University of Manchester
School of Computer Science
COMP28512: Mobile Systems
Semester 2: 2018-19
Laboratory Task 5
Communicating with Android Smart-phones

1. Introduction

After completing the previous Task, you will be familiar with Android development and emulation facilities for developing and testing applications. You have already developed a messaging system for exchanging strings of text. This Task is to continue the development of this messaging system for securely exchanging, firstly, strings of text over channels affected by noise and congestion. The channel noise will cause bit-errors which may be evenly spread out in time, or 'bursty'. The noise or congestion may sometimes cause packets to be lost completely. After experimenting with text messages, the aim is to exchange recorded speech messages, music and images.

As in the previous Task, the communication will be set up by SIP-like proxy servers running pre-developed software on PCs. The communications will ultimately be wireless using Wi-Fi, but initially it will be by wired Ethernet between Android emulators and the 'proxy-servers'. 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, a dummy client called barryBot, will return messages via a 'channel simulator'. This can be a benign lossless connection, or it may introduce bit-errors or lost packets.

2. The Server Provided

The python Server program used for the previous Task may continue to be used for this Task. It has two components:

(a) A 'proxy-server'.

(b) A 'channel simulator'.

The proxy-server uses Port 9999 as in before and responds to the same commands. The new channel simulator also uses Port 9999 and may be used by a client to send messages over a channel that may introduce bit-errors and packet loss. To remind you, the server, as used in the previous task, responds to 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 & inform username
DECLINE <username> - Decline username's invitation & inform username
MSG <username> <msg> - Send message <msg> to username (after link is established)
(This is exactly as in the previous task and does not use the channel simulator).
END <username> - Close channel 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

3. The Channel Simulator for This Task

You are still able to send messages using the 'MSG' command. However, in this Task, you can also receive communications via a channel simulator which conveys strings of "1" and "0" characters, and can be configured in different modes. The mode is an integer in the range 0 to 5. The format of messages that may be sent to a client via the '**Task 5** Channel Simulator' is as follows:

<mode>MSG∇<USERNAME>∇DATA

<mode> is an integer denoting the 'return channel mode' (0 to 5)

∇ denotes the 'Space' character,.

DATA is the string of "1" and "0" characters you wish to send.

DATA may be a literal string message such as "1010101" or it may be the name of a string containing, for example, "101010101". It will always be sent to <USERNAME> without errors, but the <mode> specified will determine how any messages returned by <USERNAME> will be affected by the channel simulator. If <mode> is zero, no errors will be introduced. However, setting <mode> to an integer greater than zero will cause the channel-simulator to introduce various degrees of error in the return path.

When experimenting with the effects of channel-errors, we use DATA in this special way to represent binary numbers or text. We only allow each character of the string DATA to be either '0' or '1'. When used in this way, DATA will be referred to as a 'bit-string'. It is wasteful of storage and transmission capacity, but easier to work with at 'bit-level', because we can see and easily modify individual bits. Your own Android software will need to convert arrays of text, speech, music or image samples to 'bit-strings' and vice-versa. This document (Section 6) gives some Java code for converting a normal positive integer to an integer array of ones and zeros. This code can be modified to convert the integer array to a 'bit-string'. To accommodate negative integers, you could add an offset to all integers at the transmitter and subtract it at the receiver.

Modes that may be specified:

0: Benign channel (no bit-errors, no lost packets)
1: Evenly spaced bit-errors (low bit-error rate)
2: Evenly spaced bit-errors (higher bit-error rate)
3: Bursty bit-errors
4: Lost packets (low loss-rate)
5. Lost packets (higher loss-rate)

These modes are only intended to be used with bit-strings. They will convey other strings, but any characters other than '0' or '1' will not be modified by the simulated channel in modes 0-3, though the whole string may be lost in modes 4 or 5. With modes 4 and 5, the bit-string is assumed to be the content of a single packet. The whole packet is occasionally lost, and when this happens, it is replaced by a 'zeroed packet' where the content is replaced by a sequence of the same number of zero characters. Therefore, as with the 'real time transmission' protocol 'RTP', the receiver will know when a packet has been lost. Of course the mechanism is a little different.

Example:

Assuming that a client is registered and connected to the dummy client ‘barryBot’, the following command sends a sequence of 18 bits as a message to barryBot without introducing simulated bit-errors:

MSG barryBot “101101110111100101”

The literal string “…” may be replaced by the name of a text string (used as a bit-string) containing only characters 1 and 0.

The following command sends a sequence of 18 bits without errors to barryBot but also causes the channel simulator in the return path to introduce a relatively high number of evenly spaced bit-errors.

2MSG barryBot “101101110111100101”

There is no space between the mode number (2) and MSG, and there must be one space before and one space after ‘barryBot’.

4. The dummy client ‘BarryBot.py’ with encryption

The dummy client ‘barryBot’ responds differently to some special commands while responding normally (as in previousTask) to all other messages. One special command is ‘ENCRYPT’ which is sent as follows:

<mode>MSG barryBot “ENCRYPT”

The special command is sent as normal text; i.e. not as a bit-string. If this were not a special command, barryBot would just send it back to you. But, barryBot does something else when it receives "ENCRYPT":

4.1. ENCRYPT will cause barryBot to return to you a random piece of secret text which will be encrypted. It you send ‘ENCRYPT’ again, you will get another (normally) different line of text. BarryBot.py is rather dumb, however, because he uses the same cipher stream for every message. Even dumber, every so often he will reply with the following ‘signature’ message, again encrypted with the same key:

“Sent by BarryBot, School of Computer Science, XXXXXXXXXXXXXXXXXXXX”

Sorry, we do not know the end of the message, so we had to replace it with X’s. You will have to guess this. Sorry again!

This ‘signature’ message is rather like a signature appended to the end of emails. All secret messages are of the same length. The signature will be sent, approximately, but not exactly, once every five messages. The encrypted message will be sent in binary form using an ’ASCII bit-array.’ In principle, you can cause bit-errors to occur by setting <mode> to be non-zero. But you do not need to try this as it just makes any attempt at decryption (eaves-dropping) a bit harder.

5. Playing sound on the ‘Android Virtual Device (AVD)’.

5.1: Playing sound from a file in the ‘resources’ directory

Create a ‘raw’ folder in the ‘res’ directory which may be viewed by expanding the ‘app’ folder label in the left hand ‘project’ window. Click ‘file’ > ‘new’> ‘Android resource directory’, click on the ‘app’ folder so that it is highlighted, then enter the directory name ‘raw’. Select resource

type ‘raw’; no other changes are needed. Copy a sound file from your desktop into ‘raw’. Maybe choose a ‘wav file with sampling rate 44100 Hz, 22.050 Hz, 11.025 Hz, or an mp3 file. A sampling rate of 8 kHz should work as well. Do not make the files too long. You must name the file with only lower-case characters and no spaces, for example, ‘soundfilename.wav’, or ‘soundfilename.mp3’.

Before the public class MainActivity.java statement (as seen in the edit window) enter:

```
import android.media.AudioManager;
import android.media.MediaPlayer;
```

After the public class MainActivity.java, statement, enter:

```
private MediaPlayer mySound ;
```

Then include in the over-rided ‘onCreate’ method, the statement:

```
mySound = MediaPlayer.create(this, R.raw.soundfilename);
```

The statement:

```
mySound.start()
```

will start the sound playing, but you are required to introduce a ‘button’ for doing this. We also need a ‘stop’ button to invoke

```
mySound.release ( )
```

To introduce the ‘start’ button and its ‘call-back method, do the following:

Select the Activity-main.xml window, and then find the ‘Design’ menu. Select ‘button’ then place the button on the AVD visualiser and note the xml code that is produced. Edit the android text = “…” statement to label the button, and then insert the following statement to name a call-back method “playSound”:

```
android:onClick=”playSound”
```

Finally, create the call-back method either by following the helpful light-bulb, or manually going back to the MainActivity.java window and adding the following call-back method:

```
public void playSound(View view) { mySound.start();}
```

Hopefully, the start button should start the sound playing, and you have a way of stopping it. Consider what will happen if the start button is pressed many times.

5.2. Playing sound from a Java array

The method described in the previous section is fine if your application just requires a set of fixed and pre-recorded sounds. However, you may need to generate sounds that are generated by your own code, such as a sine-wave whose frequency varies. The following code illustrates how this may be achieved using an ‘Audiotrack’ object called ‘track' defined using standardized parameters obtained from ‘AudioFormat’, ‘AudioTrack’ and ‘AudioManager’. To illustrate how this code may be used, a Java array ‘data’ is filled with a random noise signal, which is then written to ‘track’ using a ‘write’ method. It may then be played out as sound (‘shhhhh’).

```
import android.app.Activity;  import android.os.Bundle;
import android.text.method.ScrollingMovementMethod;
import android.util.Log;
```

```
  import android.view.Menu;    import android.view.MenuItem;    import android.view.View;
  import android.widget.Button;    import android.widget.EditText;
  import android.widget.TextView;
 // Added for playing sound
  import android.media.AudioManager;     import android.media.MediaPlayer;
  import android.media.AudioTrack;          import android.media.AudioFormat;
  import java.io.File;
  import java.io.InputStream;         import java.io.BufferedInputStream;
  import java.io.DataInputStream; import java.io.*;
  import android.content.res.* ;
  import java.util.Random;
public class msSound
  { private int Fs = 22050; // Samping rate (Hz)
    private int length = Fs*10; // length of array for 10 seconds
    private short[ ] data = new short[length]; // Array of 16-bit samples
    private int i;
    // Method for filling data array with random noise:
    void fillRandom()
     {  Random rand = new Random();
        for ( i = 0 ; i < length ; i++ ) { data[i] = (short) rand.nextInt();}   //Fill data array
      } // end of method fillRandom
     // Method for playing sound from an array:
     void arrayplay()
      { int CONF = AudioFormat.CHANNEL_OUT_MONO;
        int FORMAT = AudioFormat.ENCODING_PCM_16BIT;
        int MDE = AudioTrack.MODE_STATIC; //Need static mode.
        int STRMTYP = AudioManager.STREAM_ALARM;
       AudioTrack track = new AudioTrack(STRMTYP, Fs, CONF, FORMAT, length*2, MDE);
        fillRandom(); // Fill data array with 16-bit samples of random noise
        track.write(data, 0, length);   track.play();
        while(track.getPlaybackHeadPosition() != length) {};  //Wait before playing more
        track.stop();  track.setPlaybackHeadPosition(0);
        while(track.getPlaybackHeadPosition() != 0) {}; // wait for head position
      } // end of arrayplay method
  } //end of msSound class
```

5.3 Reading speech or music from a wav file into a Java array in Android

Adding the following method to the msSound class enables your app to read music or speech from a media file in R.raw, process it and play it out using AudioTrack. The sound is read into the Java array 'data' (defined within msSound) for processing and subsequently playing out.

```
void msSpeech()
  { // Method for reading 16-bit samples from a wav file into array data
    int i;   int b1 = 0;   int b2 = 0;
    try
     { // Make speechis an input stream object for accessing a wav file placed in R.raw
       InputStream speechis=getResources().openRawResource(R.raw.narrobandspeech8k);
       // Read & discard the 44 byte header of the wav file:
       for ( i = 0 ; i < 44 ; i++ ) { b1 = speechis.read(); }
       // Read rest of 16-bit samples from wav file byte-by-byte:
       for ( i = 0 ; i < length ; i++ )
         { b1 = speechis.read();  // Get first byte of 16-bit sample in least sig 8 bits of b1
           if (b1 == -1) {b1 = 0;}   // b1 becomes -1 if we try to read past End of File
```

```
            b2 = speechis.read();       // Get second byte of sample value in b2
            if (b2 == -1) {b2 = 0;} // trying to read past EOF
            b2 = b2<<8 ; // shift b2 left by 8 binary places
            data[i] = (short) (b1 | b2); //Concat 2 bytes to make 16-bit sample value
          }  // end of for loop
        speechis.close();
      } catch (FileNotFoundException e) {Log.e("tag", "wav file not found", e);}
        catch (IOException e) { Log.e("tag", "Failed to close input stream", e);}
    } // end of msSpeech method
```

To keep it simple, this method discards the information in the 44-byte wav header which could tell us what the sampling rate is. For the purposes of these experiments, we will just assume that the developer knows the sampling rate. Currently the public class msSound fixes the sampling rate at 22050 Hz, but it may be modified, for example to 8000 Hz, by the following statements.

```
track.setPlaybackRate(8000);
while(track.getPlaybackRate() != 8000) { } // wait for new sample rate to be established
```

5.4. Viewing and Processing Images

You can display an image on your mobile phone by first copying an image file, for example in bit-map (BMP) or JPEG form, from your desktop into the ‘raw’ directory of your mobile project. Use the Android Studio editing facility (text or graphical) to add an image view to the 'activity_main.xml' file for your project. This defines the area on the mobile phone display where the image will appear. The extra code created in the activity_main.xml file should look something like this:

```
//-----------------------------------------------------------
<ImageView
        android:id="@+id/imageView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:adjustViewBounds="true" />
//-----------------------------------------------------------
```

To display a downloaded image and process it to produce a modified version, develop your mainActivity class to include code along the following lines. The code is illustrated by displaying the bit-map image 'peppers.bmp' which can be obtained from BlackBoard:

```
package yourname;
// Default android imports
import android.app.Activity;      import android.app.ActionBar;
import android.app.Fragment;  import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;   import android.view.Menu;   import android.view.MenuItem;
import android.view.View;               import android.view.ViewGroup;
import android.os.Build;
//Added imports for this app
import android.widget.Button;       import android.widget.ImageView;
import java.io.File;                    import java.io.FileNotFoundException;
import java.io.FileOutputStream;  import java.io.IOException;
import java.io.InputStream;          import java.io.OutputStream;
import android.graphics.Bitmap;   import android.graphics.BitmapFactory;
public class MainActivity extends Activity
{ void msImage(String filename)
   { // Reads from an image file into an array for image processing in Java.
```

```
        try { int res_id = getResources().getIdentifier(filename, "raw", getPackageName() );
            InputStream image_is = getResources().openRawResource(res_id);
            int filesize = image_is.available();          //Get image file size in bytes
            byte[ ] image_array = new byte[filesize]; //Create array to hold image
            image_is.read(image_array);                   //Load image into array
            image_is.close();                             // Close in-out file stream
            // Add your code here to process image_array & save processed version to a file.
            File newImageFile = new File(getFilesDir(), filename);
            OutputStream image_os = new FileOutputStream(newImageFile);
            image_os.write(image_array, 0, filesize);
            image_os.flush();
            image_os.close();
            //Display the processed image-file
            Bitmap newBitmap = BitmapFactory.decodeFile(newImageFile.getAbsolutePath());
            // Create ImageView object
            ImageView image = (ImageView) findViewById(R.id.imageView1);
            image.setImageBitmap(newBitmap);
        } // end of try
          catch (FileNotFoundException e) { Log.e("tag", "File not found ", e);}
          catch (IOException e) { Log.e("tag", "Failed for stream", e); }
    } //end of msImage method
    @Override
    protected void onCreate(Bundle savedInstanceState)
    { super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
        //Get image display area as set in the xml file//
        ImageView image = (ImageView) findViewById(R.id.imageView1);
        //Display peppers.bmp from res/raw directory//
        image.setImageResource(R.raw.peppers);
       //Call msImage method to process the image:
       msImage("peppers");
    } // end of onCreate method
} // end of MainActivity class
```

5.5 Reading speech or music from a microphone into a Java array in Android

The following code illustrates how sound read from the microphone in your smartphone may be processed as a Java array and stored, if necessary, as a wav file. There are many ways of doing this, and our code has been compressed and simplified for the purpose of this laboratory. Probably, your smartphone already has a 'voice recorder' application and there are many alternatives to be downloaded. These applications offer a wide range of recording formats usually involving compression. Adaptive multi-rate (AMR) compression for narrowband speech, as traditionally used in cellular mobile telephone calls, offers the lowest bit-rate, but we may require higher quality and a wider frequency range. MP3 and its successors AAC, m4A and many others are often used by voice recorders, but there is often no choice as to which format is being used. For applications where the highest quality is required, uncompressed wav files remain viable, though there is a 'free lossless audio codec' FLAC which is also worthy of consideration. Where the smartphone is required to do some processing, perhaps on small segments of digitized sound, direct uncompressed sampling as used to produce wav files may be the most convenient approach. It is the approach used by the code listed below. As it requires permission to use the microphone and the Internet, the

AndroidManifest.xml file must be modified by including the following statements (before <Application) :

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

The code is presented as an application without user-interface which you have to add. The application will read a 10 second segment of monophonic sound sampled at Fs Hz from a microphone. This produces NS = 10×Fs samples each of wordlength 16 bits. The user-interface requires at least three buttons: 'recButton' for recording, 'playButton' for playing back, and 'killButton'. It should be straightforward to complete the app and implement it on a real mobile phone. There are then many possibilities for adapting it to more specific applications.

There is a deliberate flaw in the design of this application because it is not multi-threaded. You may be able to demonstrate the consequences of this flaw, and correct it. Remember that a wav file has a 44 byte header which specifies the sampling rate (Fs), the number of bits per sample (NB), the number of samples (NS) and some other parameters. A suitable header is generated by the code given below without specifying all its detail.

```
import android.media.MediaRecorder;   import android.media.AudioRecord;
import android.media.AudioFormat;       import android.media.AudioTrack;
import android.media.AudioManager;    import android.os.Bundle;
import android.support.v4.app.ActivityCompat;  import android.support.v7.app.AppCompatActivity;
import android.util.Log;   import android.view.View;
import android.widget.Button;
import java.io.File;  import java.io.FileOutputStream;  import java.io.DataOutputStream;
import java.io.IOException;   import android.os.Environment;

public class MainActivity extends AppCompatActivity
  { private static final String LOGTAG = "AudioRecordTest";
    private String wavFileName = null;
    private int Fs = 8000;                  // Sampling freq (Hz)
    private int NS = sampleRate*10;  // Number of samples
    private short soundSamples[] = new short [(int) NS];
    private int recBufferSize = AudioRecord.getMinBufferSize
                    (Fs, AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT);
    private int recChunks = ((int) (NS/recBufferSize));
    private AudioRecord recorder = new AudioRecord ( MediaRecorder.AudioSource.MIC, (int) Fs,
          AudioFormat.CHANNEL_IN_MONO,AudioFormat.ENCODING_PCM_16BIT, recBufferSize*2 );
    private AudioTrack track = new AudioTrack
       ( AudioManager.STREAM_MUSIC, Fs, AudioFormat.CHANNEL_OUT_MONO,
         AudioFormat.ENCODING_PCM_16BIT, NS*2, AudioTrack.MODE_STATIC );
     private int[] wavHeader = {0x46464952, 44+NS*2, 0x45564157, 0x20746D66,16, 0x00010001,
                                Fs, Fs*2, 0x00100002, 0x61746164, NS*2};
   @Override
   public void onCreate(Bundle savedInstanceState)
    { super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      wavFileName = getExternalFilesDir(null) + "/Barry.wav";
      Log.e(LOGTAG, "Wav file is " + wavFileName);
      // Get the following buttons as defined in Res dir xml code
      final Button killButton = findViewById(R.id.btnKill);
      final Button recButton = findViewById(R.id.btnRecord);
      final Button playButton = findViewById(R.id.btnPlay);
```

```
      // Make the buttons receptive to being clicked
       killButton.setOnClickListener
          (new View.OnClickListener() { public void onClick(View v) { System.exit(0); }  } );

       recButton.setOnClickListener
          (new View.OnClickListener()
           { public void onClick(View v)
             { recButton.setText("RECORDING");   recorder.startRecording();
               int i;
               for(i = 0; i < recChunks; i++ )
                  { recorder.read(soundSamples, i*recBufferSize, (int) recBufferSize);}
               recorder.read(soundSamples, i*recBufferSize, NS - i*recBufferSize);
               recorder.stop();   Log.e(LOGTAG, "Finished recording");
             try { File wavFile = new File(wavFileName);
                   FileOutputStream wavOutputStream = new FileOutputStream(wavFile);
                   DataOutputStream wavDataOutputStream = new DataOutputStream(wavOutputStream);
                   for (i = 0; i < wavHeader.length; i++)
                     { wavDataOutputStream.writeInt(Integer.reverseBytes(wavHeader[i]));  }
                   for (i = 0 ; i < soundSamples.length ; i++ )
                     { wavDataOutputStream.writeShort(Short.reverseBytes(soundSamples[i]));  }
                   wavOutputStream.close();  Log.e(LOGTAG, "Wav file saved");
                 } catch (IOException e) { Log.e(LOGTAG, "Wavfile write error");}
             recButton.setText("DONE");
           } // end of onClick method
          }); // end of recButton.setOnClickListener

       playButton.setOnClickListener
          ( new View.OnClickListener()
             {  public void onClick(View v)
                { playButton.setText("PLAYING");
                  track.write(soundSamples, 0, (int) NS);  track.play();
                  while(track.getPlaybackHeadPosition() < NS) { };  //Wait before playing more
                  track.stop();  track.setPlaybackHeadPosition(0);
                 while(track.getPlaybackHeadPosition() != 0) { }; // wait for head position
                 playButton.setText("PLAY");  // for next time
                }   //end of onClick method
             } ); //end of playButton.setOnClickListener
    } // end of overrided onCreate
  } // end of MainActivity class
```

6. Bit-Strings

Here is some software that may help you. You may find alternatives in the usual places on-line. The following Java methods convert an array of L positive 16 bit integers to a bit-String and vice-versa. They may be adapted to convert normal text strings to bit-strings and vice-versa.

```
public static String IntA2bitS(int[ ] IntA, int L)
{// Converts an array IntA of L positive 16-bit integers to a bit-string bitS
    String bitS = "";
    for (int i = 0; i < L; i++)
      { int d = IntA[i];
        for (int j = 0; j < 16; j++)
```

```
            { int k=i*16+j;  bitS = bitS + String.valueOf(d & 1);   d = d >> 1;  } }
    return bitS
} // end of method

public static void bitS2IntA(String bitS, int L, int[ ] IntA)
{// Converts a bit string bitS representing 16*L bits to L positive 16-bit integers
  for (int i = 0; i < L; i++)
    { int d=0;
      for (int j = 0; j < 16; j++)
         {  int k = i*16+j; d = d + (Character.getNumericValue(bitS.charAt(k))<< j); } IntA[i]=d; }
} // end of method
```

Note that a String in Java is an object. It is not a primitive data type like int, double or char. A literal like "Robert" is a String object in Java. Strings are immutable which means that, once defined, they will never change. Concatenation and other methods will create/return a new object with the new changed value. When comparing Strings, use the "equals" method and not the == operator. Also, it is best to avoid the + operator for concatenating strings. Instead use the "concat" method. Be careful if you are going to concatenate strings many times.
A char array is different from a string. It can hold a sequence of characters just like a string but it is not immutable. Here is an example:

```
char[ ] C = { 's', 'm', 'a', 'r', 't', 'p ', 'h', 'o', 'n', 'e' };
```

A char array can be converted to a string as follows:

```
String S = new String (C);
```

A string X may be converted to a char array cA as follows:

```
String X = “smartPhone”;
char [ ] cA = X.toCharArray( );
```

Finally, note that characters in Java strings and char arrays are represented by 16-bit Unicode rather than 8-bit ASCII code.

7. The Sub-tasks of Task 5

Part 5.1 [4 marks]

Modify the application you developed for the previous Task such that it converts a short one-line text-message to a bit-string and sends it to ‘BarryBot’ using <mode>MSG. The app should allow you to specify the mode. Initially specify mode 0 so that the simulated return channel should not introduce any bit-errors. Convert the received bit-string back to a text-string and display it on the emulated or real smartphone screen.

Send your message several times to barryBot using modes 1, then 2, then 3. Comment briefly on what you receive back. Note that the simulated return channel only ever changes ‘0’ to ‘1’ or vice-versa.

Questions

1. Explain and demonstrate the code for the conversion and re-conversion.

2: How did you enter the message and did you get back the same message exactly?

3: Although the simulated channel should not deliberately introduce any bit-errors in mode 0, what would happen if a real bit error occurred due, perhaps, to a malfunction of the mobile phone? How would your program deal with this occurrence?

4: How does this form of transmission increase the required bit-rate over the simulated channel?

5: Why is this form of transmission useful for experimental purposes?

6: What are your brief observations about the effect of bit-errors?

7: How realistic is it to assume that bit-errors will normally be evenly spread out in time?

Part 5.2 [4 marks]

Investigate the ENCRYPT special command sent to barryBot as detailed in Section 4 above. Send an ENCRYPT message repeatedly to barryBot using '0MSG' on Port 9999. Use mode 0 to avoid any bit-errors. The encrypted messages arrive in bit-string form. If they are converted to ordinary text string form you will not be able to read them. Process the bit-string responses to determine (crack) the encryption key. Then show that you can decipher all subsequent secret messages until barryBot is restarted and chooses a new key.

Remember that if messages A1 and A2 are encrypted by the same key K to obtain S1 = A1⊕K and S2 = A2⊕K, where ⊕ is 'exclusive or', then:

$$S1 \oplus S2 = (A1 \oplus K) \oplus (A2 \oplus K) = (A1 \oplus A2) \oplus (K \oplus K) = A1 \oplus A2.$$

If you have some way of knowing or guessing message A1 say, then you can get A2 as (A1⊕A2) ⊕ A1. You will know when you have guessed A1 correctly when you can read message A2. If you know A2, since you have S2, you can get the key K and decipher all subsequent messages.

Questions

1: What were the messages?

2: Assuming that this experiment demonstrates that using the same key more than once is dangerous, how could you avoid this danger while still allowing the receiver to de-encrypt your messages?

3: Briefly summarize your de-encryption method and indicate whether it really worked.

4: If you could not predict that an email signature will be sent regularly, what other weaknesses could make the encryption vulnerable to attack.

Part 5.3 [4 marks]

Download a speech or music file from BlackBoard and store it in 'res/raw' as explained in Section 5 above. Introduce a 'stop' button as well as the 'start' button for a sound player and check that the sound plays correctly.

Then download the 512x512x3 image files peppers.bmp and peppers.jpg to res/raw and allow the user to display either of these files. Compare them. To investigate how bit-errors would affect these images, you could transmit them to barryBot. However, to save time, this has already been done for you. The images damaged by bit-errors, were stored in a bit-map file called 'damagedpeppers.bmp' and a JPEG file called 'damagedpeppers.jpg'. These files are provided on Blackboard. The damage was caused by (a) bit-errors and (b) lost packets. The packet size was 256 x 3 = 786 bytes and packets start at sample 0, 786, 2x786, 3x786, etc. Each lost packet was replaced by a sequence of 786 consecutive zero-valued bytes. Bit-errors can occur anywhere except in lost packets. Download the damaged bmp and JPEG files into the 'raw' directory on your mobile phone, display them and observe the effect of the bit-errors.

The bit-error probability for 'damagedpeppers.bmp' was 0.05 (which is 5%) whereas for damagedpeppers.jpg', it was 10^{-4} (which is 0.01 %). Increasing the bit-error rate closer to 1% for 'jpg' images made the images totally unreadable. The techniques presented in this Task would allow you to process these images to hide some of the effects of bit-errors and lost packets, but this is not required for the marks. You just need some ideas about what you would do.

Questions

1: What happens to your sound player if you keep pressing the 'start' button?

2: Why does a 'wav' file have a 44 byte header, and how are the sound samples themselves stored? Are the samples stored in text form?

3: How did you display the original undamaged images?

4. Did you detect any difference between the bit-map image and the jpeg version?

5: What was the effect of the bit-errors and lost packets on the bit-map image?

6: How could you conceal the distortion caused by these transmission-errors (if you had time)?

7: What was the effect of the transmission errors on the JPEG image?

8: Why are 'jpg' images much more sensitive to bit-errors and lost packets than 'bmp' images?

Part 5.4 [4 marks]

A recording of monophonic narrowband speech file sampled at 8 kHz with 16 bits per sample has been stored in a wav file called 'damagedspeech.wav' and is provided on Blackboard. The damage was caused by (a) bit-errors and (b) lost packets. The packet size was 200 samples and packets start at sample 0, 200, 400, 600, etc. Each lost packet was replaced by a sequence of 200 consecutive zero-valued 16-bit samples. Bit-errors can occur anywhere except in lost packets. Download the wav file into your mobile phone and play it out using AudioPlayer as in Section 5.1. Then read the file into a Java array and play it out again using 'AudioTrack'. Ignore the header information stored in the first 44 bytes of the wav file. Develop Android code to recognize and deal with the bit-errors and lost packets in the speech transmission. In this experiment, there are no CRC checks, therefore you have to find a way of identifying some of the more serious 'spikes' caused by bit errors and smoothing them over. The simplest way of dealing with packet loss is to replace each lost packet with a copy of the previous correct packet.

Questions

1: How does the receiver know when a packet has been lost in these experiments, and how does the mechanism provided differ from that used by the real time transport protocol RTP.

2. How successful is your simple method of dealing with lost packets in speech transmissions?

3. How successful is your simple method of dealing with bit-errors in speech transmissions?

Part 5.5 [4 marks]

Complete the microphone application detailed in Section 5.4 and implement it on an Android Virtual Device (AVD) or a real smartphone or tablet. Send a short narrowband voice recording

(with Fs = 8000 Hz) to BarryBot and demonstrate the effect of the five possible bit-error rate modes. Finally, play back the 8 kHz sampled sound with a 12 kHz sampling rate to observe the effect.

Questions:

1. What was, or could be, the effect of the design flaw.

2. How could you correct the design flaw?

3. Demonstrate the effect of the bit-errors and packet-loss in modes 0 to 5.

4. What was the effect of increasing the sampling rate to 12 kHz on play-back?

5. Why is it useful to be able to develop your own application for speech recording or processing, rather than relying on downloaded apps. Suggest a useful application that you now know how to develop.

8. Demonstration, Report and Code submission

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 the Task. You must submit to Blackboard by the specified deadline:

(a) a report on Task 5 and

(b) your documented code.

The report must explain 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 Task 5. You may submit two files to Blackboard: (a) your report ideally in pdf and (b) your code in zipped form. You will not get a good mark without reasonable retort style and documentation.

Last modified by Barry 24/3/2019