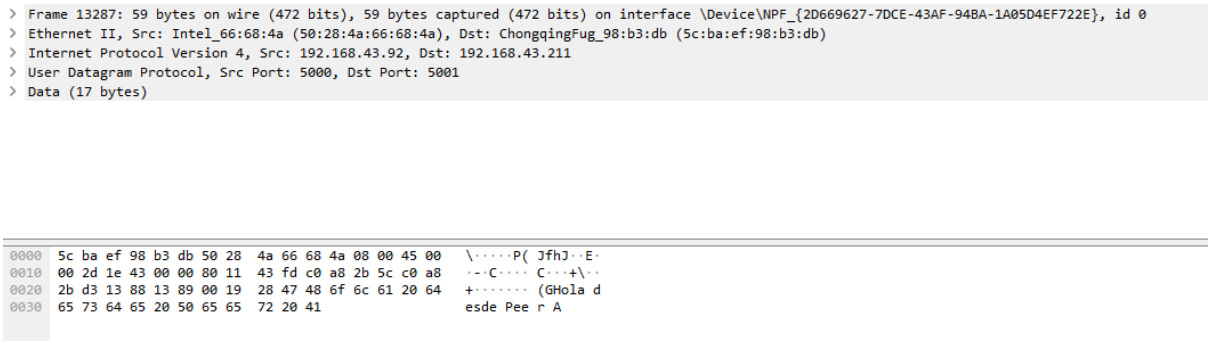
Manuel Cardona
Juan Camilo Muñoz

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



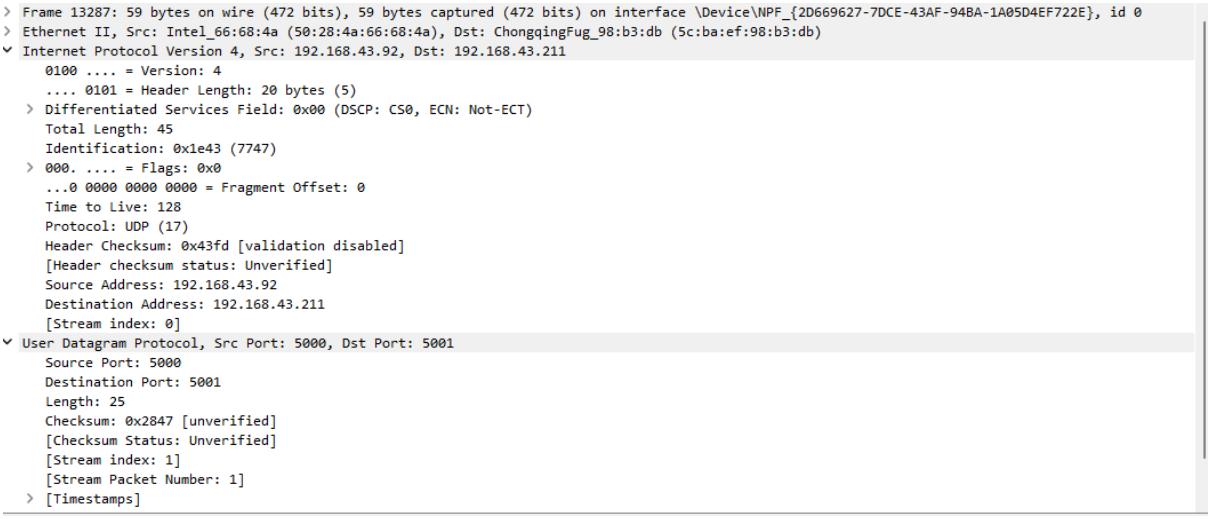
La información relevante que muestra Wireshark incluye:

- **Protocolo UDP**: El paquete utiliza UDP, con un puerto de origen 5000 y un puerto de destino 5001.
- **Direcciones IP**: El paquete se envía desde la IP 192.168.43.92 hacia 192.168.43.211.
- **Datos**: El contenido del mensaje es “Hola desde Peer A” , y está visible en la parte inferior del panel, en formato hexadecimal y ASCII.

Esto confirma que, al utilizar UDP sin encriptación, es posible ver el contenido de los mensajes en texto claro.

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

   Respuesta :



Respuesta

: **¿Cuál es el *checksum* en la captura?**

El *checksum* mostrado en el paquete UDP es 0x2847. Este valor es utilizado para verificar la integridad del paquete durante la transmisión.

¿Por qué este *checksum*?

El *checksum* en un paquete UDP es opcional pero, cuando se utiliza, se calcula sumando todos los bloques de 16 bits del paquete (incluyendo datos del encabezado y la carga útil), junto con una pseudo-cabecera que incluye información de la capa IP (como las direcciones IP de origen y destino). Si la suma resulta en un número mayor a 16 bits, los bits adicionales se suman al número final. Luego, el valor se invierte (se usa el complemento a 1) para formar el *checksum*.

En este caso, el valor 0x2847 es el resultado de ese proceso. El receptor del paquete puede verificar este valor volviendo a calcular el *checksum* y comparándolo con el recibido. Si los dos valores coinciden, se asume que el paquete llegó sin errores

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

Respuesta : **Observer (Observador)**:

- **Contexto**: El programa tiene una interfaz gráfica (PeerUI) que se actualiza cuando llegan nuevos mensajes.
- **Aplicación**: Se podría usar este patrón para desacoplar la interfaz gráfica de la lógica de red. De este modo, UDP Connection podría notificar a múltiples suscriptores (como la UI o un logger) cuando recibe un mensaje, sin depender de un solo componente como PeerUI.

Strategy (Estrategia):

- **Contexto**: En la recepción y envío de mensajes, diferentes tipos de protocolos o formatos de mensajes podrían ser necesarios en el futuro.
- **Aplicación**: Usar el patrón Strategy para manejar diferentes tipos de procesamiento de mensajes. Por ejemplo, se podría definir distintas estrategias para procesar mensajes según el protocolo (TCP/UDP) o el tipo de contenido (texto, binario, JSON).

Factory (Fábrica):

- **Contexto**: La creación de sockets y mensajes podría variar dependiendo de las configuraciones de red o de diferentes escenarios.
- **Aplicación**: Un patrón Factory puede encargarse de la creación de objetos relacionados con la conexión (por ejemplo, sockets con configuraciones personalizadas) o incluso mensajes (si se quiere soportar múltiples formatos).

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

   Respuesta :

Uso de Socket y ServerSocket:

- En lugar de usar DatagramSocket para la comunicación UDP, se debe usar Socket para el cliente y ServerSocket para el servidor en TCP.
- El cliente se conectará al servidor con algo como Socket socket = new Socket("localhost", port);, y el servidor esperará conexiones con ServerSocket serverSocket = new ServerSocket(port);.
- Una vez que el servidor acepta una conexión (Socket clientSocket = serverSocket.accept();), se puede leer y escribir datos usando flujos de entrada y salida.

Lectura y escritura de datos:

- En TCP, se utiliza InputStream y OutputStream para la comunicación, como BufferedReader y PrintWriter, para leer y escribir los datos de manera más controlada.

Manejo de conexiones persistentes:

- TCP permite conexiones persistentes, es decir, se puede mantener la conexión abierta para intercambiar múltiples mensajes sin necesidad de reabrir el socket cada vez.
- En UDP, cada paquete es independiente, lo que no es el caso en TCP.

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

   Respuesta:

   1. Validación de Entradas

- **Validar el puerto**:se asegura de que el puerto que se está utilizando esté en un rango válido (0-65535) antes de abrir el socket.
- **Validar las direcciones IP**: se verifica que las direcciones IP que se utilizan en el método sendDatagram sean válidas antes de crear un paquete

2. Cifrado de Datos

Los mensajes que se envian por UDP pueden ser interceptados fácilmente. Una opción sería **cifrar los datos** antes de enviarlos y descifrarlos al recibirlos. se pueden usar algoritmos como AES (Advanced Encryption Standard)

3 . Uso de TLS (Transport Layer Security)

Si la aplicación necesita más seguridad, en lugar de usar directamente UDP, se podría considerar utilizar **TLS sobre UDP** (DTLS). DTLS proporciona cifrado similar a TLS, pero para protocolos de transporte basados en datagramas.

4. Resistencia a Ataques DoS

Si se deja el socket abierto indefinidamente, puede ser susceptible a ataques de denegación de servicio (DoS). Limita el tiempo de espera para recibir paquetes, y cierra el socket si no se reciben datos en un tiempo razonable