CISC450/CPEG419 Programming Assignment

DUE: Friday November 17th 5:00PM

General Instructions

1. This programming assignment is to be done in **groups of two students**.
2. You are allowed to consult any Internet resource, books, the course TA, or the instructor. You are not allowed to exchange code snippets or anything across groups. While you may read on general socket programming on the Internet and look at code examples, this should be only for the purpose of understanding. Do all the coding yourself, do not copy or cut-paste code from **ANY** website.
3. You can use only socket programming for the project. Do not use any sophisticated libraries downloaded from the web.
4. You can use Python/Java/C++/C for coding.
5. You should allocate sufficient time for testing your code.
6. It is important that your code has adequate comment and proper indentation. Comment during/before coding, not at the end. For every variable/function you should have a comment. Use appropriate code comments for explaining the logic where necessary.
7. Provide documentation for your code. This should explain the code structure in terms of directories, files, how to compile etc. Detailed instructions for submitting the documentation along with code are at the end of the document.
8. Choose proper and intuitive names for the variable, function, file, directory, etc.
9. We will evaluate all of the above aspects of your code, not just whether or not the code runs.
10. You can choose to do **ANY ONE** of the following two projects.

Additional Instructions

1. The below projects involve multiple hosts. You may want to write the code such that your code can be run on a single host where the multiple hosts are mimicked by using the loopback address (127.0.0.1) with different ports. This may help during the initial phases, where you need to do a lot of debugging. The IP address/port specification for the different hosts is ideally provided through configuration files or as arguments to your code.
2. As part of each of the two projects, you need to **simulate packet losses**, i.e., have a routine in your program that drops packets uniform at random, where the loss percentage is specified either in a configuration file or as argument to your code.

Project 1: Network Performance Measurement System

In this project, you will develop a system for measuring the performance of the network (path) between two hosts A and B (from A to B). We will consider the following three metrics to measure the performance:

- Packet loss rate: this is defined as the ratio of packets, which when sent from A to B, are lost by the network path in-between (much like what `ping` does).
- UDP throughput: what is the maximum rate at which B can receive UDP segments sent from A.
- TCP throughput: what is the maximum rate at which data can be transferred over a TCP connection between A and B, with data flowing from A to B. (Can you see why this may be different from the UDP throughput?)

(The UDP and TCP throughput measurements should also factor in any **simulated packet losses**) Write the corresponding programs to be run at machines *A* and *B* that perform the appropriate experiments to measure the above three metrics. Ideally, you should just have a single program which can either act as the source of the traffic or as the destination.

As a next step, add the following functionality. From any third machine, called the controller *C*, you should be able to perform any of the three desired measurements (as specified via an argument) from *A* to *B*. We should be able to do this at any time *C* chooses. So, controller *C* should issue an appropriate command which is communicated to *A* and *B* (in what order?). After this command issue, the appropriate measurement should be performed (using the same logic as earlier). And the measurement result should be conveyed back to *C* and the results of the measurements displayed. The controller program can be a separate program from your program to do the actual measurements.

Project 2: Client-Server Chat System

In this project, you have to write a chat program under the client-server architecture for real-time communication between 3 users. Specifically, the program should permit chat sessions between 2 or 3 users. In case of 3 users, messages generated by one user should reach the other two (e.g., like a conference). The chat program needs to allow a user to type messages, which should be conveyed to the other end(s) and displayed at the other end(s). Note that under the client-server architecture, clients cannot directly communicate with each other and must communicate through a server. The user interface need not be sophisticated, but should be usable. It can be text-based or graphical.

Your program should **only use UDP sockets** and no TCP sockets. This means that you have to implement reliability (of the messages) on top of UDP, in your own program, without using TCP. (Note that you are **simulating packet losses**). You don't need to worry about congestion/flow control, but you would need to ensure packets are in order. You can choose to implement any type of reliability mechanism (even your own design as long as it works).

As a next step, implement a feature for the users to be able to exchange files. That is, in a two-user session chat between machines *A* and *B*, the user at machine *A* should be able to upload a file to machine *B*. The machine *B*'s user will choose whether or not to receive the file, and where the incoming file should be placed, and using what name. Check that your program works for both text files as well as binary files, and verify that the file has been received successfully and reliably.

Submission guidelines

Read the below instructions *carefully* and follow them meticulously. Read this well in advance, and prepare your submission, not at the last minute.

Organizing your submission

- All relevant files should be under one directory. This directory should be named after the roll numbers of the two students. For example: "RuiZhang-RaminRamazi".
- Within RuiZhang-RaminRamazi, you should have a file called "README.txt" or "README.pdf". The following contents are required in the README file:
  - **List of relevant files**: Give the list of relevant files including all source files and configuration files which you have written. This list of files should ideally have all the relevant files/directories within your main RuiZhang-RaminRamazi directory. Specifically do not have any irrelevant, old, or temporary files which clutter the directory. Clean-up before submission.
  - **Compilation instructions**: How should one go about generating the executable from your source files? Give the actual set of commands which someone can cut-paste from the README in order to compile. Provide such instructions for each executable file you have to generate.
  - **Configuration file(s)**: If your code uses any configuration file(s), describe the format of such file(s) clearly. Also include in the directory some example configuration files.

  - **Running instructions**: You may be generating more than one executable. Describe logically what each executable does. In addition, for each such executable file, how should one run it? What are the command line arguments (describe each argument)?

- You may create any number of sub-directories (multiple levels too if you want) within your main directory. The README file within the main directory should describe everything: all files/directories within any sub-directory too.

- **Commenting**:

  - Each source file should describe in the beginning, in a comment, what that source file contains logically.
  - Make sure to name variables, functions, file names with intuitive names wherever possible. In addition, provide a comment for each variable/function describing it logically. You need not do this for very trivial variables/functions; use your common sense judgment. The overall objective is that a third person should be able to easily understand what that variable/function does logically.
  - Please also comment sections of the code which will help in understanding the logical flow of the code. For example, comments for a loop can describe any non-obvious invariant involved.

How and when to submit

- In the final submission, you have to tar-gzip or zip the main directory (e.g. RuiZhang-RaminRamazi) and submit a single file. Make sure to tar-gzip or zip from the parent directory of this directory, not from within this directory itself. The tar-gzip or zip file should also have the same name as the directory (RuiZhang-RaminRamazi.tar.gz or RuiZhang-RaminRamazi.tgz or RuiZhang-RaminRamazi.zip in this case).

- Submit the code via Canvas before the deadline.

Demo & Evaluation

We will have a project evaluation session, for about 10 minutes per group. The actual date for this will be announced later. A sign-up sheet will be provided via GoogleDoc for you to sign-up for this evaluation.

During this evaluation session, you will have to show a demo of your project (show even if it is not fully working). We will also ask you questions to test your understanding of the code and its working. We will also evaluate the cleanliness of your coding, documentation, and commenting. We expect both members of the group to be present during this evaluation session. During the evaluation session, be sure to arrive at least 10-15 minutes before your slot, setup your demo using the code that was submitted by the deadline. Ensure it works correctly. No debugging will be permitted during evaluation.