L2 - UDP Multicast with Java

---

1 Introduction

1.1 Objectives

The goal of this lab is to consolidate the main concepts required for programming distributed applications that use UDP multicast communication.

Furthermore, we also expect you to gain some familiarity with the sockets Java API for multicast communication.

1.1 Multicast Communication

Multicast communication allows a process to send the same message to multiple recipients, which may run on different computers.

Multicast communication has at least two interesting features. First, it can be implemented very efficiently when the sender and recipients are connected to the same network segment that uses a broadcast medium. This is common in local area networks, either wired or wireless. Second, the sender usually does not need to enumerate explicitly the recipients. Often, a multicast address is a logical address and the sender does not even need to know the recipients.

On the other hand, the efficient implementation of multicast communication on network segments that do not support broadcast, e.g. the Internet, is not as easy.

1.2 Multicast Communication with UDP

In UDP multicast communication (with IPv4), the recipients are specified by a port number and an IP address in the range 224.0.0.0 to 239.255.255.255, which is reserved for multicast communication. Note that this address is not the address of a computer, but rather a logical address that together with the port number specifies the multicast group.

Thus, in UDP multicast communication the sender does not need to know the IP address of the computers where the destination processes are running. For a sender process, multicast communication with UDP sockets is transparent. That is, it is exactly the same as unicast communication with UDP sockets. In particular, the sender must invoke the send primitive only once, regardless of the number of receivers. Whether UDP uses unicast or multicast communication depends on the destination IP address.

A process that wishes to receive messages addressed to a multicast group must first join this group. After joining a multicast group, the reception of messages sent to that group is done as in unicast communication, i.e. by invoking the receive primitive.

2 Application

The application you shall develop is a variant of client-server application described in the previous lab.

In this variant, the server uses UDP multicast communication to broadcast the address of the socket, i.e. the IP address and port number, in which it offers the service.

More specifically, the server must send this information periodically to the multicast group specified in its command line. On the other hand, to learn the addresss of the socket where the server offers the service, clients must join this multicast group, specified in their command line, and receive and process a message sent by the server. After determining the address of the server socket, clients must then send the request (`register` or `lookup`) to the server.

Question: What are the advantages of this solution compared to that used in the previous lab?

2.1 Program Invocation

Server

The server program shall be invoked as follows:

> **java Server <srvc_port> <mcast_addr> <mcast_port>**
> where:
>
> > <srvc_port> is the port number where the server provides the service
> > <mcast_addr> is the IP address of the multicast group used by the server to advertise its service.
> > <mcast_port> is the multicast group port number used by the server to advertise its service.

Client

The client program should be invoked as follows:

> **java client <mcast_addr> <mcast_port> <oper> <opnd> ***
> where:
>
> > <mcast_addr> is the IP address of the multicast group used by the server to advertise its service;
> > <mcast_port> is the port number of the multicast group used by the server to advertise its service;
> > <oper> is "register" or "lookup", depending on the operation to invoke;
> > <opnd> * is the list of operands of the specified operation:
> >
> > > `<DNS name> <IP address>`, for `register`;
> > > `<IP address>`, for `lookup`.

2.2 Communication Protocol

In this work you will specify the communication protocol for the server to advertise the address of the socket where it offers its service.

The communication protocol to use for lookup and register requests is the same as in the previous lab.

2.3 Implementation Details and Issues

Server Concurrency

The server program should have an infinite loop in which the server waits for a request from a client, processes the request and sends the respective response to the client. The server program must also advertise its service periodically with a period of 1 s. Thus you can implement the server using either one or two threads.

In a single threaded implementation, you may find it useful to use `java.net.DatagramSocket`'s `void setSoTimeout(int timeout)`.

For a multiple-threaded implementation, you may find it useful to use the classes `Timer` and `TimerTask` of the `java.util` package. Or, for a more sophisticated implementation, the `ScheduledThreadPoolExecutor` of the `java.util.concurrent` package, which is more verstatile, but somewhat more difficult to use.

Multicast Forwarding

In principle, IPv4 routers do not forward multicast packets. However at FEUP, due to the use of VPNs or IPv6 (I'm not sure which one), multicast communication can cause significant disruption. It is therefore important that the multicast communication uses a **TTL (time-to-live) of 1**. In addition, make sure that your program does not send multicast messages at an excessive rate, e.g., in a tight loop.

In order to detect possible problems, the server should print in the terminal a message each time it multicasts a message. The format of this message may be:

> **multicast: <mcast_addr> <mcast_port>: <srvc_addr> <srvc_port>**
> where:
>
> > <mcast_addr> is the IP address of the multicast group used by the server to advertise its service;
> > <mcast_port> is the port number of the multicast group used by the server to advertise its service;
> > <srvc_addr> is the IP address where the server provides its service;
> > <srvc_port> is the port number where the server provides its service.

In addition, in order to observe the operation of your solution, the server shall reveal what it is doing by printing messages with, e.g., the following format:

> **<oper> <opnd> * :: <out>**
> where:
>
> > <oper> should be "register" or "lookup", according to the operation invoked;
> > <opnd> * is the list of operands received in the request for the operation;
> > <out> is the value returned by the operation, if any.

Client

The client program must first join the multicast group to learn the address of the socket that provides the service. Then, it must send the request to that socket address, wait for the response and terminate after receiving it.

The client program shall print on the terminal messages revealing its operation. The format of these messages may be identical to that of the messages printed by the server, i.e.:

> **multicast: <mcast_addr> <mcast_port>: <srvc_addr> <srvc_port>**
> where:
>
> > <mcast_addr> is the IP address of the multicast group used by the server to advertise its service;
> > <mcast_port> is the number of multicast group port used by the server to advertise its service;
> > <srvc_addr> is the IP address where the server provides the service;
> > <srvc_port> is the port number where the server provides the service.

In addition, the client program shall print messages on the terminal describing the actions it executes, including the requests sent to the server and the respective responses. The format of these messages may be:

<oper> <opnd> *:: <out>
where:

<oper> <opnd*> and have the same meaning and format of homonyms arguments from the command line and
<out> is the value returned by the operation invoked or "ERROR" if any error occurs.

3 Documentation

- MulticastSocket (Java Platform SE 8)
- A simple example on `java.net.MulticastSocket` (A very simple example to get started. One of thousands you can find on the Web.)
- Oracle's MulticastSocket's Tutorial (Part of a not so simple Tutorial on datagram communication.)
- A simple example on `java.util.TimerTask` and `java.util.Timer` (A very simple example to get started. One of thousands you can find on the Web.)
- Nice tutorial about the `java.util.concurrent.ScheduledExecutorService` class. You may also want to read the tutorial by the same author about the `java.util.concurrent.ExecutorService` interface.