

MULTICAST NETWORK PROGRAMMING

- Unicast is point-to-point communication.
- Broadcast is communication to all connected members.
- Multicast is communication to a selected set of connected members.

Multicast Applications

- Videoconferencing
- Usenet news
- Computer configuration

Multicast Addresses and Groups

- IP address range: **224.0.0.0** to **239.255.255.255** (i.e. first four bits is 1110)
- These are the class D addresses.
- A multicast group is a set of host that shares a multicast address.
- To create a multicast group, select a random address from **225.0.0.0** to **238.255.255.255**. After which, create an *InetAddress* object for that address and start sending it data.
- Try pinging **all-routers.mcast.net** to check multicasting capability of the network.

Common Permanent Multicast Addresses

- 224.0.0.1 (all-systems.mcast.net) – all systems on the local subnet.
- 224.0.0.2 (all-routers.mcast.net) – all routers on the local subnet.
- 224.0.0.11 (mobile-agents.mcast.net) – mobile agents on the local subnet.
- 224.0.1.1 (ntp.mcast.net) – the network time protocol.
- 224.0.1.20 (experiment.mcast.net) – experiments that do not beyond the local subnet
- 224.2.X.X (Multicast Backbone on the Internet (MBONE)) – used for audio and video broadcasts over the Internet.

Sending Multicast Data

- Communication is connectionless. Datagrams are simply UDP datagrams addressed to a multicast group.
- Need to create a *MulticastSocket* and put the multicast address in the Datagram packet that is to be sent.
- Need to set the Time-to-Live (TTL) value properly for the datagram to be received properly. TTL (0 – 255) is approximately the number of routers a frame goes through before it is discarded.
- Approximate TTL values are:
 - 0 for localhost; 1 for local subnet; 32 for high-bandwidth site in the USA; 48 for the USA; 64 for North America, 255 for worldwide.

- When the datagram is sent, it is the responsibility of the multicast routers to duplicate the data and route it to the multicast group members.

Java MulticastSocket Class

- The MulticastSocket class is a subclass of the DatagramSocket class:

```
Public class MulticastSocket extends DatagramSocket
```

- Constructors:

```
MulticastSocket ( ) throws SocketException  
MulticastSocket ( int portNum ) throws SocketException
```

- Methods:

```
public void joinGroup(InetAddress mAddr) throws  
IOexception
```

used to join a multicast group using the address, mAddr. Here is an example:

```
import java.net.*;  
import java.io.*;  
public class MulticastJoin {  
    public static void main(String [ ] args){  
        try {  
            MulticastSocket mSocket = new MulticastSocket(4001);  
            InetAddress mAddr = InetAddress.getByByName("224.0.0.1");  
            mSocket.joinGroup(mAddr);  
            byte [ ] buffer = new byte[512];  
            while (true) {  
                DatagramPacket dp = new DatagramPacket(buffer,  
                                                         buffer.length);  
                mSocket.receive(dp);  
                String str = new String(dp.getData(), "8859_1");  
                System.out.println(str);  
            }  
        }  
    }  
    catch (SocketException se){  
        System.out.println("Socket Exception : " + se); }  
    catch (IOException e) { System.out.println("Exception : " + e); }  
} //end of main  
} // end of class definition
```

```
public void leaveGroup(InetAddress mAddr) throws
                                IOException
```

used to signal the router that the computer is no longer interested in receiving multicast datagrams.

```
public void send(DatagramPacket packet, byte TTL) throws
                IOException
public void send(DatagramPacket packet) throws
                IOException
```

used to send the previously configured datagram packet with or without a specified time-to-live value. Here is an example:

```
import java.net.*;
import java.io.*;
public class MulticastSend {
    public static void main( String [ ] args) {
        try {
            MulticastSocket mSocket = new MulticastSocket( );
            InetAddress mAddr = InetAddress.getByName("224.0.0.1");
            String hostname = InetAddress.getLocalHost().getHostName();
            String sendString = "Hello from " + hostname;
            byte [ ] buffer = sendString.getBytes();

            DatagramPacket dp = new
                DatagramPacket(buffer, buffer.length, mAddr, 4001);

            mSocket.send(dp);
        }
        catch (SocketException se) {
            System.out.println("Socket Exception : " + se);
        }
        catch (IOException e) {
            System.out.println("Exception : " + e);
        }
    }

} //end of main
} // end of class definition
```

```
public void setInterface(InetAddress addr) throws  
SocketException
```

used to select the network interface to be used for multicasting in a multihomed system.

```
public InetAddress getInterface() throws  
SocketException
```

used to determine the address of the interface involved in multicasting.

```
public void setTimeToLive(int TTL) throws IOException
```

```
public int getTimeToLive( ) throws IOException  
sets/gets the time-to-live value of the datagram packet.
```

Here is one final example of multicasting:

```
import java.net.*;
import java.io.*;
public class MulticastListener {
    public static void main( String [ ] args) {
        InetAddress mAddr=null;
        MulticastSocket mSocket=null;
        final int PORT_NUM= 4001;
        try {
            mAddr = InetAddress.getByName("audionews.mcast.net");
            mSocket = new MulticastSocket(PORT_NUM);
            String hostname = InetAddress.getLocalHost().getHostName();
            byte [ ] buffer = new byte[8192];
            mSocket.joinGroup(mAddr);
            System.out.println("Listening from " + hostname + " at " +
                               mAddr.getHostName());

            while (true){
                DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
                mSocket.receive(dp);
                String str = new String(dp.getData(), "8859_1");
                System.out.println(str);
            }//end of while
        }
        catch (SocketException se) {
            System.out.println("Socket Exception : " + se);
        }
        catch (IOException e) {
            System.out.println("Exception : " + e);
        }
        finally {
            if (mSocket != null){
                try {
                    mSocket.leaveGroup(mAddr);
                    mSocket.close();
                }
                catch (IOException e){ }
            }//end of if
        }//end of finally

    }//end of main
}
```