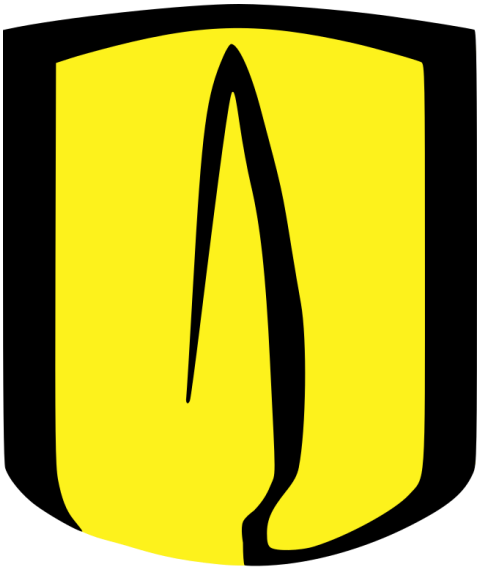
UNIVERSIDAD DE LOS ANDES
DEPARTAMENTO DE INGENIERIA DE SISTEMAS Y COMPUTACIÓN



LABORATORIO: Análisis capa de transporte y sockets

ISIS3204-INFRAESTRUCTURA DE COMUNICACIONES

Carlos Andres Lozano
Natalia Alexandra Quiroga

Grupo 8
Mariana Pineda Miranda – 2021233330
Joseph Linares – 202111887
Ángel Restrepo – 201914073

2025-2

1. UDP
2. TCP
3. Comparación
4. Preguntas de analisis

1. UDP (User Datagram Protocol)

Para la solución del ejercicio propuesto se desarrolló un sistema de publicador-suscriptor, en donde se usó el protocolo de capa de transporte UDP. Este modelo permite que varios publicadores transmitan actualizaciones o eventos sobre distintos partidos de futbol, para que múltiples suscriptores reciban dichas actualizaciones de manera simultánea, sin la necesidad de establecer una conexión directa y persistente entre cada publicador y suscriptor.

Este sistema está compuesto por tres componentes principales:

- Publicador, este envía actualizaciones de partidos, en este caso se definió el formato: “PUBLISH| BarcelonavsRealMadrid | Gol al minuto 75"

  Para este sistema no se necesita conocer las direcciones de los suscriptores ya que únicamente interactuara con el bróker. Cuando se ejecuta el suscriptor se mandan automáticamente 20 actualizaciones con goles en diferentes minutos, esto para ver cómo se comporta el protocolo cuando se quieren mandar 20 actualizaciones sin un intervalo de tiempo grande.
- Broker, actúa como intermediario entre los publicadores y suscriptores. Este recibe los mensajes de los publicadores, luego los clasifica según el partido y los reenvía a los suscriptores que se han registrado al partido. Para esto, en el bróker se define un nuevo tipo “Partido” en donde se tiene la lista de los suscriptores, junto con su respectiva IP y puerto, y se guardan todos los partidos dentro de una lista de partidos. Y luego, cuando llega una nueva publicación, se busca el partido al que pertenece dentro de la lista de partidos, y luego se manda a cada uno de los suscriptores en la lista de direcciones de los suscriptores de cada partido.
- Suscriptores, se registran a un partido por medio del mensaje:

  “SUBSCRIBE|Partido1"

  Se suscribe y luego espera a recibir las actualizaciones que serán enviadas por el bróker. En la consola se pueden observar las diferentes actualizaciones que se reciben del parido al que se encuentra suscrito.

Como se mencionó anteriormente, el broker mantiene una estructura de datos (Partido) que contiene el nombre del partido y la lista de suscriptores (direcciones IP y puertos) asociados.

Cada vez que recibe una nueva actualización, el broker ejecuta una función llamada transmitirActualizacion(), que envía el mensaje a todos los suscriptores registrados de ese partido.

En este caso, UDP es una muy buena alternativa de protocolo ya que se necesitan transmisiones en tiempo real, al no ser un protocolo orientado a conexión ni con confirmación de entrega se pueden mandar mensajes de manera más rápida, pero los mensajes pueden llegar fuera de orden o perderse. Es muy bueno para este escenario en donde la velocidad y difusión a múltiples receptores son más importantes que la confiabilidad total. Sin embargo, el orden de los mensajes si puede afectar al suscriptor ya que no sería muy bueno recibir primero actualizaciones del minuto 30 y que después se reciban las actualizaciones del minuto 20. Algunas de las ventajas de implementar este sistema con UDP son:

- Al ser las actualizaciones pequeñas y frecuentes, el costo de establecer y mantener conexiones de TCP con cada suscriptor podría llegar a ser innecesario.
- UDP permite una difusión simultanea y sin bloqueos, lo que es ideal para que un publicador pueda enviar información a muchos suscriptores a la vez.
- En este caso, teníamos una red local por lo que la perdida de paquetes es muy baja, y el sistema puede beneficiarse de la rapidez y simplicidad de UDP.

Con esto, cuando se implementa por medio de UDP resulta en un sistema ligero, rápido, flexible, que permite que múltiples suscriptores reciban actualizaciones casi en tiempo real, sin necesidad de conexiones persistentes ni mecanismos de confirmación. Aunque exista la posibilidad de pérdida de mensajes, en este sistema se prioriza velocidad y difusión sobre la confiabilidad total.

Para la ejecución de publicador suscriptor se ejecutaron simultáneamente:

Tres suscriptores cada uno a un partido:

Para el partido “Real Madrid vs Barcelona” se inicia la conexión:

```
[maripinemira@Marianas-MacBook-Air-98 UDP % ./subscriber_udp 127.0.0.1 RealMadridvsBarcelona
Se ha subscrito al partido: 'RealMadridvsBarcelona'.
```

Imagen 1.1. Suscripción a partido Real Madrid vs Barcelona

Para el partido Deportivo Cali vs Atlético Nacional se inicia también una conexión:

```
[maripinemira@Marianas-MacBook-Air-98 UDP % ./subscriber_udp 127.0.0.1 DeportivoCalivsAtleticoNacional
Se ha subscrito al partido: 'DeportivoCalivsAtleticoNacional'.
```

Imagen 1.2. Suscripción a partido Deportivo Cali vs Atletico Nacional

Y se decidio crear otra conexión al partido de Deportivo Cali vs Atlético Nacional para ver cómo se comportaba un partido con dos suscriptores:

```
maripinemira@Marianas-MacBook-Air-98 UDP % ./subscriber_udp 127.0.0.1 DeportivoCalivsAtleticoNacional
Se ha subscrito al partido: 'DeportivoCalivsAtleticoNacional'.
```

Imagen 1.3. Suscripción a partido Deportivo Cali vs Atletico Nacional

El bróker en donde se pueden ver las tres suscripciones que se tienen:

```
maripinemira@Marianas-MacBook-Air-98 UDP % ./broker_udp
Puerto del Broker: 10094
SUBSCRIBE 'RealMadridvsBarcelona' (subs=1)
SUBSCRIBE 'DeportivoCalivsAtleticoNacional' (subs=1)
SUBSCRIBE 'DeportivoCalivsAtleticoNacional' (subs=2)
```

Imagen 1.4. Ejecución del Broker con suscriptores.

Y luego se ejecutan los publicadores:

El primer publicador corresponde al partido “Real Madrid vs Barcelona”, en donde se envían las 20 actualizaciones, en este caso se numeraron para ver si había alguna perdida de los paquetes:

```
maripinemira@Marianas-MacBook-Air-98 UDP % ./publisher_udp 127.0.0.1 RealMadridvsBarcelona
Escribe la actualización y presiona Enter. Escribe 'quit' para salir.
Enviado automático (1/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 68 (1)
Enviado automático (2/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 80 (2)
Enviado automático (3/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 24 (3)
Enviado automático (4/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 39 (4)
Enviado automático (5/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 41 (5)
Enviado automático (6/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 63 (6)
Enviado automático (7/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 25 (7)
Enviado automático (8/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 9 (8)
Enviado automático (9/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 54 (9)
Enviado automático (10/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 20 (10)
Enviado automático (11/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 1 (11)
Enviado automático (12/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 36 (12)
Enviado automático (13/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 43 (13)
Enviado automático (14/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 63 (14)
Enviado automático (15/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 28 (15)
Enviado automático (16/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 84 (16)
Enviado automático (17/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 8 (17)
Enviado automático (18/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 80 (18)
Enviado automático (19/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 71 (19)
Enviado automático (20/10): PUBLISH|RealMadridvsBarcelona|Gol al minuto 83 (20)
```

Imagen 1.5. Ejecución del Publicador de Real Madrid vs Barcelona

Ahora, el segundo publicador corresponde al del partido Deportivo Cali vs Atetico Nacional, en donde también se hace el envió de 20 mensajes a los dos suscriptores que tiene este partido.

```
Last login: Tue Oct 14 17:05:52 on ttys017
maripinemira@Marianas-MacBook-Air-98 UDP % ./publisher_udp 127.0.0.1 Partido1
Escribe la actualización y presiona Enter. Escribe 'quit' para salir.
Enviado automático (1/10): PUBLISH|Partido1|Gol al minuto 68 (1)
Enviado automático (2/10): PUBLISH|Partido1|Gol al minuto 80 (2)
Enviado automático (3/10): PUBLISH|Partido1|Gol al minuto 24 (3)
Enviado automático (4/10): PUBLISH|Partido1|Gol al minuto 39 (4)
Enviado automático (5/10): PUBLISH|Partido1|Gol al minuto 41 (5)
Enviado automático (6/10): PUBLISH|Partido1|Gol al minuto 63 (6)
Enviado automático (7/10): PUBLISH|Partido1|Gol al minuto 25 (7)
Enviado automático (8/10): PUBLISH|Partido1|Gol al minuto 9 (8)
Enviado automático (9/10): PUBLISH|Partido1|Gol al minuto 54 (9)
Enviado automático (10/10): PUBLISH|Partido1|Gol al minuto 20 (10)
Enviado automático (11/10): PUBLISH|Partido1|Gol al minuto 1 (11)
Enviado automático (12/10): PUBLISH|Partido1|Gol al minuto 36 (12)
Enviado automático (13/10): PUBLISH|Partido1|Gol al minuto 43 (13)
Enviado automático (14/10): PUBLISH|Partido1|Gol al minuto 63 (14)
Enviado automático (15/10): PUBLISH|Partido1|Gol al minuto 28 (15)
Enviado automático (16/10): PUBLISH|Partido1|Gol al minuto 84 (16)
Enviado automático (17/10): PUBLISH|Partido1|Gol al minuto 8 (17)
Enviado automático (18/10): PUBLISH|Partido1|Gol al minuto 80 (18)
Enviado automático (19/10): PUBLISH|Partido1|Gol al minuto 71 (19)
Enviado automático (20/10): PUBLISH|Partido1|Gol al minuto 83 (20)
maripinemira@Marianas-MacBook-Air-98 UDP %
```

Imagen 1.6. Ejecución del Publicador de Real Madrid vs Barcelona

Ya enviadas las actualizaciones se pueden ver en cada uno de los suscriptores:

En el suscriptor 1, se pueden ver las diferentes actualizaciones del partido, en este caso se puede ver que no hay perdida de paquetes. Y llegaron todas las actualizaciones el suscriptor. Esto se puede ver con las numeraciones que se pusieron para ver si había perdida de paquetes.

```
maripinemira@Marianas-MacBook-Air-98 UDP % ./subscriber_udp 127.0.0.1 RealMadridvsBarcelona
Se ha subscrito al partido: 'RealMadridvsBarcelona'.
Actualización del partido: Gol al minuto 68 (1)
Actualización del partido: Gol al minuto 80 (2)
Actualización del partido: Gol al minuto 24 (3)
Actualización del partido: Gol al minuto 39 (4)
Actualización del partido: Gol al minuto 41 (5)
Actualización del partido: Gol al minuto 63 (6)
Actualización del partido: Gol al minuto 25 (7)
Actualización del partido: Gol al minuto 9 (8)
Actualización del partido: Gol al minuto 54 (9)
Actualización del partido: Gol al minuto 20 (10)
Actualización del partido: Gol al minuto 1 (11)
Actualización del partido: Gol al minuto 36 (12)
Actualización del partido: Gol al minuto 43 (13)
Actualización del partido: Gol al minuto 63 (14)
Actualización del partido: Gol al minuto 28 (15)
Actualización del partido: Gol al minuto 84 (16)
Actualización del partido: Gol al minuto 8 (17)
Actualización del partido: Gol al minuto 80 (18)
Actualización del partido: Gol al minuto 71 (19)
Actualización del partido: Gol al minuto 83 (20)
```

Imagen 1.7. Actualizaciones en el suscriptor al partido de Real Madrid vs Barcelona

Y luego se pueden visualizar como se ve el envio de cada uno de los partidos pasando por el bróker:

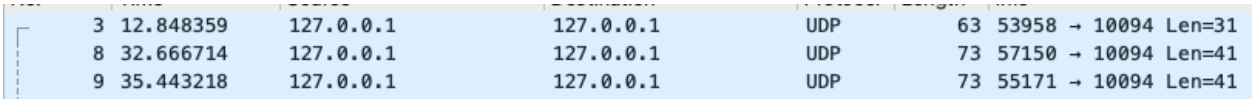
```
maripinemira@Marianas-MacBook-Air-98 UDP % ./broker_udp
Puerto del Broker: 10094
SUBSCRIBE 'RealMadridvsBarcelona' (subs=1)
SUBSCRIBE 'DeportivoCalivsAtleticoNacional' (subs=1)
SUBSCRIBE 'DeportivoCalivsAtleticoNacional' (subs=2)
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 68 (1)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 80 (2)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 24 (3)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 39 (4)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 41 (5)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 63 (6)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 25 (7)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 9 (8)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 54 (9)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 20 (10)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 1 (11)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 36 (12)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 43 (13)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 63 (14)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 28 (15)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 84 (16)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 8 (17)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 80 (18)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 71 (19)'
PUBLISH 'RealMadridvsBarcelona' → 1 subs | 'Gol al minuto 83 (20)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 68 (1)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 80 (2)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 24 (3)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 39 (4)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 41 (5)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 63 (6)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 25 (7)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 9 (8)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 54 (9)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 20 (10)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 1 (11)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 36 (12)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 43 (13)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 63 (14)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 28 (15)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 84 (16)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 8 (17)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 80 (18)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 71 (19)'
PUBLISH 'DeportivoCalivsAtleticoNacional' → 2 subs | 'Gol al minuto 83 (20)'
```

Imagen 1.8. Actualizaciones vistas en el bróker de los dos partidos.

En este caso, en ninguno de los dos suscriptores se evidencian perdidas, esto ya que el correr el sistema en una red local no hay una alta probabilidad de perdida de paquetes. Así mismo, tampoco llegan paquetes desordenados. Sin embargo, si este sistema se ejecutara en una red con una mayor congestión UDP no tiene ningún mecanismo para la perdida de paquetes o perdida de orden entre ellos.

Cuando se realizó la ejecución, también se realizó una captura de tráfico por medio de Wireshark, en donde se obtuvieron los siguientes resultados:

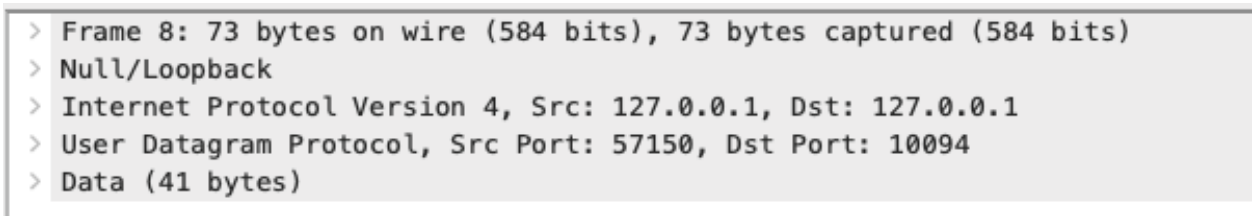
Los primeros tres paquetes son las suscripciones a los diferentes partidos:



3	12.848359	127.0.0.1	127.0.0.1	UDP	63	53958 → 10094 Len=31
8	32.666714	127.0.0.1	127.0.0.1	UDP	73	57150 → 10094 Len=41
9	35.443218	127.0.0.1	127.0.0.1	UDP	73	55171 → 10094 Len=41

Imagen 1.9. Captura de paquetes de suscripciones a los partidos

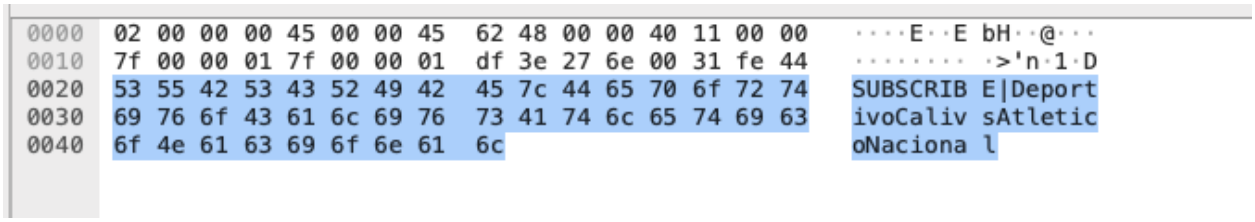
Dentro del paquete, la información del paquete se ve como:



```
> Frame 8: 73 bytes on wire (584 bits), 73 bytes captured (584 bits)
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 57150, Dst Port: 10094
> Data (41 bytes)
```

Imagen 1.10. Ejemplo información de un paquete enviado por UDP

Y luego los datos del segmento se ven como:

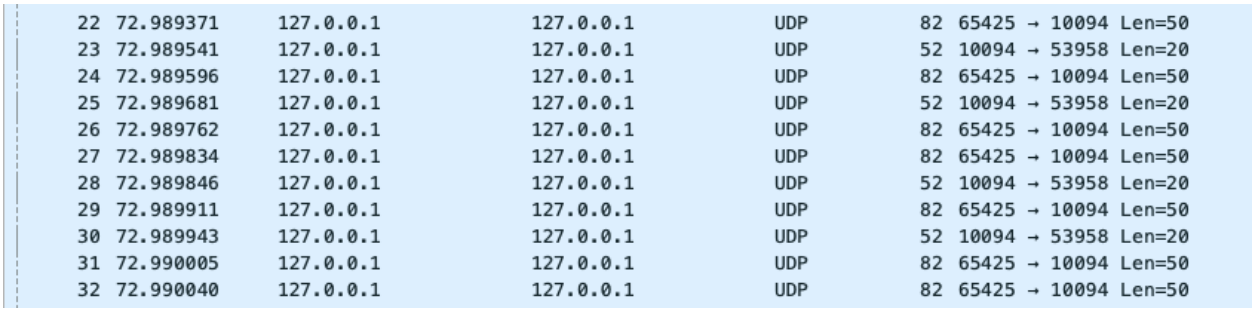


```
0000  02 00 00 00 45 00 00 45  62 48 00 00 40 11 00 00   ····E··E bH··@···
0010  7f 00 00 01 7f 00 00 01  df 3e 27 6e 00 31 fe 44   ········ ·>'n·1·D
0020  53 55 42 53 43 52 49 42  45 7c 44 65 70 6f 72 74   SUBSCRIB E|Deport
0030  69 76 6f 43 61 6c 69 76  73 41 74 6c 65 74 69 63   ivoCaliv sAtletic
0040  6f 4e 61 63 69 6f 6e 61  6c                      oNaciona l
```

Imagen 1.11. Datos dentro del paquete

En donde se evidencia que se tiene la suscripción al partido de Deportivo Cali vs Atlético Nacional.

Luego ya se realiza el envió de los datos:



22	72.989371	127.0.0.1	127.0.0.1	UDP	82	65425 → 10094 Len=50
23	72.989541	127.0.0.1	127.0.0.1	UDP	52	10094 → 53958 Len=20
24	72.989596	127.0.0.1	127.0.0.1	UDP	82	65425 → 10094 Len=50
25	72.989681	127.0.0.1	127.0.0.1	UDP	52	10094 → 53958 Len=20
26	72.989762	127.0.0.1	127.0.0.1	UDP	82	65425 → 10094 Len=50
27	72.989834	127.0.0.1	127.0.0.1	UDP	82	65425 → 10094 Len=50
28	72.989846	127.0.0.1	127.0.0.1	UDP	52	10094 → 53958 Len=20
29	72.989911	127.0.0.1	127.0.0.1	UDP	82	65425 → 10094 Len=50
30	72.989943	127.0.0.1	127.0.0.1	UDP	52	10094 → 53958 Len=20
31	72.990005	127.0.0.1	127.0.0.1	UDP	82	65425 → 10094 Len=50
32	72.990040	127.0.0.1	127.0.0.1	UDP	82	65425 → 10094 Len=50

Imagen 1.12. Captura de tráfico cuando se envían datos,

Se puede ver que para cada paquete se tiene una comunicación del publicador (Puerto: 65425) al Broker (Puerto: 10094) y luego del Broker al Suscriptor (53958), por lo que se puede ver de manera clara como se mandan los mensajes.

Un paquete que se mande desde el publicador hasta el bróker tiene la información:

```
> Frame 37: 82 bytes on wire (656 bits), 82 bytes captured (656 bits)
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 65425, Dst Port: 10094
> Data (50 bytes)
```

Imagen 1.13. Información del paquete que se manda desde el publicador al broker

Y luego el contenido (datos) del mensaje es:

```
02 00 00 00 45 00 00 4e  5d f4 00 00 40 11 00 00   ····E··N ]···@···
7f 00 00 01 7f 00 00 01  ff 91 27 6e 00 3a fe 4d   ········ ··'n·:·M
50 55 42 4c 49 53 48 7c  52 65 61 6c 4d 61 64 72   PUBLISH| RealMadr
69 64 76 73 42 61 72 63  65 6c 6f 6e 61 7c 47 6f   idvsBarc elona|Go
6c 20 61 6c 20 6d 69 6e  75 74 6f 20 31 20 28 31   l al min uto 1 (1
31 29                                              1)
```

Imagen 1.13. Datos del paquete que se manda desde el publicador al broker

En este caso, se puede ver claramente el formato que se tenía definido a la hora de mandar un mensaje desde el publicador al Broker.

Ahora se creó una gráfica en donde se pueden ver los paquetes que llegan a la captura,

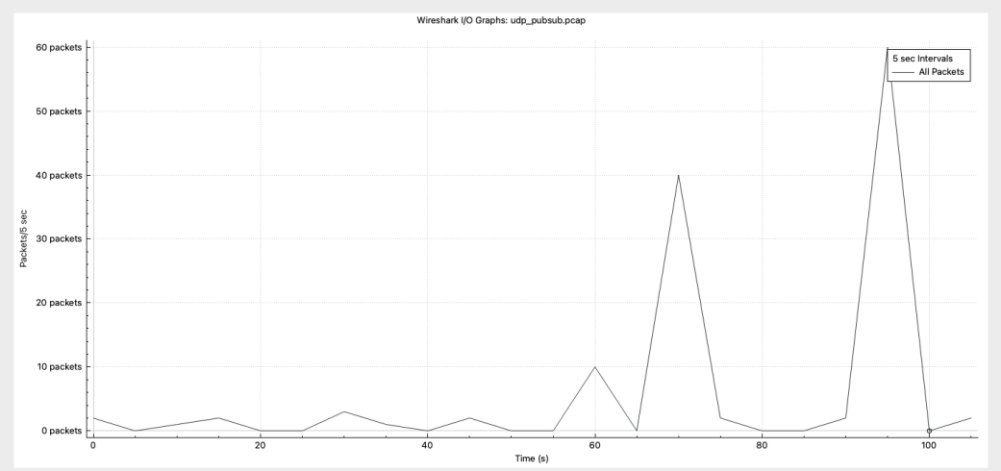


Imagen 1.14. Grafica de captura de paquetes.

En este caso, se pueden ver al inicio pocos paquetes que hacen referencia a cuando los suscriptores se registraron en cada uno de los partidos. Y luego, se pueden ver dos picos que hacen referencia a las 20 actualizaciones que se hicieron por partido. Se espera que cuando se manden una gran cantidad de paquetes se llegue a un pico ya que se están enviando varias actualizaciones de manera simultánea.

Ahora, con respecto a si hay evidencia de pérdida o desorden de paquetes en UDP no se tiene un mecanismo que avise si hay alguna perdida de paquetes, lo que produce falta de

confiabilidad y ausencia de control de orden de entrega. En el sistema construido, al final de cada mensaje se escribió un numero de secuencia, pero esto para que nosotros pudiéramos ver si se perdía o desordenaba un paquete para el análisis. Sin embargo, en UDP no se sabe nunca si se pierde un paquete o llega fuera de orden. Cuando un publicador transmite muchas actualizaciones consecutivas, no todos los mensajes van a llegar a los suscriptores, sin embargo, ni el emisor ni el receptor van a saber que se perdieron paquetes. Adicionalmente, los paquetes puede que no lleguen en el mismo orden que se enviaron, lo cual en este tipo de sistemas de actualizaciones de partidos podría no ser bueno ya que al ser noticias que deben llegar de manera temporal, se necesita un orden.

2. TCP

Para el desarrollo del segundo ejercicio propuesto, el cual buscaba replicar el protocolo de capa de transporte TCP, se siguió la estructura publicador-subscriptor ya mencionada anteriormente realizando las modificaciones respectivas para la creación y soporte de sesión. Este modelo permite que según los eventos o partidos a los cuales este subscrito cada subscriptor, reciba las respectivas actualizaciones sobre cada partido simultáneamente, mientras los publicadores realizan la transmisión de estas de forma persistente.

Este sistema está compuesto por tres componentes principales:

- Publicador, envía actualizaciones de partidos con el formato de mensaje:
  "Gol %d del %s!", donde d es un contador que permite verificar el mensaje que ha sido enviado y s es cualquiera de los siguientes temas: PARTIDO_A, PARTIDO_B, PARTIDO_C.

  Como para el laboratorio se propone el uso de un broker, no es necesario saber el puerto directo de los subscriptores debido a la estructura que en ellos se maneja. Por otro lado, al ejecutar al publicador este envía 10 mensajes en ráfaga que son con los cuales se verifica el comportamiento del programa.
- Broker, este componente actúa como intermediario entre los publicadores y suscriptores. Mantiene abiertas las conexiones de publicadores y subscriptores mientras enruta los mensajes según el tema al que el subscriptor apunte. Los publicadores envían mensajes con la estructura PUB [tema] [mensaje] y el broker se encarga de realizar la lectura de estos elementos y según los temas que sigan los

subscriptores se les reenvía el mensaje. El ruteo diseñado se basa en las conexiones guardadas por conexión, ya que de esta forma solo se necesita conocer la dirección y puerto del broker por parte del publicador mientras el broker distribuye a los subscriptores a través de sus conexiones establecidas.

- Suscriptores, se registran a un partido según el tema que escojan al momento de establecer la conexión con el broker con SUB [tema]. Por lo que ellos también definen el tema a escoger y con el mismo puerto acceden a los mensajes enviados. Una vez el tema es definido espera a recibir las actualizaciones que serán enviadas por el bróker. En la consola se pueden observar las diferentes actualizaciones que se reciben del parido al que se encuentra suscrito.

Para este protocolo el broker mantiene una tabla de clientes donde cada entrada representa una conexión TCP activa donde guarda el socket del cliente, un búfer para ensamblar líneas y los temas a los que ese cliente se ha suscrito. Cuando llega una línea SUB [partido], el broker asocia ese partido a la conexión que la envió, cuando llega una línea PUB [partido][mensaje], ejecuta la función de ruteo broadcast_topic() que reencapsula todo el mensaje, pero como un MSG [partido] [mensaje] y envía el resultado a todas las conexiones que estén suscritas a ese partido. El manejo de múltiples conexiones se hace con la función select() donde el broker acepta nuevas conexiones, lee las existentes, parsea las líneas terminadas en \n (para saber que se terminó el mensaje) y reparte.

En este caso, la implementación realizada para TCP resulta muy adecuada pues para cada flujo la cronología de eventos simplifica la lógica de la aplicación: los suscriptores no necesitan corregir desorden ni recuperar mensajes faltantes, y el publicador solo conoce el puerto del broker, encargándose de todo el ruteo sobre sockets ya establecidos.

Algunas ventajas de implementar este sistema con TCP son:

- Orden garantizado: las actualizaciones llegan en secuencia dentro de cada conexión, manteniendo la línea de tiempo del partido.
- Confiabilidad: retransmisiones y ACKs evitan pérdidas silenciosas de eventos críticos.
- Flujo integrado: el control de flujo con las funciones select() y send() evita saturar a suscriptores lentos.

- Simplicidad de direccionamiento: no se gestionan puertos por cada suscriptor, por facilidad de implementación basta el socket abierto.

De esta forma, TCP ofrece robustez y claridad en la entrega, lo que facilita el análisis con Wireshark, el depurado y la correcta experiencia del usuario al mantener las actualizaciones completas y en orden.

Para las pruebas sobre la implementación se ejecutaron las tres estructuras como se ve a continuación:

Primero se ejecuta el broker al ser él quien direccionara todo el flujo de mensajera. Esto es vital pues de no hacerlo ninguno de los hosts (publicador-subscriptor) funcionaria al no tener conexión.

Imagen 2.1. Ejecución del Broker.

Luego se inicializan dos subscriptores quienes estarán esperando la llegada de algún mensaje según el partido seleccionado. El primero de ellos que por defecto esta vinculado al PARTIDO_A y el segundo vinculado a PARTIDO_A y PARTIDO_B, esto para verificar como es el manejo de recepción de mensajes para un mismo subscriptor. Una vez realizado esto se ejecutan los respectivos publicadores y gracias a ellos se ve la llegada de mensajes.

Imagen 2.2. Publicador base PARTIDO_A.

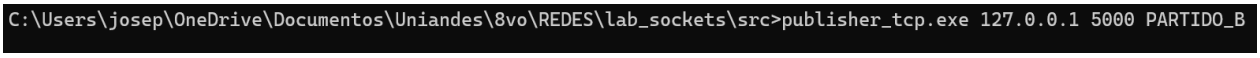


Imagen 2.3 Publicador PARTIDO_B.

Las imágenes a continuación muestran la llegada de mensajes a los respectivos subscriptores. Aunque el orden de las imágenes anteriores no describe el formato de ejecución del programa, se optó por realizar esto para reducir la extensión del informe para la muestra del protocolo:

```
C:\Users\josep\OneDrive\Documentos\Uniandes\8vo\REDES\lab_sockets\src>subscriber_tcp
Suscrito a PARTIDO_A. Esperando mensajes...
MSG PARTIDO_A Gol 1 del PARTIDO_A!
MSG PARTIDO_A Gol 2 del PARTIDO_A!
MSG PARTIDO_A Gol 3 del PARTIDO_A!
MSG PARTIDO_A Gol 4 del PARTIDO_A!
MSG PARTIDO_A Gol 5 del PARTIDO_A!
MSG PARTIDO_A Gol 6 del PARTIDO_A!
MSG PARTIDO_A Gol 7 del PARTIDO_A!
MSG PARTIDO_A Gol 8 del PARTIDO_A!
MSG PARTIDO_A Gol 9 del PARTIDO_A!
MSG PARTIDO_A Gol 10 del PARTIDO_A!
```

Imagen 2.4. Suscripción al partido A, por defecto.

```
C:\Users\josep\OneDrive\Documentos\Uniandes\8vo\REDES\lab_sockets\src>subscriber_tcp.exe 127.0.0.1 5000 PARTIDO_A PARTID
O_B
Suscrito a: PARTIDO_A, PARTIDO_B. Esperando mensajes...
MSG PARTIDO_A Gol 1 del PARTIDO_A!
MSG PARTIDO_A Gol 2 del PARTIDO_A!
MSG PARTIDO_A Gol 3 del PARTIDO_A!
MSG PARTIDO_A Gol 4 del PARTIDO_A!
MSG PARTIDO_A Gol 5 del PARTIDO_A!
MSG PARTIDO_A Gol 6 del PARTIDO_A!
MSG PARTIDO_A Gol 7 del PARTIDO_A!
MSG PARTIDO_A Gol 8 del PARTIDO_A!
MSG PARTIDO_A Gol 9 del PARTIDO_A!
MSG PARTIDO_A Gol 10 del PARTIDO_A!
MSG PARTIDO_B Gol 1 del PARTIDO_B!
MSG PARTIDO_B Gol 2 del PARTIDO_B!
MSG PARTIDO_B Gol 3 del PARTIDO_B!
MSG PARTIDO_B Gol 4 del PARTIDO_B!
MSG PARTIDO_B Gol 5 del PARTIDO_B!
MSG PARTIDO_B Gol 6 del PARTIDO_B!
MSG PARTIDO_B Gol 7 del PARTIDO_B!
MSG PARTIDO_B Gol 8 del PARTIDO_B!
MSG PARTIDO_B Gol 9 del PARTIDO_B!
MSG PARTIDO_B Gol 10 del PARTIDO_B!
```

Imagen 2.5. Suscripción al partido A y B.

Cuando se realizó la ejecución, también se realizó una captura de tráfico por medio de Wireshark, en donde se obtuvieron los siguientes resultados:

Al realizar un filtro sobre la información que utilizó el puerto 5000, definido para el envío de paquetes con TCP, con el que se obtuvo una gran cantidad de datos como se ve en la imagen a continuación.

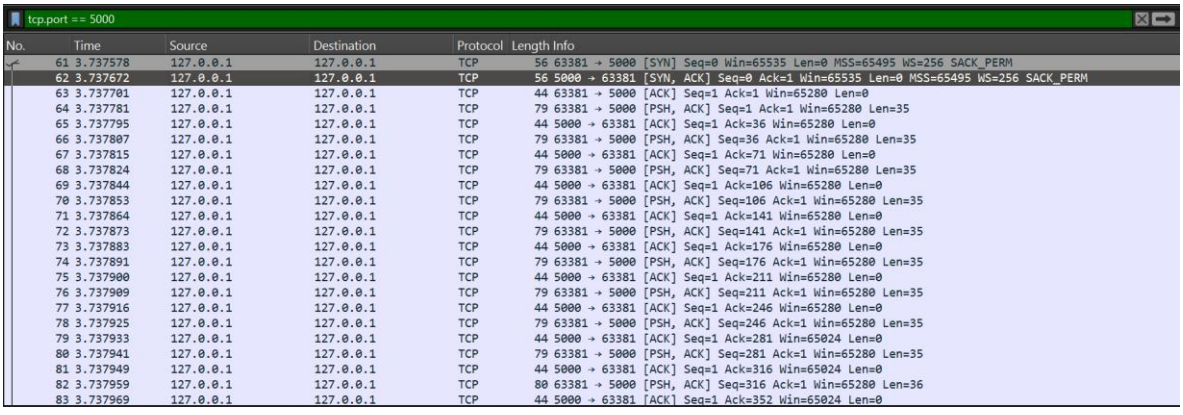


Imagen 2.6. Captura de datos Wireshark con filtro port ==5000.

Al revisar los primeros paquetes transmitidos estos concuerdan con el protocolo de 3-way handshake pues en ellos se realiza la transmisión contigua de datos desde el puerto 63381 al 5000, luego del 5000 al 63381 y finalmente la misma transmisión inicial. Adicionalmente, como se señala en la imagen a continuación, se destacan las flags de sincronización (SYN) y reconocimiento de los datos (ACK) según el paso en el que se encuentre el proceso.

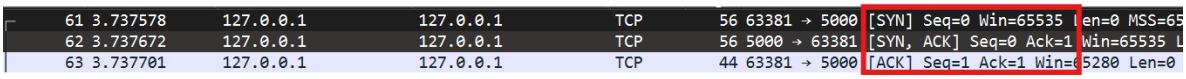


Imagen 2.7. Captura del protocolo 3-way handshake.

Siguiendo con la secuencia de flujo se observa como el propio wireshark denota el envio desde un publicador hasta un subscriptor por los puertos en los que pasa cada paquete. Así, se observa que para cada paquete se cumple el recorrido entre publicador-brocker, brocker-subscriptor, ya que para cada uno de los 10 mensajes enviados existe un par de paquetes transmitidos.

Asi mismo, wireshark permite visualizar el fin de envió de paquetes por un publicador, gracias a que al final de la siguiente imagen se puede observar como una vez termina el flujo del publicador PARTIDO_A se envía un mensaje con la flag de terminación de envio FIN.

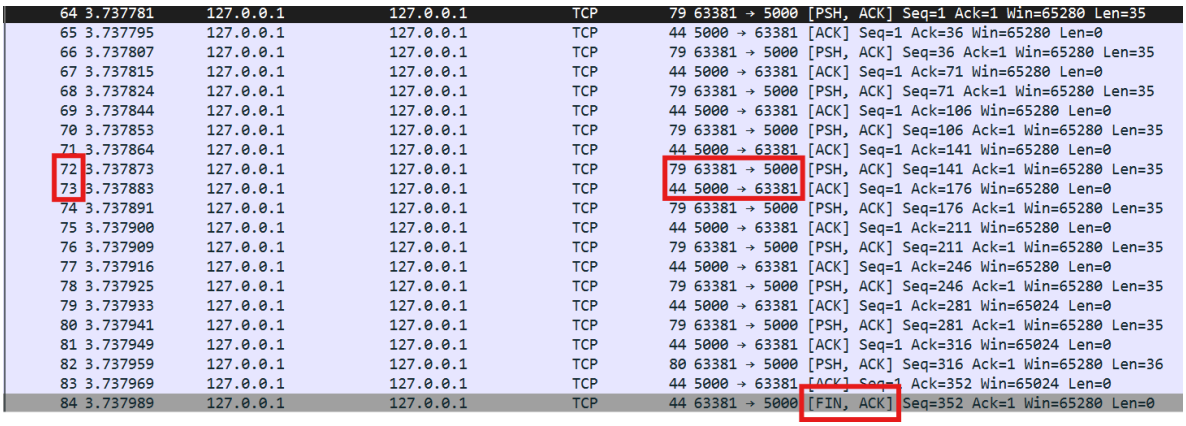


Imagen 2.8. Captura del envío de paquetes publicador PARTIDO_A.

Para confirmar la correcta secuencia de envío se explora la información del paquete 72 señalado en la imagen anterior el cual se esperaría sea el quinto paquete enviado.

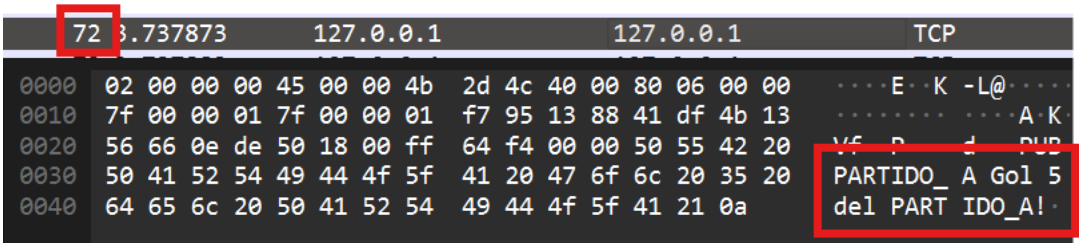


Imagen 2.9. Captura del paquete 72.

Por otro lado, para la verificar el establecimiento de sesión y envío de paquetes hacia los suscriptores del mismo tipo de tema (PARTIDO_A), se esperaría que se hayan 2 protocolos de inicio o fin de sesión que reciban la información desde la misma fuente. Con eso en mente se realizó el siguiente filtro sobre la información en donde se observan el trafico dado en el puerto 5000 y todo lo que fue emitido desde la dirección 127.0.0.1, lo que en otras palabras permitiría visualizar el tráfico del broker a sus subscriptores. La siguiente imagen ilustra como se da fin a ambas sesiones que reciben el mismo flujo.

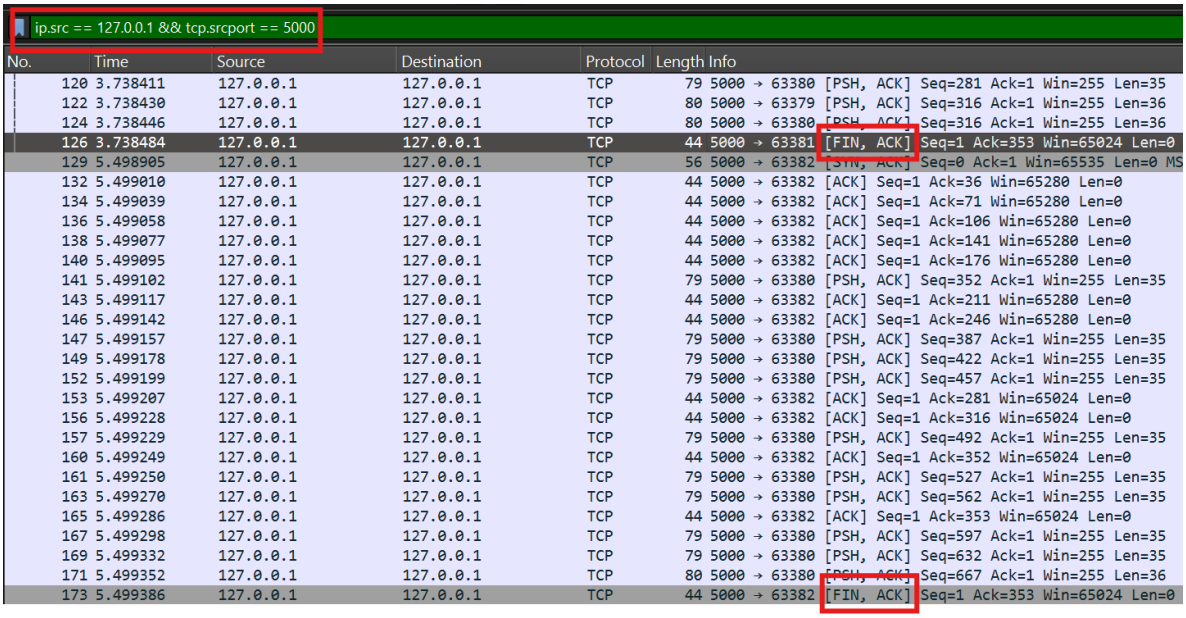


Imagen 2.10. Captura del tráfico del broker hacia subscriptores.

Finalmente se realiza la siguiente grafica de flujo que ilustra como se da el comportamiento de envio y recepción de paquetes en el tiempo de las pruebas.

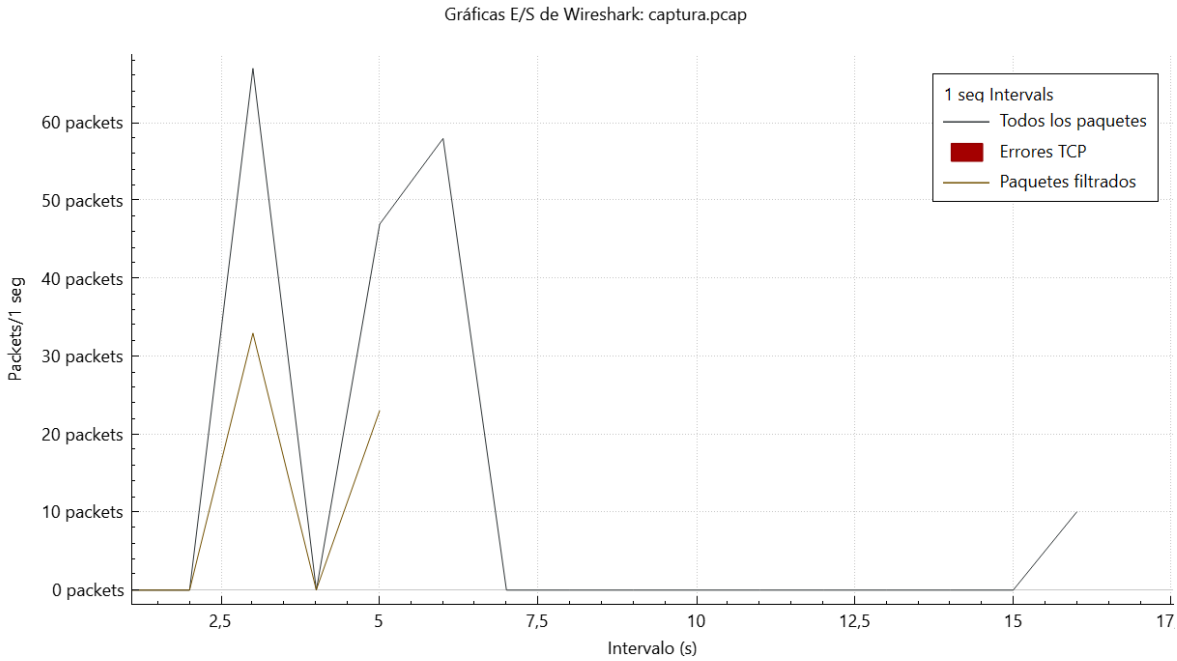


Imagen 2.11. Captura del tráfico del broker hacia subscriptores.

En el trazado se distinguen ráfagas claras de actividad: tras un periodo inicial sin tráfico, aparece un pico pronunciado, seguido de una caída y luego otra ráfaga. Ese patrón es coherente con el handshake y los envíos de publicaciones del los publicadores y los reenvíos del broker hacia los suscriptores. La curva de “Paquetes filtrados” queda por debajo de “Todos los paquetes” porque solo cuenta los mensajes de aplicación (excluye ACKs y control), y la línea de errores plana apoya que TCP entregó en orden y sin pérdidas visibles. En general, lo que la gráfica anterior evidencia es el flujo por ráfagas típico del protocolo realizado, con periodos cortos de alta actividad y luego inactividad hasta la siguiente tanda de mensajes. Al final se alcanza a observar una subida en los paquetes la cual corresponde a otro envio que se había iniciado antes de detener la toma de paquetes.

3. Comparación TCP y UDP

Criterio	TCP	UDP
Confiabilidad	Utiliza confirmaciones (ACK), retransmisiones y control de errores. Garantiza que los datos lleguen completos y sin corrupción. En las pruebas, todos los mensajes llegaron correctamente.	No hay confirmaciones ni retransmisiones. En redes locales puede no haber pérdidas (como en las pruebas), pero en redes congestionadas es probable perder mensajes.
Orden de entrega	Los números de secuencia aseguran que los mensajes lleguen en el orden enviado. Crucial para eventos deportivos donde la secuencia temporal es importante.	Los datagramas pueden llegar en cualquier orden. En las pruebas no se observó desorden, pero en redes reales es posible, lo que afectaría la coherencia de las actualizaciones.
Perdida de mensajes	Los mecanismos de retransmisión recuperan paquetes perdidos automáticamente. Ideal para eventos críticos como goles.	Depende de la congestión de la red. En redes locales la pérdida puede ser cero, pero en entornos reales es significativa. No hay recuperación automática.
Overhead de cabeceras/protocolo	Cabeceras de 20 a 60 bytes con campos para control de flujo, congestión y secuencia. Aumenta el ancho de banda consumido, especialmente para mensajes pequeños.	Cabeceras fijas de 8 bytes. Más eficiente en ancho de banda, ideal para actualizaciones frecuentes y pequeñas.

4. Preguntas de análisis

• ¿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?

TCP: El broker tendría que manejar múltiples conexiones persistentes (una por suscriptor), consumiendo más recursos de memoria y CPU debido a los buffers de cada conexión. El control de congestión de TCP podría ralentizar la transmisión en redes saturadas, pero garantizaría la entrega. La escalabilidad es limitada debido al overhead por conexión.

UDP: El broker manejaría datagramas independientes, lo que es más escalable en términos de conexiones. Sin embargo, la falta de control de congestión podría causar pérdida masiva de mensajes en redes congestionadas. La eficiencia en el envío a múltiples suscriptores es mayor, pero la confiabilidad se vería comprometida.

• 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?

En una aplicación real, los usuarios perderían eventos críticos como goles, lo que afectaría la experiencia de usuario y la confiabilidad del servicio. Por ejemplo, un suscriptor podría no enterarse de un gol importante, llevando a insatisfacción y desconfianza en la plataforma. TCP maneja mejor este escenario porque TCP incluye mecanismos de confirmación (ACK) y retransmisión. Si un mensaje no llega, el receptor solicita su reenvío automáticamente. Esto garantiza que los eventos críticos lleguen eventualmente, aunque con posible retraso, pero sin pérdida.

• En un escenario de seguimiento en vivo de partidos, ¿qué protocolo (TCP o UDP) resultaría más adecuado?

UDP sería más adecuado para la mayoría de las actualizaciones debido a su menor latencia y overhead, lo que es crucial para tiempo real. Sin embargo, para eventos críticos como goles o tarjetas, se recomienda usar TCP o implementar mecanismos de confiabilidad sobre UDP. En las pruebas, UDP funcionó bien en red local, pero en entornos reales con congestión, se necesitarían ajustes para garantizar confiabilidad.

• 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 introduce más cabeceras por mensaje (20-60 bytes) en comparación con UDP (8 bytes). Esto se observó en las capturas de Wireshark, donde los paquetes TCP incluían campos

adicionales para control de flujo y secuencia. Esto reduce la eficiencia en ancho de banda, especialmente para mensajes pequeños y frecuentes, como actualizaciones deportivas. UDP es más eficiente en este aspecto.

• Si el marcador de un partido llega desordenado en UDP, ¿qué efectos tendría en la experiencia del usuario? ¿Cómo podría solucionarse este problema a nivel de aplicación?

El usuario vería información inconsistente y confusa, lo que reduciría la confianza en la aplicación. Por ejemplo, ver un marcador 2-1 antes de 1-1 arruinaría la secuencia temporal de eventos. Para solucionar esto se podrían implementar números de secuencia en los mensajes (como se hizo en las pruebas con la numeración 1-20) y un buffer en el suscriptor para reordenar los mensajes antes de mostrarlos. También se podrían usar timestamps para sincronizar y priorizar la entrega.

• ¿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: Cada suscriptor adicional requiere una conexión separada, por lo que el broker debe enviar el mismo mensaje múltiples veces, una por cada conexión. Esto consume más ancho de banda y recursos en el broker, limitando la escalabilidad.

UDP: El broker puede enviar el mismo datagrama una vez y dirigirlo a múltiples suscriptores. Esto es más eficiente, como se vio en el partido con dos suscriptores donde el broker envió a ambos sin duplicar esfuerzo.

• ¿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 se cierran de manera controlada, y los clientes detectan inmediatamente la caída debido a la pérdida de conexión. Esto permite una reconexión rápida o notificación al usuario

UDP: No hay detección automática de caída. Los clientes continúan enviando mensajes al broker caído sin recibir respuesta. Se necesitan mecanismos de heartbeat o timeouts a nivel de aplicación para detectar la caída y reconectar, lo que añade complejidad.

• ¿Cómo garantizar que todos los suscriptores reciban en el mismo instante las actualizaciones críticas? ¿Qué protocolo facilita mejor esta sincronización y por qué?

Para garantizar que todos los suscriptores reciban actualizaciones críticas simultáneamente, se debe implementar multicast a nivel de red o aplicación, junto con mecanismos de sincronización como timestamps. UDP es el protocolo que mejor facilita esta sincronización debido a su capacidad de multidifusión nativa y menor latencia, permitiendo que un solo datagrama llegue a múltiples suscriptores casi al mismo instante, lo que es esencial para eventos en tiempo real como goles en partidos de fútbol.

• 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: El broker consume más CPU y memoria porque debe mantener el estado de cada conexión, incluyendo buffers de envío y recepción, y manejar el control de flujo y congestión para cada una. Esto limita el número de conexiones simultáneas.

UDP: El broker consume menos recursos porque no mantiene estado de conexión. Simplemente recibe y envía datagramas. Sin embargo, debe gestionar manualmente la lista de suscriptores y posibles retransmisiones, lo que añade carga a nivel de aplicación.

• 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?

Una combinación de ambos, usaríamos UDP para la mayoría de las actualizaciones en tiempo real (como posesión, eventos menores) debido a su eficiencia y baja latencia, y TCP para eventos críticos (goles, resultados finales) donde la confiabilidad es esencial. Además, implementaría mecanismos de aplicación sobre UDP para mejorar la confiabilidad cuando sea necesario, como retransmisiones selectivas o confirmaciones, basándose en la numeración secuencial probada en el laboratorio. Esto optimiza escalabilidad sin sacrificar confiabilidad en información crucial.

Link de repositorio de Github:

https://github.com/StnJoseph/lab_sockets.git

En el siguiente link se encuentra toda la información relacionada a la entrega, este informe está en la carpeta docs, las muestras de wireshark en la carpeta capturas, los respectivos códigos en src y la explicación de las librerías en el README.