Haorui Chen_Task5

2019-04-29

Menu

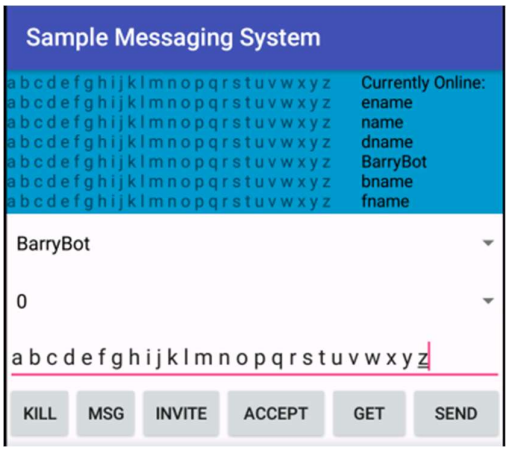
1. Part 5.1

1.1 Development

In this part my goal is to convert 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, and convert the received bit-string back to a text string and display it on the emulated or real smartphone screen.
In this part I added a spinner to select the mode, it is placed above the spinner for selecting online user. Default mode is set to 0. I implemented the part of converting a text string to bit-array in function of MSG button, as sending some content to barryBot is already implemented there. and used the given function IntA2bitS for help. This part of code is given in pic 1-2 and 1-3. For converting the received bit-string back to a text string and display it on the emulated or real smartphone screen, the code is placed in runOnUIThread. Explanation of code is given in 1.4 question 1. This part of code is given in pic 1-4 and 1-5.

1.2 Test

I used mode 0 to send messages to barryBot first and no errors are supposed to be introduced. I tested it several times and got what I want. The result of multiple trials under mode 0 are shown in pic 1-1.



1-1 result of multiple trials under mode 0

In mode 1, sporadic bit-errors shall be introduced evenly across the string; in mode 2, intense bit-errors shall be introduced evenly across the string; in mode 3, bit-errors shall appear in cluster. Multiple trials under these modes prove that their implementation is successful. Observation for these 3 modes is given in 1.4 question 6 and corresponding result in pic 1-6, 1-7 and 1-8.

1.3 Evaluation

By finishing this part, I know that implementation of int array to bit array conversion and re-conversion is working properly. Also I get a direct image of how bit-errors are implemented in mode 1 to 3.

1.4 Q&A

1. Explain and demonstrate the code for the conversion and re-conversion.
A: in convention ,we are converting strings to bit arrays. There are 3 steps to go: 1-String to character array that can be done by method toCharacterArray; 2-character array to int array, using a loop in which presentation of characters are changed to Unicode is also enough; 3-int array to bit array, which is done by given function. Relevant code is shown in pic 1-2 and 1-3. For re-conversion, there are also 3 steps: 1-bit array to int array by given function, 2-int-array to char array by a loop and 3-char array to string by a string construction method. One more thing needs to say is that a new variable 'flag' is implemented to help runOnUIThread distinguish what should do when we want to

send different stuffs to barryBot, for conversion and re-conversion I used mode 2. Relevant code is shown in pic 1-4 and 1-5.

```
msgButton.setOnClickListener((v) → {
        EditText editText = (EditText) findViewById(R.id.editText);
        message = editText.getText().toString();
        charmes = message.toCharArray();
        len = message.length()+1;
        //Log.e(LOGTAG, "raw:"+message);
        //Log.e(LOGTAG, "len:"+len);
          for(char c:charmes )
              Log.e(LOGTAG, String.valueOf(c));
        i = 0;
        intmes =new int[len];
        for(char c:charmes){

            intmes[i] = c;
            //Log.e(LOGTAG, "inted:"+intmes[i]);
            i++;
        }
        Log.e(LOGTAG,  msg: "message sent:"+IntA2bitS(intmes, len));
        // OnClick actions here
        // Instantiate the transmitter passing the output stream and text to it
        if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that output stream has be setup
            networkConnectionAndReceiver.setFlag(2);
            networkConnectionAndReceiver.len=len;
            Transmitter transmitter = new Transmitter(networkConnectionAndReceiver.getStreamOut(),
                    transmitterString: spinmode.getSelectedItem()+msg+spinner.getSelectedItem()+' '+IntA2bitS(intmes, len));
            transmitter.start();        // Run on its own thread
        }
});
```

1-2 code for string to bit array conversion

```
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
```

1-3 helper function IntA2bitS

```
if(flag==2){
    intrev = new int[len];
    chrrev = new char[len];
    //message = new StringBuilder(message).reverse().toString();
    Log.e(LOGTAG,  msg: "message sent:"+message.substring(24));
    bitS2IntA(message.substring(24), len, intrev);
    i = 0;
    for(int n:intrev){
        Log.e(LOGTAG,  msg: "re-inted:"+intrev[i]);
        chrrev[i] = (char)n;
        i++;
    }
    String strrev = new String(chrrev);
    receiverDisplay.setText(strrev + "\n" + receiverDisplay.getText());
}
```

1-4 code for bit array to string conversion

```
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 met
```

1-5 helper function bitS2IntA

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

A: the content of message is entered in the text input box that is used in lab4. Mode will be chosen from a spinner

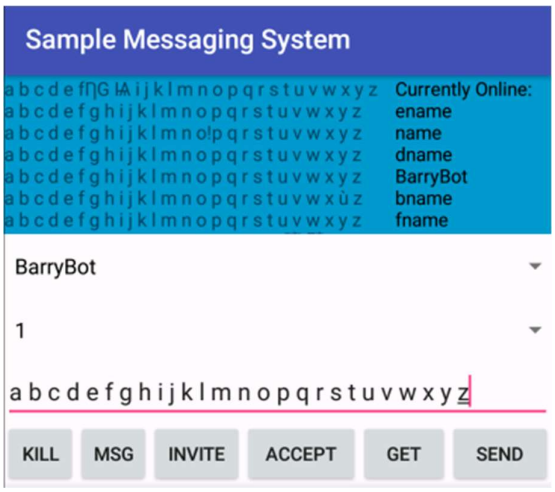
that gives 6 modes (0 to 5) when opening the app and message receiver is also chosen from spinner like what we have done in lab4. The final string is concatenated by <mode>MSG+<target user from spinner>+<content in text input box>. For mode 0 I can get back exactly the same message, yet for mode 1, 2 and 3, since we are deliberately introducing bit-errors, messages become different.

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?
A: for text message, characters may change, as shown in mode 1, 2 and 3; for picture, part of colors may become strange; for audio, clicks or distortions may appear; for video, effect may be combined by that of picture and audio. since without CRC we even don' t know whether bit-errors occurred, we cannot deal with it.

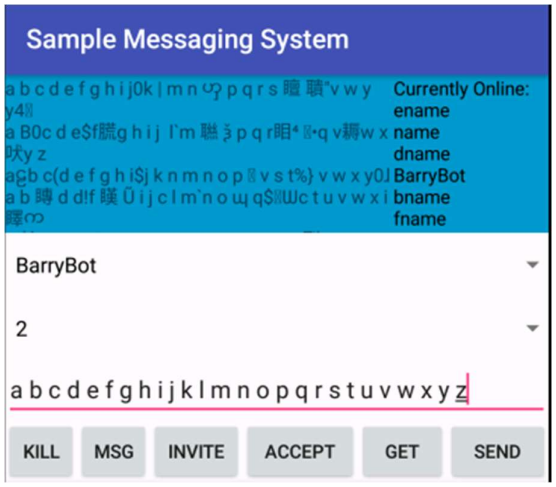
4. How does this form of transmission increase the required bit-rate over the simulated channel?
A: originally, data are bit-arrays in essence. Yet over the simulated channel, each bit is presented in '0'/'1' character, that's 16 bits in Java, which enlarges size of strings by 15 times.

5. Why is this form of transmission useful for experimental purposes?
A: Because we intend to present changes to text with different form of errors introduced in transmission, and 'bit-arrays' in the form of string is easier to operate on.

6. What are your brief observations about the effect of bit-errors?
A: for error in mode 1, the effect is like that in pic 1-6. I observe that most part of strings remain normal, yet errors appear sporadically. Yet in mode 2, errors appear more intense yet still evenly distributed across the whole string, as presented in pic 1-7.

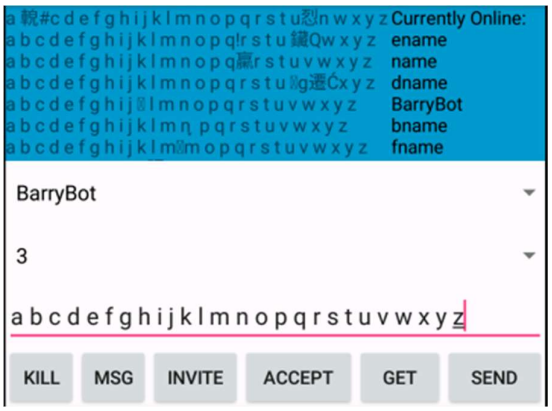


1-6 effect of bit-error in mode 1



1-7 effect of bit-error in mode 2

Under mode 3, it is clear that errors occur in cluster as part of errors are longer than what is presented in mode 1, and distribution is uneven, different from what is shown in pic 1-6 and 1-7. it can be seen from pic 1-8.



1-8 effect of bit-error in mode 3

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

It is realistic to some degree, as it can simulate the everlasting random appearance of source of interference; or to say, if we are sending a bit-array that is long enough, the bit-errors will surely be evenly distributed. However, usually we don't have such a huge bit array to send, as they are normally be separated into packages, plus when sending a shorter message, source of interference will not appear that often to make errors appear to be 'normally distributed', thus the assumption is to be qualified.

2. Part 5.2

2.1 Development

In this part my goal is to investigate the ENCRYPT special command sent to barryBot. Send an ENCRYPT message repeatedly to barryBot using '0MSG' on Port 9999. Use mode 0 to avoid any bit-errors. 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.

The procedure of developing my code is given in 2.4 question 3, relevant code is given in pic 2-1, 2-2 and 2-3. Bit array representation in this part is made up of ASCII instead of Unicode and sequence of bits in each array is like that in normal PC, so an extra function bitS2IntA2 is added for bit array to int array conversion. I got length of original string A1 an A2 by dividing length of bit array by 8 as one character is represented by 8 bits. Only for this part, message sent back by BarryBot don't have a 24- character-long prefix, so we put the whole message to conversion.

If S1 is still not got, then a newly sent back encrypted, converted string is printed and saved for later comparision, else it's regarded as S2 and converted to char array together with S1, A1 to be applied to formula (S1⊕S2)⊕A1. The result char array is then converted to string A2 and printed out t see if my guess of A1 is correct.

Application layout remain unchanged for this part.

```
if(flag==2||flag==3){
    if(flag==3)
        len = message.length()/8;
    intrev = new int[len];
    chrrev = new char[len];
    //message = new StringBuilder(message).reverse().toString();
    //Log.e(LOGTAG, "message sent:"+message.substring(24));
    switch (flag) {
        case 2:
            bitS2IntA(message.substring(24), len, intrev);
            break;
        case 3:
            bitS2IntA2(message, len, intrev);
    }
    i = 0;
    Log.e(LOGTAG,  msg: "len:"+len);
    for(int n:intrev){
        //Log.e(LOGTAG, "re-inted:"+intrev[i]);
        chrrev[i] = (char)n;
        i++;
    }
    strrev = new String(chrrev);
    if(flag==3&&found==false)
        receiverDisplay.setText("encrypted:"+strrev + "\n" + receiverDisplay.getText());
```

2-1 codes added in part 4.2-first part

```
if(flag==3&&found==false)
    receiverDisplay.setText("encrypted:"+strrev + "\n" + receiverDisplay.getText());
if(flag==3){
    if(found==false){
        if(count==0){
            temp[count]=strrev;
            count++;
        }
        else {
            for(i=0;i<count;i++){
                if(temp[i].equals(strrev)==true){
                    S1=strrev;  found=true; S1_chr = S1.toCharArray();  A1_chr = A1.toCharArray();
                    receiverDisplay.setText("decrypted" + "\n" + receiverDisplay.getText());
                    break;
                }
            }
            if(i==count){
                temp[count]=strrev;
                count++;
            }
        }
    }
    else {
        S2=strrev;
        S2_chr = S2.toCharArray();
        A2_chr = new char[S1.length()];
        Log.e(LOGTAG,  msg: "S1_len:"+S1.length());
        Log.e(LOGTAG,  msg: "S2_len:"+S2.length());
        for(i=0;i<S1.length();i++){
            A2_chr[i] = (char)(S1_chr[i]^S2_chr[i]^A1_chr[i]);
        }
        A2 = new String(A2_chr);
        receiverDisplay.setText("decrypted:"+A2 + "\n" + receiverDisplay.getText());
    }
}
```

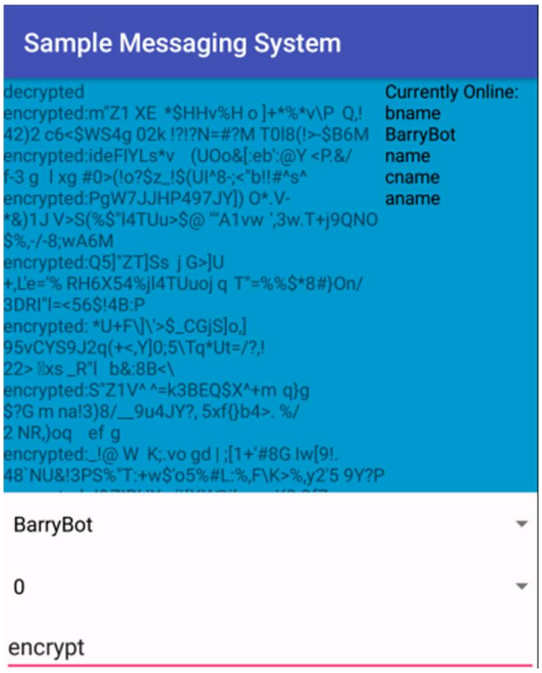
2-2 codes added in part 5.2-second part

```
public static void bitS2IntA2(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<8; j++){
            d<<=1;
            int k = i*8+j;
            d = d + Character.getNumericValue(bitS.charAt(k));
        }
        IntA[i]=d;
    }
} // end of met
```

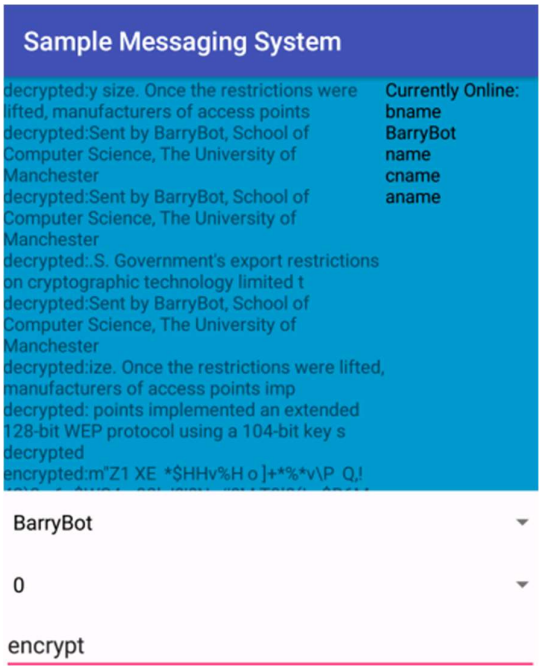
2-3 codes added in part 5.2-third part

2.2 Test

In this step, testing is mainly for whether assumed version of given array A1 is correct. By applying A2=(S1⊕S2)⊕A1 (detail is given in 2.4 question 3), I sent multiple 'encrypt' command to BarryBot, detail is given in 2.4 question 3, it can be observed from pic 2-4 and 2-5 that the string in 2.4 question 1 is a correct guess of string A1.



2-4 trials of sending 'encrypt' till encrypted version of signature is found and encryption thus can be done



2-5 every trial of sending 'encrypt' after decryption can get us the decrypted version of string sent back

2.3 Evaluation

After this step, A simple decrypting program is created and have its effectiveness checked, giving me a brief glance of how encryption is done.

2.4 Q&A

*1. What were the messages?

A: the signature is "Sent by BarryBot, School of Computer Science, the University of Manchester".

Standard 64-bit WEP uses a 40 bit key (also known as WEP-40), which is concatenated with a 24-bit initialization vector (IV) to form the RC4 key. At the time that the original WEP standard was drafted, the U.S. Government's export restrictions on cryptographic technology limited the key size. Once the restrictions were lifted, manufacturers of access points implemented an extended 128-bit WEP protocol using a 104-bit key size (WEP-104).

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?
A: we can regard the key of server as public key and assign a private key to sender and receiver of the message that is only accessible to them. Even protection of public key is deciphered, without knowing private key it is also impossible to decipher the original string. Private key can be set from content of content of messages.

3. Briefly summarize your de-encryption method and indicate whether it really worked
A: I first did the work of guessing the hidden part of the given string "Sent by BarryBot, School of Computer Science, XXXXXXXXXXXXXXXXXXXXX". From the given part, I assume the whole string will be like to indicate the belonging of barryBot, so I putted "University of Manchester" there, yet according to later work of decryption, this version did not work, so I changed it to "the University of Manchester" and it was proved to be correct.
Then it's the work of decryption—and see if an assumption of signature is correct. Since the encrypted version of signature will appear regularly, I sent 'encrypt' command several times, store them and see if a newly got string is same as one of saved ones, if it is then it's the target string.
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⊕S2 = (A1⊕K) ⊕ (A2⊕K) = (A1⊕A2) ⊕ (K⊕K) = A1⊕A2. If you have some way of knowing or guessing message A1 say, then you can get A2 as (A1⊕A2)⊕A1.
Now that we already know given string A1, the encrypted version S1 and an encrypted version of any other string S2, we can get its original A2 by calculating (S1⊕S2)⊕A1. I put the decrypted version on the text box and check whether it's readable. If it is, then I know my guess of A1 is right.
For string "Sent by BarryBot, School of Computer Science, the University of Manchester", we get the result shown in pic 2-5, in which all other random strings are readable, thus shows that my guess is right.

4. If you could not predict that an email signature will be sent regularly, what other weaknesses could make the encryption vulnerable to attack.
A: we can send a string and set the receiver to yourself, thus we can also get its encrypted version, and key can by figured out; or we can assume current received string S1 is the encrypted version of signature, and by figuring out the assumed key and try to use it to decrypt another enigmatic string, we will know we got correct key if we decrypted another readable string.

3. Part 5.3

3.1 Development

In this stage my goal is to download a speech or music file and store it in 'res/raw'. introduce a 'stop' button as well as the 'start' button for a sound player and check that the sound plays correctly. 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.
I opened a new activity and 3 horizontal layouts for presenting these 5 images clearly, details of every picture are stored in an object belongs to InputStream class, showing the essence of image is also array. Class OutputStream and File is used to printing the image on UI. A new button is added to change activity to original one. Relevant code is shown in pic 3-1, 3-2, 3-3 and 3-4.

```
bt1_2 = (Button) findViewById(R.id.btnchg);
Button.OnClickListener listener = (v) → {
        Intent intent = new Intent( packageContext: MainActivity.this, Main2Activity.class);
        startActivity(intent);
        MainActivity.this.finish();
};
bt1_2.setOnClickListener(listener);

mySound1 = MediaPlayer.create( context: this, R.raw.narrobandspeech);
```

3-1 setting up media player and button for change page

```
public void playSound1(View view) { mySound1.start(); }

public void stopSound1(View view) { mySound1.release(); }
```

3-2 callback function for starting and stopping audio

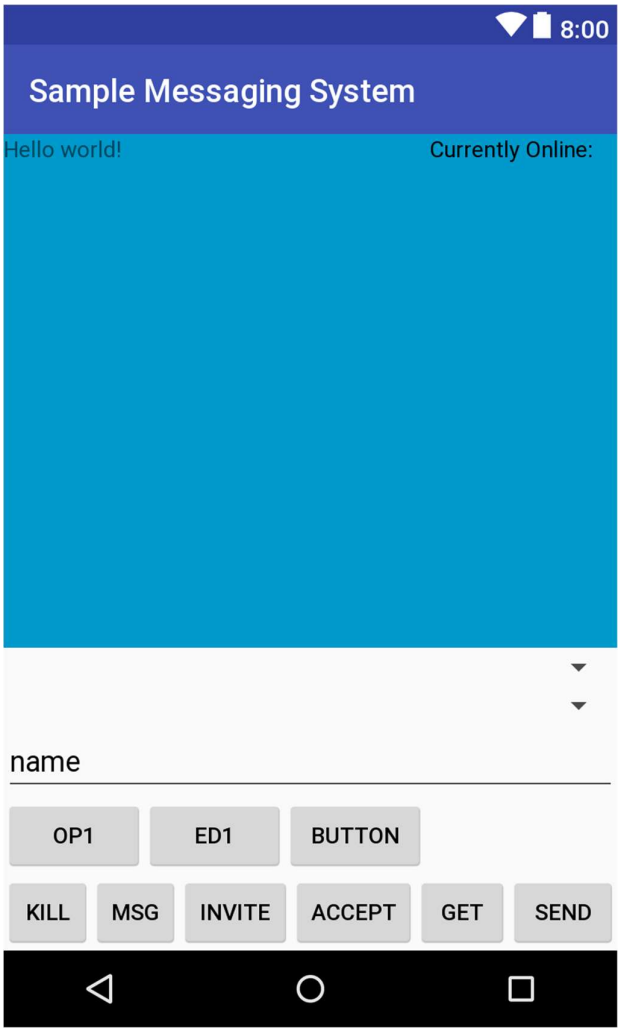
```
public class Main2Activity extends AppCompatActivity {

    private static final String LOGTAG = "Minor UI"; //Logcat messages from UI are identified
    Button bt2_1;

    void msImage(String filename, int id){ // Reads from an image file into an array for image processing in Java.
        try {
            int res_id = getResources().getIdentifier(filename, defType: "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 streamR.id.imageView1
            // 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, off: 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(id);
            image.setImageBitmap(newBitmap);
        } // end of try
        catch (FileNotFoundException e) { Log.e( tag: "tag", msg: "File not found", e);}
        catch (IOException e) { Log.e( tag: "tag", msg: "Failed for stream", e); }
    } //end of msImage me
```

3-3 basic configuration for a new activity

For playing the audio, I added 2 buttons, 1 for playing, the other for stopping, the audio. the 2 buttons used callback functions playSound(view) and stopSound(view) respectively, to be in detail, they used methods from class MediaPlayer which object will contain the whole audio file. A new button is added to change activity to the new one Relevant code is shown in pic 3- .The final layout of development is shown in pic 3-5 and 3-6.


8:00
Sample Messaging System
Hello world!
Currently Online:
name
OP1
ED1
BUTTON
KILL
MSG
INVITE
ACCEPT
GET
SEND


vodafone UK|CMCC
CMLink
17:31
Sample Messaging System
peppers.bmp
peppersjpg.jpg
damagedpeppers.bmp
damagedpeppersjpg.jpg
verydamagedpeppers.bmp
BUTTON

3-5 layout of original activity

3-6 layout of newly set activity for pics

3.2 Test

Clicking 'OP1' button will get the audio played till it ends and 'ED1' button will end it in advance, illustrating the two buttons and code for playing audio is working properly. Click button 'BUTTON' can switch me to the new page and we can see 5 images are presented normally, showing code for displaying images are working. Click button 'BUTTON' and we can get back to the original page.

3.3 Evaluation

By finishing this step, I had experience of introducing audio and image files and playing them, also I got to know the characteristics of bmp and jpg files. Information in manual also gave me basic ideas of image processing for transmission.

3.4 Q&A

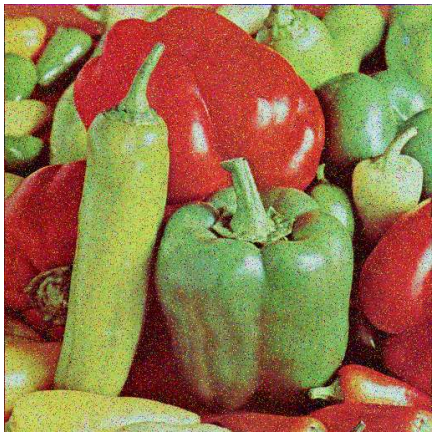
1. What happens to your sound player if you keep pressing the 'start' button?
A: when the audio is playing, clicking the same button will cause no effect; when playing is finished, clicking it will cause the app to force quit.

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?
A: this 44 bytes are metadata of audio file, including its . sound sample are stored in a short type array with its length equals to sampling rate*duration, each sample is an element in the array and is not in text form.

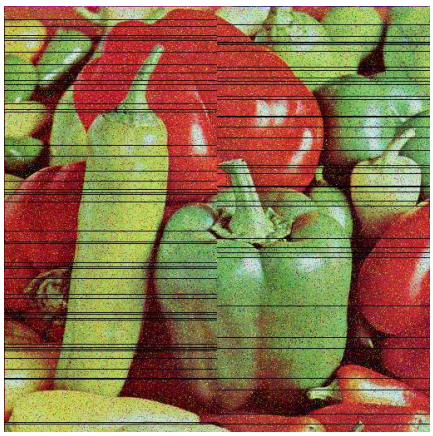
3. How did you display the original undamaged images?
A: I added an Imageview bar on UI interface and used code from manual to display the original image. The code is given in pic 3-4.

4. Did you detect any difference between the bit-map image and the jpeg version?
A: their quality looks exactly the same, the only difference is that the bit-map image is a bit larger than jpeg version.

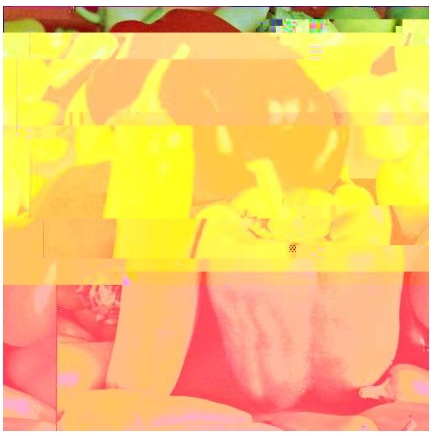
5. What was the effect of the bit-errors and lost packets on the bit-map image?
The effect of bit-error is introducing evenly distributed noise points across the whole picture, lowering the overall quality of it, the effect is shown in pic 3-7; the effect of lot packets is losing a whole shade of color for a particular part of image, introducing black lines across the whole image, left and right halves of the image is also not concatenated correctly, the effect is shown in pic 3-8.



3-7 effect of bit errors in bit-map



3-8 effect of bit errors and lost package



3-9 effect of bit errors in jpeg

6. How could you conceal the distortion caused by these transmission-errors (if you had time)?
For bit-errors, we can seek help from nearby samples like calculating errant sample by average of nearby samples, as they are likely to contain similar information like color and content. For package loss, method for bit-errors also applies and the idea of model recognition may also help. Boundaries and color to fill in can be guessed from nearby package.

7. What was the effect of the transmission errors on the JPEG image?
the effect of lot packets is losing a whole shade of color for a particular part of image, causing losses in a whole shade of color in different parts and components of each part of the image are not connected correctly as we can observe lots of sawtooth, the effect is shown in pic 3-9.

8. Why are 'jpg' images much more sensitive to bit-errors and lost packets than 'bmp' images?
Because jpg is using Huffman coding, ha compressed version, losing many minor details and redundancies that back up the image. When bit-errors are introduced in bmp image, these details and redundancies will work an keep the overall image quality, yet for jpg image, introducing error in one sample will cause more severe problems without help of those details.

4. Part 5.4

4.1 Development

In this part, my goal is to download the wav file 'damagedspeech.wav' into mobile phone and play it out using AudioPlayer as in Part 5.3. 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. Find a way of identifying some of the more serious 'spikes' caused by bit errors and smoothing them over.

For playing audio through audio player, I simply repeated what I've done in audio task part of previous task, relevant code is shown in pic 4-1; for playing through 'AudioTrack', I read it in a array, then AudioTrack class is then used that is initialized by a set of configurations and the audio is played by calling write() method of the whole array, relevant code is shown in pic 4-2 and 4-3. I simply copied the previous segment for substituting missing samples due to package loss, and for bit-errors, I check all samples 3 by 3, set any sample that is larger than mean of other two if their difference is larger than 10000. Also, I used the last sample from previous packet to smooth first samples in current packet, then taking the sample from this packet for the next. Relevant code is shown in pic 4-4.

Application layout after completing this step is given in pic 4-5.

```
public void playSound4(View view) { mySound4.start(); }

public void stopSound4(View view) { mySound4.release(); }
```

4-1 callback function for starting and stopping audio

```
private int NS= Fs*10;// Number of samples
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, bufferSizeInBytes: recBufferSize*2);
//private AudioRecord recorder = findAudioRecord();
private int[] wavHeader ={0x46464952, 44+NS*2, 0x45564157, 0x20746D66,16, 0x00010001,Fs, Fs*2,
        0x00100002,0x61746164, NS*2};
private AudioTrack track = new AudioTrack(AudioManager.STREAM_MUSIC,Fs,
        AudioFormat.CHANNEL_OUT_MONO,AudioFormat.ENCODING_PCM_16BIT, bufferSizeInBytes: NS*2,
        AudioTrack.MODE_STATIC);
```

4-2 variables needed for using AudioTrack

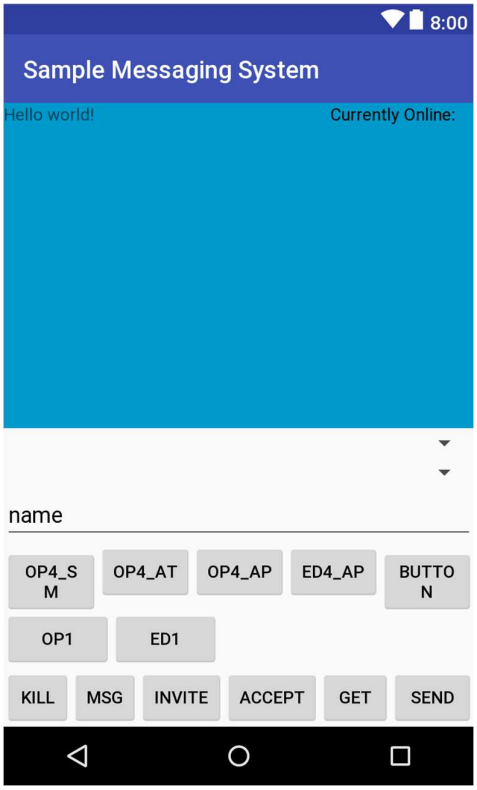
```
void msSpeech(){// Method for reading 16-bit samples from a wav file into array data
    int i;  int b1 =0;  int b2 = 0;
    try{// Make speechis an input stream object for accessing a wav file placed in R.raw
        InputStream speechis=getResources().openRawResource(R.raw.damagedspeech);
        // 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< wav_len ; 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
            dam_spe[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: "tag",  msg: "wav file not found", e);}
    catch (IOException e) {Log.e( tag: "tag",  msg: "Failed to close input stream", e);}
}// end of msSpeech metho

void arrayPlay(boolean smh_flag){
    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,  bufferSizeInBytes: wav_len*2, MDE);
    msSpeech();
    if(smh_flag==true){
        smooth();
    }
    track.write(dam_spe,  offsetInShorts: 0, wav_len);
    track.play();
    while(track.getPlaybackHeadPosition() != wav_len) {};  //Wait before playing more
    track.stop();
    track.setPlaybackHeadPosition(0);
    while(track.getPlaybackHeadPosition() != 0) {}; // wait for head position
}// end of arrayplay method
```

4-3 helper function msSpeech and arrayPlay

```
void smooth(){
    int i=0;
    while(i*200<wav_len){
        //see if this is an empty pack
        for(j=i*200;j<(i+1)*200;j++){
            if(dam_spe[j]!=0){
                break;
            }
        }
        //copy last pack to current pack
        if(j==(i+1)*200 && i!=0){
            for(j=i*200;j<(i+1)*200;j++){
                dam_spe[j]=(short)(dam_spe[j-200]/2);
            }
        }
        if(i!=0){
            for(j=i*200;j<i*200+11;j++){
                //dam_spe[j]=(short)((Math.pow(Math.E,j-i*200)*dam_spe[j]+z_smh)/(Math.pow(Math.E,j-i*200)+1));
                dam_spe[j]=(short)((28*dam_spe[j]+24*dam_spe[j-1]+20*dam_spe[j-2]+16*dam_spe[j-3]
                        +12*dam_spe[j-4]+8*dam_spe[j-5]+4*dam_spe[j-6])/(112));
            }
        }
        j=0;
        while ((j+1)*3<wav_len){
            if(Math.abs(dam_spe[j+1]-(dam_spe[j]+dam_spe[j+2])/2)>10000){
                dam_spe[j+1] = (short)((dam_spe[j]+dam_spe[j+2])/2);
            }
            else if(Math.abs(dam_spe[j]-(dam_spe[j+1]+dam_spe[j+2])/2)>10000){
                dam_spe[j] = (short)((dam_spe[j+1]+dam_spe[j+2])/2);
            }
            else if(Math.abs(dam_spe[j+2]-(dam_spe[j+1]+dam_spe[j])/2)>10000){
                dam_spe[j+2] = (short)((dam_spe[j+1]+dam_spe[j])/2);
            }
            j++;
        }
        i++;
    }
}
```

4-4 function smooth that deals with bit-errors and packet loss



4-5 app layout at this step

4.2 Test

Clicking 'OP4_AP' button will get the audio played till it ends and 'ED4_AP' button will end it in advance. Clicking 'OP4_AT' will play the original damaged wav file from array till it ends; clicking 'OP4_SM' will play the smoothed version of that audio till it ends. Playing these correctly indicates that these three requirements have been correctly met.

4.3 Evaluation

At the end of this step, I successfully played out an audio using AudioPlayer and AudioTrack. Also, I had a glance of detecting error and packet loss under circumstance of practical packet transmission in which CRC check and relevant re-transmission is no longer applicable.

4.4 Q&A

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.
A: for each packet, I will detect if all the elements are 0, those with all-0 elements are lost ones; under RTP, each packet is time-stamped and numbered.

2. How successful is your simple method of dealing with lost packets in speech transmissions?
A: it is successful, as all intervals caused by lost package are filled in by its previous package, causing no more obvious intervals in audio noticeable.

3. How successful is your simple method of dealing with bit-errors in speech transmissions?
A: unfortunately, only minor improvements have been achieved. Clicks in the middle of the audio is smoothed somehow, yet those at the head are not dealt with successfully.

5. Part 5.5

5.1 Development

In this part my goal is to 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.

Button for recording is first implemented, in which permission to use microphone is checked, a 10 sec short audio is read into array and 44-byte headers is abandoned. Helper function checkRecordPermission is used in this part. Then we need to convert the array to bit array, divide it into fragments due to capacity limit of buffer in server, get strings back from server, convert them back to short arrays and concatenate them into final playable version. Helper function IntA2bitS and bitS2IntA is used in this part. Finally, the array is played out under 12kHz. Relevant code is shown in pic 5-1, 5-2, 5-3, 5-4 and 5-5. Final layout is shown in pic 5-6.

```
recButton.setOnClickListener((v) → {
        checkRecordPermission();
        recButton.setText("RECORDING");
        recorder.startRecording();
        int i;
        for(i = 0; i < recChunks; i++ ){
            recorder.read(soundSamples,  offsetInShorts: i*recBufferSize, (int) recBufferSize);
        }
        recorder.read(soundSamples, offsetInShorts: i*recBufferSize,  sizeInShorts: NS-i*recBufferSize);
        recorder.stop();
        Log.e(LOGTAG,  msg: "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,  msg: "Wav file saved");
        } catch (IOException e) { Log.e(LOGTAG,  msg: "Wavfile write error");}
        recButton.setText("DONE");
    });// end ofrecButton.setOnClickListener

playButton.setOnClickListener((v) → {
        playButton.setText("PLAYING");
        Log.e(LOGTAG,  msg: "start playing");
          for(i=0;i<12000;i++)
              Log.e(LOGTAG, String.valueOf(soundResult[i]));

        track.write(soundResult,  offsetInShorts: 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 playButton.setOnClickListener
```

5-1 implementation of button ‘REC’ and ‘PLY’

```
final Button sendAuButton = findViewById(R.id.btnsendAu);
// end of recButton.setOnClickListener
sendAuButton.setOnClickListener((v) → {
        if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that output stream has be setup
            networkConnectionAndReceiver.setFlag(4);
            networkConnectionAndReceiver.len=NS;
            i=0;
            intmes = new int[40];

            for(i=0;i<2000;i++){
                for(j=i*40;j<(i+1)*40;j++){
                    intmes[j-i*40]=soundSamples[j];
                }
                mess = IntA2bitS(intmes, L: 40);
                Transmitter transmitter = new Transmitter(networkConnectionAndReceiver.getStreamOut(),
                        transmitterString: spinmode.getSelectedItem()+msg+spinner.getSelectedItem()+' '+mess, flag: 4);
                //Log.e(LOGTAG, "Send version:"+mess);
                transmitter.start();        // Run on its own thread
                do{
                    bitAudio = networkConnectionAndReceiver.getMessage();
                }while(bitAudio==null);
                if(bitAudio.length()<640&&(spinmode.getSelectedItem().toString().equals("4")||
                        spinmode.getSelectedItem().toString().equals("5"))){
                    bitAudio=new String(subs);
                }
                networkConnectionAndReceiver.setMessage();
                //Log.e(LOGTAG, "string-"+bitAudio);
                //Log.e(LOGTAG, "length"+bitAudio.length());
                bitS2IntA(bitAudio, L: 40,intrev);
                for(j=0;j<40;j++){
                    //Log.e(LOGTAG, "intrev-j="+intrev[j]);
                    soundResult[i*40+j] = (short)intrev[j];
                }
                Log.e(LOGTAG, msg: "i="+i);
            }
        }

    });//end of playButton.setOnClickListener
```

5-2 implementation of button 'SENDAU'

```
private void checkRecordPermission() {

    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.RECORD_AUDIO)
            != PackageManager.PERMISSION_GRANTED) {

        ActivityCompat.requestPermissions( activity: this, new String[]{Manifest.permission.RECORD_AUDIO},
                requestCode: 123);
    }
}

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 void bitS2IntA(String bitS, int L, int[] IntA){
    Log.e(LOGTAG, msg: "string-"+bitS);
    // 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 met
```

5-3 helper function checkRecordPermission, IntA2bitS and bitS2IntA

```
if(flag==4){
    save = message.substring(24);
}
```

5-4 raw processing so that save is pure bit array

```
public String getMessage(){
        return save;
}
public void setMessage(){save=null;}
```

5-5 methods of setting value of save for a primitive synchronization lock

5.2 Test

Clicking 'REC' button will cause it have the effect of rising up and a 10s recording will start. Clicking 'SENDAU' button with a mode selected will send the recording to barryBot and different facts will be attached to it when sent back. Clicking 'PLY' will play the sent back audio. sending the recording to barryBot 6 times using a different mode at one time, different effect will appear when playing it, details are given in 5.4 question 3. Effects attached and audio played correctly indicate that code implementation for this part is correct.

5.3 Evaluation

By completing this step, I successfully build a successful audio player and simulated the effect of playing audio file that has gone under a transmission, getting an idea of what effect will an audio file have when bit-errors at different severity and package loss occur simultaneously or separately.

5.4 Q&A

1. What was, or could be, the effect of the design flaw.
A: 1-buffer of the server is too small, scaling down the size of a packet and thus adding up transmission time; 2-the server is not multi-threaded, also banning us to use the method to accelerating the transmission speed; 3-information shown on server is not clear enough, causing difficulty when developers try to use it to help them debug their code, 4-schema of encryption in part 5.2 is too simple and key remain unchanged throughout lifetime of the server, making protection of secret message not that strong.

2. How could you correct the design flaw?
A: 1-enlarge the buffer; 2-enable multi-threading; 3-giving more detailed message and emolliate the layout so that programmer can figure what messages appear in one dialogue to barryBot; 4-enhancing encryption schema so that it provides better security.

3. Demonstrate the effect of the bit-errors and packet-loss in modes 0 to 5.
A: mode 0 introduce no bit-errors or packet-loss; mode 1 will give obvious noises throughout audio; mode 2 gives even more obvious noises that will even veil the presence of original sound if speaker is not loud enough; mode 3 will give clustered noises, yet it occurs rather intense and make the audio sounds like that of mode 1; mode 4 is supposed to give missing milliseconds in the recording, yet packet-loss occurs too sparsely and last for pretty short time, causing the effect is not that noticeable; mode 5 will give bit-errors and packet-loss simultaneously, again the effect of packet-loss is not obvious, yet noises are noticeable.

4. What was the effect of increasing the sampling rate to 12 kHz on play-back?
A: original pace of recording is accelerated, the duration also gets shorter and the voice gets sharper and sounds different from the original version.

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.
A: because developing own application will help protect our privacy. now we are able to develop a messenger that can send not only secured texts but also audios and pics.