

Tarea 2

DATAGRAMA ECO

FECHA

31 de marzo de 2025

MATERIA

Aplicaciones para comunicaciones de red

PROFESOR

Moreno Cervantes Axel Ernesto

ALUMNOS

- Guevara Badillo Areli Alejandra
- Ramírez Martínez Alejandro

GRUPO

6CM2

Introducción

En los sistemas de comunicación en red, los mensajes pueden ser demasiado grandes para ser enviados en un solo paquete. Esto es especialmente cierto en protocolos como **UDP (User Datagram Protocol)**, el cual no garantiza que los mensajes lleguen en orden ni que sean entregados correctamente.

Para resolver este problema, se utiliza un mecanismo de **fragmentación de mensajes**, donde la información se divide en partes más pequeñas y se envía en múltiples paquetes. Luego, en el destino, se deben **reordenar los fragmentos** y reconstruir el mensaje original.

En esta práctica, se implementó un **sistema de eco con fragmentación**, en el cual un cliente envía un mensaje fragmentado a un servidor, que debe recibir los fragmentos, ordenarlos y reconstruir el mensaje. Esto permite comprender cómo funcionan los protocolos sin conexión y cómo garantizar la entrega correcta de los datos.

Objetivo

Implementar un **sistema de eco basado en UDP**, en el cual:

- Un **cliente** fragmenta un mensaje y lo envía en paquetes.
- Un **servidor** recibe los paquetes, los almacena y los ordena.
- Al recibir todos los paquetes, el **servidor reconstruye y muestra el mensaje original**.

Desarrollo de la Práctica

El sistema está compuesto por dos programas principales:

1. **Cliente (ClienteEco.java):**
 - Solicita un mensaje al usuario.
 - Lo divide en fragmentos y lo envía por UDP.
 - Cada fragmento contiene **metadatos** para su correcta reconstrucción.
2. **Servidor (ServidorEco.java):**
 - Recibe los fragmentos en desorden.

- Guarda y reordena los fragmentos.
- Reconstruye y muestra el mensaje original cuando todos los paquetes han llegado.

CienteEco.java

El **cliente** es el encargado de solicitar un mensaje al usuario, dividirlo en fragmentos, agregar metadatos y enviarlo al servidor mediante **paquetes UDP**.

1. Importación de Librerías

```
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
import java.util.Scanner;
```

Las librerías importadas permiten:

- **DatagramPacket**: Crear los paquetes UDP con los datos del mensaje.
 - **DatagramSocket**: Crear el socket para enviar los paquetes.
 - **InetAddress**: Obtener la dirección IP del servidor.
 - **Scanner**: Capturar la entrada del usuario.
-

2. Constantes

```
private static final int MAX_BYTES = 10;  
private static final String SERVER_IP = "127.0.0.1";  
private static final int SERVER_PORT = 1234;
```

- **MAX_BYTES = 10**: Cada fragmento del mensaje tendrá un máximo de **10 bytes de datos**.
 - **SERVER_IP = "127.0.0.1"**: El servidor se encuentra en la **misma máquina** (localhost).
 - **SERVER_PORT = 1234**: Puerto donde está escuchando el servidor.
-

3. Creación del Socket y Captura del Mensaje

```
try (DatagramSocket socket = new DatagramSocket();  
    Scanner scanner = new Scanner(System.in)) {
```

- Se crea un **socket UDP** para enviar los paquetes.
- Se usa Scanner para capturar el mensaje del usuario.

```
System.out.println("Ingrese el mensaje a enviar:");  
String mensaje = scanner.nextLine();
```

- Se solicita al usuario ingresar un mensaje.
-

4. Conversión del Mensaje a Bytes y Cálculo de Fragmentos

```
byte[] datos = mensaje.getBytes();
int totalPaquetes = (int) Math.ceil((double) datos.length / MAX_BYTES);
```

- `mensaje.getBytes()`: Convierte el mensaje en un **array de bytes**.
 - `totalPaquetes`: Se calcula cuántos fragmentos se necesitan:
 - Si el mensaje tiene **30 bytes** y `MAX_BYTES = 10`, se dividirá en **3 fragmentos**.
-

5. Fragmentación y Envío de Paquetes UDP

```
for (int i = 0; i < totalPaquetes; i++) {
```

- Se inicia un ciclo que recorre cada fragmento del mensaje.

```
int inicio = i * MAX_BYTES;
int fin = Math.min(inicio + MAX_BYTES, datos.length);
byte[] fragmento = new byte[fin - inicio + 2]; // +2 para metadatos
```

- `inicio` y `fin`: Se determinan los **límites del fragmento** dentro del mensaje.
 - `fragmento`: **Nuevo array** con espacio extra para los **metadatos**.
-

6. Asignación de Metadatos

```
fragmento[0] = (byte) i; // Número de paquete
fragmento[1] = (byte) totalPaquetes; // Total de paquetes
System.arraycopy(datos, inicio, fragmento, 2, fin - inicio);
```

Cada fragmento incluye:

- `fragmento[0]`: **Número del fragmento** (0, 1, 2, ...).
 - `fragmento[1]`: **Número total de fragmentos**.
 - `fragmento[2]` en adelante: **Parte del mensaje original**.
-

7. Creación y Envío del Paquete UDP

```
DatagramPacket paquete = new DatagramPacket(fragmento, fragmento.length,
InetAddress.getByName(SERVER_IP), SERVER_PORT);
socket.send(paquete);
```

- Se crea un **DatagramPacket** con el fragmento, la IP del servidor y el puerto.
- Se **envía el paquete** al servidor con `socket.send(paquete)`.

```
System.out.println("Enviado: Paquete " + i + " de " + totalPaquetes + " - " +
new String(fragmento, 2, fragmento.length - 2));
```

- Se imprime el fragmento enviado para seguimiento.

ServidorEco.java

El **servidor** es el encargado de recibir los fragmentos del mensaje, almacenarlos, ordenarlos y reconstruir el mensaje original cuando haya recibido todos los fragmentos.

1. Librerías

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.util.Map;
import java.util.TreeMap;
```

El código usa tres librerías esenciales:

- `java.net.DatagramPacket`: Permite manejar paquetes UDP.
- `java.net.DatagramSocket`: Crea el socket para recibir los paquetes UDP.
- `java.util.TreeMap`: Almacena los fragmentos ordenados automáticamente.

2. Variables Globales

```
private static final int PUERTO = 1234;
private static final int MAX_BYTES = 12; // 1 byte de índice + 1 byte de
total + 10 bytes de datos
```

- `PUERTO`: Número del puerto en el que el servidor escuchará los paquetes.
- `MAX_BYTES`: Tamaño máximo de cada paquete (12 bytes), donde:
 - 1 byte es el **número de fragmento**.
 - 1 byte es el **total de fragmentos**.
 - 10 bytes son los **datos reales del mensaje**.

3. Estructura de Datos para Almacenar Fragmentos

```
Map<Integer, String> fragmentos = new TreeMap<>();
int totalEsperado = -1; // Inicializamos en -1 para esperar el primer
paquete
```

- `fragmentos`: **Estructura clave-valor (TreeMap)** que almacena los fragmentos. Se usa `TreeMap` porque **mantiene los fragmentos en orden automático**.
- `totalEsperado`: Variable que indica cuántos paquetes en total se deben recibir. Se inicializa en -1 hasta que llegue el primer fragmento.

4. Creación del Socket y Espera de Mensajes

```
try (DatagramSocket socket = new DatagramSocket(PUERTO)) {
    System.out.println("Servidor ECO iniciado en el puerto " + PUERTO +
"\n");
    System.out.println("Esperando mensaje del cliente...\n");
```

```
while (true) {
```

- Se **crea el socket UDP** en el puerto 1234.
 - Se entra en un **bucle infinito** para seguir recibiendo mensajes continuamente.
-

5. Recepción de un Paquete UDP

```
byte[] buffer = new byte[MAX_BYTES];  
DatagramPacket paquete = new DatagramPacket(buffer, buffer.length);  
socket.receive(paquete);
```

- Se crea un **buffer de 12 bytes** para recibir el paquete.
 - Se define un **DatagramPacket** con el buffer.
 - Se recibe un paquete UDP y se almacena en paquete.
-

6. Extracción de Metadatos del Paquete

```
int numPaquete = buffer[0]; // Número del paquete  
int totalPaquetes = buffer[1]; // Total de paquetes  
String contenido = new String(buffer, 2, paquete.getLength() - 2);
```

- buffer[0]: **Número de fragmento** dentro del mensaje.
 - buffer[1]: **Cantidad total de fragmentos** que componen el mensaje.
 - buffer[2] en adelante: Contiene **los datos del fragmento** convertidos a texto.
-

7. Almacenamiento del Fragmento

```
System.out.println("Recibido: Paquete " + numPaquete + " de " +  
totalPaquetes + " - " + contenido);  
fragmentos.put(numPaquete, contenido);
```

- Se imprime la información del paquete recibido.
 - Se **almacena en el TreeMap** con su número de fragmento como clave para mantener el orden.
-

8. Detección del Total de Paquetes Esperados

```
if (totalEsperado == -1) {  
    totalEsperado = totalPaquetes;  
}
```

- Solo en la primera recepción, se establece el total de paquetes esperados (totalEsperado), que indica cuántos fragmentos se deben recibir para completar el mensaje.
-

9. Reconstrucción del Mensaje

```
if (fragmentos.size() == totalEsperado) {
    StringBuilder mensajeCompleto = new StringBuilder();
    for (String parte : fragmentos.values()) {
        mensajeCompleto.append(parte);
    }

    System.out.println("\nMensaje reconstruido:\n" + mensajeCompleto +
"\n\n");
}
```

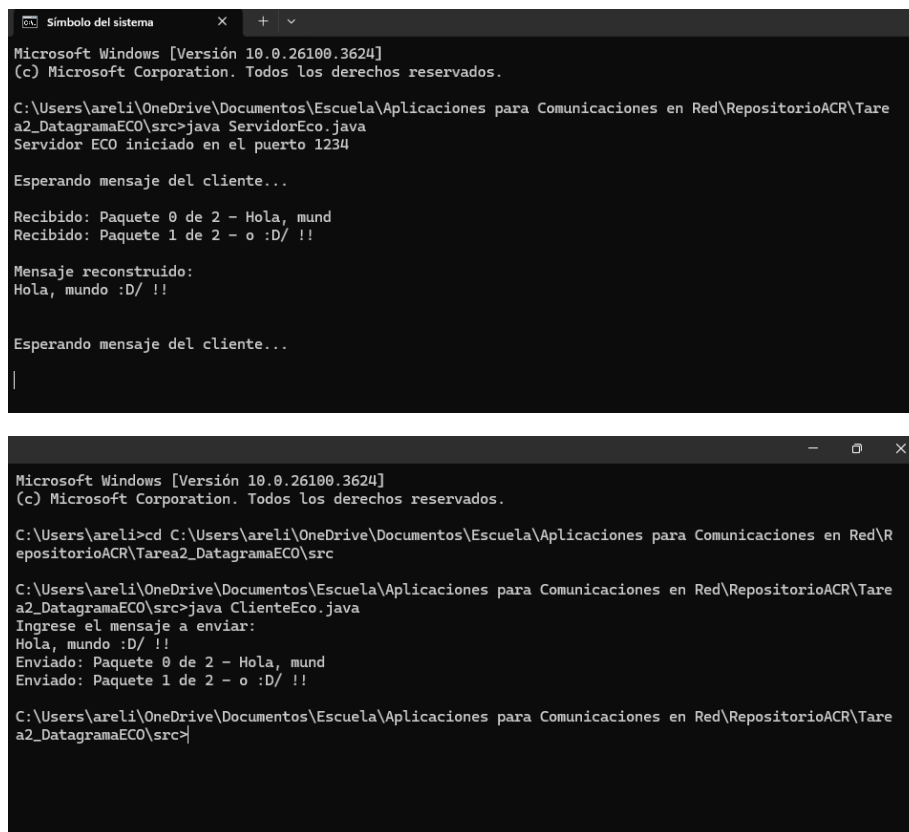
- Si ya se recibieron todos los paquetes esperados, se inicia la reconstrucción del mensaje.
- Se usa un **StringBuilder** para concatenar los fragmentos en orden.
- Se imprime el mensaje completo reconstruido.

10. Preparación para un Nuevo Mensaje

```
fragmentos.clear();
totalEsperado = -1; // Reiniciamos para el siguiente mensaje
System.out.println("Esperando mensaje del cliente...\n");
```

- Se limpia la memoria (**TreeMap**) para recibir un nuevo mensaje.
- totalEsperado se reinicia para esperar un nuevo conjunto de paquetes.

Ejecución



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.26100.3624]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\areli\OneDrive\Documentos\Escuela\Aplicaciones para Comunicaciones en Red\RepositorioACR\Tarea2_DatagramaECO\src>java ServidorEco.java
Servidor ECO iniciado en el puerto 1234

Esperando mensaje del cliente...

Recibido: Paquete 0 de 2 - Hola, mund
Recibido: Paquete 1 de 2 - o :D/ !!

Mensaje reconstruido:
Hola, mundo :D/ !!

Esperando mensaje del cliente...
|

Microsoft Windows [Versión 10.0.26100.3624]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\areli>cd C:\Users\areli\OneDrive\Documentos\Escuela\Aplicaciones para Comunicaciones en Red\RepositorioACR\Tarea2_DatagramaECO\src

C:\Users\areli\OneDrive\Documentos\Escuela\Aplicaciones para Comunicaciones en Red\RepositorioACR\Tarea2_DatagramaECO\src>java ClienteEco.java
Ingrese el mensaje a enviar:
Hola, mundo :D/ !!
Enviado: Paquete 0 de 2 - Hola, mund
Enviado: Paquete 1 de 2 - o :D/ !!

C:\Users\areli\OneDrive\Documentos\Escuela\Aplicaciones para Comunicaciones en Red\RepositorioACR\Tarea2_DatagramaECO\src>
```

Conclusión

A través de la ejecución de pruebas, se observó que los fragmentos pueden llegar desordenados debido a la naturaleza del protocolo UDP, que no garantiza la entrega ordenada de paquetes. Sin embargo, gracias a los metadatos de numeración, el servidor pudo reensamblar el mensaje correctamente.

Aprendizajes clave:

- Uso de DatagramSocket y DatagramPacket para enviar y recibir paquetes UDP.
- Implementación de fragmentación de mensajes y su manejo con estructuras de datos.
- Incorporación de metadatos para la correcta organización de paquetes.
- Identificación de desafíos en la comunicación UDP, como la posible pérdida o desorden de paquetes.

Este ejercicio refleja la importancia de la fragmentación y reensamblaje de paquetes en redes reales, conceptos fundamentales en el funcionamiento de protocolos como UDP e IP. Además, sienta las bases para aplicaciones más complejas donde la transmisión eficiente y ordenada de datos es crucial, como en transmisión de video, voz sobre IP (VoIP) o sistemas de mensajería.