CSE434 Socket Programming Project: Design Document

Kevin Liao and Kevin Van (Group 29)

February 26, 2017

1 Introduction

In this document, we describe the design of our peer-to-peer instant messaging application. This includes the data structures, formats of messages, and protocols used in our implementation.

2 Data Stuctures

We first go over the data structures used to store contact lists and contact information on our server. We entirely use common data structures from the C++ standard library, for simplicity and ease of extensibility.

At the lowest level, we store individual contacts in 3-tuples of `(contact-name, ip-address, port-number)`. We use a list (vector) `contact-list` to combine 0 or more contacts together. Then, since each `contact-list` must also be associated with a name, we contain names and lists in pair `(contact-list-name, contact-list)`. Finally, since there can be multiple contact lists, these pairs are all stored inside a list (vector) `contact-lists-list`.

More concretely, this was done using two templates (naming differs):

contact-list-name

```
typedef std::tuple<std::string,std::string,int> contact;
typedef std::vector< std::pair< std::string,
    std::vector<contact> > > list;
```

3 Message Formats

Here we explain how messages are passed between P2P nodes and the server, and between two P2P nodes.

3.1 Client-Server Messages

Clients can send queries to the server to perform a number of functions. These queries are formatted as follows

$$\texttt{<QUERY\_ID>-<ARG}_0\texttt{>-<ARG}_1\texttt{>-}\cdots\texttt{-<ARG}_n\texttt{>}$$

where each query is associated with a distinct `<QUERY_ID>`, and the arguments for each query are delimited by hypens. For example, a message passed for adding a contact might look like

```
3-alicelist-bob-127.0.0.1-9000
```

Another form of message passing is from an input file to the server (i.e. to import a contact list). These are of the form

```
Number of contact lists: 1

+mylist,3
-kevin,127.0.0.1,8000
-alice,127.0.0.1,8001
-bob,127.0.0.1,8002
```

The first line denotes the number of contact lists, although this is not necessary for importing a contact list. Lines beginning with `+` denote the name of the contact list, followed by the number of contacts in the list. Lines beginning with `-` denote the contact name, IP address, and port number. All fields are delimited by commas, and the parser ignores empty lines.

3.2 Peer-to-Peer Messages

When a P2P node sends an instant message, the message is forwarded using a cyclic overlay network. Messages between peers, as well as the initial message from the server to the node issuing the `im` query are of the form

```
<MESSAGE_CODE><MESSAGE>
-<NAME_0>,<IP_ADDRESS_0>,<PORT_NUMBER_0>
-<NAME_1>,<IP_ADDRESS_1>,<PORT_NUMBER_1>
-<NAME_2>,<IP_ADDRESS_2>,<PORT_NUMBER_2>
```

The field `<MESSAGE_CODE>` can be either 0 or 1. If 0, this means that the message is being sent from the server, and the requester should be prompted for a message to be sent. If 1, `<MESSAGE>` should be set, and the recipient will forward the message to the next node on the list. This message will have the same contents, but with the next node's information removed.

For example, if A receives a message with B, C, and D in the forward queue, A will forward the message to B, with B removed from the queue. Since the original sender of the message must also receive the message at the end of the overlay network, the server adds the sender to the end of the queue upon receiving an `im` query. Below is an example P2P message.

```
1Hello there!
-kevin,127.0.0.1,8000
-alice,127.0.0.1,8001
-bob,127.0.0.1,8002
```

4 Pseudocode

Here we include pseudocode for our contact server(contactServer.cpp) and P2P nodes (P2Pnode.cpp). It should be clear that the contact server is single-threaded, and the P2P nodes are multithreaded (one client thread, and one instant messaging thread).

4.1 Contact Server

Below is the pseudocode for the contact server.

```
procedure ContactServer
    Bind socket
    if input file then
        Parse input file and add contact lists
    end if
    while (1) do
        Block until received message
        Parse message
        switch query_id do
            case 1
                QueryLists()
                Send reply to client
            case 2
                Register(contact-list-name)
                Send reply to client
            case 3
                Join(contact-list-name, contact-name, IP-address, port)
                Send reply to client
            case 4
                Leave(contact-list-name, contact-name)
                Send reply to client
            case 5
                IM(contact-list-name, contact-name)
                Send reply to client
            case 6
                Save(file-name)
                Send reply to client
                if contact list length $> 2$ then
                    Forward message
                end if
    end while
end procedure
```

4.2 P2P Node

Below is the pseudocode for a P2P node and its instant messaging thread. A mutex is used to give threads control of standard input.

```
procedure P2PNode
    Initialize mutex
    Create instant messaging thread
    while (1) do
        Prompt user for query selection
        Parse input and send query to server
        Unlock mutex
        Time out for fraction of second
        Lock mutex
        Receive reply from server
    end while
end procedure
```

```
procedure IMThread
    Bind socket
    while (1) do
        Block until received message
        if only message header then
            Display message
        else if message_id == 0 then
            Lock mutex
            Prompt user for message
            Unlock mutex
            Remove next recipient from message queue
            Forward message
        else
            Remove next recipient from message queue
            Forward message
        end if
    end while
end procedure
```