University of Manchester
School of Computer Science
COMP28512: Mobile Systems
Semester 2: 2018-19
Laboratory Task 4
Messaging with Android Smartphones

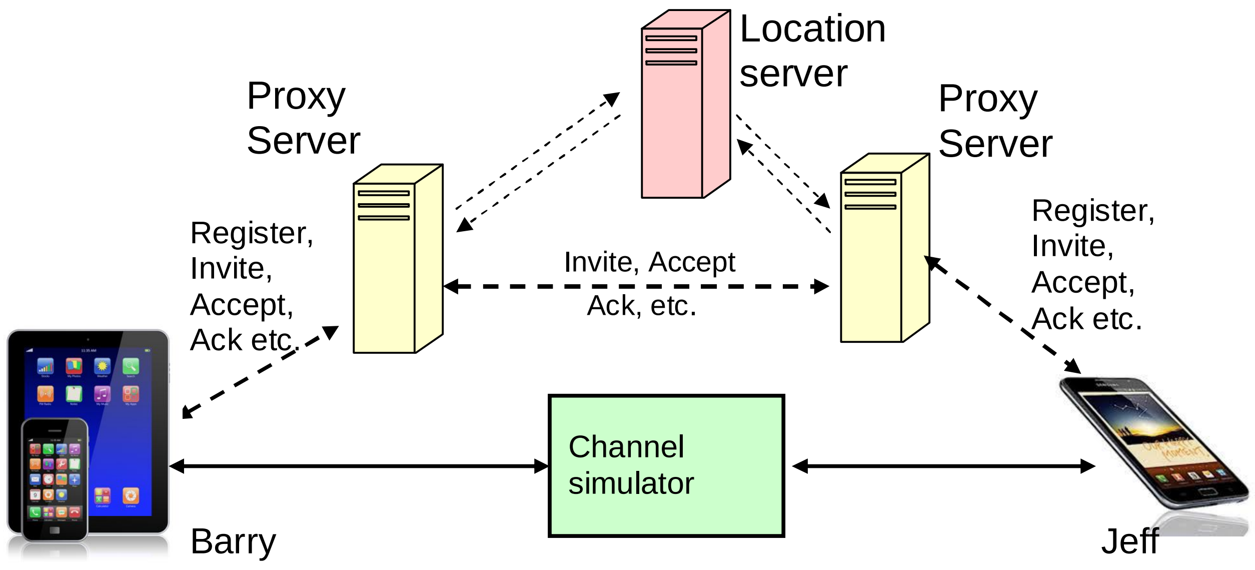
1. Introduction

The aim of this task is to become familiar with Android development and emulation facilities for developing and testing applications. Ultimately these applications can be downloaded to real smart-phones via a USB connection.

The application we aim to produce is a messaging system capable of exchanging, firstly, strings of text, secondly recorded text messages, images and MP3 files, and finally real time speech. Task 4 is concerned with introductory material and Task 5 will follow on with more advanced parts of the application.

The communication will take place via SIP-like proxy servers running some pre-developed software on PCs. The communications will ultimately be wireless using WiFi, but initially it will be by wired Ethernet between Android emulators and the 'proxy-servers'. Such an application could be useful in ad-hoc or emergency situations where the normal communications infrastructure has failed and standard global messaging services cannot be accessed.

The proxy-servers will provide registration, communication with a location-server (DNS) and protocols for inviting, accepting and acknowledging call set-up initiatives. To simplify the PC software, there will a single proxy server with a built-in location-server.



Rather than setting up a peer-to-peer link for direct message and speech communication, you will eventually (in Task 5) communicate via a channel simulator running alongside the proxy server. Initially, this will be a benign lossless connection, but later, it may start to introduce bit-errors. Worse still, there may be security attacks and eavesdroppers trying to intercept and disrupt the communication. Do not worry about this in Task 4, and keep things simple by using the ‘MSG’ command to send messages (see the list of Proxy Server commands later).

2. Android Development

To begin with, you will need to become familiar with the Android development and simulation facilities. There are many options for both Linux and Windows users. All the necessary software is open-source and can be downloaded in the laboratory or even at home. In our laboratory, we have an ‘integrated development environment’ (IDE) package known as Android studio.

2.1. Android Studio

This is the IDE made available by Google for developing Android applications. We have found it easier than Eclipse to learn and use, despite the fact that there were some problems adapting it to the laboratory machines. When using your own computer at home, you should not have too many problems using this IDE.

Android Studio provides editing and file management facilities for developing Android projects. It links to the Android ‘software development kit’ (SDK), the Java development kit (JDK), and other libraries. The Android SDK also contains the software needed to emulate mobile phones as a means of debugging and evaluating Android software on the PC itself.
To start android studio on a University machine type "start_android_studio.sh 28512".

2.2. Android Studio sample project

Users are advised to start by creating a very basic Android sample (‘dummy’) project which simply displays ‘Hello World’ on a smartphone. To do this after launching Android Studio on your PC, select 'New Project'. Then just accept the default options which request Android Studio to create a folder containing a number of files. This folder, with its many files, is the ‘dummy project’. It may then be ‘built’ to produce an application (app) that may be run on a mobile phone. After selecting the default options, you will see a window that says something like "Building 'MyApplication' Gradle project info". The IDE will now download some extra content. Gradle is Android’s package builder which is like ‘ANT’ or ‘make’. Once this is done, the main IDE will be shown with the 'Hello World!' project. In the ‘preview pane’, you will see a view of what you wish see on your mobile phone when you run the application.

The ‘preview pane’ is only for visualising the GUI (graphical user interface) layouts you create. It is not an ‘Android Virtual Device’ (AVD) capable of executing an application. It is a visualisation of the ‘content_main.xml’ file which is the xml file that specifies the intended layout of a GUI. Notice that you can switch between ‘Design’ and ‘Text’ to see either the xml file or its visualisation.

3. ANDROID Emulation

This Section gives details of two possible ways of emulating the operation of a real Android device on the PC. You can choose either, or even experiment with both.

3.1. Emulation using Android Studio facilities:

For emulation, ‘Android Studio’ can install an ‘Android Virtual Device’ (AVD). To make it do this, go to Tools > Android > AVD. When an AVD is being installed, we must select the 'Target' option which specifies the version of Android that our app will be built for. We must choose an API (‘application programming interface’) level. Select the API version you wish to install and click ‘install’. It is suggested that you install version API version 23.

There may already be a suitable AVD shown on the ‘AVD Manager’ menu display with a tick next to it. To add more API versions, open the Android SDK manager (Tools > Android > Android SDK Manager). Or click on ‘+Create VirtualDevice’ at the bottom of the AVD manager screen. A newly installed AVD should be ready for use immediately or after Android Studio has been closed down and reopened (maybe). It is suggested that you choose AVD device 'Pixel2' with API version 23

To see the real app in the emulator pane, click on the green arrow in the toolbar, or navigate to the 'Run' menu and click on 'Run'. After a little wait, a dialogue pane called 'Choose Device' will appear. Click on 'Launch emulator' with an appropriate AVD selected. You will then need to wait a further few minutes before the app is fully loaded and operational. If you have an Android mobile phone, you may prefer to download your apps to that.

Once the emulator is running, it does not need to be closed. To transfer a different version of your application to the emulator, repeat the process described above without closing the emulator. If you do close the emulator window, you may have to sit and wait while it loads again. If your app does not run automatically on the emulator, the emulated device may be locked as can happen with any smartphone. If this happens, drag the ‘padlock’ icon up and release it. If this produces your ‘home’ screen’ rather than your app, click on the middle icon which is something like a circle with squares dots inside. Then select the icon created for your app.

3.2. Emulation using ‘Virtual Box' in the Lab

To use the 'Virtual Box' device emulator from your window manager select:

Applications->System Tools -> Oracle VM VirtualBox

From the VirtualBox Manager choose: File->Preferences

Set the Default Machine Folder to "Your home directory/COMP28512VB" and Click 'OK'

Click Machine->Add and then select AndroidOS/AndroidOS.vbox.

It is safe to ignore most errors and warning messages at this point. A new virtual machine "AndroidOS" should appear. Hightlight it and click 'Start'. A new VirtualBox Android machine should then appear.

In the Andoid Studio IDE click the Terminal tab and type "adb kill-server". (This command is not required if you start the VirtualBox Android device before you start Android Studio).

Click on the Debug app icon and you should now be able to select the VirtualBox Android device. Then debug as usual.

If you get the following error message in Android Studio, just click Build->Rebuild Project

Error:Execution failed for task ':app:preDebugAndroidTestBuild'

> Conflict with dependency 'com.android.support:support-annotations' in project ':app'. Resolved versions for app (26.1.0) and test app (27.1.1) differ.

See https://d.android.com/r/tools/test-apk-dependency-conflicts.html for details

4. Debugging and Logging

While you are using the emulator, Android Studio, will be displaying debug information in a pane. This is 'logcat''. It is Android's way of debugging. You may have previously used print statements to debug Java or C programs. Although you can also use this approach for Android, it is recommended that you use the debugging tools supplied, i.e. LogCat. The statements are shorter to type; e.g. to output a debug message:

Log.d(TAG, “This is a message”);.

Note that:

1. Log.**d** : LogCat supports a few levels of debugging, and each level (or’severity’) has its own method (indicated by a single letter):

- Log.v – Verbose
- Log.d – Debug
- Log.i – Information
- Log.w – Warning
- Log.e – Error

2. The TAG argument may any string message you choose, such as:

TAG = "Activity Manager"

TAG is useful for indicating which part of your code generates a 'logcat' message, and Android Studio provides ways of grouping messages together and filtering out messages you do not want to see. When the application is running, a viewing pane should appear at the bottom of the IDE with debugging outputs. Filters may be created using the dropdown menu in the logcat panel by selecting 'Edit Filter Configuration' or similar. It is a good idea to type the package name into the relevant field (e.g. uk.ac.man.cs.XXX), so that only debugging information about the app itself will be displayed. If you wanted to concentrate on socket errors, for example, you could type “socket” into the relevant field..

5. The Proxy Server Provided

A proxy server ‘lab4-server.py’ is provided in Python. It is called a ‘proxy’ server because it will send commands to and receive commands from another client, on behalf of you as a client. The other client may be a real person you want to talk to. But to make experimentation easier, a dummy client called ‘BarryBot.py’ is provided. BarryBot will accept any invitation to connect to it, and will then return any messages that are sent to it.

To run the proxy server, type 'python lab4-server.py [IP] '. Either specify the IP address of the machine being used, or use the local loop-back (127.0.0.1). The IP address of the laboratory machine being used may be found by typing 'ifconfig'. If no IP address is supplied, then the proxy server defaults to run on the local loop-back IP address (127.0.0.1). The server always uses port 9999. Start ‘BarryBot.py’ after starting the proxy server.

The proxy-server responds to or sends the following commands:

REGISTER <myUsername> - Register myUsername with the proxy-server.

INVITE <username> - Invite another user to set up a communication link with you.

ACCEPT <username> - Accept username’s invitation and inform the user.

DECLINE <username> - Decline username’s invitation and inform the user.

MSG <username> <msg> - Send message <msg> to username (after link is established)

END <username> - Close the link between me and username & inform username

WHO - Get a list of all users who are currently on-line

DISCONNECT - Disconnect from the proxy-server

DUMP - Requests the proxy-server to dump the internal state on its terminal (for debugging)

INFO <msg> - Messages sent to the client by the proxy-server

ERROR <msg> - Error messages from the proxy-server

Before implementing this protocol in an Android application, you can test it by opening a terminal to act as a client and communicating with BarryBot via the proxy-server using TELNET. After opening a terminal, type 'telnet [ip] [port]' to establish a connection with the proxy-server. Use the correct IP address for the proxy-server and specify port 9999. Use the 'open' and 'send' commands and specify an IP address assigned to your machine or another machine. Use the 'loop back' IP address 127.0.0.1 if the proxy-server is implemented on the machine you are using. Once the terminal is connected, anything you type followed by a newline is sent to the proxy-server. Any responses from the proxy-server to you as a client will appear in your TELNET terminal. To terminate the connection to the proxy-server, exit TELNET by pressing <CTRL> + [. You will then drop into the terminal's command mode. Exit the terminal by typing 'quit'.

6. Android Support Software

To carry out this laboratory Task, you need to develop an Android project that is able to do three actions: (1) interact with a user, (2) receive any transmissions from the proxy-server, (3) send transmissions to the proxy-server. To help you with each of these actions, we provide support software in the form of a 'skeleton' project. This project will run correctly on Android Studio, but it does not do anything very useful yet. It does define a very basic user interface by displaying a message and two push-buttons which do not do anything yet. Also it sets up a socket for listening to messages received from the proxy-server. And it has code for transmitting a fixed meaningless text-message to the proxy-server. You must adapt the support software to your own needs. Ultimately you will want to produce a nice-looking well laid out mobile phone screen with lots of buttons, text -boxes and other widgets. So you need to learn how to use Android Studio's UI layout editor, and how to use text boxes to receive and transmit data.

Download the zipped project folder: 'MS19_Lab4_Client.zip'. Unzip it and take a look at the folders and files within it. You may be surprised at its general structure and the number of files it contains. Then open the folder as a complete project using Android Studio. There are just six files that you need to look at in detail. They all lie within the sub-folder '\Lab4Client\src\main'. The first three are the Java files and the other three are xml files. Here is a brief explanation of what they are:

MainActivity.java: Resides within '…\main\java' and contains the java code that runs as the main activity thread for your app and implements its user interface.

NetworkConnectionsAndReceiver.java: Resides within '…\main\java' and contains java code for a worker thread that provides the network connection to the server and keeps a receiver channel open at all times.

Transmitter.java: Resides within '…\main\java' and contains a class that sets up a new short-lived worker thread for each message that transmits it over the network.

res/layout/activity_main.xml: Resides within '…\main\res\layout' and describes the layout of the screen with buttons, text boxes and other 'widgets' that will be used to provide the application's user interface. You do not have to write the xml code yourself (though you can if you wish) because Android Studio provides a 'Layout Editor' with a 'drag and drop' graphical user interface. You just have to select a widget (for example a button or a text-box) from a menu, and then move it into place, make it the right size, and so on. A very basic layout has been provided with the support software to help you to get started. It has two push-buttons ('btnKill' and 'btnSendCmd').

res/values/strings.xml: Resides within '…\main\res\values' and is a file where string messages are stored for providing text required by the user interface. They are kept together in this one file rather than distributed as 'literal strings' among the many different files of java code. As exemplified by the code that labels the 'btnSendCmd' push-button referred to above, the syntax for accessing string resources within this xml file is: text="@string/send" '.

AndroidManifest.xml: Resides within '…\main' and defines some characteristics of your application including its name, permissions needed, SDK version (version of Android studio), etc.

7. More details of three Java files provided

7.1. MainActivity.java

This is the main java class that is run when your app starts. It extends the 'Activity' class and has many predefined methods that listen to events. You do not need to worry about any of these apart from 'onCreate()'. This is the code that runs when the app is opened. Some of this code has already been written for you, but you need to write some code here to assign actions to the UI objects already present, and to introduce and use some new ones. To begin with, you must connect the 'btnSendCmd' button (defined above) so that, when it is pressed, it sends a sensible message to the proxy-server. Also, the 'btnKill' button defined above must close the app, by executing: "System.exit(0);" when it is pressed. The code that is needed to connect a button to an action can be found at the end of the 'onCreate' method. Later, you will need to introduce a text-box allowing the user to enter a message. A message may be extracted from a text-box using the 'getText()' method of the 'cmdInput' textbox (also defined in 'activity_main.xml'). The message may then be sent to the Transmitter class which can send it to the proxy-server.

7.2. NetworkConnectionsAndReceiver.java:

This class handles the network connection between Android client and proxy-server, and remains active all the time to receive any messages from the network. This class should be run as a worker-thread because networking actions should not happen on the main UI thread. The main thread loop constantly listens for commands from the server. When commands are received, they are automatically appended to a textView ('txtServerREsponse') and shown on the Android device. If you are running the python server script on your machine (using 127.0.0.1) you need to set *SERVERIP* to be 10.0.2.2. This IP address maps the Android emulator to your localhost (127.0.0.1) interface.

We must be careful when passing data between multiple threads. Some form of synchronisation must be used to lock reads and writes to avoid race hazards (when threads try to read from and/or write to the same variables at the same time) and deadlocks. In this laboratory, we only consider communications between the main thread and a worker-thread. We do not consider direct communications between worker-threads.

Android has an elegant mechanism for allowing data to be exchanged between the main (UI) thread and other threads. It is provided by a method called 'runOnUiThread' which is used in this class. When the worker-thread wishes to communicate with the user, for example to display a message that has just been received, it simply has to use the 'runOnUiThread' method to place the required action in the event queue for the UI thread. There may be many runnable events waiting in this queue to be run by the main thread. So the UI thread will deal with these one by one in a way that will not cause any conflicts. When the UI thread is ready to deal with an action placed on the queue by a worker-thread, it will do so safely.

The NetworkConnectionsAndReceiver.java class has the following methods:

7.2.1. NetworkConnectionAndReceiver (AppCompatActivity parentRef): Class constructor which expects a reference to the UI thread/activity.

7.2.2. disconnect(): Cleanly disconnects from the server by closing I/O streams and network connection.

7.2.3. run(): This is the main thread loop that connects to the server and grabs all incoming messages. It should not be directly called. Remember that this is a thread. See the 'start()' method. To change how commands sent to the client are handled, you will need to change the code inside the while loop.

7.2.4. start(): This class extends Thread. (Thread java doc), Therefore this inherited method causes the thread to start executing. You may find it useful to consult the following documentation on threading:

http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

You are allowed, and encouraged, to modify these files as you wish.

7.3. Transmitter.java: This class receives, from the main UI thread, messages that need to be transmitted on the network. Each time such a message arrives from the main UI thread, a new short-lived worker-thread is created to transmit it. After that, the worker-thread is destroyed. The transmitter class does not need to use the 'runOnUiThread' method.

8. Further Explanation of the Task 4 Support Software

The main activity thread (sometimes called the UI thread) creates objects for displaying text, entering text, and a button listener (Send button). When the app is launched, the UI thread is executed. From the UI thread, the 'NetworkConnectionsAndReceiver' object is created. Then the UI thread starts a 'NetworkConnectionsAndReceiver' thread.

In the 'NetworkConnectionsAndReceiver' thread, a socket is created with the correct IP address and port number. Then, in a continuous loop, the input stream of the socket is read using a readLine method (which is a blocking method) and waits for a "\n" linefeed character. On receiving a "\n" character, a new runnable object is created with the object's own run method defined. We then use the 'runOnUiThread' method to pass the runnable object to the event queue of the UI thread.

If a socket cannot be created, for example because the proxy-server is not on-line, a 'logcat' message "Socket error" will be generated. If you filter logcat messages as mentioned in

Section 4 above, you can detect any such messages. Without a filter, there may be too many messages to see.

When the UI thread services this event it will execute the object's run method. In our case it will display the received message in the text display area of the user-interface. Eventually you will wish to modify this display to obtain a more professional screen.

On a Send button being clicked the UI thread creates a transmitter object. The text entered by the user is grabbed and passed to the constructor of the transmitter object. Then the UI thread starts a transmitter thread by invoking the run method of the transmitter via the start method. The transmitter thread runs the write method for the socket stream and transmits the message. The transmitter thread then "dies". This all gets repeated on the next Send button click. This Android project was built from an "Empty Activity" using API 15.

9. Getting used to Android Studio

Additional documentation is referenced on Blackboard. You may find it useful to watch the ‘You-Tube’ tutorials suggested, though they are quite long. There are many other tutorials that may be better, and shorter.

10. The sub-tasks for Task 4

Part 4.1 [4 marks]

Before implementing the protocol listed in Section 5 in an Android application, test the proxy server ‘lab4-server.py’ and the dummy client ‘BarryBot.py’ by running these programs in separate terminal windows and opening a third terminal window to act as a client using TELNET. Check that the Telnet client can register with the proxy-server, that its name and IP address are stored in the location-server and that it can send messages to BarryBot and receive his replies. Type 'telnet [ip] [port]' to establish a connection with the proxy-server which can be run using the ‘loop-back’ IP address: 127.0.0.1, and port 9999. (May not work on Windows)

Then familiarize yourself with the 'Android Studio' IDE and run the sample (‘dummy’) project, on the emulator and if possible on a real mobile phone. Modify the ‘Hello World’ message to ‘Hello BarryBot’.

Questions:

1: Are you satisfied that the two Python programs ‘lab4-server.py’ and ‘BarryBot.py’ will allow you to test a simple messaging system developed in Android?

2: If BarryBot were a real user, how would he know when the TELNET client has terminated your connection to him? How could the protocol be improved in this respect?

3: Why are literal strings discouraged in Android? What is the preferred alternative?

4: What is the purpose of the following files and directories created by Gradle:

(a) src/main/java, (b) res/ (c) AndroidManifest.xml

5: How does the naming of a package affect the file structure of an Android project?

Part 4.2 [4 marks]

Install and run the ‘Android support software’ project. contained in MS19_Lab4_Client.zip. The main software is also available as separate java and xml files. Modify the java code to make the ‘btnSendCmd’ button send the text-message ‘REGISTER myname’ to the server. Keep this part simple by using a fixed text message to begin with. Also, make the ‘btnKill’ button close the app when it is pressed. There is no need to allow the user to input any messages yet. Check that both these actions have the desired effect by using logcat messages to monitor responses from the proxy-server. Include logcat debugging statements in your code, and look for logcat messages that warn you, for example, if the proxy-server is not running.

Questions:

1: Where did you store your fixed text message?

2: How did you check that the ‘btnSendCmd’ and btnKill buttons had the desired effect?

3: Explain how you used the logcat debugging facility and show an example of the output obtained (in your demo and your report).

4: How does the ‘runOnUiThread’ method deal with inter-thread communications in Android?

5: Why is ‘runOnUiThread’ needed for the NetworkConnectionsAndReceiver thread but not for the Transmitter thread?

Part 4.3 [4 marks]

Further develop your Android application to improve the screen layout with a title, and a text-input box to enable the user to enter his or her name. Then enable the user to register with the proxy-server using this name. Once registered, the user can find out who is currently on-line. Provide a mechanism for doing this and displaying the response obtained from the proxy-server in a convenient form. Produce a ‘screen-shot’ display of the lay-out you have produced so far.

Questions:

1. How is your screen layout (however simple) made suitable for the application as developed so far?

2. What happens if the proxy-server is not on-line?

3. What happens if you try to register with a name that is already in use?

4. Could anything else go wrong with this preliminary version of the app?

Part 4.4 [4 marks]

Further develop your application to enable inviting, accepting and setting up a text message link. Demonstrate a two-way message link for interchanging short one-line messages with BarryBot using the MSG command as recognized by the proxy-server. Produce a screen-shot of your further developed layout.

Questions

1: What are the main features of your new layout and how do they work?

2. What are the possible disadvantages of using the proxy-server’s MSG command to convey the communications to and from clients, especially if you are thinking about introducing spoken messages and multimedia?

Part 4.5 [4 marks]

Review what you have produced so far, and identify any possible improvements to the screen layout, the information provided, the convenience of the user interface and the use of the protocol provided. Implement and test at least one improvement (1, 2 or 4 marks for each improvement: modest, substantial or outstanding). Produce a screen-shot of your final layout.

Questions

1: What are your improvements?

2. Are there any deficiencies in the protocol as currently implemented by the proxy-server?

3. How could the proxy-server be improved ?

7. Demonstration & Report

You will be asked to demonstrate your achievements with this Task in the laboratory. The demonstration will be worth 20 % of the total mark for Task 4.

You must submit a report on Task 4 to Blackboard by the specified deadline. The report must include the code you developed with appropriate documentation. Give details of the development, testing and evaluation in sufficient detail to allow a demonstrator and the external examiner to understand what you have achieved and what you have learned. The report with appropriate code and documentation will be worth 80% of the total mark for Task4.

Document last edited by Barry on 11/3/2019