Instituto Politécnico Nacional
IPN
Escuela Superior de Cómputo
ESCOM

Aplicaciones para comunicaciones en red

Grupo 6CM2 - Ciclo 2026A

Práctica 3

“Servicio de Chat”

Integrantes:

Alexandra Ortiz Villaseñor
Arely Amairani Cruz Rodríguez

Fecha de entrega: “ 11 - Noviembre - 2025”
Profesor: Axel Ernesto Moreno Cervantes

Marco de referencia y fundamentos técnicos

Multicast IPv4. La difusión a múltiples receptores se realiza con direcciones del bloque 224.0.0.0–239.255.255.255. Estos grupos están normalizados por IANA y su uso se recomienda en dominios controlados (p. ej., LAN) con valores de alcance acotado (TTL bajo). (IANA)

UDP como transporte. UDP es un protocolo sin conexión: no asegura entrega, orden ni control de congestión. Su especificación base es RFC 768; las guías de uso (RFC 8085) recomiendan evitar datagramas que requieran fragmentación IP, añadir control de congestión a nivel de aplicación cuando corresponda y diseñar tolerancia a pérdidas. (RFC Editor)

API de Java empleada. Para salas se usa `MulticastSocket` (unirse a grupos y enviar/recibir datagramas multicast); para control (JOIN/LEAVE) y privados (PM) se usa `DatagramSocket` en modo unicast. La documentación oficial describe su semántica (datagramas independientes que pueden llegar desordenados) y opciones disponibles. (Oracle Docs)

Procedimiento de despliegue y ejecución

Estructura de proyecto: código fuente en `src/main/java/chatapp/` con las clases: `Protocol`, `ChatServer`, `ChatClient`, `RoomSession`, `AudioUtil`.

Compilación (elegir una vía):

- **Maven (recomendada):**

```
mvn -q clean compile
```

- **javac (alternativa):**

```
mkdir -p out
javac -d out src/main/java/chatapp/*.java
```

Ejecución:

- **Servidor (una terminal):**
  - Maven: `java -cp target/classes chatapp.ChatServer 4446`
  - Javac : `java -cp out chatapp.ChatServer 4446`
- **Clientes (dos terminales):**
  - Maven:

        - `java -cp target/classes chatapp.ChatClient Ale 127.0.0.1 4446`
        - `java -cp target/classes chatapp.ChatClient Ara 127.0.0.1 4446`
    - Javac:
        - `java -cp out chatapp.ChatClient Ale 127.0.0.1 4446`
        - `java -cp out chatapp.ChatClient Ara 127.0.0.1 4446`

Objetivo y contexto

Se implementó un sistema de chat por **salas** utilizando **UDP** y **multicast** para la difusión de mensajes públicos. El **servidor** mantiene únicamente la **presencia** y difunde la **lista de usuarios activos por sala** cuando hay altas o bajas. Los **clientes** envían y reciben:

1. **Mensajes públicos** (texto, stickers e audio corto) por el grupo multicast de la sala.
2. **Mensajes privados** por **unicast UDP** directamente entre pares, aprendiendo la dirección y puerto privado del destinatario desde la `USER_LIST`. Este enfoque reduce la carga del servidor y limita su rol a directorio de presencia. (Oracle Docs)

Descripción de la solución y arquitectura

Modelo de salas. Cada sala se define por `(maddr, port)`; al ejecutar `/join`, el cliente crea una `RoomSession` que se suscribe al grupo multicast, escucha datagramas y permite enviar en ese dominio lógico. El servidor conserva `room → {usuario → ip:puertoPriv}` y, ante `JOIN/LEAVE`, emite `USER_LIST` por el canal multicast de la sala. (Oracle Docs)
Canales y responsabilidades.

- **Difusión pública:** `MulticastSocket` para tráfico de sala (texto, binarios ligeros codificados en Base64).
- **Privados (PM):** `DatagramSocket` por cliente; tras recibir `USER_LIST` con `usuario@ip:puertoPriv`, `/pm <usuario> <texto>` envía un datagrama unicast directo al destinatario. (Oracle Docs)
  Codificación y formato. El protocolo de aplicación usa **UTF-8** para texto/metadata (URL-encoding de campos); adjuntos pequeños se encapsulan en Base64.
  Límites de tamaño. Para minimizar pérdida por fragmentación IP, se restringen stickers/audio a archivos pequeños (≈30–50 KB). Este criterio está alineado con las **UDP Usage Guidelines** (sección de tamaños de mensaje). (IETF Datatracker)

Plan de pruebas y criterios de aceptación

A. Presencia y difusión

1. **Alta de sala:** en ambos clientes, `/join general 230.0.0.1 5000`.
   Esperado: recepción de `USER_LIST[general]` con los participantes y sus `ip:puertoPriv`.
2. **Baja y actualización:** un cliente ejecuta `/leave general`.
   Esperado: la `USER_LIST` de la sala se actualiza sin el usuario saliente.

B. Mensajería pública

1. **Texto y emojis:** `/msg general Hola a todos` .
   Criterio: el otro cliente ve el mensaje; se valida correcta decodificación UTF-8.
2. **Ráfaga corta:** 10 mensajes consecutivos; se acepta posible desorden (propio de UDP).

C. Adjuntos

1. **Sticker (≤50 KB):** `/sticker general "C:/…/peque.jpg"` .
   Criterio: recepción y guardado en `downloads/general/` del receptor; el archivo abre correctamente.
2. **Audio (WAV breve):** `/audio general "C:/…/clip.wav"` .
   Criterio: archivo recibido en `downloads/general/` y reproducción local si el sistema lo permite.

D. Privados (PM)

1. **Aprendizaje de destino:** confirmar que `USER_LIST` contiene `usuario@ip:puertoPriv`.
2. **Envío:** `/pm Ale Mensaje en privado` .
   Criterio: en Ale aparece `[PM] Ara: Mensaje en privado` .
   Negativo: `/pm UsuarioInexistente …` debe notificar que no se conoce su dirección.

E. Aislamiento por salas

- Un cliente se une a otra sala, p. ej., `/join juegos 230.0.0.2 5001` ; los mensajes allí **no** deben aparecer en `general` hasta que el otro cliente también se una.

F. Entorno

- Permitir el **firewall** en redes privadas al primer arranque.
- Ejecutar pruebas iniciales en la **misma máquina o LAN** para evitar interferencias de NAT/VPN.

> Justificación técnica: el uso de multicast y UDP determina aceptar cierta **tolerancia a pérdida** y ajustar el **tamaño de datagramas** a límites seguros, tal como recomiendan las guías del RFC 8085. (IETF Datatracker)

Conclusiones individuales

Cruz Rodríguez Arely Amairani
La práctica permitió implementar un chat de **difusión por salas** con **multicast** y un canal de **privados** por unicast, trasladando al cliente la mayor parte de la lógica de mensajería y dejando al servidor únicamente la **gestión de presencia**. La principal diferencia frente a diseños basados en sockets de flujo es la necesidad de definir un **protocolo de aplicación** explícito, manejar **codificación** y **tamaños de mensaje** con cuidado, y aceptar el modelo **no fiable** de UDP. El resultado fue un sistema funcional para laboratorio, con arquitectura clara, baja carga en servidor y una ruta evidente de mejora si se requirieran garantías más fuertes (confirmaciones a nivel aplicación o uso de un transporte fiable).

Ortiz Villaseñor Alexandra
El desarrollo reforzó el dominio de la **concurrencia del lado del cliente** mediante hilos dedicados por sala y para privados, coordinados con estructuras seguras para concurrencia. Se validó el **aislamiento lógico** entre salas (cada par dirección/puerto constituye un dominio de difusión) y el **ciclo de vida** correcto (join/leave y cierre ordenado). Desde la perspectiva de calidad, las pruebas mostraron que mantener adjuntos **ligeros** y diseñar **observabilidad** (registros de JOIN/LEAVE y listas de usuarios) acelera el diagnóstico. Para escenarios más exigentes, el patrón admite extensiones con **reintentos** y **confirmaciones** en la capa de aplicación, preservando el servidor ligero de presencia.

Referencias

- **RFC 768 — User Datagram Protocol (UDP)**. RFC Editor (1980). Consultado el 11-nov-2025. (RFC Editor)
- **RFC 8085 — UDP Usage Guidelines (BCP 145)**. IETF. Consultado el 11-nov-2025. (IETF Datatracker)

- **RFC 5771 — IANA Guidelines for IPv4 Multicast Address Assignments**. RFC Editor (2010). Consultado el 11-nov-2025. (RFC Editor)
- **IANA — IPv4 Multicast Address Space**. Consultado el 11-nov-2025. (IANA)
- **Oracle — `MulticastSocket` (Java SE 21 API)**. Consultado el 11-nov-2025. (Oracle Docs)
- **Oracle — `DatagramSocket` (Java SE API)**. Consultado el 11-nov-2025. (Oracle Docs)