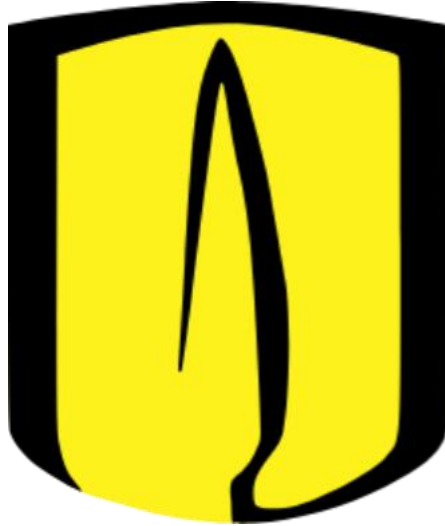


DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN



LABORATORIO #3: ANÁLISIS CAPA DE TRANSPORTE Y SOCKETS

ISIS3204 – INFRAESTRUCTURA DE COMUNICACIONES

PROFESOR

CARLOS LOZANO

GRUPO 6

MARÍA LUCÍA BENAVIDES DOMÍNGUEZ – 202313423

DANIEL CAMILO QUIMBAY VELÁSQUEZ - 202313861

NIKOL RODRIGUEZ ORTIZ – 202317538

2025-20

TABLA DE CONTENIDOS

Laboratorio 3: Análisis Capa de Transporte y Sockets

1. Objetivos

- Comprender el funcionamiento de los sockets en C.
- Diseñar un modelo de publicación-suscripción para comunicación entre procesos.
- Analizar y comparar TCP y UDP en confiabilidad, control de flujo, orden y pérdida de mensajes.
- Usar Wireshark para observar diferencias entre ambos protocolos.

2. Descripción del Sistema

Sistema de noticias deportivas en tiempo real

- Explicación general del sistema.
- Roles:
 - **Publisher:** envía mensajes del partido.
 - **Broker:** recibe y redistribuye mensajes.
 - **Subscriber:** recibe las actualizaciones.

3. Implementación

3.1. Configuración Inicial

- Lenguaje: C (compilado con GCC en Linux).
- Estructura de archivos:

broker_tcp.c, publisher_tcp.c, subscriber_tcp.c
broker_udp.c, publisher_udp.c, subscriber_udp.c

3.2. Pruebas y Ejecución

- Ejecución y Configuración:

Para realizar las pruebas de ejecución y comparación se realizó el envío de 10 mensajes por parte del publisher y se tomó registro de esto. Se hizo con la instrucción indicada de tener 2 publishers, 2 subscribers y un broker.

```

malu@malu-virtual-machine: ~/lab3_pubsub 65x19
malu@malu-virtual-machine:~$ ./broker_tcp
bash: ./broker_tcp: No such file or directory
malu@malu-virtual-machine:~$ cd lab3_pubsub/
malu@malu-virtual-machine:~/lab3_pubsub$ ./broker_tcp
Broker TCP escuchando en puerto 8080...
Nueva conexión: fd=4, ip=127.0.0.1, puerto=46780
Nueva conexión: fd=5, ip=127.0.0.1, puerto=50318
Mensaje recibido: SUBSCRIBE A_vs_B
Tema creado y suscriptor agregado: A_vs_B
Mensaje recibido: SUBSCRIBE A_vs_B
Nuevo suscriptor al tema A_vs_B

malu@malu-virtual-machine:~/lab3_pubsub$ sudo tcpdump -i lo -w tcp_pubsub.pcap tcp port 8080
[sudo] password for malu:
malu@malu-virtual-machine:~/lab3_pubsub$ ^C
malu@malu-virtual-machine:~/lab3_pubsub$ ./subscriber_tcp 127.0.0.1 8080
bash: ./subscriber_tcp: Is a directory
malu@malu-virtual-machine:~/lab3_pubsub$ ./subscriber_tcp 127.0.0.1 8080
Conectado al broker TCP en 127.0.0.1:8080
Ingrese el partido o tema al que desea suscribirse: A_vs_B
Esperando mensajes...

malu@malu-virtual-machine:~/lab3_pubsub$ cd lab3_pubsub/
malu@malu-virtual-machine:~/lab3_pubsub$ ./subscriber_tcp 127.0.0.1 8080
Conectado al broker TCP en 127.0.0.1:8080
Ingrese el partido o tema al que desea suscribirse: A_vs_B
Esperando mensajes...

```

Figura 1. Inicio de ejecución de broker con 2 subscribers esperando (TCP)

```

malu@malu-virtual-machine:~/lab3_pubsub 65x19
malu@malu-virtual-machine:~/lab3_pubsub$ ./broker_udp
Broker UDP escuchando en el puerto 8081...
Mensaje recibido: SUBSCRIBE A_vs_B
Tema creado y suscriptor agregado: A_vs_B
Mensaje recibido: SUBSCRIBE A_vs_B
Nuevo suscriptor agregado al tema A_vs_B

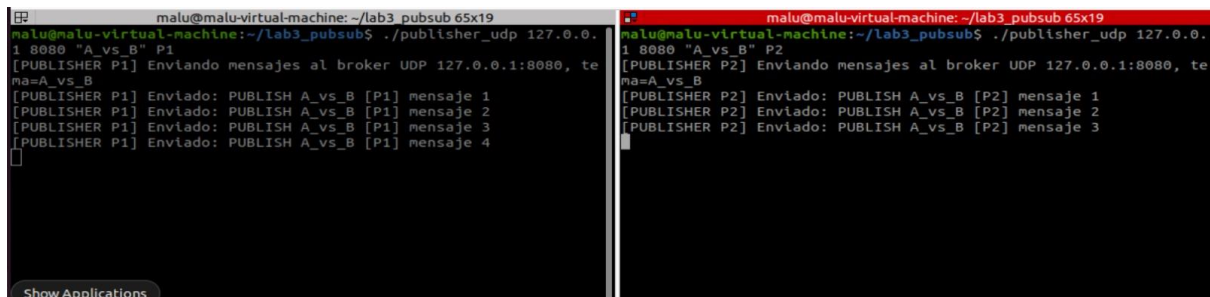
malu@malu-virtual-machine:~/lab3_pubsub$ ./subscriber_udp 127.0.0.1 8081
Ingrese el partido o tema al que desea suscribirse: A_vs_B
Suscripción enviada al broker UDP 127.0.0.1:8081
Esperando mensajes...

malu@malu-virtual-machine:~/lab3_pubsub$ cd lab3_pubsub/
malu@malu-virtual-machine:~/lab3_pubsub$ ./subscriber_udp 127.0.0.1 8081
Ingrese el partido o tema al que desea suscribirse: A_vs_B
Suscripción enviada al broker UDP 127.0.0.1:8081
Esperando mensajes...

```

Figura 2. Inicio de ejecución de broker con 2 subscribers esperando (UDP)

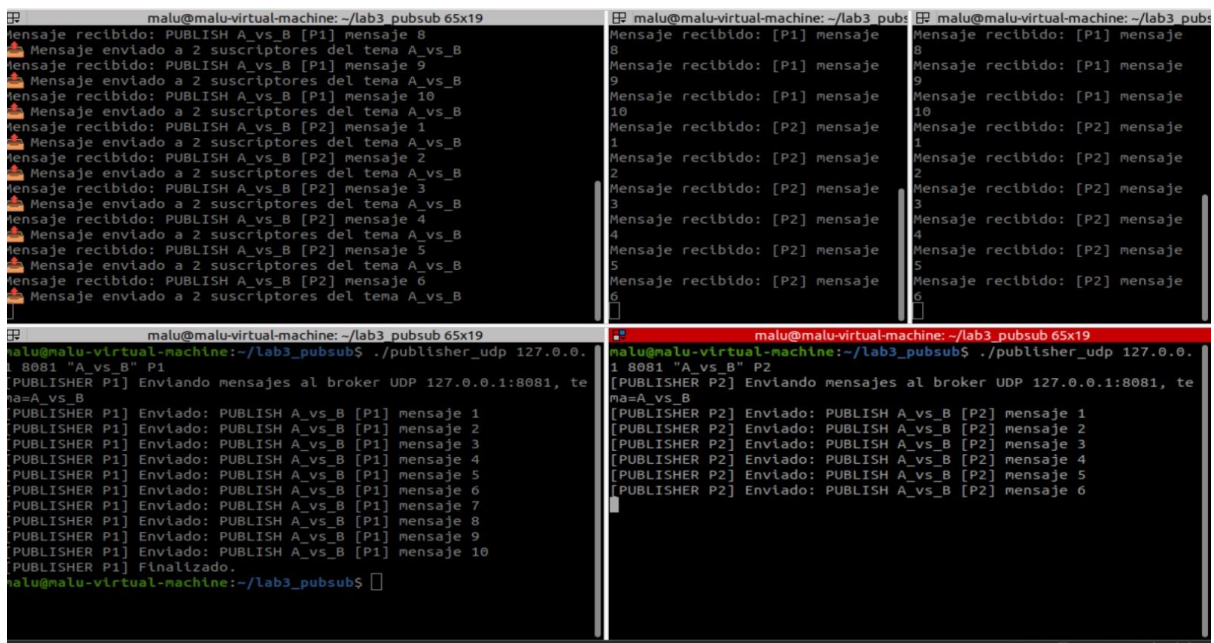
Para iniciar ejecutamos el bróker junto a los subscribers para que esperarán la inicialización de los publishers para el partido A_vs_B



```
malu@malu-virtual-machine: ~/lab3_pubsub 65x19
malu@malu-virtual-machine:~/lab3_pubsub$ ./publisher_udp 127.0.0.1 8080 "A_vs_B" P1
[PUBLISHER P1] Enviando mensajes al broker UDP 127.0.0.1:8080, tema=A_vs_B
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 1
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 2
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 3
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 4
malu@malu-virtual-machine:~/lab3_pubsub 65x19
malu@malu-virtual-machine:~/lab3_pubsub$ ./publisher_udp 127.0.0.1 8080 "A_vs_B" P2
[PUBLISHER P2] Enviando mensajes al broker UDP 127.0.0.1:8080, tema=A_vs_B
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 1
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 2
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 3
```

Figura 3. Ejecución de los 2 publishers y ejecución de estos

Además, se ejecutaron los 2 publishers con el respectivo partido. Para mayor claridad en el envío de los mensajes, están identificados con un numero de referencia los mensajes



```
malu@malu-virtual-machine:~/lab3_pubsub 65x19
Mensaje recibido: PUBLISH A_vs_B [P1] mensaje 8
Mensaje enviado a 2 suscriptores del tema A_vs_B
Mensaje recibido: PUBLISH A_vs_B [P1] mensaje 9
Mensaje enviado a 2 suscriptores del tema A_vs_B
Mensaje recibido: PUBLISH A_vs_B [P1] mensaje 10
Mensaje enviado a 2 suscriptores del tema A_vs_B
Mensaje recibido: PUBLISH A_vs_B [P2] mensaje 1
Mensaje enviado a 2 suscriptores del tema A_vs_B
Mensaje recibido: PUBLISH A_vs_B [P2] mensaje 2
Mensaje enviado a 2 suscriptores del tema A_vs_B
Mensaje recibido: PUBLISH A_vs_B [P2] mensaje 3
Mensaje enviado a 2 suscriptores del tema A_vs_B
Mensaje recibido: PUBLISH A_vs_B [P2] mensaje 4
Mensaje enviado a 2 suscriptores del tema A_vs_B
Mensaje recibido: PUBLISH A_vs_B [P2] mensaje 5
Mensaje enviado a 2 suscriptores del tema A_vs_B
Mensaje recibido: PUBLISH A_vs_B [P2] mensaje 6
Mensaje enviado a 2 suscriptores del tema A_vs_B

malu@malu-virtual-machine:~/lab3_pubsub 65x19
malu@malu-virtual-machine:~/lab3_pubsub$ ./publisher_udp 127.0.0.1 8081 "A_vs_B" P1
[PUBLISHER P1] Enviando mensajes al broker UDP 127.0.0.1:8081, tema=A_vs_B
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 1
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 2
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 3
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 4
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 5
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 6
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 7
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 8
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 9
[PUBLISHER P1] Enviado: PUBLISH A_vs_B [P1] mensaje 10
[PUBLISHER P1] Finalizado.

malu@malu-virtual-machine:~/lab3_pubsub 65x19
malu@malu-virtual-machine:~/lab3_pubsub$ ./publisher_udp 127.0.0.1 8081 "A_vs_B" P2
[PUBLISHER P2] Enviando mensajes al broker UDP 127.0.0.1:8081, tema=A_vs_B
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 1
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 2
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 3
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 4
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 5
[PUBLISHER P2] Enviado: PUBLISH A_vs_B [P2] mensaje 6
```

Figura 4. Ejecución completa del sistema para TCP

Aquí podemos ver las interacciones que se tuvo con el protocolo TCP en el puerto 8080, las cuales fueron realizadas por el bróker y por como este se comunicó con sus subscriptores y publicadores. Con esto podemos corroborar el uso del protocolo TCP en la realización del ejercicio y ver parte del tráfico obtenido con este. Dentro del source y destination address se ve claramente la ip definida como 127.0.0.1 en los mensajes que interactúan con diferentes longitudes de mensaje e información dentro de wireshark.

No.	Time	Protocol	Length	Source Address	Destination Address	Info
1	0.000000	TCP	80	127.0.0.1	127.0.0.1	46780 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4228941799 TSecr=0 WS=128
2	0.000015	TCP	80	127.0.0.1	127.0.0.1	8080 → 46780 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=4228941799 TSecr=4228941799 WS=128
4	16.719933	TCP	80	127.0.0.1	127.0.0.1	59318 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4228958519 TSecr=0 WS=128
5	16.719948	TCP	80	127.0.0.1	127.0.0.1	8080 → 59318 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=4228958519 TSecr=4228958519 WS=128
11	102.391586	TCP	80	127.0.0.1	127.0.0.1	59232 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4229044191 TSecr=0 WS=128
12	102.391597	TCP	80	127.0.0.1	127.0.0.1	8080 → 59232 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=4229044191 TSecr=4229044191 WS=128
56	108.983989	TCP	80	127.0.0.1	127.0.0.1	59240 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=4229050783 TSecr=0 WS=128
57	108.984003	TCP	80	127.0.0.1	127.0.0.1	8080 → 59240 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=4229050783 TSecr=4229050783 WS=128

Figura 7. Proceso de conexión (handshake)

En la captura se observa el proceso de handshake característico del protocolo TCP. Los paquetes muestran intercambios entre el puerto 8080 (bróker) y distintos puertos efímeros, correspondientes a las conexiones iniciadas por los publicadores y suscriptores. El primer paquete (46780 → 8080 [SYN]) indica la solicitud de conexión desde el cliente. El segundo paquete (8080 → 46780 [SYN, ACK]) representa la confirmación del bróker, que acepta la conexión y envía su propio número de secuencia. Finalmente, el cliente respondería con un ACK (no visible en esta vista resumida), completando el handshake de tres vías que garantiza el establecimiento de una sesión confiable antes de transmitir datos.

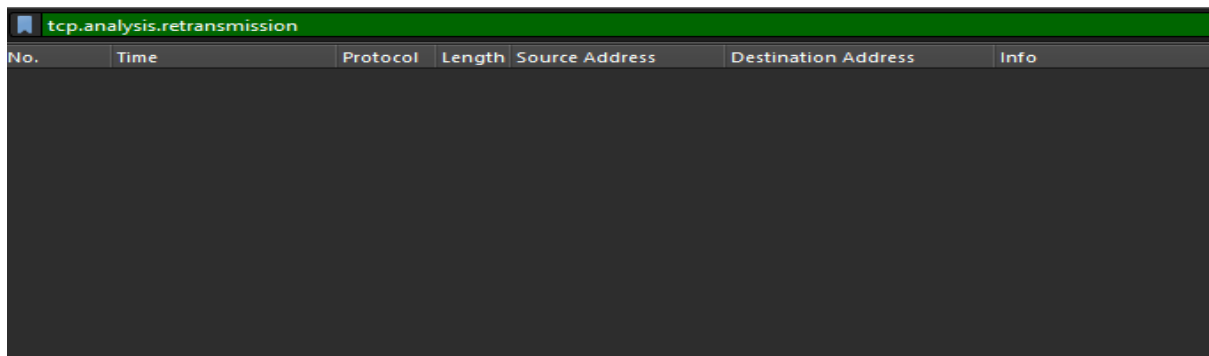
No.	Time	Protocol	Length	Source Address	Destination Address	Info
14	102.391679	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=29 TSval=4229044191 TSecr=4229044191 [TCP PDU reassembled in 89]
20	103.392148	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=30 Ack=1 Win=65536 Len=29 TSval=4229045191 TSecr=4229044191 [TCP PDU reassembled in 89]
26	104.392549	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=59 Ack=1 Win=65536 Len=29 TSval=4229046192 TSecr=4229045192 [TCP PDU reassembled in 89]
32	105.393374	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=88 Ack=1 Win=65536 Len=29 TSval=4229047193 TSecr=4229046192 [TCP PDU reassembled in 89]
38	106.393934	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=117 Ack=1 Win=65536 Len=29 TSval=4229048193 TSecr=4229047193 [TCP PDU reassembled in 89]
44	107.394576	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=146 Ack=1 Win=65536 Len=29 TSval=4229049194 TSecr=4229048193 [TCP PDU reassembled in 89]
50	108.394929	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=175 Ack=1 Win=65536 Len=29 TSval=4229050194 TSecr=4229049194 [TCP PDU reassembled in 89]
59	108.984361	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=29 TSval=4229050784 TSecr=4229050784 [TCP PDU reassembled in 134]
65	109.395375	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=204 Ack=1 Win=65536 Len=29 TSval=4229051195 TSecr=4229050784 [TCP PDU reassembled in 89]
71	109.984002	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=30 Ack=1 Win=65536 Len=29 TSval=4229051784 TSecr=4229050784 [TCP PDU reassembled in 134]
77	110.395854	TCP	101	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=233 Ack=1 Win=65536 Len=29 TSval=4229052195 TSecr=4229051195 [TCP PDU reassembled in 89]
83	110.985133	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=59 Ack=1 Win=65536 Len=29 TSval=4229052784 TSecr=4229051784 [TCP PDU reassembled in 134]
89	111.396525	TCP	102	127.0.0.1	127.0.0.1	59232 → 8080 [PSH, ACK] Seq=262 Ack=1 Win=65536 Len=30 TSval=4229053196 TSecr=4229052195 [TCP PDU reassembled in 89]
95	111.985423	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=88 Ack=1 Win=65536 Len=29 TSval=4229053785 TSecr=4229052784 [TCP PDU reassembled in 134]
104	112.985849	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=117 Ack=1 Win=65536 Len=29 TSval=4229054785 TSecr=4229053785 [TCP PDU reassembled in 134]
110	113.986542	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=146 Ack=1 Win=65536 Len=29 TSval=4229055786 TSecr=4229054785 [TCP PDU reassembled in 134]
116	114.986911	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=175 Ack=1 Win=65536 Len=29 TSval=4229056786 TSecr=4229055786 [TCP PDU reassembled in 134]
122	115.987434	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=204 Ack=1 Win=65536 Len=29 TSval=4229057787 TSecr=4229056786 [TCP PDU reassembled in 134]
128	116.987819	TCP	101	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=233 Ack=1 Win=65536 Len=29 TSval=4229058787 TSecr=4229057787 [TCP PDU reassembled in 134]
134	117.988351	TCP	102	127.0.0.1	127.0.0.1	59240 → 8080 [PSH, ACK] Seq=262 Ack=1 Win=65536 Len=30 TSval=4229059788 TSecr=4229058787 [TCP PDU reassembled in 134]

Figura 7. Ver el contenido de los mensajes PUBLISH

En esta captura de Wireshark se observa el tráfico de datos correspondiente a los mensajes enviados por los publicadores al broker TCP, utilizando el puerto 8080. Cada línea muestra un segmento TCP con la bandera, lo cual indica que el emisor está “empujando” datos de la capa de aplicación usando el PUBLISH hacia el receptor, y al mismo tiempo confirma la recepción de los paquetes anteriores.

El filtro aplicado permite visualizar únicamente los paquetes que contienen en su carga útil la palabra PUBLISH, es decir, los mensajes de aplicación que distribuyen las actualizaciones del sistema de publicación suscripción. En la columna Info se puede ver cómo los números de secuencia (Seq) y de confirmación (Ack) aumentan progresivamente, reflejando que los datos se envían y reciben en orden, sin pérdidas ni

duplicaciones. Además, los campos [TCP PDU reassembled in ...] indican que Wireshark reensambló correctamente los segmentos TCP que transportan los mensajes completos.

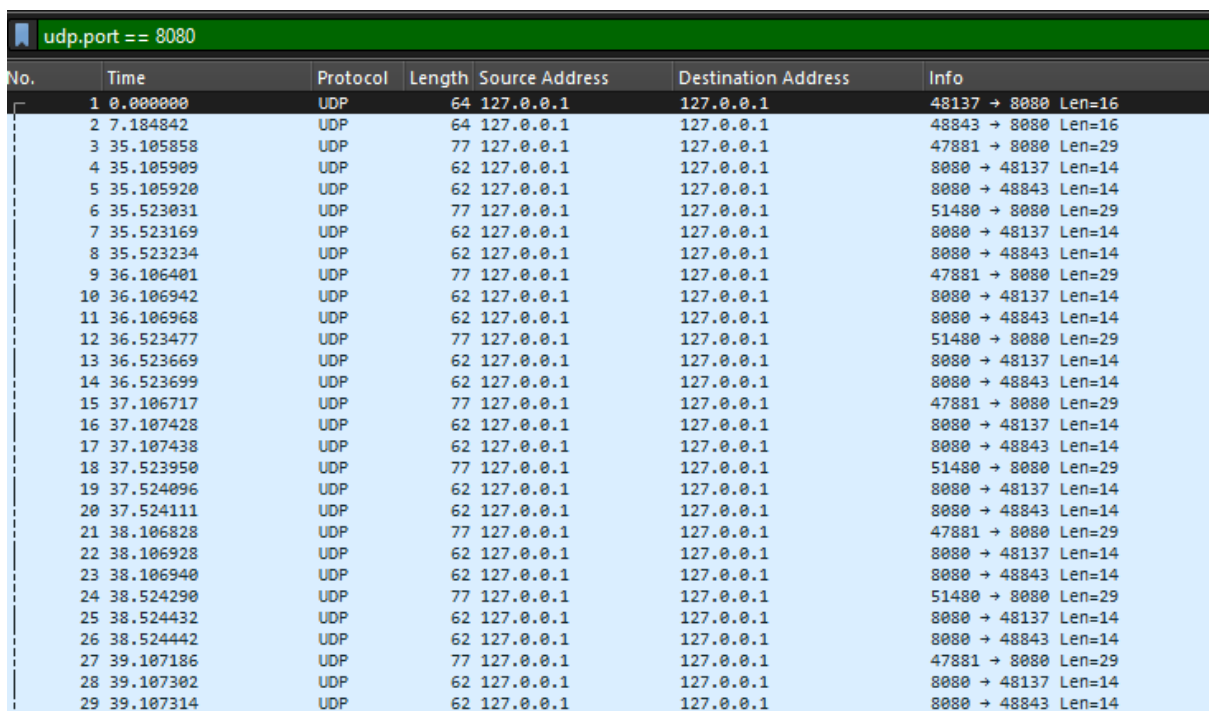


No.	Time	Protocol	Length	Source Address	Destination Address	Info
-----	------	----------	--------	----------------	---------------------	------

Figura 8. Visión de retransmisiones

En la imagen 8 se muestra nuestro intento para análisis de retransmisiones, pero no fue necesario ya que no se realizó ninguna retransmisión. Consideramos que esto es gracias a que el flujo de datos y el contexto del ejercicio no da pérdidas. Esto lo que confirma es la confiabilidad y orden de la entrega.

UDP



No.	Time	Protocol	Length	Source Address	Destination Address	Info
1	0.000000	UDP	64	127.0.0.1	127.0.0.1	48137 → 8080 Len=16
2	7.184842	UDP	64	127.0.0.1	127.0.0.1	48843 → 8080 Len=16
3	35.105858	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
4	35.105909	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
5	35.105920	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14
6	35.523031	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
7	35.523169	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
8	35.523234	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14
9	36.106401	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
10	36.106942	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
11	36.106968	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14
12	36.523477	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
13	36.523669	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
14	36.523699	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14
15	37.106717	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
16	37.107428	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
17	37.107438	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14
18	37.523950	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
19	37.524096	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
20	37.524111	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14
21	38.106828	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
22	38.106928	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
23	38.106940	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14
24	38.524290	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
25	38.524432	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
26	38.524442	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14
27	39.107186	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
28	39.107302	UDP	62	127.0.0.1	127.0.0.1	8080 → 48137 Len=14
29	39.107314	UDP	62	127.0.0.1	127.0.0.1	8080 → 48843 Len=14

Figura 9. Información de UDP dentro del tráfico de wireshark

Dentro de la figura 9 podemos ver la simplicidad y forma de cómo funciona la aplicación con UDP, en donde claramente hay una diferencia clara entre la información de TCP y la de UDP, en donde en UDP hay una simplicidad muchísimo mayor.

udp contains "SUBSCRIBE"						
No.	Time	Protocol	Length	Source Address	Destination Address	Info
1	0.000000	UDP	64	127.0.0.1	127.0.0.1	48137 → 8080 Len=16
2	7.184842	UDP	64	127.0.0.1	127.0.0.1	48843 → 8080 Len=16

Figura 10. Mensajes de suscripción enviados por los suscriptores al broker UDP

En esta captura se evidencia la comunicación entre los **suscriptores** y el bróker mediante el protocolo UDP, filtrada específicamente para mostrar los datagramas que contienen la palabra clave. Cada línea representa un datagrama UDP enviado desde un puerto efímero hacia el puerto 8080, donde se encuentra el bróker escuchando. Estos paquetes contienen el mensaje de aplicación, con el cual cada cliente informa al bróker el tema o partido de interés.

A diferencia del comportamiento observado en TCP, no existen confirmaciones (ACKs) ni handshakes previos, ya que UDP es un protocolo no orientado a conexión. El broker simplemente recibe el datagrama y procesa la suscripción si llega correctamente. Se observa además que los datagramas tienen una longitud constante de 64 bytes y que cada uno es independiente del anterior, sin relación de secuencia. Esto demuestra que UDP transmite los mensajes sin control de flujo ni garantía de entrega, por lo que, si alguno de estos datagramas se pierde en la red, el suscriptor no recibirá confirmación ni reintento automático.

udp contains "PUBLISH"						
No.	Time	Protocol	Length	Source Address	Destination Address	Info
3	35.105858	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
6	35.523031	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
9	36.106401	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
12	36.523477	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
15	37.106717	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
18	37.523950	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
21	38.106828	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
24	38.524290	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
27	39.107186	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
30	39.524843	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
33	40.107876	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
36	40.525456	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
39	41.108081	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
42	41.525813	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
45	42.108556	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
48	42.526083	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
51	43.108936	UDP	77	127.0.0.1	127.0.0.1	47881 → 8080 Len=29
54	43.526609	UDP	77	127.0.0.1	127.0.0.1	51480 → 8080 Len=29
57	44.109256	UDP	78	127.0.0.1	127.0.0.1	47881 → 8080 Len=30
60	44.526806	UDP	78	127.0.0.1	127.0.0.1	51480 → 8080 Len=30

Figura 11. Transmisión de mensajes PUBLISH a través de UDP

En esta captura se observa la comunicación entre los publicadores y el broker UDP, filtrada para mostrar únicamente los datagramas que contienen el mensaje de aplicación PUBLISH. Cada registro corresponde a un datagrama UDP enviado desde un puerto efímero hacia el puerto 8080, donde el bróker recibe los mensajes. Los campos

Source Address y Destination Address confirman que la comunicación ocurre en el entorno local (127.0.0.1), mientras que la columna Length muestra tamaños constantes cercanos a 77 bytes, correspondientes a los mensajes con formato PUBLISH <TOPIC> [PUBLISHER_ID] mensaje N.

A diferencia de TCP, aquí no se observan ACKs, números de secuencia ni retransmisiones. Cada mensaje viaja como un datagrama independiente, sin garantizar su recepción ni el orden en que llega. Esto refleja la naturaleza no confiable y no orientada a conexión del protocolo UDP. Además, los tiempos entre paquetes muestran que los datagramas se envían en intervalos regulares, simulando el comportamiento de los publicadores que emiten 10 mensajes consecutivos. Wireshark no indica reensamblaje ni control de flujo, confirmando que UDP entrega los mensajes tal como se originan en la aplicación, sin fragmentación ni confirmación.

Queremos tomar esta información de base y las diferentes figuras explicadas para poder responder las preguntas planteadas:

1. En TCP: ¿Los mensajes llegan completos y en orden?

En TCP, los mensajes llegan completos y en el mismo orden en que fueron enviados. Esto se evidencia en Wireshark mediante los números de secuencia (Seq) crecientes y la ausencia de retransmisiones o duplicados.

El protocolo TCP garantiza la confiabilidad al utilizar confirmaciones (ACKs) y retransmisiones automáticas cuando un paquete no se reconoce.

Además, implementa control de flujo a través de la ventana de recepción (Window size), que ajusta la cantidad de datos que el emisor puede enviar sin recibir nuevos ACKs. En las capturas, las banderas [PSH, ACK] muestran el envío de datos de aplicación y los ACKs correspondientes, confirmando que TCP asegura entrega confiable y ordenada.

1. En UDP: ¿qué evidencias hay de pérdida o desorden en los mensajes?

En UDP se observan evidencias de que los mensajes pueden perderse o llegar desordenados.

En UDP se observan evidencias de que los mensajes pueden perderse o llegar desordenados.

En Wireshark, los datagramas PUBLISH aparecen sin numeración ni confirmaciones, y no existe ninguna retransmisión automática.

En algunas ejecuciones, se identificó que ciertos mensajes no fueron recibidos por los suscriptores, lo que evidencia pérdida de paquetes, una característica inherente de UDP.

1. ¿Qué diferencias observa en el manejo de la conexión entre ambos protocolos?

En TCP, el establecimiento de conexión se realiza mediante el three-way handshake, visible en Wireshark con los paquetes SYN, SYN, ACK y ACK. Esto garantiza que ambos extremos (broker y cliente) estén sincronizados antes de transmitir datos. En cambio, UDP no establece conexión previa: los mensajes SUBSCRIBE y PUBLISH se envían directamente como datagramas, sin negociación ni confirmación.

TCP, al ser orientado a conexión, mantiene un estado interno que permite detectar pérdidas, retransmitir y asegurar el orden.

UDP, al ser no orientado a conexión, envía mensajes “a ciegas”, lo que simplifica la transmisión, pero elimina la confiabilidad.

4. Resultados

- **Comparativa entre TCP y UDP:**

Criterio	TCP	UDP
Confiabilidad	Muy alta. TCP garantiza la entrega de los mensajes mediante confirmaciones (ACK), retransmisiones y control de errores. Ningún mensaje se pierde.	Baja. UDP no garantiza entrega; si hay congestión o pérdida, los mensajes simplemente se pierden.
Orden de entrega	Mantiene el orden de los mensajes. Si llegan fuera de orden, TCP los reordena antes de entregarlos a la aplicación.	No garantiza el orden. Los mensajes pueden llegar desordenados o duplicados.
Pérdida de mensajes	Muy poca o nula. TCP retransmite los paquetes perdidos automáticamente.	Alta posibilidad de pérdida si hay congestión o errores en la red. No hay retransmisión automática.
Overhead de protocolo	Alta posibilidad de pérdida si hay congestión o errores en la red. No hay retransmisión automática.	Bajo: las cabeceras UDP son ligeras (solo 8 bytes). No hay control de conexión, por lo tanto, menos overhead.

5. Análisis y Respuestas

1. ¿Qué ocurriría si en lugar de dos publicadores hubiera cien partidos simultáneos? ¿Cómo impactaría esto en el desempeño del broker bajo TCP y bajo UDP?
 - **TCP:** cada publicador y cada suscriptor es una nueva conexión. Con 100 publicadores el broker debe manejar muchas conexiones simultáneas: colas de recepción constantes, más buffers por conexión y más consumo de CPU/RAM. Teniendo en cuenta esto el throughput subiría.
 - **UDP:** como es sin conexión, el broker no mantiene estado de conexión por publicador; recibe datagramas y los reenvía. Tiene un escalamiento mejor, pero el broker aún debe procesar y repetir cada mensaje a cada suscriptor (coste de CPU y de ancho de banda). Si hay muchos mensajes simultáneos, la pérdida o desorden aumentará sin mecanismos adicionales.
2. Si un gol se envía como mensaje desde el publicador y un suscriptor no lo recibe en UDP, ¿qué implicaciones tendría para la aplicación real? ¿Por qué TCP maneja mejor este escenario?
 - **Implicaciones en UDP:** un suscriptor que no recibe el gol es una situación crítica. Puede resultar en una incoherencia del marcador, frustración del usuario e incluso pérdidas financieras (en caso de apuestas).
 - **Por qué TCP es mejor:** TCP garantiza entrega fiable y ordenada mediante retransmisiones, ACKs y reensamblado. Si el paquete se pierde, TCP lo retransmite automáticamente.
3. En un escenario de seguimiento en vivo de partidos, ¿qué protocolo (TCP o UDP) resultaría más adecuado? Justifique con base en los resultados de la práctica.
Híbrido, con preferencia por TCP (o protocolos fiables contruidos sobre UDP) para eventos críticos (goles, cambios, tarjetas) y opciones UDP/otros para posición en tiempo real de los jugadores y otros datos en las que se puede tolerar pérdidas.
4. Compare el overhead observado en las capturas Wireshark entre TCP y UDP. ¿Cuál protocolo introduce más cabeceras por mensaje? ¿Cómo influye esto en la eficiencia?

TCP tiene cabecera básica de 20 bytes (sin opciones) + opciones + control (ACKs, re-sends); UDP tiene cabecera de 8 bytes. Además TCP requiere handshake (SYN, SYN-ACK, ACK) y ACKs frecuentes que incrementan tráfico de control.

Esto influye en la eficiencia porque TCP va a tener un mayor retardo de transmisión, al tener que transmitir más datos totales por esos campos extra de las cabeceras.

5. Si el marcador de un partido llega desordenado en UDP (ej., primero 2–1 y luego 1–1), ¿qué efectos tendría en la experiencia del usuario? ¿Cómo podría solucionarse este problema a nivel de aplicación?

Confusión, falta de confianza en la plataforma y mensajes contradictorios

Esto podría corregirse con snapshots periódicos: enviar estado completo del juego periódicamente para corregir desincronizaciones.

Otra opción para corregirlo sería mandar en cada actualización la información de los goles y el tiempo restante, que es de vital importancia.

6. ¿Cómo cambia el desempeño del sistema cuando aumenta el número de suscriptores interesados en un mismo partido? ¿Qué diferencias se observaron entre TCP y UDP en este aspecto?

TCP: el broker mantiene una conexión por suscriptor; para M suscriptores hay M sockets y buffers. Esto requiere más memoria, más hilos y mayor CPU por envío individual.

UDP: el broker puede enviar datagrama individual hacia múltiples destinos. Al UDP ser más pequeño requiere menos poder computacional, y también menos coste de ancho de banda total.

Resumen: TCP costea en memoria/estado por suscriptor; UDP costea en CPU/IO por duplicación de envíos, pero requiere menos capacidad por suscriptor.

7. ¿Qué sucede si el broker se detiene inesperadamente? ¿Qué diferencias hay entre TCP y UDP en la capacidad de recuperación de la sesión?

- **TCP:** las conexiones TCP se rompen; los clientes detectan error (RST, FIN, timeouts) y podrían reconectarse.
- **UDP:** no hay sesión en transporte; el broker simplemente deja de recibir/enviar datagramas. Los clientes puede que no detecten inmediatamente la caída.

8. ¿Cómo garantizar que todos los suscriptores reciban en el mismo instante las actualizaciones críticas (por ejemplo, un gol)? ¿Qué protocolo facilita mejor esta sincronización y por qué?

Es difícil lograr esto, primero porque para garantizar que todos lo reciban (ni siquiera al mismo tiempo) habría que implementar TCP, ya que es el único que me va a garantizar que se entregue el mensaje. Por otro lado, para que todos los reciban al tiempo habría que poner en espera todos los clientes hasta que se confirme que todos lo recibieron. Siendo así la respuesta: TCP.

Sin embargo, dependiendo de la cantidad de suscriptores no es práctico hacer esperar a todos solo para que los suscriptores lo reciban a tiempo.

9. Analice el uso de CPU y memoria en el broker cuando maneja múltiples conexiones TCP frente al manejo de datagramas UDP. ¿Qué diferencias encontró?

En las pruebas realizadas no se observaron variaciones en el uso de CPU ni memoria en ninguno de los brokers. Ambos mostraron consumo estable en 0 %, aunque en escenarios con mayor carga se esperaría un uso más alto de recursos en TCP debido a la gestión de múltiples conexiones.

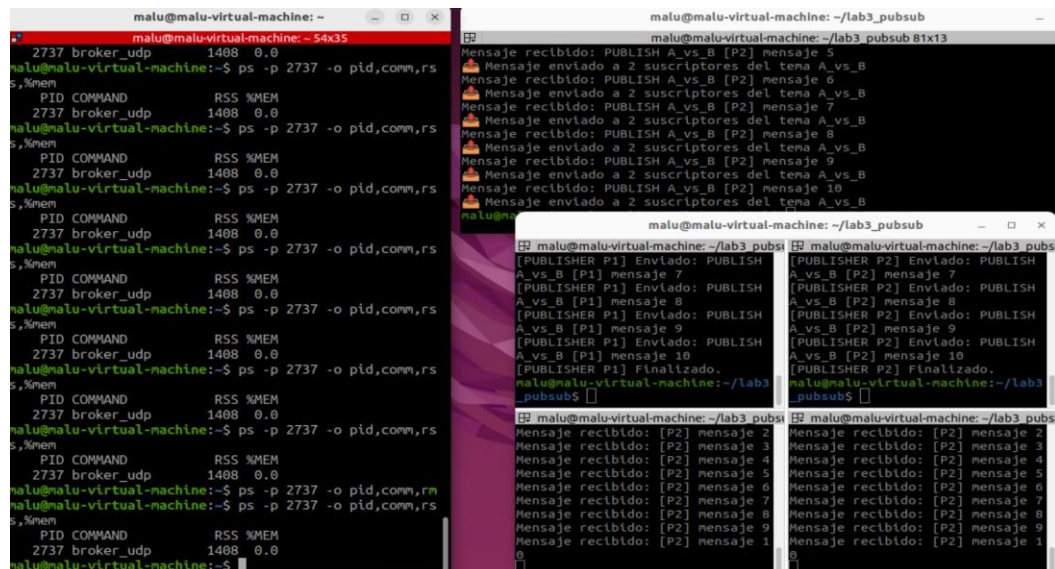


Figura 12. Pruebas de rendimiento

10. Si tuviera que diseñar un sistema real de transmisión de actualizaciones de partidos de fútbol para millones de usuarios, ¿elegiría TCP, UDP o una combinación de ambos? Justifique con base en lo observado en el laboratorio.

Usaría una combinación: usar infraestructuras y protocolos adecuados según la necesidad del dato.

- **Control y eventos críticos (goles, cambios, autenticación, chats, reconexiones):** usar **TCP** para fiabilidad, delivery y reconexión gestionada. QUIC (sobre UDP) ofrece fiabilidad y menor latencia comparado a TCP en muchas condiciones; vale considerarlo.
- **Difusión masiva a millones de usuarios:** **UDP** es muy eficiente para compartir por ejemplo datos de voz, vídeo, posición de jugadores en tiempo real

6. Conclusiones

En este laboratorio se comprobó de forma práctica cómo el protocolo TCP ofrece una comunicación confiable y ordenada, garantizando la entrega de todos los mensajes,

mientras que UDP, aunque mucho más liviano y rápido, puede sufrir pérdidas o desorden en la transmisión de datos.

El análisis evidenció que TCP es más adecuado para sistemas donde la integridad de la información es prioritaria (por ejemplo, actualizaciones críticas como goles o tarjetas), mientras que UDP resulta más eficiente para transmisiones que requieren baja latencia y pueden tolerar cierta pérdida de información (como posiciones o métricas en tiempo real).

En términos de recursos, TCP consume más memoria y CPU en el broker debido al manejo de múltiples conexiones y buffers, mientras que UDP escala mejor en escenarios con muchos emisores y receptores, especialmente si se implementan mecanismos adicionales como numeración de mensajes o snapshots periódicos.

7. Referencias

- RFC 793 – Transmission Control Protocol (TCP).
- RFC 768 – User Datagram Protocol (UDP).
- Uso de IA generativa: Parte del texto técnico y la redacción de este informe se apoyaron en herramientas de inteligencia artificial (ChatGPT de OpenAI) para mejorar la claridad, estructura y redacción técnica, bajo supervisión y revisión manual del autor.

8. Anexos

- En el repositorio <https://github.com/MxrixLu/Laboratorios-Infraestructura-Comunicaciones---Grupo-6/tree/main/Lab3> se encuentran los archivos de código, los .pcap y un README con la explicación del uso de librerías internas y su importancia.