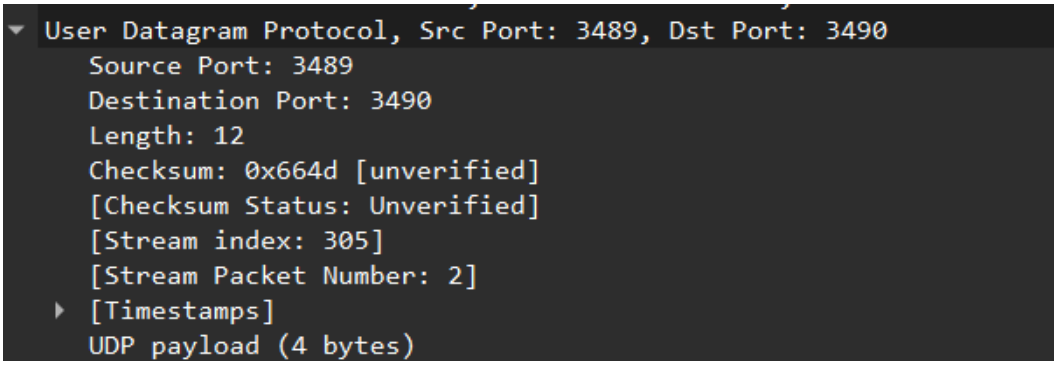
Respuestas a las preguntas

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

Sí, sí podemos verlo en la parte inferior derecha donde Wireshark muestra los datos en formato hexadecimal y ASCII.

- **Datos en formato ASCII**: En la parte inferior derecha, dentro del cuadro con la etiqueta "Data", puedes ver los caracteres "hola" en texto claro. Esto corresponde al contenido del mensaje enviado por el protocolo UDP. Está justo después de la cadena de caracteres "ff", lo que indica que el mensaje "hola" fue enviado como carga útil del paquete.
- **Datos en formato hexadecimal**: En la parte izquierda del mismo panel, Wireshark muestra el contenido del paquete en formato hexadecimal. Puedes ver que "hola" corresponde a los valores 68 6f 6c 61 en hexadecimal, que son las representaciones de las letras h, o, l, a respectivamente.

¿Cuál es el checksum de la captura? Explique e investigue



Wireshark muestra el estado de checksum como "unverified" porque no necesariamente está verificando o recalculando el checksum en este momento (esto puede depender de la configuración del software o del entorno de red).

El **checksum** de 0x664d fue calculado a partir de la suma de todos los segmentos de datos y encabezados, incluyendo la dirección IP de origen y destino, el protocolo UDP, la longitud del segmento, y la carga útil del paquete. El valor no está verificado por Wireshark debido a posibles configuraciones de hardware o software relacionadas con la interfaz de red, lo que es común en capturas realizadas en la misma máquina que genera los paquetes.

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

1. Patrón de diseño: Singleton

Contexto: En un programa de red, es común tener una instancia centralizada que gestione recursos compartidos como sockets, conexiones o configuraciones de red.

Aplicación: Utilizar el patrón Singleton asegura que solo exista una instancia del controlador de red (por ejemplo, para gestionar los sockets UDP) a lo largo de toda la aplicación, evitando conflictos y asegurando un punto de acceso global.

Ventajas:

- Control centralizado de los recursos de red.
- Ahorro de recursos al evitar múltiples instancias de gestión de conexiones.

2. Patrón de diseño: Factory (Fábrica)

Contexto: La creación de objetos relacionados con la red (por ejemplo, sockets, conexiones, paquetes) puede depender de diferentes protocolos o configuraciones.

Aplicación: Un **Factory** se puede utilizar para crear objetos de red (como conexiones TCP o UDP, dependiendo de los casos de uso), encapsulando la lógica de creación y proporcionando una interfaz abstracta.

Ventajas:

- Desacopla la creación de objetos de red, lo que facilita añadir más tipos de conexiones en el futuro.
- Promueve la reutilización de código.

3. Patrón de diseño: Observer

Contexto: En aplicaciones basadas en red, es común recibir eventos de red de manera asincrónica (por ejemplo, nuevos mensajes, conexiones, o desconexiones).

Aplicación: El patrón **Observer** permite que diferentes componentes se suscriban a eventos de red. Cuando se recibe un mensaje o ocurre algún evento de red, se notifican a todos los observadores (listeners).

Ventajas:

- Facilita la integración de diferentes partes del sistema que necesitan reaccionar a eventos de red.
- Separa el manejo de la lógica de eventos de la lógica de aplicación.

Ejemplo: Si tienes varios módulos que deben procesar los mensajes recibidos por el socket UDP, puedes hacer que se suscriban como observadores, para que reciban y procesen los mensajes sin necesidad de que el socket los maneje directamente.

4. Patrón de diseño: Command

Contexto: En un programa de red, es posible que diferentes tipos de mensajes o paquetes desencadenen diferentes acciones.

Aplicación: El patrón **Command** encapsula cada solicitud (o mensaje recibido) como un objeto. Cada comando puede ejecutarse dependiendo de su tipo, permitiendo una fácil extensión y manejo de diferentes tipos de mensajes.

Ventajas:

- Separa la lógica de ejecución de las solicitudes o mensajes.
- Facilita la implementación de operaciones como deshacer o reintentar.

5. Patrón de diseño: Reactor

Contexto: Cuando el programa de red debe manejar múltiples conexiones simultáneas (especialmente en redes asíncronas), el patrón **Reactor** es útil.

Aplicación: Este patrón utiliza un bucle de eventos para recibir y despachar eventos de entrada/salida (como recibir datos de un socket UDP o TCP). Cada evento dispara un manejador que procesa la entrada o salida.

Ventajas:

- Soporte eficiente para manejar múltiples conexiones simultáneas sin necesidad de crear un hilo por cada conexión.
- Escalabilidad, ya que se evita la sobrecarga de hilos múltiples.

Ejemplo: En lugar de tener un hilo por cada conexión UDP, un Reactor puede tener un solo hilo que escucha eventos de red y distribuye el trabajo a diferentes manejadores de eventos.

6. Patrón de diseño: Adapter

Contexto: En ocasiones, los diferentes tipos de conexiones o mensajes tienen interfaces distintas, y puede ser necesario adaptar una interfaz a otra.

Aplicación: El patrón **Adapter** permite que diferentes tipos de conexiones (por ejemplo, UDP o TCP) se adapten a una interfaz común, facilitando que el código de la aplicación pueda trabajar de forma uniforme sin preocuparse por los detalles de la implementación.

Ventajas:

- Permite un fácil cambio entre distintos tipos de conexiones.
- Promueve la reutilización del código.

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

Diferencias entre UDP y TCP

1. **Conexión**: TCP es un protocolo orientado a la conexión, lo que significa que primero se establece una conexión entre el cliente y el servidor antes de transferir datos. En UDP, los mensajes se envían sin establecer una conexión previa.
2. **Fiabilidad**: TCP garantiza la entrega de los datos en el orden correcto, lo que requiere manejo de conexiones, reenvío de paquetes perdidos, y confirmaciones de recepción. En UDP, no hay garantías de entrega ni de orden.
3. **Control de flujo**: TCP incluye mecanismos de control de flujo y congestión, mientras que UDP no.

¿Qué utilidades de codificación o seguridad agregaría al código?

Cifrado de Datos (TLS/SSL)

La seguridad más importante en una comunicación de red es el cifrado. Implementar TLS (Transport Layer Security) o SSL (Secure Sockets Layer) asegura que los datos que se envían entre el cliente y el servidor estén cifrados, de modo que incluso si los datos son interceptados, no puedan ser leídos por terceros.