REDES Y SISTEMAS DISTRIBUIDOS

PRÁCTICAS BLQ 2

- **PRÁCTICA 4**
- **PRÁCTICA 5**

MARTA LÓPEZ PÉREZ

INGENIERÍA INFORMÁTICA 2ºA

Práctica 4

Alumno 1: Apellidos, Nombre: López Pérez, Marta

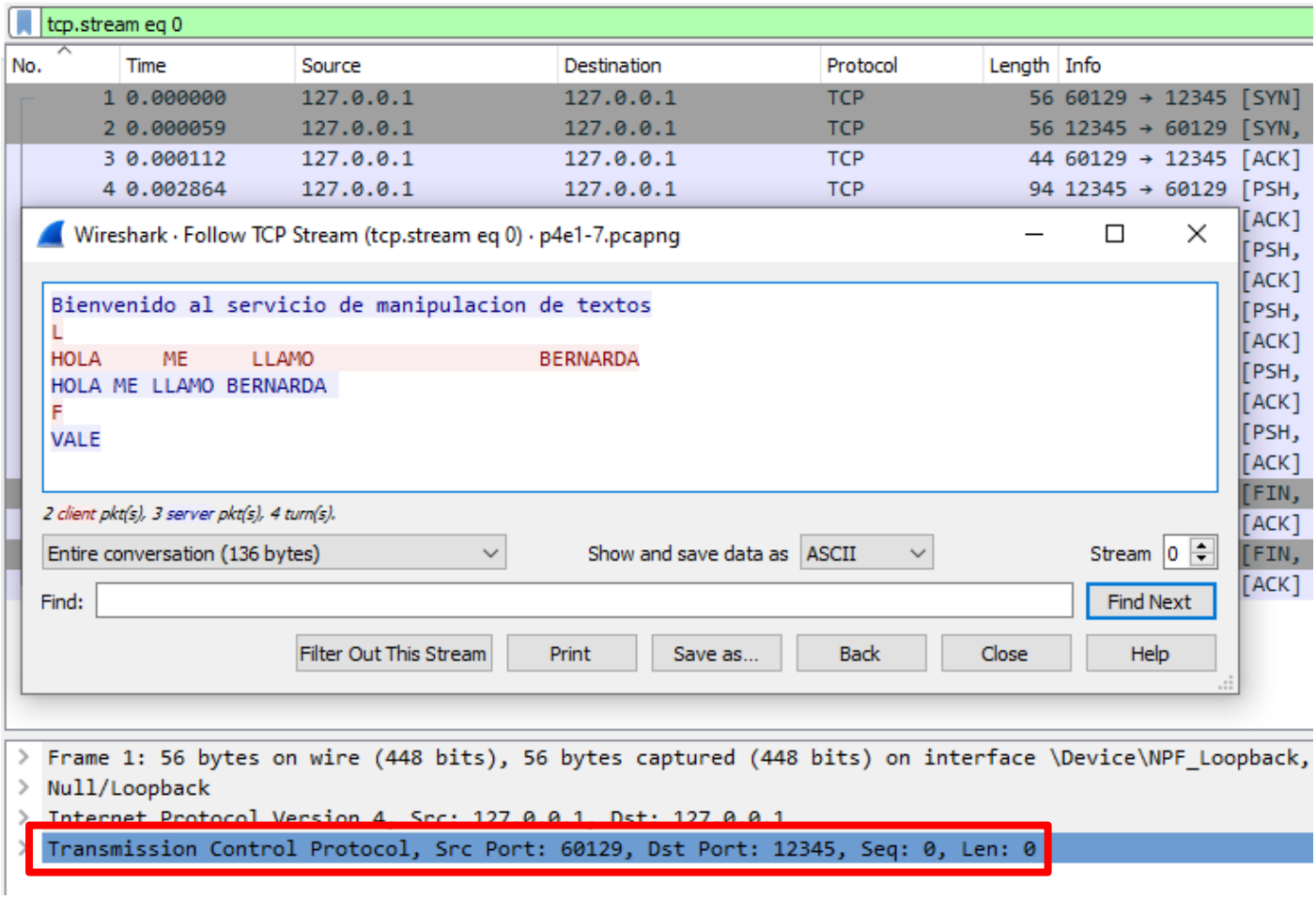
Titulación: Grado de Ingeniería Informática (Grupo A)

PC de la práctica: 012

Recuerde que debe añadir una explicación del código (incluyendo todas las sentencias relacionadas con los sockets). Puede añadir esta explicación como comentarios en el código.

Usando la traza 1 (**p4e1-7.pcapng**):

Ejercicio 1. Identifique una trama de la comunicación y use la opción "Follow TCP stream" para ver el intercambio de información entre cliente y servidor. Muestra una captura de pantalla con dicha información.



Ejercicio 2. Los mensajes enviados por el cliente (opción y texto), ¿van en el mismo segmento TCP o en segmentos separados? ¿Por qué?

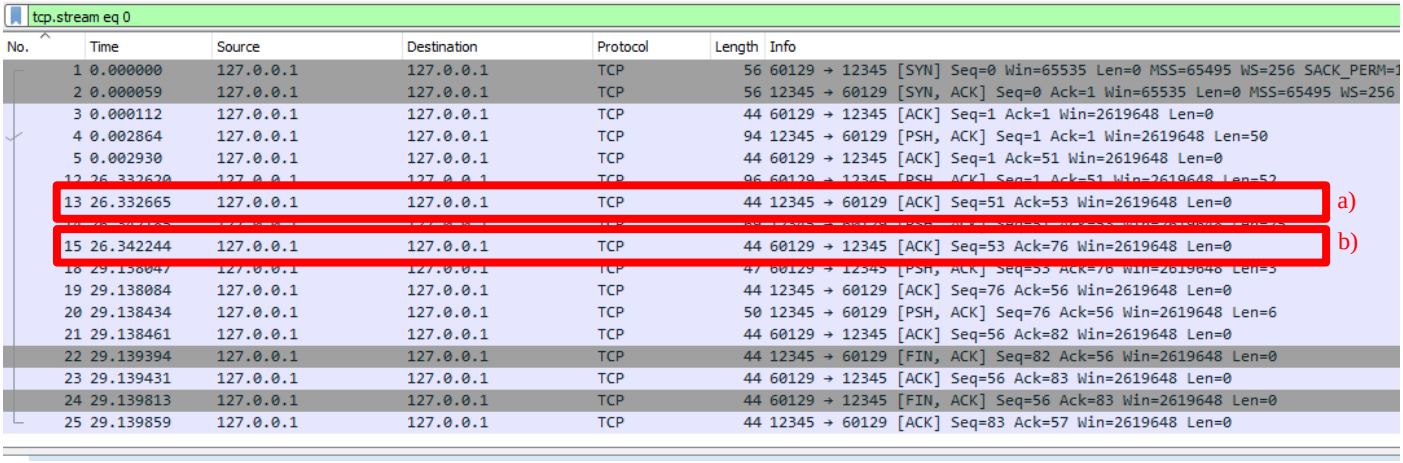
Van en segmentos separados porque se trata de mensajes distintos.

Ejercicio 3. ¿Cuál es el puerto que usa el cliente? ¿Y el servidor? ¿En qué campos de la cabecera del segmento TCP están cada uno?

El puerto del cliente es 60129 y el del servidor 12345. (Señalado en la captura del ejercicio 1)
Se encuentran en los campos source port (bits 34-35) y destination port (bits 36-37).

Ejercicio 4. ¿Cuál es el número de secuencia que se usa el cliente TCP hacia el servidor? ¿Y las respuestas del servidor al cliente?

a) Servidor → Cliente Nº sequencia = 51.
b) Cliente → Servidor Nº sequencia = 53.



Ejercicio 5. Indique los segmentos relacionados con las siguientes actividades y qué métodos de Socket y ServerSocket son responsables del intercambio de estos segmentos:

a) Inicialización de la conexión.

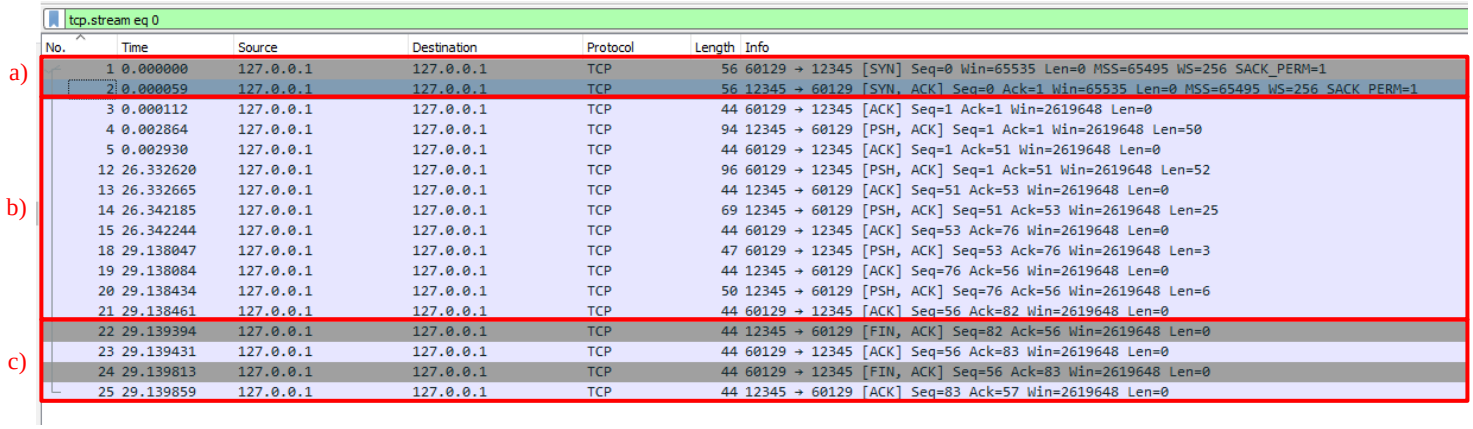
Socket echoSocket = new Socket(ip, puer) en el cliente;
Socket client = server.accept() en el server;

b) Envío de datos.

PrintWriter out = new PrintWriter(echoSocket.getOutputStream(),true);
BufferedReader in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));

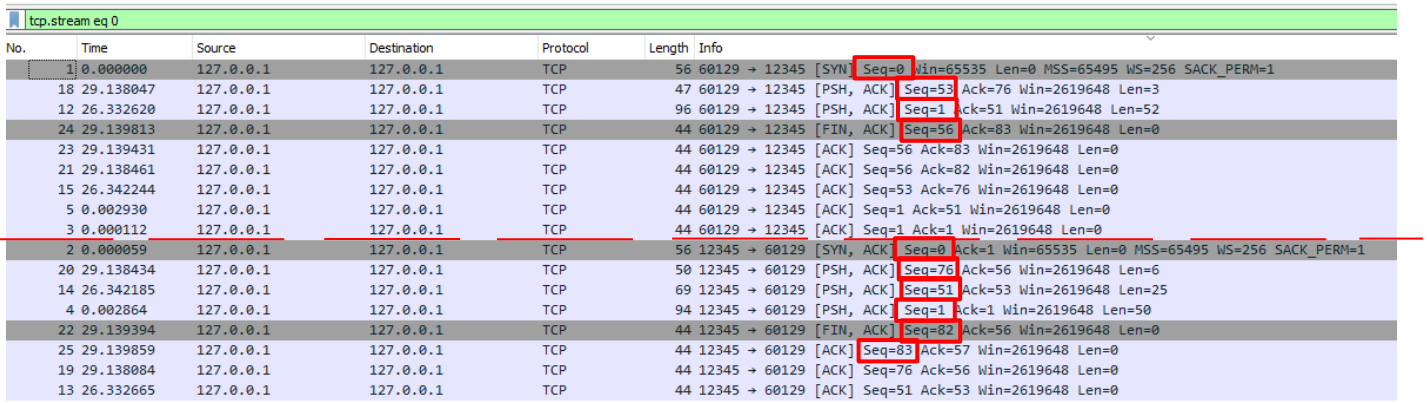
Tanto en el servidor como en el cliente, los envíos se hacen con out.println(string) y las lecturas con in.readline();

c) Finalización de la conexión.
Se cierran los streams out/in en el servidor y en el cliente con in.close() y out.close().
Desde el cliente cerramos la conexion con echoSocket.close() y desde el servidor con client.close().



Ejercicio 6. ¿Cuántos números de secuencia se consumen en cada lado (cliente y servidor) durante el inicio y cierre de la conexión?

Cliente → Se consumen 4 números de secuencia (0, 1, 53, 56).
Servidor → Se consumen 6 números de secuencia (0, 1, 51, 76, 82, 83).

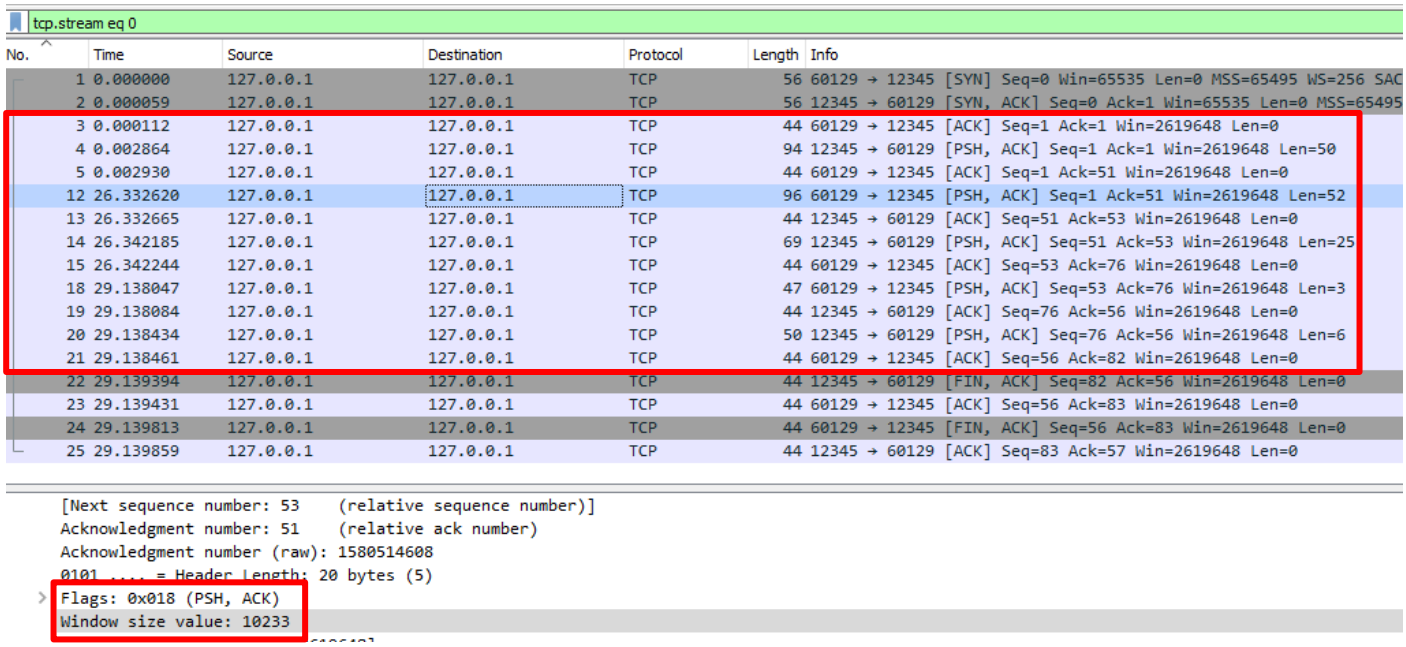


tcp.stream eq 0

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	60129 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
18	29.138047	127.0.0.1	127.0.0.1	TCP	47	60129 → 12345 [PSH, ACK] Seq=53 Ack=76 Win=2619648 Len=3
12	26.332620	127.0.0.1	127.0.0.1	TCP	96	60129 → 12345 [PSH, ACK] Seq=1 Ack=51 Win=2619648 Len=52
24	29.139813	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [FIN, ACK] Seq=56 Ack=83 Win=2619648 Len=0
23	29.139431	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=56 Ack=83 Win=2619648 Len=0
21	29.138461	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=56 Ack=82 Win=2619648 Len=0
15	26.342244	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=53 Ack=76 Win=2619648 Len=0
5	0.002930	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=1 Ack=51 Win=2619648 Len=0
3	0.000112	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
2	0.000059	127.0.0.1	127.0.0.1	TCP	56	12345 → 60129 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1
20	29.138434	127.0.0.1	127.0.0.1	TCP	50	12345 → 60129 [PSH, ACK] Seq=76 Ack=56 Win=2619648 Len=6
14	26.342185	127.0.0.1	127.0.0.1	TCP	69	12345 → 60129 [PSH, ACK] Seq=51 Ack=53 Win=2619648 Len=25
4	0.002864	127.0.0.1	127.0.0.1	TCP	94	12345 → 60129 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=50
22	29.139394	127.0.0.1	127.0.0.1	TCP	44	12345 → 60129 [FIN, ACK] Seq=82 Ack=56 Win=2619648 Len=0
25	29.139859	127.0.0.1	127.0.0.1	TCP	44	12345 → 60129 [ACK] Seq=83 Ack=57 Win=2619648 Len=0
19	29.138084	127.0.0.1	127.0.0.1	TCP	44	12345 → 60129 [ACK] Seq=76 Ack=56 Win=2619648 Len=0
13	26.332665	127.0.0.1	127.0.0.1	TCP	44	12345 → 60129 [ACK] Seq=51 Ack=53 Win=2619648 Len=0

Ejercicio 7. Observe el tamaño de la ventana deslizante del cliente y del servidor en cada segmento de envío de datos. ¿Cambia este valor? ¿Qué valores toma en el cliente y en el servidor?

El valor que toma es 10233, y durante el envío de datos se mantiene constante.



tcp.stream eq 0

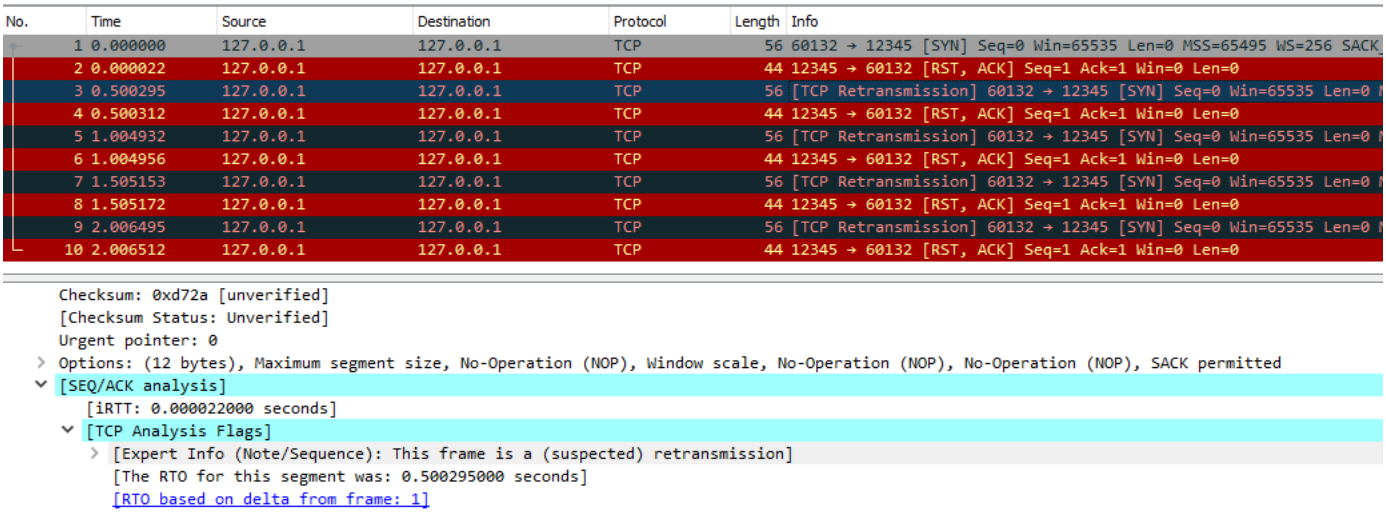
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	60129 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SAC
2	0.000059	127.0.0.1	127.0.0.1	TCP	56	12345 → 60129 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495
3	0.000112	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
4	0.002864	127.0.0.1	127.0.0.1	TCP	94	12345 → 60129 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=50
5	0.002930	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=1 Ack=51 Win=2619648 Len=0
12	26.332620	127.0.0.1	127.0.0.1	TCP	96	60129 → 12345 [PSH, ACK] Seq=1 Ack=51 Win=2619648 Len=52
13	26.332665	127.0.0.1	127.0.0.1	TCP	44	12345 → 60129 [ACK] Seq=51 Ack=53 Win=2619648 Len=0
14	26.342185	127.0.0.1	127.0.0.1	TCP	69	12345 → 60129 [PSH, ACK] Seq=51 Ack=53 Win=2619648 Len=25
15	26.342244	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=53 Ack=76 Win=2619648 Len=0
18	29.138047	127.0.0.1	127.0.0.1	TCP	47	60129 → 12345 [PSH, ACK] Seq=53 Ack=76 Win=2619648 Len=3
19	29.138084	127.0.0.1	127.0.0.1	TCP	44	12345 → 60129 [ACK] Seq=76 Ack=56 Win=2619648 Len=0
20	29.138434	127.0.0.1	127.0.0.1	TCP	50	12345 → 60129 [PSH, ACK] Seq=76 Ack=56 Win=2619648 Len=6
21	29.138461	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=56 Ack=82 Win=2619648 Len=0
22	29.139394	127.0.0.1	127.0.0.1	TCP	44	12345 → 60129 [FIN, ACK] Seq=82 Ack=56 Win=2619648 Len=0
23	29.139431	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [ACK] Seq=56 Ack=83 Win=2619648 Len=0
24	29.139813	127.0.0.1	127.0.0.1	TCP	44	60129 → 12345 [FIN, ACK] Seq=56 Ack=83 Win=2619648 Len=0
25	29.139859	127.0.0.1	127.0.0.1	TCP	44	12345 → 60129 [ACK] Seq=83 Ack=57 Win=2619648 Len=0

```
[Next sequence number: 53    (relative sequence number)]
Acknowledgment number: 51    (relative ack number)
Acknowledgment number (raw): 1580514608
0101 .... = Header Length: 20 bytes (5)
Flags: 0x018 (PSH, ACK)
Window size value: 10233
```

Usando la traza 2 (**p4e8.pcapng**):

Ejercicio 8. ¿Recibe algún tipo de respuesta el intento de conexión del cliente? En caso afirmativo ¿tiene alguna característica especial?

Sí, la traza es una retransmisión.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	60132 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK
2	0.000022	127.0.0.1	127.0.0.1	TCP	44	12345 → 60132 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.500295	127.0.0.1	127.0.0.1	TCP	56	[TCP Retransmission] 60132 → 12345 [SYN] Seq=0 Win=65535 Len=0
4	0.500312	127.0.0.1	127.0.0.1	TCP	44	12345 → 60132 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	1.004932	127.0.0.1	127.0.0.1	TCP	56	[TCP Retransmission] 60132 → 12345 [SYN] Seq=0 Win=65535 Len=0
6	1.004956	127.0.0.1	127.0.0.1	TCP	44	12345 → 60132 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
7	1.505153	127.0.0.1	127.0.0.1	TCP	56	[TCP Retransmission] 60132 → 12345 [SYN] Seq=0 Win=65535 Len=0
8	1.505172	127.0.0.1	127.0.0.1	TCP	44	12345 → 60132 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	2.006495	127.0.0.1	127.0.0.1	TCP	56	[TCP Retransmission] 60132 → 12345 [SYN] Seq=0 Win=65535 Len=0
10	2.006512	127.0.0.1	127.0.0.1	TCP	44	12345 → 60132 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

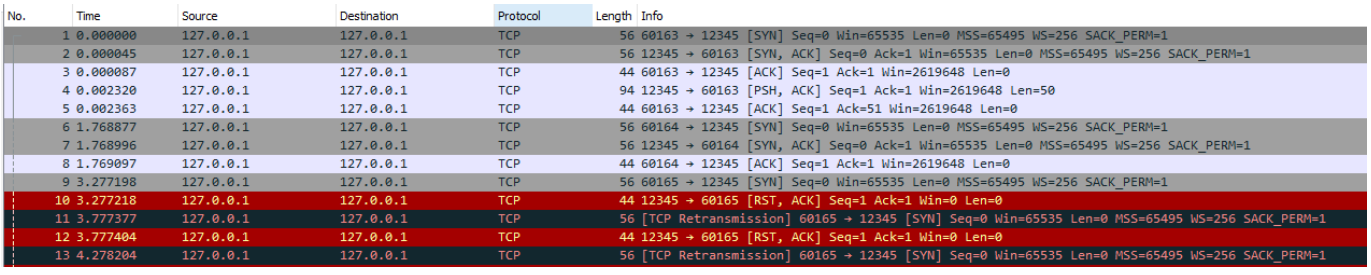
```
Checksum: 0xd72a [unverified]
[Checksum Status: Unverified]
Urgent pointer: 0
Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
[SEQ/ACK analysis]
    [iRTT: 0.000022000 seconds]
[TCP Analysis Flags]
    [Expert Info (Note/Sequence): This frame is a (suspected) retransmission]
    [The RTO for this segment was: 0.500295000 seconds]
    [RTO based on delta from frame: 1]
```

Usando la traza 3 (**p4e9-10.pcapng**):

Ejercicio 9. ¿Se logran conectar los 3 clientes? En caso de alguno no se haya podido conectar, ¿se le indica de alguna forma que la cola está llena?

No. El primer cliente se conecta correctamente, el segundo se queda en espera, pero el tercero nunca llega a conectarse.

No se indica que la cola está llena.



No.	Time	Source	Destination	Protocol	Length	Info	
1	0.000000	127.0.0.1	127.0.0.1	TCP	56	60163 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1	Cliente 1
2	0.000045	127.0.0.1	127.0.0.1	TCP	56	12345 → 60163 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1	
3	0.000087	127.0.0.1	127.0.0.1	TCP	44	60163 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0	
4	0.002320	127.0.0.1	127.0.0.1	TCP	94	12345 → 60163 [PSH, ACK] Seq=1 Ack=1 Win=2619648 Len=50	
5	0.002363	127.0.0.1	127.0.0.1	TCP	44	60163 → 12345 [ACK] Seq=1 Ack=51 Win=2619648 Len=0	
6	1.768877	127.0.0.1	127.0.0.1	TCP	56	60164 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1	Cliente 2
7	1.768996	127.0.0.1	127.0.0.1	TCP	56	12345 → 60164 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1	
8	1.769097	127.0.0.1	127.0.0.1	TCP	44	60164 → 12345 [ACK] Seq=1 Ack=1 Win=2619648 Len=0	
9	3.277198	127.0.0.1	127.0.0.1	TCP	56	60165 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1	Cliente 3
10	3.277218	127.0.0.1	127.0.0.1	TCP	44	12345 → 60165 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0	
11	3.777377	127.0.0.1	127.0.0.1	TCP	56	[TCP Retransmission] 60165 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1	
12	3.777404	127.0.0.1	127.0.0.1	TCP	44	12345 → 60165 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0	
13	4.278204	127.0.0.1	127.0.0.1	TCP	56	[TCP Retransmission] 60165 → 12345 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM=1	

Ejercicio 10. ¿Los clientes en espera (es decir los que están en la cola) tiene inicializada la conexión o esa inicialización se hace cuando se sacan de la cola (con el método accept)?

La conexión se inicializa en el método accept. Al realizar la conexión del segundo cliente, estando el primero ya conectado, este se queda esperando antes del método accept hasta que se libera el primero.

Práctica 5

Alumno 1: Apellidos, Nombre: López Pérez, Marta

Titulación: Grado de Ingeniería Informática(Grupo A)

PC de la práctica: 012

Recuerde que debe añadir una explicación del código (incluyendo todas las sentencias relacionadas con los sockets). Puede añadir esta explicación como comentarios en el código.

Usando la traza 1 (**p5e1-5.pcapng**):

Ejercicio 1. ¿Cuántos mensajes se intercambian en esta comunicación? ¿Cuántos mensajes se generarían si usáramos TCP (suponiendo que el cliente envía la opción y el texto de forma conjunta como en UDP)?

Se intercambian 4 mensajes, uno que va del servidor al cliente y otro viceversa.

udp.stream eq 0

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	55	62091 → 12345 Len=23
2	0.041738	127.0.0.1	127.0.0.1	UDP	49	12345 → 62091 Len=17

Si se usara TCP se habrían generado 10 (2 para establecer conexión, 4 de cada mensaje con sus acks correspondientes y los 4 para finalizar la conexión).

Ejercicio 2. ¿Cuál es el puerto que usa el cliente? ¿Y el servidor? ¿Qué tipo de puerto es cada uno de ellos?

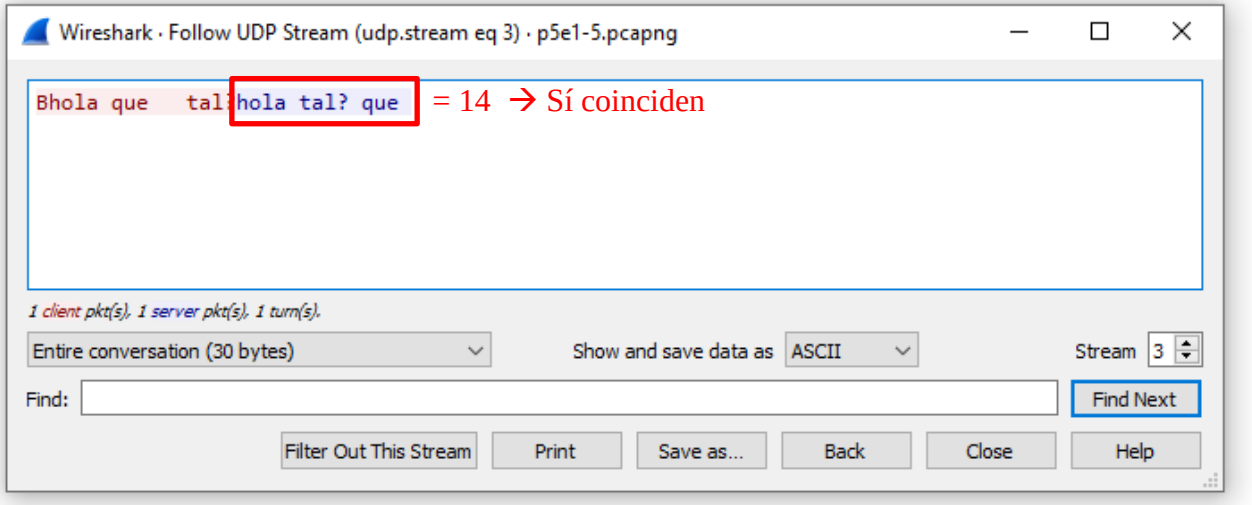
Cliente → 62091
Servidor → 12345
(Captura ejercicio 1)

Ejercicio 3. Wireshark ofrece la opción de “Follow UDP stream”, pero en UDP no existe tal concepto. ¿Cómo es capaz Wireshark de decidir de un mensaje pertenece a un “flujo” u a otro?

Ejercicio 4. Examine un mensaje enviado por el servidor, ¿coincide el tamaño indicado en el campo longitud de la cabecera de UDP con la cantidad de datos que van en el datagrama? ¿Por qué?

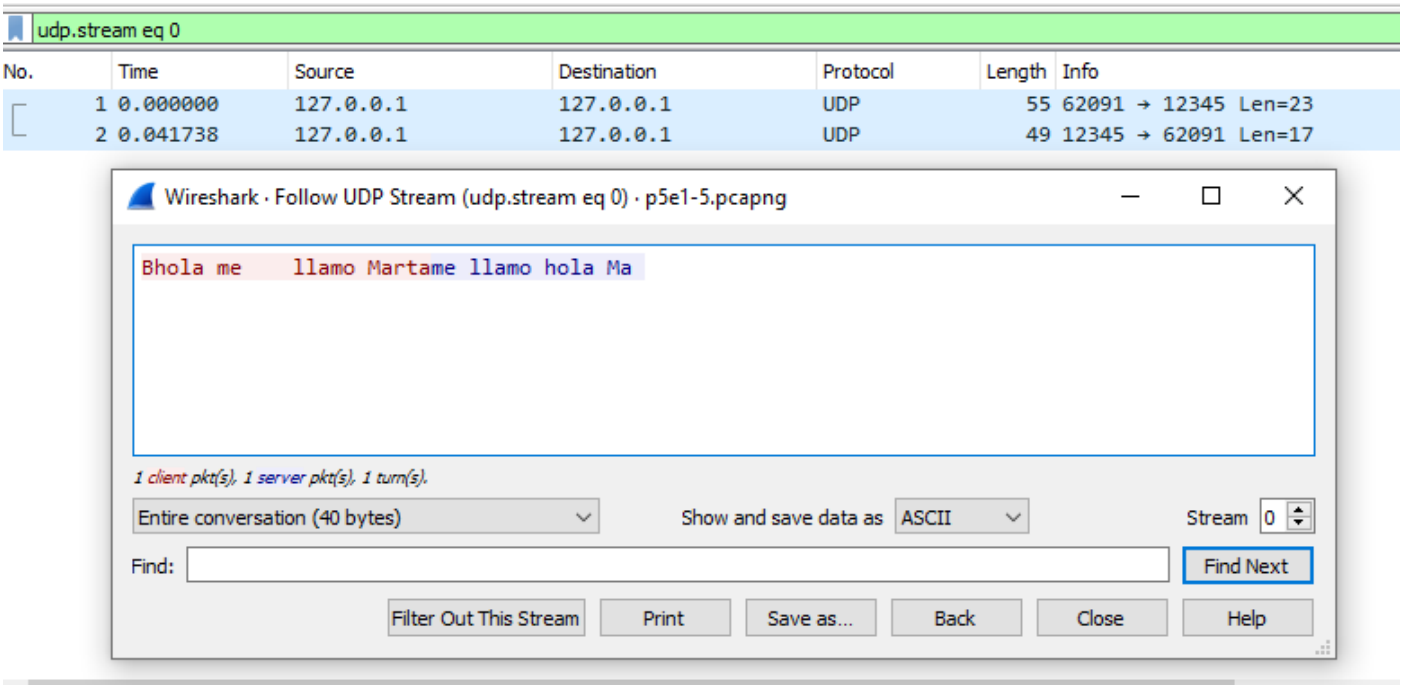
udp.stream eq 3

No.	Time	Source	Destination	Protocol	Length	Info
11	59.114446	127.0.0.1	127.0.0.1	UDP	48	50304 → 12345 Len=16
12	59.115383	127.0.0.1	127.0.0.1	UDP	46	12345 → 50304 Len=14



Ejercicio 5. Examine ahora el mensaje largo (de más de 20 letras), ¿qué cantidad de datos envía el cliente? ¿y el servidor? ¿qué pasó con los datos que superaban las 20 letras (tamaño del buffer de recepción)?

El cliente envía 23 y el servidor le responde 20 (hice mal la captura, luego me salía bien).
Los datos que superaban los datos se eliminan.



Usando la traza 2 (**p5e6-7.pcapng**):

Ejercicio 6. ¿Por qué consigue enviar si no hay ningún servidor activo? ¿Recibe alguna respuesta? En caso afirmativo, indique qué significa esa respuesta y si es tratada o no.

Recibe respuesta de error, aunque no se manda un mensaje el datagrama se crea.

Ejercicio 7. Si corta le ejecución del cliente, ¿se envía algún mensaje de cierre? ¿Por qué?

No se envían mensaje de cierre.

Sin traza:

Ejercicio 8. Asegure que captura la excepción de la creación del socket UDP y que muestra (método `getMessage()`) el error que se produce (modifique el código si no lo hacía). Intente abrir dos veces el servidor con los mismos parámetros, ¿qué error indica que se produce? ¿Qué debería hacer para solucionar ese error y tener dos servidores del mismo tipo en su equipo?

Indica el error→ Address already in use: Cannot bind.

Para solucionarlo debería cambiarse el puerto en uno de ellos.