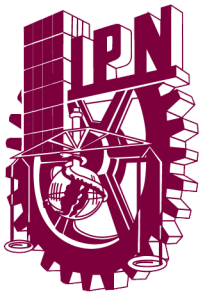


Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 3: Chat Multicast

Estudiantes:

Morales Hernández Carlos Jesús

Ramirez Hidalgo Marco Antonio

Grupo: 3CM16

Materia: Aplicaciones para comunicaciones en red

Docente: Axel Ernesto Moreno Cervantes

Carrera: Ingeniería en Sistemas Computacionales

Viernes, 26 de mayo del 2022

Índice

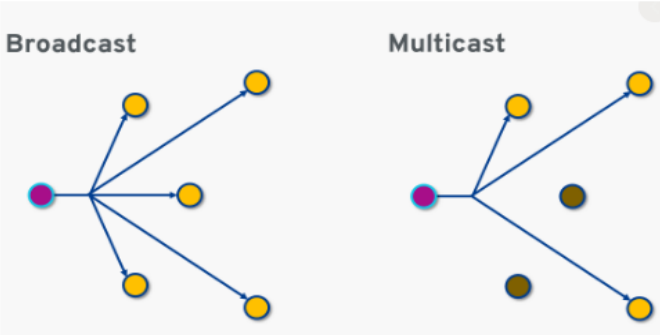
Práctica 3: Chat Multicas

1. Introducción

1.1 ¿Qué es IP multicast?

El protocolo de Internet es el protocolo estándar para la comunicación en redes informáticas. Tanto en Internet como en las redes de área local, el envío de mensajes electrónicos mediante el uso de direcciones IP, entre otros, forma parte fundamental del mundo de las redes modernas. La pila de protocolos TCP/IP de uso estándar proporciona un conjunto de protocolos y métodos que cubren un gran abanico de necesidades.

Aquí desempeñan un papel clave, por ejemplo, las formas de comunicación disponibles, de entre las cuales el el llamado multicasting el que está adquiriendo cada vez más importancia. Este concepto, también denominado multidifusión o difusión simple, permite la transmisión desde un punto a múltiples destinatarios. Este es el motivo por el que las conexiones de multidifusión también se conocen como conexión punto a multipunto.



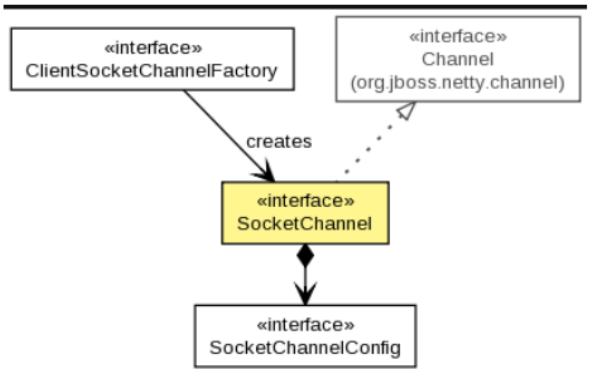
Gracias a su capacidad para enviar un único flujo de datos a múltiples destinos, multicast se distingue claramente de la transmisión unicast estándar, en la que los paquetes IP son entregados mediante una conexión directa entre sistemas que se comunican entre ellos. Aunque el multicast tiene puntos en común con la difusión amplia o broadcast, difiere en que los flujos de datos no se envían a todos los usuarios de la red, sino sólo a aquellos que han sido determinados previamente por parte del emisor y forman parte de un grupo multicast determinado.

1.2. Funcionamiento

El direccionamiento grupal antes mencionado es una de las propiedades clave de la multidifusión IP, ya que es fundamental para el funcionamiento de este concepto de comunicación. Además, existe la posibilidad de asignar un direccionamiento estático en el que, por ejemplo, se puede configurar una conexión a un servidor multicast para que este ofrezca el servicio correspondiente. Por otra parte, las direcciones multicast pueden también tener una asignación dinámica, ya que los grupos de multidifusión subyacentes no tienen porqué existir de forma permanente. Esto significa que es muy fácil crear grupos privados y también eliminarlos. Independientemente de si la asignación de direcciones ha sido realizada de forma estática o dinámica, el rango

de direcciones de las redes IP es de 224.0.0.0 a 239.255.255.255 (o bien FF00::/8), también conocido como espacio de direcciones clase D, que está reservado a tal fin.

De forma general, el acceso a los grupos IP multicast está regido por los routers de red involucrados y el protocolo de administración de grupos de Internet (IGMP). Para ello, el protocolo, que pertenece a la familia de protocolos de Internet, proporciona varios tipos de mensajes. Los servidores pueden utilizarlos para informar al router local sobre una solicitud de pertenencia al grupo y los routers, para recibir y reenviar las secuencias IP multicast que correspondan. El punto de inicio para la comunicación IGMP es siempre el router desde el que se conecta a la red el emisor del multicast. El emisor envía los paquetes de un flujo de datos una sola vez, indicando la dirección del grupo de multidifusión como dirección de destino y sin saber realmente a cuántas



terminales ha llegado.

2. Desarrollo experimental

Para lograr desarrollar esta práctica se utilizaron cuatro archivos principales que fueron nombrados *Cliente.java*, *Interfaz.java*, Mensaje.java y *Principals.java*. En seguida, se describen a detalle cada una de estas clases con sus funciones principales que hicieron posible la implementación del chat muticast.

2.1. Mensaje.java

En esta clase se engloba todo lo correspondiente al mensaje a enviar, donde se ponen los get y set para que se pueda seleccionar la opción de enviar mensajes, a que usuario se va a enviar el mensaje si es privado o público, entre otras funciones mas.

```java
import java.io.*;

public class Mensaje implements Serializable {
    private static final long serialVersionUID = 3L;
    public Mensaje(String mensaje, String usuarioOrigen, String usuarioDestino, int tipo) {
        this.mensaje = mensaje;
        this.usuarioOrigen = usuarioOrigen;
        this.usuarioDestino = usuarioDestino;
        this.tipo = tipo;
    }

    public Mensaje(String nombreArchivo, String usuarioOrigen, String usuarioDestino, int tipo, long tamanio, String ruta, int np) {
        this.nombreArchivo = nombreArchivo;
        this.usuarioOrigen = usuarioOrigen;
        this.usuarioDestino = usuarioDestino;
        this.tamanio = tamanio;
        this.ruta = ruta;
        this.np = np;
        this.tipo = tipo;
    }

    public String getMensaje() { return mensaje; }
    public String getUsuarioOrigen() { return usuarioOrigen; }
    public String getUsuarioDestino() { return usuarioDestino; }
    public int getTipo() { return tipo; }

    public void setMensaje(String mensaje) { this.mensaje = mensaje; }
    public void setUsuarioOrigen(String usuarioOrigen) { this.usuarioOrigen = usuarioOrigen; }
    public void setUsuarioDestino(String usuarioDestino) { this.usuarioDestino = usuarioDestino; }
    public void setTipo(int tipo) { this.tipo = tipo; }
```

2.2. Interfaz.java

En este archivo se almacenan cada una de las funciones relacionadas a la interfaz del programa, los paneles, los actionlistener de cada botón, la clase para poder mostrar los emojis en el JEditorPane, entre otras mas

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.net.URL;
import java.util.*;

public class Interfaz extends JFrame {
    private static final long serialVersionUID = 2L;
    private static ImageCache image_cache;
    Emoji emoji =  new Emoji();

    public Interfaz(String host, int puerto, String nombre) {
        // -----------------Recibiendo Parametros-----------------//
        this.host = host;
        this.puerto = puerto;
        this.nombre = nombre;
        // ----------------Creando Interfaz-------------------------//
        setBounds(x: 325, y: 100, width: 800, height: 500);
        setTitle("Practica 3: " + nombre);
        setResizable(resizable: false);

        panelPrincipal = new JPanel();
        panelCentral = new JPanel();
        panelInferior = new JPanel();
        panelEmojis = new JPanel();
        panelFunciones = new JPanel();
        panelUsuarios = new JPanel();
        panelCombo = new JPanel();
        editor = new JEditorPane(type: "text/html", text: null);
        editor.setEditable(b: false);
        areaMensaje = new JTextArea();
```

2.3. Principal.java

En este documento esta almacenado un pequeño JPanel para que el usuario pueda introducir su nombre y llamar al objeto cliente para crearle su objeto, concetarse al host "230.1.1.1" y puerto “9000” y comenzar con el funcionamiento del chat.

```java
import javax.swing.*;
import java.awt.BorderLayout;
import java.awt.Image;

public class Principal extends JFrame {
    private static final long serialVersionUID = 1L;

    public Principal() {
        setBounds(x: 450, y: 150, width: 250, height: 160);
        setResizable(resizable: false);
        setLocationRelativeTo(c: null);
        setTitle(title: "Practica 3");

        panelPrincipal = new JPanel();
        panelCentral = new JPanel();
        panelPrincipal.setLayout(new BorderLayout(hgap: 5, vgap: 5));
        panelCentral.setLayout(new BoxLayout(this.panelCentral, BoxLayout.Y_AXIS));
        nombreUsuario = new JLabel(text: "Introduce tu nombre de usuario:");
        campoUsuario = new JTextField(columns: 30);
        botonConectar = new JButton(text: "Conectar");


        botonConectar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                if (!campoUsuario.getText().equals(anObject: "")) {
                    setVisible(b: false);
                    new Interfaz(HOST, PUERTO, campoUsuario.getText().trim());
                } else
                    JOptionPane.showMessageDialog(Principal.this, message: "Nombre de usuario Vacio", title: "Error", JOptionPane.ERROR_MES
            }
        });
```

2.4. Cliente.java

Este archivo contiende las funciones para escuchar los mensajes y dar orden para enviar mensajes, desconectar al usuario, ver las opciones si el cliente quiere hablar con alguien en privado o público, enviar emojis, y otras opciones que se detallan a continuación.

2.4.1. class EscuchaMensajes

Código

```java
private class EscuchaMensajes implements Runnable {
    public void run() {
        System.out.println(x: "Escuchando Mensajes");
            try {
                DatagramPacket recibido = new DatagramPacket(new byte[6500], length: 6500);
                while (true) {
                    cliente.receive(recibido);
                    ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(recibido.getData()));
                    Mensaje msj = (Mensaje) ois.readObject();
                    if(msj.getTipo() == 0 && !msj.getUsuarioOrigen().equals(nombre)) {
                        byte[] bmsj = msj.getMensaje().getBytes();
                        byte[] busuario = msj.getUsuarioOrigen().getBytes();
                        String mensaje = new String(bmsj, offset: 0, msj.getMensaje().length());
                        String usuario = new String(busuario, offset: 0, msj.getUsuarioOrigen().length());
                        HTMLEditorKit kit = (HTMLEditorKit) editor.getEditorKit();
                        StringReader reader = new StringReader(mensaje);
                        kit.read(reader, editor.getDocument(), editor.getDocument().getLength());
                        usuarioConectado.addItem(usuario);
                        ByteArrayOutputStream baos = new ByteArrayOutputStream();
                        ObjectOutputStream oos = new ObjectOutputStream(baos);
                        msj.setTipo(tipo: 3);
                        msj.setUsuarioDestino(msj.getUsuarioOrigen());
                        msj.setUsuarioOrigen(nombre);
                        oos.writeObject(msj);
                        oos.flush();
                        byte[] b = baos.toByteArray();
                        DatagramPacket re = new DatagramPacket(b, b.length, grupo, puerto);
                        cliente.send(re);
                    } else if(msj.getTipo() == 1) {
```

Descripción

Mediante esta clase con ayuda de un hilo, se empieza a recibir los OIS que se han recibido, para saber que tipo de operación hacer, dependiendo del tipo, podemos identificar si el usuario se quiere desconectar del chat, desea enviar un mensaje privado o público, poner un emoji o enviar un archivo.

2.4.2. public class EnviaMensaje

Código

```
private class EnviaMensajes implements Runnable {
    private EnviaMensajes(Mensaje msj) {
        this.msj = msj;
    }

    public void run() {
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(baos);
            oos.writeObject(msj);
            oos.flush();
            byte[] msj = baos.toByteArray();
            DatagramPacket p = new DatagramPacket(msj, msj.length, grupo, puerto);
            cliente.send(p);
            oos.close();
            baos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private Mensaje msj;
}
```

Descripción

Esta clase nos permite crear un BAOS, OOS y un arreglo de byes para poder enviar dicho mensaje mediate el DatagramPaket por medio de un hilo.

2.4.3. public class EnvioArchivos

Código

```java
private class EnvioArchivos implements Runnable {
    private EnvioArchivos(File file, String dest) {
        this.file = file;
        this.destino = dest;
    }

    public void run() {
        try {
            DataInputStream dis = new DataInputStream(new FileInputStream(file));
            long tamanio = dis.available();
            long enviado = 0;
            int n = 0;
            int i = 0;
            while (enviado < tamanio) {
                Mensaje datos = new Mensaje(file.getName(), nombre, destino, tipo: 2, file.length(), ruta: "", ++i);
                ByteArrayOutputStream baos = new ByteArrayOutputStream(size: 6400);
                ObjectOutputStream oos = new ObjectOutputStream(new BufferedOutputStream(baos));
                oos.flush();
                byte[] b = new byte[4000];
                n = dis.read(b);
                byte[] b2 = new byte[n];
                System.arraycopy(b, srcPos: 0, b2, destPos: 0, n);
                datos.setDatos(b2);
                datos.setBytesEnviados(n);
                oos.writeObject(datos);
                oos.flush();
                byte[] d = baos.toByteArray();
                DatagramPacket paqueteEnvio = new DatagramPacket(d, d.length, grupo, puerto);
                cliente.send(paqueteEnvio);
                try {
```

Descripción

Este código es parecido al utilizado en la practica 1 para poder enviar archivos a los clientes disponibles, lo único que cambia fue agregar el DatagramPacket para hacer el envio.

2.4.4. Función recibirArchivo

Código

```java
private void recibirArchivo(Mensaje datos) {
    try{
            System.out.println("Numero de paquete: " + datos.getNp());
            if(datos.getNp() == 0) {
                dos = new DataOutputStream(new FileOutputStream("./" + nombre + "/" + nombreArchivo));
                for(int i = 0; i < lista.size(); i++) {
                    dos.write(lista.get(i));
                }
                dos.close();
                lista.clear();
            } else if(datos.getNp() == 1) {
                lista = new ArrayList<>();
                nombreArchivo = datos.getNombre();
                lista.add(datos.getDatos());
            } else {
                if(nombreArchivo.equals(datos.getNombre()))
                    lista.add(datos.getDatos());
            }
            Thread.sleep(millis: 500);
    }catch(Exception e) {
        e.printStackTrace();
    }

}
```

Descripción

Para recibir archivos, primero recibimos el numero de paquetes con base en la variable datos de nuestro objeto Mensaje, una vez hecho eso, escribimos los datos provenientes de nuestro emisor para simplemente ponerlo dentro de la carpeta del usuario que selecciono el usuario.

3. Conclusiones

En esta práctica se trabajaron con los sockets de datagrama para implementar el chat, ademas de utilizar funciones referentes a su tipo, de esa forma se podría saber si el cliente quería enviar un mensaje a alguna persona en especificio, al todos, enviar un archivo o un emoji.

Para la implementación de los emojis, fue necesario el uso de HTML, haciendo unas pequeñas configuraciones al JEditorPane para que trabajara como código de HTML, de esa forma con un simple código de “img” se pudo mandar ese string mediante los sockets de datagrama y poder mostrar dicha imagen en pantalla.

Para el envió de archivos, utilizamos lo visto en la practica 1, con el único cambio de usar sockets de datagrama.

Por ultimo, los sockets de multidifusión permiten tener un mejor control y fluidez en una arquitectura cliente-servidor real. La implementación de dos tipos de chats, uno grupal y otro individual, permitió entender un poco mejor la importancia del envío multicast para los mensajes individuales a través del servidor.

De esta manera, todos los clientes que se fueron conectando al servidor y a su vez a una sala general, fueron manejados por los sockets que fueron recibiendo y mandando flujos de información al servidor y a los clientes mismos. Así, se cumple con el objetivo de lograr tener este tipo de comunicación a nivel masivo e individual

Bibliografía

[1] Programacion, *Sockets en java UDP y TCP*. [En línea]. Disponible en: https://www.programacion.com.py/escritorio/java-escritorio/sockets-en-java-udp-y-tcp

[2] IONOS. Sockets Multicas (2022, marzo 12).. [En línea]. Disponible en: https://www.ionos.mx/digitalguide/servidores/know-how/multicast/#:~:text=De%20forma%20general%2C%20el%20acceso,proporciona%20varios%20tipos%20de%20mensajes.