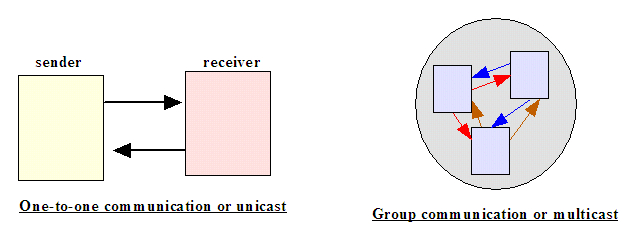
**Institute of Technology, Tralee**

**Computing Department**

Distributed Computing

Group Communication

In the IPC we have investigated so far, data is sent from a source process, the sender, to one destination process, the receiver. This is referred to as unicasting. Multicasting is defined as the sending of information to multiple receivers. Unicasting supports one to one IPC while multicasting supports one to many IPC.



An application program interface that supports multicasting must provide the following primitive operations.

* Join – This operation allows a process to join a specific multicast group. A process that has joined a multicast group is a member of the group and is entitled to receive all multicast addressed to the group. A process should be able to be a member of multiple multicast groups at any one time.
  + Note that for this and other multicast operations, a naming scheme is needed to uniquely identify a multicast group.
* Leave –This operation allows a process to stop participating in a multicast group. A process that has left a multicast group is no longer a member of the group and is thereafter not entitled to receive any multicast addressed to the group, although the process may remain a member of other multicast groups.
* Send – This operation allows a process to send a message to all processes currently participating in a multicast group.
* Receive –This operation allows a member process to receive messages sent to a multicast group.

When a multicast message is sent by a process, the runtime support of the multicast mechanism is responsible for delivering the message to each process currently in the multicast group.

# The Java Basic Multicast API

At the transport layer, the basic multicast supported by Java is an extension of UDP (the User Datagram Protocol), which is connectionless and unreliable. For the basic multicast, Java provides a set of classes which are closely related to the datagram socket API classes.

There are four major classes in the API, the first three of which we have already seen in the context of datagram sockets.

* **InetAddress:** In the datagram socket API, this class represents the IP address of the sender or receiver. In multicasting, this class can be used to identify a multicast group
* **DatagramPacket**: As with datagram sockets, an object of this class represents an actual datagram; in multicast, a DatagramPacket object represents a packet of data sent to all participants or received by each participant in a multicast group.
* **DatagramSocket**: In the datagram socket API, this class represents a socket through which a process may send or receive data.
* **MulticastSocket** : A MulticastSocket is a DatagramSocket, with additional capabilities for joining and leaving a multicast group. An object of the multicast datagram socket class can be used for sending and receiving IP multicast packets.

Instead of a single process, a multicast datagram is meant to be received by all the processes that are currently members of a specific multicast group. Hence each multicast datagram needs to be addressed to a multicast group instead of an individual process.

The Java multicast API uses the Internet Protocol (IP) multicast addresses for identifying multicast groups.

In IPv4a multicast group is specified by

(i) a class D IP address combined with

(ii) a standard UDP port number.

Class D IP addresses are those with the prefix bit string of 1110, and hence these addresses are in the range of 224.0.0.0 to 239.255.255.255, inclusive. Excluding the four prefix bits, there are 32-4=28 remaining bits, resulting in an address space of 228; that is, approximiate 268 million class D addresses are available, although the address 224.0.0.0 is reserved and should not be used by any application. IPv4 multicast addresses are managed and assigned by the Internet Assigned Numbers Authority (IANA)

An application which uses the Java multicast API must specify at least one multicast address for the application. To select a multicast address for an application, there are the following options:

1. Obtain a permanently assigned static multicast address from IANA: Permanent addresses are limited to global, well-known Internet applications, and their allocations are highly restricted.
2. Choose an arbitrary address
3. Obtain a transient multicast address at runtime; such an address can be received by an application through the Session Announcement Protocol.

For our examples and exercises, we will make use of the static address 224.0.0.1, with an equivalent domain name ALL-SYSTEMS.MCAST.NET, for processes running on all machines on the local area network, such as those in the laboratory; alternatively, we may use an arbitrary address that has not been assigned, such as a number in the range of 239.\*.\*.\* (for example, 239.1.2.3).

--HERE – 26/9

In the Java API, a MulticastSocket object is bound to a port address (e.g. 3456), and methods of the object allows for the joining and leaving of a multicast address such as 239.1.2.3

# Joining a multicast group

To join a multicast group at IP address m and UDP port p, a MulticastSocket object must be instantiated with p, then the object’s joinGroup method can be invoked specifying the address m:

// join a Multicast group at IP address 239.1.2.3 and port 3456

InetAddress group = InetAddress.getByName("239.1.2.3");

MulticastSocket s = new MulticastSocket(3456);

s.joinGroup(group);

# Sending to a multicast group

A multicast message can be sent using syntax similar with the datagram socket API.

String msg = "This is a multicast message.";

InetAddress group = InetAddress.getByName("239.1.2.3");

MulticastSocket s = new MulticastSocket(3456);

s.joinGroup(group); // optional

DatagramPacket hi = new DatagramPacket(msg.getBytes( ),msg.length( ),group, 3456);

s.send(hi);

# Receiving messages sent to a multicast group

A process that has joined a multicast group may receive messages sent to the group using syntax similar to receiving data using a datagram socket API.

byte[] buf = new byte[1000];

InetAddress group = InetAddress.getByName("239.1.2.3"); MulticastSocket s = new MulticastSocket(3456); s.joinGroup(group);

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

s.receive(recv);

# Leaving a multicast group

A process may leave a multicast group by invoking the leaveGroup method of a MulticastSocket object, specifying the multicast address of the group.

s.leaveGroup(group);

# Setting the “time-to-live”

The runtime support for a multicast API often employs a technique known as message propagation, whereby a packet is propagated from a host to a neighbouring host in an algorithm which, when executed properly, will eventually deliver the message to all the participants.

Under some anomalous circumstances, however, it is possible that the algorithm which controls the propagation does not terminate properly, resulting in a packet circulating in the network indefinitely.

Indefinite message propagation causes unnecessary overhead on the systems and the network. To avoid this occurrence, it is recommended that a “time to live” parameter be set with each multicast datagram.

The time-to-live (ttl) parameter, when set, limits the count of network links or hops that the packet will be forwarded on the network.

String msg = "Hello everyone!";

InetAddress group = InetAddress.getByName("224.0.0.1"); MulticastSocket s = new MulticastSocket(3456);

s.setTimeToLive(1); // set time-to-live to 1 hop – a count

// appropriate for multicasting to local hosts

DatagramPacket hi = new DatagramPacket(msg.getBytes( ),msg.length( ),group, 3456);

s.send(hi);

The value specified for the ttl must be in the range 0<=ttl<=255; an IllegalArgumentException will be thrown otherwise.

The recommended ttl settings are:

0 if the multicast is restricted to processes on the same host

1 if the multicast is restricted to processes on the same subnet

32 if the multicast is restricted to processes on the same site

64 if the multicast is restricted to is processes on the same region

128 is if the multicast is restricted to processes on the same continent

255 is the multicast is unrestricted