

# Sockets de Datagrama

A diferencia del socket de flujo, el socket de datagrama **no** brinda los siguientes servicios de entrega:

- Entrega en orden de los datos
- Segmentación y ensamblado de datos
- Entrega completa de los datos
- Entrega libre de errores de los datos
- Entrega sin duplicados de los datos

# Sockets de Datagrama

A diferencia del socket de flujo, el socket de datagrama si permite usar los siguientes tipos de direccionamiento:

- Unidifusión (*unicast*)
- Multidifusión (*multicast*)
- Difusión (*broadcast*)

# Sockets de Datagrama en Java

Clases que implementan el socket de datagrama:

- `java.net.DatagramPacket`
- `java.net.DatagramSocket`

# java.net.DatagramPacket

## Constructores:

- DatagramPacket(byte[ ] buf, int length)
- DatagramPacket(byte[ ] buf, int length, InetAddress address, int port)
- DatagramPacket(byte[ ] buf, int length, SocketAddress address)

# java.net.DatagramPacket

## Métodos:

- InetAddress getAddress()
- byte[ ] getData()
- int getLength()
- int getPort()
- SocketAddress getSocketAddress()
- void setAddress(InetAddress iaddr)
- void setData(byte[ ] buf)
- void setLength(int length)
- void setPort(int iport)
- void setSocketAddress(SocketAddress address)

# java.net.DatagramSocket

## Constructores:

- DatagramSocket()
- DatagramSocket(int port)
- DatagramSocket(int port, InetAddress laddr)
- DatagramSocket(SocketAddress bindaddr)

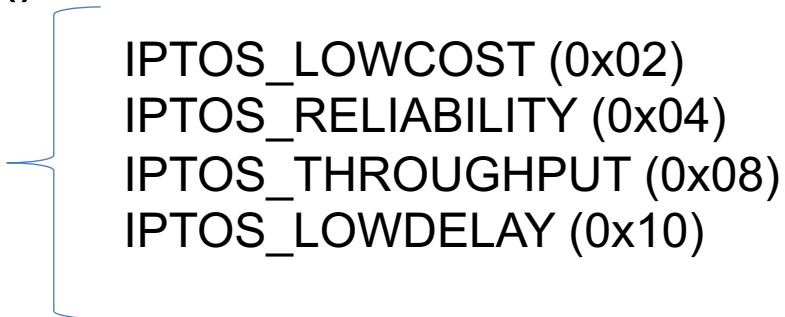
# java.net.DatagramSocket

## Métodos:

- void bind(SocketAddress addr)
- void close()
- void connect(InetAddress address, int port)
- void disconnect()
- boolean getBroadcast()
- DatagramChannel getChannel()
- InetAddress getInetAddress()
- InetAddress getLocalAddress()
- int getLocalPort()
- int getPort()
- int getReceiveBufferSize()

# java.net.DatagramSocket


## Métodos:

- boolean    getReuseAddress()
- int        getSendBufferSize()
- int        getSoTimeout()
- int        getTrafficClass() 
  - IPTOS\_LOWDELAY (0x01)
  - IPTOS\_THROUGHPUT (0x08)
  - IPTOS\_RELIABILITY (0x04)
  - IPTOS\_LOWCOST (0x02)
- boolean    isBound()
- boolean    isClosed()
- boolean    isConnected()
- void       receive(DatagramPacket p)
- void       send(DatagramPacket p)
- void       setBroadcast(boolean on)
- void       setReceiveBufferSize(int size)



# java.net.DatagramSocket

## Métodos:

- void        setReuseAddress(boolean on)
  - void        setSendBufferSize(int size)
  - void        setSoTimeout(int timeout)
  - void        setTrafficClass(int tc)
- 
- IPTOS\_LOWCOST (0x02)
  - IPTOS\_RELIABILITY (0x04)
  - IPTOS\_THROUGHPUT (0x08)
  - IPTOS\_LOWDELAY (0x10)

# ¿Cómo enviar distintos tipos de datos a través de un socket?

- Texto: Para enviar texto sin importar el tipo de codificación se usan las clases `PrintWriter` y `OutputStreamWriter`.

- Ej.

```
try{
    DatagramSocket cl = new DatagramSocket();
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    PrintWriter pw = new PrintWriter(new OutputStreamWriter(baos));
    String msj = "un mensaje";
    pw.println(msj);
    byte[] b = baos.toByteArray();
    DatagramPacket p = new DatagramPacket(b, b.length, dst, 8888);
    cl.send(p);
}catch(Exception e){
    e.printStackTrace();
}
```

# ¿Cómo enviar distintos tipos de datos a través de un socket?

- Primitivos: Para enviar tipos de dato primitivos (int, float, long, boolean, etc.) se usa la clase `DataOutputStream`.

- Ej. 

```
try{
    DatagramSocket cl = new DatagramSocket();
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream(baos);
    String msj = "un mensaje";
    dos.writeUTF(msj);
    dos.writeInt(3);
    dos.writeFloat(2.1f);
    byte[] b = baos.toByteArray();
    DatagramPacket p = new DatagramPacket(b,b.length,dst,8888);
    cl.send(p);
}catch(Exception e){
    e.printStackTrace();
}
```

# ¿Cómo enviar distintos tipos de datos a través de un socket?

- Objetos y primitivos: Para enviar objetos, o tipos de dato primitivos (int, float, long, boolean, etc.) se usa la clase ObjectOutputStream.

```
try{
    DatagramSocket cl = new DatagramSocket();
    • Ej.   ByteArrayOutputStream baos = new ByteArrayOutputStream();
           ObjectOutputStream oos = new ObjectOutputStream(baos);
           String msj = "un mensaje";
           oos.writeUTF(msj);
           oos.writeInt(3);
           oos.writeFloat(2.1f);
           Dato d = new Dato(1,2.0f,"tres");
           oos.writeObject(d);
           byte[] b = baos.toByteArray();
           DatagramPacket p = new DatagramPacket(b,b.length,dst,8888);
           cl.send(p);
           }catch(Exception e){
               e.printStackTrace();
           }
```

# ¿Cómo recibir distintos tipos de datos a través de un socket?

- Texto: Para recibir texto sin importar el tipo de codificación se usan las clases `BufferedReader` e `InputStreamReader`.
- Ej.

```
try{
    DatagramSocket cl = new DatagramSocket(1234);
    DatagramPacket p = new DatagramPacket(new byte[65535], 65535);
    cl.receive(p);
    BufferedReader br = new BufferedReader(new InputStreamReader(new
        ByteArrayInputStream(p.getData())));
    String msj = br.readLine();
} catch (Exception e) {
    e.printStackTrace();
}
```

# ¿Cómo recibir distintos tipos de datos a través de un socket?

- Primitivos: Para recibir tipos de dato primitivos (int, float, long, boolean, etc.) se usa la clase `DataInputStream`.

- Ej.

```
try{
    DatagramSocket cl = new DatagramSocket(1234);
    DatagramPacket p = new DatagramPacket(new byte[65535], 65535);
    cl.receive(p);
    DataInputStream dis = new DataInputStream(new ByteArrayInputStream(p.getData()));
    String v1 = dis.readUTF();
    int v2 = dis.readInt();
    float v3 = dis.readFloat();
} catch (Exception e) {
    e.printStackTrace();
}
```

# ¿Cómo recibir distintos tipos de datos a través de un socket?

- Objetos y primitivos: Para recibir objetos, o tipos de dato primitivos (int, float, long, boolean, etc.) se usa la clase `ObjectInputStream`.
- Ej.

```
try{
    DatagramSocket cl = new DatagramSocket(1234);
    DatagramPacket p = new DatagramPacket(new byte[65535], 65535);
    cl.receive(p);
    ObjectInputStream ois = new ObjectInputStream(new
        ByteArrayInputStream(p.getData()));
    String v1 = ois.readUTF();
    int v2 = ois.readInt();
    float v3 = ois.readFloat();
    Dato d = (Dato)ois.readObject();
} catch (Exception e) {
    e.printStackTrace();
}
```

# Ejemplo: servicio de eco

- Revisar los programas CecoD.java y SecoD.java desde los recursos



# Ejemplo: envío de objetos

- Revisar los programas CDO.java y SDO.java y Objeto.java desde los recursos

# Tarea

- Tarea: Envío de archivos en datagrama

# Difusión (broadcast)

- Para habilitar la entrega/recepción de datagramas a direcciones de difusión (broadcast), es necesario habilitar la opción de socket `SO_BROADCAST`.
- Ej.

```
Byte[ ] b = "un mensaje".getBytes();
DatagramSocket cl = new DatagramSocket();
cl.setBroadcast(true);
InetAddress dst = InetAddress.getByName("255.255.255.255");
DatagramPacket p = new DatagramPacket(b,b.length,dst,1234);
cl.send(p);
```

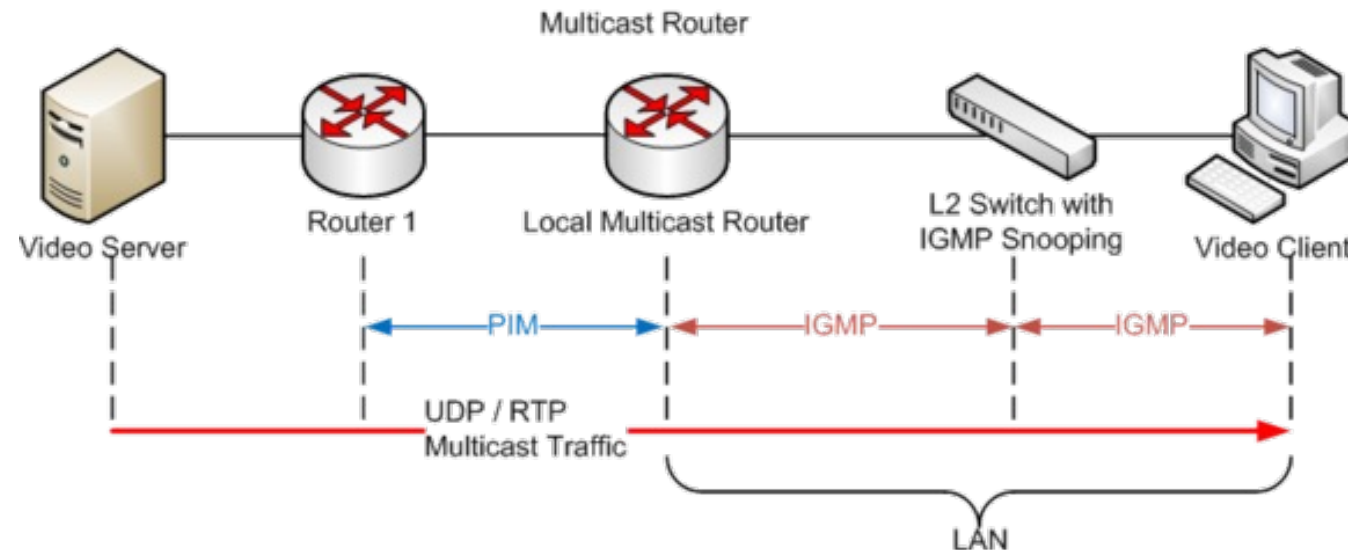
# Ejemplo: envío de datagrama broadcast

- Revisar los programas CHMD.java y SHMD.java desde los recursos

Multidifusión

# IGMP (Protocolo de Gestión de Grupos de Internet, *Internet Group Management Protocol*)

- Protocolo IGMP es utilizado para gestionar la pertenencia a grupos de multidifusión en redes LAN
- Tiene su especificación en RFC 1112(IGMPv1), 2236(IGMPv2), 3376(IGMPv3)
- Este protocolo forma parte de la pila TCP/IP y opera en la capa de red encapsulado dentro de un paquete IP (**protocolo=2**)



# Versiones de IGMP

- **IGMPv1 (RFC 1112):** Los Host pueden unirse a grupos de Multicast. **No hay mensajes de abandono** del grupo. Los enrutadores procesan las bajas del grupo usando el mecanismo **Time-out** (260 seg.) para descubrir los host que ya no están interesados en ser miembros.
- **IGMPv2 (RFC2236):** Añade la **capacidad de abandonar un grupo** al protocolo, permitiendo a los miembros del grupo abandonar activamente un grupo Multicast.
- **IGMPv3 (RFC 3376):** introduce la **seguridad gracias a las fuentes de multidifusión seleccionables**. Una revisión mayor del protocolo, que permite a los host especificar el origen deseado de tráfico Multicast. El tráfico que viene de otros host es bloqueado. Esto permite a los host bloquear paquetes que vienen desde fuentes que envían tráfico indeseado.

# Funcionamiento IGMP

- Cuando una aplicación de la capa de aplicación decide transmitir vía multicast (UDP, direcciones clase D):
  - La dirección IP multicast se mapea en una dirección MAC multicast y la interfaz de red comienza a escuchar dicha dirección además de su propia dirección MAC

Ej. ( 224.1.1.1) →(01:00:5e:01:01:01)

\*Distintas direcciones IP multicast pueden producir la misma dirección MAC multicast

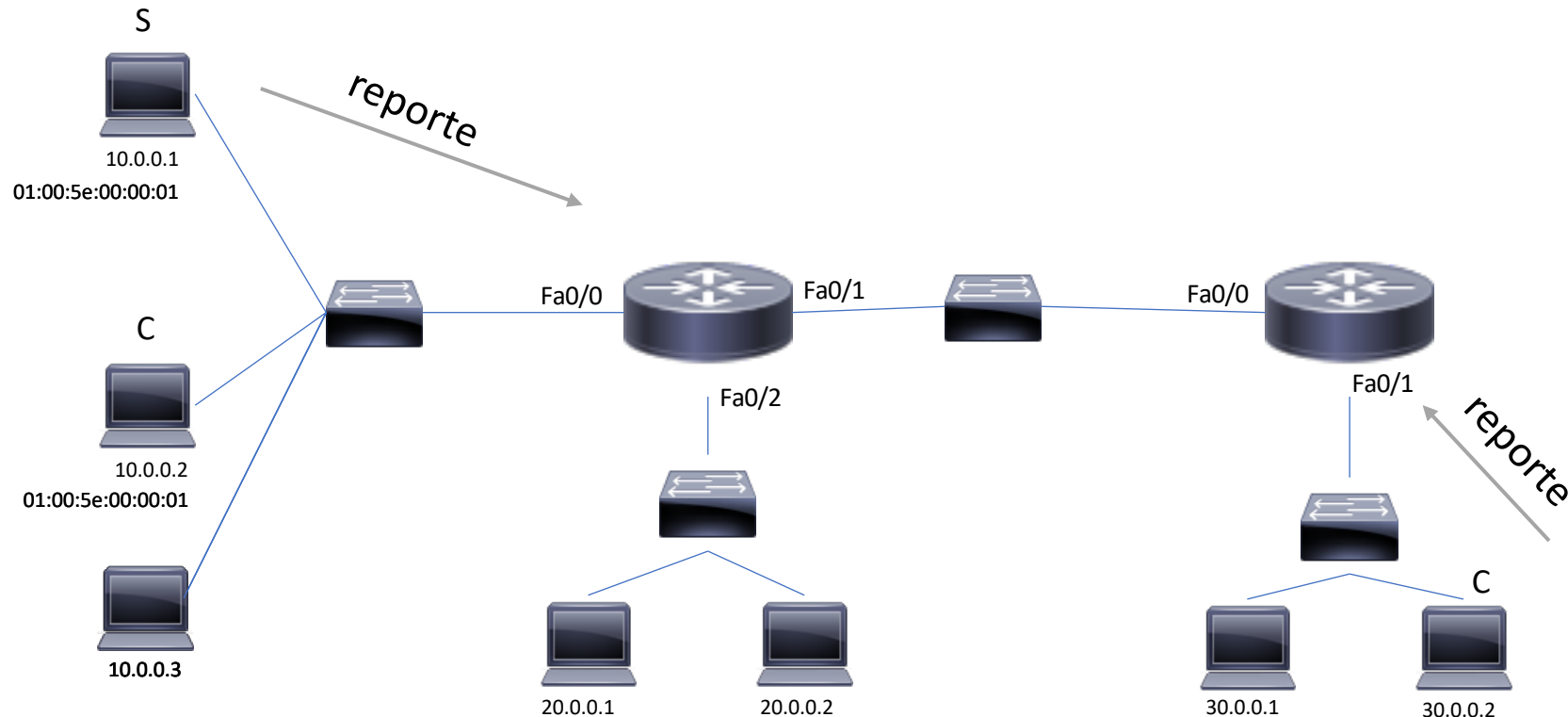
Ej. (230.129.10.10) →(01:00:5e:01:0A:0A)

(225.1.10.10) →(01:00:5e:01:0A:0A)



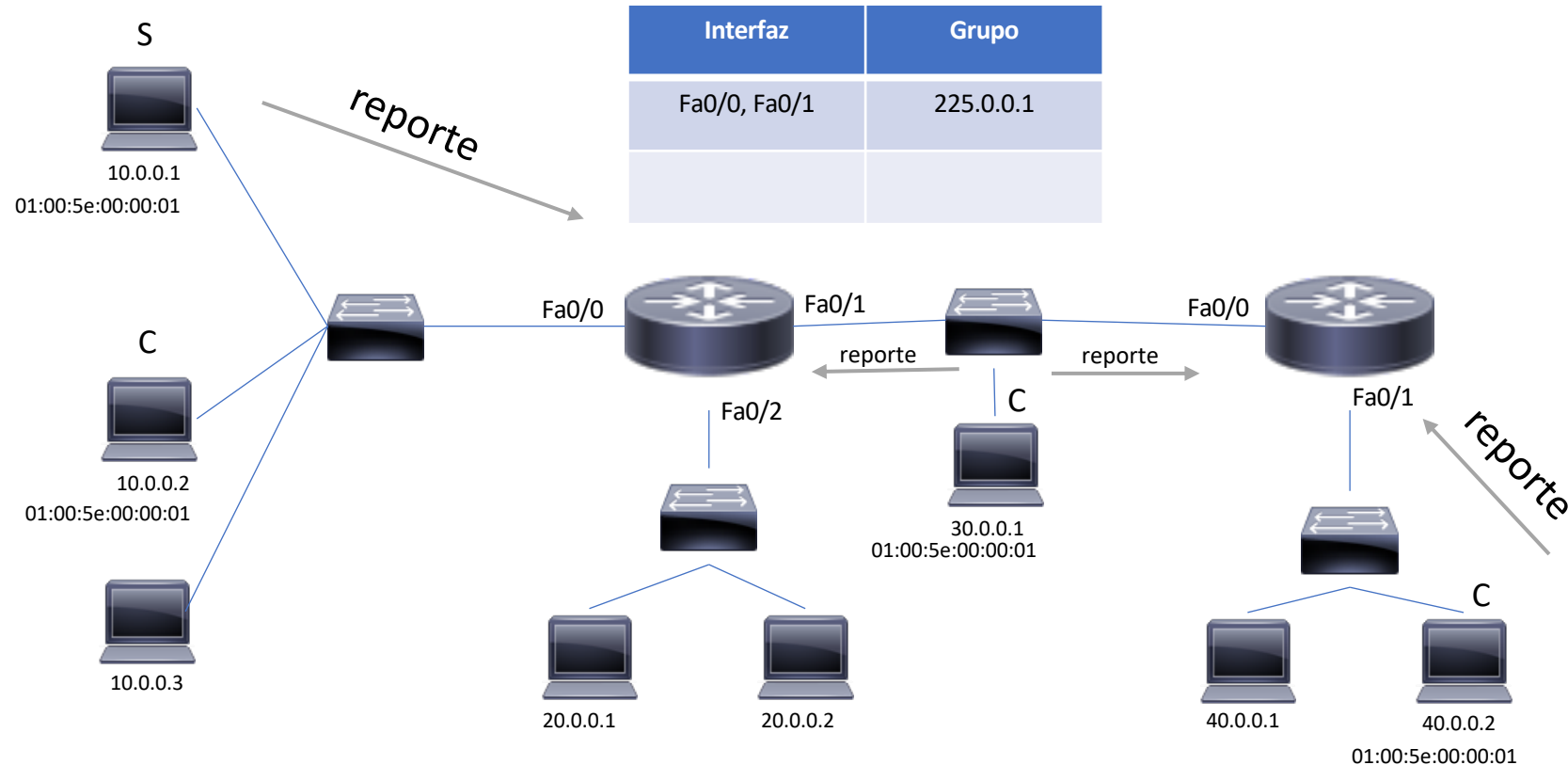
# Funcionamiento IGMP

- El host transmite un mensaje de reporte a los enrutadores IGMP cercanos avisando que escuchará la dirección de grupo



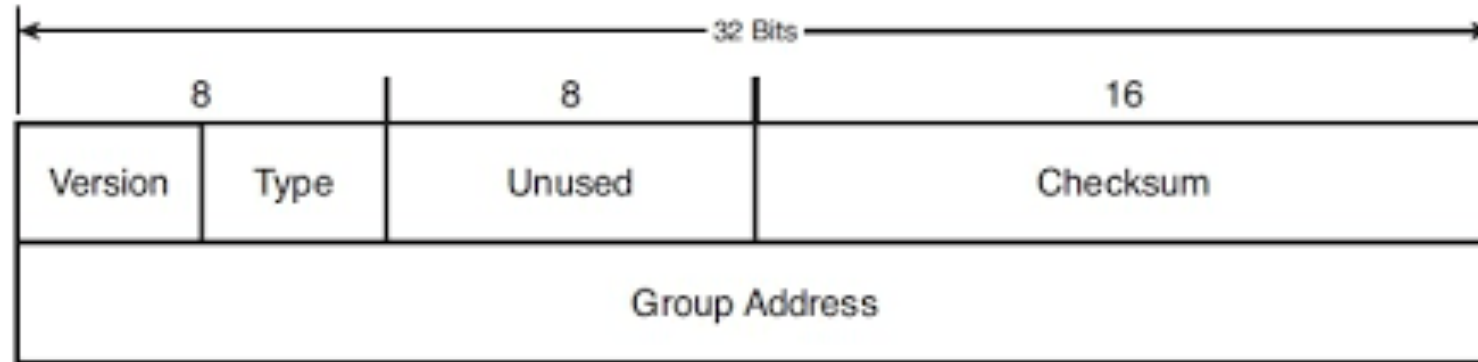
# Funcionamiento IGMP

- El enrutador recibe el reporte y actualiza su tabla IGMP agregando ya sea una nueva entrada con la dirección de grupo y la interfaz por donde se recibió el reporte, o solo añadiendo la nueva interfaz por donde hay que transmitir copias de dicho grupo



# Formato de mensaje IGMP

- IGMPv1



IGMP message type values

Message	Type value
Membership Query	0x11
IGMPv1 Membership Report	0x12
IGMPv2 Membership Report	0x16
IGMPv3 Membership Report	0x22
Leave Group	0x17

Tipo= { 1 enviado por el enrutador  
2 enviado por el host

# Formato de mensaje IGMP

- IGMPv2

0-7	8-15	16-31
Type	Max Resp Time	Checksum
Group Address		

**IGMP message type values**

Message	Type value
Membership Query	0x11
IGMPv1 Membership Report	0x12
IGMPv2 Membership Report	0x16
IGMPv3 Membership Report	0x22
Leave Group	0x17

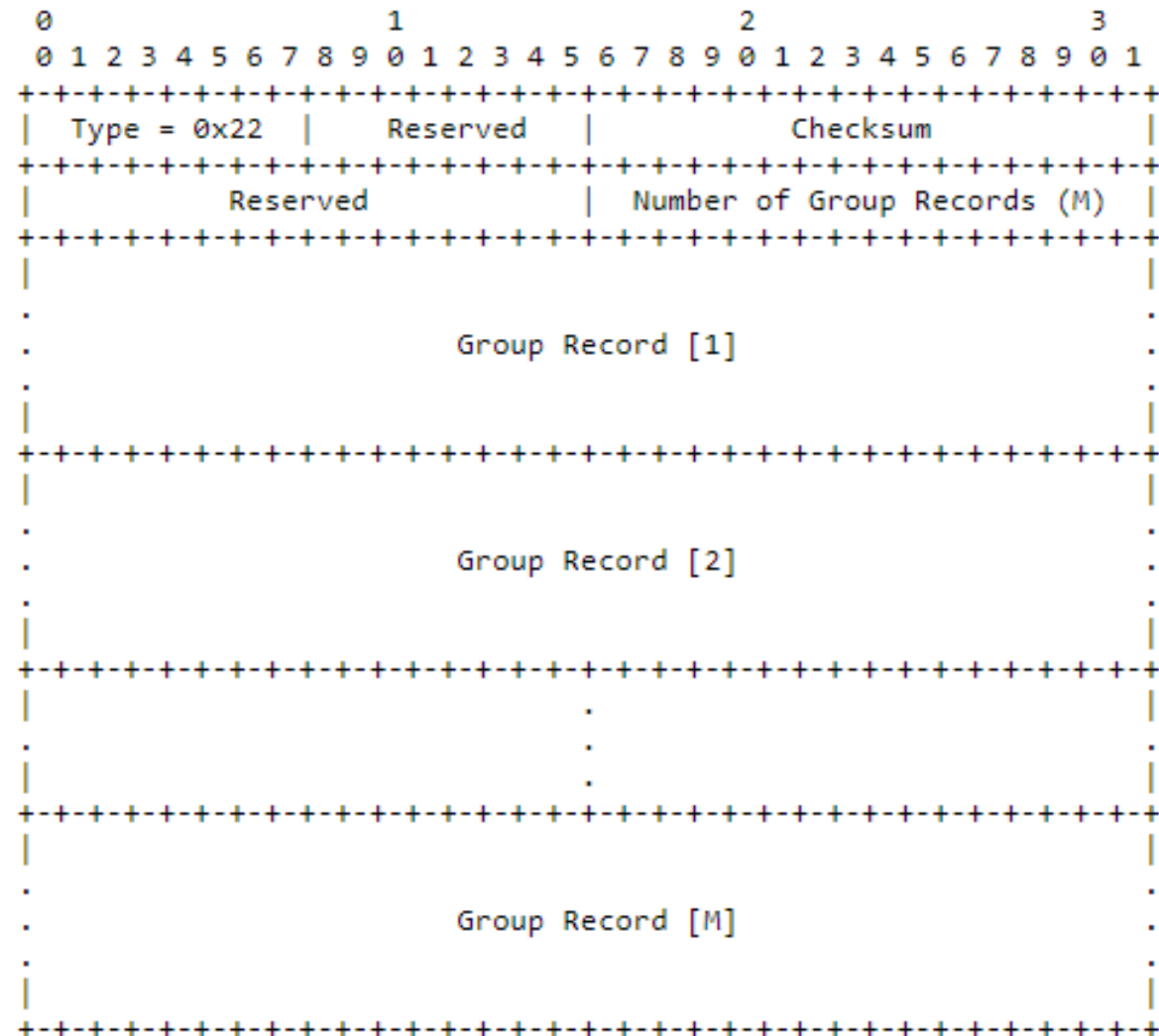
Tiempo { Solo para el tipo (0x11)<sub>16</sub> en milisegundos

# Formato de mensaje IGMP

- IGMPv3

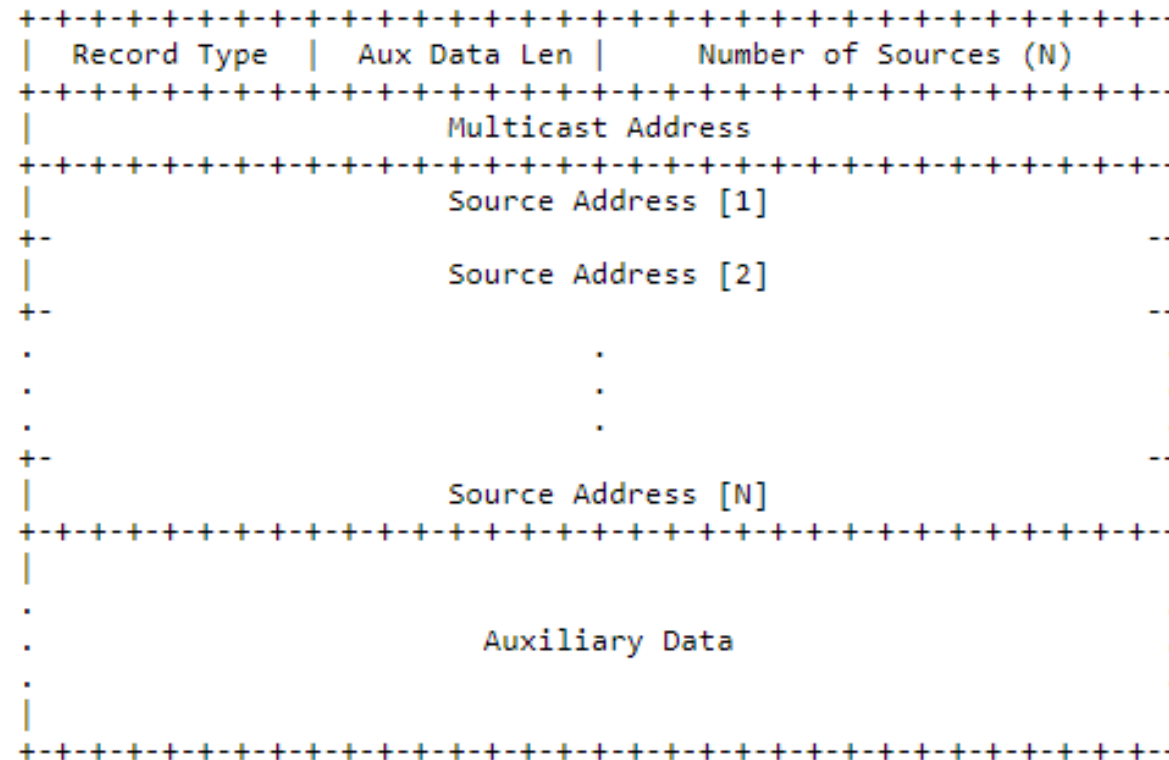
IGMP message type values

Message	Type value
Membership Query	0x11
IGMPv1 Membership Report	0x12
IGMPv2 Membership Report	0x16
IGMPv3 Membership Report	0x22
Leave Group	0x17



# Formato de mensaje IGMP

- IGMPv3



Tipo Registro=

**Como respuesta a una consulta recibida en una interfaz**

1: estado actual interfaz=MODE\_IS\_INCLUDE para la dir multicast especificada y los campos de dirección origen

2: estado actual interfaz=MODE\_IS\_EXCLUDE para la dir multicast especificada y los campos de dirección origen

**Cuando hay una invocación de cambio de modo en la interfaz de red**

3: Change\_TO\_INCLUDE\_MODE la interfaz cambia a modo INCLUDE para la dir multicast y las direcciones origen

4: Change\_TO\_EXCLUDE\_MODE la interfaz cambia a modo EXCLUDE para la dir multicast y las direcciones origen

# Clase `java.net.MulticastSocket`

Constructores:

- `MulticastSocket()`
- `MulticastSocket(int port)`
- `MulticastSocket(SocketAddress bindaddr)`

# Clase java.net.MulticastSocket

Métodos:

- InetAddress      getInterface()
- boolean          getLoopbackMode()
- NetworkInterface getNetworkInterface()
- int              getTimeToLive()
- void            joinGroup(InetAddress mcastaddr)
- void            leaveGroup(InetAddress mcastaddr)
- void            setInterface(InetAddress inf)
- void            setLoopbackMode(boolean disable)
- void            setNetworkInterface(NetworkInterface netIf)
- void            setTimeToLive(int ttl)



# Ejemplo: envío de anuncios multicast

- Revisar los programas SHMM.java y CHMM.java desde los recursos

# Sockets de datagrama bloqueantes en C

# Función socket() //<sys/socket.h>

- int socket(int dominio, int tipo, int protocolo)

```
int sd;
struct addrinfo i, *r, *p;
memset(&i, 0, sizeof(i)); //inicio
i.ai_family = AF_INET6; /* Permite IPv4 or IPv6 */
i.ai_socktype = SOCK_DGRAM;
i.ai_flags = AI_PASSIVE; // utilizado para hacer el bind
i.ai_protocol = 0; /* Any protocol */
i.ai_canonname = NULL;
i.ai_addr = NULL;
i.ai_next = NULL;
if ((rv = getaddrinfo(NULL, pto, &i, &r)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
} //if
for(p = r; p != NULL; p = p->ai_next) {
    if ((sd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
        perror("server: socket");
        continue;
    } //if
    break;
} //for
```

# Función bind()

```
#include <sys/socket.h>
```

```
#include <netinet/in.h>
```

```
int bind(int sd, const struct sockaddr *addr, socklen_t addrlen);
```

- Valor devuelto:

$\left\{ \begin{array}{l} 0 = \text{éxito} \\ -1 = \text{error} \end{array} \right.$

# Ejemplo bind()

```
int sd;
struct addrinfo i, *r, *p;
memset(&i, 0, sizeof (i)); //inicio
i.ai_family = AF_INET6; /* Permite IPv4 or IPv6 */
i.ai_socktype = SOCK_DGRAM;
i.ai_flags = AI_PASSIVE; // utilizado para hacer el bind
i.ai_protocol = 0; /* Any protocol */
i.ai_canonname = NULL;
i.ai_addr = NULL;
i.ai_next = NULL;
if ((rv = getaddrinfo(NULL, pto, &i, &r)) != 0) {
    fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
    return 1;
} //if
for(p = r; p != NULL; p = p->ai_next) {
    if ((sd = socket(p->ai_family, p->ai_socktype, p->ai_protocol)) == -1) {
        perror("server: socket");
        continue;
    } //if
    if (bind(sd, p->ai_addr, p->ai_addrlen) == -1) {
        close(sd);
        perror("server: bind");
        continue;
    } //if

    break;
} //for
```

# Función sendto()

`#include <sys/socket.h>`

`ssize_t sendto(int sd, const void *buf, size_t tam, int bandera, const struct sockaddr *dst, socklen_t tam)`

Valor devuelto: {  
    >0 = #bytes enviados  
    -1 = error  
    0 = socket cerrado

└─> {  
    0 = prioridad default  
    MSG\_OOB= alta prioridad

# Ej. sendto()

```
char *msj ="un mensaje";
```

```
int v1=htonl(5);
```

```
float v2 = 3.0f;
```

```
char b[7];
```

```
sprintf(b,"%f",v2);
```

```
struct datos *d = (struct datos*)malloc(sizeof(struct datos));
```

```
    d->v3=htons(30);
```

```
    d->v4="cadena";
```

```
if(sendto(cd, (const char*)msj, strlen(msj)+1, 0, (struct sockaddr *)rp->ai_addr, rp->ai_addrlen)==-1)
```

```
if(sendto(cd, &v1, sizeof(v1), 0, (struct sockaddr *)rp->ai_addr, rp->ai_addrlen)==-1)
```

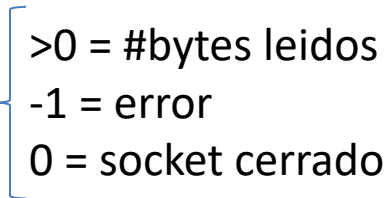
```
if(sendto(cd, b, strlen(b)+1, 0, (struct sockaddr *)rp->ai_addr, rp->ai_addrlen)==-1)
```

```
if(sendto(cd, (const char*)d, sizeof(d), 0, (struct sockaddr *)rp->ai_addr, rp->ai_addrlen)==-1)
```

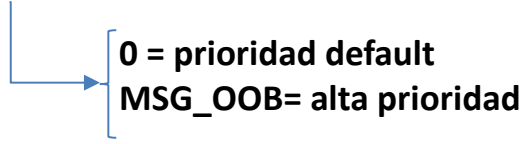
# Función recvfrom()

`#include <sys/socket.h>`

`ssize_t recvfrom(int sd, const void *buf, size_t tam, int bandera, const struct sockaddr *dst, socklen_t *tam)`

Valor devuelto: 

- >0 = #bytes leídos
- 1 = error
- 0 = socket cerrado



0 = prioridad default  
MSG\_OOB = alta prioridad

```
char buf[100];
int n = recv(cd, buf, sizeof(buf), 0);
if (n < 0)
    perror("Error en la función recv\n");
else if (n == 0) {
    perror("Socket cerrado\n");
    exit(1);
}
int v;
n = recv(cd, &v, sizeof(v), MSG_OOB);
...
```



# Ej. recvfrom()

```
char *m =(char *)malloc(sizeof(char)*20);  
memset(m,0,sizeof(m));
```

```
int v1;
```

```
float v2;  
char b[7];  
memset(b,0,sizeof(b));
```

```
struct datos *d;  
Char bb[20];
```

```
struct sockaddr_storage rp;  
memset(&rp,0,sizeof(rp));  
socklen_t ctam = sizeof(rp);
```

```
if(recvfrom(cd, m, sizeof(m), 0, (struct sockaddr *)&rp, &ctam)==-1)
```

```
if(recvfrom(cd, &v1, sizeof(v1), 0, (struct sockaddr *)&rp, &ctam)==-1)  
int vv = ntohs(v1);
```

```
if(recvfrom(cd, b, sizeof(b), 0, (struct sockaddr *)&rp, &ctam)==-1)  
v2 = atof(b);
```

```
if(recvfrom(cd, (const char*)bb, sizeof(bb), 0, (struct sockaddr *)&rp,&ctam)==-1)
```

```
d = (struct datos *)bb;
```

# Ejemplo: eco

- Revisar los programas cliente.c y servidor.c desde los recursos

# Ejemplo: envío de estructura

- Revisar los programas cliente2.c y servidor2.c desde los recursos

# Ejercicio

- Crear un programa que permita al usuario jugar el juego “Ahorcado” en red implementando el servidor en lenguaje C y el cliente en lenguaje JAVA.

# Difusión (broadcast)

- Para poder enviar/recibir datagramas usando direccionamiento de diffusion, es necesario habilitar la opción de socket SO\_BROADCAST

Ej.

```
int bc=1;
int p = setsockopt(cd, SOL_SOCKET, SO_BROADCAST, &bc, sizeof(bc));
char[16] BC = "255.255.255.255";
struct addrinfo dst;
memset(&dst, 0, sizeof(dst));
dst.ai_family = result->ai_family;
dst.ai_socktype = SOCK_DGRAM;
struct addrinfo *result1;
if ( getaddrinfo(BC, "9930", &dst, &result1) != 0 )
//if ( getaddrinfo(SRV_IP, "9930", &dst, &result1) != 0 )
//if ( getaddrinfo(SRV_IP6, "9930", &dst, &result1) != 0 )
{
    perror("getaddrinfo3() failed");
}

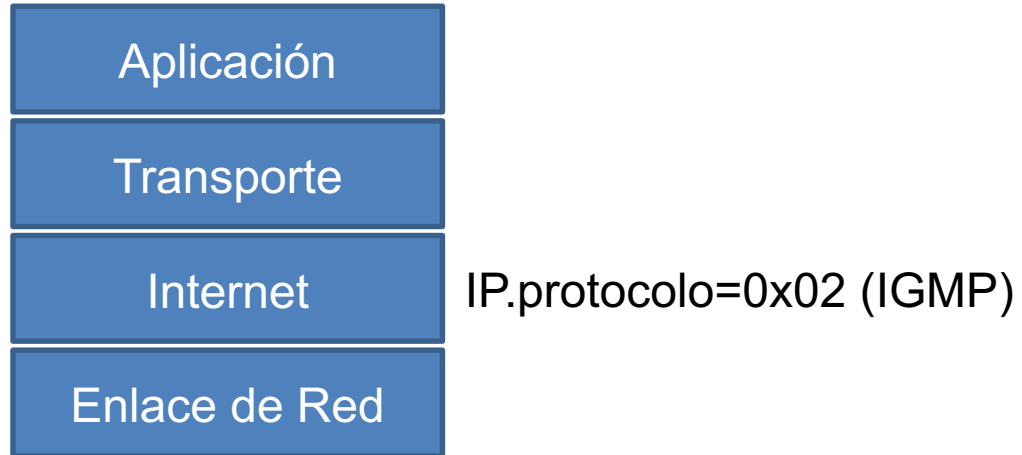
if (sendto(cd, (const char*)o1, sizeof(struct dato), 0, (struct sockaddr *)result1->ai_addr, result1->ai_addrlen)==-1){
```

# Ejemplo: envío de datagrama broadcast

- Revisar los programas cliente3.c y servidor3.c desde los recursos

# Sockets de datagrama multicast bloqueantes en C

# Internet Group Management Protocol (IGMP)





# Mensaje IGMP



Tipo {  
   $(0x11)_{16} = (17)_{10} \Rightarrow$ Consulta  
   $(0x12)_{16} = (18)_{10} \Rightarrow$ Reporte (IGMPv1)  
   $(0x16)_{16} = (22)_{10} \Rightarrow$ Reporte (IGMPv2)  
   $(0x22)_{16} = (34)_{10} \Rightarrow$ Reporte (IGMPv3)

Tiempo { Solo para el tipo  $(0x11)_{16}$  en milisegundos

# Opción de socket SO\_REUSEADDR

```
int op,v=1;  
if ( setsockopt(sd, SOL_SOCKET, SO_REUSE_ADDR, &v, sizeof(v)) != 0 )  
... perror("No se pudo modificar la opción \n ");
```

# Estructura ip\_mreq (ipv4)

```
struct ip_mreq {  
    struct in_addr imr_multiaddr; /* Dir. Grupo multicast */  
    struct in_addr imr_address;  /* Dir. Interfaz de red local */  
};
```

# Estructura ipv6\_mreq (ipv6)

```
struct ipv6_mreq {  
    struct in6_addr  ipv6mr_multiaddr; /* Dir. IPv6 multicast */  
    unsigned int     ipv6mr_interface; /* índice interfaz red */  
}
```

ffxe::/16	224.0.1.0- 238.255.255.255	Alcance Global	
-----------	-------------------------------	----------------	--

# Opción de socket IP\_ADD\_MEMBERSHIP (IPv4)

```
struct ip_mreq mr;
/* Ponemos la dirección de grupo */
    memcpy(&mr.imr_multiaddr,&((struct sockaddr_in*)(maddr->ai_addr))-
>sin_addr,sizeof(mr.imr_multiaddr));

/* Aceptamos datagramas multicast por cualquier interfaz */
mr.imr_interface.s_addr = htonl(INADDR_ANY);

/* Nos unimos a la dirección de grupo */
if ( setsockopt(sd, IPPROTO_IP, IP_ADD_MEMBERSHIP, (char*) &mr, sizeof(mr))
!= 0 )
{
    perror("setsockopt() \n");
}
```

# Opción de socket IP\_ADD\_MEMBERSHIP (IPv6)

```
struct ipv6_mreq mr; /* Multicast address join structure */

/* Especificamos la dirección de grupo IPv6 */
memcpy(&mr.ipv6mr_multiaddr, &((struct sockaddr_in6*)(maddr->ai_addr))-
>sin6_addr, sizeof(mr.ipv6mr_multiaddr));

/* Aceptamos datagramas multicast IPv6 desde cualquier interfaz de red */
mr.ipv6mr_interface = 0;

/* Nos unimos a la dirección de grupo */
if ( setsockopt(sd, IPPROTO_IPV6, IPV6_ADD_MEMBERSHIP, (char*) &mr,
sizeof(mr)) != 0 )
{
    perror("setsockopt() \n");
}
```

# Opción de socket IP\_MULTICAST\_TTL

```
Unsignet char ttl= 200;  
if ((setsockopt(sd, IPPROTO_IP, IP_MULTICAST_TTL,(void*) &ttl, sizeof(ttl))) < 0)  
    perror("setsockopt() \n");
```

\*Ej. cliente2.c, servidor2.c

# Opción de socket IPV6\_MULTICAST\_HOPS

```
Unsignet char ttl= 200;  
if ((setsockopt(sd, IPPROTO_IPV6, IPV6_MULTICAST_HOPS,(void*) &ttl, sizeof(ttl))) < 0)  
    perror("setsockopt() \n");
```

\*Ej. cliente2.c, servidor2.c



# Ejemplo: envío de anuncios multicast

- Revisar los programas cliente3.c y servidor3.c desde los recursos