

|  
Universidad de los Andes  
Departamento de Ingeniería de Sistemas y Computación



## **Laboratorio #3 – Análisis de capa de transporte y sockets**

**ISIS 3204 – Infraestructura de comunicaciones**

**Profesora: Nathalia Alexandra Quiroga Alfaro**

### **Grupo 2**

Daniel Bolívar Bernal – 202310329

Silvana Echeverry - 202310470

Miguel Ángel Velandia – 202312487

# Semestre 2025-2

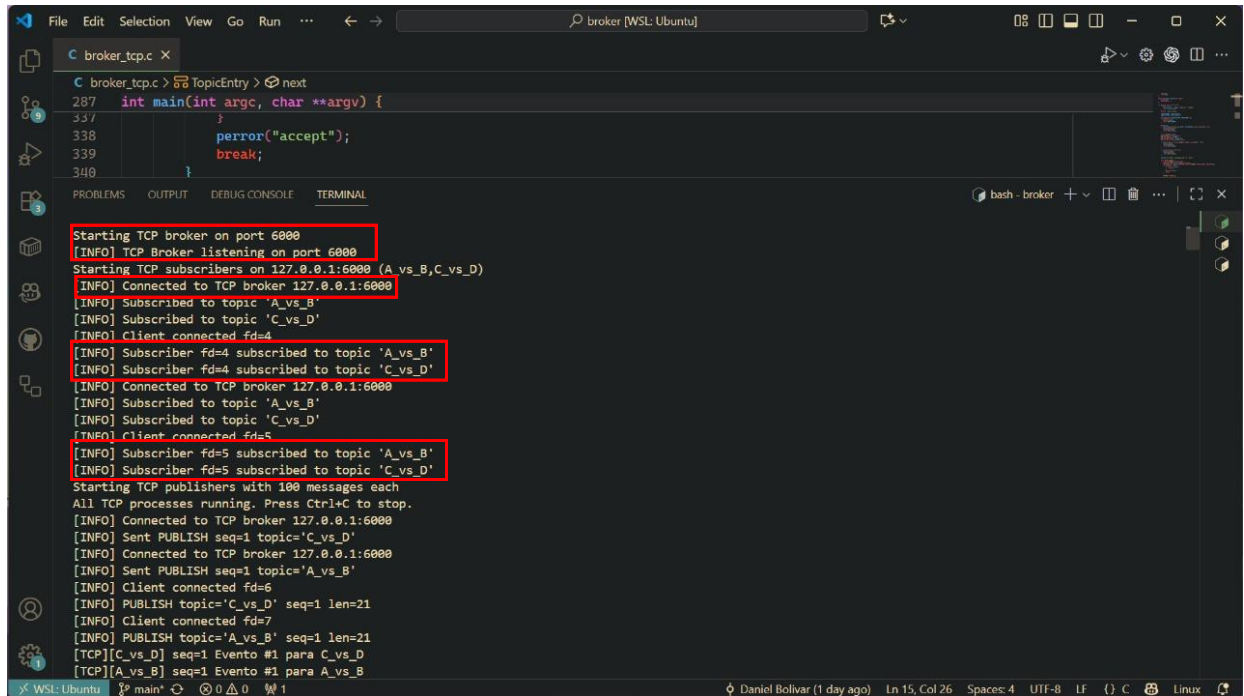
## Tabla de Contenido:

1. Pruebas y comparación
  - a. En TCP: ¿los mensajes llegan completos y en orden? ¿Cómo maneja TCP la confiabilidad y el control de flujo?
  - b. En UDP: ¿qué evidencias hay de pérdida o desorden en los mensajes?
  - c. ¿Qué diferencias observa en el manejo de la conexión entre ambos protocolos?
2. Comparación del desempeño
3. Respuesta a preguntas de análisis
  - a. ¿Qué ocurriría si en lugar de dos publicadores (partidos transmitidos) hubiera cien partidos simultáneos? ¿Cómo impactaría esto en el desempeño del broker bajo TCP y bajo UDP?
  - b. 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?
  - c. 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.
  - d. 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?
  - e. Si el marcador de un partido llega desordenado en UDP (por ejemplo, primero se recibe el 2-1 y luego el 1-1), ¿qué efectos tendría en la experiencia del usuario? ¿Cómo podría solucionarse este problema a nivel de aplicación?
  - f. ¿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?
  - g. ¿Qué sucede si el broker se detiene inesperadamente? ¿Qué diferencias hay entre TCP y UDP en la capacidad de recuperación de la sesión?
  - h. ¿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?
  - i. Analice el uso de CPU y memoria en el broker cuando maneja múltiples conexiones TCP frente al manejo de datagramas UDP. ¿Qué diferencias encontró?
  - j. 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.
4. Bibliografía.

## 1. Pruebas y comparación

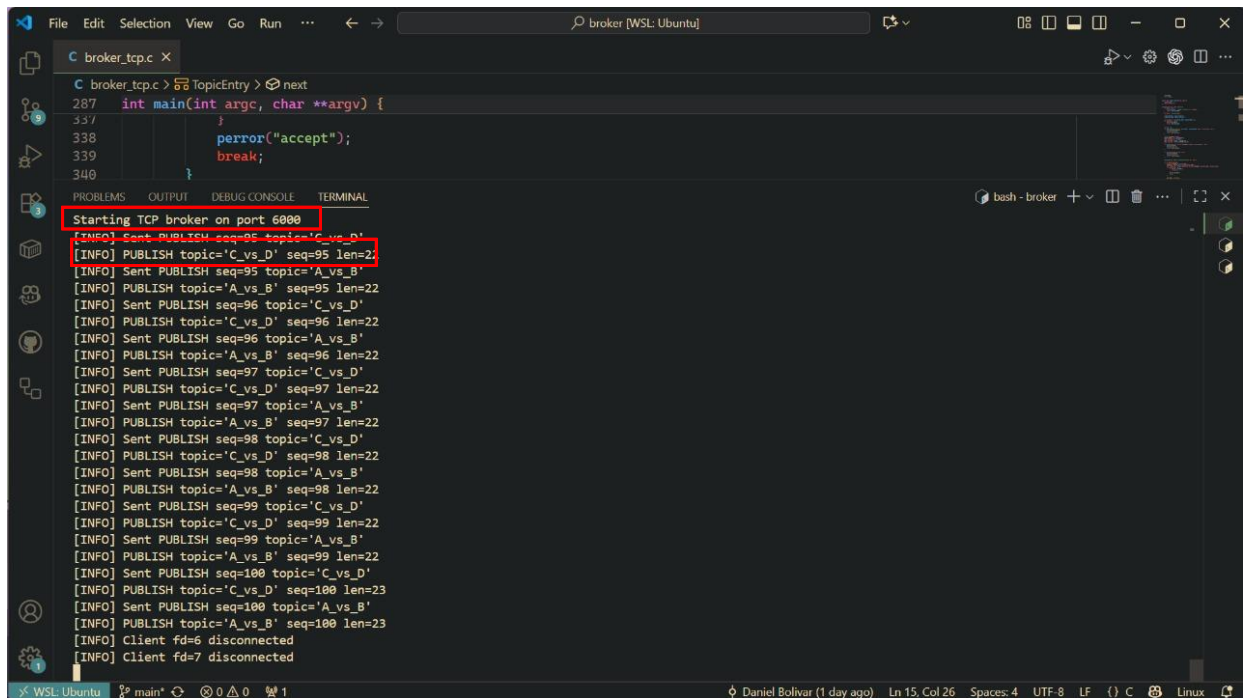
**En TCP: ¿los mensajes llegan completos y en orden? ¿Cómo maneja TCP la confiabilidad y el control de flujo?**

Capturas de inicialización de TCP con 1 bróker, 2 publicadores y 2 suscriptores.



```
File Edit Selection View Go Run ... broker [WSL: Ubuntu]
C broker_tcp.c X
C broker_tcp.c > TopicEntry > next
287 int main(int argc, char **argv) {
337 }
338 perror("accept");
339 break;
340 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Starting TCP broker on port 6000
[INFO] TCP Broker listening on port 6000
Starting TCP subscribers on 127.0.0.1:6000 (A_vs_B,C_vs_D)
[INFO] Connected to TCP broker 127.0.0.1:6000
[INFO] Subscribed to topic 'A_vs_B'
[INFO] Subscribed to topic 'C_vs_D'
[INFO] Client connected fd=4
[INFO] Subscriber fd=4 subscribed to topic 'A_vs_B'
[INFO] Subscriber fd=4 subscribed to topic 'C_vs_D'
[INFO] Connected to TCP broker 127.0.0.1:6000
[INFO] Subscribed to topic 'A_vs_B'
[INFO] Subscribed to topic 'C_vs_D'
[INFO] Client connected fd=5
[INFO] Subscriber fd=5 subscribed to topic 'A_vs_B'
[INFO] Subscriber fd=5 subscribed to topic 'C_vs_D'
Starting TCP publishers with 100 messages each
All TCP processes running. Press Ctrl+C to stop.
[INFO] Connected to TCP broker 127.0.0.1:6000
[INFO] Sent PUBLISH seq=1 topic='C_vs_D'
[INFO] Connected to TCP broker 127.0.0.1:6000
[INFO] Sent PUBLISH seq=1 topic='A_vs_B'
[INFO] Client connected fd=6
[INFO] PUBLISH topic='C_vs_D' seq=1 len=21
[INFO] Client connected fd=7
[INFO] PUBLISH topic='A_vs_B' seq=1 len=21
[TCP][C_vs_D] seq=1 Evento #1 para C_vs_D
[TCP][A_vs_B] seq=1 Evento #1 para A_vs_B
WSL: Ubuntu main 0 0 1 Daniel Bolivar (1 day ago) Ln 15, Col 26 Spaces: 4 UTF-8 LF {} C Linux
```

Ilustración 1 - Terminal TCP



```
File Edit Selection View Go Run ... broker [WSL: Ubuntu]
C broker_tcp.c X
C broker_tcp.c > TopicEntry > next
287 int main(int argc, char **argv) {
337 }
338 perror("accept");
339 break;
340 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Starting TCP broker on port 6000
[INFO] Sent PUBLISH seq=95 topic='C_vs_D'
[INFO] PUBLISH topic='C_vs_D' seq=95 len=22
[INFO] Sent PUBLISH seq=95 topic='A_vs_B'
[INFO] PUBLISH topic='A_vs_B' seq=95 len=22
[INFO] Sent PUBLISH seq=96 topic='C_vs_D'
[INFO] PUBLISH topic='C_vs_D' seq=96 len=22
[INFO] Sent PUBLISH seq=96 topic='A_vs_B'
[INFO] PUBLISH topic='A_vs_B' seq=96 len=22
[INFO] Sent PUBLISH seq=97 topic='C_vs_D'
[INFO] PUBLISH topic='C_vs_D' seq=97 len=22
[INFO] Sent PUBLISH seq=97 topic='A_vs_B'
[INFO] PUBLISH topic='A_vs_B' seq=97 len=22
[INFO] Sent PUBLISH seq=98 topic='C_vs_D'
[INFO] PUBLISH topic='C_vs_D' seq=98 len=22
[INFO] Sent PUBLISH seq=98 topic='A_vs_B'
[INFO] PUBLISH topic='A_vs_B' seq=98 len=22
[INFO] Sent PUBLISH seq=99 topic='C_vs_D'
[INFO] PUBLISH topic='C_vs_D' seq=99 len=22
[INFO] Sent PUBLISH seq=99 topic='A_vs_B'
[INFO] PUBLISH topic='A_vs_B' seq=99 len=22
[INFO] Sent PUBLISH seq=100 topic='C_vs_D'
[INFO] PUBLISH topic='C_vs_D' seq=100 len=23
[INFO] Sent PUBLISH seq=100 topic='A_vs_B'
[INFO] PUBLISH topic='A_vs_B' seq=100 len=23
[INFO] Client fd=6 disconnected
[INFO] Client fd=7 disconnected
```

Ilustración 2 - terminal tcp

En las pruebas con TCP podemos notar que todos los mensajes llegan completos y en el mismo orden en que fueron enviados. Esto se debe a la naturaleza de TCP, que es un protocolo orientado a conexión que garantiza esta entrega confiable de datos. En las capturas podemos evidenciar la conexión persistente entre el bróker y los suscriptores mediante los sockets identificados (fd=4 y fd=5) y no se presentan pérdidas ni saltos entre los números de secuencia.

Además, implementa el control de flujo utilizando una ventana deslizante, que ajusta la cantidad de datos enviados según la capacidad del receptor y estado de la red. Por esto es que los mensajes llegan completos, ordenados y sin duplicados.

En la siguiente captura podemos ver la captura de tráfico en wireshark, donde observamos ver el intercambio de segmentos TCP entre los puertos de los publicadores y el bróker en el puerto 6000:



```
File Edit Selection View Go Run ... broker [WSL: Ubuntu]
C broker_tcp.c X
C broker_tcp.c > TopicEntry > next
287 int main(int argc, char **argv) {
337 }
338 perror("accept");
339 break;
340 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash - broker
db@DBPC:~/University/rees/broker$ ./run_udp.sh
Starting UDP broker on port 6000
[INFO] UDP Broker listening on port 6000
Starting UDP subscribers on 127.0.0.1:6000 (A_vs_B,C_vs_D)
[INFO] Sent UDP SUBSCRIBE for 'A_vs_B'
[INFO] UDP subscriber 127.0.0.1:32776 subscribed to 'A_vs_B'
[INFO] Sent UDP SUBSCRIBE for 'C_vs_D'
[INFO] UDP subscriber 127.0.0.1:32776 subscribed to 'C_vs_D'
[INFO] Listening for UDP publications...
[INFO] Sent UDP SUBSCRIBE for 'A_vs_B'
[INFO] UDP subscriber 127.0.0.1:40359 subscribed to 'A_vs_B'
[INFO] Sent UDP SUBSCRIBE for 'C_vs_D'
[INFO] UDP subscriber 127.0.0.1:40359 subscribed to 'C_vs_D'
[INFO] Listening for UDP publications...
Starting UDP publishers with 100 messages each
All UDP processes running. Press Ctrl+C to stop.
[INFO] UDP publisher targeting 127.0.0.1:6000
[INFO] UDP publisher targeting 127.0.0.1:6000
[INFO] Sent UDP PUBLISH seq=1 topic='C_vs_D'
[INFO] Sent UDP PUBLISH seq=1 topic='A_vs_B'
[INFO] UDP PUBLISH topic='A_vs_B' seq=1 len=25
[UDP][A_vs_B] seq=1 Evento UDP #1 para A_vs_B
[INFO] UDP PUBLISH topic='C_vs_D' seq=1 len=25
[UDP][C_vs_D] seq=1 Evento UDP #1 para C_vs_D
[UDP][A_vs_B] seq=1 Evento UDP #1 para A_vs_B
[UDP][C_vs_D] seq=1 Evento UDP #1 para C_vs_D
[INFO] Sent UDP PUBLISH seq=2 topic='A_vs_B'
```

Ilustración 4 – terminal UDP

```
File Edit Selection View Go Run ... broker [WSL: Ubuntu]
C broker_tcp.c X
C broker_tcp.c > TopicEntry > next
287 int main(int argc, char **argv) {
337 }
338 perror("accept");
339 break;
340 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash - broker
db@DBPC:~/University/rees/broker$ ./run_udp.sh
[UDP][C_vs_D] seq=97 Evento UDP #97 para C_vs_D
[INFO] Sent UDP PUBLISH seq=98 topic='A_vs_B'
[INFO] Sent UDP PUBLISH seq=98 topic='C_vs_D'
[INFO] UDP PUBLISH topic='A_vs_B' seq=98 len=26
[UDP][A_vs_B] seq=98 Evento UDP #98 para A_vs_B
[INFO] UDP PUBLISH topic='C_vs_D' seq=98 len=26
[UDP][A_vs_B] seq=98 Evento UDP #98 para A_vs_B
[UDP][C_vs_D] seq=98 Evento UDP #98 para C_vs_D
[UDP][C_vs_D] seq=98 Evento UDP #98 para C_vs_D
[INFO] Sent UDP PUBLISH seq=99 topic='C_vs_D'
[INFO] Sent UDP PUBLISH seq=99 topic='A_vs_B'
[INFO] UDP PUBLISH topic='A_vs_B' seq=99 len=26
[UDP][A_vs_B] seq=99 Evento UDP #99 para A_vs_B
[INFO] UDP PUBLISH topic='C_vs_D' seq=99 len=26
[UDP][A_vs_B] seq=99 Evento UDP #99 para A_vs_B
[UDP][C_vs_D] seq=99 Evento UDP #99 para C_vs_D
[UDP][C_vs_D] seq=99 Evento UDP #99 para C_vs_D
[INFO] Sent UDP PUBLISH seq=100 topic='A_vs_B'
[INFO] Sent UDP PUBLISH seq=100 topic='C_vs_D'
[INFO] UDP PUBLISH topic='A_vs_B' seq=100 len=27
[UDP][A_vs_B] seq=100 Evento UDP #100 para A_vs_B
[INFO] UDP PUBLISH topic='C_vs_D' seq=100 len=27
[UDP][A_vs_B] seq=100 Evento UDP #100 para A_vs_B
[UDP][C_vs_D] seq=100 Evento UDP #100 para C_vs_D
[UDP][C_vs_D] seq=100 Evento UDP #100 para C_vs_D
```

Ilustración 5 - Terminal UDP

En UDP se evidencian posibles pérdidas o desorden de mensajes al observar que los números de secuencia (seq) en los mensajes publicados no siempre llegan de forma consecutiva a los



suscriptores, o algunos números no aparecen en el orden esperado. Esto ocurre porque UDP no garantiza la entrega ni el orden de los datagramas: cada mensaje viaja de forma independiente, sin confirmación de recepción ni retransmisión en caso de pérdida.

En las capturas podemos observar que los mensajes seq=97, seq=98, seq=99, seq=100 pueden no llegar todos o pueden recibirse fuera de orden dependiendo del tráfico y la congestión de red.

En la siguiente captura podemos ver la captura de tráfico en wireshark, donde observamos ver el intercambio de datagramas UDP entre los puertos de los publicadores y el bróker en el puerto 6000:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	70	32776 → 6000 Len=28
2	0.000190	127.0.0.1	127.0.0.1	UDP	70	32776 → 6000 Len=28
3	0.001122	127.0.0.1	127.0.0.1	UDP	70	40359 → 6000 Len=28
4	0.001353	127.0.0.1	127.0.0.1	UDP	70	40359 → 6000 Len=28
5	1.005325	127.0.0.1	127.0.0.1	UDP	95	54599 → 6000 Len=53
6	1.005330	127.0.0.1	127.0.0.1	UDP	95	60637 → 6000 Len=53
7	1.005545	127.0.0.1	127.0.0.1	UDP	95	6000 → 40359 Len=53
8	1.005592	127.0.0.1	127.0.0.1	UDP	95	6000 → 32776 Len=53
9	1.005668	127.0.0.1	127.0.0.1	UDP	95	6000 → 40359 Len=53
10	1.005679	127.0.0.1	127.0.0.1	UDP	95	6000 → 32776 Len=53
11	1.205701	127.0.0.1	127.0.0.1	UDP	95	60637 → 6000 Len=53
12	1.205702	127.0.0.1	127.0.0.1	UDP	95	54599 → 6000 Len=53
13	1.205827	127.0.0.1	127.0.0.1	UDP	95	6000 → 40359 Len=53
14	1.205882	127.0.0.1	127.0.0.1	UDP	95	6000 → 32776 Len=53
15	1.205960	127.0.0.1	127.0.0.1	UDP	95	6000 → 40359 Len=53
16	1.206009	127.0.0.1	127.0.0.1	UDP	95	6000 → 32776 Len=53
17	1.406119	127.0.0.1	127.0.0.1	UDP	95	60637 → 6000 Len=53
18	1.406119	127.0.0.1	127.0.0.1	UDP	95	54599 → 6000 Len=53
19	1.406351	127.0.0.1	127.0.0.1	UDP	95	6000 → 40359 Len=53
20	1.406451	127.0.0.1	127.0.0.1	UDP	95	6000 → 32776 Len=53
21	1.406540	127.0.0.1	127.0.0.1	UDP	95	6000 → 40359 Len=53
22	1.406581	127.0.0.1	127.0.0.1	UDP	95	6000 → 32776 Len=53
23	1.606626	127.0.0.1	127.0.0.1	UDP	95	54599 → 6000 Len=53
24	1.606626	127.0.0.1	127.0.0.1	UDP	95	60637 → 6000 Len=53
25	1.606858	127.0.0.1	127.0.0.1	UDP	95	6000 → 40359 Len=53
26	1.606967	127.0.0.1	127.0.0.1	UDP	95	6000 → 32776 Len=53
27	1.607119	127.0.0.1	127.0.0.1	UDP	95	6000 → 40359 Len=53
28	1.607216	127.0.0.1	127.0.0.1	UDP	95	6000 → 32776 Len=53
29	1.807381	127.0.0.1	127.0.0.1	UDP	95	54599 → 6000 Len=53
30	1.807381	127.0.0.1	127.0.0.1	UDP	95	60637 → 6000 Len=53

> Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0  
 > Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)  
 > Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 > User Datagram Protocol, Src Port: 32776, Dst Port: 6000  
 > Data (28 bytes)

0000  
 0010 00 38 ca 2e 40 00 40 11 72 84 7f 00 00 01 7f 00 .....8..@.r.....  
 0020 00 01 80 08 17 70 00 24 fe 37 00 01 00 01 00 00 .....p.\$..7.....  
 0030 00 00 00 00 01 99 cb 3d 1f b3 00 06 00 00 00 00 .....=.....  
 0040 41 5f 76 73 5f 42 .....A\_vs\_B

Ilustración 6 - Captura wireshark UDP

Se observa que, a diferencia de TCP, en UDP no aparecen paquetes de control ni flags como SYN, ACK o FIN, lo que confirma que este protocolo no establece conexión ni garantiza la entrega de los mensajes. Los datagramas presentan diferentes longitudes y puertos de origen, reflejando transmisiones independientes sin conexión persistente. Además, los números de secuencia muestran saltos y los tiempos de llegada varían, evidenciando pérdida o desorden en los paquetes. Esto es consistente con la naturaleza no confiable de UDP, donde los mensajes se envían sin verificación ni retransmisión.

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

En las pruebas realizadas se evidenció que TCP establece una conexión previa entre el publicador, el bróker y los suscriptores mediante el three-way handshake (SYN, SYN-ACK, ACK). Esto permite mantener una sesión persistente, garantizando la entrega ordenada y confiable de los mensajes. Cada extremo confirma la recepción de los datos y, en caso de que existiera una pérdida, el protocolo retransmitiría entonces los segmentos.

Por el contrario, UDP no establece conexión ni mantiene sesión entre los participantes; cada mensaje se envía como un datagrama independiente sin verificación ni confirmación de entrega. Esto reduce la latencia y el overhead, pero implica que algunos mensajes pueden perderse o llegar desordenados. En resumen, TCP prioriza la confiabilidad y el control, mientras que UDP privilegia la velocidad y simplicidad en el envío.

*Ilustración 8 - Captura wireshark UDP - 10 mensajes*



Tabla comparando el desempeño de TCP contra el desempeño de UDP:

Tabla 1 - Comparación desempeño TCP vs UDP

Criterio	TCP	UDP
<b>Confiabilidad</b>	Alta: establece conexión mediante <i>three-way handshake</i> (SYN, ACK) y confirma cada segmento. Retransmite en caso de pérdida.	Baja: no confirma la recepción de paquetes ni los retransmite si se pierden. Cada datagrama se envía de forma independiente.
<b>Orden de entrega</b>	Garantiza el orden de los mensajes utilizando números de secuencia y acuses de recibo ( <i>Seq/Ack</i> ).	No garantiza orden; los datagramas pueden llegar desordenados o duplicados.
<b>Pérdida de mensajes</b>	No hay pérdida perceptible; el protocolo gestiona retransmisiones automáticas.	Pueden presentarse pérdidas si la red se congestiona o si el receptor no procesa a tiempo los datagramas.
<b>Overhead de cabeceras / protocolo</b>	Mayor: incluye cabeceras de 20 bytes más campos de control (SYN, ACK, FIN), aumentando el tráfico y la latencia.	Menor: solo 8 bytes de cabecera, sin campos de control; menor latencia y mayor eficiencia en velocidad.

Basándonos en los resultados de la tabla podemos concluir que, TCP ofrece confiabilidad y orden con el precio de tener un mayor overhead y latencia, mientras que UDP es mucho más rápido y ligero, pero sin garantías de entrega ni orden.

### 3. Respuesta a preguntas de análisis

#### a. ¿Qué ocurriría si en lugar de dos publicadores (partidos transmitidos) hubiera cien partidos simultáneos? ¿Cómo impactaría esto en el desempeño del broker bajo TCP y bajo UDP?

En el caso de UDP, si el sistema se ampliara a cien publicadores transmitiendo partidos simultáneamente, el broker bajo podría manejar un mayor volumen de mensajes con menor consumo de recursos, ya que este protocolo no requiere mantener conexiones activas ni confirmar la entrega de cada datagrama. Sin embargo, al aumentar el tráfico, también crecería la probabilidad de pérdida, duplicación o desorden de mensajes, afectando la precisión de la información recibida por los suscriptores. Esto significaría entonces en tener actualizaciones incompletas o inconsistentes (por ejemplo, goles o eventos que no llegan a todos los clientes).

En cambio, para TCP, el broker debería mantener cien conexiones activas (una por cada publicador), lo que incrementaría significativamente el uso de memoria y ancho de banda. Aunque el protocolo garantizaría que todos los mensajes llegaran completos y en orden, el alto número de conexiones simultáneas generaría mayor overhead y podría provocar congestión o retardos en la transmisión. Entonces, TCP ofrecería confiabilidad a costa de un menor rendimiento y mayor carga sobre el broker en escenarios masivos.

- b. 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?

Si un gol se envía como mensaje mediante UDP y un suscriptor no lo recibe, la aplicación perdería información crítica, ya que el protocolo no garantiza la entrega ni retransmite los datagramas perdidos. Esto implicaría que el usuario no vería una actualización importante del marcador o recibiría datos inconsistentes respecto a otros suscriptores. Dado que UDP no cuenta con confirmaciones ni control de errores, la aplicación debería implementar mecanismos adicionales a nivel de software (como reenvíos o validaciones) para mitigar estas pérdidas.

Por otro lado, TCP maneja mejor este escenario porque garantiza la entrega confiable de los datos mediante acuses de recibo (ACK) y retransmisión automática en caso de pérdida. Si el mensaje del gol no llegara inicialmente, el protocolo detectaría la falta de confirmación y reenviaría el segmento, asegurando que el evento se muestre a todos los suscriptores. Esto permite mantener la coherencia del marcador y una experiencia más precisa para los usuarios, aunque con un aumento en la latencia respecto a UDP.

- c. 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.

Si lo que se necesita es que la información llegue siempre correcta y en orden, como los goles o las tarjetas, TCP es la mejor opción porque garantiza que los mensajes lleguen completos y en el orden correcto.

En cambio, si lo importante es la rapidez y no pasa nada si se pierde uno que otro mensaje (por ejemplo, los comentarios en tiempo real o estadísticas que se actualizan cada segundo), UDP puede ser más útil porque tiene menos retardo. En la práctica, lo ideal sería usar los dos: TCP o QUIC para lo crítico, y UDP para lo rápido.

- d. 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?

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	70	50921 → 6000 Len=28
2	0.000057	127.0.0.1	127.0.0.1	UDP	70	50921 → 6000 Len=28
3	8.579872	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
4	8.579991	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
5	8.784660	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
6	8.784209	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
7	8.985316	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
8	8.985459	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
9	9.194937	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
10	9.195223	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
11	9.405286	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
12	9.405363	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
13	9.605448	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
14	9.605522	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
15	9.805775	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
16	9.805875	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
17	10.012946	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
18	10.013102	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
19	10.221707	127.0.0.1	127.0.0.1	UDP	95	60833 → 6000 Len=53
20	10.221875	127.0.0.1	127.0.0.1	UDP	95	60800 → 50921 Len=53
21	10.430762	127.0.0.1	127.0.0.1	UDP	96	60833 → 6000 Len=54
22	10.430897	127.0.0.1	127.0.0.1	UDP	96	60800 → 50921 Len=54
23	10.631341	127.0.0.1	127.0.0.1	UDP	96	60833 → 6000 Len=54
24	10.632006	127.0.0.1	127.0.0.1	UDP	96	60800 → 50921 Len=54
25	10.832499	127.0.0.1	127.0.0.1	UDP	96	60833 → 6000 Len=54
26	10.832674	127.0.0.1	127.0.0.1	UDP	96	60800 → 50921 Len=54
27	11.044283	127.0.0.1	127.0.0.1	UDP	96	60833 → 6000 Len=54
28	11.044455	127.0.0.1	127.0.0.1	UDP	96	60800 → 50921 Len=54
29	11.052483	127.0.0.1	127.0.0.1	UDP	95	24974 → 6000 Len=53

Protocol	Length	Info
TCP	74	47014 → 6000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=233080305 TSecr=0 WS=128
TCP	66	47014 → 6000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=233080305 TSecr=233080305
TCP	70	47014 → 6000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=233080305 TSecr=233080305
TCP	66	6000 → 47014 [ACK] Seq=1 Ack=58 Win=65536 Len=0 TSval=233080305 TSecr=233080305
TCP	94	47014 → 6000 [PSH, ACK] Seq=5 Ack=1 Win=65536 Len=28 TSval=233080305 TSecr=233080305
TCP	66	6000 → 47014 [ACK] Seq=1 Ack=33 Win=65536 Len=0 TSval=233080305 TSecr=233080305
TCP	70	47014 → 6000 [PSH, ACK] Seq=33 Ack=1 Win=65536 Len=4 TSval=233080305 TSecr=233080305
TCP	66	6000 → 47014 [ACK] Seq=1 Ack=37 Win=65536 Len=0 TSval=233080305 TSecr=233080305
TCP	94	47014 → 6000 [PSH, ACK] Seq=37 Ack=1 Win=65536 Len=28 TSval=233080305 TSecr=233080305
TCP	66	6000 → 47014 [ACK] Seq=1 Ack=45 Win=65536 Len=0 TSval=233080305 TSecr=233080305
TCP	74	47014 → 6000 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=233080874 TSecr=0 WS=128
TCP	74	6000 → 47014 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=233080874 TSecr=233080874 WS=128
TCP	66	47014 → 6000 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=233080874 TSecr=233080874
TCP	70	47014 → 6000 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=4 TSval=233080874 TSecr=233080874 [TCP PDU reassembled in 41]
TCP	66	6000 → 47014 [ACK] Seq=1 Ack=58 Win=65536 Len=0 TSval=233080874 TSecr=233080874
TCP	115	47014 → 6000 [PSH, ACK] Seq=5 Ack=1 Win=65536 Len=49 TSval=233080874 TSecr=233080874 [TCP PDU reassembled in 41]
TCP	66	6000 → 47014 [ACK] Seq=1 Ack=54 Win=65536 Len=0 TSval=233080874 TSecr=233080874
TCP	70	6000 → 47014 [PSH, ACK] Seq=1 Ack=65 Win=65536 Len=4 TSval=233080874 TSecr=233080305 [TCP PDU reassembled in 21]
TCP	66	47014 → 6000 [ACK] Seq=65 Ack=5 Win=65536 Len=0 TSval=233080874 TSecr=233080874
XII	115	Error: Success
TCP	66	47014 → 6000 [ACK] Seq=65 Ack=54 Win=65536 Len=0 TSval=233080874 TSecr=233080874
TCP	70	47014 → 6000 [PSH, ACK] Seq=54 Ack=1 Win=65536 Len=4 TSval=233080874 TSecr=233080874 [TCP PDU reassembled in 41]
TCP	66	6000 → 47014 [ACK] Seq=1 Ack=58 Win=65536 Len=0 TSval=233080874 TSecr=233080874
TCP	115	47014 → 6000 [PSH, ACK] Seq=58 Ack=1 Win=65536 Len=49 TSval=233080874 TSecr=233080874 [TCP PDU reassembled in 42]
TCP	66	6000 → 47014 [ACK] Seq=1 Ack=107 Win=65536 Len=0 TSval=233080874 TSecr=233080874
TCP	70	6000 → 47014 [PSH, ACK] Seq=54 Ack=45 Win=65536 Len=4 TSval=233080874 TSecr=233080874 [TCP PDU reassembled in 28]
TCP	66	47014 → 6000 [ACK] Seq=65 Ack=58 Win=65536 Len=0 TSval=233080874 TSecr=233080874
XII	115	Event: c:\unknown eventCode: 69; s-Event: KeyPress

Ilustración 9 - Comparación del tamaño y cabeceras en TCP (derecha) y UDP (izquierda) según Wireshark. TCP muestra campos de control (SYN, ACK, Seq), mientras que UDP solo indica longitud y puertos.

En las capturas se ve que los paquetes UDP pesan entre 70 y 96 bytes, mientras que los TCP están entre 66 y 74 bytes. En algunos casos, los paquetes TCP alcanzaron longitudes de **115 bytes**, lo

que indica que el segmento llevaba datos de la aplicación (no solo control). Aunque parecen parecidos, TCP tiene una cabecera más grande (unos 20 bytes) con campos de control como SYN, ACK y Sequence Number, mientras que UDP solo usa 8 bytes. Esto hace que TCP sea más pesado pero confiable, y UDP más liviano y rápido, aunque sin control de entrega. En resumen, TCP sacrifica eficiencia por confiabilidad, y UDP hace lo contrario.

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

El usuario podría ver resultados incoherentes, como si el marcador retrocediera (como el ejemplo, primero 2-1 y luego 1-1). Para evitar eso, cada mensaje debería tener un número de secuencia o un tiempo de envío. Así el cliente puede saber cuál mensaje es el más reciente y mostrarlo correctamente. También se pueden enviar confirmaciones desde el cliente para saber si algo se perdió.

- f. ¿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?

En TCP, el broker debe mantener una conexión con cada suscriptor, y eso aumenta mucho el uso de memoria y CPU. Con UDP, el broker puede mandar un solo mensaje y que muchos lo reciban (o usar multicast), lo que ahorra recursos, aunque algunos mensajes pueden perderse. En resumen: TCP escala peor, pero es más confiable; UDP escala mejor, pero puede perder mensajes.

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

En TCP, las conexiones se cierran y los clientes se dan cuenta porque el socket se cae o hay un timeout. Es más fácil detectar el fallo. En UDP no hay conexión, así que los clientes solo notan que no están llegando mensajes. En ese caso, se deberían usar mensajes de prueba para detectar que el servidor sigue vivo.

- h. ¿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?

No se puede lograr una sincronización perfecta en Internet, pero sí se puede acercar. Lo más común es que todos los mensajes incluyan una marca de tiempo (timestamp) y que los clientes esperen un poco antes de mostrarlo, para que todos vean el evento al mismo tiempo. UDP multicast ayuda a reducir la latencia, pero TCP o QUIC mantienen la consistencia y aseguran que nadie se quede sin recibir la actualización.

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

TCP usa más CPU y memoria porque necesita guardar información de cada conexión (buffers, control de congestión, etc.). UDP es más liviano porque no guarda estado, y por eso puede atender más clientes con menos recursos. Sin embargo, si la aplicación tiene que agregar su propio control de errores, ese ahorro se puede reducir un poco.

- j. 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 mezcla de los dos: TCP o QUIC para los eventos importantes (como el marcador o los goles) y UDP o multicast para las actualizaciones rápidas o los datos que se pueden perder sin problema. Así se aprovecha lo mejor de ambos mundos: la confiabilidad de TCP y la velocidad de UDP.

#### 4. Bibliografía

- Kurose, J. F., & Ross, K. W. (2021). *Computer Networking: A Top-Down Approach* (8th ed.). Pearson.
- Universidad de los Andes – Departamento de Ingeniería de Sistemas y Computación. (2025). *Laboratorio 3 – ANÁLISIS CAPA DE TRANSPORTE Y SOCKETS*. ISIS-3204 Infraestructura de Comunicaciones.