CS 456 Final Project Report

Devin Howard 20293980 dmhoward@uwaterloo.ca
Mingqian Ye 20281333 m4ye@uwaterloo.ca

Overview of the Application

The application has four activities. PrefActivity, MainActivity, ContactListActivity, and InboxActivity. The first run of the application calls up the PrefActivity, where the user must set their interests string before being able to enter the MainActivity. This is accomplished using a PreferenceScreen class. Once this occurs, the MainActivity is opened. From there, the user can open a menu to access any of the other three activities (including the PrefActivity to change their interests string). If the interests string is ever set to be blank, the PrefActivity will not be dismissed until a new one is set. Below, we detail the function of the three activities. After that, we'll detail the function of the server. Finally, we'll talk about our message sending protocol.

MainActivity

This activity is what opens when the application is launched (given a non-empty interests string in the PrefActivity). Its layout consists of two buttons and a ListView. The first button reads “Look for users”; it will run an HTTP GET. Upon doing so, it will populate the ListView with the user's information returned by the server. It also adds an image, which would be a potential for expansion to our application. If no users are returned the ListView will say “No users found”.

The MainActivity also begins a Runnable that will send a POST message to the server. It will check what the public and private IP address of the user are, as well as their latitude and longitude. It sends these values to the server along with the interests string every minute. Each time it does this, if a message is returned by the server an AlertDialog will pop up informing the user they have a message. They can choose to view it now or cancel, but regardless it will be saved in the InboxActivity.

From the user's perspective, the POST is essentially checking for messages. That is why the second button is labelled “Check for messages”. It will send a POST, which updates the user on the server (more frequent updates than every minute are not a problem; the server's behaviour will only change if no POST is sent for three minutes). Clicking this button will send the POST and if messages are on the server, it will immediately notify the user as desired.

The final function of MainActivity is in sending messages. Clicking an item from the ListView will call up a dialog asking if the user would like to send a message. If they say yes, the ComposeMessageActivity is opened and provided with the information of the selected recipient. See below for the details of how ComposeMessageActivity works. The MainActivity will also store this user in the ContactListActivity by creating an intent to run the ContactListActivity and providing it with the necessary data to store an entry in ContactListActivity's database. See below for more details.

This activity also has an options menu. Opening it shows three options: “Recently Contacted”, “Inbox”, and “Set Interests”. The latter will open the PrefActivity. The other two open ContactListActivity and InboxActivity, respectively.

ContactListActivity

The ContactListActivity can be run in two different ways. If it is given no extras, it will display the contact list. If it is given extras (the information on a user that has been contacted), it will store that user in its SQL database.

In the latter case, it will not even call up the UI. Instead, it adds the information provided about the user just contacted to ContactListActivity's database. It then calls finish () to return to MainActivity.

In the former case, however, it populates a ListView from the database of contacts. Contacts expire after 24 hours. If no users are in the database the ListView will say “No users found”. If there are users, they look just like the users in MainActivity. Clicking one will call up the ComposeMessageActivity just like in MainActivity. This is mainly a method to message offline users.

InboxActivity

The InboxActivity is also a ListView but it has marked differences from MainActivity and ContactListActivity. For one, it has a context menu that provides users with the option to delete all messages in the inbox; this will clear the ListView.

Received messages are stored in a SharedPreferences file. MessageHelper retrieves them on behalf of InboxActivity, which then can populate a ListView. Again, clicking an item in this ListView also sends a message to the user – this enables “reply” functionality for easier conversation.

Server

The server has three responsibilities. It needs to store user entries and delete them after 3 minutes, it needs to send and receive messages between users, and it needs to provide a list of nearby users when queried. It makes use of an “iptuple” to uniquely identify users – the user's public and private IP addresses separated by a comma.

The server makes use of two MySQL tables to do this. The messages table stores messages along with sender iptuple and receiver iptuple. The users table stores entries consisting of an iptuple, a longitude and latitude, an interests string, and a TTL. The TTL is simply the time it was stored; when the server is queried for users if the TTL is more than 180 seconds old it will be deleted.

There are three ways to interact with the server. Sending an HTTP GET will return output of the following form:

```
n                (where n is the number of entries)
internal1
external1
latitude1
longitude1
interests1
internal2
...
longn
interestsn
```

If no users are stored on the server, a single line containing “0” is returned. If users are, the total number of users is output as n on the first line, and then 5n lines are returned containing information describing each user. This information can be used to create the list of users in the MainActivity of our application.

One can also send an HTTP POST with 5 parameters: internal, external, latitude, longitude, and interests. If these exact five parameters are sent, an entry is stored in the users table with TTL equal to

the server's system time. The server then checks for messages destined for the iptuple (internal,external) and returns them one to a line in the following format:

```
n                    (where n is the number of messages for the given user)
internal1
external1
message1
internal2
...
externaln
messagen
```

In this way messages are “sent” to users every time they do a POST. Our application does a POST every minute so this is fairly frequent.

The final way to interact is to send an HTTP POST with 8 parameters. The first five are identical to the previous method of communication, and the new three are destInternal, destExternal, and message. These are stored in the messages table along with the internal and external (so the messages can be tagged with who they are from). The server will also check after “sending” these messages (storing them in the table) if there are any outstanding messages, and will return them as above.

If a POST is sent that is not of the two forms described, it returns an HTTP DECLINED.

Message Handling(Device-side)

In this device-to-server communication, the device is uniquely identified by its internal IP address and external IP address.

Sending messages to another device

1. When the user selects a contact from ContactListActivity, MessageHelper.showSendMessageDialog() is invoked and displays an AlertDialog.
2. The AlertDialog will prompt the user to confirm the message-sending action and proceed to ComposeMessageActivity.
3. In ComposeMessageActivity, after the user finishes entering the message and clicks “Send” button, Client.postUserInfoWithMessage() is invoked and the followings are interpreted as text and sent to the server via Http Post method.
    - Self (UserInfoObject)
    - Internal IP address of the destination device
    - External IP address of the destination device
    - message (String)
4. After the message is sent, it is the responsibility of the server to retain and distribute the message to the designated device.

Retrieving messages from server

1. Periodically, the device application sends HTTP POST requests to the server with Self (UserInfoObject), which uniquely identifies each device, to retrieve the messages from the server. This is done by Client.postUserInfoWithMessage(). The feedback entity retrieved from the server has the following structure:
    - Number of messages
    - Internal IP address of sender 1
    - External IP address of sender 1
    - Message 1
    - …
    - Internal IP address of sender n
    - External IP address of sender n
    - Message n
2. If the number of messages is not zero (i.e., there are new messages) ,Client.postUserInfoWithMessage() will then interpret the feedback and return a list of MessageObjects. Each MessageObject contains the message entity and the necessary information to reply to the sender.
3. The list of MessageObjects will be added to persistence storage. This is done by using the SharedPreferences utilities from Android.Since the MessageObject implements Serializable interface, all MessageObjects are converted into bytes and persistently stored.
4. MessageHelper.showViewMessageDialog() is invoked. An AlertDialog will display and prompt the user to switch to InboxActivity to view the new message(s).
5. In InboxActivity, the activity looks up all the MessageObjects from SharedPreferences and displays them in a ListView.