Week 12, 13: BROADCASTING

- What is Broadcasting
- Local versus Global Broadcasting
- Implementing Broadcasting with C#
    - Broadcasting Examples
    - Bad Broadcasting Examples
- Overview of Multicasting

Broadcasting

What Is Broadcasting?

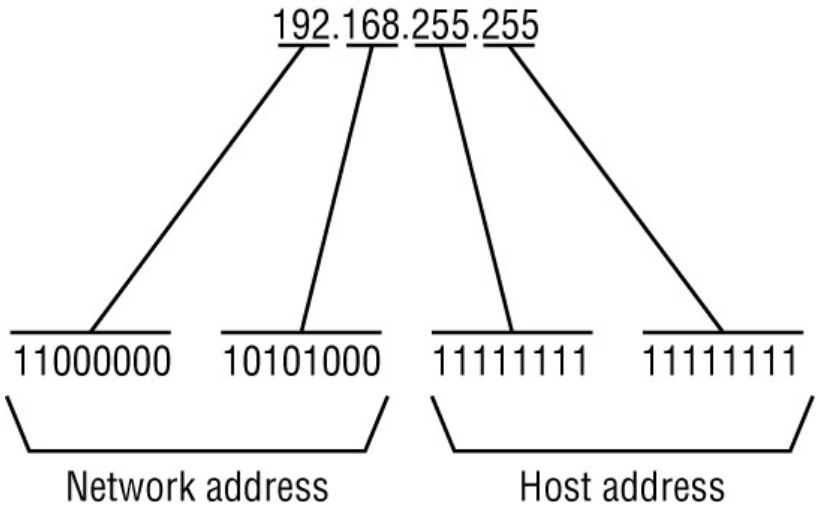
IP broadcasting is used by network devices to send a single packet of information that can be accessible by every device on the network.

For broadcasting, UDP packets must be used. This protocol has the capability of sending messages without a specific connection being defined. TCP communication requires that two devices have a dedicated connection. Therefore, it is not possible to send broadcast messages in a strictly TCP environment.

IP Address in Broadcasting

An IP address is divided into two parts, a network address and a host address. The standard network address part makes up the first part of the local broadcast address, and all 1s are used for the host part of the address (which is the decimal value 255 in the address octet).

Thus, for the class B network 192.168.0.0, using a subnet mask of 255.255.0.0, the local broadcast address would be 192.168.255.255.



The IP network and host address parts of a local broadcast address

Similarly, if the subnet is further divided using a subnet mask of 255.255.255.0, each subnet would have its own local broadcast address. The subnet 192.168.1.0 would have a broadcast address of 192.168.1.255, and so on up to the subnet 192.168.254.0, which would have the broadcast address 192.168.254.255.

Local versus Global Broadcasts

Broadcast messages contain a special destination IP address. The IP address format allows for two types of broadcast message addresses:

- **Local Broadcasts**: uses *local broadcast address*. In this type, a broadcast message is sent to all devices on a particular subnet. That is the message is localized so that other networks are not affected by the broadcast.
- **Global Broadcasts**: the message is sent to all devices on all networks accessible from the device sending the message. The *global broadcast* was originally intended to allow a device to send a packet

to all devices on an internetwork. It uses all 1s in the IP address, creating an address of 255.255.255.255.

To eliminate bogus broadcast possibility, routers do not send global IP broadcasts to other networks unless specifically configured to do so. Instead, they silently ignore global broadcasts and turn them into local broadcasts that are seen only on the local network over which they were sent.

Implementing Broadcasting with C#

The .NET network library contains elements for sending and receiving broadcast packets.

Bad Broadcasting Example

By default, sockets are not allowed to send broadcast messages. You can test this by running this simple program.

The BadBroadcast.cs program

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class BadBroadcast
{
  public static void Main()
  {
   Socket sock = new Socket(AddressFamily.InterNetwork,
      SocketType.Dgram, ProtocolType.Udp);
   IPEndPoint iep = new IPEndPoint(IPAddress.Broadcast, 9050);
   byte[] data = Encoding.ASCII.GetBytes("This is a test
message");
   sock.SendTo(data, iep);
   sock.Close();
  }
}
```

When you try to run this program, you will get an Exception message:

```
C:\>BadBroadcast
Unhandled Exception: System.Net.Sockets.SocketException: An attempt was made to
access a socket in a way forbidden by its access permissions
 at System.Net.Sockets.Socket.SendTo(Byte[] buffer, Int32 offset, Int32 size,
SocketFlags socketFlags, EndPoint remoteEP)
 at System.Net.Sockets.Socket.SendTo(Byte[] buffer, EndPoint remoteEP)
 at BadBroadcast.Main()
C:\>
```

The *BadBroadcast.cs* program seems simple enough, but when it runs, the .NET environment doesn't allow the **SendTo**() method to send the message. As seen, if you attempt to send a broadcast message from a default UDP socket, you will get a SocketException error.

Broadcasting Example

For a C# application to send broadcast packets, ***the broadcast socket option must be set on the created socket*** using the **SetSocketOption()** method of the Socket class:

```
Socket sock = new Socket(AddressFamily.InterNetwork,
        SocketType.Dgram, ProtocolType.Udp);
sock.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.Broadcast, 1);
```

- The SocketType must be set to Dgram for broadcast messages.
- The SocketOptionName.Broadcast option is a Socket level option, so the SocketOptionLevel parameter must be set to Socket.
- Because the SetSocketOption() method does not allow *Boolean* values for the data parameter, you must set the value parameter to any non-zero value to represent the true Boolean value. Zero represents the ***false*** Boolean value.

After the socket option is set, you specify a broadcast address and port to use for the broadcast and use the SendTo() method to send a message on that address:

```
IPEndPoint iep = new IPEndPoint(IPAddress.Broadcast, 9050);
byte[] data = Encoding.Ascii.GetBytes("this message is for
all");
sock.SendTo(data, iep);
```

- Using the Broadcast property of the **IPAddress** class as a destination address, the program will send the broadcast message out to all network interfaces configured on the device.
- To limit the broadcast to a specific interface, you must manually supply a local broadcast address to use for the destination address.

The Broadcst.cs program

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
class Broadcst
{
  public static void Main()
  {
```

```
   Socket sock = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
   IPEndPoint iep1 = new IPEndPoint(IPAddress.Broadcast, 9050);
   IPEndPoint iep2 = new IPEndPoint(IPAddress.Parse("192.168.1.255"),
           9050);
   string hostname = Dns.GetHostName();
   byte[] data = Encoding.ASCII.GetBytes(hostname);
   sock.SetSocketOption(SocketOptionLevel.Socket,
              SocketOptionName.Broadcast, 1);
   sock.SendTo(data, iep1);
   sock.SendTo(data, iep2);
   sock.Close();
  }
}
```

- The *Broadcst* program creates two separate IPEndPoint objects, one using the IPAddress.Broadcast address and one using a local broadcast address 192.168.1.255.
- Next, the local hostname of the device is determined using the standard Dns *GetHostName* () method, and it is sent out in two separate broadcast messages.

Receiving Broadcast Packets****

Receiving broadcast packets is a little less complex than sending them. By default, no special options are required for a socket to receive a broadcast message. If a socket is listening on a specified UDP port, it will accept any message destined for either a broadcast address or the local IP address of the device. The following program demonstrates this by creating a socket to listen for packets on UDP port 9050, the same port from which the *Broadcst* program is sending broadcast messages.

The RecvBroadcst.cs program

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

class RecvBroadcst
{
  public static void Main()
  {
   Socket sock = new Socket(AddressFamily.InterNetwork,
SocketType.Dgram, ProtocolType.Udp);
   IPEndPoint iep = new IPEndPoint(IPAddress.Any, 9050);
   sock.Bind(iep);
   EndPoint ep = (EndPoint)iep;
```

```
    Console.WriteLine("Ready to receive…");
    byte[] data = new byte[1024];
    int recv = sock.ReceiveFrom(data, ref ep);
    string stringData = Encoding.ASCII.GetString(data, 0, recv);
    Console.WriteLine("received: {0} from: {1}",  stringData,
ep.ToString());
    data = new byte[1024];
    recv = sock.ReceiveFrom(data, ref ep);
    stringData = Encoding.ASCII.GetString(data, 0, recv);
    Console.WriteLine("received: {0} from: {1}", stringData,
ep.ToString());
    sock.Close();
   }
}
```

The *RecvBroadcst* program is a simple UDP program that creates a socket and binds it to all network interfaces on the device, using port 9050, the same port that the Broadcst program uses to send its broadcast messages. After binding the socket to the UDP port, the program attempts to receive two messages sent to the port and displays the message and the sending host address information.

Testing the program

You can start the *RecvBroadcst* program in a command prompt window, and then run the Broadcst program from either another command prompt window on the same device, or from another device on the same subnet.

A sample output from the RecvBroadcst program is as follows:

```
C:\>RecvBroadcst
received: abednego from: 192.168.1.6:1042
received: abednego from: 192.168.1.6:1042
C:\>
```

As expected, the RecvBroadcst program received both forms of the broadcast message that were sent by the Broadcst program.

Warning ***When testing broadcast programs, remember to use devices on the same subnet.***

Overview of Multicasting

What Is Multicasting?

Broadcasting is an excellent way to send information to all devices on a subnet, but the broadcast packets are restricted to the local subnet.

IP multicasting was devised to allow an application to send a single packet to a selected subset of devices both on the local subnet and across network boundaries. This feature allows an application to join a multicast group to participate in a wide-area conference.

IP multicasting uses special IP addresses. The IP multicasting scheme uses a particular range of IP addresses to designate different *multicast groups*. Each multicast group consists of a group of devices listening to the same IP address. As packets are sent out destined for the multicast group address, each device listening to the address receives them.

The IP address range **224.0.0.1** through **239.255.255.255** represents multicast groups. According to Internet Request For Comments (RFC) 3171, the groups are divided as shown below:

IP Multicast Address Assignments

Range	***Assignment***
224.0.0.0–224.0.0.255	Local network control block
224.0.1.0–224.0.1.255	Internetwork control block
224.0.2.0–224.0.255.0	AD-HOC block
224.1.0.0–224.1.255.255	ST multicast groups
224.2.0.0–224.2.255.255	SDP/SAP block
224.252.0.0–224.255.255.255	DIS transient block
225.0.0.0–231.255.255.255	Reserved
232.0.0.0–232.255.255.255	Source-specific multicast block
233.0.0.0–233.255.255.255	GLOP block
234.0.0.0–238.255.255.255	Reserved
239.0.0.0–239.255.255.255	Administratively scoped block

Within each of these address blocks, individual IP addresses are assigned to specific projects. For example, addresses 224.0.0.1 and 224.0.0.2 are reserved for routers to communicate multicast group information between themselves. You should avoid using multicast group addresses that occur within these blocks.

Multicast Techniques

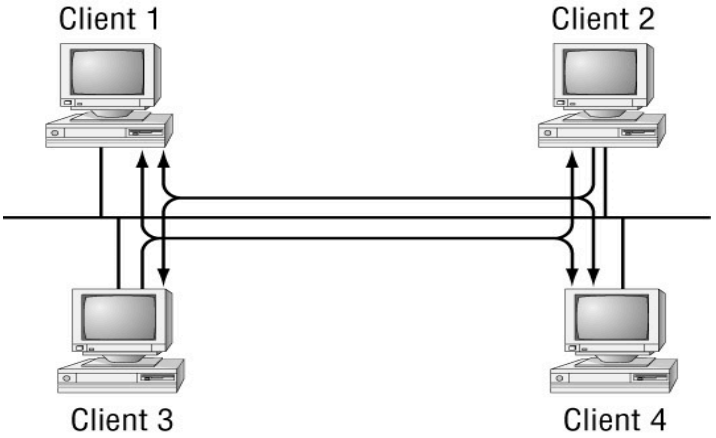
There are two techniques used to control multicast sessions:

- A peer-to-peer technique, in which all clients can send messages to all other clients in the group

- A central server that sends messages to group clients

Peer-to-Peer Technique

In a peer-to-peer multicast group, all of the clients in the multicast group have equal rights in the group. Any client in the group has the capability to exchange messages with any other client in the group.

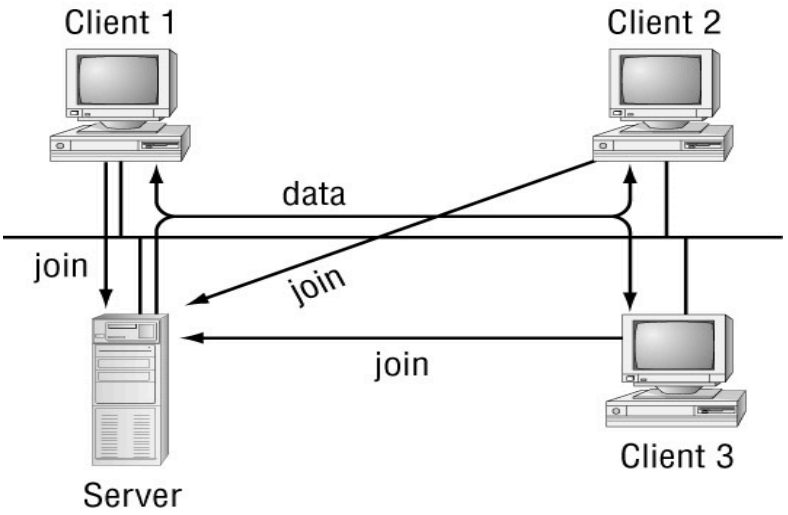


In a peer-to-peer multicast group, all clients can exchange messages with any client in the group.

The IP system supports peer-to-peer multicast groups by allowing any device on the network to accept and send packets destined for the multicast group IP address. By default, there are no restrictions on which clients can join a multicast group. Some implementations use encryption to prevent unauthorized clients from interpreting the data received in the multicast group, but there is still no way to block the clients' receipt of the data.

Central Server

The other multicast system employs a central server, a single device on the network that controls all multicast group activity. An individual client wanting to join the multicast group must ask permission from the central server. If the central server denies the client access to that multicast group, no multicast packets will be forwarded to the requesting client. This technique is shown below.



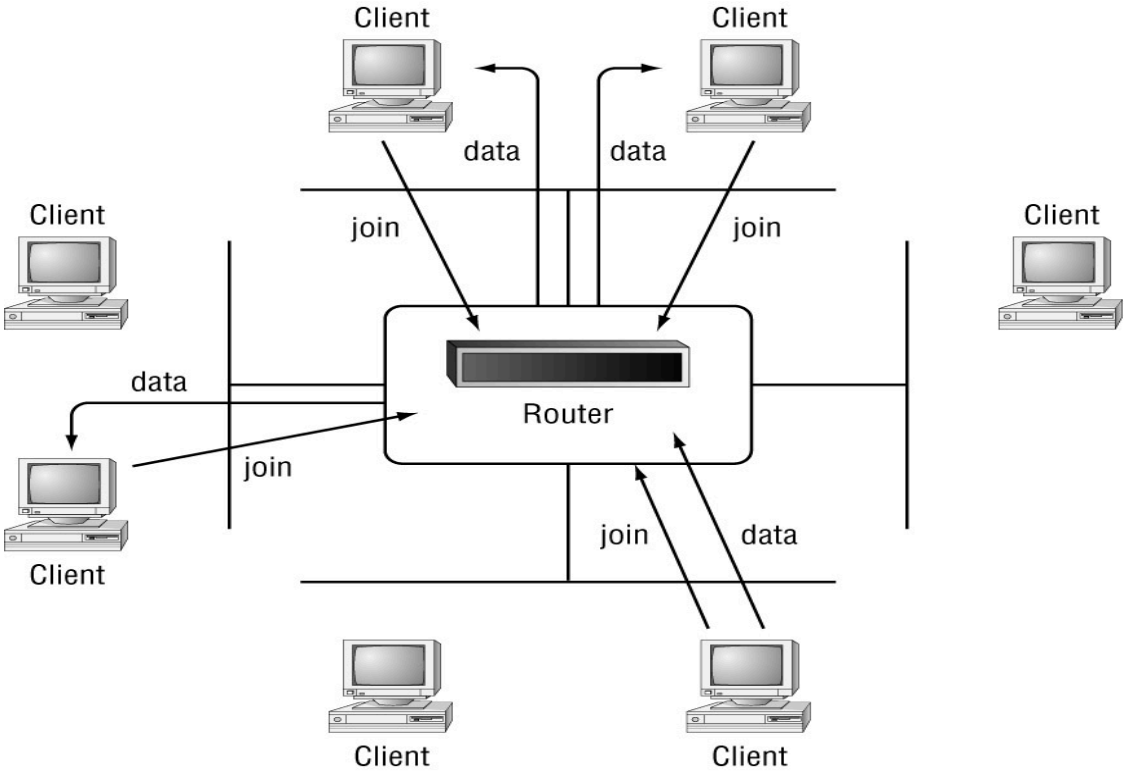
The central server setup for controlling multicast groups

The central server multicast group system is not supported by IP. Currently, Asynchronous Transfer Mode (ATM) networks are the only networks capable of supporting central server multicast groups.

Sending Multicast Packets through Routers

By default, most routers do not pass multicast packets through to other subnets. If a router passed every received multicast packet to every interface, it would put the network at risk of being flooded with multicast packets. Instead, a system has been developed to allow selective forwarding of multicast packets.

The Internet Group Management Protocol (IGMP) was developed to aid in notifying routers when multicast packets should to be passed to various subnets. When a network device wants to join a multicast group, it sends an IGMP packet to the local router on its subnet. The IGMP packet registers the network device and the multicast group address from which that device must receive messages. This enables the router to know that it must forward any received multicast messages for that group to the subnet of the specified network device.



Network devices registering multicast group memberships on a router

When each network device registers its intention to receive packets for the multicast group, the router must forward any received multicast packets to each interface that contains a registered host. Interfaces that do not contain registered hosts do not need to receive the multicast packet.

Similarly, when a network host leaves the multicast group, another IGMP packet is sent to the router, notifying it that the host no longer needs to have packets for that multicast group forwarded. When the last network host on a specific router interface leaves the group, the router can stop forwarding multicast packets to that interface.

References:

- ✔ C# Network Programming, Richard Blum, ISBN:0782141765, Sybex, 2003
  - o Chapter 10