Multicasting

# 1. What is multicasting?

Say you need to send some data to multiple users. You can do it in two ways. Unicast is one method of doing it: just sending the same packet to each and every users. So, if there are n number of users, you will be sending it n number of times. This is inefficient and costly, as traffic goes linearly. The other method is Multicast which allows you to send the packet just once, and let the routers take care of splitting the traffic as needed.

Connection-oriented programming/networking: A link that is open and active from the time the application is executed until it is closed. TCP (Transmission Control Protocol) is a connection oriented protocol. It is reliable connection - packets are guaranteed to arrive in the order they are sent.

Connectionless programming/networking: In 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.

# 2. 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 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!

# 3. 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 listen to all messages sent to that particular group. Different IP addresses are explained in the following section.

# 4. Different types of IP addresses

Before we look at different types of IP addresses, let’s get a brief idea about IP address itself.

An IP (Internet Protocol) address is a unique number used to label any device connected to a network on which the Internet Protocol is used as the medium for communication. It was developed in the 1970s and it defines everything about how devices on the internet exchange information.

The IP address is, in turn, one of the cornerstones of the Internet Protocol. Information is transmitted over the network in discrete chunks called packets; each packet is mostly made up of whatever data the sender is trying to communicate, but also includes a header, consisting of metadata about that packet.

Among other pieces of data stored in the packet header are the IP address of the device that sent the packet and the IP address of device where the packet is heading. Routers and other network infrastructure use this information to make sure the packets get to where they’re supposed to go.

There are two primary types of IP address formats used today — IPv4 and IPv6.

## IPv4:

An IPv4 address consist of four sets of numbers from 0 to 255, separated by three dots. For example, the IP address of TechTerms.com is 67.43.14.98. This number is used to identify the TechTerms website on the Internet. When you visit http://techterms.com in your web browser, the DNS system automatically translates the domain name "techterms.com" to the IP address "67.43.14.98."

There are three classes of IPv4 address sets that can be registered through the InterNIC. The smallest is Class C, which consists of 256 IP addresses (e.g. 123.123.123.xxx — where xxx is 0 to 255). The next largest is Class B, which contains 65,536 IP addresses (e.g. 123.123.xxx.xxx). The largest block is Class A, which contains 16,777,216 IP addresses (e.g. 123.xxx.xxx.xxx).

The total number of IPv4 addresses ranges from 000.000.000.000 to 255.255.255.255. Because 256 = 28, there are 28 x 4 or 4,294,967,296 possible IP addresses. While this may seem like a large number, it is no longer enough to cover all the devices connected to the Internet around the world. Therefore, many devices now use IPv6 addresses.

## IPv6:

The IPv6 address format is much different than the IPv4 format. It contains eight sets of four hexadecimal digits and uses colons to separate each block. An example of an IPv6 address is: 2602:0445:0000:0000:a93e:5ca7:81e2:5f9d. There are 3.4 x 1038 or 340 undecillion) possible IPv6 addresses, meaning we shouldn't run out of IPv6 addresses anytime soon.

There are three types of IP 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.

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

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.

# 5. Life of Multicast Packets (TTL’s)

Broadcast packets need to have a finite life in order to avoid bouncing around the network forever. Each packet has a time to live (TTL) value, a counter that is decremented every time the packet passes through a 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 a router. When the packet's TTL reached 0, the packet would die and not be passed further. Generally multicasts have long TTLs -- perhaps 200 - to guarantee that the information will reach around the world

# 6. Java and Multicasting

Java makes programming for multicasting 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, User Datagram Protocol) 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).

**Example 1**

A simple example of multicasting where the current date is being broadcast to its multiple clients.

## A: Server side:

**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*(args[0]);

socket = **new** MulticastSocket();

//join a Multicast group

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);

}//end for

}//end main

}//end class BroadcastServer

## B: The client:

**import** java.net.\*;

**import** java.io.\*;

**public** **class** BroadcastClient {

**public** **static** **final** **int** ***PORT*** = 1200;

**public** **static** **void** main(String args[])**throws** 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);

}//end for

}//end main

}//end class Broadcast

**Description:** The getByName() method of class InetAddress determines the IP address of a host, given the host's name.

Here are some basic guide lines to write a multicasting program:

## The receiving-socket:

1. //Import some needed classes

import sun.net.\*;

import java.net.\*;

1. //Which port should we listen to

int port = 5000;

//Which address

String group = "225.4.5.6";

1. //Create the socket and bind it to port ‘port’

MulticastSocket s = new MulticastSocket(port);

1. //Join the multicast group

s.joinGroup(InetAddress.getByName(group));

//Now the socket is set up and we are ready to receive packets

1. //Create a DatagramPacket and do a receive

byte buf[] = byte[1024];

DatagramPacket pack = new DatagramPacket(buf, buf.length);

s.receive(pack);

1. //Finally, let’s do something useful with data we just received, like print it ☺

System.out.println("Received data from: " + pack.getAddress().toString() +

":" + pack.getPort() + " with length: " +

pack.getLength());

System.out.write(pack.getData(),0,pack.getLength());

System.out.println();

1. //And when we have finished receiving data leave the multicast group and close the socket

s.leaveGroup(InetAddress.getByName(group));

s.close();

## The sending-socket:

1. //Import some needed classes

import sun.net.\*;

import java.net.\*;

1. //Which port should we send to

int port = 5000;

//Which address

String group = "225.4.5.6";

//Which ttl

int ttl = 1;

1. //Create the socket but we don’t bind it as we are only going to send data

MulticastSocket s = new MulticastSocket();

1. //Note that we don’t have to join the multicast group id we are only sending and not receiving
2. //Fill the buffer with some data

byte buf[] = byte[10];

for (int i=0; i<buf.length; i++) buf[i] = (byte)i;

1. //Create a DatagramPacket

DatagramPacket pack = new DatagramPacket(buf, buf.length,

InetAddress.getByName(group), port);

1. //Do a send. Note that send takes a byte for the ttl and not an int

s.send(pack,(byte)ttl);

1. //And when we have finished sending data close the socket

s.close();

# 7. Advantages of Multicasting

The first major advantage of 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 advantage of multicasting 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.

Multicasting can’t be implemented for applets.