Universidad de los Andes

Departamento de Ingeniería de Sistemas

Laboratorio #3: Análisis De Capa de Transporte y Sockets

ISIS3204 - Infraestructura de Comunicaciones

Grupo 3:

Juan Esteban Quiroga - 202013216
Juan Manuel Rodriguez - 202013372
Andres Felipe Ortiz - 201727662

2025-10

Contents

Introducción

En este laboratorio implementamos un sistema de mensajería usando el modelo publicador–suscriptor con sockets TCP y UDP para simular la transmisión en tiempo real de eventos en partidos de la UEFA Champions League. Los publicadores actúan como periodistas que envían actualizaciones en vivo de los partidos, el broker gestiona la distribución de los mensajes por partido, y los suscriptores reciben las notificaciones relevantes a las que estan suscritos. El sistema demuestra la comunicación asíncrona, la escalabilidad y los compromisos entre confiabilidad y rendimiento de los protocolos de transporte TCP y UDP en aplicaciones basadas en sockets.

Escenario de prueba

Nostros creamos un sistema que permite generar simulaciones de partidos entre los siguientes equipos:

- Real Madrid CF
- Juventus FC
- Paris Saint-Germain FC
- FC Barcelona
- Liverpool FC
- FC Bayern Munich

Sin embargo, para las pruebas realizadas en Wireshark, montamos 3 partidos en particular: `RealMadrid_vs_Juventus`, `Liverpool_vs_BayernMunich`, y `Barcelona_vs_PSG` en los cuales hay un publicador por partido transmitiendo al broker. Por otro lado tenemos 4 suscriptores:

- Suscriptor #1: `RealMadrid_vs_Juventus`, `Liverpool_vs_BayernMunich`, y `Barcelona_vs_PSG`
- Suscriptor #2: `RealMadrid_vs_Juventus`
- Suscriptor #3: `Liverpool_vs_BayernMunich`
- Suscriptor #4: `Barcelona_vs_PSG` y `Liverpool_vs_BayernMunich`

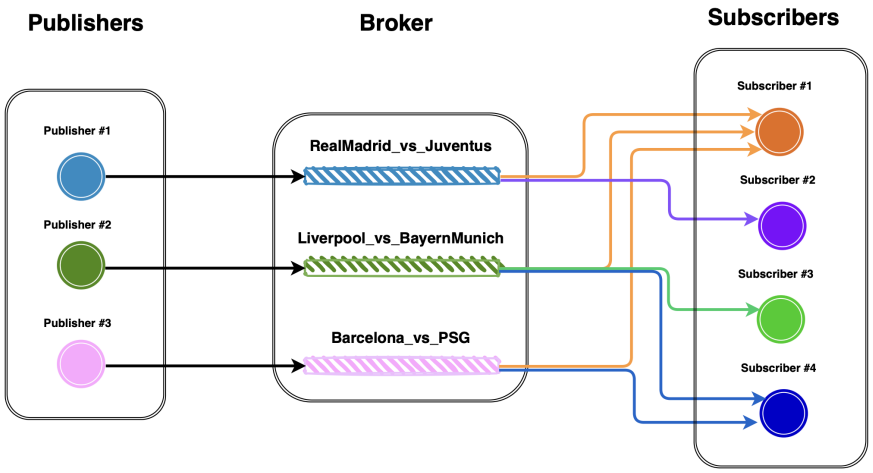

Figure 1: *Configuración de escenario de prueba*

1 Modelo Publicador–Suscriptor

1.1 Configuración

Nuestro proyecto tiene la siguiente configuración que fue diseñada para poder reutilizar codigo compartido entre los archivos TCP y UDP.

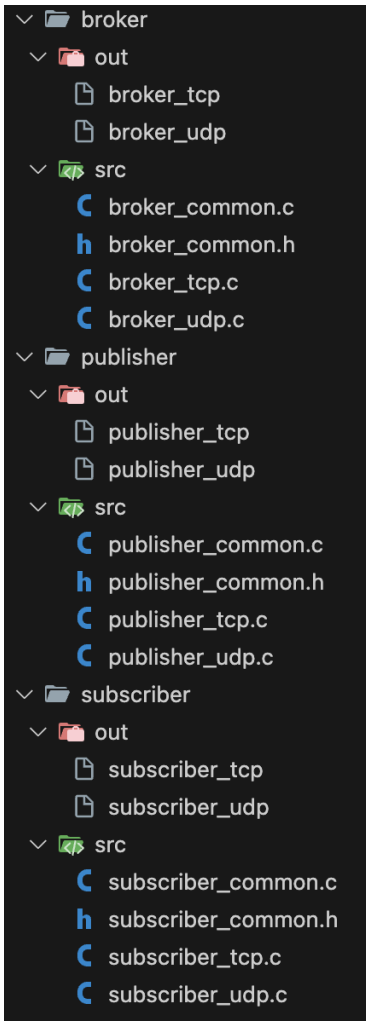

Figure 2: *Estructura de proyecto*

1.2 Publicador

1.2.1 Descripción

El publicador es el componente encargado de generar y enviar los mensajes correspondientes a los eventos de cada partido. Existen dos versiones: una basada en TCP y otra en UDP, que utilizan el mismo módulo de simulación pero son distintos en la forma en que transmiten los datos.

- En la versión TCP, el publicador establece una conexión confiable con el broker mediante un socket orientado a conexión. Una vez conectado, envía los eventos de forma secuencial usando la función `sendTo()`, garantizando que los mensajes lleguen en orden y sin pérdidas.
- En la versión UDP, el publicador utiliza un socket sin conexión, enviando cada evento mediante `send()` al puerto del broker especificado Es necesario usar esta función ya que UDP no establece un 3-way handshake entonces hay que identificar el suscriptor para enviar el evento). Aunque este método no asegura la entrega o el orden de los mensajes, permite una transmisión más rápida y ligera, simulando escenarios donde la inmediatez es prioritaria sobre la confiabilidad.

En ambos casos, los mensajes se formatean con el tema del partido y el tipo de evento, y se envían con un intervalo de aproximadamente un segundo entre ellos.

Cada mensaje del publicador sigue el siguiente formato:

```
<Match_name>|[EVENT_TYPE] <Event description> at <minute>' -- <TeamA> <scoreA>-<scoreB> <TeamB>
```

1.2.2 Tipos de eventos

Estructura de partido:

Cada partido dura 90 minutos y es dividido en 12 eventos. Para cada evento del partido, se elige un tiempo aleatorio entre rangos de valores de cada "bucket" de tiempo. Por ejemplo, el primer bucket incluye minutos del [1-7] entonces estos son posibles valores para lanzar el primer evento del partido. Cada uno de los 10 eventos del partido puede pertenecer a una de las siguientes categorías:

- **[GOAL]** – algún jugador anota un gol y otro jugador marca la asistencia
- **[CARD]** – algún jugador recibe tarjeta amarilla o tarheta roja
- **[SUB]** – algún jugador que está jugando en el campo es reemplazado por algún jugador que está en la banca del mismo equipo y aún no ha jugado en ese partido. (solo sucede después de 55')
- **[HT]** – indica el marcador en el mediotiempo (45')
- **[FT]** – indica el marcador final (90')

1.2.3 Reglas de negocio

Tarjetas

- La primera vez que un jugador es señalado, se le muestra la tarjeta amarilla
- La segunda vez que un jugador es señalado, se le muestra la tarjeta roja y es expulsado del partido dejando al equipo con 10 hombres en el campo
- 20% de las veces en cualquier evento, a un jugador se le puede mostrar una tarajeta roja directa

Anotar

- El sistema elige uno de los dos equipos para anotar cuando se genera un evento de tipo [GOAL]
- El jugador solo se elige si no ha sido expulsado (igual que el jugador que da la asistencia)

1.3 Broker

1.3.1 Descripción

El broker actúa como un intermediario entre los publicadores y los suscriptores dentro del sistema de notificaciones de eventos. El broker recibe los mensajes enviados por los publicadores (a través de TCP o UDP), identifica el partido al que pertenecen y los reenvia únicamente a los suscriptores que estén suscritos a ese partido.

Cada suscriptor envía un mensaje al broker que indica el interés en un partido. El broker guarda esa suscripción en una tabla interna de partidos y cada vez que llega un mensaje con ese mismo nombre de partido, lo reenvía a todos los suscriptores asociados.

(El broker no contiene lógica de negocio, ya que su función se limita al enrutamiento de mensajes. No interpreta el contenido de los eventos, ya que solo actúa como un intermediario que se encarga de que cada suscriptor reciba únicamente la información correspondiente a los partidos a los que está suscrito.)

1.4 Suscriptor

1.4.1 Descripción

El suscriptor representa al cliente final que recibe los eventos transmitidos por el sistema. Su función principal es conectarse al broker (mediante TCP o UDP) y registrarse en uno o varios temas específicos, enviando un mensaje con el siguiente formato:

```
SUBSCRIBE|<topic_1> <topic_2> ... <topic_n>
```

Una vez suscrito, el suscriptor permanece a la espera de mensajes provenientes del broker. Cada mensaje recibido tiene información sobre un evento en tiempo real que corresponde a los partidos a los que se ha suscrito.

En la versión TCP, el suscriptor mantiene una conexión persistente con el broker y recibe los mensajes mediante `recv()`. En la versión UDP, no existe una conexión formal, sino que los mensajes llegan directamente al puerto local asociado al socket mediante `recvfrom()`.

2 Pruebas y comparación

En esta sección, verificamos el funcionamiento del modelo de comunicación.

2.1 TCP

TCP es el protocolo que tiene envío de paquetes confiable, los mensajes llegan ordenados, y require una conexión persistente que se establece con un handshake. El flujo que sigue nuestra aplicación se describe a continuación:

1. El broker crea un socket y escucha en un puerto esperando conexiones de publicadores y suscriptores.
2. Los suscriptores se conectan al broker por medio de conexión TCP usando el 3-way handsahke (SYN, SYN-ACK, ACK).
3. El suscriptor manda un mensaje al broker que contiene las suscripciones que quiere.
4. Los publicadores establecen una conexión TCP con el broker.
5. Los publicadores envían eventos de partidos al broker.
6. El broker identifica los suscriptores que están suscritos a los partidos asociados a los eventos que llegan de los publicadores.
7. El broker envía los mensajes de los eventos por medio de la conexión TCP a los suscriptores.
8. El suscriptor recibe y muestra el mensaje de cada evento.
9. Se cierran las conexiones de los publicadores automaticamente después de enviar el último evento (los suscriptores siguen escuchando hasta desconectarse manualmente).

2.2 UDP

TCP es el protocolo que tiene envío de sin handshake, es más rápido que TCP pero no es confiable ya que algunos mensajes podrían no llegar o llegar fuera de orden. El flujo que sigue nuestra aplicación se describe a continuación:

1. El broker crea un socket UDP y escucha en un puerto esperando conexiones de publicadores y suscriptores.
2. Los suscriptores crean un socket UDP y se vincula a un puerto aleatorio.
3. El suscriptor manda un mensaje al broker que contiene las suscripciones que quiere.
4. Los publicadores crean un socket UDP (no hay necesidad de un handshake).
5. Los publicadores envían eventos de partidos al broker.
6. El broker busca todos los suscriptores de los partidos según los eventos que llegan.
7. El broker envía el evento a cada IP/puerto que corresponde.
8. El suscriptor recibe y muestra el mensaje de cada evento.

2.2.1 Conexión suscriptores a broker

Primero, vamos a demostrar que los suscriptores establecen un 3-way handshake con el broker. En los screenshots vemos que el suscriptor #1 hace bind con un puerto aleatorio, establace el handshake, y envía mensajes al broker indicando los partidos a los que se quiere suscribir.

En las captura de Wireshark, los segmentos TCP con las banderas PSH y ACK indican que el emisor está enviando datos que deben entregarse inmediatamente a la aplicación receptora, sin esperar más bytes en el búfer. El campo Seq (secuencia) muestra el número del primer byte transmitido en ese segmento, mientras que Ack (acuse) confirma los bytes recibidos del otro extremo. Por ejemplo, un paquete con Seq=1 seguido de otro con Seq=34 significa que el primer mensaje contenía 33 bytes y que el segundo continúa justo después, reflejando la naturaleza secuencial y confiable de la transmisión TCP.

Suscriptor #1 (Puerto 64376)

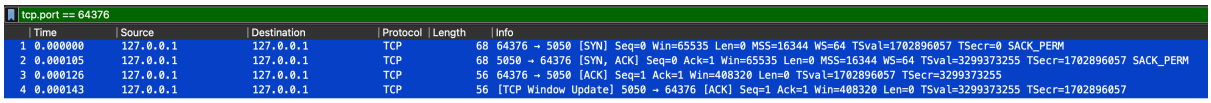

Figure 3: *Conexión de suscriptor 1*

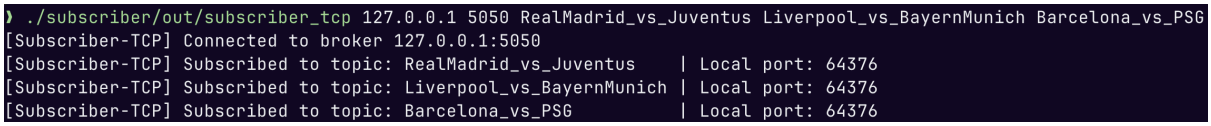

Figure 4: *Verificación de conexión de suscriptor 1 en la terminal de suscriptor*

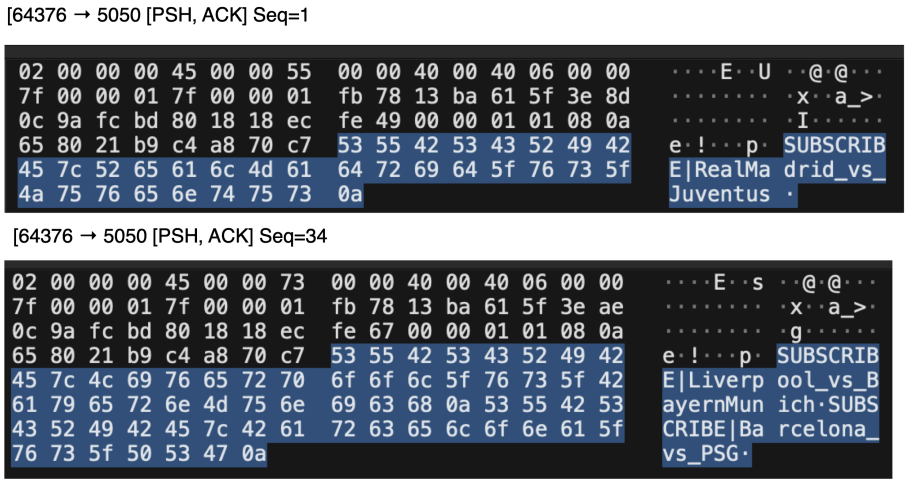

Figure 5: *Mensaje de suscripción al broker*

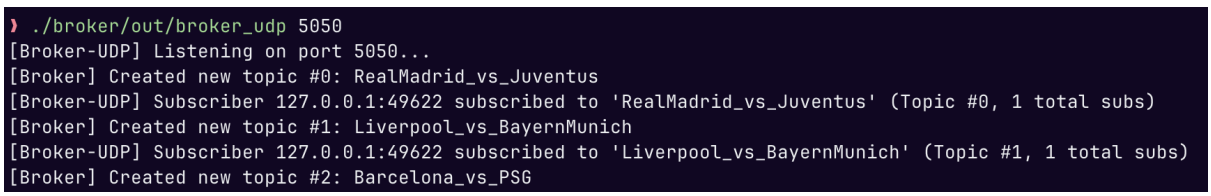

Figure 6: *Suscripciones de suscriptor 1 en terminal del broker*

Suscriptor #2 (Puerto 64377)

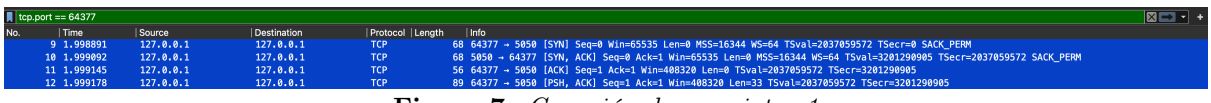

Figure 7: *Conexión de suscriptor 1*

```
❯ ./subscriber/out/subscriber_tcp 127.0.0.1 5050 RealMadrid_vs_Juventus Liverpool_vs_BayernMunich Barcelona_vs_PSG
[Subscriber-TCP] Connected to broker 127.0.0.1:5050
[Subscriber-TCP] Subscribed to topic: RealMadrid_vs_Juventus    | Local port: 64376
[Subscriber-TCP] Subscribed to topic: Liverpool_vs_BayernMunich | Local port: 64376
[Subscriber-TCP] Subscribed to topic: Barcelona_vs_PSG          | Local port: 64376
```

Figure 8: *Verificación de conexión de suscriptor 1 en la terminal de suscriptor*

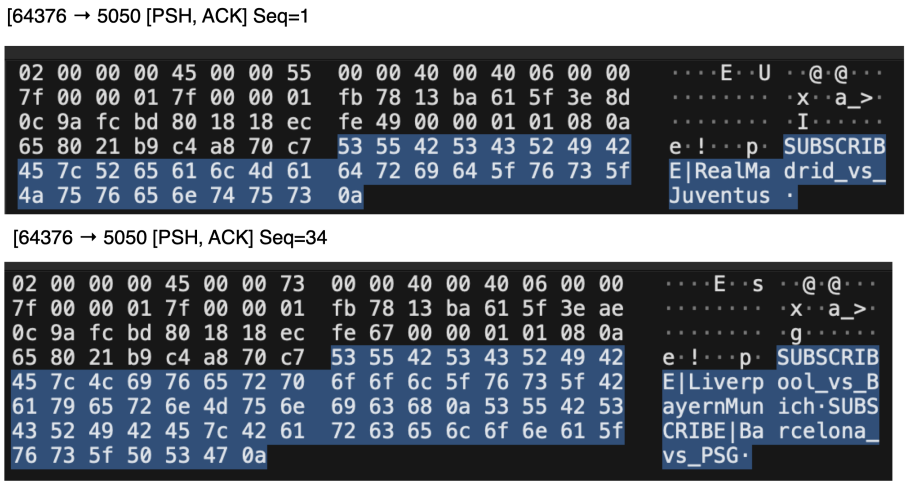

Figure 9: *Mensaje de suscripción al broker*

```
❯ ./broker/out/broker_udp 5050
[Broker-UDP] Listening on port 5050...
[Broker] Created new topic #0: RealMadrid_vs_Juventus
[Broker-UDP] Subscriber 127.0.0.1:49622 subscribed to 'RealMadrid_vs_Juventus' (Topic #0, 1 total subs)
[Broker] Created new topic #1: Liverpool_vs_BayernMunich
[Broker-UDP] Subscriber 127.0.0.1:49622 subscribed to 'Liverpool_vs_BayernMunich' (Topic #1, 1 total subs)
[Broker] Created new topic #2: Barcelona_vs_PSG
```

Figure 10: *Suscripciones de suscriptor 1 en terminal del broker*

Cada vez que el broker detecta un partido nuevo, crea una fila en la tabla interna y si llega un suscriptor futuro que quiere recibir eventos de ese partido, solo lo asigna en la tabla.

Luego de establecer la conexión y registrar los partidos que quiere escuchar, el broker queda pendiente de mensajes con eventos de los publicadores.

Conclusion final

- **Información de la capa de aplicación:** Se identificó el uso del protocolo **RTMP (Real-Time Messaging Protocol)**, encargado de la transmisión en tiempo real de audio, video y datos, soportado en mensajes como *Handshake*, *Window Acknowledgement Size*, *Set Peer Bandwidth* y *NetConnection.Connect.Success*.
- **Protocolo de la capa de transporte:** La transmisión hace uso de **TCP**, que garantiza una comunicación confiable mediante el *three-way handshake* (SYN, SYN-ACK, ACK), retransmisiones y confirmaciones.
- **Puertos utilizados:** El servidor emplea el puerto estándar **1935** para RTMP, mientras que el cliente se conecta desde un puerto dinámico, identificado en la captura como el **2255**.

References

[1] Computer Networking, a top-down approach. James Kurose, Keith Ross. Addison-Wesley, 6th ed.