COMP202-17B - Decentralised Chat with Multicast

September 6, 2017

1 Introduction

This practical will introduce you to programming network applications with Java using UDP datagrams and multicast sockets. You will do so by writing a small decentralised chat system where people interested in chatting can interact, but there is no server and no connections involved. Please take some time to look through Lectures 10 and 14, available on Moodle, for an overview of UDP and Multicast.

This practical will involve you using the DatagramPacket and MulticastSocket classes. You can find technical documentation on these classes in the official Java documentation available online at:

`http://docs.oracle.com/javase/8/docs/api/`

My implementation is 65 lines long; it should not take you long to do.

2 Academic Integrity

The files you submit **must** be your own work. You may discuss the assignment with others but the actual code you submit must be your own. You must fully also understand your code and be capable of reproducing and modifying it. If there is anything in your code that you don't understand, seek help until you do.

You **must** submit your files to Moodle in order to receive any marks recorded on your verification page.

This assignment is due Monday September 25th by 11am and is worth 6% of your final grade.

3 Specification

Create a ChatClient class. In that class, create a MulticastSocket, and have it listen on port 40202 on the multicast address 239.0.202.1. The next step is a bit ugly: we want to monitor for messages from other clients, but we also want to be able to type our own messages. The easiest thing to do is

1. spawn a thread that listens on the multicast socket for incoming messages.
2. have the ChatClient class read a line of text from the keyboard and send it out in a DatagramPacket. You can use the BufferedReader and InputStreamReader classes on System.in to read a line of text from the keyboard.

The DatagramPacket class takes bytes, rather than text, so you will need to use the String methods that encode and decode an array of bytes for the DatagramPackets that you send and receive. For each packet that you receive, obtain the IP address of the sender and print that with the message. Your output will look very similar to the output below:

```
cms-r1-1:[106]~/202>java ChatClient
hello world
130.217.252.1: hello world
130.217.252.17: hello, there
the quick brown fox.
130.217.252.1: the quick brown fox.
```

While you're debugging your implementation you might find it convenient to test in isolation from your class mates. Please choose a random port between 32768 and 65535 to do so. If you choose a port outside of this range your packets will be discarded by the firewall and your chat client will appear to not work. Note that if someone sends a message while you're typing yours, the terminal will break your message up. There's not much that can be done about this, except for writing a GUI, which is not required for this assignment.

VERIFICATION PAGE

Name: _____

Id: _____

Date: _____

Note: this practical is due Monday 25th of September, and the code submitted to the Moodle assignment page after you have had the assignment verified. It will be marked out of 6 and is worth up to 6% of your final grade.

1. Explain the structure of your program, and allow the marker to read and test your code.
2. Demonstrate that your program receives multicast messages from other systems on the local network.
3. Why did we chose to use a multicast address in 239.0.0.0/8, rather than a different multicast address from 224.0.0.0/3?

4. Compare your multicast implementation with a theoretical approach that used unicast TCP sockets. Which is the more complex approach? What advantage would TCP sockets bring over multicast?