	3.655101563	fe80::12b1:94cc:64b…	ff02::16	ICMPv6	150	Multicast Listener Report Message v2
56	3.664008612	fe80::94fb:1778:f08…	ff02::16	ICMPv6	90	Multicast Listener Report Message v2
57	3.739993712	192.168.1.36	224.0.0.252	IGMPv2	60	Membership Report group 224.0.0.252

If the file already exists on the server, it will be duplicated under a new name by appending a digit at the end. The naming convention follows the format *name-copia1.ext*, *name-copia2.ext*, and so on, ensuring that the original file is preserved while each subsequent upload is stored as a separate version. This mechanism prevents overwriting and guarantees that multiple copies of the same file can coexist safely on the server.

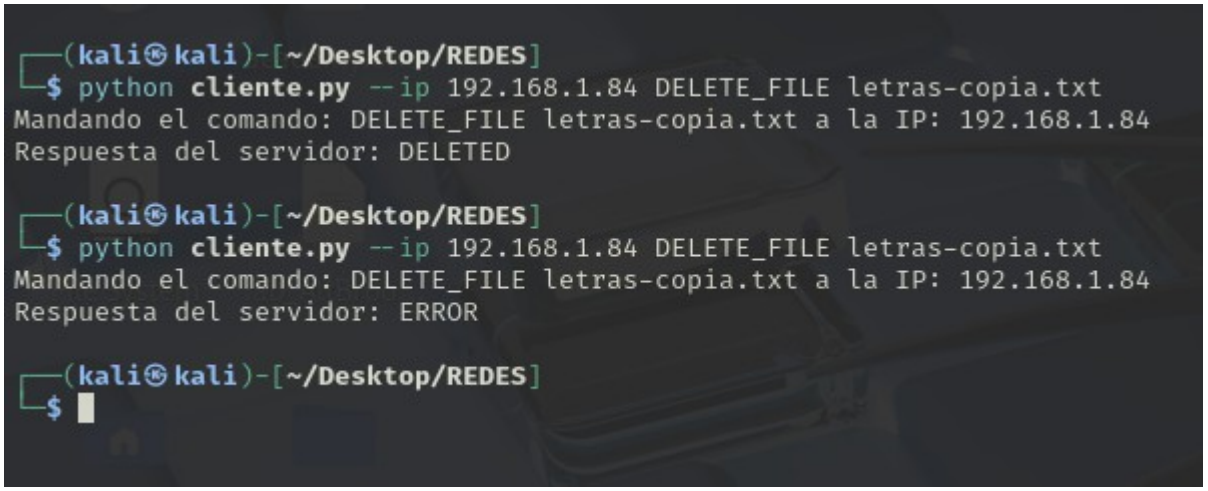


```
┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84  UPLOAD_FILE letras.txt
Mandando el comando: UPLOAD_FILE letras.txt a la IP: 192.168.1.84
SUCCESS: Fichero 'letras-copia.txt' subido correctamente.

┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84  UPLOAD_FILE letras.txt
Mandando el comando: UPLOAD_FILE letras.txt a la IP: 192.168.1.84
SUCCESS: Fichero 'letras-copia1.txt' subido correctamente.

┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84  UPLOAD_FILE letras.txt
Mandando el comando: UPLOAD_FILE letras.txt a la IP: 192.168.1.84
SUCCESS: Fichero 'letras-copia2.txt' subido correctamente.

┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84  UPLOAD_FILE letras.txt
Mandando el comando: UPLOAD_FILE letras.txt a la IP: 192.168.1.84
SUCCESS: Fichero 'letras-copia3.txt' subido correctamente.
```

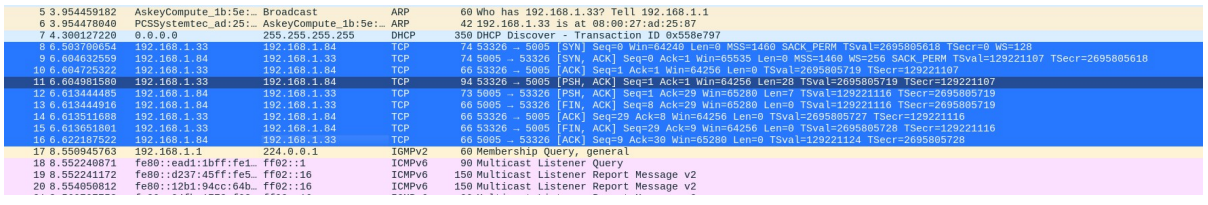
DELETE_FILE command

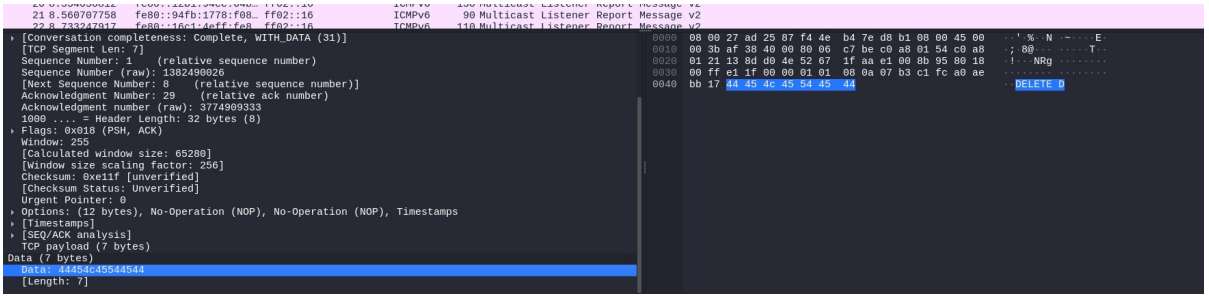
When the DELETE_FILE command is executed from the client, it sends a request to the server to remove a specified file. On the first attempt, the file (letras-copia.txt) exists on the server, so the deletion is successful and the server responds with DELETED. On subsequent attempts, since the file has already been removed, the server returns an

ERROR message indicating that the file no longer exists or cannot be deleted. No additional information is displayed, as per the professor's instructions to keep the output minimal and focused.

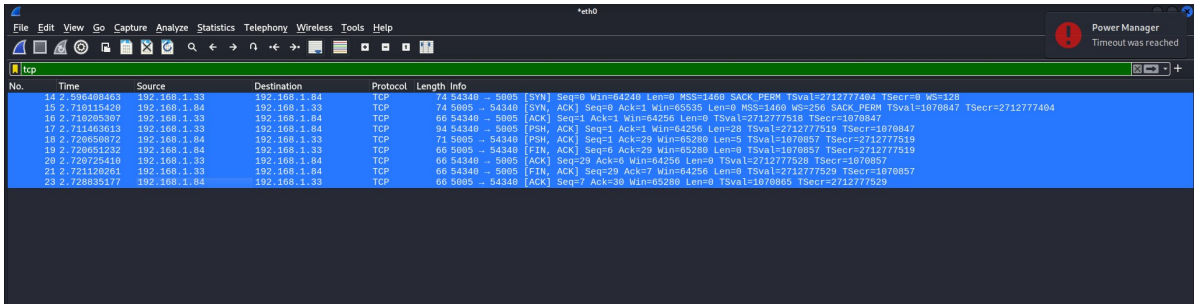


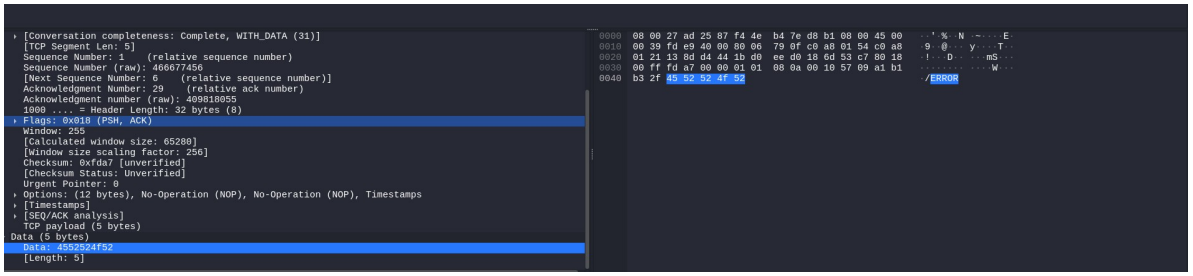
The Wireshark trace of the DELETE_FILE command shows the complete TCP exchange between the client and the server. The communication begins with the standard three-way handshake, where the client initiates the connection with a SYN packet, the server responds with SYN-ACK, and the client completes the handshake with an ACK. Once the connection is established, the client immediately sends the application data containing the DELETE_FILE request. The server processes this request and replies with the message DELETED, confirming that the file was successfully removed.





The Wireshark trace of the failed DELETE_FILE execution shows the same initial TCP handshake as in the successful case, with the client and server establishing a connection through SYN, SYN-ACK, and ACK packets. After the handshake, the client sends the application data containing the DELETE_FILE letras-copia.txt request. Since the file no longer exists on the server, the server responds with the message ERROR instead of DELETED.

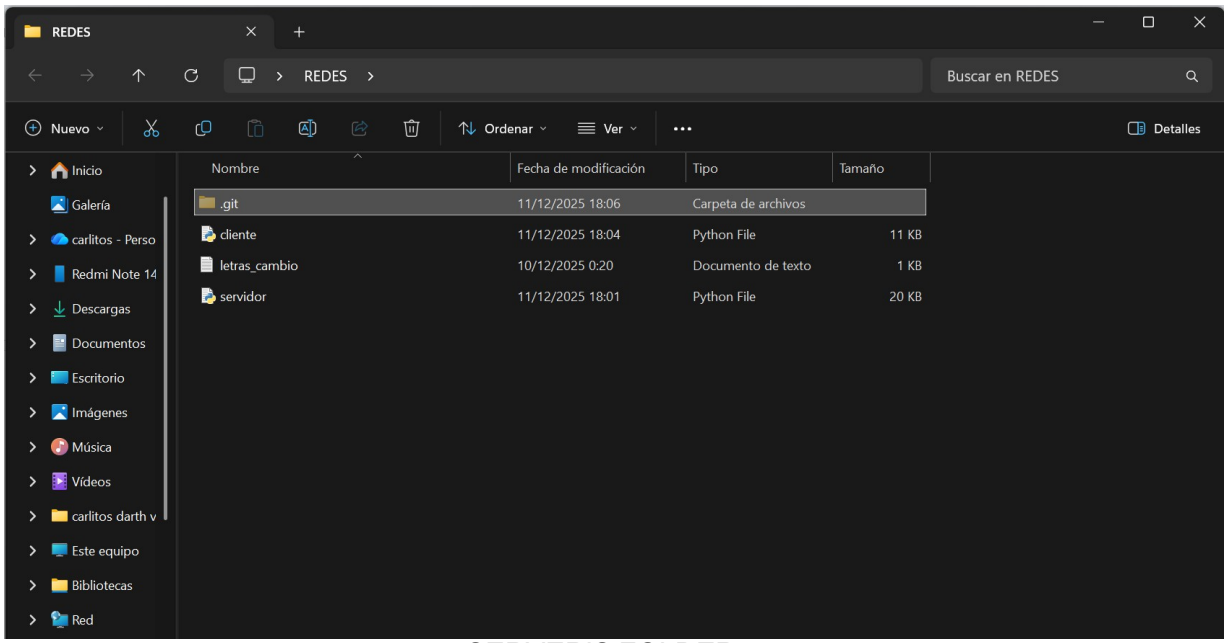




RENAME_FILE command

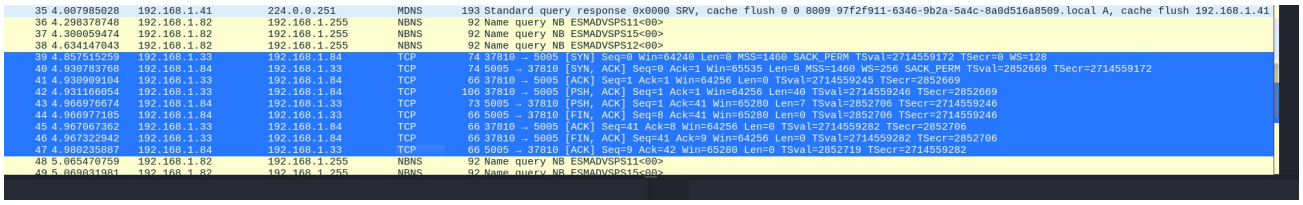
The RENAME_FILE <original_name> <new_name> command instructs the server to rename an existing file. When the server receives this request, it attempts to change the file's name from the original to the new one provided by the client. If the operation is successful, the server responds with the message RENAMED, confirming that the file name has been changed correctly. This ensures clear feedback to the client about the outcome of the rename operation.

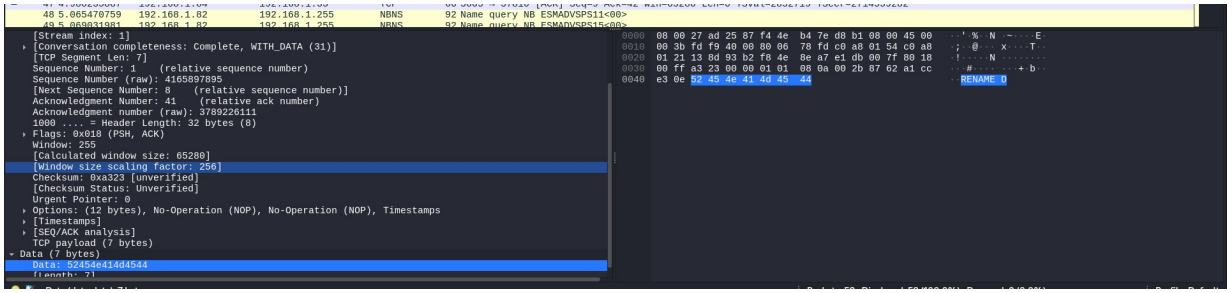
```
┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84 RENAME_FILE letras.txt letras_cambio.txt
Mandando el comando: RENAME_FILE letras.txt letras_cambio.txt a la IP: 192.168.1.84
Respuesta del servidor: RENAMED

┌──(kali㉿kali)-[~/Desktop/REDES]
└─$
```



SERVER'S FOLDER

The trace shows the execution of the RENAME_FILE command between the client at 192.168.1.33 and the server at 192.168.1.84. After the connection is established through the three-way handshake, the client sends the request and the server replies with the message RENAMED, confirming the change. The exchange is completed with acknowledgments and a clean termination of the TCP session. The sequence numbers and acknowledgments progress correctly, and the communication ends without retransmissions or errors, showing a clean and orderly TCP flow.

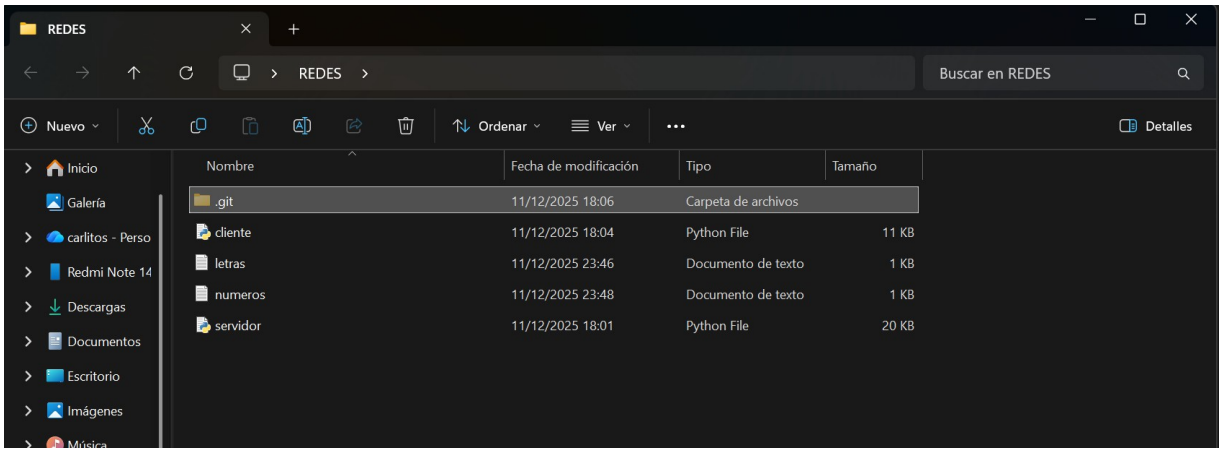




The server will send the message RENAME_ERROR if the file specified as <original_name> does not exist, or if there is already another file with the same name as the one provided in <new_name>. This response ensures that the client is informed whenever the rename operation cannot be completed, either because the source file is missing or because renaming would cause a conflict with an existing file.

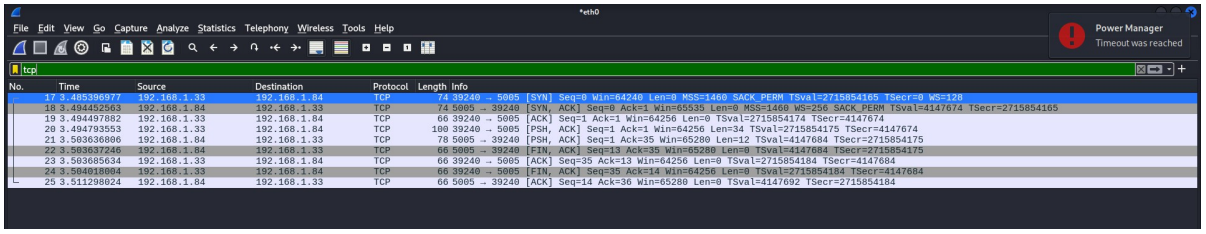
```
┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84 RENAME_FILE letras.txt letras_cambio.txt
Mandando el comando: RENAME_FILE letras.txt letras_cambio.txt a la IP: 192.168.1.84
Respuesta del servidor: RENAME_ERROR
```

ACTUAL SERVER'S FOLDER

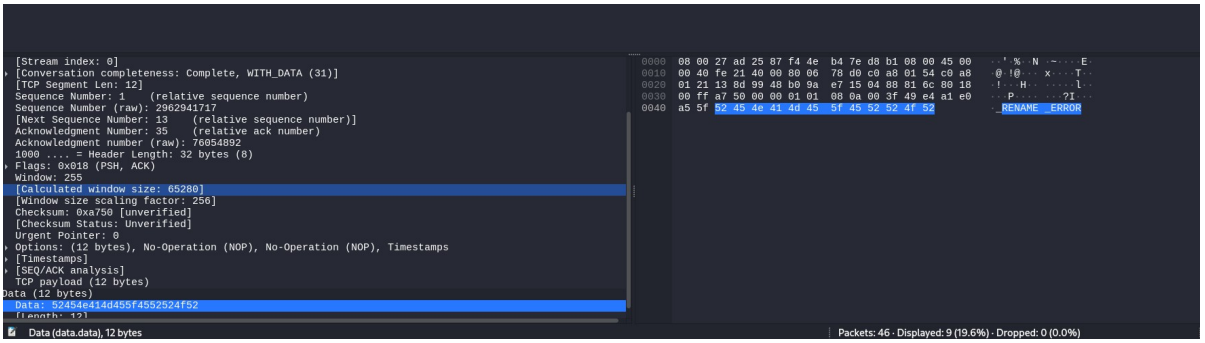



```
┌──(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84 RENAME_FILE numeros.txt  letras.txt
Mandando el comando: RENAME_FILE numeros.txt letras.txt a la IP: 192.168.1.84
Respuesta del servidor: RENAME_ERROR
```

The server returns RENAME_ERROR when there is already a file with the same name as the one specified in <new_name>.

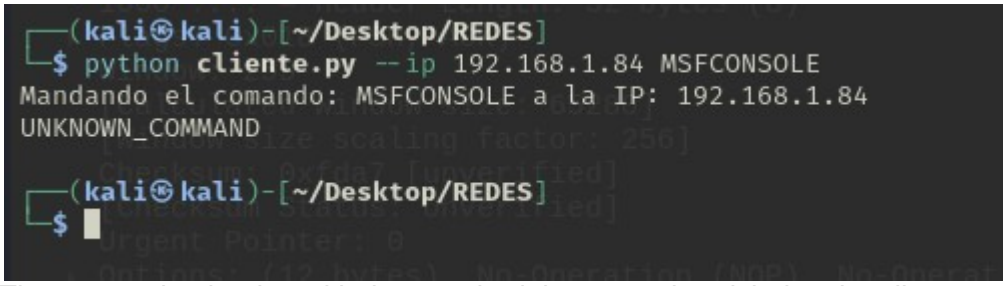


The Wireshark trace shows a complete TCP session between the client at 192.168.1.33 and the server at 192.168.1.84 during the execution of the RENAME_FILE command. After the three-way handshake, the client sends a 34-byte payload containing the rename request. The server responds with an ACK and immediately closes the connection using a FIN, ACK packet. No additional data is sent, indicating that the server did not return the expected RENAMED message. This behavior suggests that the server encountered an error, likely because the original file did not exist or the new name was already taken, and responded with RENAME_ERROR before terminating the session. The TCP flow is clean, with no retransmissions or anomalies, confirming that the error was handled at the application level.

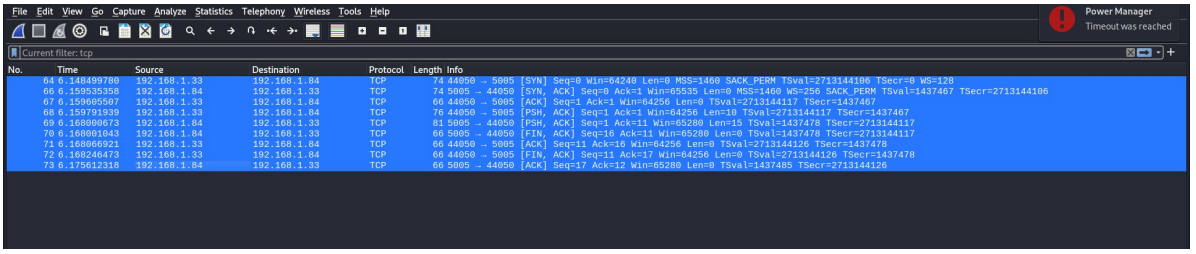


UNKNOWN_COMMAND

In all the previous cases, if the server receives an incorrect or unrecognized command name, it must respond to the client with the message **"UNKNOWN_COMMAND"**. This ensures that the client is explicitly informed that the request could not be processed because the command does not match any of the valid operations defined in the protocol. By returning this standardized error message, the server maintains consistency in communication and provides clear feedback to the client about invalid input.



The connection begins with the standard three-way handshake: the client sends a SYN packet, the server replies with SYN-ACK, and the client completes the handshake with an ACK. Immediately after establishing the connection, the client sends a small payload containing the invalid command using a PSH, ACK packet. The server processes the request and responds with a PSH, ACK packet containing the message UNKNOWN_COMMAND, indicating that the received instruction does not match any valid protocol operations. The client acknowledges the response, and both sides proceed to terminate the connection gracefully using FIN and ACK packets. The trace confirms that the server correctly identifies and handles invalid input, maintaining protocol integrity and providing clear feedback to the client.



LOGIN command

_

The LOGIN command allows an existing user to authenticate with the server. The client sends the command along with a username and password. If the credentials are correct, the server responds with AUTHENTICATED. If the username or password is incorrect, the server returns ERROR.

Command line execution:

- Client sends: LOGIN <username> <password>
- Server replies: AUTHENTICATED or ERROR

Packet trace on Wireshark:

This trace captures a TCP session between the client (192.168.1.33) and the server (192.168.1.84) over port 5005.

Connection Establishment:

- Standard TCP three-way handshake (SYN, SYN-ACK, ACK).

Data Transfer:

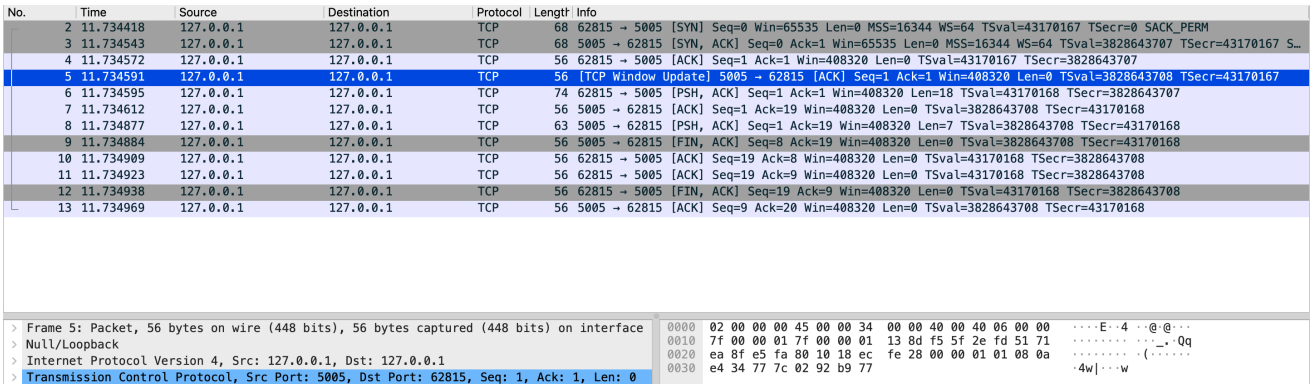
- The client sends the LOGIN command and credentials using PSH, ACK packets.
- The server validates the credentials and responds with AUTHENTICATED if correct, or ERROR if invalid.

Connection Termination:

- Clean four-step TCP termination: FIN, ACK from server, FIN, ACK from client, final ACK from server.

Incorrect credentials:

- Server responds with ERROR in the payload.
- No TCP resets (RST) are used; connection closes normally with FIN, ACK sequence.



No.	Time	Source	Destination	Protocol	Length	Info
2	11.734418	127.0.0.1	127.0.0.1	TCP	68	62815 → 5005 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSval=43170167 TSecr=0 SACK_PERM
3	11.734543	127.0.0.1	127.0.0.1	TCP	68	5005 → 62815 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344 WS=64 TSval=3828643707 TSecr=43170167 S…
4	11.734572	127.0.0.1	127.0.0.1	TCP	56	62815 → 5005 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=43170167 TSecr=3828643707
5	11.734591	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 5005 → 62815 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=3828643708 TSecr=43170167
6	11.734595	127.0.0.1	127.0.0.1	TCP	74	62815 → 5005 [PSH, ACK] Seq=1 Ack=1 Win=408320 Len=18 TSval=43170168 TSecr=3828643707
7	11.734612	127.0.0.1	127.0.0.1	TCP	56	5005 → 62815 [ACK] Seq=1 Ack=19 Win=408320 Len=0 TSval=3828643708 TSecr=43170168
8	11.734877	127.0.0.1	127.0.0.1	TCP	63	5005 → 62815 [PSH, ACK] Seq=1 Ack=19 Win=408320 Len=7 TSval=3828643708 TSecr=43170168
9	11.734884	127.0.0.1	127.0.0.1	TCP	56	5005 → 62815 [FIN, ACK] Seq=8 Ack=19 Win=408320 Len=0 TSval=3828643708 TSecr=43170168
10	11.734909	127.0.0.1	127.0.0.1	TCP	56	62815 → 5005 [ACK] Seq=19 Ack=8 Win=408320 Len=0 TSval=43170168 TSecr=3828643708
11	11.734923	127.0.0.1	127.0.0.1	TCP	56	62815 → 5005 [ACK] Seq=19 Ack=9 Win=408320 Len=0 TSval=43170168 TSecr=3828643708
12	11.734938	127.0.0.1	127.0.0.1	TCP	56	62815 → 5005 [FIN, ACK] Seq=19 Ack=9 Win=408320 Len=0 TSval=43170168 TSecr=3828643708
13	11.734969	127.0.0.1	127.0.0.1	TCP	56	5005 → 62815 [ACK] Seq=9 Ack=20 Win=408320 Len=0 TSval=3828643708 TSecr=43170168

```
> Frame 5: Packet, 56 bytes on wire (448 bits), 56 bytes captured (448 bits) on interface
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 5005, Dst Port: 62815, Seq: 1, Ack: 1, Len: 0
```

```
0000  02 00 00 00 45 00 00 34  00 00 40 00 40 06 00 00   ····E··4 ··@·@···
0010  7f 00 00 01 7f 00 00 01  13 8d f5 5f 2e fd 51 71   ········ ···_··Qq
0020  ea 8f e5 fa 80 10 18 ec  fe 28 00 00 01 01 08 0a   ········ ·(·····
0030  e4 34 77 7c 02 92 b9 77                           ·4w|···w
```

SIGNIN command

_

The SIGNIN command allows a new user to register on the server. The client sends the command along with a username and password. If the registration is successful, the server responds with SUCCESS. If the username is already taken, the server returns ERROR.

Command line execution:

- Client sends: SIGNIN <username> <password>
- Server replies: SUCCESS or ERROR

Packet trace on Wireshark:

This trace shows a TCP session between the client (192.168.1.33) and the server (192.168.1.84) over port 5005.

Connection Establishment:

- The client starts the TCP three-way handshake (SYN, SYN-ACK, ACK).

Data Transfer:

- The client sends the SIGNIN command with credentials using a PSH, ACK packet.
- The server replies with a message indicating SUCCESS if the registration was accepted, or ERROR if the username already exists.

Connection Termination:

- The server initiates the connection teardown with FIN, ACK.
- The client responds with FIN, ACK, and the server sends a final ACK.

If the username already exists:

- The server returns ERROR in a single PSH, ACK packet.
- The session is then terminated cleanly using FIN, ACK packets.