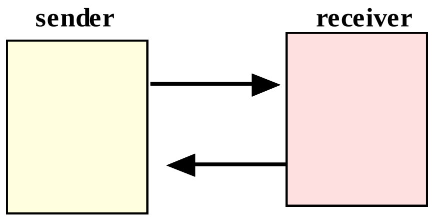
Multicast socket
Group Communication

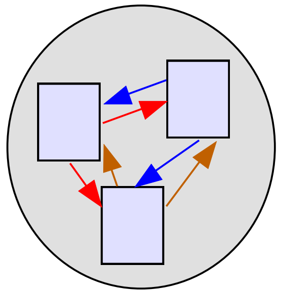
61FIT3NPR -Network Programming

Faculty of Information Technology
Hanoi University
Fall 2020

Unicast vs. Multicast



<u>**One-to-one communication or unicast**</u>



<u>**Group communication or multicast**</u>

Multicast

- Whereas the majority of network services and network applications use unicast for IPC, multicasting is useful for applications such as:

  online conferences, interactive distance learning, and can be used for applications such as online auction. It can also be used in replication of services for fault tolerance.

Mutlicast group

- In an application or network service which makes use of multicasting, a set of processes form a group, called a ***multicast group***. Each process in a group can send and receive message. A message sent by any process in the group can be received by each participating process in the group. A process may also choose to leave a multicast group.

- In an application such as online conferencing, a group of processes interoperate using multicasting to exchange audio, video, and/or text data.

An Archetypal Multicast API

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.

Multicast API Operations - continued

- **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.

Reliable Multicast vs. Unreliable Multicast

- 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.
- Due to factors such as failures of network links and/or network hosts, routing delays, and differences in software and hardware, the time between when a unicast message is sent and when it is received may vary among the recipient processes.

Reliable Multicast vs. Unreliable Multicast

- Moreover, a message may not be received by one or more of the processes at all, due to errors and/or failures in the network, the machines, or the runtime support.
- Whereas some applications, such as video conferencing, can tolerate an occasional miss or misordering of messages, there are applications – such as database applications – for which such anomalies are unacceptable.
- Therefore, when employing a multicasting mechanism for an application, it is important that you choose one with the characteristics appropriate for your application. Otherwise, measures will need to be provided in the coding of the application in order to handle the anomalies which may occur in message delivery.

Classification of multicasting mechanisms in terms of message delivery -1

Unreliable multicast: At its most basic, a multicast system will make a good-faith attempt to deliver messages to each participating process, but the arrival of the correct message at each process is not guaranteed.

In the best case, the correct message is received by all processes.

- In some cases, the message may be received by some but not all,
- The messages may be received by some processes in a corrupted form. Such a system is said to provide unreliable multicast.

Classification of multicasting mechanisms in terms of message delivery - 2

Reliable multicast: A multicast system which guarantees that each message is eventually delivered to each process in the group in uncorrupted form is said to provide **reliable multicast.**

Classification of multicasting mechanisms in terms of message delivery - 3

- The definition of reliable multicast requires that each participating process receives exactly one copy of each message sent. However, the definition places **no** restriction on the order that the messages are delivered to each process: each process may receive the messages in any permutation of those messages.
- For applications where the order of message delivery is significant, it is helpful to further classify reliable multicast systems based on the order of the delivery of messages.

Classification of reliable multicast

Unordered

An unordered reliable multicast system provides no guarantee on the delivery order of the messages.

Example: Processes P_1, P_2, and P_3 have formed a multicast group. Further suppose that three messages, m_1, m_2, m_3 have been sent to the group. Then an unordered reliable multicast system may deliver the messages to each of the three processes in any of the 3! = 6 permutations (m_1-m_2-m_3, m_1-m_3-m_2, m_2-m_1-m_3, m_2-m_3-m_1, m_3-m_1-m_2, m_3-m_2-m_1). Note that it is possible for each participant to receive the messages in an order different from the orders of messages delivered to other participants.

Classification of reliable multicast - 2

FIFO multicast

A system which guarantees that the delivery of the messages adhere to the following condition is said to provide FIFO (first-in-first-out) or send-order multicast:

If process P sent messages m_i and m_j, in that order, then each process in the multicast group will be delivered the messages m_i and m_j, in that order.

Suppose P_1 sends messages m_1, m_2, and m_3 in order, then each process in the group is guaranteed to have those messages delivered in that same order: m_1, m_2, then m_3.

Classification of reliable multicast - 2

Note that FIFO multicast places no restriction on the delivery order among messages sent by different processes.

To illustrate the point, let us use a simplified example of a multicast group of two processes: P_1 and P_2. Suppose P_1 sends messages m_{11} then m_{12}, while P_2 sends messages m_{21} then m_{22}. Then a FIFO multicast system can deliver the messages to each of the two processes in any of the following orders:

m_{11}-m_{12}-m_{21}-m_{22},
m_{11}-m_{21}-m_{12}-m_{22},
m_{11}-m_{21}-m_{22}-m_{12},
m_{21}-m_{11}-m_{12}-m_{22}
m_{21}-m_{11}-m_{22}-m_{12}
m_{21}-m_{22}-m_{11}-m_{12}.

Classification of reliable multicast - 3

Causal Order Multicast

A multicast system is said to provide **causal** multicast if its message delivery satisfies the following criterion:

If message $\boldsymbol{m_i}$ causes (results in) the occurrence of message $\boldsymbol{m_j}$, then $\boldsymbol{m_i}$ will be delivered to each process prior to $\boldsymbol{m_j}$. Messages $\boldsymbol{m_i}$ and $\boldsymbol{m_j}$ are said to have a causal or happen-before relationship, denoted $\boldsymbol{m_i} \to \boldsymbol{m_j}$. The happen-before relationship is **transitory**: if $\boldsymbol{m_i} \to \boldsymbol{m_j}$ and $\boldsymbol{m_j} \to \boldsymbol{m_k}$, then $\boldsymbol{m_i} \to \boldsymbol{m_j} \to \boldsymbol{m_k}$. In this case, a causal-order multicast system guarantees that these three messages will be delivered to each process in the order of $\boldsymbol{m_i}$, $\boldsymbol{m_j}$, then $\boldsymbol{m_k}$.[

Causal Order Multicast – example1

Suppose three processes $\boldsymbol{P_1}$, $\boldsymbol{P_2}$, and $\boldsymbol{P_3}$ are in a multicast group. $\boldsymbol{P_1}$ sends a message $\boldsymbol{m_1}$, to which $\boldsymbol{P_2}$ replies with a multicast message $\boldsymbol{m_2}$. Since $\boldsymbol{m_2}$ is triggered by $\boldsymbol{m_1}$, the two messages share a causal relationship of $\boldsymbol{m_1}$-> $\boldsymbol{m_{2.}}$ Suppose the receiving of $\boldsymbol{m_2}$ in turn triggers a multicast message $\boldsymbol{m_3}$ sent by $\boldsymbol{P_3}$, that is, $\boldsymbol{m_2}$-> $\boldsymbol{m_{3.}}$ The three messages share the causal relationship of $\boldsymbol{m_1}$-> $\boldsymbol{m_2}$-> $\boldsymbol{m_3}$. A causal-order multicast message system ensures that these three messages will be delivered to each of the three processes in the order of $\boldsymbol{m_1}$- $\boldsymbol{m_{2-}}$ $\boldsymbol{m_3}$.

Causal Order Multicast – example2

As a variation of the above example, suppose $\boldsymbol{P_1}$ multicasts message $\boldsymbol{m_1}$, to which $\boldsymbol{P_2}$ replies with a multicast message $\boldsymbol{m_2}$, and independently $\boldsymbol{P_3}$ replies to $\boldsymbol{m_1}$ with a multicast message $\boldsymbol{m_3}$. The three messages now share these causal relationships: $\boldsymbol{m_1}$ -> $\boldsymbol{m_2}$ and $\boldsymbol{m_1}$ -> $\boldsymbol{m_3}$. A causal-order multicast system can delivery these message in either of the following orders:

$\boldsymbol{m_1 - m_2 - m_3}$

$\boldsymbol{m_1 - m_3 - m_2}$

since the causal relations are preserved in either of the two sequences. In such a system, it is not possible for the messages to be delivered to any of the processes in any other permutation of the three messages, such as $\boldsymbol{m_2 - m_1 - m_3}$ or $\boldsymbol{m_3 - m_1 - m_2}$, the first of these violates the causal relationship $\boldsymbol{m_1}$ -> $\boldsymbol{m_2}$, while the second permutation violates the causal relationship $\boldsymbol{m_1}$ -> $\boldsymbol{m_3}$.

Classification of reliable multicast - 4

Atomic order multicast

In an atomic-order multicast system, all messages are guaranteed to be delivered to each participant in the exact same order. Note that the delivery order does not have to be FIFO or causal, but must be identical for each process.

Example:

P_1 sends m_1, P_2 sends m_2, and P_3 sends m_3.

An atomic system will guarantee that the messages will be delivered to each process in only one of the six orders:

m_1-m_2- m_3, m_1- m_3- m_2, m_2- m_1-m_3,

m_2-m_3-m_1, m_3-m_1- m_2, m_3-m_2-m_1.

Atomic Multicast Example

P_1 sends m_1 then m_2.

P_2 replies to m_1 by sending m_3.

P_3 replies to m_3 by sending m_4.

Although atomic multicast imposes no ordering on these messages, the sequence of the events dictates that P_1 must be delivered m_1 before sending m_2. Likewise, P_2 must receive m_1 then m_3, while P_3 must receive m_3 before m_4. Hence any atomic delivery order must preserve the order m_{1-} m_{3-} m_4. The remaining message m_2 can, however, be interleaved with these messages in any manner. Thus an atomic multicast will result in the messages being delivered to each of the processes in one of the following orders: m_{1-} m_{2-} m_{3-} $m_{4,}$ m_{1-} m_{3-} m_{2-} $m_{4,}$ or m_{1-} m_{3-} m_{4-} $m_{2.}$ For example, each process may be delivered the messages in this order m_{1-} m_{3-} m_{2-} $m_{4,}$.

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, as you recall, is connectionless and unreliable. For the basic multicast, Java provides a set of classes which are closely related to the datagram socket API classes.

The Java Basic Multicast API - 2

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

1. **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 (see next section).
2. **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.

The Java Basic Multicast API - 3

3. **DatagramSocket**: In the datagram socket API, this class represents a socket through which a process may send or receive data.
4. **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.

IP Multicast addresses

- 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.

IP Multicast addresses - 2

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 2^{28}; that is, approximate 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)

IP Multicast addresses - 3

An application which uses the Java multicast API must specifiy 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, assuming that the combination of the random address and port number will not likely be in use.
3. Obtain a transient multicast address at runtime; such an address can be received by an application through the Session Announcement Protocol.

IP Multicast addresses - 4

Some of the most interesting of the assigned addresses:

224.0.0.1 All Systems on this Subnet

224.0.0.11 Mobile-Agents 224.0.1.23 XINGTV

224.0.1.84 jini-announcement

224.0.1.85 jini-request

224.0.1.115 Simple Multicast

224.0.6.000-224.0.6.127 Cornell ISIS Project

224.0.7.000-224.0.7.255 Where-Are-You

224.0.8.000-224.0.8.255 INTV

224.0.9.000-224.0.9.255 Invisible Worlds

2240.12.000-224.0.12.063 Microsoft and MSNBC

224.0.18.000-224.0.18.255 Dow Jones

224.0.19.000-224.0.19.063 Walt Disney Company

224.0.22.000-224.0.22.255 WORLD MCAST

224.2.0.0-224.2.127.253 Multimedia Conference Calls

IP Multicast addresses - 5

- 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 your 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).

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);
  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);
```

Multicast peer joins a group and sends and receives datagrams

```
import java.net.*;
import java.io.*;
public class MulticastPeer{
        public static void main(String args[]){
         // args give message contents & destination multicast group (e.g. "228.5.6.7")
        MulticastSocket s =null;
         try {
                InetAddress group = InetAddress.getByName(args[1]);
                s = new MulticastSocket(6789);
                s.joinGroup(group);
                byte [] m = args[0].getBytes();
                DatagramPacket messageOut =
                        new DatagramPacket(m, m.length, group, 6789);
                s.send(messageOut);
```

// this figure continued on the next slide

continued

```
            // get messages from others in group
                  byte[] buffer = new byte[1000];
                  for(int i=0; i< 3; i++) {
                     DatagramPacket messageIn =
                           new DatagramPacket(buffer, buffer.length);
                     s.receive(messageIn);
                     System.out.println("Received:" + new String(messageIn.getData()));
                  }
                  s.leaveGroup(group);
            }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
            }catch (IOException e){System.out.println("IO: " + e.getMessage());}
          }finally {if(s != null) s.close();}
    }
}
```

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 neighboring 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.

Setting the “time-to-live” - 2

- 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.

Setting the “time-to-live” - 3

```
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.

Setting the “time-to-live” - 4

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

Reliable Multicast API

- the Java basic Multicast API provides **unreliable** **multicast**
- ***The Java Reliable Multicast Service*** (***JRM Service***) **A Reliable Multicast Library**
- provides the capabilities for a receiver to repair multicast data that are lost or damaged, as well as security measures to protect data privacy.

- marrying the bandwidth savings of multicast with the guaranteed delivery of unicast transport protocols such as TCP.
- Some multicast system that do not tolerate miss or corruption: multi-receiver file transfers, stock tickers, and collaborative applications

Summary - 1

- **Unicast** vs. **multicast**: unicast is one-to-one communication, while multicast is one-to-many communication.
- An archetypal multicast API must provide operations for **joining** a multicast group, **leaving** a multicast group, **sending** to a group, and **receiving** multicast messages sent to a group.
- **Basic multicast** is **connectionless** and **unreliable**; in an unreliable multicast system, messages are not guaranteed to be safely delivered to each participant.

Summary - 2

A **reliable** multicast system ensures that each message sent to a multicast group is delivered correctly to each participant. Reliable multicasts can be further categorized by the order of message delivery they support:

- **Unordered** multicast may deliver the messages to each participant in any order.
- **FIFO** multicast preserves the order of messages sent by each host.
- **Causal** multicast preserves causal relationships among the messages.
- **Atomic** multicast delivers the messages to each participant in the same order.

Summary - 3

- **IP multicast addressing** uses a combination of a Class D address and a UDP port number. Class D IP addresses are managed and assigned by IANA. A multicast application may use a **static** Class D address, a **transient** address obtained at run time, or an **arbitrary unassigned** address.
- The **Java basic multicast API** provides **unreliable multicast**. A ***MulticastSocket*** is created with the specification of a port number. The ***joinGroup*** and ***leaveGroup*** methods of the ***MulticastSocket*** class, a subclass of ***DatagramSocket***, can be invoked to join or leave a specific multicast group; and the ***send*** and ***receive*** methods can be invoked to send and receive a multicast **datagram**. The DatagramPakcet class is also needed to create the datagrams.
- There are existing packages that provide **reliable multicast**, including the Java Reliable Multicast Service (JRM Service).

More example

- https://www.developer.com/java/data/how-to-multicast-using-java-sockets.html#:~:text=The%20MulticastSocket%20class%20defined%20in,0.0%20to%20239.255.

- https://www.irif.univ-paris-diderot.fr/~hf//verif/ens/an09-10/internet/udp-multicast.ppt