Lab 2: Socket Programming

ECE156 - Networking
Duke University

1 Lab Overview

1.1 Introduction

Peer-to-peer applications are a common feature of computing today. Services from BitTorrent to Skype all rely on direct client to client contact. In this programming assignment you will construct a simple peer-to-peer based relay application. This application will demonstrate the basic operation of peer-to-peer communication.

For this lab, you need to write a client and a server utilizing TCP socket connections. The client must be able to connect to the server and obtain a list of available clients. After obtaining this list, the client must be able to connect to one other client on the list. The clients must then be capable of relaying text messages directly to one another without further help from the server.

You can define whatever convention you would like to indicate that a text message is complete (e.g. hitting [Enter], clicking send). Each client must send an acknowledgement to the other client that a text message has been received and displayed. You must also provide some way to disconnect and end the conversation. After disconnecting, the user should be able to retrieve a new copy of the server’s client list and connect to a new client. The connection between the two clients must be duplex, so both clients are capable of transmitting and receiving at the same time.

This can be envisioned as a simple chat application, where you can figure out who is online from the server and then talk to those other users directly.

2 Network Programming

The goal of this assignment is to learn socket programming in a peer-to-peer context. You can use any IP addresses you would like for communication. The use of the C programming language is strongly encouraged, but not required. Other languages that support sockets such as Java or even ADA may be used if you email the TAs for approval. The idea behind using C is to ensure that you really use the sockets to interface with the network - don’t use any libraries that abstract away the networking part of this network application.

If you are using C, you have to use system calls like socket(), bind(), listen(), accept(), connect(), and close(). For a detailed tutorial refer to Beej’s guide:


Don’t hesitate to seek out other tutorials or reference sources online.
2.1 Hints

Here are a few hints that may help you as you write the program.

- You have to choose a server port to connect to. Ports from 1-1023 are mostly used for certain services and require administrative privileges. Use port numbers greater than at least 1023.
- Close your sockets cleanly before exiting the program. If you abort the program, the port may not be freed.
- You can run all of the processes on the same machine. For your machine, just use localhost. You can use ifconfig (unix) or ipconfig (windows) to determine the IP address for testing across multiple machines.
- Be wary of overzealous firewalls stopping your connections - try temporarily disabling firewalls if you find your connections timeout or are denied.
- Don’t forget that Wireshark can watch what your program is transmitting, possibly helping you during debugging.

3 Logistics

3.1 Grading

Partial credit will exist, and significant credit is available to those who demonstrate a functioning program that fulfills the requirements listed above. Clean code, good commenting, and intelligent/descriptive error handling combined are important components of making this assignment manageable - and important components of your grade. You must also provide brief written documentation of your work to allow a third party, without knowledge of your source code, to write another client that can use your server and talk to your client. Diagrams and flowcharts may be used to minimize the amount of writing you need. 1-2 pages should be sufficient in most cases. MSPaint diagrams and/or hand drawings are acceptable if legible and adequately explained.

3.2 Bonus

Bonus points will be awarded for extra work that improves functionality, and more work will provide significantly more bonus points. For example, the ability to add an arbitrary number of people to the chatroom would improve functionality. Other examples would include the ability to chat with a user behind a NAT, to have distributed list servers, or any other creative extension of the communications component of the application. Keep in mind that a non-functional application with lots of features is a lot less useful than a simple but functional application.

Bonus will also be awarded if you give the option to use either UDP or TCP. Explain why the difference choices are meaningful. Calculate the round trip time (RTT) of your message under both protocols. Make sure that the RTT includes the time for sending your message and receiving the acknowledgement. How does the RTT vary with message size? Is there any difference between UDP and TCP in mean RTT? the variance of RTT?

3.3 Groups and Protocols

You may work in groups of TWO to write the server component of the application. However, each student must individually write their own client. You may collaborate within your pair to jointly write the documentation for how to connect to your server and different clients.