Instituto Politécnico Nacional
Escuela Superior de Cómputo
Ingeniería en Sistemas Computacionales

Aplicaciones para Comunicaciones en Red

Sockets de datagrama

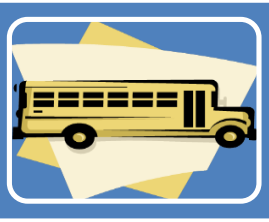
M. en C. Sandra Ivette Bautista Rosales

2.1 DATAGRAMAS EN EL DOMINO DE INTERNET

Arquitectura TCP/IP

Aplicación

- HTTP, FTP, TFTP, etc.

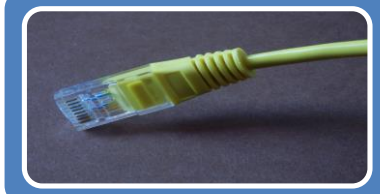


Transporte

- TCP
- UDP

Internet

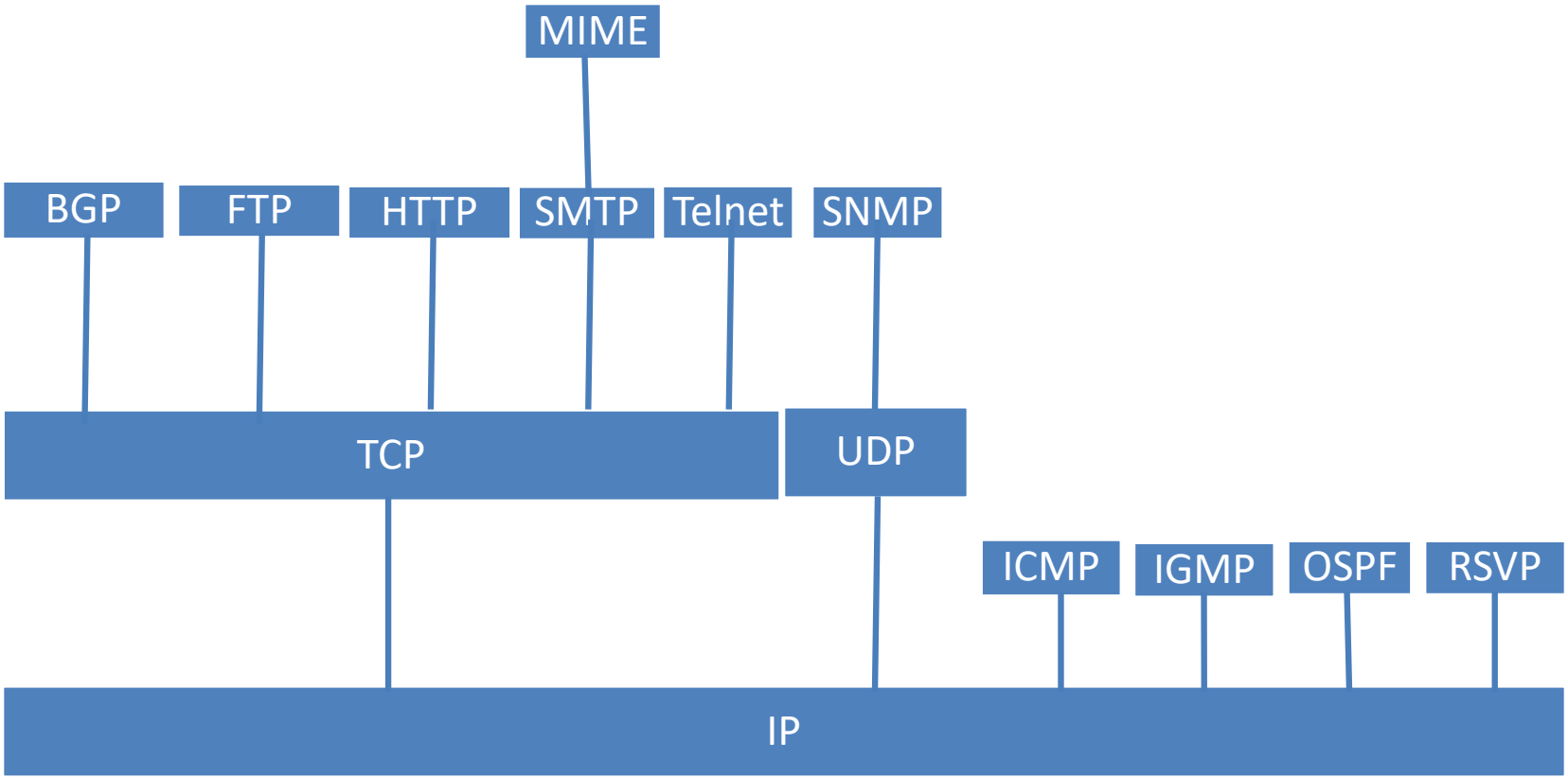
- IP
- IGMP



Acceso a la red

- LLC
- MAC

TCP y UDP


MIME
BGP
FTP
HTTP
SMTP
Telnet
SNMP
TCP
UDP
ICMP
IGMP
OSPF
RSVP
IP

UDP (Datagramas)

- *UDP* es un protocolo no orientado a conexión de la capa de transporte que es una reflexión directa de los servicios de datagramas de IP.
- Excepto que UDP proporciona un modo de pasar la parte del mensaje de UDP al protocolo de la capa de aplicación (multiplexación).

Características de UDP (1/2)

- No orientados a conexión
  - Los mensajes de UDP se envían sin la negociación del establecimiento de conexión de TCP
- No fiable
  - Los mensajes de UDP se envían como datagramas sin secuencia y sin reconocimiento.
  - El protocolo de aplicación que utiliza los servicios de UDP debe recuperarse de la perdida de mensajes.
  - Los protocolos típicos de nivel de aplicación que utilizan los servicios de UDP, proporcionan sus propios servicios de fiabilidad o retransmiten periódicamente los mensajes de UDP o tras un periodo de tiempo preestablecido.

Características de UDP (2/2)

- Proporciona identificación de los protocolos de nivel de aplicación
  - UDP proporciona un mecanismo para enviar mensajes a un protocolo o proceso del nivel de aplicación en un host de una red.
  - La cabecera UDP proporciona identificación tanto del proceso origen como del proceso destino

Qué no ofrece UDP (1/2)

- Buffer
  - UDP no proporciona ningún tipo de buffer de los datos de entrada, ni de salida.
  - Es el protocolo de nivel de aplicación quien debe proveer todo el mecanismo de buffer.
- Segmentación
  - UDP no proporciona ningún tipo de segmentación de grandes bloques de datos.
  - Por lo tanto la aplicación debe de enviar los datos en bloques suficientemente pequeños para que tanto los datagramas de IP como los mensajes de UDP, no sean mayores que la MTU de la tecnología de Nivel de Interfaz de Red por la que se envían.
  - El tamaño estándar de datos (carga útil) de UDP es de 512 bytes.

Qué no ofrece UDP (2/2)

- Control de flujo
  - UDP no proporciona control de flujo ni del extremo emisor, ni del extremo receptor.
  - Los emisores de mensajes UDP pueden reaccionar a la recepción de los mensajes Control de flujo de origen de ICMP, pero no se requiere.

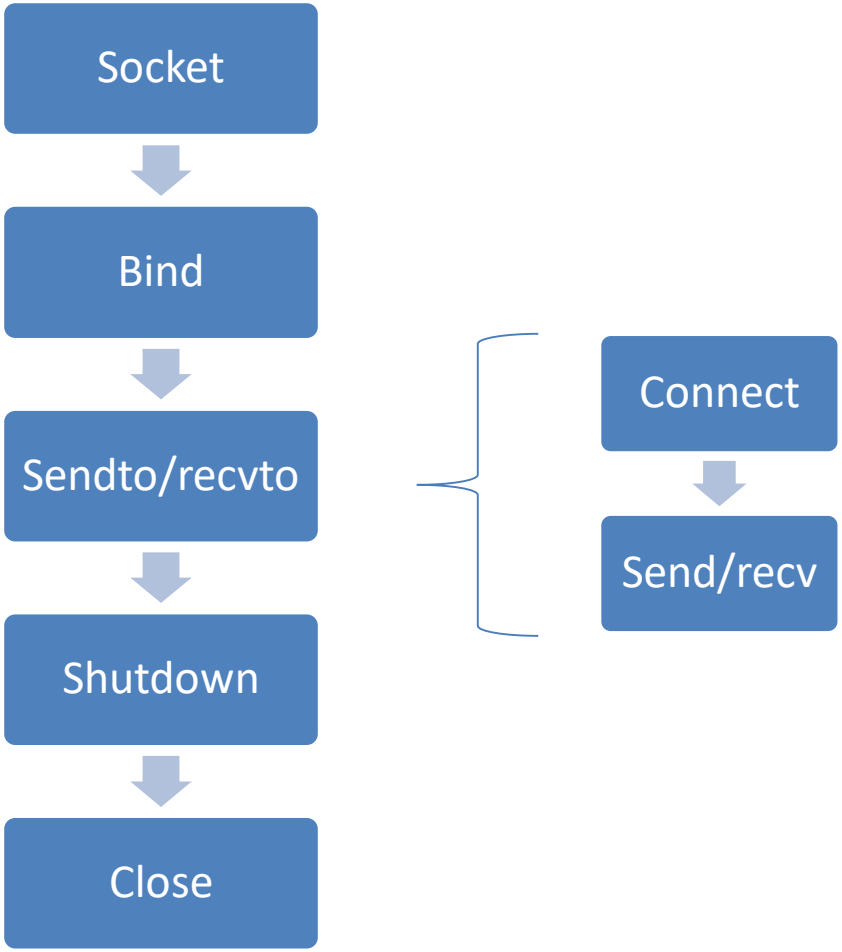
Usos de UDP (1/2)

- Protocolos ligeros
  - Para conservar recursos de memoria y procesador, algunos protocolos del Nivel de aplicación requieren del uso de un protocolo ligero que realice una función concreta mediante un simple intercambio de mensajes.
  - Un buen ejemplo de protocolo ligero es la petición de nombres de DNS
- El protocolo de Nivel de aplicación proporciona fiabilidad
  - Si el protocolo de Nivel de aplicación proporciona su propio servicio de transferencia fiable de datos, no se necesita un servicio fiable como TCP.
  - Ejemplo de protocolos de Nivel de Aplicación fiable son:
    TFTP (Trivial File Transfer Protocol) y NFS (Network File System)

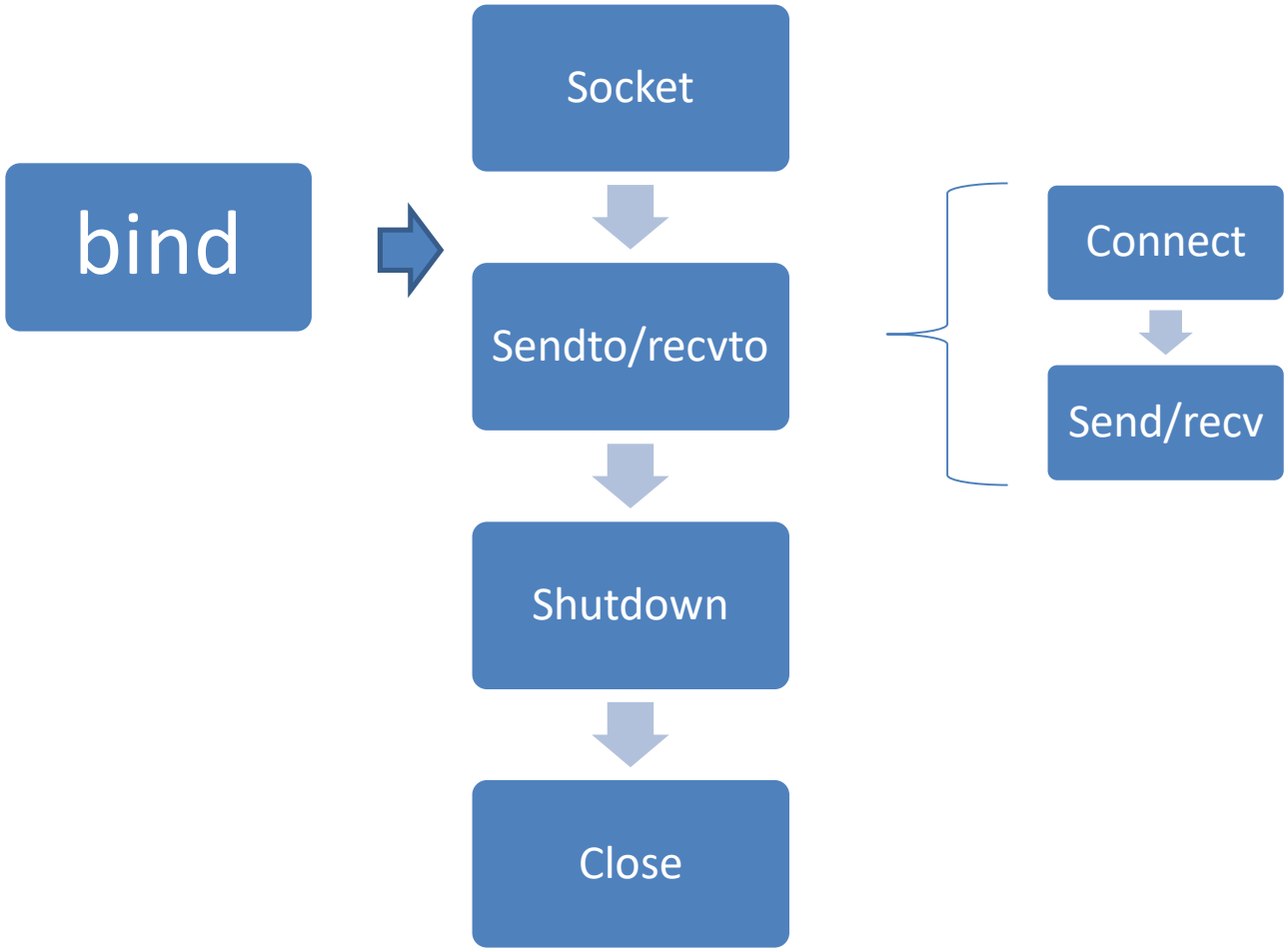
Usos de UDP (2/2)

- No se requiere fiabilidad por un proceso periódico de anuncios
  - Si el protocolo de Nivel de aplicación publica periódicamente la información, no se requiere un envío fiable.
  - Si se pierde un mensaje, se vuelve a anunciar de nuevo tras el periodo de publicación.
  - Un ejemplo de protocolo de Nivel de aplicación que usa anuncios periódicos (30 segundos) es el Protocolo de Información de Enrutamiento (RIP – Routing Information Protocol).
- Envío de uno a muchos
  - UDP se utiliza como protocolo de Nivel de transporte siempre que se debe enviar datos de Nivel de aplicación a múltiples destinos mediante direcciones de IP de difusión o multidifusión.
  - A diferencia de TCP, que se puede usar sólo en envío de uno a uno.
  - Ejemplo: Un envío de señal de video o voz a través de la red de paquetes.

Servidor


Socket
Bind
Sendto/recvto
Shutdown
Close
Connect
Send/recv

Cliente


bind
Socket
Sendto/recvto
Shutdown
Close
Connect
Send/recv

Clases

- DatagramPacket
  - Representa un paquete no orientado a conexión
- DatagramSocket
  - Es el socket no orientado a conexión

Constructores DatagramPacket

DatagramPacket(byte[] buf, int length) Crea DatagramPacket para recibir paquetes de longitud length.

DatagramPacket(byte[] buf, int length, InetAddress address, int port) Crea DatagramPacket para enviar paquetes de longitud length, a una dirección y puerto específico.

DatagramPacket(byte[] buf, int offset, int length) Crea DatagramPacket para recibir paquetes de longitud length, especificando un corrimiento en el buffer.

DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port) Crea DatagramPacket para enviar paquetes de longitud length, especificando un corrimiento en el buffer, una dirección y un número de puerto

DatagramPacket(byte[] buf, int offset, int length, SocketAddress address)) Crea DatagramPacket para enviar paquetes de longitud length, especificando un corrimiento en el buffer, una dirección y un número de puerto

DatagramPacket(byte[] buf, int length, SocketAddress address)) Crea DatagramPacket para enviar paquetes de longitud length, a una dirección y puerto específico.

Métodos DatagramPacket

- getAddres()
- getAddres(InetAddres d)
- getPort()
- getPort(int p)
- getData()
- getData(byte b)

Constructores DatagramSocket

DatagramSocket() Crea un socket de datagrama y lo liga a algún puerto de la computadora local

DatagramSocket(DatagramSocketImpl impl) Crea un socket no relacionado a ningún puerto con las especificaciones del DatagramSocketImpl.

DatagramSocket(int port) Crea un socket de datagrama y lo liga a un puerto específico.

DatagramSocket(int port, InetAddress laddr) Crea un socket de datagrama y lo liga a un puerto específico y a una dirección IP específica

DatagramSocket(SocketAddress bindaddr) Crea un socket de datagrama y lo liga al SocketAddress específico.

Servidor echo con DatagramSocket

```
import java.net.*;
import java.io.*;

public class SHolaD {
   public static void main(String[] args){
      try{
         DatagramSocket s = new DatagramSocket(2000);
         System.out.println("Servidor iniciado, esperando cliente");
         for(;;){
            DatagramPacket p = new DatagramPacket(new byte[2000],2000);
            s.receive(p);
            System.out.println("Datagrama recibido desde"+p.getAddress()+":"+p.getPort());
            String msj = new String(p.getData(),0,p.getLength());
            System.out.println("Con el mensaje:"+ msj);
         }//for
         //s.close()
      }catch(Exception e){
         e.printStackTrace();
      }//catch
   }//main
}
```

Cliente echo con datagramSocket

```
import java.net.*;
import java.io.*;

public class CHolaD {
   public static void main(String[] args){
      try{
         DatagramSocket cl = new DatagramSocket();
         System.out.print("Cliente iniciado, escriba un mensaje de saludo:");
         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
         String mensaje = br.readLine();
         byte[] b = mensaje.getBytes();
         String dst = "127.0.0.1";
         int pto = 2000;
         DatagramPacket p = new DatagramPacket(b,b.length,InetAddress.getByName(dst),pto);
         cl.send(p);
         cl.close();
      }catch(Exception e){
         e.printStackTrace();
      }//catch
   }//main
}
```

Ejercicio

- Modificar el programa para que envié y reciba mensajes entre el cliente y el servidor.
- Considerar los mensajes muy grandes, es decir, si los datos a enviar son más grandes que el tamaño estándar de un datagrama, entonces se deben de enviar en varios paquetes.
- En el código, deben agregar comentarios sobre las instrucciones del programa.

Datos primitivos, servidor

```
import java.net.*;
import java.io.*;

public class SPrimD {
   public static void main(String args[]){
      try{
         DatagramSocket s = new DatagramSocket(2000);
         System.out.println("Servidor iniciado, esperando cliente");
         for(;;){
            DatagramPacket p = new DatagramPacket(new byte[2000],2000);
            s.receive(p);
            System.out.println("Datagrama recibido desde"+p.getAddress()+":"+p.getPort());
            DataInputStream dis = new DataInputStream(new ByteArrayInputStream(p.getData()));
            int x = dis.readInt();
            float f = dis.readFloat();
            long z = dis.readLong();
            System.out.println("\n\nEntero:"+ x + " Flotante:"+f+" Entero largo:"+z);
         }//for
         //s.close()
      }catch(Exception e){
         e.printStackTrace();
      }//catch
   }//main
}
```

Datos primitivos, cliente

```
import java.net.*;
import java.io.*;
public class CPrimD {
   public static void main(String args []){
      try{
         int pto = 2000;
         InetAddress dst = InetAddress.getByName("127.0.0.1");
         DatagramSocket cl = new DatagramSocket();
         ByteArrayOutputStream baos = new ByteArrayOutputStream();
         DataOutputStream dos = new DataOutputStream(baos);
         dos.writeInt(4);
         dos.flush();
         dos.writeFloat(4.1f);
         dos.flush();
         dos.writeLong(72);
         dos.flush();
         byte[] b = baos.toByteArray();
         DatagramPacket p = new DatagramPacket(b,b.length,dst,pto);
         cl.send(p);
         cl.close();
      }catch(Exception e){
         e.printStackTrace();
      }//catch
   }//main
}
```

Clase DatagramChannel

- Canal seleccionable para sockets orientados a datagrama
- Constructor:
  - DatagramChannel(SelectorProvider provider);
- Este constructor requiere definir un socket con características especiales (como una capa de encriptación) por lo que suele usar el método estático open() para usarlo con sockets genéricos
  - DatagramChannel dch = DatagramChannel.open();

Algunos métodos de la clase DatagramChannel

- bind(SocketAddress local); liga el canal a una dirección local
- connect(SocketAddress remote); conecta el canal
- getLocalAddress(); nos regresa la dirección del canal al que esta conectado de forma local
- getRemoteAddress(); regresa la dirección remota a la que esta conectado
- isConnected(); nos dice si un canal esta conectado o no
- socket(); nos regresa el socket al que esta asociado el canal.

Opciones de DatagramChannel

- Las opciones se modifican usando el método setOption()

Opción	Descripción
SO_SNDBUF	Tamaño del buffer de salida
SO_RCVBUF	Tamaño del buffer de entrada
SO_REUSEADDR	Vuelve reutilizable el socket
SO_BROADCAST	Permite la transmisión de datagramas de broadcast
IP_TOS	Define el tipo de servicio (ToS) en la cabecera IP
IP_MULTICAST_IF	La interfaz de red para datagramas multicas de IP
IP_MULTICAST_TTL	El tiempo de vida para datagramas multicas (TTL)
IP_MULTICAST_LOOP	Loopback para los datagramas multicas de IP

Ejemplo: UDPServidor.java

```
import java.net.*;
import java.io.*;
import java.nio.*;
import java.nio.channels.*;
import java.util.*;

public class UDPServidor{
public final static int PUERTO = 7;
public final static int TAM_MAXIMO = 65507;

public static void main(String[] args) {
   int port = PUERTO;
   try{
      DatagramChannel canal = DatagramChannel.open();
      canal.configureBlocking(false);
      DatagramSocket socket = canal.socket();
      SocketAddress dir = new InetSocketAddress(port);
      socket.bind(dir);
      Selector selector = Selector.open();
      canal.register(selector,SelectionKey.OP_READ);
      ByteBuffer buffer =
ByteBuffer.allocateDirect(TAM_MAXIMO);
while(true){
        selector.select(5000);
        Set sk = selector.selectedKeys();
        Iterator it = sk.iterator();
        while(it.hasNext()){
          SelectionKey key = (SelectionKey)it.next();
          it.remove();
          if(key.isReadable()){
            buffer.clear();
            SocketAddress client = canal.receive(buffer);
            buffer.flip();
            int eco = buffer.getInt();
            if(eco==1000){
              canal.close();
              System.exit(0);
            }else{
              System.out.println("Dato leido: "+eco);
              buffer.flip();
              canal.send(buffer,client);
            }//else
          }//if
        }//while2
      }//while
   }catch(IOException e){
      System.err.println(e);
   }//catch
}//main
}//class
```

Ejemplo: UDPCliente.java

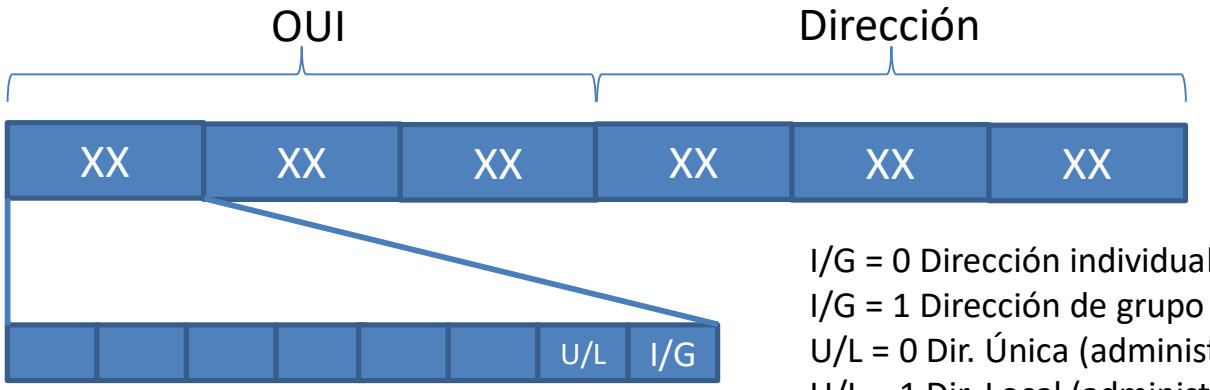
```
import java.net.*;
import java.io.*;
import java.nio.*;
import java.nio.channels.*;
import java.util.*;

public class UDPCliente{
public final static int PUERTO = 7;
private final static int LIMITE = 100;
public static void main(String[] args) {
   boolean bandera=false;
   SocketAddress remoto =
                new InetSocketAddress("127.0.0.1",PUERTO);
   try{
      DatagramChannel canal = DatagramChannel.open();
      canal.configureBlocking(false);
      canal.connect(remoto);
      Selector selector = Selector.open();
      canal.register(selector,SelectionKey.OP_WRITE);
      ByteBuffer buffer = ByteBuffer.allocateDirect(4);
      int n = 0;
      while(true){
         selector.select(5000); //espera 5 segundos por la conexión
         Set sk = selector.selectedKeys();
         if(sk.isEmpty() && n == LIMITE || bandera){
            canal.close();
            break;
         }else{
            Iterator it = sk.iterator();
            while(it.hasNext()){
               SelectionKey key = (SelectionKey)it.next();
               it.remove();
               if(key.isWritable()){
                  buffer.clear();
                  buffer.putInt(n);
                  buffer.flip();
                  canal.write(buffer);
                  System.out.println("Escribiendo el dato: "+n);
                  n++;
                  if(n==LIMITE){
                     //todos los paquetes han sido escritos;
                     buffer.clear();
                     buffer.putInt(1000);
                     buffer.flip();
                     canal.write(buffer);
                     bandera = true;
                     key.interestOps(SelectionKey.OP_READ);
                     break;
                  }//if
               }//if
            }//while
         }//else
      }//while
   }catch(Exception e){
      System.err.println(e);
   }//catch
}//main
}//class
```

2.2 DATAGRAMAS DE MULTIDIFUSIÓN

2.2.1 DIRECCIONES DE MULTIDIFUSIÓN

Direcciones multicast en Ethernet



I/G = 0 Dirección individual (unicast)
I/G = 1 Dirección de grupo (multi./broad)
U/L = 0 Dir. Única (administrada globalmente IEEE)
U/L = 1 Dir. Local (administrada localmente)

- En Ethernet los bits dentro de cada byte se representan en orden inverso. Por tanto el bit I/G es el último del primer byte.
- Regla:

  En ethernet una dirección es multicast si y solo si el segundo dígito hexadecimal es impar

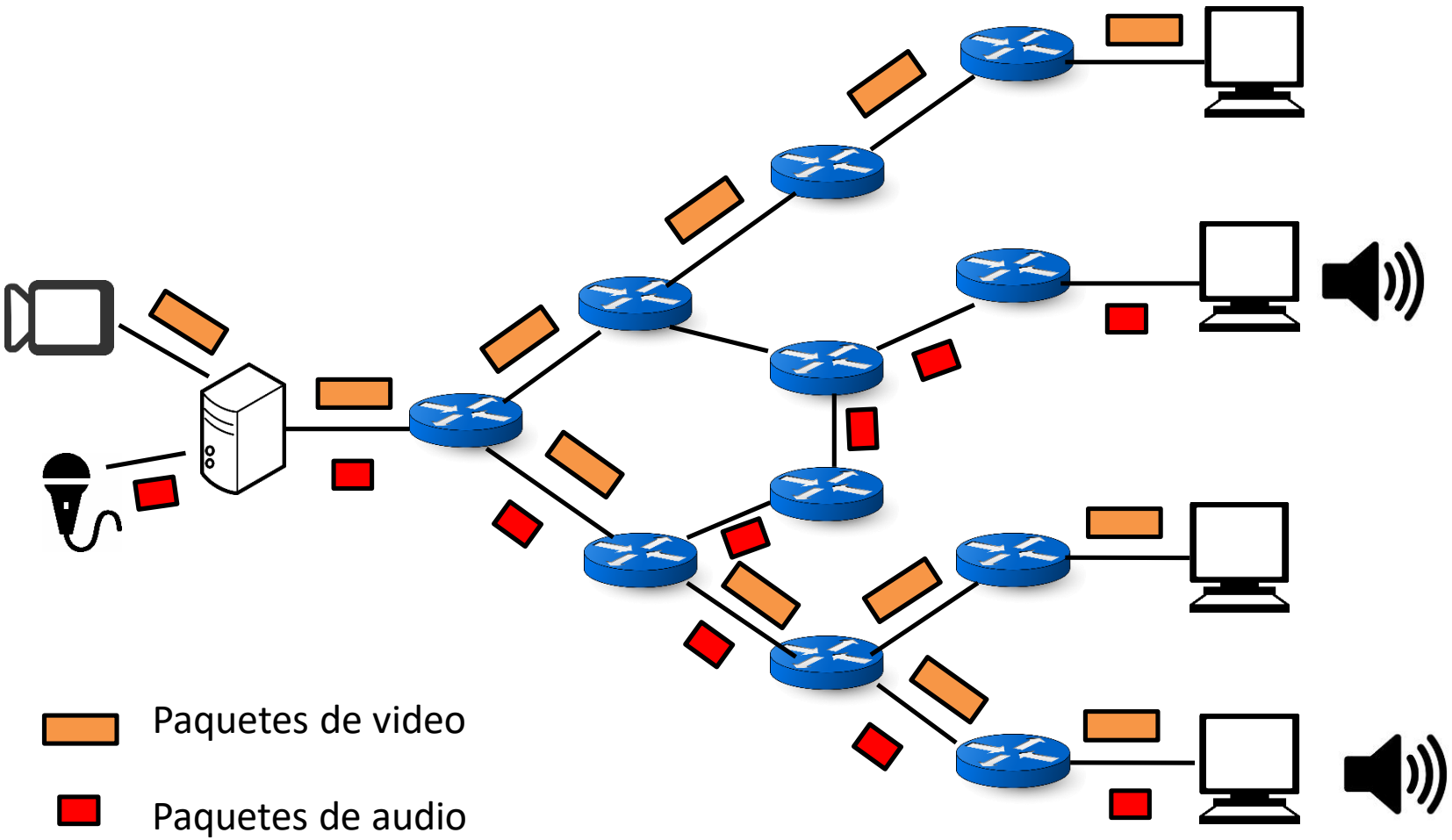
  Ej.: la dirección AB:00:03:00:00:00 es multicast

Direcciónes multicast

- Las direcciones multicast tienen estructura plana (no jerárquica)
- Las direcciones multicast solo pueden aparecer como direcciones de destino, nunca de origen

Multicast, clase D, rango 224.0.0.0 al 239.255.255.255

Emisión de un grupo multicast en una WAN



Direcciones Multicast en IP

- Las direcciones multicast tienen estructura plana (no jerárquica)
- Las direcciones multicast solo pueden aparecer como direcciones de destino, nunca de origen
- No pueden aparecer en los campos opcionales source route o record route
- ICMP y multicast:
  - Los datagramas multicast no pueden dar lugar a mensajes ICMP DESTINATION UNREACHABLE
  - Tampoco pueden dar lugar a mensajes ICMP TIME EXCEEDED. Sin embargo el TTL se decrementa normalmente y cuando vale cero el datagrama se destruye
  - Los mensajes multicast ICMP ECHO REQUEST generan respuestas unicast de todos los miembros del grupo. Las respuestas, unicast, llevan como dirección de origen la del emisor y destino la del host que envió el ICMP multicast.

Algunas direcciones IPv4 multicast reservadas

Locales

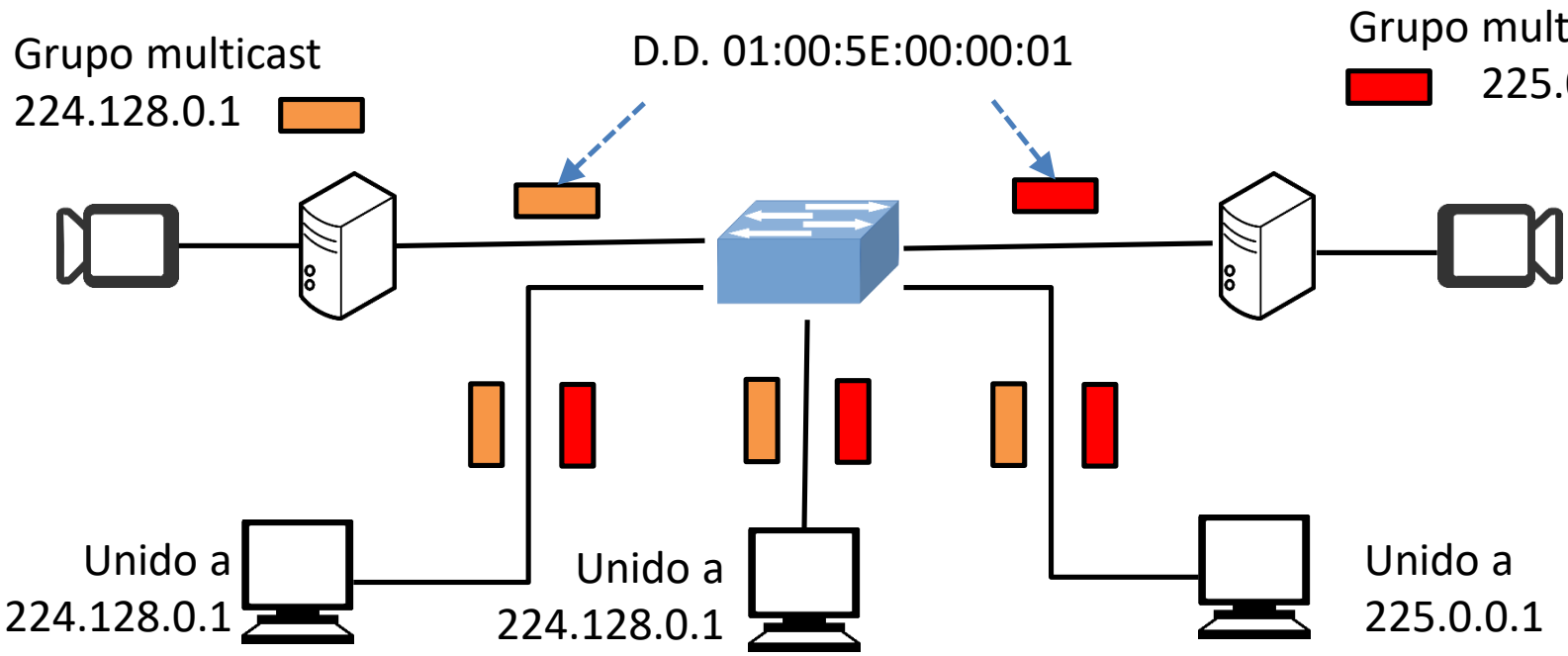
Dirección	Uso
224.0.0.0	Reservada
224.0.0.1	Host con soporte multicast
224.0.0.2	Routers con soporte multicast
224.0.0.4	Routers DVMRP
224.0.0.5	Routers OSPF
224.0.0.6	Routers OSPF designado
224.0.0.9	Routers RIPv2
224.0.0.10	Routers IGRP
224.0.0.11	Agentes móvilrd

Globales

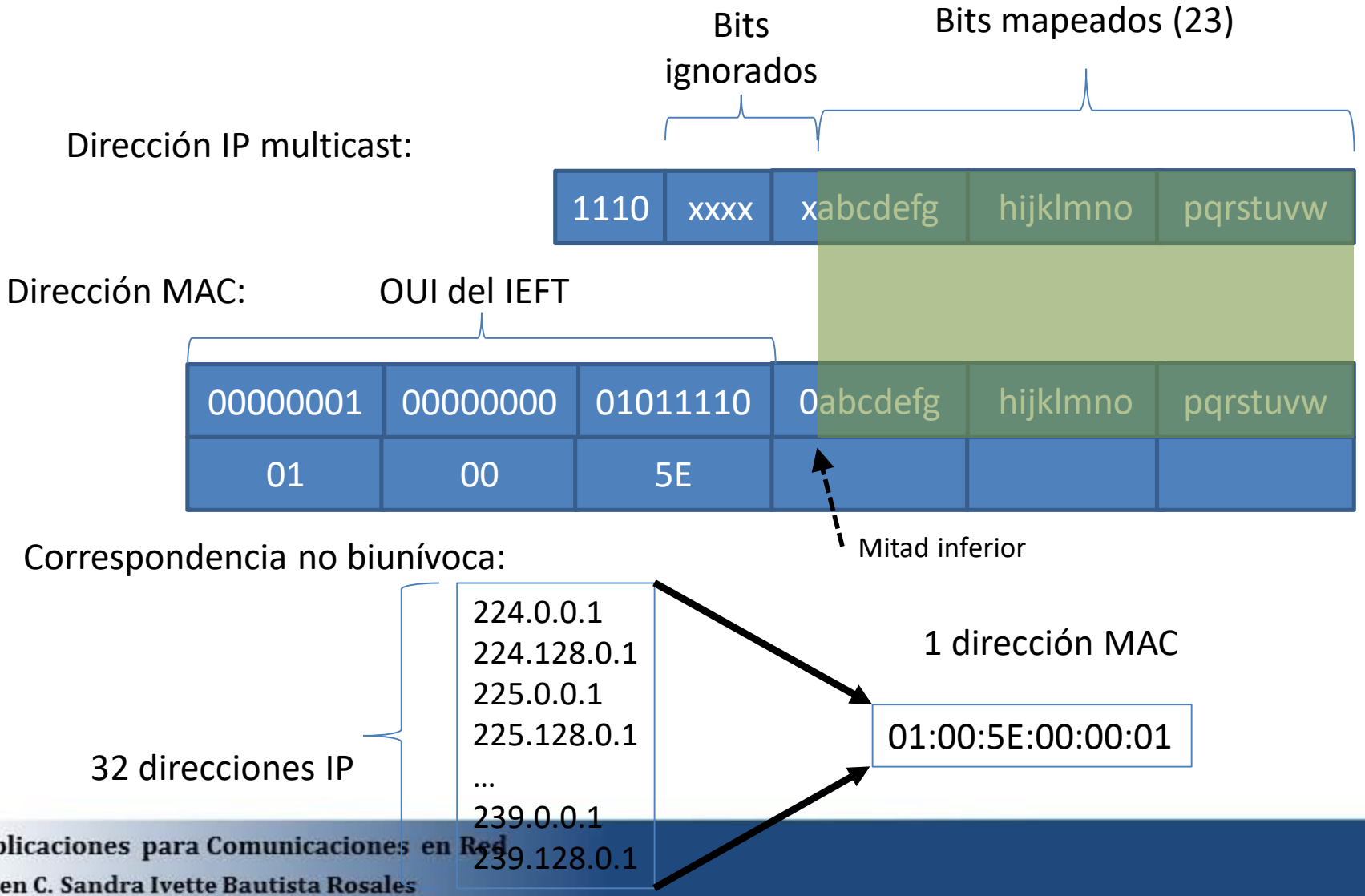
Dirección	Uso
224.0.1.1	NTP – Network Time protocol
224.0.1.7	Noticias en audio
224.0.1.12	Video IETF – 1
224.0.1.16	Servicio de música
224.1.1.39	Anuncio RP (PIM)
224.0.1.40	Descubrimiento RP (PIM)
224.0.1.41	Gatekeepers (H.323)
224.0.1.52	Directorio VCR de MBone
224.0.1.68	Protocolo MADCAP

2.2.2 RESOLUCIÓN DE DIRECCIONES DE MULTIDIFUSIÓN LÓGICAS A FÍSICAS

Resolución de direcciones multicast



Resolución de direcciones multicast IP a Ethernet



2.2.3 PROTOCOLO IGMP (INTERNET GROUP MANAGEMENT PROTOCOL)

Introducción

- Protocolo que permite a los *hosts* comunicar su interés, o no, en pertenecer a grupos *multicast,* dinámicamente.
- Este interés se comunica a los *routers multicast* que usarán la información para construir o podar árboles de distribución *multicast* y usarlos en algún algoritmo de encaminamiento *multicast.*
- Los mensajes IGMP van encapsulados dentro de datagramas IP, con número de protocolo IP = 2, TTL = 1 y con la opción *IP Router Alert* en la cabecera IP.
- Existen 3 versiones incrementales. La más usada es la versión 2.

Multicasting

- Direcciones IPv4:
  - *unicast*, dirigida a un nodo (Clases A, B y C).
  - *broadcast,* dirigida a todos los nodos de una subred.
  - *multicast,* dirigida a un grupo de nodos de una subred (Clase D).
- Dirección multicast:

Multicast, clase D, rango 224.0.0.0 al 239.255.255.255

Rango de direcciones multicast

- 224.0.0.0 - 239.255.255.255, bloques control para red local.
- 224.0.1.0 - 224.0.1.255, bloques control para Internet.
- 224.0.2.0 - 224.0.255.0, bloques AD-HOC.
- 224.1.0.0 - 224.1.255.255, grupos *multicast* estándar.
- 224.2.0.0 - 224.2.255.255, bloques SDP/SAP.

Interfaz MulticastChannel

- Canal para el manejo de multicast IP
- Soporta los RFC:
  - 2236 IGMP v2 para IPv4
  - 3376 IGMP v3 para IPv4
  - 2710 MLD para IPv6
  - 3810 MLD v2 para IPv6

Métodos

- close(); para cerrar el canal
- join(InetAddress group, NetworkInterface if); para unirse a un canal donde se van a recibir datagramas.
- join(InetAddress group, NetworkInterface if, InetAddress source); para unirse a un canal donde se van a recibir y emitir datagramas

Elementos a considerar

- En la creación del canal se debe de especificar la familia de protocolos a utilizar
- La opción SO_REUSEADDR debe de estar habilitada

Ejemplo de uso

```
// Unirse a un grupo en una determinada interfaz
//para enviar y recibir
NetworkInterface ni = NetworkInterface.getByName("hme0");

DatagramChannel dc =
        DatagramChannel.open(StandardProtocolFamily.INET)
        .setOption(StandardSocketOption.SO_REUSEADDR, true)
        .bind(new InetSocketAddress(5000))
        .setOption(StandardSocketOption.IP_MULTICAST_IF, ni);

InetAddress group = InetAddress.getByName("225.4.5.6");

MembershipKey key = dc.join(group, ni);
```

Ejemplo Multicast MultiC.java (1/3)

```
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.util.Collections;
import java.util.Enumeration;
import java.util.Iterator;

public class MultiC {
   static void displayInterfaceInformation(NetworkInterface netint) throws SocketException {
      System.out.printf("Interfaz: %s\n", netint.getDisplayName());
      System.out.printf("Nombre: %s\n", netint.getName());
      Enumeration<InetAddress> inetAddresses = netint.getInetAddresses();
      for (InetAddress inetAddress : Collections.list(inetAddresses)) {
         System.out.printf("InetAddress: %s\n", inetAddress);
      }
      System.out.printf("\n");
    }
```

MultiC.java (2/3)

```
public static void main(String[] args){
  int pto=2000;
    String hhost="230.0.0.1";
    SocketAddress remote=null;
  try{
    try{
        remote = new InetSocketAddress(hhost, pto);
    }catch(Exception e){
        e.printStackTrace();
    }//catch
    Enumeration<NetworkInterface> nets = NetworkInterface.getNetworkInterfaces();
    for (NetworkInterface netint : Collections.list(nets))
        displayInterfaceInformation(netint);
    NetworkInterface ni = NetworkInterface.getByName("eth3");
    DatagramChannel cl = DatagramChannel.open(StandardProtocolFamily.INET);
    cl.setOption(StandardSocketOptions.SO_REUSEADDR, true);
    cl.setOption(StandardSocketOptions.IP_MULTICAST_IF, ni);
    cl.configureBlocking(false);
    Selector sel = Selector.open();
    cl.register(sel, SelectionKey.OP_READ|SelectionKey.OP_WRITE);
    InetAddress group = InetAddress.getByName("230.0.0.1");
    cl.join(group, ni);
    ByteBuffer b = ByteBuffer.allocate(4);
    int n=0;
```

```
        while(n<100){
            sel.select();
            Iterator<SelectionKey>it = sel.selectedKeys().iterator();
            while(it.hasNext()){
               SelectionKey k = (SelectionKey)it.next();
               it.remove();
               if(k.isWritable()){
                  DatagramChannel ch = (DatagramChannel)k.channel();
                  b.clear();
                  b.putInt(n++);
                  b.flip();
                  ch.send(b, remote);
                  continue;
               }
            }//while
          }//while
          cl.close();
          System.out.println("Termina envio de datagramas");
        }catch(Exception e){
           e.printStackTrace();
        }//catch
    }//main
}
```

MultiC.java (3/3)

MultiS.java (1/3)

```
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.util.Collections;
import java.util.Enumeration;
import java.util.Iterator;

public class MultiS {

    static void displayInterfaceInformation(NetworkInterface netint)
                    throws SocketException {
      System.out.printf("Interfaz: %s\n", netint.getDisplayName());
      System.out.printf("Nombre: %s\n", netint.getName());
      Enumeration<InetAddress> inetAddresses =
                netint.getInetAddresses();
      for (InetAddress inetAddress : Collections.list(inetAddresses)) {
         System.out.printf("InetAddress: %s\n", inetAddress);
      }
      System.out.printf("\n");
    }
```

MultiS.java (2/3)

```
public static void main(String[] args){
      try{
        int pto=2000;

        Enumeration<NetworkInterface> nets =
                                    NetworkInterface.getNetworkInterfaces();
        for (NetworkInterface netint : Collections.list(nets))
                          displayInterfaceInformation(netint);

        NetworkInterface ni = NetworkInterface.getByName("eth3");

        InetSocketAddress dir = new InetSocketAddress(pto);
        DatagramChannel s =
                           DatagramChannel.open(StandardProtocolFamily.INET);
        s.setOption(StandardSocketOptions.SO_REUSEADDR, true);
        s.setOption(StandardSocketOptions.IP_MULTICAST_IF, ni);
        InetAddress group = InetAddress.getByName("230.0.0.1");
        s.join(group, ni);
        s.configureBlocking(false);
        s.socket().bind(dir);
        Selector sel = Selector.open();
        s.register(sel, SelectionKey.OP_READ);
        ByteBuffer b = ByteBuffer.allocate(4);
        System.out.println("Servidor listo.. Esperando datagramas...");
        //int n=0;
```

MultiS.java (3/3)

```
while(true){
        sel.select();
        Iterator<SelectionKey>it = sel.selectedKeys().iterator();
        while(it.hasNext()){
          SelectionKey k = (SelectionKey)it.next();
          it.remove();
          if(k.isReadable()){
            DatagramChannel ch = (DatagramChannel)k.channel();
            b.clear();
            SocketAddress emisor = ch.receive(b);
            b.flip();
            InetSocketAddress d = (InetSocketAddress)emisor;
            System.out.println("Datagrama recibido desde "+
            d.getAddress()+":"+d.getPort());
            System.out.println("Dato: "+b.getInt());
            continue;
          }
        }//while
      }//while
    }catch(Exception e){
       e.printStackTrace();
    }//catch
  }//main
}
```

Ejemplo, conexión a un canal, servidor

```
import java.net.*;
import java.io.*;
public class SMulticast {
   public static void main(String[] args ){
      InetAddress gpo=null;
      try{
         MulticastSocket s= new MulticastSocket(9876);
         s.setReuseAddress(true);
         s.setTimeToLive(1);
         String msj ="hola";
         byte[] b = msj.getBytes();
         gpo = InetAddress.getByName("228.1.1.1");
         s.joinGroup(gpo);
         for(;;){
            DatagramPacket p = new DatagramPacket(b,b.length,gpo,9999);
            s.send(p);
            System.out.println("Enviando mensaje "+msj+ " con un TTL= "+ s.getTimeToLive());
            try{
                  Thread.sleep(3000);
               }catch(InterruptedException ie){
                  ie.printStackTrace();
               }
            }//for
         }catch(Exception e){
            e.printStackTrace();
         }//catch
      }//main
}
```

Ejemplo, conexión a un canal, cliente

```
import java.net.*;
import java.net.MulticastSocket;

public class CMulticast {
   public static void main(String[] args ){
      InetAddress gpo=null;
      try{
         MulticastSocket cl= new MulticastSocket(9999);
         System.out.println("Cliente escuchando puerto "+
                                    cl.getLocalPort());
         cl.setReuseAddress(true);
         try{
            gpo = InetAddress.getByName("228.1.1.1");
         }catch(UnknownHostException u){
            System.err.println("Direccion no valida");
         }//catch
         cl.joinGroup(gpo);
         System.out.println("Unido al grupo");
for(;;){
            DatagramPacket p = new DatagramPacket(new
byte[10],10);
            cl.receive(p);
            String msj = new String(p.getData());
            System.out.println("Datagrama recibido.."+msj);
            System.out.println("Servidor descubierto: " +
                        p.getAddress()+" puerto:"+p.getPort());

         }//for

      }catch(Exception e){
         e.printStackTrace();
      }//catch
   }//main
}
```