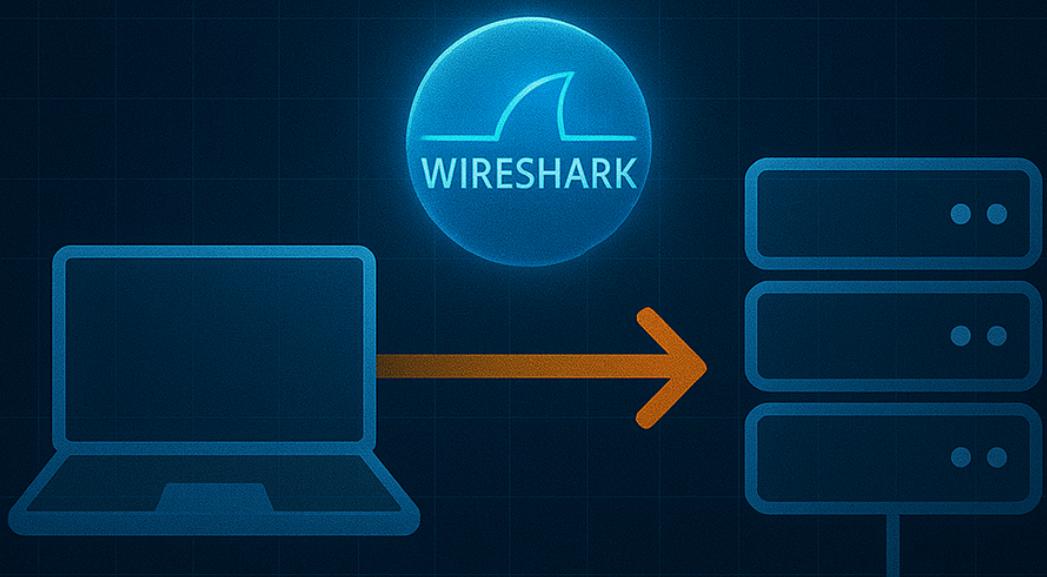


CLIENT-SERVER EVALUATION USING SOCKETS



LIST_FILES Command.....	3
DOWNLOAD_FILE command.....	5
UPLOAD_FILE command.....	7
DELETE_FILE COMMAND.....	9
RENAME_FILE COMMAND.....	11
UNKNOWN_COMMAND.....	14

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.

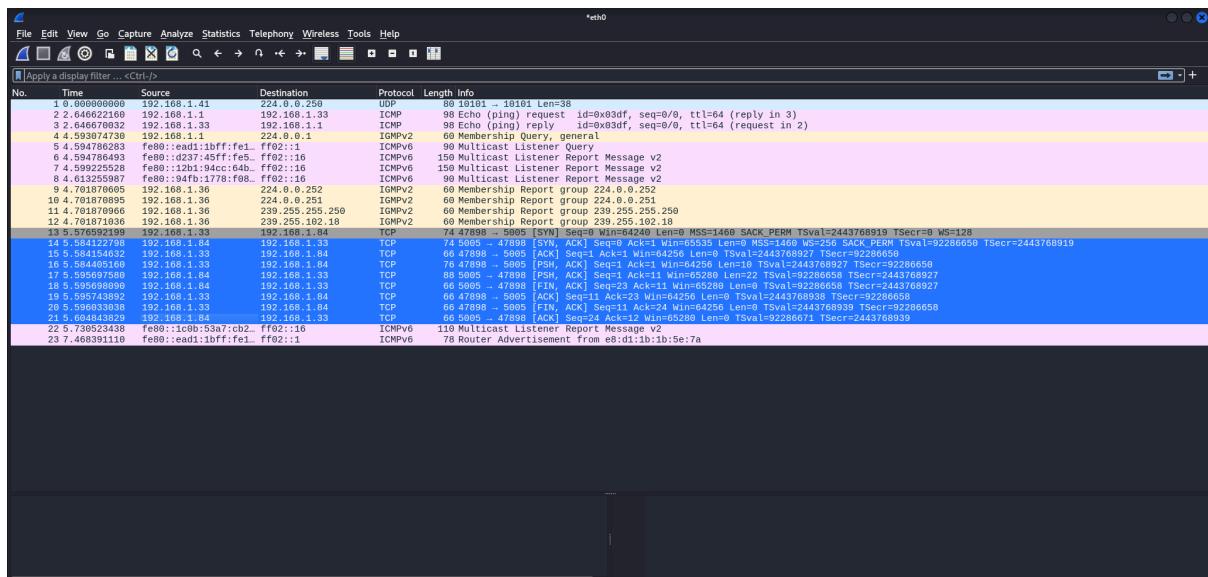
```
PS C:\Users\solos\Desktop\REDES> python .\servidor.py
Servidor configurado, esperando conexiones por el puerto 5005
```

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: 192.168.1.1 192.168.1.33
cliente.py 646670032 192.168.1.33 192.168.1.1
servidor.py 593074730 192.168.1.1 224.0.0.1
```

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 doesn't exist:

```
(kali㉿kali)-[~/Desktop/REDES]
└─$ python cliente.py --ip 192.168.1.84 LIST_FILES ./fgjdkfgh
Mandando el comando: LIST_FILES(./fgjdkfgh a la IP: 192.168.1.84
Respuesta del servidor:
Error: La ruta './fgjdkfgh' no existe. offset: 0
```

No.	Time	Source	Destination	Protocol	Length Info
1	0.0000000000	192.168.1.41	224.0.0.250	UDP	80 10101 - 10101 Len=38
2	2.867208876	192.168.1.33	192.168.1.84	TCP	74 38908 - 5005 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM Tsvl=2444125716 Tsecr=0 WS=128
3	2.876171900	192.168.1.33	192.168.1.84	TCP	64 5005 - 38908 [ACK] Seq=1 Ack=1 Win=64240 Len=0 Tsvl=2444125725 Tsecr=2444125726
4	2.876175958	192.168.1.33	192.168.1.84	TCP	87 38908 - 5005 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=0 Tsvl=2444125725 Tsecr=2444125726
5	2.876526565	192.168.1.33	192.168.1.84	TCP	104 5005 - 38908 [PSH, ACK] Seq=1 Ack=22 Win=65286 Len=38 Tsvl=2444125726 Tsecr=2444125725
6	2.883992265	192.168.1.84	192.168.1.33	TCP	66 5005 - 38908 [FIN, ACK] Seq=23 Ack=22 Win=65286 Len=0 Tsvl=2444125726 Tsecr=2444125725
7	2.883992476	192.168.1.84	192.168.1.33	TCP	66 5005 - 38908 [FIN, ACK] Seq=23 Ack=22 Win=65286 Len=0 Tsvl=2444125726 Tsecr=2444125725
8	2.884252635	192.168.1.33	192.168.1.84	TCP	66 38908 - 5005 [FIN, ACK] Seq=22 Ack=23 Win=64256 Len=0 Tsvl=2444125733 Tsecr=2444125726
9	2.884252635	192.168.1.33	192.168.1.84	TCP	66 38908 - 5005 [FIN, ACK] Seq=22 Ack=23 Win=64256 Len=0 Tsvl=2444125733 Tsecr=2444125726
10	2.892954569	192.168.1.84	192.168.1.33	TCP	66 5005 - 38908 [ACK] Seq=4 Ack=23 Win=65286 Len=0 Tsvl=2444125733 Tsecr=2444125733
11	2.896907788	192.168.1.33	192.168.1.33	TCP	98 Err? (ping) request 0x9eac6, seq=0/0, ttl=64 (reply 1/12)
12	2.896907788	192.168.1.33	224.0.0.1	ICMP	98 Echo (ping) reply 0x9eac6, seq=0/0, ttl=64 (request in 11)
13	4.675132638	192.168.1.33	224.0.0.1	ICMPv2	68 Membership Query, general
14	4.675261110	fe80::lead1:1bff:fe1...ff02::1	ff02::1	ICMPv6	98 Multicast Listener Query
15	4.676269380	fe80::d237:45ff:fe5...ff02::16	ff02::16	ICMPv6	158 Multicast Listener Report Message v2
16	4.678771323	fe80::12b1:94cc:64b...ff02::16	ff02::16	ICMPv6	158 Multicast Listener Report Message v2
17	4.687845079	fe80::194fb:1778:f08...ff02::16	ff02::16	ICMPv6	98 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

No.	Time	Source	Destination	Protocol	Length Info
0000	08 00 27 ad 25 87	F4 4e b4 7e d8 b1 08 00 45 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	%. N ~ . E
0001	08 00 56 c7 77 40 00 80 00	af 64 c0 a8 01 54 c0 a8	V w b . d . T .		
0020	01 21 13 8d b3 36 7a f2	a1 f2 5a 26 60 00 80 18	! . 6z . Z8 .		
0021	00 ff 73 05 09 00 01 01	08 0a 06 a2 58 88 92 cb	s		
0040	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	Error: La ruta '/sfds' no existe.
0050	00 27 2c 2f 73 66 64 73	27 29 0e f6 26 65 78 00	ste.		
0060	73 74 65 26				

```
.... .... ..0. = Syn: Not set
.... .... ..0. = Fin: Not set
[TCP Flags: ..AP...]
Window scale: 1
[Calculated window size: 65280]
[window size scaling factor: 256]
Checksum: 0x73d5 [unverified]
[Checksum Status: Unverified]
[Initial Payload Length: 0]
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
, TCP Option - No-Operation (NOP)
, TCP Option - No-Operation (NOP)
, TCP Option - TimeStamps: Tsvl 111302792, Tsecr 2462789270
[Timestamps]
[Time since first frame in this TCP stream: 0.018811204 seconds]
[Time since previous frame in this TCP stream: 0.010379936 seconds]
[SEQ/ACK analysis]
    [RTT to ACK the segment in frame: 10]
    [The RTT to ACK the segment was: 0.010379936 seconds]
    [iRTT: 0.008136861 seconds]
    [Bytes in flight: 34]
    [Time since last PSH flag: 34]
    [TCP payload (34 bytes)
Data (34 bytes)
Data: 4572726723a284c6126725740128272e7f3066473272066ef206578697374652e
[Length: 34]
```

DOWNLOAD_FILE command

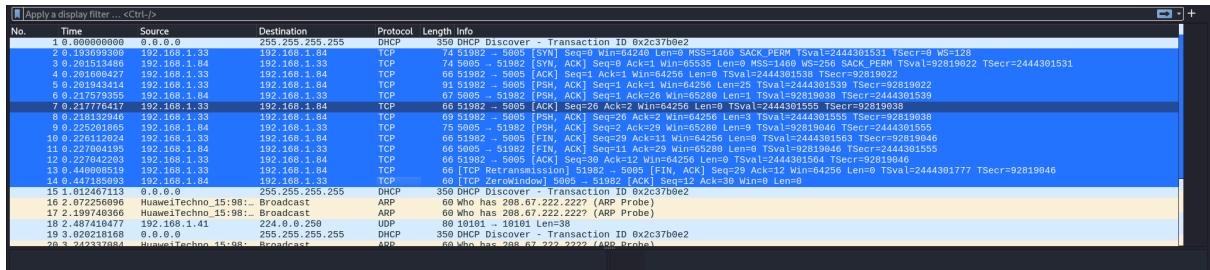
Command Line execution :

```
(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:



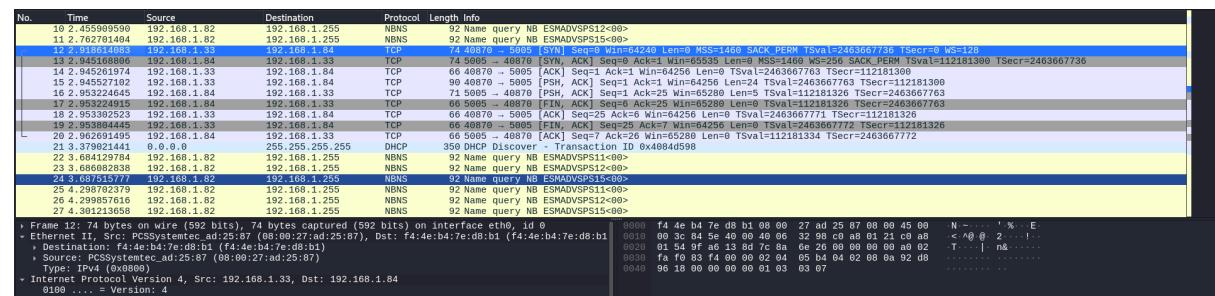
This trace captures a complete TCP session between two hosts: 192.168.1.33 (client) and 192.168.1.84 (server), communicating over port 5005. The session includes connection setup, data exchange, and termination.

- 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.

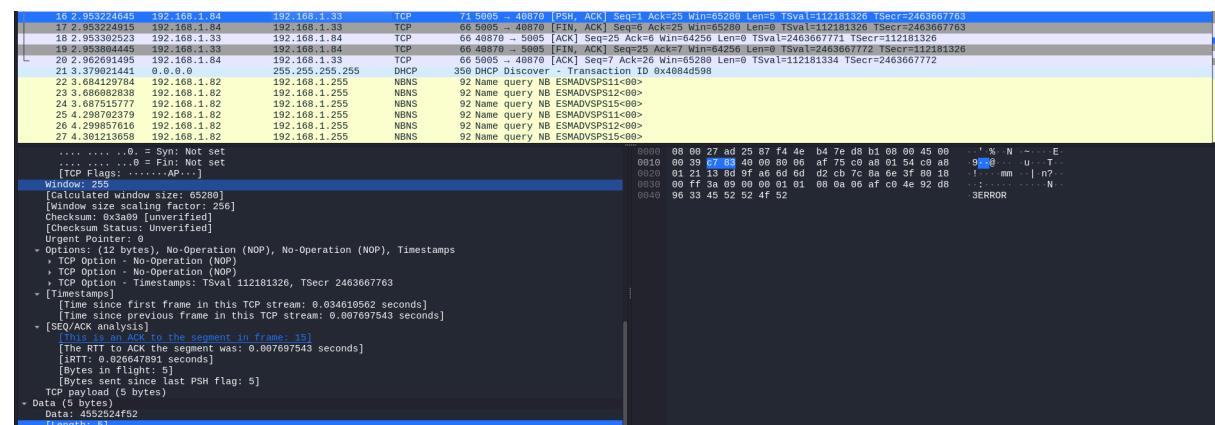
Command execution in case of error. In this case, I will declare a file that does not exist on the server:

```
(kali㉿kali)-[~/Desktop/REDES] Port: 68, Dst Port: 67
$ 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.

```
(kali㉿kali)-[~/Desktop/REDES]
└─$ echo "abcde" > letras.txt

(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.txt' subido correctamente.

(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 539641254 192.168.1.84
  letras.txt2 539816462 192.168.1.33
  servidor.py 547565032 192.168.1.84
```

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<0>
25	2.483138764	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS15<0>
26	2.516081415	192.168.1.33	192.168.1.84	TCP	74	32344 - 5085 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSeq=12465880499 TSecr=114394159 TSeq=2465880499 WS=128
27	2.523974638	192.168.1.84	192.168.1.33	TCP	74	5085 - 32344 [SYN, ACK] Seq=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TSeq=114394159 TSeq=2465880499
28	2.524253040	192.168.1.33	192.168.1.84	TCP	60	32344 - 5085 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=22 TSeq=12465880499 TSecr=114394159
29	2.524259488	192.168.1.33	192.168.1.84	TCP	88	32344 - 5085 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=22 TSeq=12465880499 TSecr=114394159
30	2.531989339	192.168.1.84	192.168.1.33	TCP	76	5085 - 32344 [PSH, ACK] Seq=1 Ack=23 Win=65280 Len=18 TSeq=114394167 TSecr=2465880498
31	2.531993628	192.168.1.33	192.168.1.84	TCP	66	32344 - 5085 [ACK] Seq=23 Ack=31 Win=64240 Len=0 TSeq=12465880496 TSecr=114394167
32	2.532000113	192.168.1.33	192.168.1.84	TCP	67	5085 - 32344 [ACK] Seq=24 Ack=31 Win=64240 Len=0 TSeq=12465880496 TSecr=114394167
33	2.5320041254	192.168.1.33	192.168.1.84	TCP	70	5085 - 32344 [PSH, ACK] Seq=24 Ack=31 Win=64240 Len=0 TSeq=12465880496 TSecr=114394167
34	2.538816462	192.168.1.33	192.168.1.84	TCP	72	32344 - 5085 [PSH, ACK] Seq=24 Ack=31 Win=64240 Len=0 TSeq=12465880496 TSecr=114394174
35	2.547565632	192.168.1.84	192.168.1.33	TCP	79	5085 - 32344 [PSH, ACK] Seq=21 Ack=30 Win=65280 Len=13 TSeq=114394183 TSecr=2465880514
36	2.549023849	192.168.1.84	192.168.1.33	TCP	117	5085 - 32344 [FIN, PSH, ACK] Seq=34 Ack=30 Win=65280 Len=51 TSeq=114394183 TSecr=2465880514
37	2.549432796	192.168.1.33	192.168.1.84	TCP	60	32344 - 5085 [ACK] Seq=34 Ack=30 Win=65280 Len=51 TSeq=114394183 TSecr=2465880514
38	2.549432796	192.168.1.33	192.168.1.84	TCP	66	32344 - 5085 [FIN, ACK] Seq=34 Ack=36 Win=64256 Len=6 TSeq=12465880524 TSecr=114394183
39	2.5587674959	192.168.1.84	192.168.1.33	TCP	66	5085 - 32344 [ACK] Seq=86 Ack=31 Win=65280 Len=8 TSeq=114394192 TSecr=2465880524
40	2.6506950038	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS11<0>
41	2.738976062	192.168.1.82	192.168.1.255	Broadcast	0	0 Who has 192.168.1.82 Tell 192.168.1.1?
42	2.738976069	192.168.1.82	192.168.1.251	MDNS	82	Standard query 0x0000 SRV _googlecast._tcp.local, "QM" question
43	2.955294776	192.168.1.41	224.0.0.251	MDNS	126	Standard query 0x0000 SRV _chromecast._tcp.fqdn.2454c8a0d516a8509, _googlecast._tcp.local, "QM" question
44	2.957046725	192.168.1.36	224.0.0.251	MDNS	126	Standard query 0x0000 SRV _chromecast._tcp.fqdn.2454c8a0d516a8509, _googlecast._tcp.local, "QM" question
45	2.957683237	fe80::d237:45ff:fe5:ff02::fb	224.0.0.251	MDNS	146	Standard query 0x0000 SRV _chromecast._tcp.fqdn.2454c8a0d516a8509, _googlecast._tcp.local, "QM" question
46	2.962873412	192.168.1.41	224.0.0.251	MDNS	408	Standard query 0x0000 PTR Chromecast. 97f2f911634692a554c8a0d516a8509, _chromecast._tcp.local TXT, cache Flush SRV, ca...
47	2.962873482	192.168.1.41	224.0.0.251	MDNS	193	Standard query response 0x0000 SRV, cache flush 0 8 8089 97f2f911-6346-9b2a-5a4c-8a0d516a8509.local A, cache Flush 192.168.1.41
48	3.143159249	192.168.1.41	224.0.0.251	MDNS	193	Standard query response 0x0000 SRV, cache flush 0 8 8089 97f2f911-6346-9b2a-5a4c-8a0d516a8509.local A, cache Flush 192.168.1.41
49	3.151698427	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS12<0>
50	3.166816869	192.168.1.82	192.168.1.255	NBNS	92	Name query NB ESMADVSPS12<0>
51	3.632157003	192.168.1.36	192.168.1.255	BROADCAST	288	Request AnnounceQuery RES01 TMS45E, Workstation, Server, SQL Server, NT Workstation
52	3.646739917	192.168.1.1	224.0.0.1	ICMPv2	68	Membership Query, general
53	3.648167384	fe80::ed11:bf:fe1:ff02::1	ICMPv6	99	Multicast Listener Query	
54	3.648167584	fe80::d237:45ff:fe1:ff02::16	ICMPv6	159	Multicast Listener Report Message V2	
55	3.658516439	fe80::d237:45ff:fe1:ff02::16	ICMPv6	159	Multicast Listener Report Message V2	
56	3.664898612	fe80::94fb:1778::ff08:ff02::16	ICMPv6	99	Multicast Listener Report Message V2	
57	3.739993712	192.168.1.36	224.0.0.252	ICMPv2	68	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-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.

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

[└(kali㉿kali)-[~/Desktop/REDES]
$ python cliente.py --ip 192.168.1.84 DELETE_FILE letras-copia.txt
Mandando el comando: DELETE_FILE letras-copia.txt a la IP: 192.168.1.84
Respuesta del servidor: ERROR

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

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.

5 3.954459182	AckeyCompute_ib:5e:.. Broadcast	ARP	69 Who has 192.168.1.33? Tell 192.168.1.1
6 3.954460846	PoECompuete_ad:25:.. AckeyCompute_ib:5e:.. ARP	ARP	42 192.168.1.33 192.168.1.1 08:00:27:00:00:00
7 3.954461209	0 0 0 0	DHCP	359 DHCP Discover Transaction ID 0x5506797
8 6.503700054	192.168.1.33	TCP	74 53326 - 5005 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TStamp=2095805618 TSecr=0 WS=128
9 6.664632559	192.168.1.84	TCP	74 5005 - 53326 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM TStamp=129221107 TSecr=2095805618
10 6.664725322	192.168.1.33	TCP	66 53326 - 5005 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TStamp=2095805719 TSecr=129221107
11 6.664981580	192.168.1.33	TCP	94 53326 - 5005 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=28 TStamp=2095805719 TSecr=129221107
12 6.664981580	192.168.1.84	TCP	74 53326 - 5005 [ACK] Seq=29 Ack=28 Win=65535 TStamp=1292211106 TSecr=2095805719
13 6.613444916	192.168.1.84	TCP	66 5005 - 53326 [FIN, ACK] Seq=29 Ack=28 Win=65535 TStamp=1292211106 TSecr=2095805719
14 6.613511680	192.168.1.33	TCP	66 53326 - 5005 [ACK] Seq=29 Ack=28 Win=64256 Len=0 TStamp=2095805727 TSecr=1292211106
15 6.613651801	192.168.1.33	TCP	66 53326 - 5005 [FIN, ACK] Seq=29 Ack=28 Win=64256 Len=0 TStamp=2095805728 TSecr=1292211106
16 6.622187522	192.168.1.84	TCP	66 5005 - 53326 [ACK] Seq=29 Ack=30 Win=65280 Len=0 TStamp=129221124 TSecr=2095805728
17 8.550945763	192.168.1.1	IGMPv2	69 Membership Query, general
18 8.552241373	f0:::eadd1:1bff:fe:ff02::1	ICMPv6	89 Multicast Listener Query
19 8.552241412	f880::d237:40ff:fe:ff02::16	ICMPv6	158 Multicast Listener Report Message v2
20 8.554050812	f880::1201:94cc:64b::ff02::16	ICMPv6	150 Multicast Listener Report Message v2

```

21.8.560707758 fe80::94fb:1778:ff02::16 ICMPv6 99 Multicast Listener Report Message v2
22.8.733247917 fe80::16c1:4eff:ff02::16 ICMPv6 118 Multicast Listener Report Message v2

> [Conversation completeness: Complete, WITH_DATA (31)]
[TCP Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 1382498826
[Next Sequence Number: 8 (relative sequence number)]
Acknowledgment Number: 29 (relative ack number)
Acknowledgment number (raw): 3774999333
Header ... = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 255
[Calculated window size: 65280]
[Window size scaling factor: 256]
Checksum: 0xe1ff [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]
TCP payload (7 bytes)
Data: 44454c45544544
[Length: 7]

0000 08 00 27 ad 25 87 f4 4e b4 7e d8 b1 08 00 45 00 .%N--E
0001 08 00 af 38 00 00 00 00 c7 be c8 a8 01 54 c9 a8 ;86.....T
0002 01 21 13 8d d4 44 1b d0 ee d0 18 0d 53 c7 80 18 !D...MS...
0003 00 ff fd a7 08 00 01 01 08 0a 00 10 57 09 a1 b1 .....W...
0040 b0 17 44 45 4c 45 54 45 44 .DELETE 0

```

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.

```

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Power Manager
Timeout was reached
tcp
No. Time Source Destination Protocol Length Info
14 2.596408463 192.168.1.84 192.168.1.84 TCP 74 54348 - 5865 [SYN] Seq=0 Win=64248 Len=146 SACK_PERM Tsvl=212777404 Tscr=0 WS=128
15 2.748119063 192.168.1.84 192.168.1.84 TCP 74 50895 - 54340 [SYN, ACK] Seq=1 Win=65535 Len=146 SACK_PERM Tsvl=212777404 Tscr=1078847 WS=128
16 2.748135307 192.168.1.84 192.168.1.84 TCP 64 54340 - 5865 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=28 Tsvl=212777519 Tscr=1078847
17 2.7481463613 192.168.1.84 192.168.1.84 TCP 94 54348 - 5865 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=28 Tsvl=212777519 Tscr=1078847
18 2.72659872 192.168.1.84 192.168.1.84 TCP 73 5085 - 54340 [PSH, ACK] Seq=1 Ack=29 Win=65280 Len=5 Tsvl=1078857 Tscr=2712777519
19 2.72659872 192.168.1.84 192.168.1.84 TCP 66 5085 - 54340 [FIN, ACK] Seq=29 Ack=29 Win=65280 Len=5 Tsvl=1078857 Tscr=2712777519
20 2.72659872 192.168.1.84 192.168.1.84 TCP 66 5085 - 54340 [FIN, ACK] Seq=29 Ack=29 Win=64256 Len=5 Tsvl=1078857 Tscr=2712777519
21 2.7211520201 192.168.1.84 192.168.1.84 TCP 66 54348 - 5865 [FIN, ACK] Seq=29 Ack=7 Win=64256 Len=0 Tsvl=212777529 Tscr=1078857
23 2.728835177 192.168.1.84 192.168.1.84 TCP 66 5085 - 54340 [ACK] Seq=7 Ack=38 Win=65280 Len=0 Tsvl=1078865 Tscr=2712777529

```

```

> [Conversation completeness: Complete, WITH_DATA (31)]
[TCP Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 466677456
[Next Sequence Number: 8 (relative sequence number)]
Acknowledgment Number: 29 (relative ack number)
Acknowledgment number (raw): 409818055
10980 .... = Header Length: 32 bytes (8)
Flags: 0x018 (PSH, ACK)
Window: 255
[Calculated window size: 65280]
[Window size scaling factor: 256]
Checksum: 0xdaf7 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[Timestamps]
[SEQ/ACK analysis]
TCP payload (5 bytes)
Data: 4552524f52
[Length: 5]

0000 08 00 27 ad 25 87 f0 ec 9c 7e d8 01 00 00 45 00 .%N--E
0001 08 00 89 48 00 00 06 75 00 00 a8 01 54 c9 a8 9.....T
0002 01 21 13 8d d4 44 1b d0 ee d0 18 0d 53 c7 80 18 !D...MS...
0003 00 ff fd a7 08 00 01 01 08 0a 00 10 57 09 a1 b1 .....W...
0040 b3 2f 45 52 52 4f 52 .ERROR

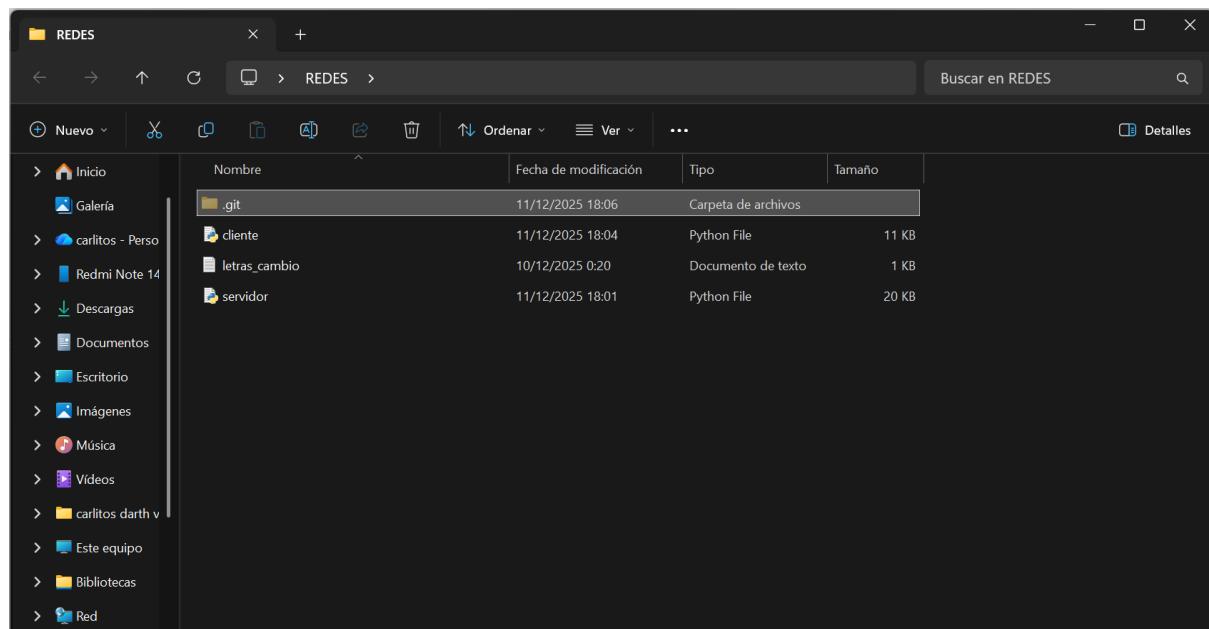
```

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.

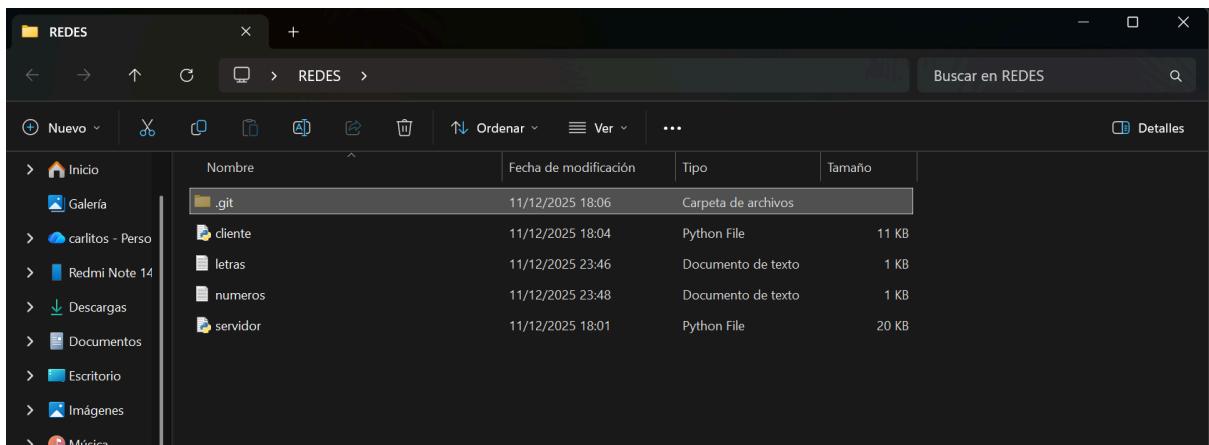
34. 4. 667859529	192.168.1.41	224.0.0.251	NBNS	193 Standard query response 0x0800 SRV, cache flush 0 0 8089 97f2f911-6346-9b2a-5a4c-8aa0d516a8599.local_A, cache flush 192.168.1.41
34. 4. 398373748	192.168.1.255	192.168.1.255	NBNS	92 Name query NB ESMADVSPS11<0>
37. 4. 398095472	192.168.1.82	192.168.1.255	NBNS	92 Name query NB ESMADVSPS15<0>
38. 4. 634174743	192.168.1.82	192.168.1.255	NBNS	92 Name query NB ESMADVSPS12<0>
39. 4. 457515259	192.168.1.33	192.168.1.84	TCP	74 37810 - 5905 [SYN] Seq=0 Win=64240 Len=MSS=1460 SACK_PERM TStamp=2714559172 TSecr=0 WNs=128
40. 4. 530873748	192.168.1.84	192.168.1.33	TCP	74 37810 - 5905 [SYN, ACK] Seq=1 Win=64240 Len=0 TStamp=2714559172 TSecr=2852669
41. 4. 530873748	192.168.1.33	192.168.1.84	TCP	66 37810 - 5905 [ACK] Seq=1 Win=64256 Len=0 TStamp=2714559245 TSecr=2852669
42. 4. 931166593	192.168.1.33	192.168.1.84	TCP	106 37810 - 5905 [PSH, ACK] Seq=1 Win=64256 Len=40 TStamp=2714559246 TSecr=2852669
43. 4. 566697674	192.168.1.84	192.168.1.33	TCP	73 5095 - 37810 [PSH, ACK] Seq=1 Win=65280 Len=0 TStamp=2852700 TSecr=2714559246
44. 4. 366677185	192.168.1.33	192.168.1.84	TCP	66 5095 - 37810 [FIN, ACK] Seq=1 Win=65280 Len=0 TStamp=2852700 TSecr=2714559246
45. 4. 567677302	192.168.1.84	192.168.1.33	TCP	66 37810 - 5905 [ACK] Seq=1 Win=65280 Len=0 TStamp=2714559282 TSecr=2852706
46. 4. 567677302	192.168.1.33	192.168.1.84	TCP	66 5095 - 37810 [ACK] Seq=1 Win=64256 Len=0 TStamp=2714559282 TSecr=2852706
47. 4. 988255887	192.168.1.84	192.168.1.33	TCP	66 5095 - 37810 [ACK] Seq=9 Win=65280 Len=0 TStamp=2852719 TSecr=2714559282
48. 5. 065477975	192.168.1.82	192.168.1.255	NBNS	92 Name query NB ESMADVSPS11<0>
49. 5. 063031981	192.168.1.82	192.168.1.255	NBNS	92 Name query NB ESMADVSPS15<0>

```
48 5.065470759 192.168.1.82 192.168.1.255 NBNS 92 Name query NB.ESMAWPSVS11<0>
49 5.065470759 192.168.1.82 192.168.1.255 NBNS 92 Name query NB.ESMAWPSVS11<0>
[Stream Index: 1]
+ [Conversation completeness: Complete, WITH Data (31)]
[TCP Segment Len: 7]
Sequence Number (Raw): 0x00000000 (relative sequence number)
Source Sequence Number (Raw): 0x00000000 (relative sequence number)
[Next Sequence Number: 8 (relative sequence number)]
Acknowledgment Number: 0x41 (relative ack number)
Acknowledgment Number (Raw): 316922011
1000: 0x00000000 - Header Length: 32 bytes (8)
Flags: 0x010 (PSH, ACK)
Window: 255
[Window size: window size: 65200]
[Window size scaling factor: 256]
Checksum: 0xa323 [unverified]
[Checksum Status: Unverified]
User Options: 0
+ Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
[[Timestamps]]
+ [SACK/Adaptive]
[LSR/Payload (7 bytes)]
- Data (7 bytes)
Data: 0x2d45e414d4544
Length: 71
```

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.

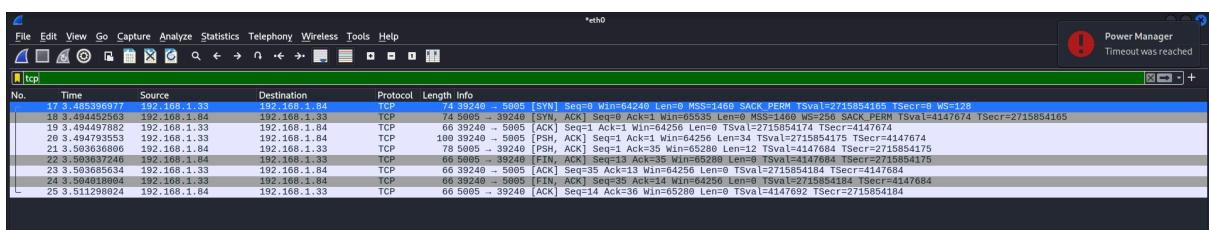
```
[kali㉿kali)-[~/Desktop/REDES]
$ python cliente.py --ip 192.168.1.84 RENAME_FILE letras.txt letras_cambio.txt
Mandando el comando: RENAME_FILE letras.txt letras_cambio.txt a la IP: 192.168.1.84
Respuesta del servidor: RENAME_ERROR
```

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.

```

[Stream index: 0]
> [Conversation completeness: Complete, WITH_DATA (31)]
[TCP Segment Len: 12]
Sequence Number: 1 (relative sequence number)
Sequence Number (raw): 2962941717
[Next Sequence Number: 13 (relative sequence number)]
Acknowledgment Number: 30 (relative ack number)
Acknowledgment Number (raw): 76654892
1000 .... = Header length: 32 bytes (8)
Flags: 0x019 (PSH, ACK)
Window: 255
[Current window size: 65280]
[Window size scaling factor: 256]
Checksum: 0xa750 [Unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
Timestamps
[SPWAK analysis]
TCP payload (32 bytes)
Data (12 bytes)
Data: 524546414d455f4552524f52
Length: 12
Data(data,data), 12 bytes

```

Packets: 46 - Displayed: 9 (19.6%) - Dropped: 0 (0.0%)

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.

```

└─(kali㉿kali)-[~/Desktop/REDES]
$ python cliente.py --ip 192.168.1.84 MSFCONSOLE
Mandando el comando: MSFCONSOLE a la IP: 192.168.1.84
UNKNOWN_COMMAND
└─(kali㉿kali)-[~/Desktop/REDES]
$ 

```

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.

No.	Time	Source	Destination	Protocol	Length	Info
64	6.148499788	192.168.1.33	192.168.1.84	TCP	74	44850 - 5005 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM Tsvl=2713144106 Tscr=0 WS=128
66	6.159533538	192.168.1.84	192.168.1.33	TCP	74	5005 - 44850 [SYN, ACK] Seq=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM Tsvl=1437467 Tscr=2713144106
67	6.159560693	192.168.1.33	192.168.1.84	TCP	66	44850 - 5005 [ACK] Seq=1 Ack=1 Win=64256 Len=0 Tsvl=2713144106 Tscr=1437467
68	6.159561359	192.168.1.33	192.168.1.84	TCP	74	44850 - 5005 [SYN, ACK] Seq=0 Win=65535 Len=0 Tsvl=2713144117 Tscr=2713144117
69	6.1595800673	192.168.1.84	192.168.1.33	TCP	81	5005 - 44850 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=15 Tsvl=1437478 Tscr=2713144117
70	6.160801043	192.168.1.84	192.168.1.33	TCP	66	5005 - 44850 [FIN, ACK] Seq=16 Ack=11 Win=65280 Len=0 Tsvl=1437478 Tscr=2713144117
71	6.1608066921	192.168.1.33	192.168.1.84	TCP	66	44850 - 5005 [ACK] Seq=11 Ack=16 Win=64256 Len=0 Tsvl=2713144126 Tscr=1437478
72	6.1608240473	192.168.1.33	192.168.1.84	TCP	66	44850 - 5005 [FIN, ACK] Seq=11 Ack=17 Win=64256 Len=0 Tsvl=2713144126 Tscr=1437478
73	6.1793012918	192.168.1.84	192.168.1.33	TCP	66	5005 - 44850 [ACK] Seq=17 Ack=32 Win=65280 Len=0 Tsvl=1437450 Tscr=2713144126