

CSCE 416 (Fall 2019); Lab Assignment 3

Peer-to-Peer Chat and Wireshark

Points possible: 150
Due date: Oct. 9, 2019

This assignment can be done in a group of at most 2 students.

In this assignment, you will implement a peer-to-peer chat program and see the program in action using Wireshark. The chat client should be able to enter a chat server and “connect” to an available client. Once connected, the chat server will only receive a message from one client and forward the message to another client to give them a “feel” of peer-to-peer connection.

Chat Server

The chat server should act as a message forwarder, and it should only forward messages from one client to another. The jobs of the chat server are as follows: accept a socket connection from a new client; maintain a list of existing clients' name, their state (busy or available), and their socket; show the existing clients' name and their state to the new client; maintain a connection table with the list of peer connections; and forward message from one client to another as per the connection table.

Note that the server should be able to accept new connection at any state, without waiting for input from the existing clients. So, the server should be able to maintain the connection to all existing sockets, receive packets from and send packets to the sockets, and poll for a new socket connection from a new client. You can assume the server can support a maximum of 100 clients. You should implement the main class for the chat server in a file called `chat.server.java`.

Chat Client

When a new client enters the chat program, it should choose an available client from a list of clients to get connected to and start the chat. A client can get out of the program at any time they want. This should not affect the existing clients; but the server needs to remove the client's name, state and update the peer connection table. You should implement the main class for the chat client in a file called `chat.client.java`.

See below for a sample output of the chat program. The clients are connected to the chat server sequentially — Client 1; then, Client 2; and then, Client 3. The outputs are color-coded to distinguish the relationship between peer connections (via server). Of course, you do not have to color-code your program output.

Wireshark

We will now use the Wireshark tool to debug the chat server and client. First, open Wireshark GUI in a VM where you will run the chat server; set the Wireshark to capture packets from `enp0s8`. (Recall from the lab assignment 2, the VM uses `enp0s3` to talk to the Internet and `enp0s8` to talk to the host machine and other VMs; thus, we will only capture packets from `enp0s8`.) Second, set the Wireshark filter to `tcp` to only show the TCP packets. Finally, run the chat server and connect your clients one after another.

Your server should run as: \$ java chat_server #port Waiting for a client ... Client accepted List of clients and states albert free Waiting for a client ... Client accepted List of clients and states albert free alan free Waiting for a client ... Client accepted List of clients and states albert busy alan busy claude free : : :	Client 1: \$ java chat_client #IP #port Enter client name: albert Welcome to the 416 chat server List of clients & states albert free List of clients & states albert free alan free Connect to which client? Connect to which client? Received request from alan Connect? y You are connected to alan albert: hi, alan! alan: yo albert. hru? albert: gd. wbu? : alan disconnected : List of clients and states albert free claude free Connect to which client? claude : : :	Client 2: \$ java chat_client #IP #port Enter client name: alan Welcome to the 416 chat server List of clients & states albert free alan free Connect to which client? albert You are connected to albert alan: albert: hi, alan! alan: yo albert. hru? albert: gd. wbu? // alan disconnects	Client 3: \$ java chat_client #IP #port Enter client name: claude Welcome to the 416 chat server List of clients & states albert busy alan busy claude free List of clients and states albert free claude free Connect to which client? Received request from albert Connect? y You are connected to albert claude: heyyy ... albert: hi : : :
---	---	--	--

Now, looking at your Wireshark output, answer the following questions. You should write your answers in a file called chat_wireshark_ans.txt.

Questions:

- (1) How many TCP control packets are exchanged over the network when a new client connects to the chat server? Briefly explain the packet types.
- (2) When a client connects to server, what is RTT for SYN-ACK packet? How did you find it in the Wireshark?
- (3) How many TCP packets are exchanged when one client sends a chat message to the other client? How many TCP control packets are exchanged when a client leaves the chat program?
- (4) How many TCP control packets are exchanged when a client wants to join the chat server, but no server is running? Briefly explain the packet types.

Now, connect 4 clients to the chat server sequentially. You should take a .jpg snapshot of your Wireshark output when all the 4 clients are connected, and name the snapshot as chat_wireshark.jpg. Note that your Wireshark output should show all TCP control packets for all 4 clients.

Submission

- ★ First, create a zip file containing the chat_server.java, chat_client.java, Makefile, chat_wireshark_ans.txt, and chat_wireshark.jpg;
- ★ Second, rename the zip file as YOURLASTNAME.la3.zip (YOURLASTNAME in all caps). If you are working in a group of two, then you should include an additional README file inside the zip file. The README file should briefly, but clearly, discuss the responsibilities of both students in the group. Also, rename the zip file as STUDENT1LASTNAME.STUDENT2LASTNAME.la3.zip
- ★ Third, upload it in the Blackboard.

Grade Breakdown

- (1) 5%: Your code is modular and well-documented. You can implement different functionalities in different classes, but your server and client main classes should be `chat_server` and `chat_client`. Do not forget to include all the `.java` files inside `Makefile`. Grader will use `make` to compile the files. Zero point if the files do not compile.
- (2) 20%: At least one pair of clients can run the peer-to-peer chat correctly.
- (3) 15%: At least 4 clients can connect to the chat server correctly.
- (4) 30%: All pair of clients can run the peer-to-peer chat.
- (5) 20%: All answers inside `chat_wireshark_ans.txt` are correct.
- (6) 10%: `chat_wireshark.jpg` shows at least 4 clients are connected to the chat server.

Notes

- ★ You must use JAVA (`openjdk-11-jdk`) on 64-bit Ubuntu 18.04 LTS. Your submission must have a `Makefile`; running `make` should build the executable classes.
- ★ The assignment can be done in a group of at most two students.
- ★ Do not put the executable class files inside your submission: 5% penalty.
- ★ Your code must run on the test setup, which is just some Ubuntu 18.04 LTS, running on VirtualBox.
 - We will not look at or run your program on your personal machine.
 - See lab assignment 1 for how to replicate the test setup for VirtualBox for Ubuntu 18.04 LTS.
- ★ Late penalty: 10% of total points possible per 5 hours.