INFORME LABORATORIO #3 DE INFRAESTRUCTURA DE COMUNICACIÓN

Juan Pablo Rivera - 202211439
José David Romero Caicedo - 202222606
Juan David Cárdenas - 202221834

1. **Implementar el sistema publicador–suscriptor en TCP y en UDP.**

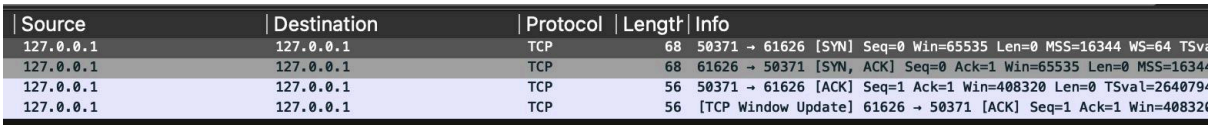
   Archivos adjuntos de:
   - broker_tcp.c
   - publisher_tcp.c
   - subscriber_tcp.c
   - broker_udp.c
   - publisher_udp.c
   - subscriber_udp.c

2. **No está permitido el uso de librerías de implementación de sockets a menos que se especifique punto a punto la interacción del programa con cada función de la librería, debe estar totalmente documentado.**

3. **Envío de 10 mensajes desde cada publicador y verificar que lleguen correctamente a los suscriptores.**

TCP:

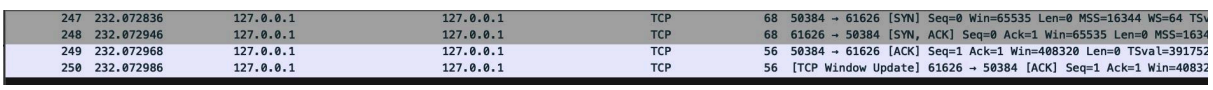
Para ambos casos nuestro broker está escuchando en el puerto 61626, por lo que la conexión con el publisher en wireshark se ve mas o menos así:



Source	Destination	Protocol	Length	Info
127.0.0.1	127.0.0.1	TCP	68	50371 → 61626 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSva
127.0.0.1	127.0.0.1	TCP	68	61626 → 50371 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=16344
127.0.0.1	127.0.0.1	TCP	56	50371 → 61626 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=2640794
127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 61626 → 50371 [ACK] Seq=1 Ack=1 Win=40832

Conexión publisher broker en WireShark

Mientras que la conexión con el subscriber se puede ver de esta manera:



247	232.072836	127.0.0.1	127.0.0.1	TCP	68	50384 → 61626 [SYN] Seq=0 Win=65535 Len=0 MSS=16344 WS=64 TSv
248	232.072946	127.0.0.1	127.0.0.1	TCP	68	61626 → 50384 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1634
249	232.072968	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=1 Ack=1 Win=408320 Len=0 TSval=391752
250	232.072986	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 61626 → 50384 [ACK] Seq=1 Ack=1 Win=40832

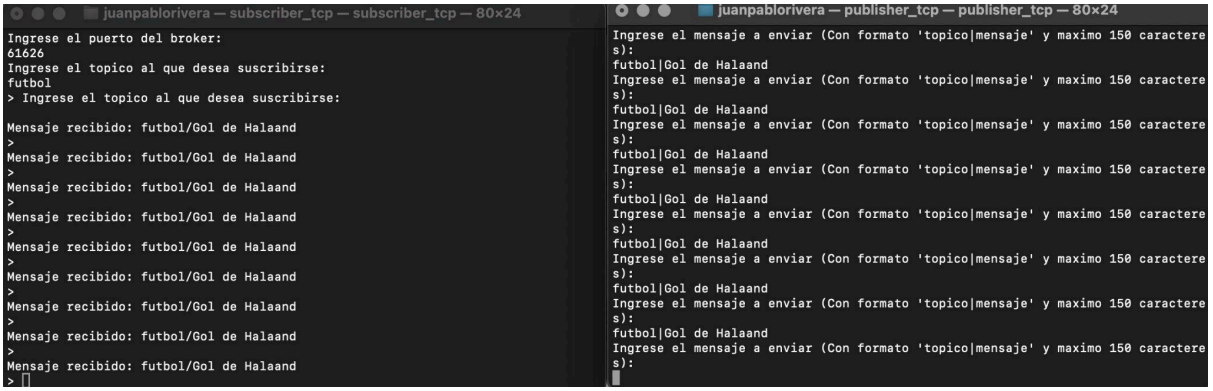
Conexión subscriber broker en WireShark

Y así se observa la suscripción a un tópico

439	399.890677	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
440	399.890726	127.0.0.1	127.0.0.1	TCP	56	61626 → 50384 [ACK] Seq=1 Ack=5 Win=408320 Len=0 TSval=8926847
441	399.890753	127.0.0.1	127.0.0.1	WebSocket	68	WebSocket Continuation[FRAGMENT]
442	399.890758	127.0.0.1	127.0.0.1	TCP	56	61626 → 50384 [ACK] Seq=1 Ack=17 Win=408320 Len=0 TSval=892684

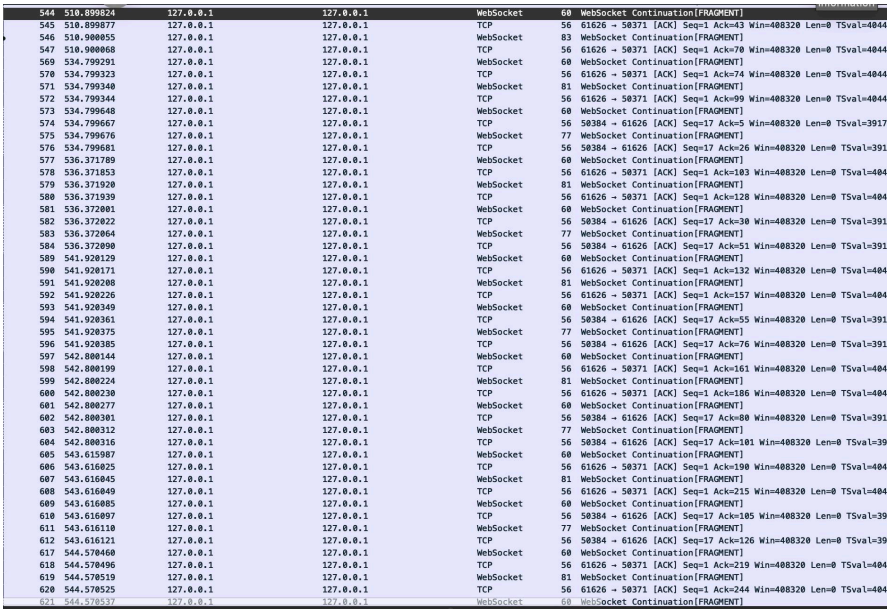
Subscripción a tópico desde el subscriber al broker en WireShark

Vamos a simular el envío de 10 paquetes en el tópico “futbol” para ver que si está funcionando:



Paquetes enviados en TCP funcionan

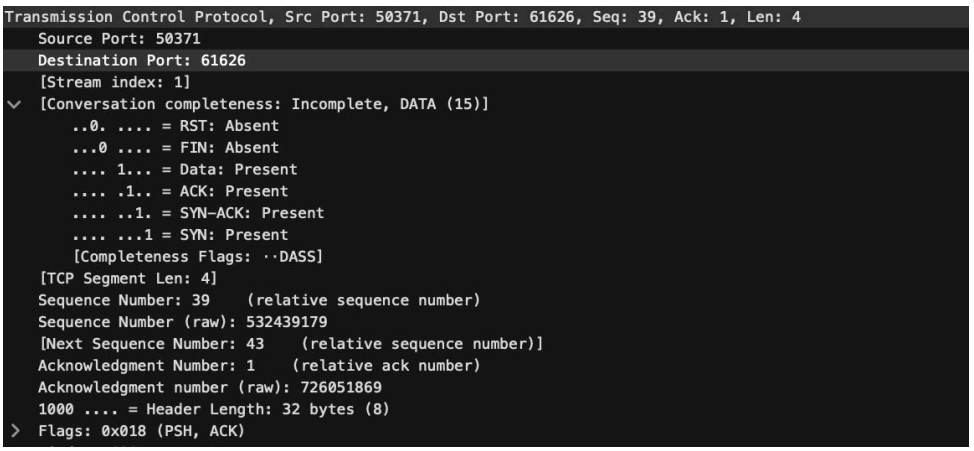
En este caso podemos ver que el subscriber está efectivamente recibiendo los mensajes mandados por el publisher, en wireshark se ve mas o menos así:



544	510.899824	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
545	510.899877	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=43 Win=408320 Len=0 TSval=4044
546	510.900055	127.0.0.1	127.0.0.1	WebSocket	83	WebSocket Continuation[FRAGMENT]
547	510.900068	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=70 Win=408320 Len=0 TSval=4044
569	534.799291	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
570	534.799323	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=74 Win=408320 Len=0 TSval=4044
571	534.799340	127.0.0.1	127.0.0.1	WebSocket	81	WebSocket Continuation[FRAGMENT]
572	534.799344	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=99 Win=408320 Len=0 TSval=4044
573	534.799648	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
574	534.799667	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=5 Win=408320 Len=0 TSval=3917
575	534.799676	127.0.0.1	127.0.0.1	WebSocket	77	WebSocket Continuation[FRAGMENT]
576	534.799681	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=26 Win=408320 Len=0 TSval=391
577	536.371789	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
578	536.371853	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=103 Win=408320 Len=0 TSval=404
579	536.371920	127.0.0.1	127.0.0.1	WebSocket	81	WebSocket Continuation[FRAGMENT]
580	536.371939	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=128 Win=408320 Len=0 TSval=404
581	536.372001	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
582	536.372022	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=30 Win=408320 Len=0 TSval=391
583	536.372064	127.0.0.1	127.0.0.1	WebSocket	77	WebSocket Continuation[FRAGMENT]
584	536.372090	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=51 Win=408320 Len=0 TSval=391
589	541.920129	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
590	541.920171	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=132 Win=408320 Len=0 TSval=404
591	541.920208	127.0.0.1	127.0.0.1	WebSocket	81	WebSocket Continuation[FRAGMENT]
592	541.920226	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=157 Win=408320 Len=0 TSval=404
593	541.920349	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
594	541.920361	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=55 Win=408320 Len=0 TSval=391
595	541.920375	127.0.0.1	127.0.0.1	WebSocket	77	WebSocket Continuation[FRAGMENT]
596	541.920385	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=76 Win=408320 Len=0 TSval=391
597	542.800144	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
598	542.800199	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=161 Win=408320 Len=0 TSval=404
599	542.800224	127.0.0.1	127.0.0.1	WebSocket	81	WebSocket Continuation[FRAGMENT]
600	542.800230	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=186 Win=408320 Len=0 TSval=404
601	542.800277	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
602	542.800301	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=80 Win=408320 Len=0 TSval=391
603	542.800312	127.0.0.1	127.0.0.1	WebSocket	77	WebSocket Continuation[FRAGMENT]
604	542.800316	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=101 Win=408320 Len=0 TSval=39
605	543.615987	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
606	543.616025	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=190 Win=408320 Len=0 TSval=404
607	543.616045	127.0.0.1	127.0.0.1	WebSocket	81	WebSocket Continuation[FRAGMENT]
608	543.616049	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=215 Win=408320 Len=0 TSval=404
609	543.616085	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
610	543.616097	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=105 Win=408320 Len=0 TSval=39
611	543.616110	127.0.0.1	127.0.0.1	WebSocket	77	WebSocket Continuation[FRAGMENT]
612	543.616121	127.0.0.1	127.0.0.1	TCP	56	50384 → 61626 [ACK] Seq=17 Ack=126 Win=408320 Len=0 TSval=39
617	544.570460	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]
618	544.570496	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=219 Win=408320 Len=0 TSval=404
619	544.570519	127.0.0.1	127.0.0.1	WebSocket	81	WebSocket Continuation[FRAGMENT]
620	544.570525	127.0.0.1	127.0.0.1	TCP	56	61626 → 50371 [ACK] Seq=1 Ack=244 Win=408320 Len=0 TSval=404
621	544.570537	127.0.0.1	127.0.0.1	WebSocket	60	WebSocket Continuation[FRAGMENT]

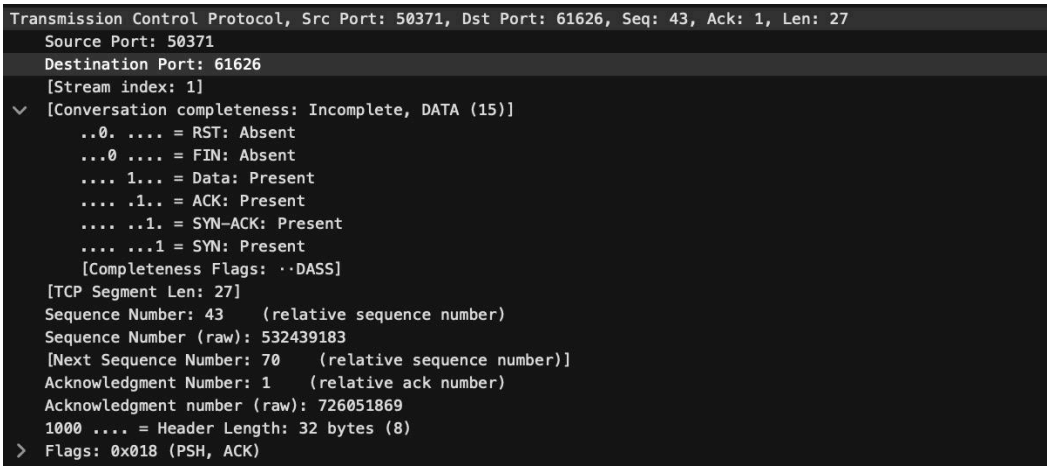
Captura de WireShark de los 10 paquetes enviados por TCP

En este caso, podemos dividir los envíos en bloques de a 4, los paquetes 2 y 4, son más bien confirmaciones entre el publisher con el broker y del subscriber con el broker, mientras que el 1, envía 4 bytes al broker indicando la cantidad de bytes que tiene el payload completo, funcionando como un header:



Paquete con el header de TCP

El 4 es el que contiene los datos, que es el envio al broker mediante el WebSocket como es el caso de este paquete:



Paquete con el PayLoad de TCP

Podemos identificarlo ya que usa el protocolo de WebSocket y la longitud de los datos enviados es mucho mayor al resto.

Esto lo sabemos ya que en nuestra implementación, primero enviamos el resultado1, que es el encabezado y luego enviamos el resultado2 que es el payload.

```c
static int send_frame_tcp(
#ifdef _WIN32
    SOCKET socket_tcp,
#else
    int socket_tcp,
#endif
    const char* payload, int len)
{

    uint32_t nlen = htonl((uint32_t)len);
    // Enviar longitud
    int resultado1 = send_all_tcp(socket_tcp, (const char*)&nlen, sizeof(nlen));
#ifdef _WIN32
    if (resultado1 == SOCKET_ERROR) return SOCKET_ERROR;
#else
    if (resultado1 < 0) {
        return -1;
    }
#endif
    // Enviar payload
    int resultado2 = send_all_tcp(socket_tcp, payload, len);
#ifdef _WIN32
    if (resultado2 == SOCKET_ERROR) return SOCKET_ERROR;
#else
    if (resultado2 < 0) {
        return -1;
    }
#endif
    return resultado1 + resultado2;
}
```

Codigo implementación de TCP

Una vez con los datos, el broker envía los mensajes al subscriber, como se pueden ver en la imagen cuando del puerto 61626 (donde esta el broker), enviamos al puerto donde tenemos el subscriber.

Para el pcap, recomendamos filtrar con el tcp.port == 61626 para poder ver el envío claro de los paquetes.

UDP:

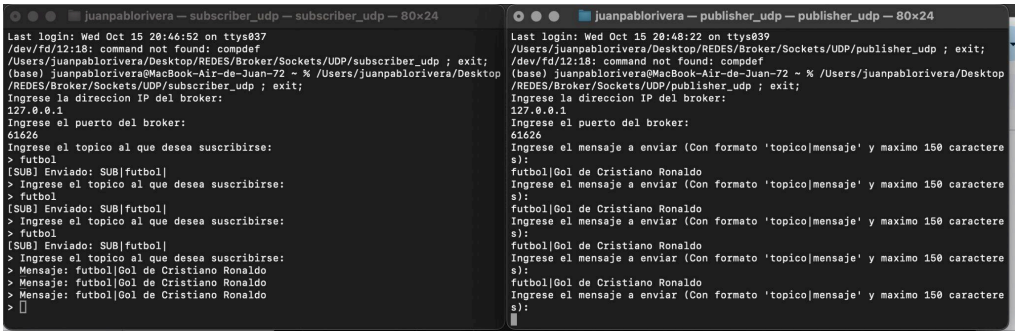
En este caso, de igual forma, nuestro broker está funcionando en el puerto 61626, por lo que vamos a usar el filtro de udp.port==61626 en Wireshark para poder filtrar los paquetes mandados ahí.

Empecemos mostrando el momento en el que el subscriber se inscribe a un tópico en el broker. En este caso, encontramos este paquete, que corresponde simplemente a la solicitud al broker de suscribirse al tema “futbol”:

```
> Frame 10: 43 bytes on wire (344 bits), 43 bytes captured (344 bits) on interface lo0, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
v User Datagram Protocol, Src Port: 49280, Dst Port: 61626
    Source Port: 49280
    Destination Port: 61626
    Length: 19
    Checksum: 0xfe26 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 3]
  > [Timestamps]
    UDP payload (11 bytes)
```

Suscriptor de inscribir a un tema del broker

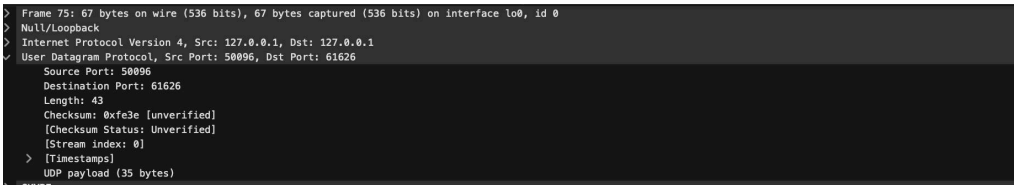
Ahora, vamos a enviar un paquete de prueba para analizarlo:



Paquete de prueba enviado en UDP

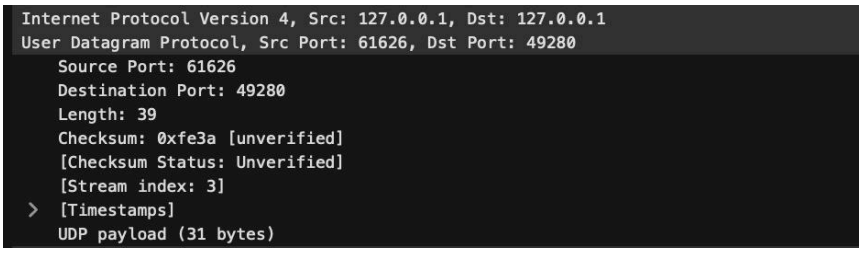
En este caso, en WireShark, vamos a observar 2 paquetes, uno de envío al broker y otro en el que el broker manda al subscriber, debido a que UDP no tiene que hacer handshake ya que su función principal no es la de enviar seguro sino de enviar rápido.

En este primer paquete, podemos identificar rápidamente que es del publisher al broker, ya que el puerto de destino es en el que está alojado el broker, tiene una longitud de 43, por lo que aún más podemos confirmar que se trata, ya que contiene la longitud del mensaje enviado.



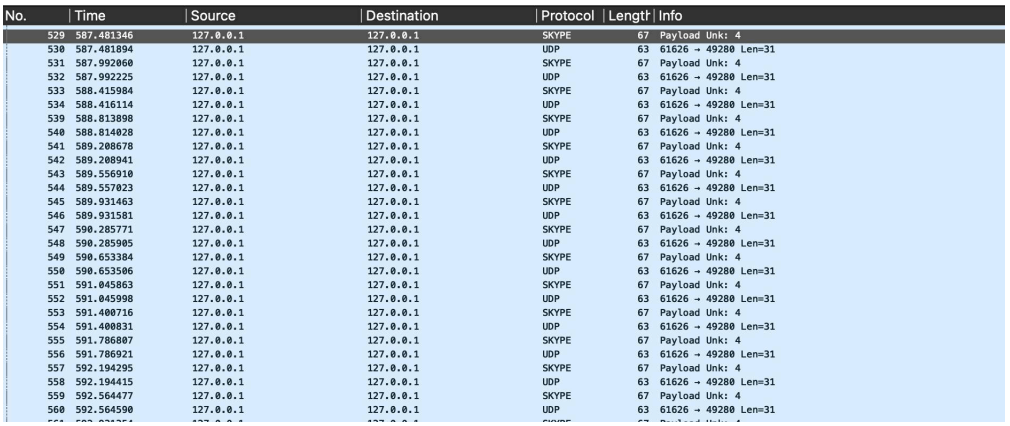
PayLoad del UDp publicador al broker

En este segundo paquete, vemos que se trata del envío del broker al subscriber, algo interesante de ver es que la longitud es diferente, a pesar de ser el mismo mensaje, debido a que estamos usando un formato de tópico|mensaje, el broker elimina el tópico, lo que hace que se disminuya la longitud del mensaje enviado al subscriber.



PayLoad UDP del broker al suscriptor

Por último, vamos a enviar los 10 paquetes para probar que funciona:



No.	Time	Source	Destination	Protocol	Length	Info
529	587.481346	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
530	587.481894	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
531	587.992060	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
532	587.992225	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
533	588.415984	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
534	588.416114	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
539	588.813898	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
540	588.814028	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
541	589.208678	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
542	589.208941	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
543	589.556910	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
544	589.557023	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
545	589.931463	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
546	589.931581	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
547	590.285771	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
548	590.285905	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
549	590.653384	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
550	590.653506	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
551	591.045863	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
552	591.045998	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
553	591.400716	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
554	591.400831	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
555	591.786807	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
556	591.786921	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
557	592.194295	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
558	592.194415	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31
559	592.564477	127.0.0.1	127.0.0.1	SKYPE	67	Payload Unk: 4
560	592.564590	127.0.0.1	127.0.0.1	UDP	63	61626 → 49280 Len=31

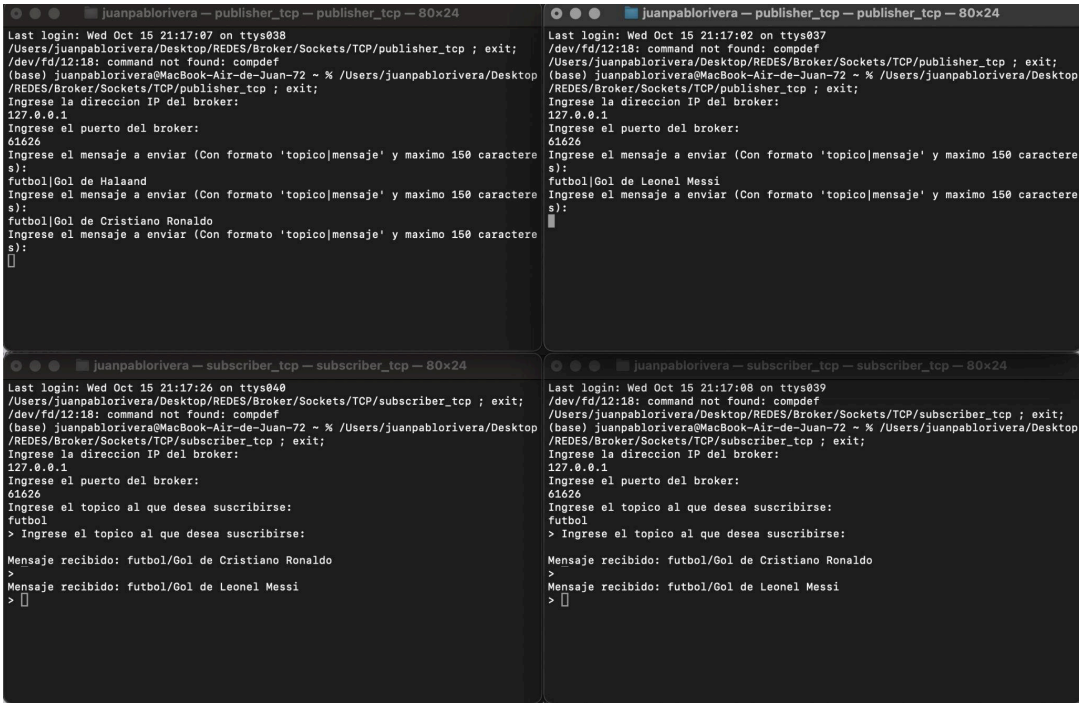
10 paquetes enviados en UDP en WireShark

5. Comparar el desempeño de TCP y UDP en una tabla considerando los siguientes criterios:

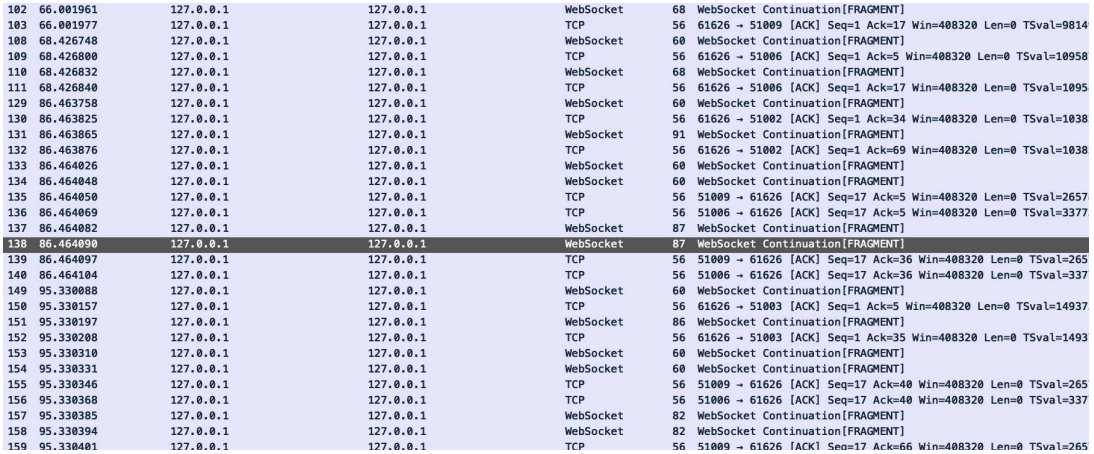
Criterio	TCP	UDP
Confiabilidad	Alta, TCP garantiza la entrega de los datos mediante confirmaciones (ACK), retransmisión y control de errores.	Baja, su funcionalidad principal es entregar paquetes a tiempo, por lo que los paquetes pueden perderse sin aviso.
Orden de entrega	Si, En TCP se asegura que los datos lleguen en el mismo orden en el que fueron enviados mediante numeración de secuencia (los seq=)	No, en UDP los datagramas pueden llegar en desorden
Pérdida de mensajes	Baja, ya que los paquetes son transmitidos en el momento en el que hay una pérdida	Alta, si hay una pérdida de un paquete, el protocolo no tiene re transmisión de datagramas debido a su función principal
Overhead de cabeceras y protocolos	Mucho mayor a la de UDP, ya que tiene que manejar muchas mas flags y características que aseguran su confiabilidad	Menor que en TCP, ya que es mucho más rápido de procesar

6. Ejecute simultáneamente al menos un broker, dos suscriptores y dos publicadores:

TCP:



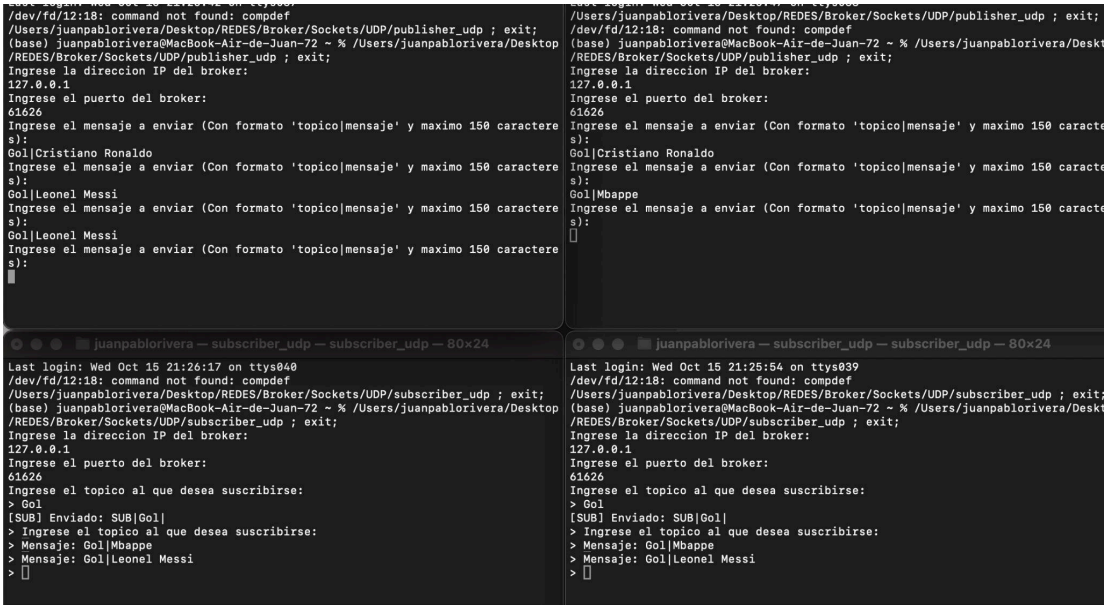
Paquetes enviados simultáneamente de 2 suscriptores, 2 publicadores y un broker en TCP



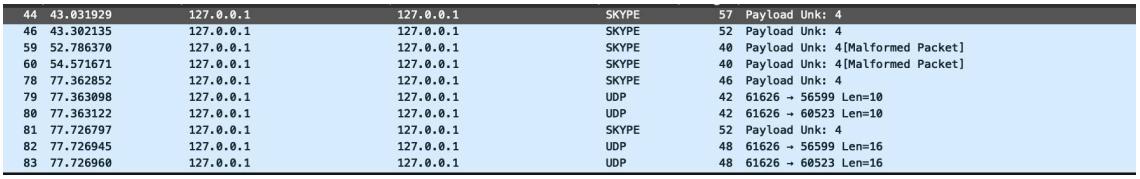
Paquetes enviados simultáneamente de 2 suscriptores, 2 publicadores y un broker en TCP en WireShark

En este caso, podemos observar que el sistema sigue funcionando, ambos mensajes están llegando de forma consistente. No se puede observar muy bien el cruce de paquetes de un mensaje al otro, principalmente porque son cadenas de texto relativamente pequeñas, pero hay que tener en cuenta que sin importar esto, el proceso es por conexión, por lo que el broker podría manejar esto en TCP de buena forma.

UDP:



Paquetes enviados simultáneamente de 2 suscriptores, 2 publicadores y un broker en UDP



44	43.031929	127.0.0.1	127.0.0.1	SKYPE	57	Payload Unk: 4
46	43.302135	127.0.0.1	127.0.0.1	SKYPE	52	Payload Unk: 4
59	52.786370	127.0.0.1	127.0.0.1	SKYPE	40	Payload Unk: 4[Malformed Packet]
60	54.571671	127.0.0.1	127.0.0.1	SKYPE	40	Payload Unk: 4[Malformed Packet]
78	77.362852	127.0.0.1	127.0.0.1	SKYPE	46	Payload Unk: 4
79	77.363098	127.0.0.1	127.0.0.1	UDP	42	61626 → 56599 Len=10
80	77.363122	127.0.0.1	127.0.0.1	UDP	42	61626 → 60523 Len=10
81	77.726797	127.0.0.1	127.0.0.1	SKYPE	52	Payload Unk: 4
82	77.726945	127.0.0.1	127.0.0.1	UDP	48	61626 → 56599 Len=16
83	77.726960	127.0.0.1	127.0.0.1	UDP	48	61626 → 60523 Len=16

Paquetes enviados simultáneamente de 2 suscriptores, 2 publicadores y un broker en UDP en WireShark

En este caso, podemos observar que el broker de igual forma funciona bien y envía los mensajes de forma correcta, al igual que en el caso de TCP, no se puede apreciar bien la velocidad debido a que son datagramas pequeños.

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

Si, los mensajes cómo se pueden ver en las pruebas llegan en orden y completos, ya que todos los paquetes van numerados con el “seq=”, y dependiendo del control implementado (n-back o fast), los paquetes si llegan en desorden o se pierden, o se guardan para acomodarlos más adelante mediante los “seq=” en caso de pérdida. La confiabilidad y el control de flujo se manejan mediante los ACK, en los que el broker en este caso, le dice al cliente que está esperando el paquete con el número de secuencia x.

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

En este caso no, no se evidencia pérdida de paquetes en UDP, sin embargo, cabe recalcar que si los hubiera, no hay un mecanismo para la retransmisión, pues UDP está diseñado para enviar rápido, no confiable

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

En este caso, la diferencia más clara entre ambos se puede ver en la cantidad de paquetes o datagramas que utilizan, ya que mientras que en el TCP por cada request se mandan 4 paquetes entre confirmaciones y payload del publisher con el broker y el broker con el subscriber, en UDP simplemente se envían 2 paquetes, uno de envio al broker y uno de envio al subscriber, ya que no realiza ningún tipo de handshake y no hay control de flujo o protocolos para pérdidas de paquetes

7. 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?

En TCP, al broker requerir una conexión independiente para cada publicador y suscriptor, con 100 partidos habrían 200 conexiones simultáneas con los publicadores y suscriptores, cada conexión individual tiene su propio control de flujo, buffer, estado y retransmisiones, el broker se recargará de una forma muy grave, lo que afectará mucho en el tiempo de entrega. Sin embargo, se garantiza confiabilidad y que los datos lleguen en orden. En UDP, por otro lado, no requiere de conexiones simultaneas sin controles de nada, por lo que los headers son muy livianos, esto causa que la latencia del broker no se aumente tanto, ya que procesar cada solicitud es mucho más liviano que en TCP, sin embargo, por la cantidad de publicadores que hay, no se podría garantizar la confiabilidad en caso de pérdida.

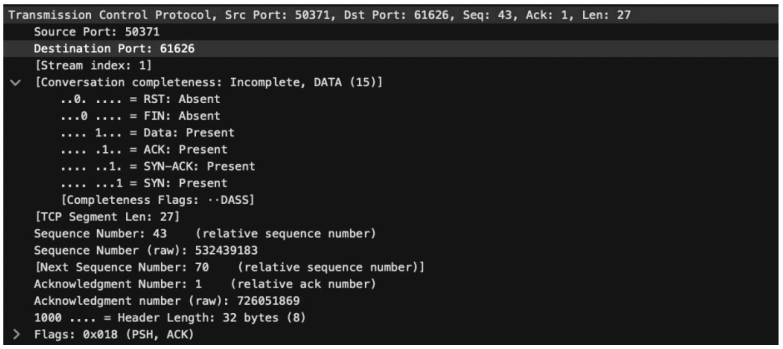
• Si un gol se envía como mensaje desde el publicador y un suscriptor no lo recibe en UDP, ¿qué implicaciones tendría para la aplicación real? ¿Por qué TCP maneja mejor este escenario?

En un escenario real, esto causaría un problema, debido a que el suscriptor no estaría enterado en tiempo real de la noticia, y si el suscriptor llegará a ser algún ente importante, esta información sería bastante clave. TCP sería mucho mejor ya que asegura que incluso si el paquete se pierde, se retransmite para que pueda llegar incluso si es con una mayor latencia.

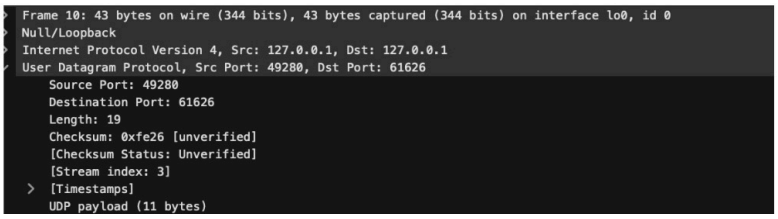
• En un escenario de seguimiento en vivo de partidos, ¿qué protocolo (TCP o UDP) resultaría más adecuado? Justifique con base en los resultados de la práctica.

Depende, para la transmisión del partido en si, seria mucho mejor UDP, puesto que se encarga a que la transmisión pueda ser casi que en tiempo real, ya que su objetivo es que los paquetes llegan en el menor tiempo posible, y si en ese caso se llegara a perder un datagrama no afectará ya que los siguientes podrían seguir con la transmisión. Por otro lado, para noticias de goles, penales, tiros de esquina, etc… TCP sin duda es mucho mejor, ya que a pesar de que los paquetes puedan demorarse un poco más, garantiza que lleguen en orden y completos, y debido a que no son paquetes tan pesados, sino cadenas de caracteres relativamente pequeñas, el tiempo no se considera un problema.

• Compare el overhead observado en las capturas Wireshark entre TCP y UDP. ¿Qué protocolo introduce más cabeceras por mensaje? ¿Cómo influye esto en la eficiencia?



Header TCP



Header UDP

Vemos claramente que mientras que TCP produce una cabecera de 32 bytes, UDP de apenas 11 bytes, esta diferencia es clave ya que para cada conexión, una cabecera más grande tarda en procesarse mucho más que una pequeña, esto a costo de no tener control de flujo o confiabilidad.

• 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?

En términos de experiencia del usuario, esto afectaría ya que el usuario no sabría cual de los marcadores es el correcto, la solución para UDP sería pasar la responsabilidad de ordenar los datagramas a la siguiente capa.

• ¿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 rendimiento empeorará ya que el broker debe enviar una copia diferente a cada suscriptor, esto genera más carga y latencia. Mientras que en UDP mediante broadcast sin abrir múltiples conexiones, se puede enviar el mismo mensaje a todos los suscriptores.

• ¿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 detectan la pérdida. Sin embargo, al existir control de flujo, en el momento de recuperar la sesión, se puede detectar desde qué paquete se ha perdido, lo que permitiría seguir con la sesión anterior como si no hubiera pasado nada. En UDP sin embargo, los mensajes se pierden sin aviso, y al los clientes no tener un control de flujo, cuando se recupere la sesión. No se podrá seguir continuamente con la sesión anterior.

• ¿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?

UDP, ya que debido a una transmisión por ejemplo de broadcast, se envía el mismo mensaje a múltiples clientes, reduciendo retardo en hacer cada envío individual y manteniendo sincronía.

• 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 maneja significativamente consume más CPU ya que mantiene conexión y las flags, procesa copias para cada suscriptor por lo que genera mucho más trabajo. En UDP, al no existir control de flujo y tener headers mucho más pequeños, responder las solicitudes es mucho más sencillo, aparte del broadcast para enviar el mismo mensaje a cada suscriptor reduciendo la latencia.

• 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.

Una combinación, por un lado, aprovecharía el broadcast que tiene UDP para enviarle las actualizaciones en tiempo real a los clientes y el control de flujo que tiene TCP para el envío de los publicadores al broker.