Multicasting

Computer Networks

Lab #6 - Multicasting Communication

- IP Multicasting
- Using Multicast Sockets in Java
- Example
- Exercise: Multicasted Media Streaming under Timing Constraints

Multicasting

Multicasting is a communication mode allowing one message to be sent from one process to a group of processes. The sender does not need to send the message, one at a time, to each one of the destination processes (as in unicast or point-to-point mode). Instead, the message is sent (at once) to all destination processes that belong to the multicast group.

IP Multicasting

To use IP multicasting we need IP Multicast Addresses (as destination IP endpoints). IP Multicast addresses in IPV4 are IP addresses in the range 224.0.0.0 to 239.255.255.255 (inclusive).

IP Multicast Communication (IPMC) in the Internet is very limited or discouraged. IPMC traffic is usually blocked or not routed (by different Internet Service Providers). For practical use, it is generally limited to traffic in **LANs** - Local Area Networks or **INTRANETS** - Interconnections of different LANs of the same organization.

Sender processes can also limit the routing of IPMC packets, for instance, so that they are only received by receiver processes running on machines in the same local network where the sender is. This limit is set using the TTL mechanism (TTL field in IP headers): Setting the TTL to 1, the packet will not be routed to outside of the sender local network.

Multicast Sockets in Java

In Java, the regular DatagramSocket class can be used to send packets to multicast addresses. To receive, the use of the `MulticastSocket` class is mandatory. MulticastSockets are an extension of (UDP) DatagramSockets, with some additional capabilities for joining "multicast groups" (defined as IP Multicast Addresses or IP Group Addresses) or setting TTLs, for example.

A multicast message is sent in a datagram packet with a destination IP Multicast address and port.

To receive IP multicast messages, the receiver must use, necessarily, a MulticastSocket (created in a certain port). Then, the created socket must join to a multicast group address, invoking the `joinGroup(InetAddress groupAddr)` method. Then it is able to receive multicast datagrams addressed to the multicast address joined. The receiver can also use the `leaveGroup(InetAddress groupAddr)` method, when it is not interested in receiving more multicast messages sent to the defined group.

To setup the TTL (hop count) for Multicasting datagrams there is the method `setTimeToLive(int ttl)` in the class `MulticastSocket`.

Example of code excerpt (sending and receiving multicasting datagram packets):

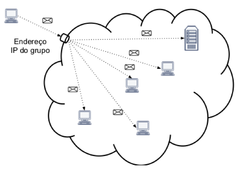
```
// join a Multicast group and send the group salutations
...
String msg = "Hello";
InetAddress group = InetAddress.getByName("228.5.6.7"); // the used multicast group
MulticastSocket s = new MulticastSocket(6789); // to use port 6789
s.setTimetoLeave(1)  // set TTL (hop counter) - to just one hop
s.joinGroup(group);

DatagramPacket hi = new DatagramPacket(msg.getBytes(), msg.length(),
                                       group, 6789);
s.send(hi);
// get responses!
byte[] buf = new byte[1000];
DatagramPacket recv = new DatagramPacket(buf, buf.length);
s.receive(recv);
...
// OK, I'm done talking - leave the group...
s.leaveGroup(group);
```

Summary:

- When one process sends a message to a multicast group, all subscribing processes joined to that multicast group receive the message (within the time-to-live range of the packet).
- The socket of the sender does not have to be member of a multicast group if it is only used to send messages to that group.
- When a `MulticastSocket` in a port P subscribes or joins to a multicast group G (using the `joinGroup (InetAddress addr)` method, it receives datagrams sent by other processes/hosts to the <G,P> endpoint.
- A `MulticastSocket` relinquishes membership in a group to not receive messages using the `leaveGroup(InetAddress addr)` method.
- Multiple Multicast Sockets may subscribe to a multicast group and port concurrently, and they will all receive group datagrams.

The following picture is a representation of a multicast communication environment.



Example

In this example we use two programs (see the classes in the source code repository):

MulticastSender.java

This program computes the date/time in the machine where it runs, and multicasts this information in each time interval to interested processes. Multicast address, port and time interval are input arguments for the program.

MulticastReceiver.java

This program receives multicasted messages sent from the MulticastSender program. You can check that you can run several MulticastReceivers simultaneously. All of them will receive the multicasted messages sent from the MulticastSender.

Note: To run programs using MulticastSockets in computers with dual TCP/IP stack (IPv6 and IPv4), sometimes you need to force the use of the IPv4 stack. To do this, you can use the following option when you run java programs:

java -Djava.net.preferIPv4Stack=true MulticastReceiver 224.2.2.2 9000

Another possibility is to include in your code the following line:

System.setProperty("java.net.preferIPv4Stack", "true");

```
import java.net.*;
import java.io.*;
import java.util.*;

public class MulticastSender {
    public static void main(String[] args ) throws Exception {
        if( args.length != 3 ) {
            System.err.println("usage: java MulticastSender  multicast-address port time-interval") ;
            System.exit(0) ;
        }
        int moreQuotes=20; // change if needed
        int port = Integer.parseInt( args[1]) ;
        InetAddress group = InetAddress.getByName( args[0] ) ;
        int timeinterval = Integer.parseInt( args[2]) ;
        String msg;
        if( !group.isMulticastAddress() ) {
            System.err.println("Multicast address required...") ;
            System.exit(0) ;
        }
        MulticastSocket ms = new MulticastSocket() ;
                // You can see that you can also use a DatagramSocket
        do {
            msg = new Date().toString();;
            ms.send( new DatagramPacket(msg.getBytes(), msg.getBytes().length, group, port ) ) ;
            --moreQuotes;
            try {
                Thread.sleep(1000*timeinterval);
            }
            catch (InterruptedException e) { }

        } while( moreQuotes >0 ) ;
        msg="tchau!";
        ms.send( new DatagramPacket( msg.getBytes(), msg.getBytes().length, gro\
up, port ) ) ;
            ms.close();

    }
}
```

```
import java.io.*;
import java.net.*;
import java.io.*;
import java.util.*;

public class MulticastReceiver {

    public static void main(String[] args ) throws Exception {
        if( args.length != 2 ) {
            System.err.println("usage: java MulticastReceiver multicast-address p\
ort") ;
            System.exit(0) ;
        }

        // to force IPv4 in dual IP stacks
        // System.setProperty("java.net.preferIPv4Stack", "true");

        int port = Integer.parseInt( args[1]) ;
        InetAddress group = InetAddress.getByName( args[0] ) ;

        if( !group.isMulticastAddress() ) {
            System.err.println("Multicast address required...") ;
            System.exit(0) ;
        }

        MulticastSocket rs = new MulticastSocket(port) ;
                // You can only use here a MulticastSocket ...
        rs.joinGroup(group);

        DatagramPacket p = new DatagramPacket( new byte[65536], 65536 ) ;
        String msgdate;

        do {
            p.setLength(65536); // resize with max size
            rs.receive(p) ;
            msgdate =  new String( p.getData(), 0, p.getLength() ) ;

            System.out.println("Data/Hora recebida: "+ msgdate  ) ;
        } while(!msgdate.equals("tchau!")) ;

        // rs.leave if you want leave from the multicast group ...
        rs.close();

    }
}
```

Exercise

In this exercise the challenge is to write a program (**SenderProgram.java**) for media streaming. The program must be able to send a movie using IP multicasting to a group of clients (the audience). For the client, you can use the **VLC tool (Video Lan Client)** available here. VLC can play UDP Streams (in real time) received from the network, as unicast or multicast datagram packets. You must download VLC and install it for the exercise.

Background for the exercise

The UDP protocol is particularly suitable for real time multimedia streaming. As you know, UDP is a non-reliable transport, used as a connectionless service to transmit datagrams. Non-reliability is not a problem in this case. On the contrary, it is appropriate for streaming, because the loss of some packets can be tolerated. For the purpose of real-time playing of multimedia contents, the occasional loss of a packet is not fatal and will usually only cause a momentary loss of quality in the received information, or the replacement of the missing information by momentary noise. The retransmission of lost messages is impracticable, because it is necessary to attend the real time requirements for playing. On one hand, the allowable delay of messages for playing purposes is very small. On the other hand, lost messages are no longer useful, given the time synchronization requirements.

The challenge

Write the multimedia streaming application (SenderProgram.java), using UDP (in multicasting mode) to transmit a movie encoded in fragments sent as a sequence of datagram packets. The packets must be sent continuously and repeatedly, to a destination multicast group: `<IP multicast address, port>`.

The transmission rate must comply with time constraints to allow the receiver (VLC player receiving in the IP multicast address and port) to play the movie, offering a pleasant playing experience to the user.

The time constraints related to fragments (movie frames corresponding to the sequence of messages that your program must send) are stored as records in the movie encoding file (**streaming.dat**) used by the sender to transmit the movie. You can find the file in the source code repository

Format of the records stored in the file *streaming.dat*

A record has the following format:

(media-packet-size, timestamp, packet-content), where:

- media-packet-size: is a short integer
- timestamp: is a long integer (more on this further on)
- packet-content: is a byte array (containing an encoded movie frame) with *media-packet-size* bytes

Your program SenderProgram can read the *streaming.dat* file using the following code excerpt:

```
....
byte[] buffer = new byte[ BUF_SIZE ] ;
DataInputStream dis = new DataInputStream(
                              new FileInputStream("/tmp/streaming.dat"));

while ( ... ) {
    int size = dis.readShort();
    long timeStamp = dis.readLong();
    dis.readFully( buffer, 0, size);
    ....
}
```

Parameterization in VLC:

To play the movie in VLC, use the network capture configuration options: **udp://@IPMC-ADDRESS:PORT**

Ex: udp://@224.224.224.44:9999

VLC will then receive the movie sent by your program to the group multicast address 224.224.224.44, port 9999

Guidelines to develop the SenderProgram

- First try to send the packets, in sequence, without any control of the temporal constraints (or timestamps). You will see (in VLC) that it doesn't work well.
- Then use the timestamps in the file **streaming.dat** to send each packet in sequence but in the proper moment to comply with the required rate. The timestamp is stored in *nanoseconds*, representing the instant when the datagram must be sent, relative to the initial moment when you start the transmission.
- Remember that you can use `System.nanoTime()` to control the correct transmission rate when sending datagram packets. You must note that the initial value of the *logic-clock* in your sequence is not "0" when you start the execution of your program. Thus, only relative values are interesting in the sender processing.

The file *streaming.dat* must be used to demonstrate your implementation. You must have the file ready to be used (read) by your SenderProgram. You can implement the SenderProgram to use the following arguments:

```
SenderProgram filename 224.224.224.44 9999
```

The following figure represents your real-time streaming environment.