

UNIVERSIDAD DE LOS ANDES
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y
COMPUTACIÓN



LABORATORIO #3: ANÁLISIS DE CAPA DE TRANSPORTE Y SOCKETS

ISIS 3204 - INFRAESTRUCTURA DE COMUNICACIONES

Nathalia Quiroga

Grupo 7

David Quiroga 202310820

Nicolas Gonzalez 202310041

Samuel Rodríguez Torres 202310140

2025-20

Contenido

Actividad 5.1: Configuración Inicial	3
a. Repositorio GitHub	3
Actividad 5.2: Pruebas y Comparación	3
a. Captura de paquetes UDP y TCP.....	3
b. Comparación de rendimiento UDP y TCP.....	5
Actividad 6: Preguntas de análisis	7
Conclusión	12

2. Actividad 5.1: Configuración inicial

a. Repositorio GitHub

Siguiendo las instrucciones de la guía, se crearon los siguientes archivos:

- broker_tcp.c, publisher_tcp.c, subscriber_tcp.c
- broker_udp.c, publisher_udp.c, subscriber_udp.c

Cada uno de estos archivos fueron creados en C y publicados en el siguiente repositorio

<https://github.com/LabsRedes/Laboratorio-3>, luego fueron clonados dentro de cada máquina virtual y dentro de esta se compilaron con **gcc** para poder ejecutarse como se puede ver en la siguiente imagen.

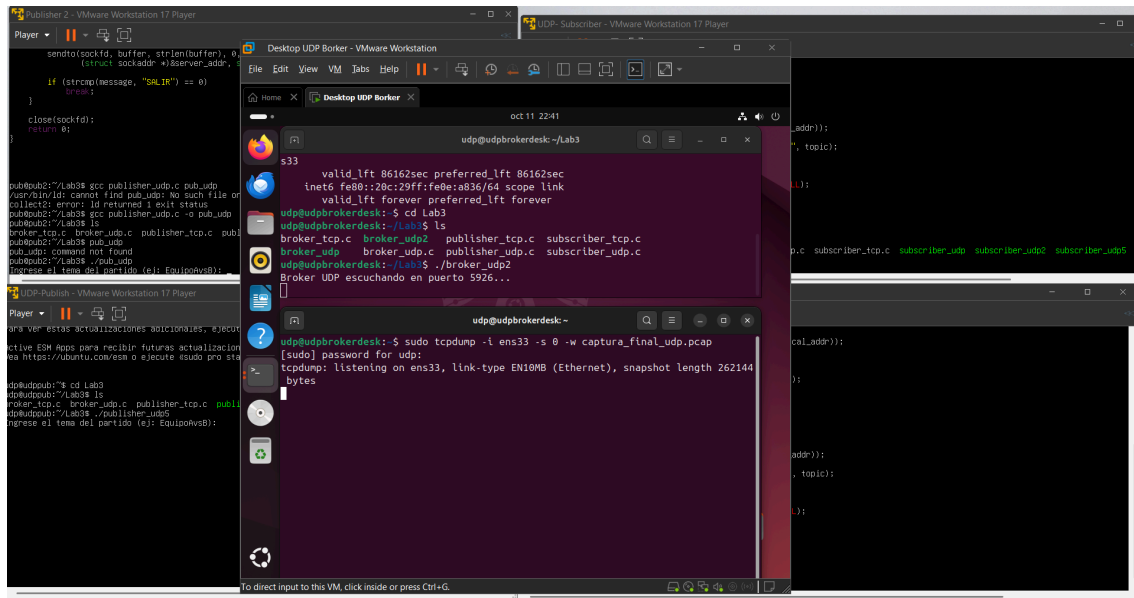


Imagen 1: Realizando el envío de mensajes de pub/sub para udp.

La imagen 1 muestra la configuración que se siguió para realizar las pruebas, 2 máquinas publicadores y 2 máquinas suscriptoras conectadas a un broker que se hizo con una imagen de Ubuntu con interfaz gráfica (Desktop) para poder abrir 2 terminales al mismo tiempo, una para ejecutar el archivo compilado de broker_udp.c y otra para capturar el tráfico generado.

Los detalles de la implementación de cada archivo se pueden encontrar en el repositorio en el archivo [ReadMe.md](#) y dentro de cada archivo con su respectiva documentación.

3. Actividad 5.2: Pruebas y Comparación

a. Captura de paquetes UDP y TCP

Para las pruebas y captura de paquetes, se ejecutó un broker, dos suscriptores y dos publicadores y para cada escenario (UDP y TCP), se mandaron 10 mensajes por publicador, cada uno en un tema diferente, obteniendo el siguiente tráfico:

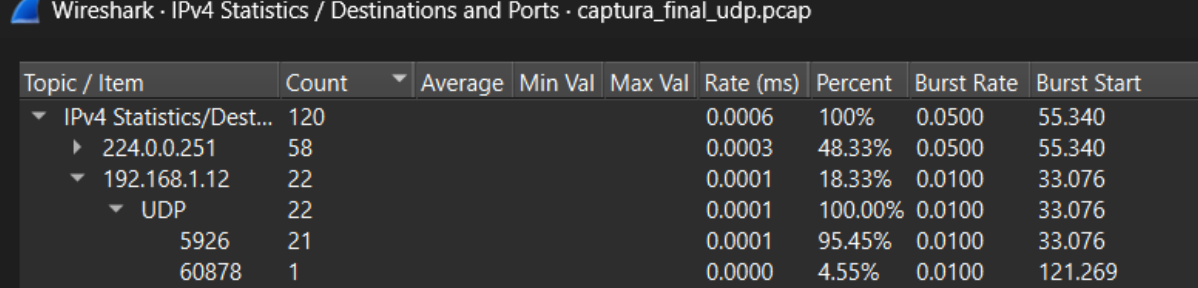
tcp.port == 5927									
No.	Time	Source	Destination	Protocol	Length	Info			
6	6.699738	192.168.1.16	192.168.1.12	TCP	64	59210 → 5927 [PSH, ACK] Seq=1 Acks=1 Win=502 Len=0 TSval=1476287515 TSecr=4207567715			
7	6.699914	192.168.1.12	192.168.1.16	TCP	66	5927 → 59210 [ACK] Seq=1 Acks=19 Win=509 Len=0 TSval=4207676337 TSecr=1476287515			
8	6.701547	192.168.1.12	192.168.1.16	TCP	88	5927 → 59210 [PSH, ACK] Seq=1 Acks=19 Win=509 Len=2 TSval=4207676338 TSecr=1476287515			
9	6.702373	192.168.1.16	192.168.1.12	TCP	66	59210 → 5927 [ACK] Seq=19 Acks=23 Win=502 Len=0 TSval=1476287516 TSecr=4207676338			
12	10.808234	192.168.1.13	192.168.1.12	TCP	83	34036 → 5927 [PSH, ACK] Seq=1 Acks=1 Win=502 Len=17 TSval=1761863243 TSecr=1451437641			
13	10.808394	192.168.1.12	192.168.1.13	TCP	66	5927 → 34036 [ACK] Seq=1 Acks=18 Win=509 Len=0 TSval=1451492118 TSecr=1761863243			
14	10.808392	192.168.1.12	192.168.1.13	TCP	87	5927 → 34036 [PSH, ACK] Seq=1 Acks=18 Win=509 Len=1 TSval=1451492118 TSecr=1761863243			
15	10.809596	192.168.1.13	192.168.1.12	TCP	66	34036 → 5927 [ACK] Seq=18 Acks=22 Win=502 Len=0 TSval=1761863256 TSecr=1451492118			
25	15.603977	192.168.1.15	192.168.1.12	TCP	100	41432 → 5927 [PSH, ACK] Seq=1 Acks=1 Win=502 Len=34 TSval=4549420776 TSecr=267155320			
26	15.604042	192.168.1.12	192.168.1.15	TCP	66	5927 → 41432 [ACK] Seq=1 Acks=15 Win=509 Len=0 TSval=1671120912 TSecr=4549420776			
27	15.604378	192.168.1.12	192.168.1.13	TCP	85	5927 → 34036 [PSH, ACK] Seq=22 Acks=18 Win=509 Len=19 TSval=1451509094 TSecr=1761863256			
28	15.604061	192.168.1.13	192.168.1.12	TCP	66	34036 → 5927 [ACK] Seq=18 Acks=41 Win=502 Len=0 TSval=1761872053 TSecr=1451509094			
39	24.852580	192.168.1.15	192.168.1.12	TCP	104	41432 → 5927 [PSH, ACK] Seq=35 Acks=1 Win=502 Len=38 TSval=4549420832 TSecr=2671120912			
40	24.852658	192.168.1.12	192.168.1.15	TCP	66	5927 → 41432 [ACK] Seq=1 Acks=73 Win=509 Len=0 TSval=1671120912 TSecr=4549420832			
41	24.852634	192.168.1.12	192.168.1.13	TCP	89	5927 → 34036 [PSH, ACK] Seq=41 Acks=18 Win=509 Len=43 TSval=1451506023 TSecr=1761872053			
42	24.854480	192.168.1.13	192.168.1.12	TCP	66	34036 → 5927 [ACK] Seq=18 Acks=64 Win=502 Len=0 TSval=1761877301 TSecr=1451506023			
47	38.238080	192.168.1.15	192.168.1.12	TCP	112	41432 → 5927 [PSH, ACK] Seq=73 Acks=1 Win=502 Len=46 TSval=4549393410 TSecr=2671120912			
48	38.238965	192.168.1.12	192.168.1.15	TCP	66	5927 → 41432 [ACK] Seq=1 Acks=119 Win=509 Len=0 TSval=1671120912 TSecr=4549393410			
49	38.231177	192.168.1.12	192.168.1.13	TCP	97	5927 → 34036 [PSH, ACK] Seq=64 Acks=18 Win=509 Len=31 TSval=1451519611 TSecr=1761877301			
50	38.232089	192.168.1.13	192.168.1.12	TCP	66	34036 → 5927 [ACK] Seq=18 Acks=95 Win=502 Len=0 TSval=1761890679 TSecr=1451519611			
51	46.012118	192.168.1.15	192.168.1.12	TCP	103	41432 → 5927 [PSH, ACK] Seq=119 Acks=1 Win=502 Len=47 TSval=4549471591 TSecr=2671120912			
52	46.012200	192.168.1.12	192.168.1.15	TCP	66	5927 → 41432 [ACK] Seq=1 Acks=156 Win=509 Len=0 TSval=1671155120 TSecr=4549471591			
53	46.012418	192.168.1.12	192.168.1.13	TCP	88	5927 → 34036 [PSH, ACK] Seq=95 Acks=18 Win=509 Len=22 TSval=1451515792 TSecr=1761890679			
54	46.014279	192.168.1.13	192.168.1.12	TCP	66	34036 → 5927 [ACK] Seq=18 Acks=117 Win=502 Len=0 TSval=1761890661 TSecr=1451515792			
58	52.827333	192.168.1.15	192.168.1.12	TCP	129	41432 → 5927 [PSH, ACK] Seq=156 Acks=1 Win=502 Len=63 TSval=4549494007 TSecr=2671155320			
59	52.827236	192.168.1.12	192.168.1.15	TCP	66	5927 → 41432 [ACK] Seq=1 Acks=119 Win=509 Len=0 TSval=1761162135 TSecr=4549494007			
60	52.827739	192.168.1.12	192.168.1.13	TCP	114	5927 → 34036 [PSH, ACK] Seq=117 Acks=18 Win=509 Len=48 TSval=1451534200 TSecr=1761890661			
61	52.828555	192.168.1.13	192.168.1.12	TCP	66	34036 → 5927 [ACK] Seq=18 Acks=165 Win=502 Len=0 TSval=1761185276 TSecr=1451534200			
72	75.232433	192.168.1.11	192.168.1.12	TCP	124	49202 → 5927 [PSH, ACK] Seq=1 Acks=1 Win=502 Len=58 TSval=2308287215 TSecr=1647640445			
73	75.232747	192.168.1.12	192.168.1.11	TCP	66	5927 → 49202 [ACK] Seq=1 Acks=59 Win=509 Len=0 TSval=1647647393 TSecr=2308287215			
74	75.233802	192.168.1.12	192.168.1.16	TCP	100	5927 → 59210 [PSH, ACK] Seq=19 Acks=19 Win=509 Len=2 TSval=1427774570 TSecr=1476287518			
75	75.234311	192.168.1.12	192.168.1.12	TCP	66	59210 → 5927 [ACK] Seq=19 Acks=45 Win=502 Len=0 TSval=1476287518 TSecr=142774570			
76	78.754872	192.168.1.11	192.168.1.12	TCP	98	49202 → 5927 [PSH, ACK] Seq=59 Acks=1 Win=502 Len=32 TSval=2308290738 TSecr=1647837993			
77	78.755142	192.168.1.12	192.168.1.11	TCP	64	5927 → 49202 [ACK] Seq=1 Acks=91 Win=509 Len=0 TSval=1647841505 TSecr=2308290738			
78	78.755643	192.168.1.12	192.168.1.16	TCP	82	5927 → 59210 [PSH, ACK] Seq=19 Acks=13 Win=509 Len=16 TSval=1427774932 TSecr=1476375958			
79	78.756210	192.168.1.16	192.168.1.12	TCP	66	59210 → 5927 [ACK] Seq=19 Acks=81 Win=502 Len=0 TSval=1476306872 TSecr=142774932			
84	88.365947	192.168.1.11	192.168.1.12	TCP	124	49202 → 5927 [PSH, ACK] Seq=91 Acks=1 Win=502 Len=58 TSval=2308300849 TSecr=1647841505			
85	88.366210	192.168.1.12	192.168.1.11	TCP	66	5927 → 49202 [ACK] Seq=1 Acks=149 Win=509 Len=0 TSval=1647851115 TSecr=2308300849			
86	88.365931	192.168.1.12	192.168.1.16	TCP	100	5927 → 59210 [PSH, ACK] Seq=91 Acks=19 Win=509 Len=42 TSval=1427775902 TSecr=1476306872			
87	88.366196	192.168.1.12	192.168.1.12	TCP	66	59210 → 5927 [ACK] Seq=19 Acks=123 Win=502 Len=0 TSval=1476375182 TSecr=142775902			
95	96.237497	192.168.1.11	192.168.1.12	TCP	120	49202 → 5927 [PSH, ACK] Seq=149 Acks=1 Win=502 Len=54 TSval=2308380221 TSecr=1647851115			
96	96.237571	192.168.1.12	192.168.1.11	TCP	66	5927 → 49202 [ACK] Seq=1 Acks=283 Win=509 Len=0 TSval=1647850988 TSecr=2308380221			
97	96.237712	192.168.1.16	192.168.1.12	TCP	104	5927 → 59210 [PSH, ACK] Seq=119 Acks=19 Win=509 Len=38 TSval=1427776075 TSecr=1476370182			
98	96.238411	192.168.1.16	192.168.1.12	TCP	66	59210 → 5927 [ACK] Seq=19 Acks=161 Win=502 Len=0 TSval=1476378954 TSecr=142776075			
105	97.551342	192.168.1.11	192.168.1.12	TCP	84	49202 → 5927 [PSH, ACK] Seq=203 Acks=1 Win=502 Len=58 TSval=2308389535 TSecr=1647850988			
106	97.551408	192.168.1.12	192.168.1.11	TCP	66	5927 → 49202 [ACK] Seq=1 Acks=211 Win=509 Len=0 TSval=1647850858 TSecr=2308389535			
107	97.551370	192.168.1.12	192.168.1.16	TCP	66	5927 → 59210 [PSH, ACK] Seq=161 Acks=19 Win=509 Len=2 TSval=1427776819 TSecr=1476378954			
108	97.554678	192.168.1.16	192.168.1.12	TCP	66	59210 → 5927 [ACK] Seq=19 Acks=163 Win=502 Len=0 TSval=1476379368 TSecr=142776819			
109	98.524069	192.168.1.11	192.168.1.12	TCP	66	49202 → 5927 [PSH, ACK] Seq=211 Acks=1 Win=502 Len=0 TSval=2308318518 TSecr=1647850858			
110	98.534112	192.168.1.12	192.168.1.11	TCP	66	5927 → 49202 [ACK] Seq=1 Acks=241 Win=509 Len=0 TSval=1647861284 TSecr=2308318518			
111	98.534948	192.168.1.12	192.168.1.16	TCP	70	5927 → 59210 [PSH, ACK] Seq=163 Acks=19 Win=509 Len=4 TSval=1427776971 TSecr=1476379368			
112	98.534948	192.168.1.16	192.168.1.12	TCP	66	59210 → 5927 [ACK] Seq=19 Acks=187 Win=502 Len=0 TSval=1476380351 TSecr=142776971			
113	99.606748	192.168.1.11	192.168.1.12	TCP	86	49202 → 5927 [PSH, ACK] Seq=241 Acks=1 Win=502 Len=20 TSval=2308311591 TSecr=1647861284			
114	99.606817	192.168.1.12	192.168.1.11	TCP	66	5927 → 49202 [ACK] Seq=1 Acks=261 Win=509 Len=0 TSval=1647862157 TSecr=2308311591			
115	99.607022	192.168.1.12	192.168.1.16	TCP	70	5927 → 59210 [PSH, ACK] Seq=167 Acks=19 Win=509 Len=4 TSval=1427778244 TSecr=1476380351			
116	99.610867	192.168.1.16	192.168.1.12	TCP	66	59210 → 5927 [ACK] Seq=19 Acks=171 Win=502 Len=0 TSval=1476381427 TSecr=1427778244			
117	102.223498	192.168.1.11	192.168.1.12	TCP	98	49202 → 5927 [PSH, ACK] Seq=261 Acks=1 Win=502 Len=32 TSval=2308314208 TSecr=1647862157			
118	102.223764	192.168.1.12	192.168.1.11	TCP	66	5927 → 49202 [ACK] Seq=1 Acks=293 Win=509 Len=0 TSval=1647864974 TSecr=2308314208			

Imagen 2: Tráfico TCP en las pruebas pub/sub.

udp.port == 5926									
No.	Time	Source	Destination	Protocol	Length	Info			
26	33.075665	192.168.1.16	192.168.1.12	UDP	60	50785 → 5926 Len=17			
29	37.046323	192.168.1.13	192.168.1.12	UDP	60	49905 → 5926 Len=14			
32	43.804355	192.168.1.11	192.168.1.12	UDP	71	59017 → 5926 Len=19			
33	43.807265	192.168.1.12	192.168.1.16	UDP	55	5926 → 50785 Len=13			
38	50.424436	192.168.1.15	192.168.1.12	UDP	67	54985 → 5926 Len=25			
39	50.424767	192.168.1.12	192.168.1.13	UDP	54	5926 → 49905 Len=12			
43	53.208784	192.168.1.15	192.168.1.12	UDP	62	54985 → 5926 Len=20			
44	53.208030	192.168.1.12	192.168.1.13	UDP	49	5926 → 49905 Len=7			
47	56.473420	192.168.1.15	192.168.1.12	UDP	68	54985 → 5926 Len=26			
68	56.473574	192.168.1.12	192.168.1.13	UDP	55	5926 → 49905 Len=13			
77	59.580741	192.168.1.15	192.168.1.12	UDP	67	54985 → 5926 Len=25			
78	59.580929	192.168.1.12	192.168.1.13	UDP	54	5926 → 49905 Len=12			
79	64.533293	192.168.1.15	192.168.1.12	UDP	77	54985 → 5926 Len=35			
80	64.533451	192.168.1.12	192.168.1.13	UDP	64	5926 → 49905 Len=22			
106	78.273796	192.168.1.12	192.168.1.12	UDP	100	59017 → 5926 Len=50			
107	78.271962	192.168.1.12	192.168.1.16	UDP	92	5926 → 50785 Len=50			
125	85.732298	192.168.1.11	192.168.1.12	UDP	85	59017 → 5926 Len=43			

b. Comparación de rendimiento UDP y TCP

Teniendo los archivos tcp_pubsub.pcap y udp_pubsub.pcap (disponibles en el repositorio de GitHub), se abrieron en Wireshark para poder aplicar el filtro del puerto definido en el broker y en los archivos del publisher y subscriber y generar las siguientes imágenes y gráficas:

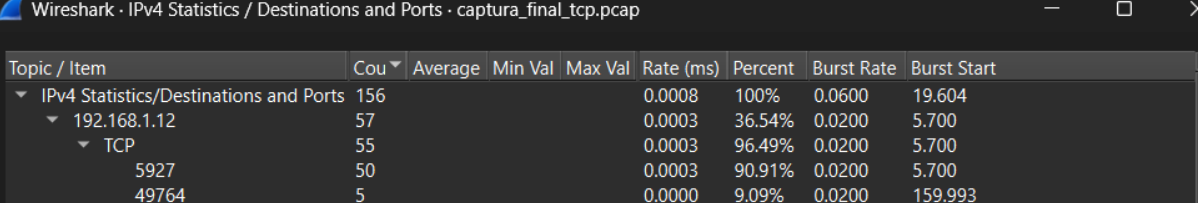


Wireshark · IPv4 Statistics / Destinations and Ports · captura_final_udp.pcap

Topic / Item	Count	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
▼ IPv4 Statistics/Dest...	120				0.0006	100%	0.0500	55.340
▶ 224.0.0.251	58				0.0003	48.33%	0.0500	55.340
▼ 192.168.1.12	22				0.0001	18.33%	0.0100	33.076
▼ UDP	22				0.0001	100.00%	0.0100	33.076
5926	21				0.0001	95.45%	0.0100	33.076
60878	1				0.0000	4.55%	0.0100	121.269

Imagen 4: Estadísticas del tráfico UDP.

En la imagen 4 se observa el tráfico correspondiente al protocolo UDP, con un total de 22 paquetes capturados. El valor promedio y la tasa de ráfaga son bajos, lo que refleja un envío rápido y sin confirmación, característico de un protocolo no orientado a conexión, donde los paquetes viajan de forma unidireccional y pueden perderse sin retransmisión.



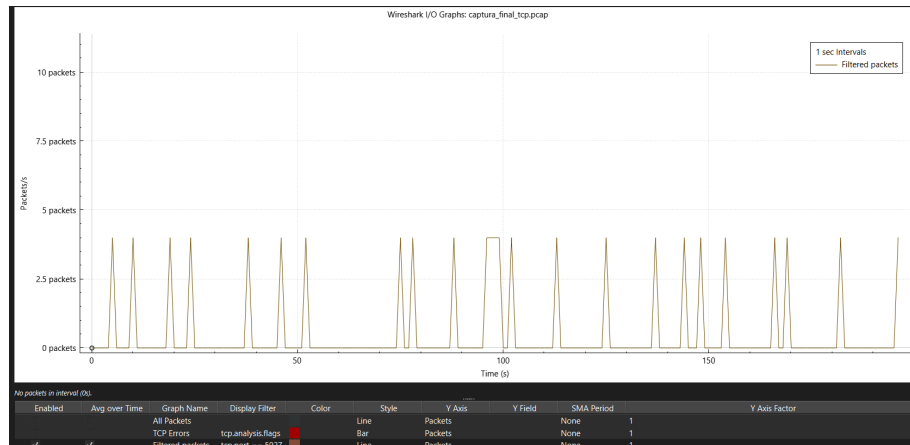
Wireshark · IPv4 Statistics / Destinations and Ports · captura_final_tcp.pcap

Topic / Item	Cou	Average	Min Val	Max Val	Rate (ms)	Percent	Burst Rate	Burst Start
▼ IPv4 Statistics/Destinations and Ports	156				0.0008	100%	0.0600	19.604
▼ 192.168.1.12	57				0.0003	36.54%	0.0200	5.700
▼ TCP	55				0.0003	96.49%	0.0200	5.700
5927	50				0.0003	90.91%	0.0200	5.700
49764	5				0.0000	9.09%	0.0200	159.993

Imagen 5: Estadísticas del tráfico TCP.

En la imagen 5 se muestra el tráfico TCP, que registra 50 paquetes totales. Los valores de tasa promedio y ráfaga son ligeramente superiores a los de UDP, lo que indica una mayor carga de control debido al establecimiento de conexión, confirmaciones (ACKs) y mecanismos de fiabilidad. Esto demuestra un tráfico más constante y estructurado, propio de un protocolo orientado a conexión que garantiza el orden y la entrega de los mensajes.

Comparando ambos resultados, se evidencia que UDP genera menos tráfico y mayor velocidad, pero sin mecanismos de garantía ni control de flujo, mientras que TCP produce más paquetes y mayor estabilidad, lo que se puede evidenciar en las siguientes gráficas:



Gráfica 1: Gráfica I/O del tráfico TCP. Time vs Packets/s



Gráfica 2: Gráfica I/O del tráfico UDP. Time vs Packets/s

En las gráficas 1 y 2 se observa claramente la diferencia en el comportamiento de TCP y UDP durante la transmisión. En la captura de UDP, la cantidad de paquetes es menor y las ráfagas aparecen de forma más irregular, lo que refleja su naturaleza unidireccional y sin conexión: los datos se envían sin necesidad de confirmaciones ni control de flujo. En cambio, la gráfica de TCP muestra una cantidad mayor de paquetes distribuidos de manera más constante y simétrica, evidenciando el intercambio bidireccional entre emisor y receptor. Esto ocurre porque TCP requiere enviar confirmaciones (ACK) por cada bloque de datos recibido, además de manejar retransmisiones y control de congestión, lo que aumenta el número total de paquetes y genera un patrón más ordenado y predecible en el tiempo.

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

Los mensajes llegan completos y en orden, ya que el protocolo garantiza la entrega secuencial mediante el uso de números de secuencia y confirmaciones (ACK). Si un paquete se pierde, TCP lo retransmite automáticamente, asegurando que todos los datos lleguen correctamente al destino. De acuerdo a la teoría, para aprovechar la utilización del canal y optimizar el tiempo para transferir un archivo grande, se envían varios paquetes dentro del marco de una ventana deslizante que se mueve

de acuerdo a si se usa el mecanismo de GBN o SR. Además, TCP implementa control de flujo mediante el campo Window Size, que ajusta dinámicamente la cantidad de datos que pueden enviarse antes de recibir una confirmación, evitando saturar al receptor (cwnd y rwnd), de esta manera inicia un slow start y va incrementando, si se pasa el umbral de la ventana cwnd, se debe ralentizar o reiniciar el flujo que envía el emisor, dependiendo de la versión, TCP Tahoe o TCP Reno. De esta forma, mantiene una comunicación confiable, ordenada y con corrección de errores.

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

Se pueden observar posibles pérdidas o desorden en los mensajes, ya que este protocolo no confirma la entrega ni verifica el orden de llegada. En Wireshark, esto se evidencia cuando hay paquetes ausentes o intervalos irregulares en la gráfica de tráfico. UDP simplemente envía los datagramas sin importar si el receptor los recibe o no, pues tampoco tiene manera de saber como es la situación del lado del receptor.

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

TCP es un protocolo orientado a conexión, lo que significa que requiere un proceso de establecimiento previo (handshake) para la sesión y mantiene un intercambio continuo de control entre emisor y receptor. En contraste, UDP es no orientado a conexión: no realiza handshake, no establece sesión y cada paquete se envía de forma independiente, lo que lo hace más rápido, pero menos confiable y como consecuencia, tiene un encabezado mucho menor.

4. Actividad 6: 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?

Cada conexión broker-publicador tiene una conexión TCP o UDP. En este sentido, cada que se quiera tener un nuevo publicador, se deberá añadir una correspondiente conexión TCP y seguir el procedimiento propio de este protocolo, es decir, el handshake, control de flujo y congestión, y mantener la sesión para el canal persistente en caso de pérdida y retransmisión. Por otro lado, en UDP, los nuevos publicadores mandan los eventos de cada partido sin necesidad de crear un canal persistente, con menos overhead y el broker no tiene la necesidad de mandar ACKs o de control de sesión (SYN, FIN).

Teniendo esto en cuenta, si se transmitieran 100 partidos de manera simultánea, en el caso de TCP podrían producirse pérdidas de paquetes según el tamaño del buffer del receptor y la velocidad de lectura de la aplicación. En estos escenarios, el emisor debería reenviar los mensajes perdidos y los demás paquetes tendrían que esperar para mantener el orden de entrega, lo que generaría una sobrecarga en el broker. Esta situación demandaría una infraestructura con mayor capacidad de CPU y memoria RAM, capaz de gestionar múltiples conexiones, numerosos paquetes de confirmación (ACKs) y los mecanismos de control de flujo y congestión inherentes al protocolo.

Por otro lado, UDP solo envía los paquetes directamente del emisor al receptor y ese es todo el proceso; el broker UDP no tendría tanta carga de trabajo, ya que no mantiene estados de conexión, no realiza retransmisiones ni controla el flujo de datos. Esto reduce significativamente el uso de CPU y memoria, permitiendo manejar un mayor número de transmisiones simultáneas con menor consumo de recursos. Sin embargo, si se pierde un paquete o los mensajes llegan fuera de orden, no existe

ningún mecanismo interno que los recupere u ordene, por lo que el broker podría entregar información incompleta o inconsistente.

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 el mundo del fútbol, cada evento que pasa es supremamente importante, si UDP no recibiera el gol y se estuviera usando el patrón publish/subscriber para una aplicación de apuestas deportivas, el usuario podría apostar a que un equipo hace el gol y como el paquete se perdió, el usuario perdió la apuesta y su dinero; y siendo que el resultado final es diferente al que tiene la aplicación, la aplicación de apuestas deportivas podría tener problemas legales, perdería toda credibilidad y tendría que corregir muchos casos manualmente.

Para TCP, este protocolo maneja mejor este caso ya que los paquetes llegarán ordenados y si alguno se pierde, se retransmitirá. Esto permite evitar inconsistencias y problemas de confiabilidad del usuario, por ejemplo, si un paquete que lleva el gol 1-0 se perdiera y llega justo el 2-0, TCP esperaría la retransmisión del primer paquete para que la aplicación luego pueda leer los datos y mostrar 1-0 y luego 2-0, evitando confusiones y problemas de consistencia (2-0 y luego 1-0 el usuario puede pensar que anularon el gol y van 1-0).

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 del objetivo de la aplicación, si los datos son críticos, por ejemplo, goles, tarjetas rojas, o eventos que puedan alterar apuestas deportivas, resultados finales o la clasificación de un equipo, se debería usar TCP para garantizar que el orden de llegada de los paquetes (goles, eventos, etc) y aunque se pierdan, puedan ser retransmitidos.

De lo contrario, si no se trata de datos críticos —por ejemplo, la posición del balón en tiempo real—, que cambia constantemente incluso dentro de un solo minuto, es preferible utilizar UDP. Este protocolo permite enviar los paquetes de manera continua y sin la espera de confirmaciones, garantizando que la información más reciente llegue con la menor latencia posible.

Así, se evita que ocurra una situación en la que el servidor envía un paquete indicando que el balón está en el metro 20, pero debido a los retrasos y retransmisiones propias de TCP, ese paquete llega después de otro que indica que el balón está en el metro 1. En consecuencia, UDP mantiene la coherencia temporal del flujo de datos, priorizando la actualidad de la información sobre la entrega confiable de cada paquete. Asimismo, tampoco importa mucho si algún paquete se pierde, el sistema puede seguir funcionando con la siguiente posición del balón que llegue.

Esto se puede ver durante la práctica cuando en las gráficas I/O, los gráficos 1 y 2, la cantidad de paquetes que puede mandar UDP por segundo es mucho más alta que en TCP

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?

De acuerdo con la teoría, UDP tiene una cabecera de 8 bits mientras que TCP usa de 32 bits para poder llevar todas las flags, números de secuencia, ventana de transmisión y esto se evidenció también en la práctica como se puede ver a continuación en la imagen 6 y 7:

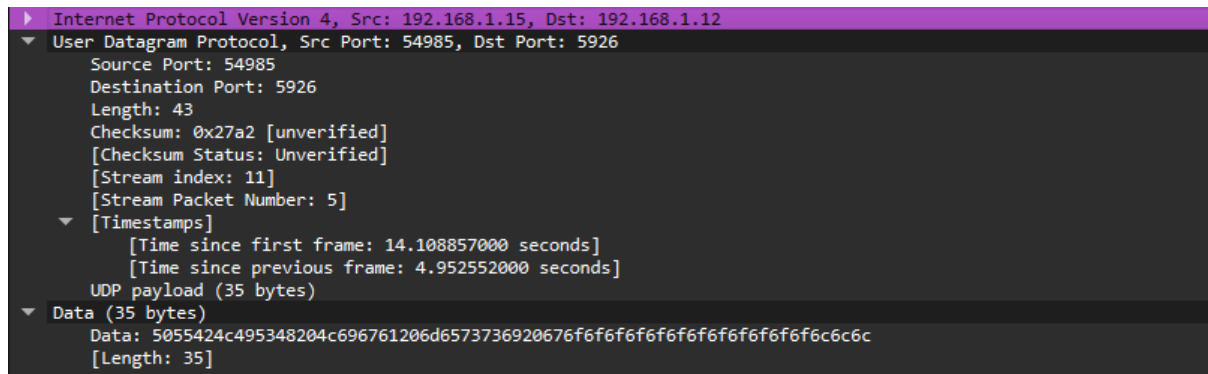


Imagen 6: Cabecera de UDP

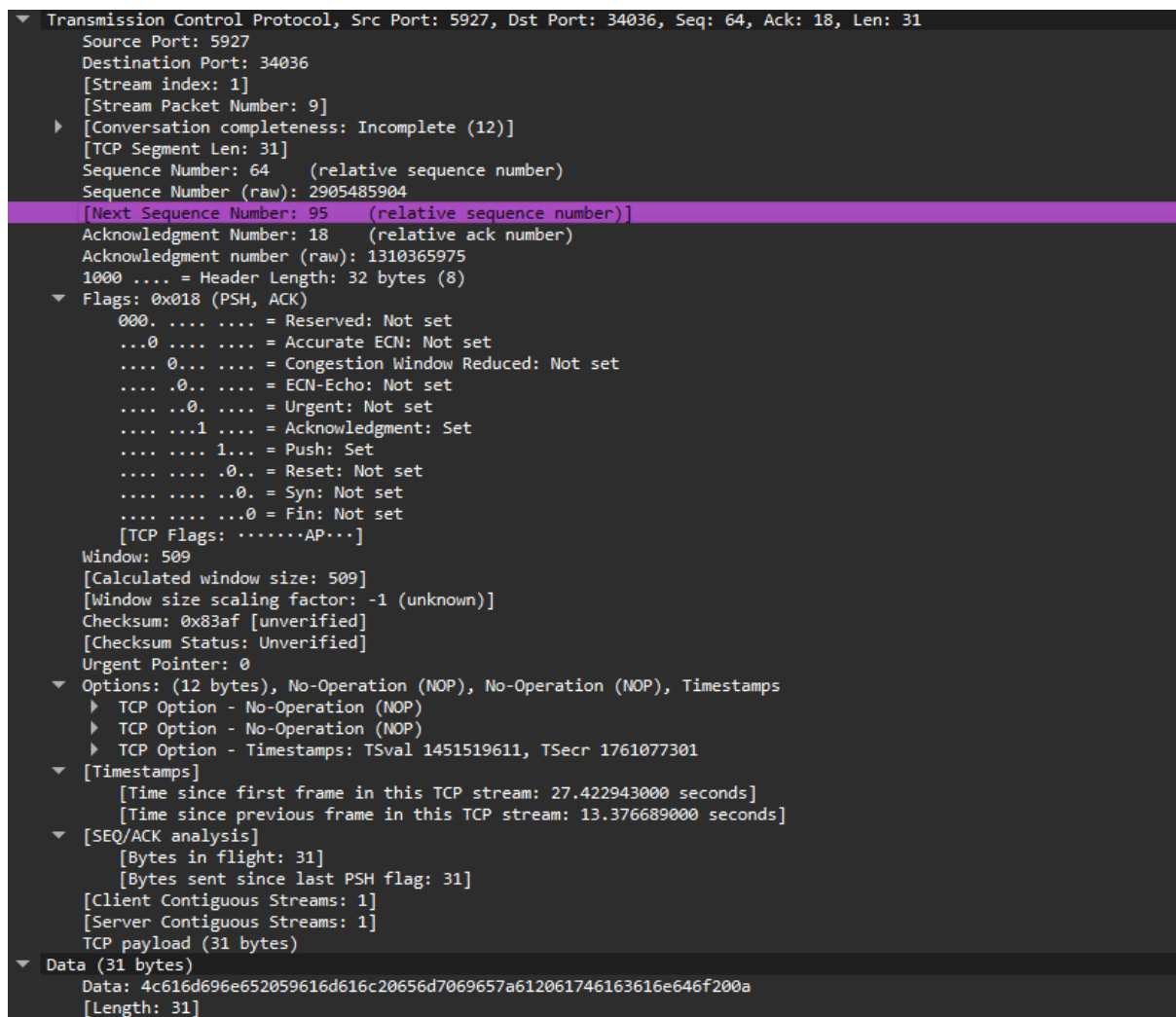


Imagen 7: Cabecera de TCP.

El uso de TCP puede disminuir ligeramente el *throughput* efectivo en la transmisión de mensajes o archivos, debido a que cada paquete incluye una cabecera más grande que la de UDP y requiere

intercambios adicionales de control. Como resultado, una parte mayor de los bits transmitidos pertenece al protocolo y no al contenido útil. Sin embargo, este impacto es significativo solo en mensajes pequeños o comunicaciones muy frecuentes donde el porcentaje de bits dedicados al protocolo es bastante más grande que en transferencias grandes donde la diferencia de rendimiento es mínima. A cambio, TCP ofrece mecanismos de fiabilidad, control de flujo y corrección de errores que garantizan una entrega ordenada y segura de los 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?

El usuario experimentará confusión y pérdida de confianza en la aplicación, ya que el sistema mostraría información inconsistente o sería poco creíble. Para evitarlo, la aplicación debe implementar mecanismos de verificación y control, como validar la coherencia de los datos (por ejemplo, que el número de goles solo aumente secuencialmente, de 1 en 1), asignar números de secuencia o marcas de tiempo a los eventos y descartar o reordenar aquellos que lleguen fuera de orden lógico (Un gol de un jugador y una tarjeta roja deben ir en orden, un jugador expulsado no pudo haber marcado gol). De esta forma se garantiza que el estado mostrado al usuario siempre sea lógico y cronológicamente correcto.

¿Cómo cambia el desempeño del sistema cuando aumenta el número de suscriptores interesados en un mismo partido? ¿Qué diferencias se observaron entre TCP y UDP en este aspecto?

Cuando aumenta el número de suscriptores interesados en un mismo partido, el desempeño del sistema se ve afectado de forma distinta según el protocolo. En UDP, el broker comienza a mostrar pérdida de paquetes y saturación en la cola de envío, ya que el protocolo no controla la congestión ni garantiza la entrega. A medida que crece la cantidad de receptores, los datagramas pueden perderse o llegar fuera de orden, y algunos clientes simplemente dejan de recibir actualizaciones. En cambio, en TCP, aunque el sistema mantiene la integridad de los datos gracias a las retransmisiones automáticas, el uso de CPU y memoria aumenta considerablemente, ya que el broker debe mantener un canal de comunicación individual y confiable con cada cliente, esto se expande en la pregunta correspondiente al uso de CPU de más abajo. Esto genera mayor latencia y tiempos de respuesta más largos, pero sin pérdida de información. En resumen, UDP escala mejor en cantidad pero con riesgo de pérdida, mientras que TCP asegura entrega, pero a costa de rendimiento y consumo de recursos.

¿Qué sucede si el broker se detiene inesperadamente? ¿Qué diferencias hay entre TCP y UDP en la capacidad de recuperación de la sesión?

Si el broker se detiene inesperadamente (Ctrl + C en las prácticas hechas), la comunicación entre clientes se interrumpe de inmediato. En el caso de UDP, los mensajes en tránsito se pierden por completo, ya que el protocolo no mantiene conexiones ni garantiza la entrega. Los publicadores seguirán enviando datagramas sin recibir errores, pero los suscriptores dejarán de recibirlos hasta que el broker se reinicie. En consecuencia, no existe una recuperación automática de sesión ni reenvío de datos perdidos.

En cambio, con TCP, cuando el broker se detiene, las conexiones establecidas se cierran abruptamente y los clientes reciben errores como connection reset o broken pipe. Aunque la sesión se interrumpe, TCP sí detecta la pérdida de conexión y puede restablecer la sesión una vez que el servidor vuelve a

estar disponible, mediante un nuevo handshake. En nuestras pruebas se observó que en UDP los clientes continúan enviando datos sin saber que el servidor cayó, mientras que en TCP los clientes son notificados inmediatamente y requieren reconectarse para continuar.

Además, en la versión TCP se implementó la opción `SO_REUSEADDR` mediante `setsockopt()`, documentada en el archivo `broker_tcp.c`. Esta configuración permite que el broker reinicie rápidamente sin tener que esperar a que el sistema operativo libere el puerto, incluso si las conexiones anteriores permanecen en estado `TIME_WAIT`. Gracias a esto, el broker puede volver a estar operativo de inmediato tras un reinicio manual, evitando errores como `EADDRINUSE` y reduciendo el tiempo de inactividad.

¿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?

En el contexto de redes, es difícil asegurar que todos los paquetes van a llegar exactamente al mismo instante a todos los clientes, por el tiempo de propagación de los paquetes, el router al que vaya, la ruta que tomen los paquetes, etc. Aún así, hay una diferencia clave entre TCP y UDP que hace que TCP sea mejor para este tipo de escenarios.

En UDP, aunque el envío es rápido y sin confirmaciones, los paquetes pueden llegar en momentos distintos o perderse, lo que hace imposible asegurar sincronía exacta entre todos los clientes. Por otro lado, en TCP la entrega simultánea y confiable se ve facilitada por el establecimiento de conexiones persistentes y garantiza que todos los receptores reciban los datos en orden y sin pérdidas.

Aún así, por lo general se suele optar por un modelo híbrido como QUIC para abordar estos casos.

Analice el uso de CPU y memoria en el broker cuando maneja múltiples conexiones TCP frente al manejo de datagramas UDP. ¿Qué diferencias encontró?

The image shows a terminal window on the left with the following text:

```
[Broker] PUB: topic=2 msg=4
[Broker] PUB: topic=2 msg=234
[Broker] PUB: topic=2 msg=2
[Broker] PUB: topic=2 msg=34
[Broker] PUB: topic=2 msg=234
[Broker] PUB: topic=2 msg=23
[Broker] PUB: topic=2 msg=4
[Broker] PUB: topic=2 msg=234
[Broker] PUB: topic=2 msg=234
[Broker] PUB: topic=2 msg=324
[Broker] PUB: topic=2 msg=23
^C
udp@udpbrokerdesk:~/Lab3$ gcc broker_udp.c -o broker_udp
udp@udpbrokerdesk:~/Lab3$ ./broker_udp
Broker UDP escuchando en puerto 5926...
```

On the right, the output of the `top` command is shown, with process 3533 highlighted in blue. The relevant row is:

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3533	udp	20	0	8664	5376	3840	S	0.0	0.1	0:00.02	bash

Imagen 8: Uso de CPU y Memoria del proceso 3533 (broker) en UDP

The image shows a terminal window on the left with the following text:

```
[Broker] PUB: topic=Premies msg=Funciona
[Broker] PUB: topic=Liga msg=2
[Broker] PUB: topic=Liga msg=2
[Broker] SUB: fd=9 topic=Liga
[Broker] SUB: fd=8 topic=Liga
[Broker] PUB: topic=Liga msg=2
[Broker] PUB: topic=Liga msg=8
[Broker] PUB: topic=Liga msg=99
^C
udp@udpbrokerdesk:~/Lab3$ echo $$
3533
udp@udpbrokerdesk:~/Lab3$ ./broker_tcp
Broker TCP escuchando en puerto 5927...
[Broker] SUB: fd=5 topic=Liga
[Broker] SUB: fd=4 topic=2
[Broker] SUB: fd=6 topic=2
```

On the right, the output of the `top` command is shown, with process 3533 highlighted in blue. The relevant row is:

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3533	udp	20	0	8664	5376	3840	S	10.0	1.1	0:00.01	bash

Imagen 9: Uso de CPU y Memoria del proceso 3533 (broker) en TCP

Como se puede ver en la imagen 8 y 9, en el caso de TCP, el broker mostró un mayor uso de CPU y memoria al manejar múltiples conexiones simultáneas. Esto se debe a que TCP requiere mantener un estado por conexión, incluyendo buffers de envío y recepción, colas de retransmisión, y estructuras de control para garantizar la entrega y el orden de los paquetes. Cada cliente conectado agrega una sobrecarga adicional, ya que el kernel debe gestionar confirmaciones (*ACKs*), control de flujo y posibles retransmisiones.

Por el contrario, en el broker UDP, el consumo de CPU y memoria fue considerablemente menor y más estable. Como UDP es un protocolo sin conexión, el broker no necesita mantener información persistente de cada cliente ni realizar confirmaciones. Solo procesa los datagramas entrantes y los reenvía, lo que se traduce en un manejo más ligero y eficiente a nivel de recursos.

En resumen, TCP ofrece fiabilidad a costa de mayor carga de procesamiento, mientras que UDP sacrifica esa fiabilidad en favor de un rendimiento más alto y menor uso de recursos, lo que resulta ideal para transmisiones rápidas o eventos donde la inmediatez prima sobre la confirmación de entrega.

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

Para un sistema real que deba transmitir actualizaciones de partidos de fútbol a millones de usuarios en tiempo real, elegiría QUIC como protocolo base. QUIC combina lo mejor de TCP y UDP: ofrece baja latencia gracias a su transporte sobre UDP y, al mismo tiempo, garantiza fiabilidad, cifrado y control de congestión integrados como en TCP. A diferencia de UDP puro, maneja la pérdida de paquetes de forma independiente por flujo, evitando bloqueos entre transmisiones concurrentes, y permite reconexiones instantáneas al cambiar de red. Basado en lo observado en el laboratorio, donde UDP demostró ser más liviano pero menos confiable y TCP más estable pero costoso en recursos, QUIC surge como una solución intermedia pues logra la velocidad y escalabilidad de UDP con la seguridad y consistencia de TCP, lo que lo hace perfecto para una plataforma global de resultados deportivos en tiempo real.

Conclusión

El desarrollo de este laboratorio permitió comprender de manera práctica el funcionamiento de la capa de transporte, las diferencias entre los protocolos TCP y UDP, y cómo estas afectan la confiabilidad, la velocidad y el uso de recursos en una comunicación cliente-servidor. A través de la implementación de los programas broker, publisher y subscriber, se evidenció que TCP garantiza la entrega ordenada y confiable de los mensajes mediante el uso de mecanismos como los números de secuencia, confirmaciones (*ACK*) y control de flujo, mientras que UDP ofrece una transmisión más rápida y ligera, sacrificando la fiabilidad a cambio de menor latencia y consumo de recursos.

Todo este trabajo fue posible gracias a la comprensión y correcto uso de la librería `sockets.h` en C, la cual permitió crear y manejar conexiones de red de manera controlada. Funciones como `socket()` para la creación de descriptores, `bind()` y `listen()` para la preparación del servidor, `accept()` para gestionar conexiones entrantes, y especialmente `connect()` para establecer sesiones TCP, así como `send()` y `recv()` para el envío y recepción de datos, fueron fundamentales para implementar la comunicación bidireccional entre procesos y observar las características de cada protocolo en acción.