

**UNIVERSIDAD DE LOS ANDES
DEPARTAMENTO DE INGENIERIA DE
SISTEMAS Y COMPUTACIÓN**



**LABORATORIO: ANÁLISIS DE PROTOCOLOS DE LA
CAPA DE APLICACIÓN
ISIS 3204 – INFRESTRUCTURA DE
COMUNICACIONES**

Grupo 8

Edward Camilo Sánchez Novoa – 202113020

Daniel Gómez - 202215189

David Lara - 202215317

Contenido

Laboratorio 3	3
1. Tabla comparativa TCP y UDP	3
2. Preguntas de Análisis.....	4
2.1 ¿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?	4
2.2 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?	4
2.3 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.	5
2.4 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?	5
2.5 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?.....	6
2.6 ¿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?	7
2.7 ¿Qué sucede si el broker se detiene inesperadamente? ¿qué diferencias hay entre tcp y udp en la capacidad de recuperación de la sesión?.....	7
2.8 ¿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?	7
2.9 Analice el uso de cpu y memoria en el broker cuando maneja múltiples conexiones tcp frente al manejo de datagramas udp. ¿qué diferencias encontró?	8
2.10 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.	8

Laboratorio 3

1. Tabla comparativa TCP y UDP

	TCP	UDP
Confiabilidad	Durante las pruebas se comprobó que TCP garantiza la entrega de los mensajes, gracias al uso de confirmaciones (ACKs) y retransmisiones automáticas. En las evidencias se observa que todos los mensajes enviados fueron recibidos correctamente por el subscriber, sin pérdidas ni repeticiones.	Durante las pruebas se comprobó que UDP no garantiza la entrega de los mensajes, ya que no utiliza confirmaciones (ACKs) ni retransmisiones automáticas. Cada datagrama se envía de manera independiente y el broker simplemente lo reenvía a los suscriptores registrados. En las capturas de Wireshark se observan los mensajes enviados y recibidos correctamente, pero el protocolo no ofrece mecanismos que aseguren que todos los mensajes lleguen a destino. Esto hace que UDP sea un protocolo ligero y rápido, pero menos confiable que TCP.
Orden de entrega	En las evidencias se aprecia que los mensajes llegan en el mismo orden en que fueron enviados. Esto ocurre porque TCP utiliza números de secuencia y mecanismos internos de control que aseguran la entrega ordenada de los segmentos antes de pasarlos a la aplicación.	En las evidencias obtenidas los mensajes llegan en el mismo orden en que fueron enviados, aunque esto no está garantizado. UDP no mantiene una secuencia de paquetes ni control de flujo, por lo que si la red presenta congestión o los datagramas se enrutan de manera diferente, podrían llegar fuera de orden. En este tipo de aplicación, el orden lógico (por ejemplo, los goles o parciales del partido) debe manejarse a nivel de aplicación si se requiere consistencia entre los suscriptores.
Pérdida de mensajes	No se evidencia pérdida de mensajes en las pruebas realizadas. Las capturas de terminal muestran que el subscriber recibe la totalidad de los mensajes enviados, lo que confirma el correcto funcionamiento de los mecanismos de retransmisión y fiabilidad implementados por TCP.	UDP no detecta ni recupera pérdidas de datagramas. Si un mensaje se pierde en tránsito, no existe retransmisión ni aviso de error al publicador o al broker. En la práctica, durante las simulaciones de los partidos no se evidenció pérdida visible, pero en un entorno real podrían omitirse algunos eventos. Por ello, UDP resulta adecuado para aplicaciones donde la inmediatez es más importante que la confiabilidad absoluta, como transmisiones en vivo o notificaciones rápidas.

Overhead de cabeceras/protocolo	Se observa un encabezado más grande (20–32 bytes) que incluye campos como <i>Sequence Number</i> , <i>Acknowledgment Number</i> , <i>Flags</i> (SYN, ACK, PSH) y <i>Window Size</i> . Estos permiten controlar el flujo, confirmar la recepción y mantener el orden de los datos. Por ello, TCP ofrece fiabilidad y control de conexión, pero con mayor overhead.	Presenta un encabezado mucho más simple y pequeño (8 bytes) que solo contiene <i>Source Port</i> , <i>Destination Port</i> , <i>Length</i> y <i>Checksum</i> . No realiza control de flujo ni confirmación de entrega. Esto hace que UDP sea más rápido y liviano, pero sin garantías de entrega, orden o ausencia de duplicados.
---------------------------------	---	---

2. Preguntas de Análisis

2.1 ¿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?

Si el número de publicadores aumentara de dos a cien, el impacto sería diferente para cada protocolo. En TCP, cada publicador mantiene una conexión individual con el broker. Esto implica más sockets abiertos, más confirmaciones (ACKs) y, por tanto, mayor uso de CPU y memoria. El broker tendría que gestionar múltiples conexiones activas y colas de envío, lo que degradaría el rendimiento al crecer el número de partidos. En cambio, con UDP no existen conexiones persistentes: los mensajes se envían como datagramas independientes, por lo que el broker puede procesar un volumen mucho mayor de mensajes con menor carga. Sin embargo, UDP no garantiza entrega ni orden, por lo que podrían perderse actualizaciones en condiciones de alta carga.

2.2 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?

Si un gol no llega a un suscriptor en UDP, la aplicación no tendría forma de recuperarlo, lo que implicaría un marcador desactualizado o incorrecto. TCP maneja mejor este escenario

porque detecta pérdidas mediante ACKs y retransmite automáticamente el mensaje perdido. Por tanto, TCP asegura que todos los suscriptores reciban los eventos críticos, mientras que en UDP esta responsabilidad recaería en la lógica de la aplicación.

2.3 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.

En un sistema de seguimiento en vivo, TCP resulta más adecuado cuando la prioridad es la confiabilidad y la integridad de los datos, como en un marcador o resumen de jugadas. Aunque UDP ofrece menor retardo y mayor velocidad, su falta de confirmaciones puede causar pérdida de información relevante. Por lo tanto, según los resultados del laboratorio, TCP ofrece una comunicación más estable y coherente para la aplicación simulada.

2.4 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?

En las capturas de Wireshark, TCP mostró mayor overhead debido a las cabeceras adicionales: control de flujo, número de secuencia, ACKs, flags, y opciones. UDP tiene una cabecera fija de 8 bytes, mientras que TCP incluye al menos 20 bytes más información de control. Esto hace que UDP sea más eficiente en el tamaño total por paquete, aunque a costa de no tener las funciones de control y fiabilidad que TCP proporciona.

```

✓ Transmission Control Protocol, Src Port: 33784, Dst Port: 9000, Seq: 11, Ack: 1, Len: 32
  Source Port: 33784
  Destination Port: 9000
  [Stream index: 0]
  [Stream Packet Number: 6]
  > [Conversation completeness: Incomplete, DATA (15)]
  [TCP Segment Len: 32]
  Sequence Number: 11 (relative sequence number)
  Sequence Number (raw): 3169813776
  [Next Sequence Number: 43 (relative sequence number)]
  Acknowledgment Number: 1 (relative ack number)
  Acknowledgment number (raw): 3974639417
  1000 ... = Header Length: 32 bytes (8)
  > Flags: 0x018 (PSH, ACK)
  Window: 512
  [Calculated window size: 65536]
  [Window size scaling factor: 128]
  Checksum: 0xfe48 [unverified]
  [Checksum Status: Unverified]
  Urgent Pointer: 0
  > Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
  > [Timestamps]
  > [SEQ/ACK analysis]
  TCP payload (32 bytes)

```

Ilustración 1 Overhead de un paquete por TCP

```

✓ User Datagram Protocol, Src Port: 41568, Dst Port: 5001
  Source Port: 41568
  Destination Port: 5001
  Length: 38
  Checksum: 0xfe39 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  [Stream Packet Number: 1]
  > [Timestamps]
  UDP payload (30 bytes)

```

Ilustración 2 Overhead de un paquete UDP

2.5 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?

Si en UDP los mensajes llegaran desordenados (por ejemplo, un marcador 2–1 antes del 1–1), el usuario vería resultados incoherentes o inconsistentes. TCP evita este problema al garantizar que los datos lleguen en el mismo orden en que se enviaron. En UDP, la aplicación debería implementar mecanismos de control, como numerar los mensajes o mantener un buffer ordenado por tiempo o secuencia, para reconstruir el orden original.

2.6 ¿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?

Cuando crece el número de suscriptores conectados a un mismo partido, TCP tiende a saturarse más rápido, ya que el broker debe enviar cada mensaje por conexiones individuales, manteniendo confirmaciones y colas separadas. En UDP, el broker puede enviar un único datagrama que llegue a múltiples clientes (broadcast o multicast), reduciendo la carga general. No obstante, en UDP algunos clientes podrían perder paquetes, por lo que se sacrifica fiabilidad a cambio de eficiencia.

2.7 ¿Qué sucede si el broker se detiene inesperadamente? ¿qué diferencias hay entre tcp y udp en la capacidad de recuperación de la sesión?

Si el broker se detiene inesperadamente, en TCP las conexiones abiertas se cierran automáticamente y los clientes detectan el corte (ya que ``recv()`` devuelve 0). Esto permite una recuperación ordenada o reconexión controlada. En UDP, como no existen conexiones persistentes, los clientes no detectan inmediatamente la caída del servidor; simplemente dejan de recibir mensajes, lo que dificulta la detección de fallos.

2.8 ¿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?

Para garantizar que todos los suscriptores reciban al mismo tiempo eventos importantes (como un gol), UDP puede parecer más rápido al no tener confirmaciones. Sin embargo, TCP asegura que todos reciban los datos sin pérdida ni desorden, lo cual es más importante en una aplicación de resultados oficiales. La sincronización precisa dependería más del diseño de la

aplicación y del servidor que del protocolo, pero TCP facilita mantener coherencia en los datos.

2.9 Analice el uso de cpu y memoria en el broker cuando maneja múltiples conexiones tcp frente al manejo de datagramas udp. ¿qué diferencias encontró?

El broker TCP consume más recursos porque mantiene estructuras por conexión (buffers de envío, recepción, control de flujo y retransmisión). UDP, al ser sin conexión, reduce el uso de CPU y memoria, ya que cada datagrama se procesa de forma independiente. Durante las pruebas del laboratorio, el broker TCP mostró mayor carga, pero una entrega confiable, mientras que UDP resultó más liviano pero con posibilidad de pérdida.

2.10 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.

En una aplicación masiva de seguimiento de partidos para millones de usuarios, lo ideal sería una combinación de ambos protocolos:

- **TCP** para información crítica (marcadores, goles, estadísticas oficiales).
- **UDP** para información en tiempo real no crítica (animaciones, comentarios, actualizaciones rápidas).

De esta forma se equilibran confiabilidad y rendimiento, aprovechando las ventajas de cada protocolo según el tipo de dato transmitido.

3. Evidencias Ejecución

3.1 EVIDENCIAS TCP

Para validar el correcto funcionamiento del sistema de publicación y suscripción (modelo *Publish/Subscribe*), se ejecutó una prueba con **dos publicadores (publishers)**, **dos**

suscriptores (subscribers) y un **broker central** encargado de gestionar las conexiones y distribuir los mensajes. El objetivo de esta prueba fue evidenciar la forma en que el broker recibe los eventos publicados y los reenvía correctamente a todos los suscriptores registrados en el mismo tema, verificando la concurrencia, el enrutamiento de mensajes y el comportamiento del sistema bajo múltiples emisores y receptores simultáneos. Durante la ejecución se tomaron capturas de terminal y de Wireshark que muestran claramente las interacciones entre los nodos y el flujo de los mensajes.

En esta imagen se observa el inicio del broker, el cual queda en modo escucha y registra las nuevas suscripciones y publicaciones. En la terminal se muestran los mensajes recibidos desde los publishers y reenviados a los suscriptores, confirmando la función de intermediación del bróker.

```
camilo@LAPTOP-H586OKQJ:/mnt/c/Users/camil/Documents/Redes/LAB03/tcp$ ./broker_tcp 9000
[BROKER] Escuchando en 0.0.0.0:9000 (select)
[BROKER] Nueva conexión 127.0.0.1:51606 (cfd=4)
[BROKER] fd=4 -> "ROLE: SUB"
[BROKER] fd=4 -> "SUB: Real_Azul_vs_Atletico_Rojo"
[BROKER] fd=4 suscrito a "Real_Azul_vs_Atletico_Rojo"
[BROKER] Nueva conexión 127.0.0.1:33368 (cfd=5)
[BROKER] fd=5 -> "ROLE: SUB"
[BROKER] fd=5 -> "SUB: Millonarios_vs_SantaFe"
[BROKER] fd=5 suscrito a "Millonarios_vs_SantaFe"
[BROKER] Nueva conexión 127.0.0.1:33856 (cfd=6)
[BROKER] fd=6 -> "ROLE: PUB"
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Inicio del partido"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Atletico_Rojo al 1'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Falta de Real_Azul al 2'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #6 para Atletico_Rojo al 4'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #2 para Real_Azul al 7'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Falta de Real_Azul al 10'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Atletico_Rojo al 12'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #2 para Real_Azul al 14'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Marcador parcial Real_Azul 0 - 0 Atletico_Rojo al 15'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Cambio en Real_Azul: #3 por #27 al 18'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Real_Azul al 19'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Real_Azul al 24'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Atletico_Rojo al 26'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #6 para Atletico_Rojo al 28'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Marcador parcial Real_Azul 0 - 0 Atletico_Rojo al 30'"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] fd=6 -> "PUB: Real_Azul_vs_Atletico_Rojo|Final del partido - Resultado Real_Azul 0 - 0 Atletico_Rojo"
[BROKER] PUB topic="Real_Azul_vs_Atletico_Rojo" -> entregados=1
[BROKER] Cliente fd=6 desconectado
[BROKER] Nueva conexión 127.0.0.1:49432 (cfd=6)
[BROKER] fd=6 -> "ROLE: PUB"
[BROKER] fd=6 -> "PUB: Millonarios_vs_SantaFe|Inicio del partido"
[BROKER] PUB topic="Millonarios_vs_SantaFe" -> entregados=1
[BROKER] fd=6 -> "PUB: Millonarios_vs_SantaFe|Tarjeta amarilla #8 para Millonarios al 1'"
[BROKER] PUB topic="Millonarios_vs_SantaFe" -> entregados=1
[BROKER] fd=6 -> "PUB: Millonarios_vs_SantaFe|Falta de SantaFe al 1'"
[BROKER] PUB topic="Millonarios_vs_SantaFe" -> entregados=1
[BROKER] fd=6 -> "PUB: Millonarios_vs_SantaFe|Tarjeta amarilla #5 para SantaFe al 2'"
[BROKER] PUB topic="Millonarios_vs_SantaFe" -> entregados=1
[BROKER] fd=6 -> "PUB: Millonarios_vs_SantaFe|Falta de SantaFe al 3'"
```

Ilustración 3 Resultado ejecución broker tcp

En estas figuras se muestran los dos suscriptores recibiendo los mensajes enviados por ambos publicadores. Las terminales evidencian que los mensajes llegan correctamente y en orden, sin duplicados ni pérdidas, demostrando la fiabilidad del protocolo.

```
camilo@LAPTOP-HS860KQJ: /mnt/c/Users/camil/Documents/Redes/LAB03/tcp$ ./subscriber_tcp 127.0.0.1 9000 Real_Azul_vs_Atletico_Rojo
[SUB] Conectado a 127.0.0.1:9000
[SUB] Enviada suscripción a "Real_Azul_vs_Atletico_Rojo"
[SUB] <- ACK
[SUB] <- SUBOK
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Inicio del partido
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Atletico_Rojo al 1'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Falta de Real_Azul al 2'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #6 para Atletico_Rojo al 4'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #2 para Real_Azul al 7'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Falta de Real_Azul al 10'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Atletico_Rojo al 12'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #2 para Real_Azul al 14'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Marcador parcial Real_Azul 0 - 0 Atletico_Rojo al 15'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Cambio en Real_Azul: #3 por #27 al 18'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Real_Azul al 19'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Real_Azul al 24'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Atletico_Rojo al 26'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #6 para Atletico_Rojo al 28'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Marcador parcial Real_Azul 0 - 0 Atletico_Rojo al 30'
[SUB] <- MSG: Real_Azul_vs_Atletico_Rojo|Final del partido - Resultado Real_Azul 0 - 0 Atletico_Rojo
```

Ilustración 4 Resultado ejecución suscribir partido 1 tcp

```
camilo@LAPTOP-HS860KQJ: /mnt/c/Users/camil/Documents/Redes/LAB03/tcp$ ./subscriber_tcp 127.0.0.1 9000 Millonarios_vs_SantaFe
[SUB] Conectado a 127.0.0.1:9000
[SUB] Enviada suscripción a "Millonarios_vs_SantaFe"
[SUB] <- ACK
[SUB] <- SUBOK
[SUB] <- MSG: Millonarios_vs_SantaFe|Inicio del partido
[SUB] <- MSG: Millonarios_vs_SantaFe|Tarjeta amarilla #8 para Millonarios al 1'
[SUB] <- MSG: Millonarios_vs_SantaFe|Falta de SantaFe al 1'
[SUB] <- MSG: Millonarios_vs_SantaFe|Tarjeta amarilla #5 para SantaFe al 2'
[SUB] <- MSG: Millonarios_vs_SantaFe|Falta de SantaFe al 3'
[SUB] <- MSG: Millonarios_vs_SantaFe|Gol de Millonarios al 7' (marcador 1-0)
[SUB] <- MSG: Millonarios_vs_SantaFe|Tarjeta amarilla #9 para Millonarios al 7'
[SUB] <- MSG: Millonarios_vs_SantaFe|Tiro de esquina para SantaFe al 9'
[SUB] <- MSG: Millonarios_vs_SantaFe|Falta de Millonarios al 10'
[SUB] <- MSG: Millonarios_vs_SantaFe|Tarjeta amarilla #9 para Millonarios al 15'
[SUB] <- MSG: Millonarios_vs_SantaFe|Falta de SantaFe al 15'
[SUB] <- MSG: Millonarios_vs_SantaFe|Cambio en Millonarios: #4 por #16 al 15'
[SUB] <- MSG: Millonarios_vs_SantaFe|Marcador parcial Millonarios 1 - 0 SantaFe al 15'
[SUB] <- MSG: Millonarios_vs_SantaFe|Cambio en SantaFe: #2 por #27 al 17'
[SUB] <- MSG: Millonarios_vs_SantaFe|Falta de Millonarios al 20'
[SUB] <- MSG: Millonarios_vs_SantaFe|Tarjeta amarilla #11 para Millonarios al 27'
[SUB] <- MSG: Millonarios_vs_SantaFe|Falta de SantaFe al 28'
[SUB] <- MSG: Millonarios_vs_SantaFe|Cambio en SantaFe: #4 por #15 al 29'
[SUB] <- MSG: Millonarios_vs_SantaFe|Marcador parcial Millonarios 1 - 0 SantaFe al 30'
[SUB] <- MSG: Millonarios_vs_SantaFe|Final del partido - Resultado Millonarios 1 - 0 SantaFe
```

Ilustración 5 Resultado ejecución suscribir partido 2 tcp

Aquí se evidencia cómo se inicializan ambos publicadores, los que empiezan a enviar el flujo de eventos hacia el broker.

```
camilo@LAPTOP-HS86OKQJ: /mnt/c/Users/camil/Documents/Redes/LAB03/tcp$ ./publisher_tcp 127.0.0.1 9000 "Real Azul" "Atletico Rojo" 90 150
[PUB] Conectado a 127.0.0.1:9000 (topic=Real_Azul_vs_Atletico_Rojo)
[PUB] Partido terminado (Real_Azul 4 - 4 Atletico_Rojo)
```

Ilustración 6 Resultado ejecución Publisher partido 1 tcp

```
camilo@LAPTOP-HS86OKQJ: /mnt/c/Users/camil/Documents/Redes/LAB03/tcp$ ./publisher_tcp 127.0.0.1 9000 "Millonarios" "SantaFe" 30 150
[PUB] Conectado a 127.0.0.1:9000 (topic=Millonarios_vs_SantaFe)
[PUB] Partido terminado (Millonarios 1 - 0 SantaFe)
camilo@LAPTOP-HS86OKQJ: /mnt/c/Users/camil/Documents/Redes/LAB03/tcp$
```

Ilustración 7 Resultado ejecución Publisher partido 2 tcp

3.2 EVIDENCIAS UDP

Para la segunda parte del laboratorio, se implementó el mismo sistema Publish/Subscribe pero utilizando el protocolo UDP como medio de transporte.

El objetivo fue observar las diferencias en el comportamiento del sistema cuando no existe conexión persistente ni mecanismos de control de entrega, y analizar cómo el broker, los publishers y los suscriptores interactúan en un entorno sin confirmaciones ni retransmisiones.

Durante la prueba, se ejecutaron dos publishers, dos suscriptores y un broker central, encargados de transmitir los eventos de dos partidos de fútbol en paralelo:

- Partido 1: Real Azul vs Atlético Rojo
- Partido 2: Verde CF vs Universidad FC

Cada publicador simuló eventos como goles, tarjetas y parciales del encuentro, los cuales fueron reenviados por el broker hacia los suscriptores correspondientes según el tema (topic) al que estaban suscritos.

En las siguientes imágenes se evidencia el funcionamiento del sistema UDP y la distribución de los mensajes en tiempo real.

Ejecución del Broker UDP

En esta captura se muestra la ejecución del broker_udp, el cual inicia en el puerto 5001 y comienza a recibir suscripciones desde los clientes.

Se registran correctamente las suscripciones de ambos partidos:

```
daniel-gomez@ubuntu-lab3-VMware-Virtual-Platform:~/LAB03/udp$ ./broker_udp
^
d [BROKER UDP] Escuchando en puerto 5001...
d [BROKER UDP] Nueva suscripción 'Real_Azul_vs_Atletico_Rojo'
d [BROKER UDP] Nueva suscripción 'Verde_CF_vs_Universidad_FC'
```

Ilustración 8 Resultado ejecución broker UDP

Ejecución de los Subscribers UDP

En las siguientes terminales se observan los dos suscriptores recibiendo los mensajes reenviados por el broker.

Cada uno muestra en tiempo real los eventos correspondientes al partido al que está suscrito. Los mensajes llegan en orden, aunque, al no existir confirmación ni control de flujo, no se garantiza completamente la entrega o el orden bajo condiciones de congestión de red.

```
daniel-gomez@ubuntu-lab3-VMware-Virtual-Platform:~/LAB03/udp$ ./subscriber_udp 127.0.0.1 5001 Real_Azul_vs_Atletico_Rojo
[SUB UDP] Suscrito a 'Real_Azul_vs_Atletico_Rojo'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Inicio del partido
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Tiro de esquina para Atletico Rojo al 4'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Falta de Real Azul al 5'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Parcial Real Azul 0 - 0 Atletico Rojo al 5'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Gol de Real Azul al 10' (marcador 1-0)
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Parcial Real Azul 1 - 0 Atletico Rojo al 10'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Gol de Atletico Rojo al 12' (marcador 1-1)
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Tarjeta amarilla #2 para Real Azul al 12'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Falta de Atletico Rojo al 12'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Gol de Real Azul al 13' (marcador 2-1)
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Tarjeta roja #11 para Real Azul al 13'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Falta de Real Azul al 14'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Parcial Real Azul 2 - 1 Atletico Rojo al 15'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Parcial Real Azul 2 - 1 Atletico Rojo al 20'
[SUB UDP] <- MSG:Real_Azul_vs_Atletico_Rojo|Final del partido - Resultado Real Azul 2 - 1 Atletico Rojo
```

Ilustración 9 Resultado ejecución subscriber partido 1 (Real Azul vs Atlético Rojo)

```
daniel-gomez@ubuntu-lab3-VMware-Virtual-Platform:~/LAB03/udp$ ./subscriber_udp 127.0.0.1 5001 Verde_CF_vs_Universidad_FC
[SUB UDP] Suscrito a 'Verde_CF_vs_Universidad_FC'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Inicio del partido
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Tiro de esquina para Universidad FC al 2'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Tarjeta roja #10 para Verde CF al 3'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Parcial Verde CF 0 - 0 Universidad FC al 5'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Tiro de esquina para Verde CF al 6'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Parcial Verde CF 0 - 0 Universidad FC al 10'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Parcial Verde CF 0 - 0 Universidad FC al 15'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Tarjeta amarilla #9 para Universidad FC al 16'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Tarjeta amarilla #11 para Verde CF al 20'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Parcial Verde CF 0 - 0 Universidad FC al 20'
[SUB UDP] <- MSG:Verde_CF_vs_Universidad_FC|Final del partido - Resultado Verde CF 0 - 0 Universidad FC
```

Ilustración 10 Resultado ejecución subscriber partido 2 (Verde CF vs Universidad FC)

Ejecución de los Publishers UDP

En estas imágenes se evidencia cómo se inicializan ambos publishers, encargados de generar los eventos simulados para cada partido.

Cada publicador envía los mensajes de tipo PUB:<topic>|<evento> hacia el broker, el cual los retransmite inmediatamente a todos los suscriptores del mismo tema.

A diferencia de TCP, aquí no se establecen conexiones persistentes; cada envío se realiza con un datagrama independiente mediante `sendto()`.

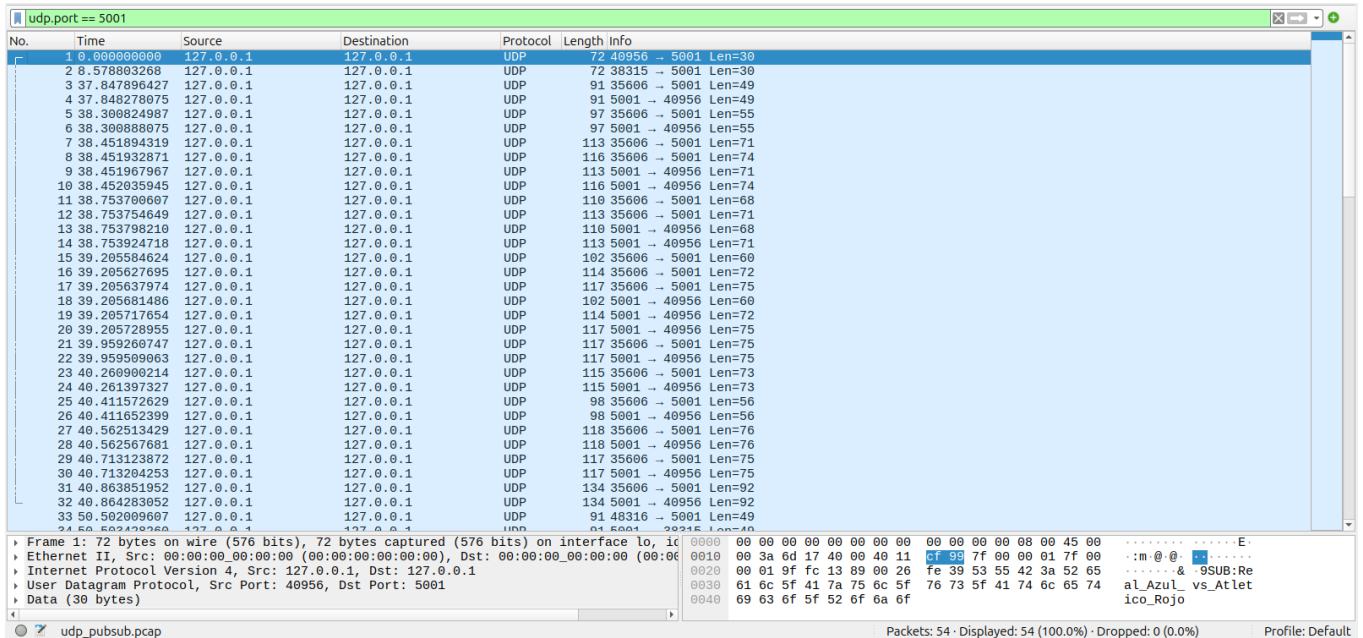
```
daniel-gomez@ubuntu-lab3-VMware-Virtual-Platform:~/LAB03/udp$ ./publisher_udp 127.0.0.1 5001 "Real Azul" "Atletico Rojo" 20 150
daniel-gomez@ubuntu-lab3-VMware-Virtual-Platform:~/LAB03/udp$
daniel-gomez@ubuntu-lab3-VMware-Virtual-Platform:~/LAB03/udp$ ./publisher_udp 127.0.0.1 5001 "Verde CF" "Universidad FC" 20 150
```

Ilustración 11 Resultado ejecución publisher de ambos partidos

Captura Wireshark (UDP)

En Wireshark se observa el intercambio de datagramas entre el broker y los clientes usando el filtro:

En Wireshark se observa el intercambio de datagramas entre el broker y los clientes usando el filtro:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	UDP	72	40956 → 5001 Len=30
2	0.578803268	127.0.0.1	127.0.0.1	UDP	72	38315 → 5001 Len=30
3	0.847896427	127.0.0.1	127.0.0.1	UDP	91	35606 → 5001 Len=49
4	0.848278075	127.0.0.1	127.0.0.1	UDP	91	5001 → 40956 Len=49
5	38.380824987	127.0.0.1	127.0.0.1	UDP	97	35606 → 5001 Len=55
6	38.380888075	127.0.0.1	127.0.0.1	UDP	97	5001 → 40956 Len=55
7	38.451894319	127.0.0.1	127.0.0.1	UDP	113	35606 → 5001 Len=71
8	38.451932871	127.0.0.1	127.0.0.1	UDP	116	35606 → 5001 Len=74
9	38.451967967	127.0.0.1	127.0.0.1	UDP	113	5001 → 40956 Len=71
10	38.452035945	127.0.0.1	127.0.0.1	UDP	116	5001 → 40956 Len=74
11	38.753700607	127.0.0.1	127.0.0.1	UDP	110	35606 → 5001 Len=68
12	38.753754649	127.0.0.1	127.0.0.1	UDP	113	35606 → 5001 Len=71
13	38.753798210	127.0.0.1	127.0.0.1	UDP	110	5001 → 40956 Len=68
14	38.753924718	127.0.0.1	127.0.0.1	UDP	113	5001 → 40956 Len=71
15	39.205584624	127.0.0.1	127.0.0.1	UDP	102	35606 → 5001 Len=60
16	39.205627695	127.0.0.1	127.0.0.1	UDP	114	35606 → 5001 Len=72
17	39.205637974	127.0.0.1	127.0.0.1	UDP	117	35606 → 5001 Len=75
18	39.205681486	127.0.0.1	127.0.0.1	UDP	102	5001 → 40956 Len=60
19	39.205717654	127.0.0.1	127.0.0.1	UDP	114	5001 → 40956 Len=72
20	39.205728955	127.0.0.1	127.0.0.1	UDP	117	5001 → 40956 Len=75
21	39.959260747	127.0.0.1	127.0.0.1	UDP	117	35606 → 5001 Len=75
22	39.959509063	127.0.0.1	127.0.0.1	UDP	117	5001 → 40956 Len=75
23	40.260900214	127.0.0.1	127.0.0.1	UDP	115	35606 → 5001 Len=73
24	40.261397327	127.0.0.1	127.0.0.1	UDP	115	5001 → 40956 Len=73
25	40.411572629	127.0.0.1	127.0.0.1	UDP	98	35606 → 5001 Len=56
26	40.411652399	127.0.0.1	127.0.0.1	UDP	98	5001 → 40956 Len=56
27	40.562513429	127.0.0.1	127.0.0.1	UDP	118	35606 → 5001 Len=76
28	40.562567681	127.0.0.1	127.0.0.1	UDP	118	5001 → 40956 Len=76
29	40.713123872	127.0.0.1	127.0.0.1	UDP	117	35606 → 5001 Len=75
30	40.713204253	127.0.0.1	127.0.0.1	UDP	117	5001 → 40956 Len=75
31	40.863851952	127.0.0.1	127.0.0.1	UDP	134	35606 → 5001 Len=92
32	40.864283052	127.0.0.1	127.0.0.1	UDP	134	5001 → 40956 Len=92
33	50.562009607	127.0.0.1	127.0.0.1	UDP	91	48316 → 5001 Len=49

Frame 1: 72 bytes on wire (576 bits), 72 bytes captured (576 bits) on interface lo, 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: 40956, Dst Port: 5001
 Data (30 bytes)

0000
 0010 00 3a 6d 17 40 00 40 11 5f 99 7f 00 00 01 7f 00E
 0020 00 01 9f fc 13 89 00 26 fe 39 53 55 42 3a 52 65& 9SUB:Re
 0030 61 6c 5f 41 7a 75 6c 5f 76 73 5f 41 74 6c 65 74 al_Azul_vs_Atlet
 0040 69 63 6f 5f 52 6f 6a 6f 76 73 5f 41 74 6c 65 74 ico_Rojo

Ilustración 12 Captura Wireshark

En Wireshark, con el filtro `udp.port == 5001`, se observan los datagramas enviados al broker. En la columna Info aparece `40956 → 5001`, indicando el puerto efímero del cliente hacia el puerto del broker. En el panel de datos se ve el payload con nuestro protocolo de aplicación (`SUB:<topic>` o `PUB:<topic>|<mensaje>`), evidenciando la suscripción/publicación y el fan-out posterior del broker