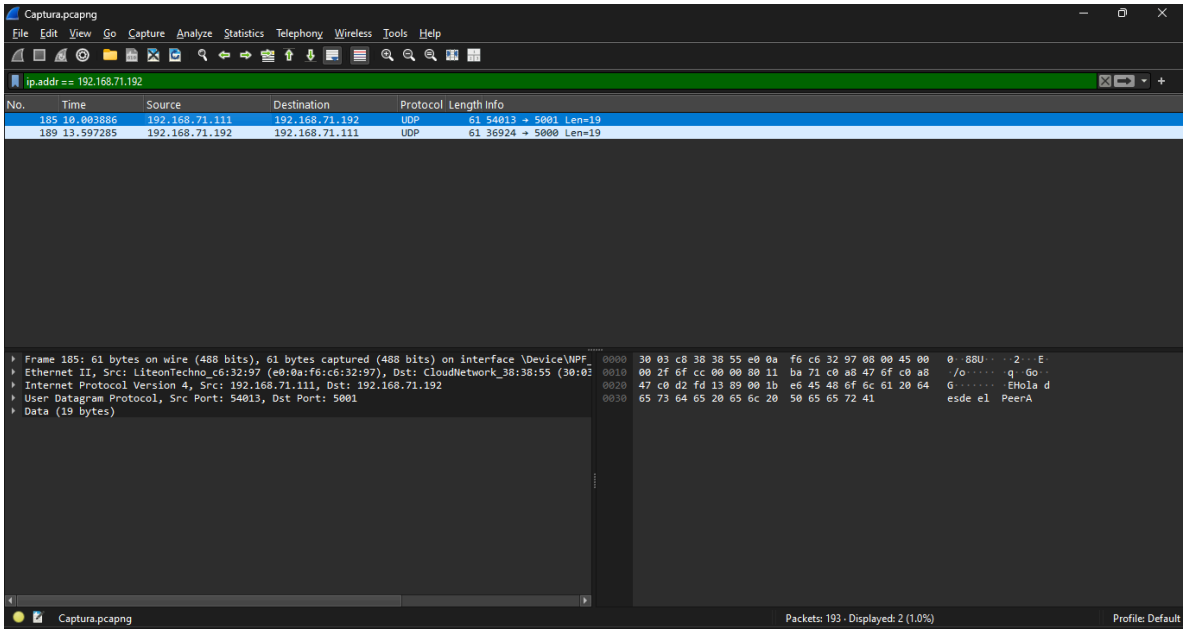
Integrantes:

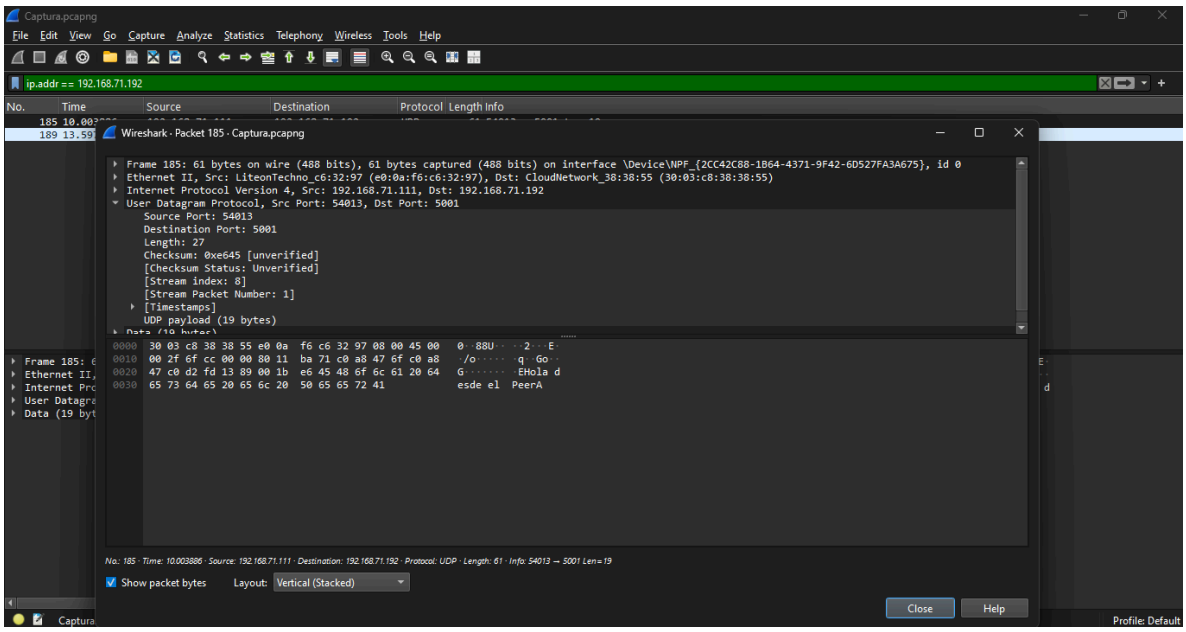
- David Artunduaga Penagos
- Rony Farid Ordóñez García

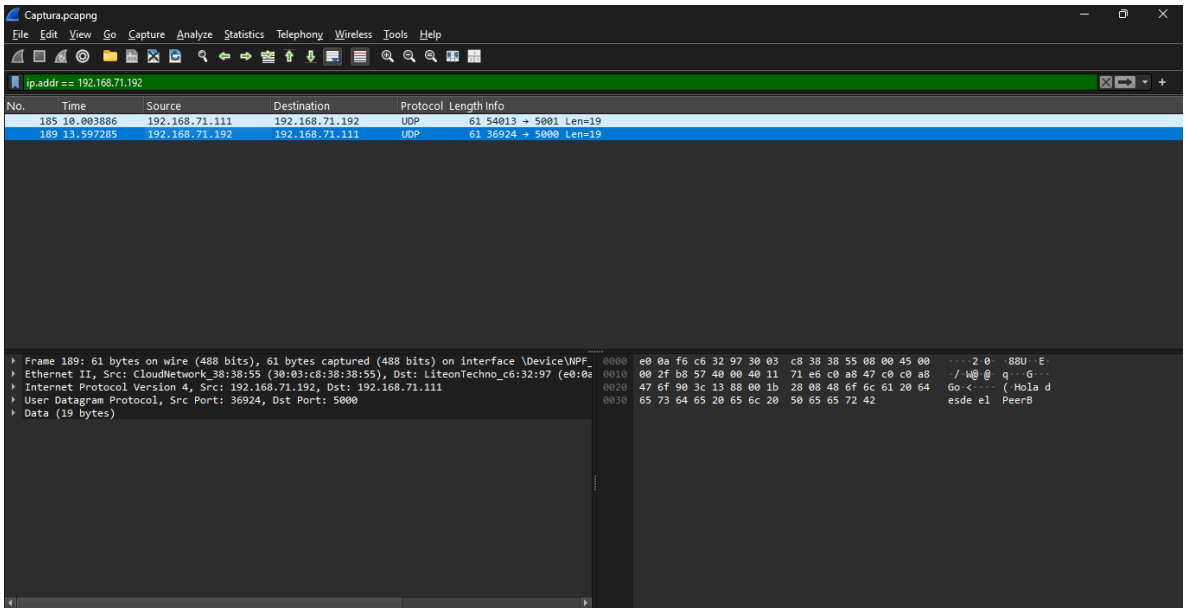
Repositorio: https://github.com/Rony7v7/UDP-Workshop

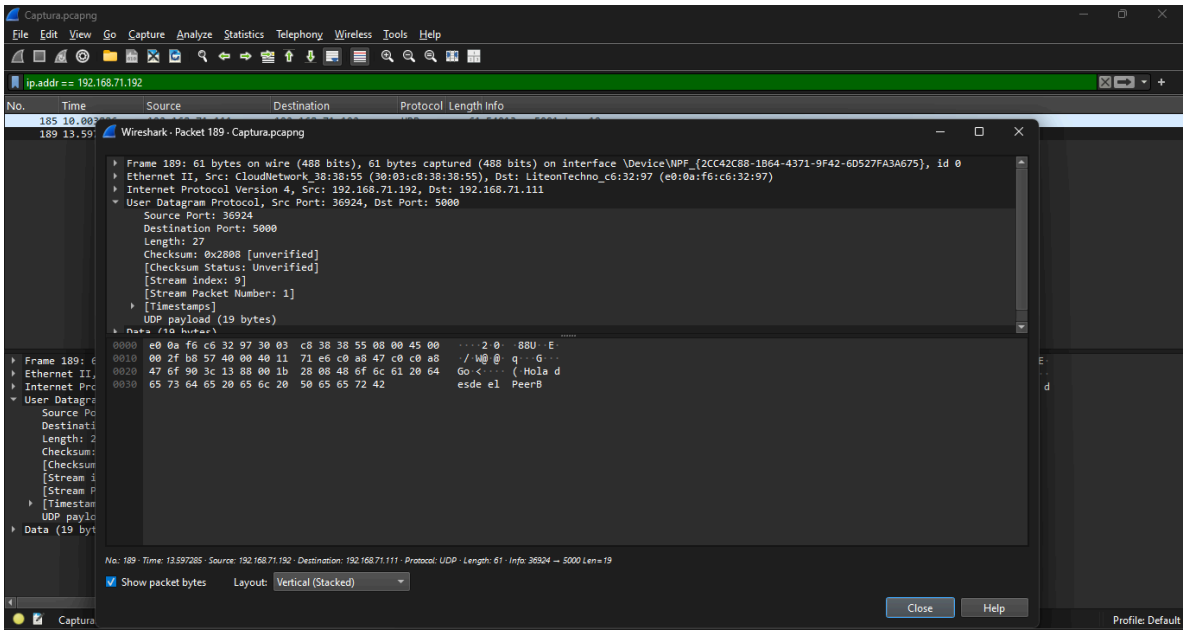
Capturas:

David:

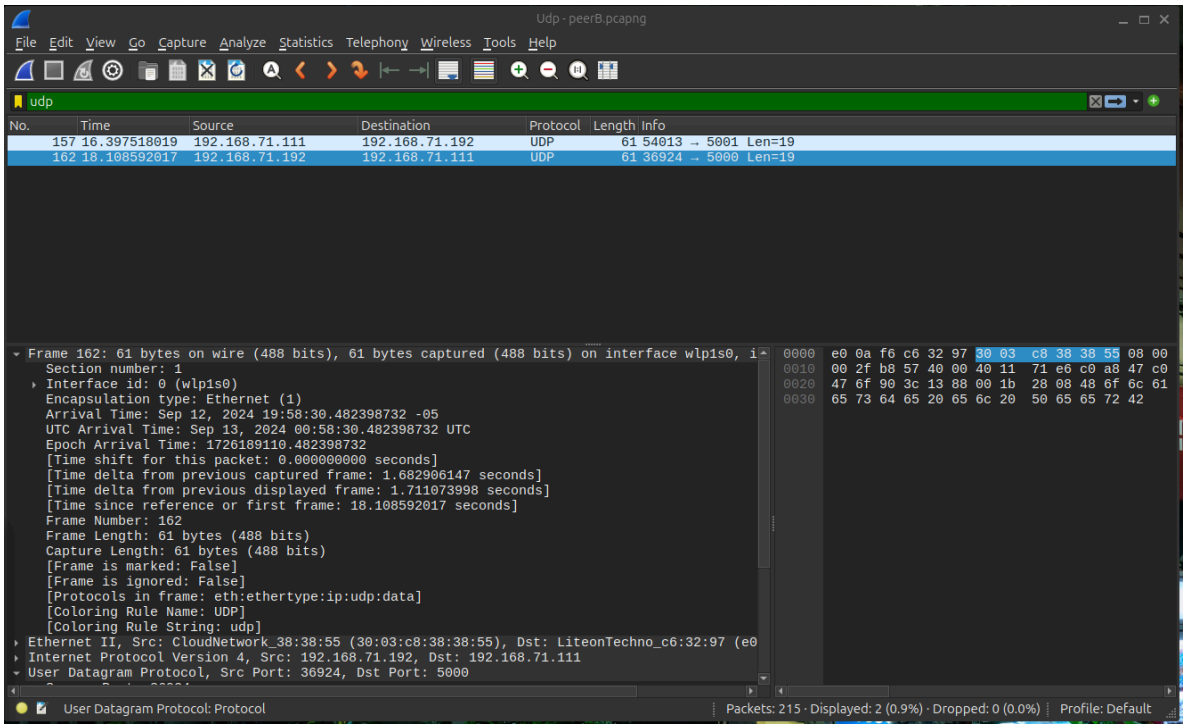


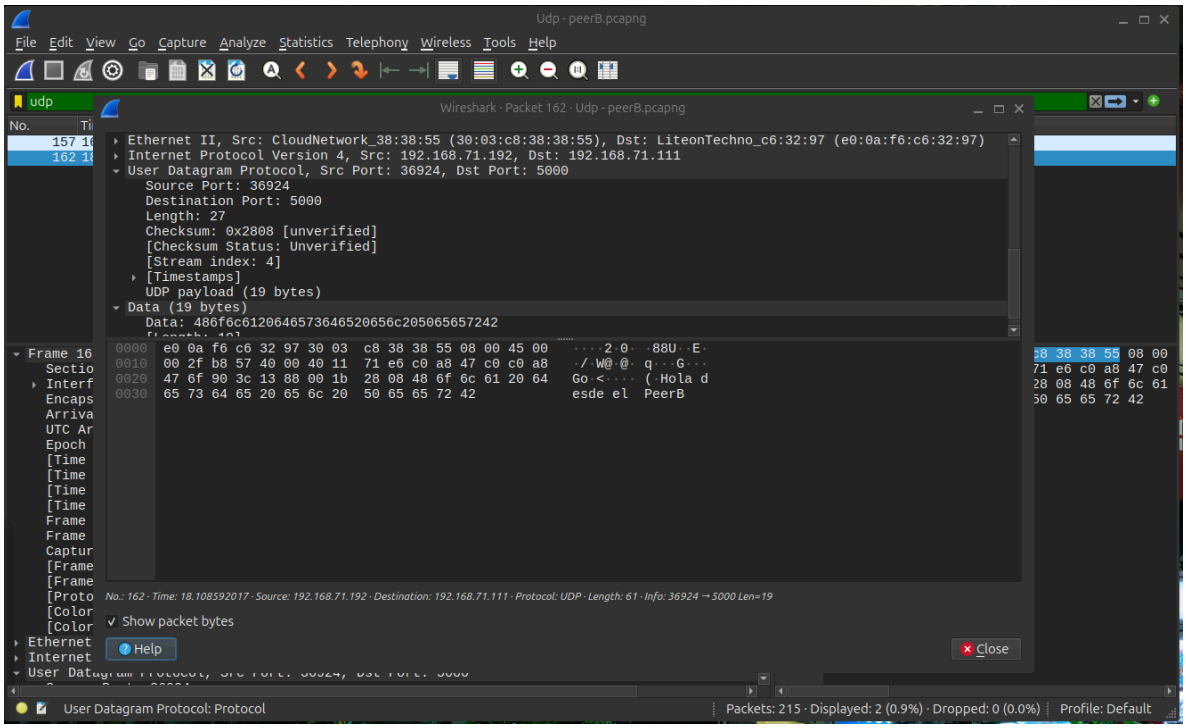



Captura.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 192.168.71.192
No. Time Source Destination Protocol Length Info
185 10.003886 192.168.71.111 192.168.71.192 UDP 61 54013 → 5001 Len=19
189 13.597285 192.168.71.192 192.168.71.111 UDP 61 36924 → 5000 Len=19
Frame 189: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface \Device\NPF_
Ethernet II, Src: CloudNetwork_38:38:55 (30:03:c8:38:38:55), Dst: LiteonTechno_c6:32:97 (e0:0a
Internet Protocol Version 4, Src: 192.168.71.192, Dst: 192.168.71.111
User Datagram Protocol, Src Port: 36924, Dst Port: 5000
Data (19 bytes)
0000 e0 0a f6 c6 32 97 30 03 c8 38 38 55 08 00 45 00
0010 00 2f b8 57 40 00 40 11 71 e6 c0 a8 47 c0 c0 a8
0020 47 6f 90 3c 13 88 00 1b 28 08 48 6f 6c 61 20 64
0030 65 73 64 65 20 65 6c 20 50 65 65 72 42
Go (Hola d
esde el PeerB


Captura.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
ip.addr == 192.168.71.192
Wireshark · Packet 189 · Captura.pcapng
Frame 189: 61 bytes on wire (488 bits), 61 bytes captured (488 bits) on interface \Device\NPF_{2CC42C88-1B64-4371-9F42-6D527FA3A675}, id 0
Ethernet II, Src: CloudNetwork_38:38:55 (30:03:c8:38:38:55), Dst: LiteonTechno_c6:32:97 (e0:0a:f6:c6:32:97)
Internet Protocol Version 4, Src: 192.168.71.192, Dst: 192.168.71.111
User Datagram Protocol, Src Port: 36924, Dst Port: 5000
Source Port: 36924
Destination Port: 5000
Length: 27
Checksum: 0x2808 [unverified]
[Checksum Status: Unverified]
[Stream index: 9]
[Stream Packet Number: 1]
[Timestamps]
UDP payload (19 bytes)
0000 e0 0a f6 c6 32 97 30 03 c8 38 38 55 08 00 45 00
0010 00 2f b8 57 40 00 40 11 71 e6 c0 a8 47 c0 c0 a8
0020 47 6f 90 3c 13 88 00 1b 28 08 48 6f 6c 61 20 64
0030 65 73 64 65 20 65 6c 20 50 65 65 72 42
Go (Hola d
esde el PeerB
No.: 189 · Time: 13.597285 · Source: 192.168.71.192 · Destination: 192.168.71.111 · Protocol: UDP · Length: 61 · Info: 36924 → 5000 Len=19
Show packet bytes
Layout: Vertical (Stacked)
Close
Help
Profile: Default

Rony:





En las anteriores imágenes se pueden observar la captura de paquetes entre los dos Peers. Adicionalmente adjuntamos el archivo .pcapng

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

R/ Sí, es posible ver el contenido del mensaje enviado en las capturas de Wireshark. De hecho, en la parte inferior derecha de las imágenes 1 y 3, o en la parte inferior de las imágenes 2 y 4, se puede observar dicho contenido. Allí se muestra la representación hexadecimal del mensaje, y Wireshark también intenta convertir esos valores a caracteres ASCII, lo que permite leer el mensaje siempre que esté en un formato compatible (como texto plano).

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

R/ Para el paquete de la imagen 2 el checksum de la captura es 0xe645 y para la imagen 4 es 0x2808. Este valor se genera sumando todos los campos del encabezado UDP, los datos del mensaje y una pseudo-cabecera que incluye las direcciones IP de origen y destino, el protocolo UDP y la longitud del paquete. El checksum asegura la integridad de los datos, permitiendo al receptor verificar si los datos llegaron sin errores.Además, Wireshark muestra "Unverified" porque no siempre verifica el checksum.

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

R/ Algunos patrones podrían ser:

Patrón Singleton:

Descripción: Asegura que solo haya una única instancia de una clase en todo el sistema, garantizando el control centralizado de los recursos de red (como sockets).

Aplicación: Utilizado para gestionar la conexión de red (como en UDPConnection), evitando múltiples conexiones conflictivas.

Patrón de Productor-Consumidor:

Descripción: Separa el componente que genera mensajes (productor) del componente que los consume (consumidor), permitiendo que trabajen de manera asíncrona.

Aplicación: Los Peers pueden actuar como productores cuando envían mensajes y consumidores cuando reciben, gestionando una cola de mensajes entrantes y salientes de manera asíncrona.

Patrón Observer (Observador):

Descripción: Permite que un objeto notifique a otros objetos sobre cambios en su estado sin conocer los detalles de esos objetos.

Aplicación: Un servidor puede notificar a múltiples clientes cuando ocurre un evento, como un mensaje recibido, sin que el servidor necesite saber qué clientes están conectados.

Patrón Reactor:

Descripción: Gestiona múltiples conexiones simultáneas de manera no bloqueante, delegando el manejo de eventos de red a diferentes manejadores.

Aplicación: Ideal para aplicaciones que manejan muchas conexiones de red concurrentes, como un servidor que recibe múltiples mensajes UDP de diferentes clientes.

Patrón Proxy:

Descripción: Proporciona un objeto intermediario que controla el acceso a otro objeto, añadiendo control o funciones adicionales.

Aplicación: Se puede usar para representar un cliente o servidor remoto en un sistema distribuido, ocultando la complejidad de las conexiones de red.

Patrón Facade (Fachada):

Descripción: Proporciona una interfaz simplificada a un sistema complejo.

Aplicación: Se puede encapsular la lógica de manejo de sockets, envío y recepción de datos detrás de una clase ConnectionManager, ocultando los detalles de bajo nivel a los Peers.

Patrón State (Estado):

Descripción: Permite a un objeto cambiar su comportamiento cuando cambia su estado interno.

Aplicación: Los Peers pueden cambiar de estado entre "escuchando" y "enviando" mensajes, con cada estado manejando la lógica de red de manera diferente.

Patrón Circuit Breaker:

Descripción: Protege un sistema de fallos prolongados al limitar el acceso a recursos fallidos hasta que se restauren.

Aplicación: Útil para controlar la disponibilidad de conexiones de red, cerrando temporalmente la conexión si hay fallos repetidos.

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

R/ Las principales modificaciones serían:

Cambiar DatagramSocket a Socket (para el cliente) y ServerSocket (para el servidor): TCP es orientado a conexión, por lo que se necesita establecer una conexión antes de enviar/recibir datos.

Manejo de Conexiones: TCP requiere que el servidor acepte conexiones con accept(), y el cliente debe conectarse usando connect().

Envío y Recepción de Datos: En TCP, se utiliza InputStream y OutputStream en lugar de DatagramPacket. Los datos se envían y reciben a través de estos flujos.

Conexión Persistente: A diferencia de UDP, TCP mantiene la conexión abierta, por lo que no se necesita crear y cerrar sockets repetidamente.

¿Qué utilidades de codificación o seguridad agregaría al código?

R/ Para mejorar el código se pueden agregar:

Cifrado de Datos: Usar SSL/TLS para proteger los mensajes.

Autenticación: Verificar que solo clientes autorizados accedan.

Validación de Entradas: Prevenir datos malformados o ataques.

Manejo de Excepciones: Mejorar la captura de errores.

Timeouts: Desconectar clientes inactivos automáticamente.

Protección contra DoS: Limitar el número de conexiones y el tráfico.