UNIVERSIDAD DE LOS ANDES

DEPARTAMENTO DE INGENIERIA DE SISTEMAS Y COMPUTACIÓN

LABORATORIO #3: ANÁLISIS DE CAPA DE TRANSPORTE

ISIS3204 – INFRESTRUCTURA DE COMUNICACIONES

Profesores:

Yuri Pinto

Nathalia Quiroga

Grupo 3:

Jerónimo A. Pineda Cano – 202212778

Juan Felipe Hernández – 202310576

Julián David González - 202311757

2025 – 2

Tabla de contenido

Resultados

UDP

Protocol	Lengt	Info
UDP	57	53760 → 8081 Len=15
UDP	78	8081 → 53760 Len=36
UDP	65	8081 → 53760 Len=23
UDP	67	8081 → 53760 Len=25
UDP	66	8081 → 53760 Len=24
UDP	65	8081 → 53760 Len=23
UDP	66	8081 → 53760 Len=24
UDP	64	8081 → 53760 Len=22
UDP	64	8081 → 53760 Len=22
UDP	67	8081 → 53760 Len=25
UDP	67	8081 → 53760 Len=25
UDP	64	8081 → 53760 Len=22

Imagen 1: Entradas y salidas del puerto 53760 (Susbcriber)

UDP	92	40375 → 8081 Len=50
UDP	79	40375 → 8081 Len=37
UDP	96	40375 → 8081 Len=54
UDP	80	40375 → 8081 Len=38
UDP	82	40375 → 8081 Len=40
UDP	78	40375 → 8081 Len=36
UDP	82	40375 → 8081 Len=40
UDP	81	40375 → 8081 Len=39
UDP	81	40375 → 8081 Len=39
UDP	83	40375 → 8081 Len=41

Imagen 2: Entradas y salidas del puerto 40375 (Publisher)

En las capturas se observa el flujo de comunicación entre el broker UDP (puerto 8081) y dos clientes distintos: el subscriber (puerto 53760) y el publisher (puerto 40375).

En la Imagen 1, el subscriber con puerto 53760 envía y recibe mensajes cortos al broker, lo que representa el proceso de suscripción a un tema y recepción de mensajes desde el broker.

```c
sprintf(buffer, "SUBSCRIBE:%s", topic);
sendto(sockfd, (const char *)buffer, strlen(buffer), 0,
    (const struct sockaddr *) &servaddr, sizeof(servaddr));

printf("Suscrito a '%s'. Esperando mensajes...\n", topic);
```

Imagen 3: Fragmento de código de subscriber_udp.c

Aquí el subscriber construye el mensaje "SUBSCRIBE:<tema>" y lo envía al broker usando UDP (sendto).

```c
if (topic != NULL && message != NULL) {
    printf("Publicando en el tema '%s': %s\n", topic, message);
    // Reenviar a los suscriptores
    for (int j = 0; j < sub_count; j++) {
        if (strcmp(subscriptions[j].topic, topic) == 0) {
            // Usamos sendto para enviar el mensaje a la dirección guardada del suscriptor
            sendto(sockfd, message, strlen(message), 0, (const struct sockaddr *)
            &subscriptions[j].subscriber_addr, sizeof(subscriptions[j].subscriber_addr));
        }
    }
}
```

Imagen 4: Fragmento de código de broker_udp.c

Este código muestra cómo el broker recibe un mensaje “SUBSCRIBE” del cliente y guarda su dirección (IP y puerto) en la lista de suscriptores.

Esto explica los paquetes pequeños observados en la Imagen 1, donde el subscriber intercambia mensajes cortos con el broker.

En la Imagen 2, el publisher con puerto 40375 mantiene una comunicación con el broker mediante mensajes de mayor tamaño (entre 36 y 54 bytes), lo que sugiere que está publicando información o datos en un tema específico.

```
sprintf(buffer, "PUBLISH:%s:%s", topic, message);

// Enviar el mensaje al broker usando sendto
sendto(sockfd, (const char *)buffer, strlen(buffer), 0,
    (const struct sockaddr *) &servaddr, sizeof(servaddr));
printf("Mensaje enviado.\n\n");
```

Imagen 5: Fragmento de código de publisher_udp.c

Este fragmento del publisher_udp.c construye un mensaje con el formato "PUBLISH:<tema>:<mensaje>" y lo envía al broker (127.0.0.1:8081) usando sendto().

Esto explica los paquetes más grandes enviados desde el puerto 40375 hacia el broker en la Imagen 2.

```
} else if (command != NULL && strcmp(command, "PUBLISH") == 0) {
    char *topic = strtok(NULL, ":");
    char *message = strtok(NULL, "");
    if (topic != NULL && message != NULL) {
        printf("Publicando en el tema '%s': %s\n", topic, message);
        // Reenviar a los suscriptores
        for (int j = 0; j < sub_count; j++) {
            if (strcmp(subscriptions[j].topic, topic) == 0) {
                // Usamos sendto para enviar el mensaje a la dirección guardada del suscriptor
                sendto(sockfd, message, strlen(message), 0, (const struct sockaddr *)
                &subscriptions[j].subscriber_addr, sizeof(subscriptions[j].subscriber_addr));
            }
        }
    }
}
```

Imagen 5: Fragmento de código de broker_udp.c

Este fragmento muestra cómo el broker recibe un mensaje “PUBLISH” de un cliente y reenvía el contenido a todos los suscriptores registrados en ese tema.

Esto explica los paquetes más largos que salen del puerto del publisher (40375) hacia el broker (8081).

Finalmente, el broker actúa como intermediario, recibiendo las publicaciones y reenviándolas al subscriber registrado, garantizando así la transmisión de datos entre ambos clientes sin que se comuniquen directamente.

```c
while(1) {
    n = recvfrom(sockfd, (char *)buffer, BUFFER_SIZE, MSG_WAITALL, NULL, NULL);
    buffer[n] = '\0';
    printf(">> Actualizacion: %s\n", buffer);
}
```

Imagen 5: Fragmento de código de subscriber_udp.c

El subscriber permanece escuchando mensajes del broker y los imprime en pantalla como actualizaciones.

```c
n = recvfrom(sockfd, (char *)buffer, BUFFER_SIZE, MSG_WAITALL, (struct sockaddr *) &cliaddr, &len);
buffer[n] = '\0';
```

Imagen 6: Fragmento de código de broker_udp.c

Esta línea representa la recepción de los datagramas UDP que se observan en las capturas, y es el punto donde el broker procesa los mensajes provenientes tanto del subscriber como del publisher.

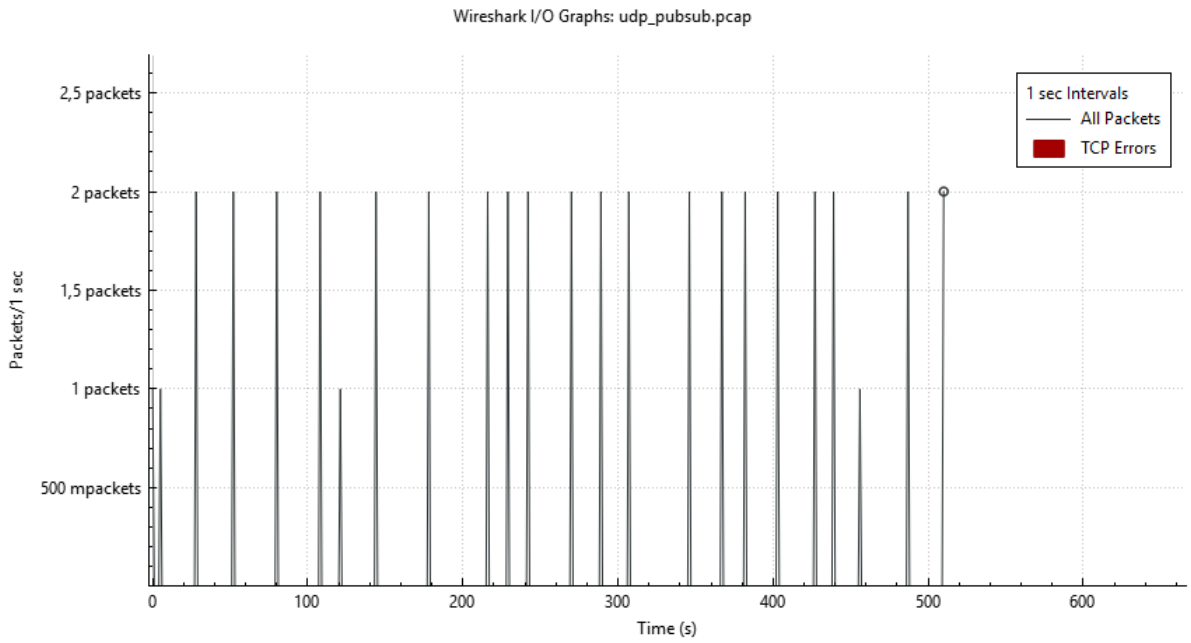


Imagen 6: Gráfico de paquetes contra tiempo

La gráfica muestra la cantidad de paquetes UDP enviados y recibidos a lo largo del tiempo durante la ejecución del sistema. Se observa que los paquetes se transmiten de forma intermitente, con picos en distintos momentos, lo que indica que la comunicación

no es continua, sino que ocurre solo cuando hay actividad entre el broker y los clientes. En general, refleja un tráfico estable y sin errores durante la prueba.

TCP

Source	Destination	Protocol	Lengt	Info
127.0.0.1	127.0.0.1	TCP	74	53872 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3610354125 TSecr=0 WS=128
127.0.0.1	127.0.0.1	TCP	74	8080 → 53872 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3610354125 TSecr=3610354125 WS=128
127.0.0.1	127.0.0.1	TCP	66	53872 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3610354125 TSecr=3610354125
127.0.0.1	127.0.0.1	TCP	74	53886 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3610356863 TSecr=0 WS=128
127.0.0.1	127.0.0.1	TCP	74	8080 → 53886 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3610356863 TSecr=3610356863 WS=128
127.0.0.1	127.0.0.1	TCP	66	53886 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3610356863 TSecr=3610356863
127.0.0.1	127.0.0.1	TCP	74	33530 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3610359069 TSecr=0 WS=128
127.0.0.1	127.0.0.1	TCP	74	8080 → 33530 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3610359069 TSecr=3610359069 WS=128
127.0.0.1	127.0.0.1	TCP	66	33530 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3610359069 TSecr=3610359069
127.0.0.1	127.0.0.1	TCP	74	33544 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3610360999 TSecr=0 WS=128
127.0.0.1	127.0.0.1	TCP	74	8080 → 33544 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3610360999 TSecr=3610360999 WS=128
127.0.0.1	127.0.0.1	TCP	66	33544 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3610360999 TSecr=3610360999

Imagen 7: Three-way handshake

En la imagen 7 se observa el three-way handshake, el proceso con el que se establece una conexión TCP confiable entre el cliente y el broker. Primero, el cliente envía un paquete SYN al broker (127.0.0.1:8080) para solicitar la conexión. El broker responde con un SYN-ACK, confirmando la recepción y aceptación de la solicitud. Finalmente, el cliente envía un ACK de respuesta, completando la negociación de la conexión. Este intercambio inicial garantiza que ambas partes estén listas para transmitir datos, a diferencia de UDP, que no requiere este paso previo.

TCP		74	53872 → 8080	[SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3610354125 TSecr=0 WS=128
TCP		74	8080 → 53872	[SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3610354125 TSecr=3610354125 WS=128
TCP		66	53872 → 8080	[ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3610354125 TSecr=3610354125
TCP		82	53872 → 8080	[PSH, ACK] Seq=1 Ack=1 Win=65536 Len=16 TSval=3610365251 TSecr=3610354125
TCP		66	8080 → 53872	[ACK] Seq=1 Ack=17 Win=65536 Len=0 TSval=3610365251 TSecr=3610365251
TCP		104	8080 → 53872	[PSH, ACK] Seq=1 Ack=17 Win=65536 Len=38 TSval=3610393416 TSecr=3610365251 [TCP PDU reassembled in 9
TCP		66	53872 → 8080	[ACK] Seq=17 Ack=39 Win=65536 Len=0 TSval=3610393416 TSecr=3610393416
TCP		87	8080 → 53872	[PSH, ACK] Seq=39 Ack=17 Win=65536 Len=21 TSval=3610463656 TSecr=3610393416 [TCP PDU reassembled in
TCP		66	53872 → 8080	[ACK] Seq=17 Ack=60 Win=65536 Len=0 TSval=3610463656 TSecr=3610463656
TCP		92	8080 → 53872	[PSH, ACK] Seq=60 Ack=17 Win=65536 Len=26 TSval=3610532578 TSecr=3610463656 [TCP PDU reassembled in
TCP		66	53872 → 8080	[ACK] Seq=17 Ack=86 Win=65536 Len=0 TSval=3610532578 TSecr=3610532578
TCP		92	8080 → 53872	[PSH, ACK] Seq=86 Ack=17 Win=65536 Len=26 TSval=3610592883 TSecr=3610532578 [TCP PDU reassembled in
TCP		66	53872 → 8080	[ACK] Seq=17 Ack=112 Win=65536 Len=0 TSval=3610592883 TSecr=3610592883

Imagen 8: Entradas y salidas del puerto 53872 (Subscriber)

TCP		74	33530 → 8080	[SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3610359069 TSecr=0 WS=128
TCP		74	8080 → 33530	[SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3610359069 TSecr=3610359069 WS=128
TCP		66	33530 → 8080	[ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3610359069 TSecr=3610359069
TCP		119	33530 → 8080	[PSH, ACK] Seq=1 Ack=1 Win=65536 Len=53 TSval=3610393415 TSecr=3610359069 [TCP PDU reassembled in 93
TCP		66	8080 → 33530	[ACK] Seq=1 Ack=54 Win=65536 Len=0 TSval=3610393415 TSecr=3610393415
TCP		127	33530 → 8080	[PSH, ACK] Seq=54 Ack=1 Win=65536 Len=61 TSval=3610418113 TSecr=3610393415 [TCP PDU reassembled in 9
TCP		66	8080 → 33530	[ACK] Seq=1 Ack=115 Win=65536 Len=0 TSval=3610418113 TSecr=3610418113
TCP		101	33530 → 8080	[PSH, ACK] Seq=115 Ack=1 Win=65536 Len=35 TSval=3610438244 TSecr=3610418113 [TCP PDU reassembled in
TCP		66	8080 → 33530	[ACK] Seq=1 Ack=150 Win=65536 Len=0 TSval=3610438244 TSecr=3610438244
TCP		106	33530 → 8080	[PSH, ACK] Seq=150 Ack=1 Win=65536 Len=40 TSval=3610553960 TSecr=3610438244 [TCP PDU reassembled in
TCP		66	8080 → 33530	[ACK] Seq=1 Ack=190 Win=65536 Len=0 TSval=3610553960 TSecr=3610553960
TCP		101	33530 → 8080	[PSH, ACK] Seq=190 Ack=1 Win=65536 Len=35 TSval=3610680840 TSecr=3610553960 [TCP PDU reassembled in
TCP		66	8080 → 33530	[ACK] Seq=1 Ack=225 Win=65536 Len=0 TSval=3610680840 TSecr=3610680840
TCP		107	33530 → 8080	[PSH, ACK] Seq=225 Ack=1 Win=65536 Len=41 TSval=3610695238 TSecr=3610680840 [TCP PDU reassembled in
TCP		66	8080 → 33530	[ACK] Seq=1 Ack=266 Win=65536 Len=0 TSval=3610695238 TSecr=3610695238
TCP		107	33530 → 8080	[PSH, ACK] Seq=266 Ack=1 Win=65536 Len=41 TSval=3610722287 TSecr=3610695238 [TCP PDU reassembled in
TCP		66	8080 → 33530	[ACK] Seq=1 Ack=307 Win=65536 Len=0 TSval=3610722287 TSecr=3610722287

Imagen 9: Entradas y salidas del puerto 53872 (Publisher)

En la imagen 8, se filtra la comunicación del cliente 53872 (subscriber), donde se observan segmentos con las banderas [PSH, ACK], lo que indica que el usuario está enviando o recibiendo datos del broker (por ejemplo, solicitudes de suscripción o mensajes recibidos). El protocolo TCP asegura la entrega ordenada y confiable de los mensajes, lo que se refleja en los números de secuencia (Seq) y acknowledgment (Ack) que acompañan cada paquete.

En la imagen 9, se observa la comunicación del cliente 33530 (publisher) con el broker. Los mensajes [PSH, ACK] que se muestran representan las publicaciones enviadas por el publisher al broker y las confirmaciones que este envía de vuelta.

En las capturas se observa el flujo de comunicación entre el broker TCP (puerto 8080) y dos clientes distintos: el subscriber (puerto 53872) y el publisher (puerto 33530). A diferencia de UDP, el protocolo TCP establece una conexión confiable entre el broker y cada cliente mediante un three-way handshake antes de transmitir datos.

```
int port = atoi(argv[1]);
listen_fd = socket(AF_INET, SOCK_STREAM, 0);
if (listen_fd<0) die("socket");
set_reuse(listen_fd);

if (bind(listen_fd,(struct sockaddr*)&addr,sizeof(addr))<0) die("bind");
if (listen(listen_fd, BACKLOG)<0) die("listen");

printf("Broker TCP escuchando en puerto %d...\n", port);
```

Imagen 10: Fragmento de código de broker_tcp.c

Aquí el broker crea un socket TCP, lo marca como reutilizable con setsockopt() y empieza a escuchar peticiones de conexión en el puerto 8080. Cuando un cliente intenta conectarse, el broker acepta la conexión mediante la llamada accept(), lo que completa el proceso de handshake mostrado en la imagen.

```
if (FD_ISSET(listen_fd,&rfds)){
    int cfd = accept(listen_fd,NULL,NULL);
    if (cfd>=0) add_client(cfd);
}
```

Imagen 11: Fragmento de código de broker_tcp.c que acepta nuevas conexiones

En la comunicación entre el subscriber y el broker. Luego del handshake, el subscriber envía pequeños paquetes con la bandera [PSH, ACK], los cuales contienen el comando SUBSCRIBE <topic> para registrarse en un tema determinado. A su vez, el broker responde con mensajes cortos confirmando la suscripción o reenviando publicaciones.

```c
if (connect(fd,(struct sockaddr*)&addr,sizeof(addr))<0) die("connect");


for(int i=3;i<argc;i++){
    char msg[512];
    snprintf(msg, sizeof(msg), "SUBSCRIBE %s\n", argv[i]);
    if (send(fd, msg, strlen(msg), 0) < 0) die("send");
}
fprintf(stderr,"Suscrito. Esperando eventos...\n");
```

Imagen 11: Fragmento de código de subscriber_tcp.c

Este código construye el mensaje de suscripción con el formato SUBSCRIBE <topic> y lo envía al broker a través del canal TCP establecido. Esto explica los paquetes pequeños observados en la Imagen 9, donde el subscriber envía comandos de suscripción y recibe confirmaciones (OK SUB <topic>).

```c
if (strncmp(line, "SUBSCRIBE ", 10)==0){
    const char *topic = line + 10;
    subscribe_topic(c, topic);
    return;
}
```

Imagen 12: Fragmento de código de broker_tcp.c

Aquí el broker TCP identifica el comando recibido, registra el tema en la lista de suscripciones del cliente, y confirma con un mensaje de respuesta. Este intercambio se corresponde con los paquetes [PSH, ACK] que se ven en la captura, representando los datos de aplicación sobre el canal TCP.

En la captura 9 se aprecia el flujo entre el publisher (puerto 33530) y el broker (puerto 8080). Los paquetes muestran segmentos TCP con longitudes variables (entre 90 y 120 bytes), lo que indica que contienen publicaciones de datos enviadas al broker mediante el comando PUBLISH.

```c
char out[8192];
snprintf(out, sizeof(out), "PUBLISH %s|%s\n", topic, line);
if (send(fd, out, strlen(out), 0) < 0){
    perror("send");
    break;
}
```

Imagen 13: Fragmento de código de publisher_tcp.c

Este fragmento muestra cómo el publisher construye el mensaje con el formato PUBLISH <topic>|<mensaje> y lo envía al broker a través de TCP. Esto explica los paquetes de mayor tamaño observados en la captura, donde el cliente publica mensajes en el tema suscrito.

```c
if (strncmp(line, "PUBLISH ", 8)==0){

    char *p = line + 8;
    char *sep = strchr(p, '|');
    if (!sep){ dprintf(c->fd, "ERR MALFORMED\n"); return; }
    *sep = '\0';
    const char *topic = p;
    const char *msg = sep+1;
    broadcast_to_topic(topic, msg, c->fd);
    dprintf(c->fd, "OK PUB %s\n", topic);
    return;
}
```

Imagen 14: Fragmento de código de broker_tcp.c

Aquí el broker TCP recibe el mensaje PUBLISH, separa el tema y el contenido usando el carácter |, y luego ejecuta broadcast_to_topic(), que envía el mensaje a todos los clientes que estén suscritos a ese tema. Este comportamiento se refleja en los paquetes [PSH, ACK] donde el broker reenvía los datos al subscriber conectado.

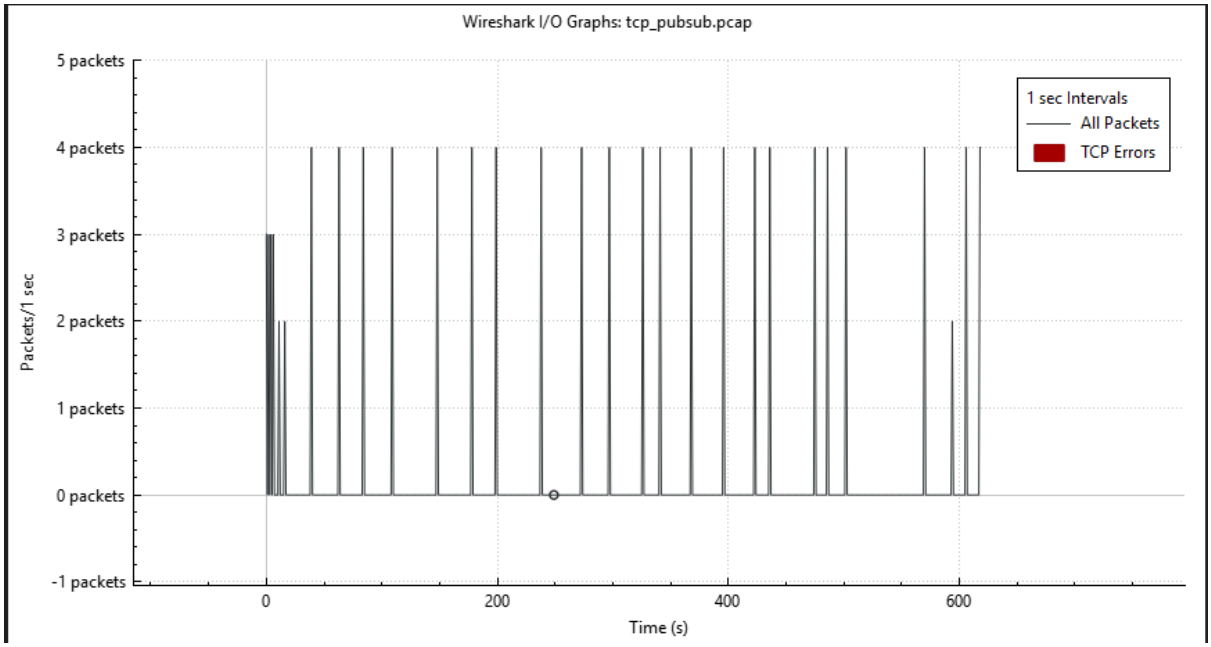


Imagen 14: Gráfico de paquetes contra tiempo

La gráfica muestra el tráfico de paquetes TCP capturado durante la ejecución del sistema *publish/subscribe*. Se observa que los paquetes se transmiten de forma intermitente, con varios picos de actividad a lo largo del tiempo, lo que indica momentos en los que se establecen conexiones y se envían mensajes entre el broker y los clientes. A diferencia de UDP, el protocolo TCP requiere establecer una conexión antes de

transmitir datos, lo que implica un proceso de tres pasos (three-way handshake) para iniciar la comunicación. Por eso, en esta captura se pueden identificar esos tres intercambios iniciales, mientras que en la gráfica anterior (UDP) no aparecen, ya que UDP no establece conexión y envía los datos directamente.

Desempeño de UDP vs TCP

	UDP	TCP
Confiabilidad	No confiable: no garantiza que los mensajes lleguen al destino.	Confiable: garantiza la entrega mediante confirmaciones (ACK) y retransmisiones.
Orden de entrega	No garantiza el orden de los paquetes.	Garantiza el orden correcto de entrega de los datos.
Pérdida de mensajes	Es posible perder mensajes sin detección ni corrección. (En las capturas no pasa pq las comunicaciones ocurren en localhost)	Detecta y reenvía los mensajes perdidos.
Overhead de cabeceras / protocolo	Bajo (cabecera de 8 bytes y sin control de conexión).	Alto (cabecera mínima de 20 bytes y mecanismos de control de conexión y flujo).

En este laboratorio, UDP resulta más adecuado para un sistema tipo broker simple cuando se busca rapidez y baja latencia, ya que no requiere establecer una conexión ni mantener control de flujo, lo que reduce el tiempo de transmisión y el uso de recursos. Por otro lado, TCP es preferible cuando se necesita confiabilidad y entrega garantizada, como en aplicaciones críticas donde no se puede tolerar la pérdida de mensajes o el desorden en la entrega.

Además, dado que el sistema se desarrolló y probó en localhost, no se presentó pérdida de paquetes, ya que la comunicación ocurre dentro del mismo equipo sin interferencias de red.

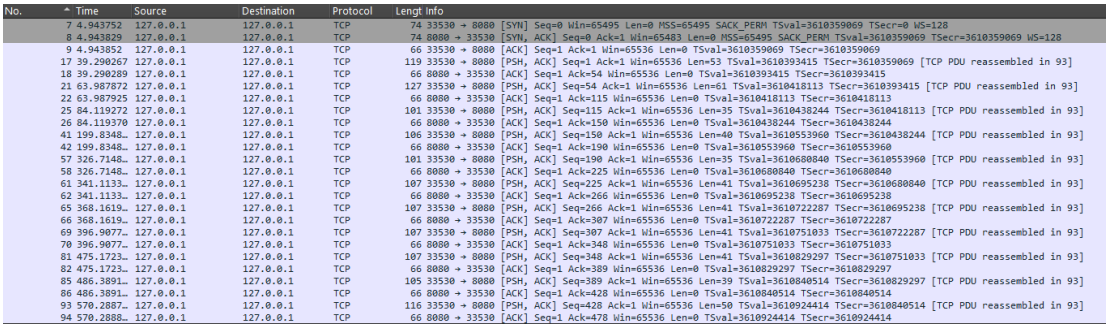
Preguntas

Pruebas y Comparación

- **En TCP: ¿los mensajes llegan completos y en orden?**
  Sí, en TCP los mensajes llegan completos y en el mismo orden en que fueron enviados. Esto se debe a que el protocolo implementa mecanismos internos de numeración de secuencia y confirmaciones de recepción (ACKs) que garantizan la entrega confiable. En las capturas de Wireshark se observa cómo cada paquete enviado desde el publisher o subscriber al broker TCP (puerto 8080) es

seguido por un paquete de confirmación (“ACK”) en sentido contrario, lo que indica que el receptor ha recibido correctamente los datos.

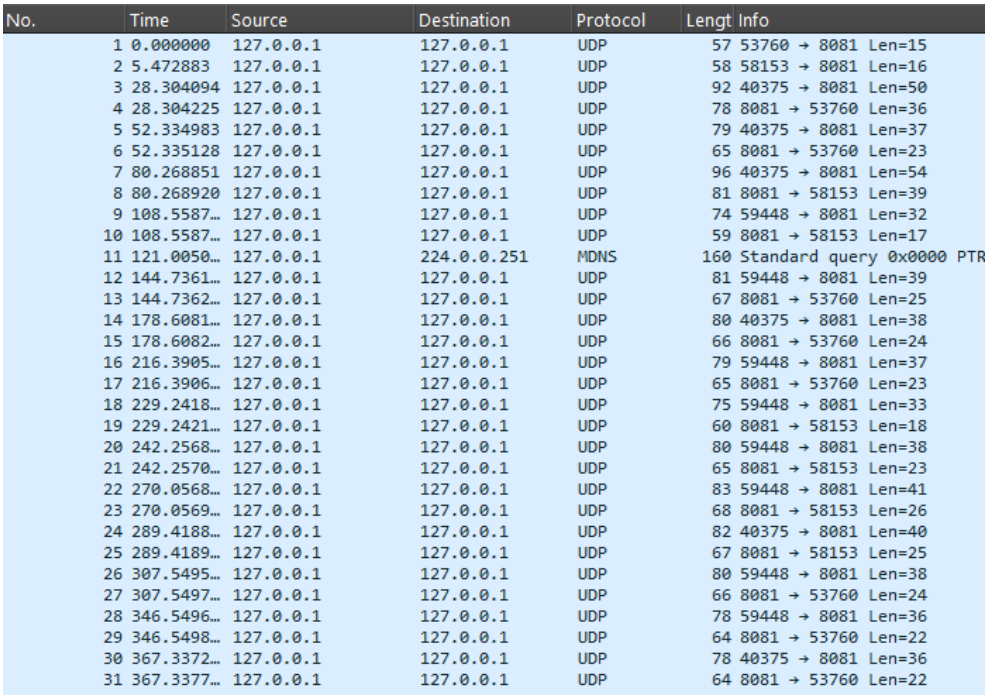
- **¿Cómo maneja TCP la confiabilidad y el control de flujo?**
  TCP maneja el control de flujo mediante la ventana deslizante (*sliding window*), que regula cuántos bytes puede enviar el emisor antes de recibir nuevas confirmaciones. Esto evita saturar al receptor y mantiene un envío equilibrado. Gracias a estos mecanismos, no se observa pérdida ni duplicación de paquetes en las capturas, y el intercambio de datos entre los clientes y el broker ocurre de forma ordenada y confiable.



No.	Time	Source	Destination	Protocol	Lengt	Info
7	4.943752	127.0.0.1	127.0.0.1	TCP	74	33530 → 8080 [SYN] Seq=0 Win=65495 Len=0 MSS=65495 SACK_PERM TSval=3610359069 TSecr=0 WS=128
8	4.943829	127.0.0.1	127.0.0.1	TCP	74	8080 → 33530 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65495 SACK_PERM TSval=3610359069 TSecr=3610359069 WS=128
9	4.943852	127.0.0.1	127.0.0.1	TCP	66	33530 → 8080 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=3610359069 TSecr=3610359069
17	39.290267	127.0.0.1	127.0.0.1	TCP	119	33530 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=53 TSval=3610393415 TSecr=3610359069 [TCP PDU reassembled in 93]
18	39.290289	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=54 Win=65536 Len=0 TSval=3610393415 TSecr=3610393415
21	63.987872	127.0.0.1	127.0.0.1	TCP	127	33530 → 8080 [PSH, ACK] Seq=54 Ack=1 Win=65536 Len=61 TSval=3610418113 TSecr=3610393415 [TCP PDU reassembled in 93]
22	63.987925	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=115 Win=65536 Len=0 TSval=3610418113 TSecr=3610418113
25	84.119272	127.0.0.1	127.0.0.1	TCP	101	33530 → 8080 [PSH, ACK] Seq=115 Ack=1 Win=65536 Len=35 TSval=3610438244 TSecr=3610418113 [TCP PDU reassembled in 93]
26	84.119370	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=150 Win=65536 Len=0 TSval=3610438244 TSecr=3610438244
41	199.8348…	127.0.0.1	127.0.0.1	TCP	106	33530 → 8080 [PSH, ACK] Seq=150 Ack=1 Win=65536 Len=40 TSval=3610553960 TSecr=3610438244 [TCP PDU reassembled in 93]
42	199.8348…	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=190 Win=65536 Len=0 TSval=3610553960 TSecr=3610553960
57	326.7148…	127.0.0.1	127.0.0.1	TCP	101	33530 → 8080 [PSH, ACK] Seq=190 Ack=1 Win=65536 Len=35 TSval=3610680840 TSecr=3610553960 [TCP PDU reassembled in 93]
58	326.7148…	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=225 Win=65536 Len=0 TSval=3610680840 TSecr=3610680840
61	341.1133…	127.0.0.1	127.0.0.1	TCP	107	33530 → 8080 [PSH, ACK] Seq=225 Ack=1 Win=65536 Len=41 TSval=3610695238 TSecr=3610680840 [TCP PDU reassembled in 93]
62	341.1133…	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=266 Win=65536 Len=0 TSval=3610695238 TSecr=3610695238
65	368.1619…	127.0.0.1	127.0.0.1	TCP	107	33530 → 8080 [PSH, ACK] Seq=266 Ack=1 Win=65536 Len=41 TSval=3610722287 TSecr=3610695238 [TCP PDU reassembled in 93]
66	368.1619…	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=307 Win=65536 Len=0 TSval=3610722287 TSecr=3610722287
69	396.9077…	127.0.0.1	127.0.0.1	TCP	107	33530 → 8080 [PSH, ACK] Seq=307 Ack=1 Win=65536 Len=41 TSval=3610751033 TSecr=3610722287 [TCP PDU reassembled in 93]
70	396.9077…	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=348 Win=65536 Len=0 TSval=3610751033 TSecr=3610751033
81	475.1723…	127.0.0.1	127.0.0.1	TCP	107	33530 → 8080 [PSH, ACK] Seq=348 Ack=1 Win=65536 Len=41 TSval=3610829297 TSecr=3610751033 [TCP PDU reassembled in 93]
82	475.1723…	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=389 Win=65536 Len=0 TSval=3610829297 TSecr=3610829297
85	486.3891…	127.0.0.1	127.0.0.1	TCP	105	33530 → 8080 [PSH, ACK] Seq=389 Ack=1 Win=65536 Len=39 TSval=3610840514 TSecr=3610829297 [TCP PDU reassembled in 93]
86	486.3891…	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=428 Win=65536 Len=0 TSval=3610840514 TSecr=3610840514
93	570.2887…	127.0.0.1	127.0.0.1	TCP	116	33530 → 8080 [PSH, ACK] Seq=428 Ack=1 Win=65536 Len=50 TSval=3610924414 TSecr=3610840514 [TCP PDU reassembled in 93]
94	570.2888…	127.0.0.1	127.0.0.1	TCP	66	8080 → 33530 [ACK] Seq=1 Ack=478 Win=65536 Len=0 TSval=3610924414 TSecr=3610924414

Imagen 15: Gráfico de paquetes en la conexión tcp

- **En UDP: ¿qué evidencias hay de pérdida o desorden en los mensajes?**
  En las capturas correspondientes al broker UDP (puerto 8081), se observa que los mensajes intercambiados entre el subscriber (puerto 53760) y el publisher (puerto 40375) son datagramas independientes, sin confirmaciones de recepción ni numeración de secuencia. Esto refleja el comportamiento no confiable de UDP.
  Sin embargo, dado que la comunicación se realizó en un entorno localhost, no se evidencian pérdidas ni desorden en los mensajes: todos los datagramas llegan correctamente. Si la prueba se ejecutara en una red real, podría observarse pérdida de paquetes o recepción fuera de orden, ya que UDP no implementa control de errores ni de flujo. En este caso, las capturas muestran un intercambio rápido y sin reenvíos, característico de la simplicidad de UDP.



No.	Time	Source	Destination	Protocol	Lengt	Info
1	0.000000	127.0.0.1	127.0.0.1	UDP	57	53760 → 8081 Len=15
2	5.472883	127.0.0.1	127.0.0.1	UDP	58	58153 → 8081 Len=16
3	28.304094	127.0.0.1	127.0.0.1	UDP	92	40375 → 8081 Len=50
4	28.304225	127.0.0.1	127.0.0.1	UDP	78	8081 → 53760 Len=36
5	52.334983	127.0.0.1	127.0.0.1	UDP	79	40375 → 8081 Len=37
6	52.335128	127.0.0.1	127.0.0.1	UDP	65	8081 → 53760 Len=23
7	80.268851	127.0.0.1	127.0.0.1	UDP	96	40375 → 8081 Len=54
8	80.268920	127.0.0.1	127.0.0.1	UDP	81	8081 → 58153 Len=39
9	108.5587…	127.0.0.1	127.0.0.1	UDP	74	59448 → 8081 Len=32
10	108.5587…	127.0.0.1	127.0.0.1	UDP	59	8081 → 58153 Len=17
11	121.0050…	127.0.0.1	224.0.0.251	MDNS	160	Standard query 0x0000 PTR
12	144.7361…	127.0.0.1	127.0.0.1	UDP	81	59448 → 8081 Len=39
13	144.7362…	127.0.0.1	127.0.0.1	UDP	67	8081 → 53760 Len=25
14	178.6081…	127.0.0.1	127.0.0.1	UDP	80	40375 → 8081 Len=38
15	178.6082…	127.0.0.1	127.0.0.1	UDP	66	8081 → 53760 Len=24
16	216.3905…	127.0.0.1	127.0.0.1	UDP	79	59448 → 8081 Len=37
17	216.3906…	127.0.0.1	127.0.0.1	UDP	65	8081 → 53760 Len=23
18	229.2418…	127.0.0.1	127.0.0.1	UDP	75	59448 → 8081 Len=33
19	229.2421…	127.0.0.1	127.0.0.1	UDP	60	8081 → 58153 Len=18
20	242.2568…	127.0.0.1	127.0.0.1	UDP	80	59448 → 8081 Len=38
21	242.2570…	127.0.0.1	127.0.0.1	UDP	65	8081 → 58153 Len=23
22	270.0568…	127.0.0.1	127.0.0.1	UDP	83	59448 → 8081 Len=41
23	270.0569…	127.0.0.1	127.0.0.1	UDP	68	8081 → 58153 Len=26
24	289.4188…	127.0.0.1	127.0.0.1	UDP	82	40375 → 8081 Len=40
25	289.4189…	127.0.0.1	127.0.0.1	UDP	67	8081 → 58153 Len=25
26	307.5495…	127.0.0.1	127.0.0.1	UDP	80	59448 → 8081 Len=38
27	307.5497…	127.0.0.1	127.0.0.1	UDP	66	8081 → 53760 Len=24
28	346.5496…	127.0.0.1	127.0.0.1	UDP	78	59448 → 8081 Len=36
29	346.5498…	127.0.0.1	127.0.0.1	UDP	64	8081 → 53760 Len=22
30	367.3372…	127.0.0.1	127.0.0.1	UDP	78	40375 → 8081 Len=36
31	367.3377…	127.0.0.1	127.0.0.1	UDP	64	8081 → 53760 Len=22

Imagen 16: Gráfico de paquetes en la conexión udp

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

  La principal diferencia radica en que TCP es orientado a conexión, mientras que UDP es un protocolo sin conexión. En las capturas de Wireshark del broker TCP, se observa el proceso de establecimiento de conexión mediante el handshake de tres vías (SYN, SYN-ACK, ACK) antes de comenzar la transmisión de datos. Esto garantiza que ambos extremos estén listos para comunicarse y que se reserve un canal confiable para el intercambio. Por el contrario, en las capturas del broker UDP no hay establecimiento de conexión: los clientes simplemente envían datagramas al broker sin negociación previa ni cierre formal de sesión. Esto permite una transmisión más rápida, pero sin las garantías de entrega ni orden que ofrece TCP.

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?**

  Respuesta: Con cien publicadores simultáneos, el broker enfrentaría una carga significativamente mayor en términos de procesamiento y manejo de conexiones. Bajo TCP, el desempeño se vería más afectado debido a la necesidad de mantener un canal confiable con cada cliente, gestionar ventanas de flujo y retransmisiones, lo que incrementa el uso de CPU y memoria. En cambio, UDP, al no requerir establecimiento ni control de conexión, podría manejar un mayor volumen de mensajes con menor latencia, aunque con el riesgo de pérdida o desorden de paquetes. Por tanto, UDP escalaría mejor en cantidad de flujos, mientras que TCP ofrecería mayor confiabilidad a costa de desempeño.

- **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?**

  Respuesta: La pérdida de un mensaje UDP implica que el suscriptor no recibiría el evento del gol, afectando la integridad de la información mostrada y generando inconsistencia en la experiencia de usuario. Esto se debe a que UDP no implementa confirmaciones ni retransmisiones. TCP, en cambio, garantiza la entrega mediante ACKs y reenvío de paquetes perdidos, por lo que el mensaje siempre llegaría al destino, aunque con un ligero retardo adicional.

- **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.**

  Respuesta: De acuerdo con los resultados, UDP sería más adecuado para un sistema de seguimiento en vivo donde la prioridad es la baja latencia y la actualización rápida. En la práctica, se observó que UDP transmite mensajes sin establecer conexión ni realizar verificaciones de entrega, reduciendo el retardo. Sin embargo, si la precisión y confiabilidad son críticas (por ejemplo, en aplicaciones oficiales o financieras), TCP sería preferible por garantizar entrega y orden, aunque a costa de velocidad.

- **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?**

  Respuesta: Las capturas de Wireshark muestran que TCP introduce un mayor overhead debido a sus cabeceras más extensas (mínimo 20 bytes) y al intercambio adicional de control (SYN, ACK). UDP, con cabeceras de solo 8 bytes, resulta más eficiente al requerir menos procesamiento y menor consumo de ancho de banda por mensaje. Este menor overhead contribuye a una mayor velocidad y eficiencia en la transmisión de datos.

- **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?**
  Respuesta: El desorden en la llegada de mensajes podría generar confusión en el usuario, mostrando resultados incoherentes o cronológicamente incorrectos. Este comportamiento es inherente a UDP, ya que no garantiza el orden de entrega. A nivel de aplicación, puede mitigarse incorporando numeración de secuencia o marcas de tiempo en los mensajes, de modo que el cliente reordene los eventos antes de mostrarlos.
- **¿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?**
  Respuesta: Con más suscriptores, el broker TCP debe mantener múltiples conexiones activas, aumentando el uso de recursos y el retardo total. En UDP, el broker solo reenvía datagramas a varias direcciones sin mantener estado de conexión, lo que permite una mayor escalabilidad. En las pruebas, UDP mostró menor carga en CPU y tiempos de envío más cortos al manejar múltiples suscriptores.
- **¿Qué sucede si el broker se detiene inesperadamente? ¿Qué diferencias hay entre TCP y UDP en la capacidad de recuperación de la sesión?**
  Respuesta: En TCP, la interrupción del broker provoca el cierre de las conexiones activas; al reiniciarse, los clientes deben restablecer el canal mediante un nuevo handshake. En UDP, al no existir una sesión persistente, los clientes pueden seguir enviando mensajes al puerto del broker una vez que este vuelve a estar disponible, sin necesidad de renegociación. Por tanto, la recuperación es más inmediata con UDP, aunque sin garantías de entrega durante la caída.
- **¿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?**
  Respuesta: La sincronización exacta depende más del diseño de la aplicación que del protocolo. No obstante, UDP facilita una distribución más simultánea al transmitir datagramas multicast o broadcast sin establecer múltiples sesiones. TCP, al enviar secuencialmente por conexión, introduce pequeñas diferencias de tiempo entre clientes. Por tanto, para difusión simultánea, UDP es más apropiado.
- **Analice el uso de CPU y memoria en el broker cuando maneja múltiples conexiones TCP frente al manejo de datagramas UDP. ¿Qué diferencias encontró?**
  Respuesta: El uso de CPU y memoria en el broker TCP es mayor debido al mantenimiento de estructuras de control por conexión (buffers, estados,

confirmaciones). En UDP, el procesamiento es más liviano, pues el broker solo recibe y reenvía datagramas sin almacenar contexto de sesión. En las pruebas, TCP mostró una carga superior en memoria y tiempos de respuesta ligeramente más altos al aumentar el número de clientes.

- **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.**
  Respuesta: La mejor opción sería una combinación híbrida. UDP serviría para la distribución masiva de actualizaciones rápidas en tiempo real (como eventos o goles), aprovechando su baja latencia y capacidad de difusión. TCP se emplearía para funciones críticas que requieren confiabilidad, como confirmaciones o estadísticas oficiales. Este enfoque equilibra rendimiento, escalabilidad y precisión, replicando los resultados observados en las pruebas del laboratorio.

Enlace código

JFH000/lab-3-redes