

# Machine Problem 1: Let's chat

Post Date 09/08/2009

Due Date 09/25/2009

In MP1, we will build a P2P **Chat** App for Android, where several android phones can exchange messages with each other pair-wise using Internet over WiFi. The purpose of this MP is to learn the GUI design, to use intent and broadcast to communicate between GUI interfaces, and to practice threading and socket programming. It should be the most difficult MP since there is a big learning curve for most of you. Start early with your group members. Since message exchange is the building block for P2P file sharing application in MP2 and other P2P applications in MP3, think in advance what components you can reuse and the corresponding API specification.

## Assignment Description

Implement an Android App **Chat**, which enables mobile phones to exchange messages over Internet from phone GUI (For now, we will program and demo on Android Emulators). A Chat client is uniquely defined by the IP address of the mobile phone and TCP listening port, through which the other Chat clients or membership server can communicate with it using sockets. Yet, users view the other clients by their usernames for convenience. **Peer info** for a Chat client is a tuple containing the username, IP address of the mobile phone and Chat's TCP listening port. Chat application consists of 2 components.

### **Component 1: Membership Management (Peer Registration and Membership Update)**

Step 1: Chat obtains the username via **Registration** GUI;

Step 2: Chat registers the phone with the membership server, which listens at the public IP address and TCP port, by sending a 'register' request message with the peer information from the peer to the membership server. (By now, Chat should have established the server listening socket at the specified TCP port.) Upon receiving the 'register' request message, the membership server starts a **Peer List** if this is the first peer registration, or updates the existing peer list with subsequent peer registrations.

Step 3: In return, the membership server sends back the **Peer List**, which contains all the peer info for **active** Chat clients. An active client is the one which has recently registered with the server for less than 30 seconds.

Step 4: Chat lists all the active peers in **Peer List** GUI.

Step 5: Chat periodically registers with the membership server every 15 seconds (Repeat Step 2 and 3) and updates the Peer List GUI whenever needed. This step not only keeps the Chat client active in server's peer list, but also reflects the updated peers (newly joining or leaving peers) which register afterwards locally.

Membership management component is designed for user mobility. It means that no matter which sub-network (WiFi) the Android peer nodes reside on and which socket port they are listening at, two Android phones can exchange messages. In the emulator, a phone uses IP address of the computer to

communicate with other emulators or the membership server. In the upcoming MPs (MP2 and MP3), real devices get a public IP address by using WiFi connection.

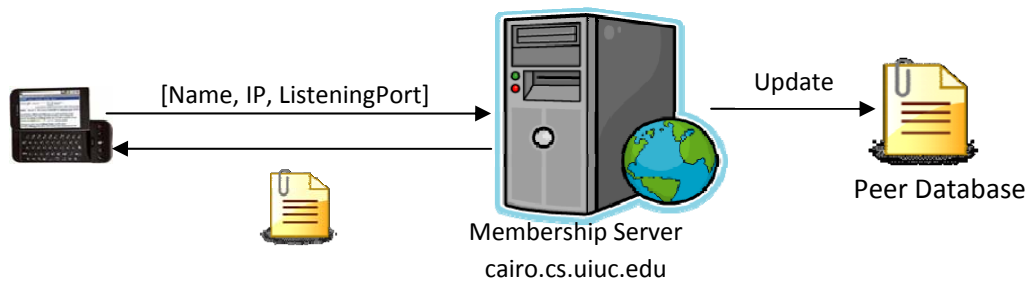
Languages to implement the membership servers are at your choice, like php, java, c++ or python. You can look at the following reference for server implementation.

[1] <http://www.prasannatech.net/2008/07/socket-programming-tutorial.html>

[2] <http://beej.us/guide/bgnet/output/html/multipage/index.html>

Peer list can be implemented as either a database or a file. However, a file-based implementation is sufficient for this MP. TCP socket is recommended so you do not worry about packet loss.

We illustrate the above steps in Figure 1.

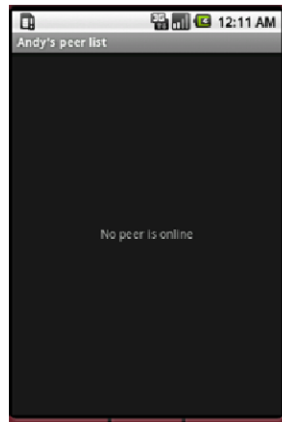


**Figure 1 Registration**

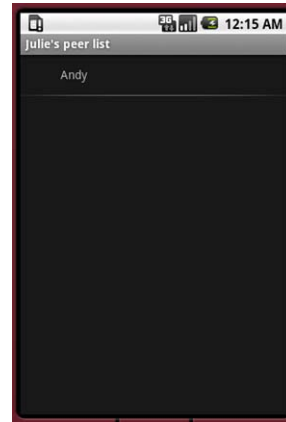
We show some example screen shots in Table 1.



Registration GUI



Peer List GUI  
when Andy first registers



Peer List GUI  
after Julie joins @ Julie's phone

**Table 1 Example Screen Shot for Registration Phase**

### **Component 2: P2P Message Exchange among multiple clients**

Step 1: A user (**U1 @phone P1**) selects a peer with username **U2** from the Peer List GUI.

Step 2: Chat client **C1 (C1 @ U1's phone P1)** initiates a TCP socket connection to Chat client **C2 (C2 @ U2's phone P2)**.

Step 3: **U1** writes a message in **Messaging** GUI. After the **send** button is clicked in GUI, **C1** sends on behalf of **U1** the message to Chat server **C2**, through the TCP connection in step 2.

Step 4: Consider a third user (**U3@phone P3**), where Chat client **C3 (C3@U3's phone P3)** sends a message to Chat server **C1 (C1@U1's phone P1)**. After receiving a message at **P1** from user **U3** through **C1's** server socket, **C1** shows the message in Messaging GUI if it currently has an active conversation with **U3** (Messaging GUI is on top) or indicates that a new message is received from **U3**.

Step 3 and Step 4 can alternate in arbitrary order, depending on the message sending sequence. Nevertheless, **U1** and other Chat users (**U2 and U3**) can exchange message reliably with each other, back and forth. There is a separate messaging GUI for each peer, which means that a conversation happens between a pair of nodes with multiple concurrent conversations.

There are several things you need to pay special attention to.

**First**, programs are built upon user-friendly GUI. You can utilize different GUI components to simplify the users' operation, such as editable text, text view, button, list and images.

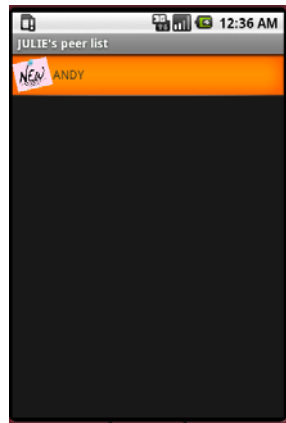
**Second**, programs should be responsive to users' interaction. There is a 5 second rule, which says that an application must respond to any user action, such as a key press or screen touch, within 5 seconds. However, in the common case, users may expect much shorter delay, like 1 second. We use threading and background service heavily to enable responsiveness, when performing time-consuming tasks, such as complex computation, network operation and file IO.

**Third**, you need to make sure the phone application is able to handle multiple messaging sessions with different peers simultaneously. Thread is the key again. A separate thread is dispatched for each socket connection request.

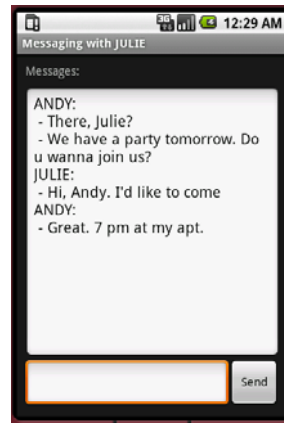
Example screen shots are shown in Table 2. However, you are **not** limited to those choices.



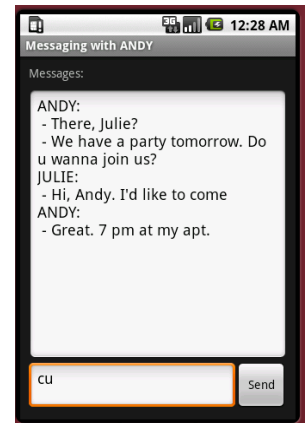
ANDY initiate some messages



An icon of new messages shows up @ Julie's Phone



Messages exchange back and forth @ ANDY's phone



Messages exchange back and forth @ JULIE's phone

**Table 2 Example Screen Shots for Message Exchange**

### Comments (Tips):

- 1) Socket programming

Ref: [http://www.anddev.org/socket\\_programming-t325.html](http://www.anddev.org/socket_programming-t325.html)

- 2) Thread programming

Ref: [1] <http://developerlife.com/tutorials/?p=290>

[2] <http://saeedsiam.blogspot.com/2009/02/first-look-into-android-thread.html>

- 3) How to run two networking emulators in a computer A using the public IP address of A, during debugging and demo?

Use telnet localhost to forward port (or adb forward as shown in tutorial, use *-s* option to specify the emulator instance). Suppose we have emulator A listening at TCP port 15216 and emulator B listening at TCP port 13126. For emulator A, perform the following 2 steps in command line

- i) Port forward to connect Android from localhost

**Type "telnet localhost 5554"; On the telnet console, type "redir add tcp:15216:15216"**

```

c:\ Telnet localhost
Android Console: type 'help' for a list of commands
OK
redir add tcp:15216:15216
OK
=
    
```

- ii) Use a proxy server which can listen on my\_public\_ip:15216 and forward the data to localhost:15216. stcpip.exe program can be found in <http://aluigi.altervista.org/mytoolz.htm>

**Type "stcpip localhost 15216 15216 "**

Do the similar things for Emulator B (telnet localhost 5556; redir add tcp:13126:13126; stcp pipe localhost 13126 13126). Now the two emulators can communicate with each other using public IP address of the computer. Basic concepts and tools for emulator networking are located at <http://developer.android.com/guide/developing/tools/emulator.html#emulatornetworking>

- 4) Correct permission in manifest.xml for socket.

```
<uses-permission android:name="android.permission.INTERNET" />
```

- 5) Work in the lab, campus network or at home?

It is important to have a public IP address for your programming computer to test the correctness of socket-related implementation. This means that your computer is not behind router or firewalls. Otherwise, you will face the network address translation (NAT) problem. Hence, it is OK for you to work in lab or campus network. But it is troublesome to work at home if you want to test the socket-related functions.

Since the purpose of MPs is to understand the important concepts and related techniques in the area of distributed systems, you are not required to solve the NAT problem. If you are interested in solving it for practical purpose or for fun, you can refer to wiki website ([http://en.wikipedia.org/wiki/NAT\\_traversal](http://en.wikipedia.org/wiki/NAT_traversal)) or the post (<http://www.blog.jasonederle.com/?p=361>).

## Delivery

Each group delivers:

- Source Java of your **Chat** program in the particular group directory (we will create a SVN directory for each group. Detail instruction will follow). The source code evaluation will be based on how well is your code documented. If you use some code you found on the net (You must understand the code you found and include in your code, not just blindly copy the code !!!) or in a local system directories, document it. It is very important that you give credit to people who developed the previous code. Your own code should include the following information at the beginning of each Java file
- Each major source file should include
  - File Name: Name of the File
  - Description: Short description what the file includes (general description, what kind of classes, interfaces are embedded in the file).
  - Version: version of your code. You start with version 0 and as you improve the code, at some point you increase the version.
  - Programmer's Name: your name(s) who developed the code
  - Company/University Name: you put the name of the course, department and university you implemented the code for;
  - Date:
- Each function in your Java file should have a header with information:
  - Function Name: Name of the Function

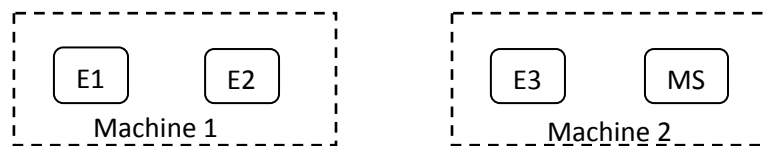
- Description: Short description what the function does.
- Arguments: Specification of each input argument parameter entering the function and its description.
- Results: Specification of returning parameters exiting the function and their description.
- Comments: some special system issues connected with this function
- Group representative(s) comes in lab 0216 at the scheduled time. Please sign-up at <http://spreadsheets.google.com/ccc?key=0AmVHs8Rfyiz8dHlrZUhpOGRJalN1ZG9KRFD0ZzhoZXc&hl=en> for demo time between 4 and 6pm on Friday, Sep 25th and show your demo of the required programs in 0216 Siebel Center.

The demonstration of the whole assignment for one group should take no more than 15 minutes. Please setup the demo environment beforehand. Get familiar with the port forwarding command to save time.

## Evaluation Scenarios of the Assignment (100 Points)

The points you can earn for each case is listed in parenthesis before dash. (A+B) means A points for demonstration and B points for answering to questions for this part during the demonstration. In total, we have 100 points and bonus 10 points.

- **Registration of 3 Peers (20+5)** – Register 3 Chat clients (3 emulators, E1, E2, and E3) with the membership server (MS) one by one. Show updated **Peer List** at MS, and **Peer List** at Peer 1 to 3 (P1, P2, and P3). 3 peers are running in at least 2 physical machines. In case you cannot grab enough machines, suggested client and server deployment topology is:



- **Disabling 1 Peer (5+5)** – Show updated **Peer List** at membership server, and Peer List at the remaining peers if one peer leaves the group.
- **2-Peer Messaging (25+5)** – Create several messages @ Peer A, send them to Peer B; create several messages @ Peer B, send them back to Peer A. Repeat for 2-4 rounds. Show recent message history.
- **3-Peers Messaging (10+5)** – Chat with two other peers simultaneously.
- **Responsiveness (5+0)** – Always respond to users' input
- **New Message Indicator (5+0)** – indicate when a new message arrives (in both Peer List and Messaging GUIs)
- **Documentation (10+0)** - Each group should write a REPORT file in pdf format. Please check the requirements for documentation.
- **Bonus Points (5+5)** – Timely peer list update. In the scheme above, a mobile peer periodically pulls the up-to-date peer list from the membership server, in order to learn the peers who recently join or leave the system. The accuracy of peer list is determined by registration period (In our setting, 15 seconds). An alternative scheme is letting server push the change to peer list to Chat clients upon peer leaving and joining. In this way, peer list is updated on demand and timely. In addition, message overhead from server is reduced by sending only those changes. You can implement this pushing scheme and show us.

### Requirement for Documentation

In the Documentation README file, you should answer the following questions. Email this document to TA at [huang23@illinois.edu](mailto:huang23@illinois.edu), and also store it in your group SVN directory.

- Overview
  - Describe UI interface design at a peer node
  - Describe the design of communication (intent, broadcast) between interfaces
  - Describe the threading structure of your Chat app at a peer node
- For membership management component
  - Describe the protocol design for new registration
  - Describe the protocol design for peer list update (join and leave)
  - Describe message format and peer list structure
- For message exchange component
  - Describe the protocol design of message exchange between a pair of peers
  - Describe message format

Additional programming tips will be listed in [Tip](#) document at Assignments tab, during the MP based on the questions. Also you can check Android programming tutorial ppt in the course website, which will be released on Sept. 14.

Feel free to post questions about any of these topics on newsgroup or to come by TA office hours at Monday 2:00pm-3:00pm or Thursday 3:15pm-4:15pm at room 0207 SC.