**CHAPTER 1**

**INTRODUCTION**

* 1. **INTRODUCTION**

Communication systems have developed steadily and new means of communications are being developed from time to time. Cell phones have evolved from being simple communication devices to a powerful portable computer.

The instrument has become commercial that it’s available as a key-item in everyone’s pocket, benefitting the user and serving as a lucrative business for the manufacturers. The increase in the number of service providers has made it a battle-field for each of them to gain more customers, and the cheaper one always manages to outstrip others. Service providers such as VODAFONE, CSL and star hub have already earned a name in market and are still expanding their service all over the world. The idea of having common channel (air) for sending innumerable frequencies and use the bandwidth that’s available has always been fascinating for all of us – but at what cost? It is a hectic task to design a communicating system knowing how badly it is susceptible to noise.

There is now emerging demands to have cheap communication within a fixed range, like in an office or a township. Intercom is also a similar system but in this case it has fixed phones connected by guided media. The recent advancements in mobile phone technology have incorporated the features of accessing Wi-Fi from such a small device. The presence of Wi-Fi in the latest mobiles allows the user to access the internet with the help of a Wi-Fi router. Exploiting the entire bandwidth of 2.4 GHz for making voice calls between devices, it eliminates the need of using the service provider’s bandwidth. Hence voice can be made at zero cost.

Most of the latest models of the phones come with Wi-Fi. The number of people using devices has been increasing and may even rise higher this year. Our proposal shall eliminate the usage of service providers for short-distance calls and decreases the cost that gets accrued in the customer’s monthly bill. The base idea is unifying voice and data on to a single network infrastructure by digitizing the voice signals, convert into IP packets and send them through an IP network together with the data information, instead of using a separate telephony network.

**1.2 OVERVIEW OF THE PROJECT**

Voice over Internet Protocol (VOIP) provides the ideas for connecting two clients through voice over the internet. Although IP phone communication over the data networks such as LAN exists but these IP phones are fixed type. We implement wireless IP phone communication using the Wi-Fi network. ANDROID application can be used in mobile devices that is Wi-Fi enabled to communicate with a router. This is very useful tool because by communicating with the router directly, many processes of networking can be simplified. ANDROID provides a robust, flexible environment for applications running on mobile. It includes flexible user interfaces, robust security, built-in network protocols and support for networked and offline applications that can be downloaded dynamically.

Voice communications are more delay sensitive than error sensitive. UDP provides less efficient communication but at rates faster than TCP and also UDP is used for wireless systems. So in transport layer we use UDP protocols for communication. Several signalling protocols have been proposed for IP phone applications.SIP is peer to -peer protocols. Being simple and similar to HTTP, SIP will bring the benefits of WWW architecture into IP telephony and readily run wherever HTTP runs. It is a gradual evolution from existing circuit-switched networks to IP packet-switched network. In this model of communication voice to be transmitted by one user us encrypted using the device.

Then this is packetized are communicated to router through Wi-Fi channel is in the same way data is communicated. The router performs header decryption and finds the destination phone from it routing table. The encrypted voice is sent by same means as communication of data by Wi-Fi.

**1.3 LITERATURE SURVEY**

IP phone communications over the data networks such as LAN exist but these IP phones are fixed type. We implement wireless IP phone communication using the Wi-Fi network. Unlicensed mobile access is 3rd generation partnership program global specification that provides a standard for service providers to merge mobile networks and wireless LANs into a single seamless access networks services for both voice and data. When lacing a call to phone within the network, the number is dialled and it passes to the router. The application then sends the number in 128 bit encrypted from to the route, requesting a call to be placed. The ANDROID application at destination intimates to user of the incoming call to be placed. The ANDROID application at destination intimates the user of the incoming call; if the call is accepted, the router changes both phones status to busy. When busy no further calls can from be made or received by the phones. This is because we are using the limited free bandwidth of the Wi-Fi, so conference calls and similar features are not supported easily. After a call is made the connection is established to the router and the router transmits the packets to the destination number.

**1.4 EXISTING SYSTEM**

Voice over Internet Protocol (VoIP) provides the ideas for connecting two clients through voice over the internet. The advent of Voice over Internet Protocol(VoIP) had fundamentally been transforming the way telecommunication evolves.

Driven by the ongoing deployment of broadband infrastructure and the increasing demand of telecommunication service, VoIP technologies and applications have led to the development of economical IP phone equipment based on embedded systems. IP phone application can satisfyingly provide the necessary interfaces between telephony signals and IP networks. Although IP phone communication over the data networks such as LAN exists but these IP phones are fixed type. We implement wireless IP phone communication using the Wi-Fi network, VoIP phones call without the use of a computer; instead they connect directly to the IP network (using technologies such as Wi-Fi of Ethernet).

**1.5 PROPOSED SYSTEM**

We implement wireless IP phone communication using the Wi-Fi network, VOIP phones call without the use of a computer; instead they connect directly to the IP network. Our proposal allows free calls within the network with high quality voice transmission. Our Proposed model allows ANDROID phones to communicate with other in the same Wi-Fi enabled Region. This model will be a prototype of inherent devices communicating through the Wi-Fi bandwidth and will greatly reduce the communication cost in large organisation.

Search Wi-Fi

Connect with AP

Create AP

Exit

Make a Call

Transmit to Destination

Fig no.1.1 BLOCK DIAGRAM

**1.6 OBJECTIVES**

On the Wi-Fi route IP phones registers its fixed IP, where the router will update its routing table with this IP phone being active. IP addressing and sub-netting is available in the same way as in any computer system. Each phone is identified by a user name. So the routing table updates its IP with a corresponding user name. Calls can be made to any user in the routing table identified by user name and this information is made available to all users logged into the network.

**1.7 SCOPE**

Scope of this project is broad in terms of making calls without using SIM card.

Few of them are:-

* This can be used in educational institution as well as incorporate world.
* Can be used anywhere and anytime in the Wi-Fi environment.

**1.8 VOIP TECHNOLOGY OVERVIEW**

VoIP refers to the movement of voice traffic over internet protocol IP Based network. To permit the traffic over the computer network, the analog signals are turned into digital packets. The digital packets have a destination address but they follow no fixed path. To enable VoIP, broadband access, computer, and software. Additional hardware such as server, switches, router, and other may be required depending on the volume and nature if traffic. VoIP permits the integration of data, voice into one communication channel.

The term digital convergence refers to this phenomenon of multiple media delivered over a single network. Some of the application and service include PC based distance learning solutions, video conferencing and team management software. Early providers of voice over IP services offered business models (and technical solutions) that mirrored the architecture of the legacy telephone network.

**1.9 ANDROID OVERVIEW**

Android is software for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tool and APIs necessary to begin developing application on the android platform using the java programming language.

Android will ship with a set of core application including an email client, SMS program, calendar, maps, browser, contact, and other. All applications are written using the java programming language.

Android include a set of C/C++ libraries used by the various component of the Android System. These capabilities are exposed to developers through the android application framework. Some of the core libraries are listed below:

* **System C Library** - a BSD-derived implementation of the standard C system libraries (libc), tuned for embedded Linux-based devices.
* **Media Libraries** – based on the packet video Open CORE, the libraries support playback and recording of many popular audio and video formats, as well as static images files, including MPEG4, H.264, MP3,AAC,AMR,JPG and PNG.
* **Surface Manager** – manages to access the display the subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications.

**CHAPTER 2**

**SYSTEM ANALYSIS**

**2.1 INTRODUCTION**

This project is mainly used for making a call within the Wi-Fi range instead of using service provider using Wi-Fi range. If the project arrived in large organisation the communication cost is nil.

**2.2 OVERALL DESCRIPTION**

In the voice calls over Wi-Fi, the requirements are mobile phones that are

Wi-Fi enabled and wireless router. Wireless router is used to send the data to particular user. Router maintaining the routing table. And a final requirement is Wi-Fi connection

**2.2.1** **Product Descriptive**

Voice Calls over Wi-Fi involves making a free call within the Wi-Fi range.

The sender and receiver should have a Wi-Fi mobile. The mobile phones should be

connected to Wi-Fi range. Both the users will be provided with a Username. Senders make a call and receiver attending the call within the Wi-Fi range.

**2.2.2 Product Functions**

Our approach adopts wireless communication, doesn’t required wired. Our Approach used mobile instead of using pc. We are giving mobility for making a call. In previous approach we make a call but fixed type. The advantage of our method is that, using mobiles for making a call and wireless communication.

**2.2.3 Product Perspective**

The proposed project is **INSTANT VOICE MESSAGES OVER WIFI**

The project includes the following stages.

* Status Field
* Make a Call
* Conversation
* End a Call

**STATUS FIELD**

There is a status field which determines what the current activity taking place in the application.

**MAKE A CALL**

When the user enters into application it displays search/create button to create AP or to Search the Wi-Fi devices and get connected automatically. The user can transmit the voice by selecting call button.

**CONVERSATION**

The user can interact with each other with no cost and time limits. The quality depends upon the Wi-Fi signal strength.

**DISCONNECT THE CALL**

The user can disconnect the call by selecting the end call option available in the application. The user can call again or exit the application.

**2.2.4 Product Features**

The features that are available to the user are:

* Can create AP.
* Can search for Wi-Fi AP.
* Can add Username.
* Can send voice within Wi-Fi region.

**2.2.5OPERATING ENVIRONMENT**

**2.2.5.1 Software Environment:**

1. Microsoft os
2. JDK 1.6.2**3**
3. Eclipse
4. Android SDK 2.3.4

**2.2.5.2 Hardware Environment:**

1. Wi-Fi enabled Android Mobile Phone
2. Android Mobile phone with Hotspot

**2.2.6 CONSTRAINTS**

**2.2.6.1 Constraints in Analysis**

1. Constraints as normal voice
2. Constraints as Operational Restrictions
3. Constraints Integrated in Existing Model Concepts
4. Constraints as a Separate Concept
5. Constraints Implied by the Model Structure

**2.2.6.2 Constraints in Design**

1. Determination of the Involved Classes
2. Determination of the Involved Objects
3. Determination of the Involved Actions
4. Determination of the Require Clauses
5. Global actions and Constraint Realization

**2.2.6.3 Constraints in Implementation**

A hierarchical structuring of relations may result in more classes and a more complicated structure to implement. Therefore it is advisable to transform the hierarchical relation structure to a simpler structure such as a classical flat one. It is rather straightforward to transform the developed hierarchical model into a bipartite, flat model, consisting of classes on the one hand and flat relations on the other. Flat relations are preferred at the design level for reasons of simplicity and implementation ease. There is no identity or functionality associated with a flat relation. A flat relation corresponds with the relation concept of entity-relationship modeling and many object oriented methods.

**2. 3 SPECIFIC REQUIREMENTS**

It mainly analyzes the user requirements and gives criteria to acquire the requirement.

**2.3.1 EXTERNAL INTERFACE REQUIREMENTS**

## 2.3.1.1. User Interfaces

1. All the contents in the project are implemented using Graphical User

Interface (GUI) – Android SDK.

2. Every conceptual part of the projects is reflected using the Android

SDK.

3. System gets the input and delivers through the Android’s GUI based.

**2.3.1.2. Software Interfaces**

Using xml we have created front end tool design and java for logical operation.

**2.3.2 DATA FLOW DIAGRAM**

Search Wi-Fi

From phone

Create Access Point

If Available

Waiting For Connection

Get Connected

Make Call

End Call

Fig 2.2 Data Flow Diagram

**2.3.3** **PERFORMANCE REQUIREMENTS**

The application may be safety critical, if so, there are issues associated with its integrity level. The application may not be safety critical although it forms part

of safety critical system. In our project the performance can be improved by giving the Wi-Fi signal strength. The routers also change the performance of the project. The system performance and system configuration also increase the performance of the project.

**2.3.4 SOFTWARE QUALITY ATTRIBUTES**

**Functionality**:

The required functions available, including interoperability and security.

**Reliability:**

It is the ability of the system to recover from the failure after detection. The project is prone to fault tolerance and recoverability.

**Reusability**

It is the ability of the system to be reused. This project can be reused in various domains.

**Usability:**

It is effectiveness of the use of system. The user interface of this project is developed in such a way that the user can understand easily, learn at a faster rate and operate with great level of comfort.

**Efficiency:**

The project is having high performance and resource behaviour.

**CHAPTER 3**

**DETAILED DESIGN**

**3.1 DECOMPOSISTION DESCRIPTION**

There are various components and modules present in our project. Everything in the project can’t be done at the same time; hence we had decomposed or project in to some of the modules. Deposition gives us an easier way to solve any type of big problems. It has a power to convert very huge things in to number of smaller ones. Therefore the work stress will be reduced and much possibility to understand the project very easily.

**3.1.1 MODULE DECOMPOSITION**

Module is part of the project. Module decomposition is that dividing the work into smaller parts such a Way that to that to make our work easier to understand and face the problem that occurs while handling the project work. So based up on the project the module description and decomposition had been done.

**3.2 DEPENDENCY DESCRIPTION**

**3.2.1 INTER MODULE DEPENDENCY**

In our project the first module consists of user interface design and the process of getting the input from the user. The second module of our project performs various operations such as encryption such as encryption and transferring the information. The third module includes the decryption the information and gave to the particular user.

**3.2.2 SYSTEM ARCHITECTURE**

The architecture diagram explains the features and process involved in the system.

****

Fig 2.1: System Architecture Diagram

In the architecture of our project include mobiles and router.user1 and user2 are connected to the router. Normal mobiles are not used. Wi-Fi mobiles are used for making a connection with Wi-Fi. Mobile phones and router are connected to Wi-Fi.user1 or user2 anyone intimate the call, router send the data to the particular receiver. The range of Wi-Fi within the particular area. If the user goes out of the Wi-Fi range then calls get disconnected.

**3.3 DETAILED DESIGN**

In our project the modules had been designed based up on its work. The design of our project is that to build a new architecture for the voice call.

**Module 1: Wi-Fi Search & Creating AP**

When a user starts the application, he must be able to create AP or Search for Wi-Fi.

**Module 2: Programming**

The programming module uses xml for layout designs and java for programming language for functionalities.

**Module 3: Username**

A menu button, when touched displays Settings to Add Username.

**Module 4: Record & Playback**

When the user click Send Voice button on the phone he can talk to other phone. We again exchange a series of Voice messages to complete this task. RTP Messages will be used for transferring the actual voice packets.

**Module 5: Exit**

To close the Wificall application, the user should select the exit option from the main layout. The Wificall program is closed and the user can run any other applications he so wishes to.

**3.4 OOAD DIAGRAMS**

The typical object oriented analysis and design diagrams for the project description can be shown according to the work we have done on the project. The OOAD diagram consists of the following categories;

**3.4.1 Use-case Diagram**

A use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, a communication association between the actors and the use cases, and a generalization among use cases.

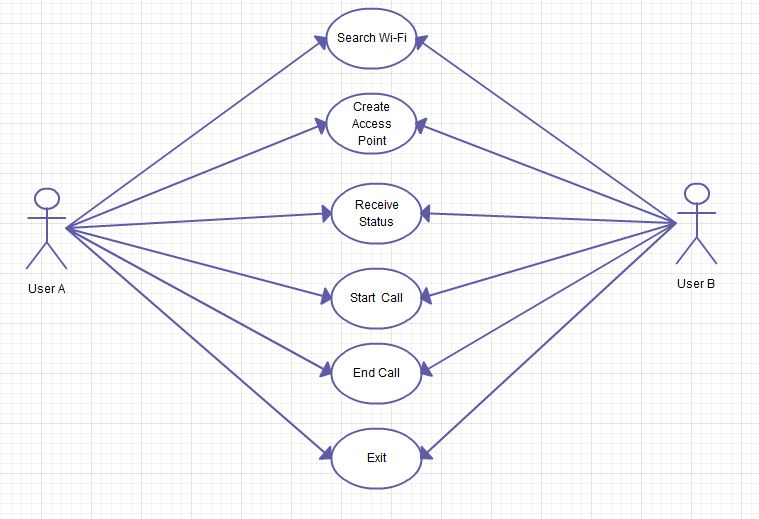


Fig 3.1: Use-Case Diagram

Figure 3.1 represents use case diagram. The actors in the use case diagram are the User. A User first search for the Wi-Fi else create the AP and wait for the other user to get connected. Then user can make call to that particular user and conversation between the two users take place, and user can end the call if conversation is over.

**3.4.2 Activity Diagram**

An activity diagram is a variation or special case of a state machine , in which the states are the activities representing the performance of operations and the transition are triggered by the completion of operation.

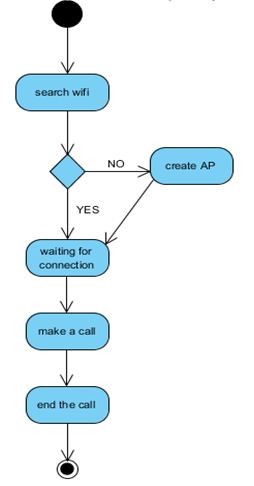
****

Fig 3.2: Activity Diagram

Figure 3.2 represents the activity diagram. It gives the step by step explanation of the activities. Once a user searches the Wi-Fi it is checked for its AP name and its password and if it matches only it allows to send voice to the other mobile which is connected. The user is allowed to make call with in Wi-Fi range.

**3.4.3 Sequence Diagram**

A sequence diagram describes the behavior of the system by viewing the interaction between the system and environment.

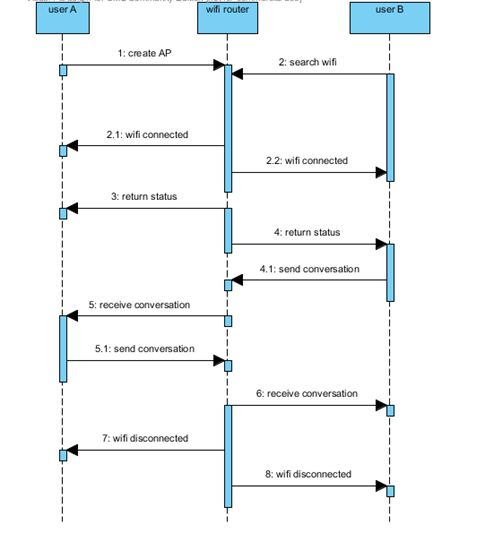


Fig 3.3: Sequence Diagram

Figure3.3 represents the sequence diagram. It shows the sequence of events occurred. Here the User A creates the AP that acts as a Wi-Fi router. If the User B searches for Wi-Fi the AP that is created by User A is used and thereby the request is accepted by the AP and conversation takes place.

**3.5 TESTING**

Testing is a process of executing a program with intent of finding an error. A good test case is one that has high probability an as-yet-undiscovered error. A successful test is one that uncovers an as-yet-discovered error.

**Table 3.1:Test plan for username**

|  |  |  |
| --- | --- | --- |
| S.NO | INPUT | EXPECTED OUTPUT |
| 1 | Adding username for mobile | The username is added successfully. |
| 2 | Diplaying the username on mobile to which it is connected. | the user name were displayed for identification purposes to know who is connected with. |

**Table 3.2:Test plan for main screen**

|  |  |  |
| --- | --- | --- |
| S.NO | INPUT | EXPECTED OUTPUT |
| 2 | Searching for Wi-Fi Access Point | The list Wi-Fi access Point must be displayed. |
| 3 | Creating the New Access Point | It should create the new access point. |
| 5 | Sending Voice messages to receiver | The voice messages are transmitted to receiver. |
| 6 | Listening to voice messages | The voice messages which are transmitted are received and played in receiver. |
| 8 | Exit | It allows to close the application |

**CHAPTER 4**

**IMPLEMENTATION PROCEDURE**

**4.1 IMPLEMENTATION**

We propose a approach to making a free call in ANDROID mobiles over Wi-Fi. Unlicensed Mobile Access (UMA) is a 3rd Generation Partnership Program global specification that provides a standard for service providers to merge mobile networks and wireless LANs into a single seamless access network with one mobile device, one user interface, and a common set of network services for both voice and data.

When making a call to a phone within the network, the users within the Wi-Fi range are displayed and the required username is selected. The application then sends Id in 128bit encrypted form to the router, requesting a call to be placed.

The ANDROID application at the destination intimates the user of the incoming call. If the call is accepted, the router changes both phones status with others username.

The result of the voice calls over Wi-Fi application using Wi-Fi enabled mobile phone devices mainly requires ANDROID Wi-Fi enabled mobile phone and router. Both users must install the application in their mobile phones. A user specifies the username for the first time when he uses the application further calls can be made by selecting send voice. Using these application free calls can be made since it functions within the range of Wi-Fi. The **ANDROID** application at destination intimates the user of the incoming call; if the call is accepted, the router changes both phones to make transmission of voice messages. ANDROID provides a robust, flexible environment for applications running on mobile. It includes flexible user interfaces, robust security, built-in network protocols and support for networked and offline applications that can be downloaded dynamically.

**INSTALLING WIFICALL APPLICATION**

Step 1: Start.

Step 2: Install the application on android mobile.

Step 3: Click to open the application.

Step 4: Provide the user name by selecting menu.

Step 5: Once the username is entered click ok.

**MAKING A CALL**

Step 1: Start

Step 2: click on Search/Create button.

Step 3: This will search for the Wi-Fi AP if not found then it creates its own AP.

Step 4: Then in another mobile open the same application and select Search/Create button and now the Wi-Fi AP will be available and get the username of that mobile.

Step 5: Once the username are sent then select send voice button and start speaking this will record those message and send to the destination instantly.

Step 6: Then select Stop Sending voice button to stop.

**CHAPTER 5**

**CONCLUSION AND FUTURE WORK**

**5.1 CONCLUSION**

Communication is a field where it is highly possible for the data to get corrupted, and its high imperative to secure the data to be transmitted. Our proposal provides an efficient mechanism to send voice calls over the Wi-Fi bandwidth by using encryption and decryption mechanism that render secured packet reception. This project provides a cheap, effective and secure means of communication within a specified network. The cost involved is only the initial setup cost and all calls within the network are free. This model will be very useful to solve the communication problems in large organizations, by making free voice calls through Wi-Fi.

**5.2 FUTURE WORK**

Our application involves sending instant voice messages through Wi-Fi without the need of SIM card. Our future work includes connecting to a database so that several user can communicate with each other simultaneously and also the communication can be made as full duplex and the quality of voice can be improved.

**APPENDIX-A**

**SAMPLE CODING**

**Layout Design**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:orientation="vertical" >

<EditText

android:id="@+id/editText1"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:singleLine="true"

android:editable="false"

android:text="Idle." />

<EditText

android:id="@+id/editText2"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:singleLine="true"

android:editable="false"

android:text="Connected with: ---" />

<Button

android:id="@+id/button1"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="Search/Create" />

<Button

android:id="@+id/button2"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="Call"

android:enabled="false" />

<Button

android:id="@+id/button3"

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="top"

android:text="Exit" />

</LinearLayout>

**Main activity**

package com.vvv.wificall.activity;

import com.vvv.wificall.activity.R;

import com.vvv.wificall.activity.task.StartNetworkTask;

import com.vvv.wificall.event.CallbackEvent;

import android.app.Activity;

import android.content.Context;

import android.content.Intent;

import android.os.Bundle;

import android.os.PowerManager;

import android.os.PowerManager.WakeLock;

import android.view.Menu;

import android.view.MenuItem;

import android.view.View;

import android.view.View.OnClickListener;

import android.widget.Button;

import android.widget.EditText;

public class MainActivity extends Activity

{

private CallbackEvent startNetworkCallback = null;

private StartNetworkTask startNetworkTask = null;

private WakeLock wakeLock = null;

public static EditText StatusField = null;

public static EditText NameField = null;

public static Button TransmitButton = null;

public static Button ConnectButton = null;

public static Button ExitButton = null;

@Override

public void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

PowerManager powerManager = (PowerManager)getSystemService(Context.POWER\_SERVICE);

wakeLock = powerManager.newWakeLock(PowerManager.FULL\_WAKE\_LOCK, "DoNotDimScreen");

StatusField = (EditText)findViewById(R.id.editText1);

NameField = (EditText)findViewById(R.id.editText2);

TransmitButton = (Button)findViewById(R.id.button2);

TransmitButton.setTag(false);

TransmitButton.setOnClickListener(new OnClickListener()

{

public void onClick(View arg0)

{

TransmitButton.setTag(!(Boolean)TransmitButton.getTag());

if ((Boolean)TransmitButton.getTag())

TransmitButton.setText("End Call");

else

TransmitButton.setText("Call");

}

});

ConnectButton = (Button)findViewById(R.id.button1);

ConnectButton.setTag(false);

ConnectButton.setOnClickListener(new OnClickListener()

{

public void onClick(View arg0)

{

if ((Boolean)ConnectButton.getTag())

{

startNetworkTask.stopNotifier.stop();

ConnectButton.setText("Disconnecting...");

ConnectButton.setEnabled(false);

TransmitButton.setEnabled(false);

if ((Boolean)TransmitButton.getTag())

TransmitButton.performClick();

}

else

{

startNetworkTask = new StartNetworkTask(MainActivity.this, startNetworkCallback);

startNetworkTask.execute(null, null);

ConnectButton.setText("Cancel");

ConnectButton.setTag(true);

}

}

});

ExitButton = (Button)findViewById(R.id.button3);

ExitButton.setOnClickListener(new OnClickListener()

{

public void onClick(View arg0)

{

System.exit(0);

}

});

startNetworkCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

TransmitButton.setEnabled(false);

ConnectButton.setTag(false);

ConnectButton.setText("Search/Create");

ConnectButton.setEnabled(true);

}

};

}

@Override

protected void onPause()

{

super.onPause();

wakeLock.release();

}

@Override

protected void onResume()

{

super.onResume();

wakeLock.acquire();

}

@Override

public boolean onCreateOptionsMenu(Menu menu)

{

menu.add(0, 0, 0, "Preferences");

return true;

}

@Override

public boolean onOptionsItemSelected(MenuItem item)

{

switch (item.getItemId())

{

case 0:

{

Intent intent = new Intent(MainActivity.this, PreferencesActivity.class);

startActivity(intent);

return true;

}

}

return false;

}

}

**Start network Activity**

package com.vvv.wificall.activity.task;

import com.vvv.wificall.activity.MainActivity;

import com.vvv.wificall.event.CallbackEvent;

import com.vvv.wificall.event.ObjectHolder;

import com.vvv.wificall.event.StopEvent;

import com.vvv.wificall.wifi.NetworkAdapter;

import com.vvv.wificall.wifi.SessionAdapter;

import com.vvv.wificall.wifi.TransmissionAdapter;

import com.vvv.wificall.wifi.SessionAdapter.SessionMessage;

import android.content.Context;

import android.media.AudioFormat;

import android.media.AudioManager;

import android.media.AudioRecord;

import android.media.AudioTrack;

import android.media.MediaRecorder.AudioSource;

import android.os.AsyncTask;

public class StartNetworkTask extends AsyncTask<Object, Object, Object>

{

private final static int SendPacketTimeout = 5000;

private final static int ReceivePacketTimeout = 500;

private final static int TransactionTimeout = 10000;

private final static int FastTimeSpan = 100;

public StopEvent stopNotifier = null;

private Context context = null;

private CallbackEvent callback = null;

private CallbackEvent statusCallback = null;

private CallbackEvent runtimeCallback = null;

private CallbackEvent interfaceCallback = null;

private CallbackEvent timeoutCallback = null;

private AudioTrack voice = null;

public StartNetworkTask(Context context, CallbackEvent callback)

{

super();

currently non-return for an application

this.context = context;

this.callback = callback;

stopNotifier = new StopEvent();

statusCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

publishProgress(new Object[] { 0, arg });

}

};

runtimeCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

publishProgress(new Object[] { 1, arg });

}

};

interfaceCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

publishProgress(new Object[] { 2, arg });

}

};

timeoutCallback = new CallbackEvent()

{

public void onCallback(Object arg)

{

publishProgress(new Object[] { 3, arg });

}

};

try

{

voice = new AudioTrack(AudioManager.STREAM\_MUSIC,

8000,

AudioFormat.CHANNEL\_CONFIGURATION\_MONO,

AudioFormat.ENCODING\_PCM\_16BIT,

TransmissionAdapter.MaxPacketSize - 2,

AudioTrack.MODE\_STREAM);

voice.setPlaybackRate(8000);

voice.play();

}

catch (Exception e)

{

e.printStackTrace();

}

}

@Override

protected void onPreExecute()

{

super.onPreExecute();

}

private String ByteArrayToString(byte[] bytes)

{

String result = "";

for (int i = 0; ((i < bytes.length) && (bytes[i] != 0)); i ++)

result += (char)bytes[i];

return result;

}

private void OnConnected()

{

byte[] inputBuffer = new byte[TransmissionAdapter.MaxPacketSize];

byte[] payloadBuffer = new byte[TransmissionAdapter.MaxPacketSize - 2];

statusCallback.onCallback("Waiting...");

int timeout = -1;

while (!stopNotifier.isStopped())

{

if ((Boolean)MainActivity.TransmitButton.getTag())

{

if (TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_TRANSMISSION\_BEGIN,

null),

stopNotifier, SendPacketTimeout, FastTimeSpan)

)

{

statusCallback.onCallback("Talking...");

AudioRecord record = null;

try

{

record = new AudioRecord(AudioSource.MIC,

8000,

AudioFormat.CHANNEL\_CONFIGURATION\_MONO,

AudioFormat.ENCODING\_PCM\_16BIT,

TransmissionAdapter.MaxPacketSize - 2);

record.startRecording();

}

catch (Exception e)

{

e.printStackTrace();

}

timeout = TransactionTimeout;

while (!stopNotifier.isStopped() && (Boolean)MainActivity.TransmitButton.getTag())

{

if (record != null)

record.read(payloadBuffer, 0, TransmissionAdapter.MaxPacketSize - 2);

if (TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_TRANSMISSION\_PAYLOAD,

payloadBuffer),

stopNotifier, SendPacketTimeout, FastTimeSpan))

timeout = TransactionTimeout;

else

{

timeout -= SendPacketTimeout;

if (timeout < 0)

{

timeout = -1;

timeoutCallback.onCallback(null);

break;

}

}

}

record.stop();

statusCallback.onCallback("Waiting...");

TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_TRANSMISSION\_END,

null),

null, SendPacketTimeout, FastTimeSpan);

}

}

else

{

if (TransmissionAdapter.ReceivePackets(inputBuffer, null, stopNotifier, ReceivePacketTimeout, FastTimeSpan))

{

switch (SessionAdapter.PacketDispatcher(inputBuffer, payloadBuffer))

{

case MSG\_TRANSMISSION\_BEGIN:

{

timeout = TransactionTimeout;

statusCallback.onCallback("Listening...");

interfaceCallback.onCallback(false);

break;

}

case MSG\_TRANSMISSION\_PAYLOAD:

{

timeout = TransactionTimeout;

voice.write(payloadBuffer, 0, payloadBuffer.length);

break;

}

case MSG\_TRANSMISSION\_END:

{

timeout = -1;

statusCallback.onCallback("Waiting...");

interfaceCallback.onCallback(true);

break;

}

case MSG\_DISCONNECT:

{

return;

}

}

}

if (timeout != -1)

{

if (timeout >= 0)

timeout -= ReceivePacketTimeout;

if (timeout < 0)

{

timeout = -1;

statusCallback.onCallback("Waiting...");

interfaceCallback.onCallback(true);

}

}

}

}

timeout = TransactionTimeout;

while (!TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_DISCONNECT,

null),

null, SendPacketTimeout, FastTimeSpan))

{

timeout -= SendPacketTimeout;

if (timeout < 0)

break;

}

}

@Override

protected Object doInBackground(Object... params)

{

byte[] inputBuffer = new byte[TransmissionAdapter.MaxPacketSize];

byte[] payloadBuffer = new byte[TransmissionAdapter.MaxPacketSize - 2];

if (NetworkAdapter.BeginNetworkScan(context, statusCallback, stopNotifier))

{

if (NetworkAdapter.BeginNetworkConnection(context, statusCallback, stopNotifier))

{

NetworkAdapter.SetOtherClientIP(NetworkAdapter.GetServerIP(context));

statusCallback.onCallback("Sending authorization request...");

if (TransmissionAdapter.SendPackets(

SessionAdapter.PacketGenerator(

SessionMessage.MSG\_CONNECTION\_REQUEST,

NetworkAdapter.GetUserName(context).getBytes()),

stopNotifier, -1, -1)

)

{

statusCallback.onCallback("Receiving autorization response...");

if (TransmissionAdapter.ReceivePackets(inputBuffer, null, stopNotifier, -1, -1))

{

switch (SessionAdapter.PacketDispatcher(inputBuffer, payloadBuffer))

{

case MSG\_CONNECTION\_SUCCESS:

{

NetworkAdapter.SetOtherName(ByteArrayToString(payloadBuffer));

statusCallback.onCallback("Outcoming connection established.");

runtimeCallback.onCallback(true);

OnConnected();

runtimeCallback.onCallback(false);

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

NetworkAdapter.SetOtherName("---");

statusCallback.onCallback("Conection closed.");

break;

}

case MSG\_CONNECTION\_FAILURE:

{

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

statusCallback.onCallback("Authorization rejected by server.");

break;

}

default:

{

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

statusCallback.onCallback("Authorization failed.");

break;

}

}

}

else

{

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

statusCallback.onCallback("Server is not responding.");

}

}

else

{

NetworkAdapter.StopWifi(context, statusCallback, null);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

statusCallback.onCallback("Network unreachable.");

}

}

else

{

statusCallback.onCallback("Connection failed.");

}

}

else if (!stopNotifier.isStopped())

{

if (NetworkAdapter.StartAccessPoint(context, statusCallback, stopNotifier))

{

boolean isUserConnected = false;

ObjectHolder<String> senderIPHolder = new ObjectHolder<String>();

while (!stopNotifier.isStopped() && !isUserConnected)

{

statusCallback.onCallback("Waiting for connections...");

if (TransmissionAdapter.ReceivePackets(inputBuffer, senderIPHolder, stopNotifier, -1, -1))

{

switch (SessionAdapter.PacketDispatcher(inputBuffer, payloadBuffer))

{

case MSG\_CONNECTION\_REQUEST:

OnConnected();

runtimeCallback.onCallback(false);

NetworkAdapter.SetOtherClientIP("0.0.0.0");

NetworkAdapter.SetOtherName("---");

}

NetworkAdapter.StopAccessPoint(context, statusCallback, null);

statusCallback.onCallback("Conection closed.");

}

else

{

statusCallback.onCallback("Access Point not created.");

}

}

if (stopNotifier.isStopped())

statusCallback.onCallback("Idle.");

return 0;

}

@Override

protected void onProgressUpdate(Object... progress)

{

super.onProgressUpdate(progress);

switch ((Integer)progress[0])

{

case 0:

{

MainActivity.StatusField.setText((String)progress[1]);

break;

}

case 1:

{

if ((Boolean)progress[1])

{

MainActivity.NameField.setText("Connected with: " + NetworkAdapter.GetOtherName());

MainActivity.TransmitButton.setEnabled(true);

}

else

{

MainActivity.NameField.setText("Connected with: ---");

MainActivity.TransmitButton.setEnabled(false);

}

break;

}

case 2:

{

MainActivity.TransmitButton.setEnabled((Boolean)progress[1]);

MainActivity.ConnectButton.setEnabled((Boolean)progress[1]);

break;

}

case 3:

{

MainActivity.TransmitButton.performClick();

break;

}

}

}

@Override

protected void onPostExecute(Object result)

{

super.onPostExecute(result);

try

{

if (voice != null)

voice.stop();

}

catch (Exception e)

{

e.printStackTrace();

}

callback.onCallback(false);

}

}

**NetworkAdapter**

package com.vvv.wificall.wifi;

import java.lang.reflect.Method;

import com.vvv.wificall.event.CallbackEvent;

import com.vvv.wificall.event.ConditionalEvent;

import com.vvv.wificall.event.StopEvent;

import android.content.Context;

import android.net.ConnectivityManager;

import android.net.wifi.ScanResult;

import android.net.wifi.WifiConfiguration;

import android.net.wifi.WifiManager;

import android.preference.PreferenceManager;

import android.text.format.Formatter;

import android.util.Log;

public class NetworkAdapter

{

private static String OtherName = "---";

static String OtherClientIP = "0.0.0.0";

final static String NetworkSSID = "wificall";

final static String NetworkKey = "wificall";

final static int StartWifiTimeout = 10000;

final static int StopWifiTimeout = 10000;

final static int StartApTimeout = 10000;

final static int StopApTimeout = 10000;

final static int BeginScanTimeout = 5000;

final static int BeginConnectionTimeout = 30000;

final static int WaitTimeSpan = 250;

static int ActionCounter = 0;

private static boolean ConditionalWait(int timeout, CallbackEvent statusCallback, StopEvent stopNotifier, ConditionalEvent event)

{

int ActionId = ActionCounter ++;

try

{

Log.d("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ precondition ]");

if (event.checkCondition())

return true;

Log.d("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ start ]");

if (!event.startEvent())

return false;

statusCallback.onCallback(event.getOperationName());

boolean status = false;

timeout /= WaitTimeSpan;

if (stopNotifier != null && stopNotifier.isStopped())

{

Log.i("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ cancelled ]");

event.onTimeout();

return false;

}

if (status)

{

Log.i("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ success ]");

return true;

}

else

{

Log.w("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ timeout ]");

event.onTimeout();

return false;

}

}

catch (Exception e)

{

Log.e("Wifi action", "Action # " + ActionId + ": " + event.getOperationName() + " [ exception ]");

try

{

event.onTimeout();

}

catch (Exception e2)

{

e2.printStackTrace();

}

e.printStackTrace();

return false;

}

}

public static boolean StartWifi(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

final Context contextHandler = context;

final CallbackEvent statusCallbackHandler = statusCallback;

final StopEvent stopNotifierHandler = stopNotifier;

return ConditionalWait(StartWifiTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public String getOperationName()

{

return "Turning WiFi on...";

}

});

}

public static boolean StopWifi(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

return ConditionalWait(StopWifiTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

return wifiManager.setWifiEnabled(false);

}

public boolean checkCondition() throws Exception

{

return (wifiManager.getWifiState() == WifiManager.WIFI\_STATE\_DISABLED);

}

public void onTimeout() throws Exception

{

}

public String getOperationName()

{

return "Turning WiFi off...";

}

});

}

public static boolean StartAccessPoint(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

final Context contextHandler = context;

final CallbackEvent statusCallbackHandler = statusCallback;

final StopEvent stopNotifierHandler = stopNotifier;

return ConditionalWait(StartApTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

if (!StopWifi(contextHandler, statusCallbackHandler, stopNotifierHandler))

return false;

Method SetWifiApEnabled = wifiManager.getClass().getMethod("setWifiApEnabled", WifiConfiguration.class, boolean.class);

WifiConfiguration netConfig = new WifiConfiguration();

netConfig.SSID = NetworkSSID;

netConfig.preSharedKey = NetworkKey;

netConfig.allowedAuthAlgorithms.set(WifiConfiguration.AuthAlgorithm.SHARED);

netConfig.allowedProtocols.set(WifiConfiguration.Protocol.RSN);

netConfig.allowedProtocols.set(WifiConfiguration.Protocol.WPA);

netConfig.allowedKeyManagement.set(WifiConfiguration.KeyMgmt.WPA\_PSK);

netConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.CCMP);

netConfig.allowedPairwiseCiphers.set(WifiConfiguration.PairwiseCipher.TKIP);

netConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.CCMP);

netConfig.allowedGroupCiphers.set(WifiConfiguration.GroupCipher.TKIP);

return (Boolean)SetWifiApEnabled.invoke(wifiManager, netConfig, true);

}

public boolean checkCondition() throws Exception

{

Method IsWifiApEnabled = wifiManager.getClass().getMethod("isWifiApEnabled");

return ((Boolean)IsWifiApEnabled.invoke(wifiManager));

}

public void onTimeout() throws Exception

{

StopAccessPoint(contextHandler, statusCallbackHandler, null);

}

public String getOperationName()

{

return "Turning AP on...";

}

});

}

public static boolean StopAccessPoint(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

return ConditionalWait(StopApTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

Method SetWifiApEnabled = wifiManager.getClass().getMethod("setWifiApEnabled", WifiConfiguration.class, boolean.class);

return (Boolean)SetWifiApEnabled.invoke(wifiManager, null, false);

}

public boolean checkCondition() throws Exception

{

Method IsWifiApEnabled = wifiManager.getClass().getMethod("isWifiApEnabled");

return (!(Boolean)IsWifiApEnabled.invoke(wifiManager));

}

public String getOperationName()

{

return "Turning AP off...";

}

});

}

public static boolean BeginNetworkScan(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

final Context contextHandler = context;

final CallbackEvent statusCallbackHandler = statusCallback;

final StopEvent stopNotifierHandler = stopNotifier;

public void onTimeout() throws Exception

{

StopWifi(contextHandler, statusCallbackHandler, null);

}

public String getOperationName()

{

return "Scanning for network...";

}

});

}

public static boolean BeginNetworkConnection(Context context, CallbackEvent statusCallback, StopEvent stopNotifier)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

final ConnectivityManager connectivityManager = (ConnectivityManager)context.getSystemService(Context.CONNECTIVITY\_SERVICE);

final Context contextHandler = context;

final CallbackEvent statusCallbackHandler = statusCallback;

final StopEvent stopNotifierHandler = stopNotifier;

return ConditionalWait(BeginConnectionTimeout, statusCallback, stopNotifier, new ConditionalEvent()

{

public boolean startEvent() throws Exception

{

if (!BeginNetworkScan(contextHandler, statusCallbackHandler, stopNotifierHandler))

return false;

if (wifiManager.getConfiguredNetworks() != null)

{

for (WifiConfiguration Network : wifiManager.getConfiguredNetworks())

{

if (Network.SSID.equals(NetworkSSID) || Network.SSID.equals('"' + NetworkSSID + '"'))

{

wifiManager.removeNetwork(Network.networkId);

break;

}

}

}

WifiConfiguration netConfig = new WifiConfiguration();

netConfig.SSID = '"' + NetworkSSID + '"';

netConfig.preSharedKey = '"' + NetworkKey + '"';

int networkId = wifiManager.addNetwork(netConfig);

if (networkId == -1)

return false;

public static void SetOtherClientIP(String address)

{

OtherClientIP = address;

}

public static String GetUserName(Context context)

{

if (PreferenceManager.getDefaultSharedPreferences(context) == null)

return "User";

PreferenceManager.getDefaultSharedPreferences(context).getString("nickNamePref", "User");

}

public static String GetOtherName()

{

return OtherName;

}

public static void SetOtherName(String name)

{

OtherName = name;

}

public static String GetServerIP(Context context)

{

final WifiManager wifiManager = (WifiManager)context.getSystemService(Context.WIFI\_SERVICE);

if (wifiManager.getDhcpInfo() == null)

return "0.0.0.0";

return Formatter.formatIpAddress(wifiManager.getDhcpInfo().gateway);

}

**APPENDIX-B**

**SNAP SHOTS**

**PREFERENCE SCREEN:**

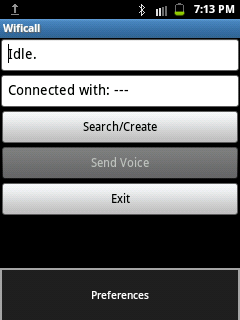
****

Fig no.b.1 Preference Screen

The above snapshot will take to the screen to enter the username for that user.

**USERNAME FOR A:**

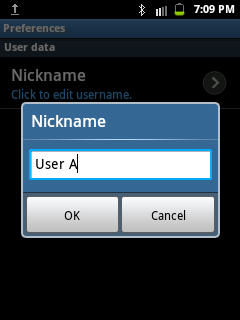
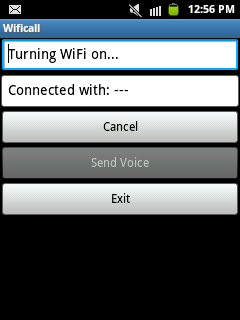
****

Fig no.b.2 Username for A

This screen provide the textbox which allows to enter the username for that user and select ok to confirm.

**SEARCH/CREATE FOR A:**

****

Fif no.b.3 Search/Create for A

This snapshot shows when Search/Create button is clicked make to turn Wi-Fi on and search for the wifi AP.

**TURNING WI-FI OFF:**

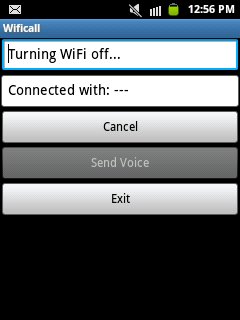
****

Fig no.b.4 Turning Wi-Fi Off

This snapshot displays the status when no AP found after the small amount of time.

**TURNING AP ON:**

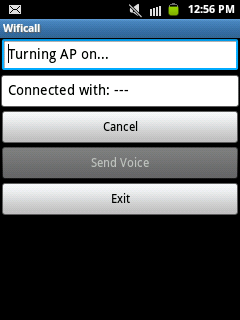
****

Fig no.b.5 Turning AP On

This snapshot displays the status message turning AP on and start the wifi tethering hotspot which acts as AP.

**WAITING FOR CONNECTIONS:**

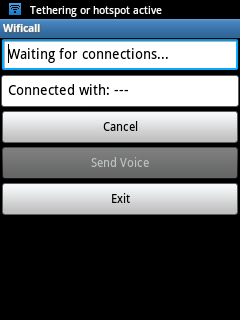
****

Fig no.b.6 Waiting for connection

This snapshot displays the status message waiting for connection that will be available when searched for the Wi-Fi by other mobiles.

**USERNAME FOR B:**

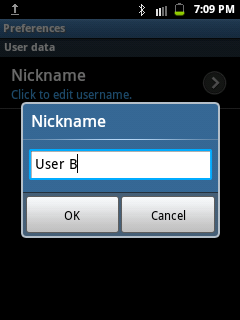
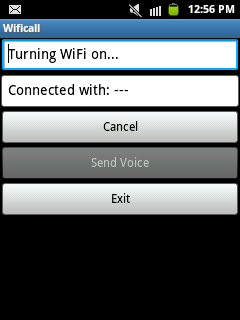
****

Fig no b.7 Username for B

This screen provide the textbox which allows to enter the username for that user and select ok to confirm.

**SEARCH/CREATE FOR B:**

****

Fif no.b.8 Search/Create for B

This snapshot shows when Search/Create button is clicked make to turn Wi-Fi on and search for the wifi AP.

**SEND VOICE BY USER B:**

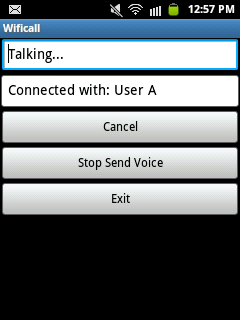
****

Fig no.b.9 Send Voice by User B

This snapshot shows the voice that are being send by User B to User A and stop send voice button get enabled.

**LISTEN VOICE BY USER A:**

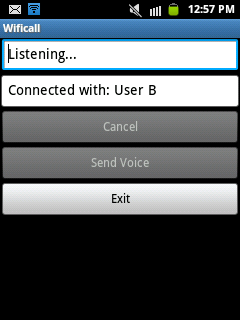
****

Fig no.b.10 Listen Voice By USER A:

This snapshot shows that User A is listening to the User B’s Voice that are being transmitted through Wi-Fi.

**SEND VOICE BY USER A:**

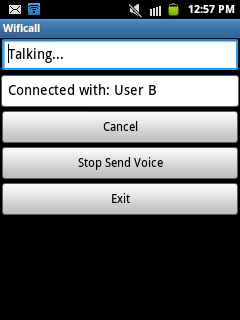
****

Fig no.b.11 Send Voice by User A

This snapshot shows the voice that are being send by User A to User B and stop send voice button get enabled.

**LISTEN VOICE BY USER B:**

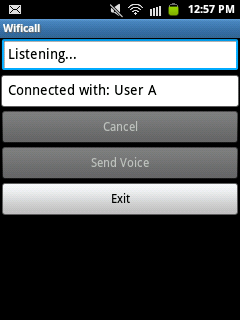
****

Fig no.b.12 Listen Voice By USER A:

This snapshot shows that User B is listening to the User A’s Voice that are being transmitted through Wi-Fi.

**REFERENCES**

1.Implementing 802.11,802.16,802.20 Wireless Networks: Planning, Troubleshooting, and Operations, Ron Olexa

2.Real 802.11 Security: Wi-Fi Protected Access and 802.11i,Jon Edney,William A.Arabaugh

3. Cao Peng and Yang Xuejun, "Performance analysis of SIP-based push-to-talk service for GPRS/cdma2000 network," 2005 2nd International Conference on Mobile Technology, Applications and Systems, 15-17 Nov. 2005, pp. 1-4.

4.Nigel M. May, Mark A. Marick, Bradely R. Schaefer, Mark L. Shaughnessy, and William J. Sugden, "Push-to-Talk call setup for a mobile packet data dispatch network," No. 7231223B2, Jun. 12, 2007

5. Jenq-Muh Hsu; Wei-Bin Lain; Jui-Chih Liang "A Context-Aware Push-to-Talk Service",  Multimedia and Ubiquitous Engineering, 2008.

6. GSM, http://www.gsmworld.com/technologvigsm/index.htill

7. WIFI IEEE 802.11, http://www.wifinotes.com/IEEE-802.11.html.273