

Taller UDP Connection

Integrantes:

- Juan Manuel Díaz - A00394477
- Esteban Gaviria - A00396019

1. ¿Es posible ver en la captura de Wireshark el contenido del mensaje enviado?

Sí, es posible capturar y visualizar el contenido de los mensajes enviados a través de nuestro programa que utiliza la clase `UDPConnection` con Wireshark. La clase `Receiver`, empleada por `UDPConnection`, utiliza el protocolo UDP para transmitir mensajes a través de un `DatagramSocket`. Los datos se envían sin cifrado, a menos que se agregue una capa de seguridad como TLS. Dado que los mensajes no están cifrados, Wireshark mostrará su contenido en texto plano (si son texto) o en formato hexadecimal (si son datos binarios). Para evitar que los mensajes sean visibles en Wireshark, sería necesario implementar algún tipo de cifrado antes de enviarlos.

Peer A:

ip.addr == 192.168.231.40

No.	Time	Source	Destination	Protocol	Length	Info
17506	107.348041	192.168.231.40	192.168.231.52	UDP	179	53832 → 9993 Len=137
17507	107.348103	192.168.231.52	192.168.231.40	UDP	101	9993 → 9993 Len=59
17508	107.348234	192.168.231.52	192.168.231.40	UDP	101	9993 → 38198 Len=59
17509	107.348306	192.168.231.52	192.168.231.40	UDP	101	9993 → 53832 Len=59
17510	107.348357	192.168.231.52	192.168.231.40	UDP	101	9993 → 9993 Len=59
17511	107.348412	192.168.231.52	192.168.231.40	UDP	101	9993 → 38198 Len=59
17512	107.348465	192.168.231.52	192.168.231.40	UDP	101	9993 → 53832 Len=59
17513	107.348519	192.168.231.52	192.168.231.40	UDP	101	9993 → 9993 Len=59
17514	107.348572	192.168.231.52	192.168.231.40	UDP	101	9993 → 38198 Len=59
17515	107.348626	192.168.231.52	192.168.231.40	UDP	101	9993 → 53832 Len=59
18625	113.156739	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
18644	113.164621	192.168.231.40	192.168.231.52	UDP	79	9993 → 47550 Len=68
18883	113.694459	192.168.231.52	192.168.231.40	UDP	108	5000 → 5000 Len=66
19008	117.600143	192.168.231.52	192.168.231.40	UDP	67	5000 → 5000 Len=25
19021	118.246053	192.168.231.52	192.168.231.40	UDP	70	44881 → 9993 Len=28
19051	118.251646	192.168.231.40	192.168.231.52	UDP	79	9993 → 44881 Len=37
19085	119.020825	192.168.231.52	192.168.231.40	UDP	67	5000 → 5000 Len=25
19216	121.894767	192.168.231.40	192.168.231.52	UDP	61	5000 → 5000 Len=19
19261	122.465977	192.168.231.40	192.168.231.52	UDP	70	9993 → 9993 Len=28
19262	122.465977	192.168.231.40	192.168.231.52	UDP	70	38198 → 9993 Len=28

Frame 19216: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface \Device\NPF...
 Ethernet II, Src: LiteonTechno...al:21:05 (74:4c:a1:21:05), Dst: CloudNetwork_9a:e9:b7 (50:c2:00:00:2f:2a)
 Internet Protocol Version 4, Src: 192.168.231.40, Dst: 192.168.231.52
 User Datagram Protocol, Src Port: 5000, Dst Port: 5000
 Data (49 bytes)
 0000 50 c2 e8 9a e9 b7 74 c a1 a1 21 05 00 00 45 00
 0010 00 2f 2a ba 00 00 80 11 c0 55 c0 a8 e7 28 c0 a8
 0020 c7 34 13 88 13 88 00 15 cd 76 55 53 55 41 52 49
 0030 4f 3a 20 41 4a 4a 41 4a 41 4a 41 4a 41 4a 41

P...tl...!...E
 /...U...(-
 4...VUSUARI
 0: AJAJAJ AJAJAJ

Peer B:

ip.addr == 192.168.231.52						
No.	Time	Source	Destination	Protocol	Length	Info
31274	215.204506	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
35654	240.740603	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
39274	255.903597	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
42815	271.038467	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
7674	49.070233	192.168.231.52	192.168.231.40	UDP	411	47550 → 9993 Len=369
10437	80.015765	192.168.231.52	192.168.231.40	UDP	79	47550 → 9993 Len=37
28283	200.028539	192.168.231.52	192.168.231.40	UDP	79	47550 → 9993 Len=37
41331	263.600645	192.168.231.40	192.168.231.52	UDP	54	5000 → 5000 Len=12
40650	260.849319	192.168.231.40	192.168.231.52	UDP	58	5000 → 5000 Len=16
43374	279.720482	192.168.231.40	192.168.231.52	UDP	61	5000 → 5000 Len=19
43570	282.948472	192.168.231.40	192.168.231.52	UDP	61	5000 → 5000 Len=19
43210	275.483050	192.168.231.52	192.168.231.40	UDP	67	5000 → 5000 Len=25
43245	276.904409	192.168.231.52	192.168.231.40	UDP	67	5000 → 5000 Len=25
36913	246.684229	192.168.231.40	192.168.231.52	UDP	83	5000 → 5000 Len=41
36340	244.530858	192.168.231.52	192.168.231.40	UDP	84	5000 → 5000 Len=42
42930	271.575188	192.168.231.52	192.168.231.40	UDP	108	5000 → 5000 Len=66
10388	79.988973	192.168.231.40	192.168.231.52	UDP	70	53832 → 44881 Len=28
> Frame 36340: 84 bytes on wire (672 bits), 84 bytes captured (672 bits) on interface 0 > Ethernet II, Src: CloudNet_9a:e9:b7 (50:c2:e8:9a:e9:b7), Dst: 192.168.231.40 (08:00:27:00:00:00) > Internet Protocol Version 4, Src: 192.168.231.52, Dst: 192.168.231.40 > User Datagram Protocol, Src Port: 5000, Dst Port: 5000 > Data (42 bytes)						
0000 74 4c a1 a1 21 05 50 c2 e8 9a e9 b7 08 00 45 00 0010 00 46 ba 24 00 00 00 11 30 d4 c0 a8 e7 34 c0 a8 0020 e7 28 13 88 13 88 00 32 94 64 45 73 74 65 62 61 0030 6e 20 47 61 76 69 72 69 61 3a 20 42 75 65 6e 61 0040 73 20 54 61 72 64 65 73 20 6d 69 20 65 73 74 69 0050 6d 61 64 6f						

2. ¿Cuál es el checksum de la captura? ¿Explique/investiguen por qué este checksum?

El checksum en el paquete UDP (en esta captura es '0x6b59') se utiliza para verificar la integridad de los datos transmitidos. En este caso, está marcado como "unverified" porque la tarjeta de red probablemente lo calcula después de que Wireshark captura el paquete (esto se llama checksum offloading). Por eso, Wireshark no puede verificar su validez. El valor exacto del checksum depende de los datos del paquete (direcciones IP, puertos, y el mensaje enviado), calculándose sumando ciertos campos del paquete.

25331	179.819629	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
27264	195.009668	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
31274	215.204506	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
35654	240.740603	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
39274	255.903597	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
42815	271.038467	192.168.231.52	192.168.231.40	UDP	70	47550 → 9993 Len=28
7674	49.070233	192.168.231.52	192.168.231.40	UDP	411	47550 → 9993 Len=369
10437	80.015765	192.168.231.52	192.168.231.40	UDP	79	47550 → 9993 Len=37
36913	246.684229	192.168.231.40	192.168.231.52	UDP	83	5000 → 5000 Len=41
36340	244.530858	192.168.231.52	192.168.231.40	UDP	84	5000 → 5000 Len=42
42930	271.575188	192.168.231.52	192.168.231.40	UDP	108	5000 → 5000 Len=66
10388	79.988973	192.168.231.40	192.168.231.52	UDP	70	53832 → 44881 Len=28
> Frame 36913: 83 bytes on wire (664 bits), 83 bytes captured (664 bits) on interface 0 > Ethernet II, Src: LiteonTe_a1:21:05 (74:4c:a1:a1:21:05), Dst: 192.168.231.52 (08:00:27:00:00:00) > Internet Protocol Version 4, Src: 192.168.231.40, Dst: 192.168.231.52 > User Datagram Protocol, Src Port: 5000, Dst Port: 5000 Source Port: 5000 Destination Port: 5000 Length: 49 Checksum: 0x6b59 [unverified] [Checksum Status: Unverified] [Stream index: 1764] > [Timestamps] UDP payload (41 bytes) > Data (41 bytes)						
0000 50 c2 e8 9a e9 b7 74 4c a1 a1 21 05 08 00 45 00 0010 00 45 2a a6 00 00 00 11 c0 53 c0 a8 e7 28 c0 a8 0020 e7 34 13 88 13 88 00 31 6b 59 55 53 55 41 52 49 0030 4f 3a 20 48 6f 6c 61 20 63 6f 6d 70 61 c3 b1 65 0040 72 6f 2c 20 c2 bf 51 75 c3 a9 20 74 61 6c 20 74 0050 c3 ba 20						

3. ¿Qué patrones de diseño/arquitectura aplicaría al desarrollo de un programa basado en red como este?

Patrón Singleton: Garantizaría que cada peer tenga una única instancia de la clase que maneja las comunicaciones y que esta instancia esté disponible globalmente dentro de la aplicación. Este patrón en específico fue el utilizado para la implementación del código.

Patrón de Proactor: Este patrón es ideal para aplicaciones de red asíncronas como un chat, ya que permite manejar eventos de red sin bloquear el programa principal, delegando la recepción de mensajes a un manejador de eventos.

Patrón Reactor: Es una alternativa al Proactor y es útil para manejar eventos de E/S (envío y recepción de mensajes). En este caso, se podría implementar un solo hilo para monitorizar múltiples sockets de forma no bloqueante.

Patrón Observer: Este patrón permitiría que la interfaz de usuario u otras partes del programa se suscriban a eventos de recepción de mensajes. De esta manera, cuando un mensaje es recibido por el socket, los suscriptores son notificados para actualizar la UI o realizar otras acciones.

Patrón Adapter: Dado que UDP es un protocolo orientado a datagramas, podrías usar este patrón para adaptar los mensajes de la aplicación (objetos o estructuras más complejas) a un formato adecuado para ser enviado a través de la red, y viceversa.

Patrón Peer-to-Peer (P2P): Desde el punto de vista de arquitectura, el sistema podría implementarse con una arquitectura P2P, donde cada cliente actúa tanto como cliente y servidor, permitiendo una comunicación directa entre ambos peers.

Thread Pool: Este patrón es útil para gestionar un conjunto de hilos reutilizables, lo que puede mejorar la eficiencia al manejar múltiples conexiones o tareas simultáneas sin crear y destruir hilos constantemente. En una aplicación de red como un chat, un Thread Pool podría gestionar la recepción y envío de mensajes, o la monitorización de varios sockets de manera eficiente.

Patrón Strategy: Podría ser aplicado para definir diferentes comportamientos de envío o recepción de mensajes, especialmente si se quiere cambiar entre diferentes protocolos de red (como UDP o TCP) o manejar diferentes tipos de eventos.

Circuit Breaker (desde el punto de vista arquitectónico): Este patrón sería útil para manejar fallos en la red, desconexiones temporales o caídas de peers. Permitirá que el sistema reaccione de manera controlada ante fallas, evitando intentos repetidos de conexión que puedan sobrecargar el sistema.

Proxy: Este patrón puede ser relevante si se decide abstraer la comunicación entre peers. El patrón Proxy permitiría manejar la interacción con otros peers o redes de manera más estructurada y controlada, especialmente si se necesita agregar aspectos como control de acceso o logs.

4. Investiguen qué modificaciones son necesarias para implementar este mismo sistema pero para la comunicación TCP en java.

Para modificar el sistema de comunicación de UDP a TCP, se deben ajustar varios aspectos clave. Primero, es necesario cambiar el uso de `DatagramSocket` por `Socket` y `ServerSocket` para establecer una conexión persistente entre cliente y servidor. El cliente se conectará a través de un `Socket`, mientras que el servidor utilizará un `ServerSocket` para aceptar conexiones. Además, en lugar de enviar y recibir datos con `DatagramPacket`, se usarán `InputStream` y `OutputStream` para manejar el flujo de datos de manera continua. También se deben considerar cambios en la gestión del cierre de la conexión, asegurando que los sockets se cierren adecuadamente al terminar la comunicación.

Por otro lado, TCP requiere un control más explícito del flujo de datos y puede bloquearse mientras espera la transmisión, a diferencia de UDP. Por lo tanto, se debe ajustar la lógica para gestionar estos comportamientos y manejar las excepciones relacionadas con problemas en la conexión.

5. ¿Qué utilidades de codificación o seguridad agregarías al código?

Cifrado de datos: Para proteger la información durante la transmisión, se puede implementar cifrado utilizando bibliotecas como Java Cryptography Extension (JCE). Por ejemplo, se podrían cifrar los mensajes antes de enviarlos usando algoritmos de cifrado simétrico (AES) o asimétrico (RSA), y descifrarlos al recibirlos.

Autenticación y validación: Se puede incorporar un sistema de autenticación para verificar la identidad de los peers antes de iniciar la comunicación. Esto podría hacerse mediante un intercambio de claves públicas/privadas o utilizando certificados digitales con SSL/TLS para establecer una conexión segura.

Control de acceso: Se podría agregar una capa de control que verifique qué usuarios tienen permiso para establecer la conexión o enviar ciertos comandos, usando listas de control de acceso (ACL) o tokens de autenticación.

Detección y prevención de ataques: Incluir lógica para prevenir ataques comunes como ataques de denegación de servicio (DoS) o spoofing, limitando la cantidad de conexiones simultáneas, estableciendo tiempos de espera (timeouts), y validando la dirección IP de los peers conectados.