**TCP/IP, UDP & Multicasting Through Java's Socket**

**Part I - Basic Networking through Java**

**Introduction**

Since birth of network programming, it has been error-prone, difficult, and complex. The programmer had to know many details about the network and sometimes even the hardware. You usually needed to understand the various layers of the networking protocol, and there were a lot to different functions in each networking library concerned with connecting, packing, unpacking blocks of information, handshaking, etc. It was a difficult task. However, the concept of networking is not so difficult. You want to get some information from that machine over there and move it to this machine here, or vice-versa. Its quite similar to reading and writing files, except that the files exist on the remote machine.

One of Java's great strengths is painless networking. As much as possible, the underlying details of networking have been abstracted away. The programming model we use is that of the file programming model. In addition, Java's built-in multithreading is exceptionally handy when dealing with another networking issue: handling multiple connections at once. The java.net package includes classes that enable the programmer to easily pass data across networks.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Networking Concepts**

The primary function of the TCP/IP is to provide a point to point communication mechanism. One process on one machine communicates with the another process on another machine or within the same machine. This communication appears as two streams of data. One stream carries data from one process to the other, while the other carries data in the other direction. Each process can read the data that have been written by the other, and in normal conditions, the data received are the same, and in the same order, as when they are sent.

In order to tell one machine from another machine and to make sure that you are connected with the machine you want, there must be some way of uniquely identifying machines on a network. Early networks were satisfied to provide unique names for machines within the local network. However, Java works within the Internet, which requires a way to uniquely identify a machine from all the others in the world. This is accomplished with the IP ( Internet Protocol ) address, a 32 bit number.

IP Address in two forms :

* The DNS ( Domain Name Service ) form. Suppose, if my domain name is calsoftlabs.com and if I have a computer called Hari in my domain. Its domain name would be Hari.calsoftlabs.com.
* Alternatively, we can use dotted quad form, which is four numbers separated by dots, such as 199.2.24.246

In addition to the machine addresses provided by the Internet Protocol part of the network system, TCP/IP has a mechanism for identifying individual processes on a machine, analogous to an office block. The building has phone number , but each room inside is also identified by an extension number. When a call arrives at the building, it must be connected to the correct room for handling . Payment requests go to accounts payable, orders to sales, and so forth. In the TCP/IP system, the extension numbers are called ports, and they are represented by a 16-bit binary number. To communicate with the correct part of a particular computer, the sending machine must know both the machine address and the port number to which the message should be sent. Many common services have a dedicated port. Because some ports are reserved for common services, the programmer cannot use any port. Ports numbered under 1024 are often referred to as reserved ports, many of which are reserved for a specific program. It is important that you only attempt to use ports over number 1024.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

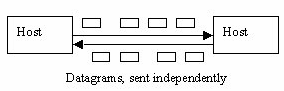
**Types Of Network Programming**

Two general types are :

* Connection-oriented programming
* Connectionless Programming

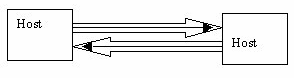
**Connection-oriented Networking**

The client and server have a communication link that is open and active from the time the application is executed until it is closed. Using Internet jargon, the Transmission control protocol os a connection oriented protocol. It is reliable connection - packets are guaranteed to arrive in the order they are sent.



**Connection-less Networking**

The this type each instance that packets are sent, they are transmitted individually. No link to the receiver is maintained after the packets arrive. The Internet equivalent is the User Datagram Protocol (UDP). Connectionless communication is faster but not reliable. Datagrams are used to implement a connectionless protocol, such as UDP.



**Common Services Port Number**

**Port Number**

**Service**

21

FTP

23

Telnet

25

SMTP(mail)

80

HTTP(Web)

119

NNTP(News)

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Client-Server Programming**

The most common model of network programming is referred to as client-server programming. The concept is simple: A client machine makes a request for information or sends a command to a server; in return, the server passes back the data or results of the command. Most often, the server only responds to clients; it does not initiate communication.

So the job of the server is to listen for a connection, and that's performed by the special server object that we create. The job of the client is to try to make a connection to the server, and this is performed by the special client object we create. Once the connection is made, you will see that at the server and client ends, the connection is magically just turned into the IO Stream object, and from then onwards you can treat the connection as if you were reading and writing into the file.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Socket Terminology**

The socket is a software abstraction used to represent the "terminals" of a connection between two machines or processes. For a given connection , there's a socket on each machine, and you can imagine a hypothetical "cable" running between the two machines with each end of the "cable" plugged into the socket. Of course, the physical hardware and cabling between machines is completely unknown.

In Java, we need to create a socket to make the connection to the other machine. Then you can get an *InputStream* and *OutputStream* from the socket in order to be able to treat the connection as an IOStream object. There are two stream based socket classes in the java.net package. They are java.net.*ServerSocket* that a server uses to listen for incoming connections and a java.net.Socket that a client uses in order to initiate a connection. Once a client makes a Socket connection, the ServerSocket returns a corresponding server side socket through which direct communications will take place.

When we create a ServerSocket, you give it only a port number. You don't have to give it an IP address because it's already on the machine it represents. When you create a Socket, however, you must give both the IP address and the port number where you're trying to connect.

**Socket Classes**

Socket  
ServerSocket  
DatagramSocket  
MulticastSocket( explained in the Part II of this paper )

Before discussing the constructors and methods of Socket and ServerSocket, the class InetAddress must be mentioned. An InetAddress represents the actual number, not the name or IP address of a computer. The name Hari.calsoftlabs.com is never used by your program; instead it uses the corresponding address, 199.2.24.246. The InetAddress class has no constructors, instead it has some methods that returns the InetAddress address.

**Socket**

Socket object is the Java representation of a TCP connection. when a socket is created, a connection is opened to the specified destination.

*Constructors:*The Socket provides the programmer with four constructors. The address of the server may be specified as a string or an InetAddress, and the port number on the host to connect to. In each case, an optional Boolean parameter implements a connectionless socket if set to false.

*Methods:*The two most important methods are getInputStream() and getOutputStream(), which return stream objects that can be used to communicate through the socket. A close() method is provided to tell the underlying operating system to terminate the connection. Methods are also provided to retrieve information about the connection to the local host and remote port numbers and an integers representing the remote host.

**ServerSocket**

The ServerSocket represents a listening TCP connection. Once an incoming connection is requested, the ServerSocket object will return a Socket object representing the connection.

*Constructors:*The ServerSocket provides two constructors. Both take argument the local port number to listen for connection requests. A constructor is provided that also takes the maximum time to wait for a connection as a second argument.

*Methods:*The most important method is accept(). It returns a Socket that is connected to the client. The close() method tells the operating system to stop listening for requests on the socket. Methods to retrieve the host name, the socket is listening on and the port number being listened to are also provided.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Datagrams**

Datagrams are used to implement a connectionless protocol, such as UDP. Two classes are used to implement datagrams in Java:

* java.net.DatagramPacket
* java.net.DatagramSocket

DatagramPacket is the actual packet of information, an array of bytes, that is transmitted over the network. DatagramSocket is a socket that sends and receives DatagramPackets across the network. You can think of the DatagramPacket as a letter and a DatagramSocket as the mailbox that the mailcarrier uses to pick up and drop off your letters.

**DatagramPacket**

The DatagramPacket class provides the programmer with two constructors. The first is used for DatgramPackets that receive information. This constructor needs to be provided with an array to store the data and the amount of data to receive. The second is used to create DatagramPackets that send data. The constructor requires the same information, plus the destination address and the port number.

*Methods:* There are four methods in this class - allowing the data, datagram length, and addressing (InetAddress) and port number information for the packet to be extracted.

**DatagramSocket**

The DatgramSocket represents a connectionless datagram socket. This class works with the DatagramPacket class to provide for communication using the UDP protocol. It provides two constructors, the programmer can specify a port to use or allow the system to randomly use one.

*Methods:* The two most important methods are - send() and receive(). Each takes as an argument an appropriately constructed DatagramPacket. In the case of the send() method, the data contained in the packet is sent to the specified host and the port. The receive() method will block the execution until a packet is received by the underlying socket, at which time the data will be copied into the packet provided.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Error handling**

Error handling is done by the class called *SocketException* which extends IOException class. This exception is thrown when there is a problem using socket. (i.e. ) error in the underlying protocol, such as a TCP error.

One possible cause is that the local port you are using for is already in use. Another cause is that the user cannot bind to that particular port. Because, on most operating systems, port numbers less than 1,024 cannot be used by the programmer except the super user. This is the security measure, because most well known services reside on the ports in this range.

**Some New SocketExceptions**

All the new exceptions extends the SocketException class: They are as follows:

*BindException:*The local port is in use, or the requested bind address couldn't be assigned locally.

*ConnectException:*This exception is raised when a connection is refused at the remote host (i.e., no process is listening on that port).

*NoRouteToHostException:*The connect attempt timed out, or the remote host is otherwise unreachable.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**A Simple Server and Client Examples**

A simple client and the server programs are explained for all the types of techniques. A code samples for the connection-oriented, next for the connectionless and then for broadcasting are as follows.

**Connection-oriented Client and Server : ( TCP )**

The client sends the message *" Hi from client "* and the server response will be *" Hello from Server ".*

**Server Program**

import java.net.\*;

import java.lang.\*;

import java.io.\*;

public class Server{

//port number should be more than 1024

public static final int PORT = 1025;

public static void main( String args[])

{

ServerSocket sersock = null;

Socket sock = null;

System.out.println(" Wait !! ");

try

{

// Initialising the ServerSocket

sersock = new ServerSocket(PORT);

// Gives the Server Details Machine name,

Port number

System.out.println("Server Started :"+sersock);

try

{

// makes a socket connection to particular

client after

// which two way communication take place

sock = sersock.accept();

System.out.println("Client Connected :"+ sock);

// Receive message from client i.e Request

from client

DataInputStream ins = new

DataInputStream(sock.getInputStream());

// Send message to the client i.e Response

PrintStream ios = new

sPrintStream(sock.getOutputStream());

ios.println("Hello from server");

ios.close();

// Close the Socket connection

sock.close();

}

catch(SocketException se)

{

System.out.println("Server Socket

problem "+se.getMessage());

}

catch(Exception e)

{

System.out.println("Couldn't start "

+ e.getMessage()) ;

}

// Usage of some methods in Socket class

System.out.println(" Connection from : " +

sock.getInetAddress());

} // main

} // Server class

**Client Program**

import java.lang.\*;

import java.io.\*;

import java.net.\*;

import java.net.InetAddress;

class client

{

Public static void main(String args[])

{

Socket sock=null;

DataInputStream dis=null;

PrintStream ps=null;

System.out.println(" Trying to connect");

try

{

// to get the ip address of the

server by the name

InetAddress ip =InetAddress.getByName

("Hari.calsoftlabs.com");

// Connecting to the port 1025 declared

in the Serverclass

// Creates a socket with the server

bind to it.

sock= new Socket(ip,Server.PORT);

ps= new PrintStream(sock.getOutputStream());

ps.println(" Hi from client");

DataInputStream is = new

DataInputStream(sock.getInputStream());

System.out.println(is.readLine());

}

0catch(SocketException e)

{

System.out.println("SocketException " + e);

}

catch(IOException e)

{

System.out.println("IOException " + e);

}

// Finally closing the socket from

the client side

finally

{

try

{

sock.close();

}

catch(IOException ie)

{

System.out.println(" Close Error :" +

ie.getMessage());

}

} // finally

} // main

} // Class Client

**Running the Server and Client**

After you've successfully compiled the server and the client programs, you run them. You have to run the server program first. Just use the Java interpreter and specify the Server class name. Once the server has started, you can run the client program. After the client sends a request and receives a response from the server, you should see output similar to this :

On client side:

Hello from Server

On Server side:

Hi from client

**Connectionless Client and Server : (UDP)**

A datagram is an independent, self-contained message sent over the network whose arrival, arrival time, and content are not guaranteed.

The java.net package contains two classes to help you write Java programs that use datagrams to send and receive packets over the network: DatagramSocket, DatagramPacket, and MulticastSocket An application can send and receive DatagramPackets through a DatagramSocket. In addition, DatagramPackets can be broadcast to multiple recipients all listening to a MulticastSocket.

The following source code demonstrates a slightly more complex server that uses datagrams instead of sockets. After a connection is made, It echoes back whatever is sent by the client instead of simply hanging up the connection. It is Called as echo Server.

**Server Program**

import java.net.\*;

import java.io.\*;

public class EchoServer

{

//Initialize Port number and Packet Size

static final int serverPort = 1026;

static final int packetSize = 1024;

public static void main(String args[])

throws SocketException{

DatagramPacket packet;

DatagramSocket socket;

byte[] data; // For data to be

Sent in packets

int clientPort;

InetAddress address;

String str;

socket = new DatagramSocket(serverPort);

for(;;){

data = new byte[packetSize];

// Create packets to receive the message

packet = new DatagramPacket(data,packetSize);

System.out.println("Waiting to receive

the packets");

try{

// wait infinetely for arrive of the packet

socket.receive(packet);

}catch(IOException ie)

{

System.out.println(" Could not Receive

:"+ie.getMessage());

System.exit(0);

}

// get data about client in order to

echo data back

address = packet.getAddress();

clientPort = packet.getPort();

// print string that was received

on server's console

str = new String(data,0,0,packet.getLength());

System.out.println("Message :"+ str.trim());

System.out.println("From :"+address);

// echo data back to the client

// Create packets to send to the client

packet = new DatagramPacket(data,packetSize,

address,clientPort);

try

{

// sends packet

socket.send(packet);

}

catch(IOException ex)

{

System.out.println("Could not Send

"+ex.getMessage());

System.exit(0);

}

} // for loop

} // main

} // class EchoServer

**Client Program**

import java.net.\*;

import java.io.\*;

public class EchoClient{

static final int serverPort = 1026;

static final int packetSize = 1024;

public static void main(String args[]) throws

UnknownHostException, SocketException{

DatagramSocket socket; //How we send packets

DatagramPacket packet; //what we send it in

InetAddress address; //Where to send

String messageSend; //Message to be send

String messageReturn; //What we get back

from the Server

byte[] data;

//Checks for the arguments that sent to

the java interpreter

// Make sure command line parameters correctr

if(args.length != 2)

{

System.out.println("Usage Error :

Java EchoClient < Server name> < Message>");

System.exit(0);

}

// Gets the IP address of the Server

address = InetAddress.getByName(args[0]);

socket = new DatagramSocket();

data = new byte[packetSize];

messageSend = new String(args[1]);

messageSend.getBytes

(0,messageSend.length(),data,0);

// remember datagrams hold bytes

packet = new

DatagramPacket(data,data.length,address,serverPort);

System.out.println(" Trying to Send the packet ");

try

{

// sends the packet

socket.send(packet);

}

catch(IOException ie)

{

System.out.println("Could not Send :"+ie.getMessage());

System.exit(0);

}

packet is reinitialized to use it for recieving

packet = new DatagramPacket(data,data.length);

try

{

// Receives the packet from the server

socket.receive(packet);

}

catch(IOException iee)

{

System.out.println("Could not receive :

"+iee.getMessage() );

System.exit(0);

}

// display message received

messageReturn = new String (packet.getData(),0);

System.out.println("Message Returned : "+

messageReturn.trim());

} // main

} // Class EchoClient

**Running the Server and Client**

The client side and the server side networking code looks actually very similar.This is true with many applications that use datagrams because the java.net.DatagramSocket class is used to both send and receive DatagramPackets.

Suppose server running on the machine named Hari.calsoftlabs.com, whereas the client running on the xxxx.info.com. As you can see at the end of the example the server is running waiting for the another connection, while the execution of the client has halted.

Server Side :

To start the server :

Java EchoServer

Output:

Message:Hello

From :xxxx.info.com

Client Side :

Java EchoClient Hari.calsoftlabs.com Hello

Output:

Message Returned : Hello

**Conclusion**

In this paper introduction to full spectrum of techniques in java networking are explained. We have seen techniques for programming clients and servers using both TCP and UDP services. If you have previous experience of network programming, you will probably agree that java makes network programming much easier than do most other languages. Thus, one of Java's great strengths is painless networking. As much as possible, the underlying details of networking have been abstracted away.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Part II - Multicasting Through Java**

**Introduction**

Multicasting, is the Internet's version of broadcasting. A site that multicasts information is similar in many ways to a television or radio station that broadcasts its signal. The signal originates from one source, but it can reach everyone in the station's signal area. The information passes on by those who don't want to catch the signal or don't have the right equipment.

Broadcasting is generally suited to any applications that requires a number of machine in a distributed group to receive the same data; for example conferencing, group mail and news distribution, and network management.

Most high-level network protocols only provide a unicast transmission service. That is, nodes of the network only have the ability to send to one other node at a time. All transmission with a unicast service is inherently point-to-point. If a node wants to send the same information to many destinations using a unicast transport service, it must perform a replicated unicast, and send N copies of the data to each destination in turn.

A better way to transmit data from one source to many destinations is to provide a multicast transport service. With a multicast transport service, a single node can send data to many destinations by making just a single call on the transport service:

For those applications which involve a single node sending to many recipients, a multicast facility is clearly a more natural programming paradigm than unicast. When a multicast service is implemented over such a network, there is a huge improvement in performance. If the hardware supports multicast, A packet which is destined for N recipients can be sent as just a single packet!

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Multicast Groups**

The notion of *"group"* is essential to the concept of multicasting. By definition a multicast message is sent from a source to a group of destination hosts. In IP multicasting, multicast groups have an ID called multicast group ID. Whenever a multicast message is sent out, a multicast group ID specifies the destination group. These group ID's are essentially a set of IP addresses called "Class D" explained in the following section. Therefore, if a host (a process in a host) wants to receive a multicast message sent to a particular group, it needs to somehow listens to all messages sent to that particular group. Different Ipv4 addresses are explained in the following section.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Different types of IPv4 Addresses**

There are three types of IPv4 addresses: unicast, broadcast, and multicast.

*Unicast* addresses are used for transmitting a message to a single destination node.

Broadcast addresses are used when a message is supposed to be transmitted to all nodes in a subnetwork.

For delivering a message to a group of destination nodes which are not necessarily in the same subnetwork,*multicast* addresses are used.

Class A, B, and C IP addresses are used for unicast messages, whereas as class D IP address, those in the range 224.0.0.1 to 239.255.255.255, inclusive, and by a standard UDP port number are used for multicast messages.

**IP Addresses of Class D - Multicasting**

IP addresses of Class D have the following format:

Bit no. 0 1 2 3 4 5 6 7 8 16 24 31

Class D 1 1 1 0 |---------multicast address(28)--------|

Class D addresses are identified by a one in bit 0,1 and 2 and a zero in bit 3 of the address. This means that 6.25% of all available IP addresses are of this class.

The range of Class D addresses are in dotted decimal notation from 224.h.h.h.h to 239.h.h.h, where h is a number from 0 to 255. Address 224.0.0.0 is reserved and can not be used, while address 224.0.0.1 is used to address all hosts that take part in IP multicasting.

Class D addresses are used for multicasting and does not have a network part and hosts part. IP multicasting makes it possible to send IP datagrams to a group of hosts, which may be spread across many networks.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Life of Multicast Packets (TTL's)**

Broadcast packets need to have a finite life inorder to avoid bouncing of the packets around the network forever. Each packet has a time to live (TTL) value, a counter that is decremented every time the packet passes through an hop i.e a router between the network. Because of TTLs, each multicast packet is a ticking time bomb.

Take for example, a TV station where TTLs would be the station's signal area -- the limitation of how far the information can travel. As the packet moved around the company's internal network, its TTL would be notched down every time it passed through an router. When the packet's TTL reached 0, the packet would die and not be passed further. Generally multicast with long TTLs -- perhaps 200 - to guarantee that the information will reach around the world

**Java's Support**

Java makes the programmer easier by implementing classes in java.net package that facilitates our need i.e multicasting. java.net includes a class called *MulticastSocket.* This kind of socket is used on the client-side to listen for packets that the server broadcasts to multiple clients. The multicast datagram socket class is useful for sending and receiving IP multicast packets. Thus, java.net.*MulticastSocket.* and java.net.*DatagramPacket* together used to implement multicasting in java and makes programming easier.

A MulticastSocket is a (UDP) DatagramSocket, with additional capabilities for joining "groups" of other multicast hosts on the internet. One would join a multicast group by first creating a MulticastSocket with the desired port, then invoking the *joinGroup*(InetAddress groupAddr) method. One can leave a group by using the method *leaveGroup*(InetAddress groupAddr).

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**A Simple Server and Client Examples**

**MulticastSocket class**

Constructors:

public MulticastSocket() throws IOException

This allows us to Create a multicast socket.

public MulticastSocket(int port) throws IOException

This allows us to Create a multicast socket and

bind it to a specific port.

*Methods:*Methods other than the joinGroup(addr) and leaveGroup(addr), this class also provides functions, that set and gets the time-to-live for multicast packets sent out on the socket. The TTL sets the IP time-to-live for DatagramPackets sent to a MulticastGroup, which specifies how many "hops" that the packet will be forwarded on the network before it expires. They are setTTl(ttl) and getTTl(ttl). The parameter 'ttl' is an unsigned 8-bit quantity.

All the methods of this class throws IOEXception.

*Limitations:*Currently applets are not allowed to use multicast sockets.

**A Simple Example that explains multicasting**

The Following source code is an simple example for broadcasting. It broadcast date and time to its multiple clients. ( from java 1.1 developers handbook )

**Server Program**

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class BroadcastServer

{

public static final int PORT = 1200;

public static void main(String args[])

throws Exception{ MulticastSocket socket;

DatagramPacket packet; InetAddress address;

address = InetAddress.getByName();

socket = new MulticastSocket();

// join a Multicast group and send the

group salutations socket.joinGroup(address);

byte[] data = null;

for(;;)

{

Thread.sleep(1000);

System.out.println("Sending ");

String str = (new Date()).toString();

data = str.getBytes();

packet = new DatagramPacket

(data,str.length(),address,PORT);

// Sends the packet socket.send(packet);

}

//

for

}

// main

} // class BroadcastServer

**Client program**

import java.net.\*;

import java.io.\*;

public class BroadcastClient{

public static final int PORT = 1200;

public static void main(String args[])

sthrows Exception{

MulticastSocket socket;

DatagramPacket packet;

InetAddress address;

address = InetAddress.getByName(args[0]);

socket = new MulticastSocket(BroadcastServer.PORT);

//join a Multicast group and send the

group salutations

socket.joinGroup(address);

byte[] data = null;

packet = new DatagramPacket(data,data.length);

for(;;)

{

// receive the packets

socket.receive(packet);

String str = new String(packet.getData());

System.out.println(" Time signal received from"+

packet.getAddress() + " Time is : " +str);

} // for

} // main

} // class Broadcast

**Running the Server and Client**

First you will need to ensure that a name is associated with a suitable multi-cast address. This requires access to - and understanding of your system's configuration. Add a suitable name, such as BroadTest and assign it the address between 224.0.0.1 to 239.255.255.255.

Now run the server program by issuing the command:

Java BroadcastServer BroadTest.

You will see a message "Sending " appear at one-second intervals. This suggests that a server is running.

Run the client side by issuing the command:

Java BrodcastClient BroadTest.

You should now see the date and time printed at regular intervals on the client machine

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**Advantages of Multicasting**

In many cases the multicasting capability is desirable because of some following advantages.

The first major advantage of using multicasting is the *decrease of the network load.* There are many applications like stock ticker applications which are required to transmit packets to hundreds of stations. The packets sent to these stations share a group of links on their paths to their destinations. Since multicasting requires the transmission of only a single packet by the source and replicates this packet only if it is necessary multicast transmission can conserve the so much needed network bandwidth.

Another place where multicasting can be very helpful is in *resource discovery.* There are many applications in which a host needs to find out whether a certain type of service is available or not. Using multicast messages and sending the query to those hosts which are potentially capable of providing this service would be of great help. Although some applications use multicast messages to transmit a packet to a group of hosts residing on the same network, there is no reason to impose this limitation. The scope of multicast packets can be limited by using the time-to-live (TTL) field of these packets.

Another important feature of multicasting is its support for *datacasting* applications. In recent years, multimedia transmission has become more and more popular. The audio and video signals are captured, compresses and transmitted to a group of receiving stations. Instead of using a set of point-to-point connections between the participating nodes, multicasting can be used for distribution of the multimedia data to the receivers. In real world stations may join or leave an audio-cast or a video-cast at any time.

Another feature of multicasting is *flexibility* in joining and leaving a group provided by multicasting can make the variable membership much easier to handle.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**MBONE (Multicast Bone)**

A brief explanation of the MBONE network is given in this section.

Today, the MBONE is a critical piece of the technology that's needed to make multiple person data voice, and video conferencing on the Internet -- in fact, sharing any digital information -- cheap and convenient. Unfortunately, the majority of the routers on the Internet today don't know how to handle multicasting. Most routers are set up to move traditional Internet Protocol (IP) unicast packets -- information that has a single, specific destination. Most routers today are unicast routers: They are designed to move information from a specific place to another specific place. However, routers that include multicasting capabilities are becoming more common.

So in order to facilitate multicasting MBONE network was created. It's a *"virtual network"* - a network that runs on top of the Internet and it allows multicast packets to traverse the Net. The MBONE is called a virtual network because it shares the same physical media -- wires, routers and other equipment -- as the Internet.

The MBONE allows multicast packets to travel through routers that are set up to handle only unicast traffic. Software that utilizes the MBONE hides the multicast packets in traditional unicast packets so that unicast routers can handle the information.The scheme of moving multicast packets by putting them in regular unicast packets is called tunneling. In the future, most commercial routers will support multicasting, eliminating the headaches of tunneling information through unicast routers.

Machines (workstations or routers) that are equipped to support multicast IP are called mrouters (multicast routers). Mrouters are either commercial routers that can handle multicasting or (more commonly) dedicated workstations running special software that works in conjunction with standard routers. *Multicasting is a network routing facility -- a method of sending packets to more than one site at a time.* The MBONE is a loose confederation of sites that currently implement IP multicasting.

**MBONE Advantages**

You can use the MBONE to do the following:

* Collaborate with colleagues in real time by using a shared virtual "whiteboard"
* Hear and see live lectures from famous professors or scientists, and even ask them questions
* Listen to radio stations that "broadcast" on the Internet
* Start your own radio show
* See live pictures of spacebound NASA astronauts on the space shuttle
* Attend a virtual poetry reading where you hear the words in the author's own voice
* See and hear rehearsals of Silicon Valley garage bands
* Attend an Internet Engineering Task Force meeting without leaving your office

In the future, the MBONE may make it possible for you to do the following:

* Engage 5,000 other people in a huge intercontinental computer game
* See reruns of "Gilligan's Island" and share your snide comments in real time with faraway friends
* Put your own garage band's rehearsals online for all to see (and hear)
* Automatically download and install authorized upgrades and bug-fixes to your computer software, without your intervention
* "Chat" in real time with 20 other users (as you can with Internet Relay Chat, except that you'll use your voice instead of your overworked fingers)
* Do plenty of other things that haven't been thought of yet

**Conclusion**

In this paper, we first reviewed why and for what applications multicasting is important. Then, the essential concept of group membership was discussed. Here a sample program in java for broadcasting shows the easiness to implement such a complex muticasting. Thus java with it's MulticastSocket class and methods provides the easier way to implement the multicasting by abstracting the underlying details of networking. It also introduces the concept of MBONE and it's advantages.

[[Click here to go to Top](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)](http://www.calsoftlabs.com/whitepapers/java-networking.html#top)

**References**

* JAVA UNLEASHED by Sams.net Publishing
* Java 1.1 Developers Handbook by Heller and Roberts.
* Thinking in Java, by bruce Eckel
* JAVA ( Techmedia )
* S