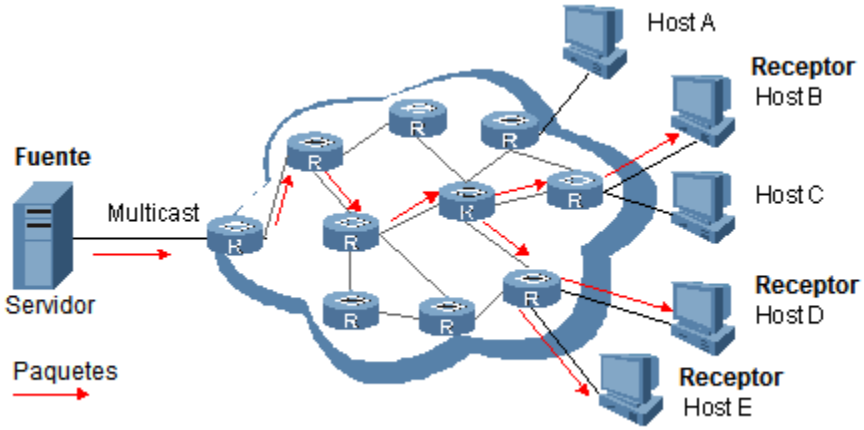
Multicast

Elaborado por: Ukranio Coronilla

Es posible con el protocolo UDP enviar un datagrama hacia un grupo de nodos de una subred mediante un mensaje de multicast (multitransmision). En multicast los datagramas serán duplicados por los routers en caso de que esto sea necesario, evitando así la sobrecarga de la red. De este modo si un servidor envía un stream de video de 1Mbps y la red es de 3Mbps, entonces solo podrá haber 3 clientes, a menos que se utilice multicast. Multicast se utiliza normalmente como una señal de difusión como se muestra en la siguiente figura, y no para la comunicación bidireccional como se realiza con unicast.



Para hacer uso de multicast se debe distinguir primero si se trata del emisor o del receptor del datagrama. Observe que solo puede haber un emisor, pero pueden existir varios receptores simultáneos del datagrama a los cuales se les conoce como grupo.

El socket de comunicación es bastante similar al que se utiliza en UDP unicast, salvo con las excepciones y agregados que se especifican a continuación.

Ya sea que se trate del grupo receptor o del emisor, se debe llamar a la función socket como sigue:

```
s = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
```

En el caso de error, la función socket devolverá un valor negativo.

Si se trata del **emisor** se debe llamar a la función `setsockopt` después de la función `socket` con los siguientes parámetros, y validar que no devuelva un valor negativo.

```
unsigned char TTL;

setsockopt(s, IPPROTO_IP, IP_MULTICAST_TTL, (void *) &TTL, sizeof(TTL))
```

donde la variable TTL almacena el tiempo de vida(Time To Live) del datagrama y significa el número de routers que tiene permitido pasar, esto con el objeto de limitar la propagación del datagrama en la red. Este valor debe ser inicializado por la aplicación antes de llamar a setsockopt y normalmente se inicializa con 1 para redes locales.

Para que el receptor reciba mensajes multicast, es necesario unirse al grupo de receptores ejecutando la siguiente función:

```
setsockopt(s, IPPROTO_IP, IP_ADD_MEMBERSHIP, (void *) &multicast, sizeof(multicast))
```

La cual permite al proceso unirse al grupo de receptores de datagramas multicast. En el caso de que el proceso desee salirse del grupo, el tercer parámetro debe ser `IP_DROP_MEMBERSHIP`.

La variable `multicast` en el cuarto parámetro debe ser de tipo `struct ip_mreq` el cual se define como:

```
struct ip_mreq
{
   struct in_addr imr_multiaddr; /* Dirección IP del grupo multicast */
   struct in_addr imr_interface; /* Dirección IP de la interfaz local IP */
};
```

Recordando que:

```
struct in_addr
{
   u_long s_addr; /* 32 bits que contienen la IP */
}
```

De modo que tendremos que inicializar algo similar a:

```
multicast.imr_multiaddr.s_addr = inet_addr(multicastIP);
multicast.imr_interface.s_addr = htonl(INADDR_ANY);
```

Las direcciones multicast van desde la 224.0.0.0 hasta la 239.255.255.255, y cuando un emisor envía un datagrama a esta dirección, los routers se encargarán de hacer las copias necesarias para hacerlo llegar a todos los procesos que se hayan agregado al grupo con anterioridad, mediante la apertura de su correspondiente socket multicast.

En el caso de la función `bind()`, `recvfrom()` y `sendto()` se siguen utilizando de la misma forma y con los mismos parámetros que en unicast.

Parte 1

Elabore la clase `SocketMulticast` para implementar los sockets de envío y recepción de mensajes multicast, su interfaz se muestra a continuación:

```
class SocketMulticast{
public:
   SocketMulticast(int);
   ~SocketMulticast();
   int recibe(PaqueteDatagrama & p);
   int envia(PaqueteDatagrama & p, unsigned char ttl);
   //Se une a un grupo multicast, recibe la IP multicast
   void unirseGrupo(char *);
   //Se sale de un grupo multicast, recibe la IP multicast
   void salirseGrupo(char *);
private:
   int s; //ID socket
```

```
};
```

En el método `envia` el segundo parámetro es el valor de TTL necesario para enviar el datagrama. En los métodos `unirseGrupo` para unirse al grupo de receptores y `salirseGrupo` para salir del grupo de receptores, el parámetro recibido es la dirección IP de multicast.

Para probar la clase `SocketMulticast` elabore un programa emisor que recibe como parámetros en línea de comandos la IP de multicast, el puerto, el valor de TTL y una cadena encerrada entre comillas. El receptor recibe como parámetros la IP de multicast y el puerto en el que escucha. Al ejecutarse ambos, el recpetor debe imprimir la IP y puerto de quien le ha enviado el mensaje multicast, así como la cadena recibida.

Parte 2

En muchas ocasiones es necesario que dos procesos dentro de la misma computadora pertenezcan al mismo grupo de procesos multicast, lo cual implica que también ambos procesos se encuentren escuchando en el mismo puerto.

Para lograrlo solo es necesario añadir inmediatamente después de la función `socket()` en el constructor de la clase `SocketMulticast` lo siguiente:

```
int reuse = 1;
if (setsockopt(s, SOL_SOCKET, SO_REUSEPORT, &reuse, sizeof(reuse)) == -1) {
  printf("Error al llamar a la función setsockopt\n");
  exit(0);
}
```

Pruébelo con receptores en distintas computadoras cada una con dos receptores o más. El emisor deberá enviar un arreglo de dos enteros en un mensaje multicast, mientras que los receptores deberán imprimir los dos enteros recibidos, posteriormente deben sumar dichos números y regresar el resultado de la suma al emisor por un socket unicast.

Antes de que el emisor termine, deberá imprimir el resultado recibido por todas las computadoras, así como las IP´s de las computadoras que si contestaron. Es importante que el emisor ejecute el método `recibeTimeout` pues podría no recibir la respuesta de un receptor y quedarse bloqueado para siempre.