Peer to Peer Audio Chat among Mobile Handhelds via Wireless Networking

Liwen Zhang
H00160144

August 2014

Computer Science
School of Mathematical and Computer Sciences

Dissertation submitted as part of the requirements for the award of the degree of MSc in Information Technology (Software Systems)
DECLARATION

I, ........................................................................................................,
confirm that this work submitted for assessment is my own and is
expressed in my own words. Any uses made within it of the works
of other authors in any form (e.g., ideas, equations, figures, text,
tables, programs) are properly acknowledged at any point of their
use. A list of the references employed is included (Heriot-Watt
University 2013).

Signed: ...........................................................................................

Date: .................................................................................................
Abstract

Nowadays communication applications especially IM (Instant Messaging) are growing at a fast rate in the smartphone market and gaining the attention of investors. Recently, there have been several IM app company acquisitions, where high prices were paid. This is evidence that such means of communication is a one in high demand. On the other hand, since smartphones have become widely used in the world, an expectation for communication everywhere and anytime has become established. People have become addicted to communication applications, spend lots of time using such means and are keen to recommend using them to their friends. A smartphone user can talk to anyone he wants so easily with just a small device. However no matter whether users use cellular telephone networks or 3G and 4G data networks, they must pay for the service. This project is aiming at developing a free walkie talkie style audio chat application, that lets users have a conversation peer to peer. When they enjoy using the application, they won’t need to pay anyone or rely on any other devices but their Android phones.

Key words: Wifi p2p;Wifi direct; Audio transfer; Real time chat; Android; Ad hoc network
# Table Of Contents

**Chapter I: Introduction**

1. Overview  

**Chapter II: Literature Review**

1. Overview  
2. Android Platform  
3. Ad Hoc Network  
4. Wifi Direct  
5. Real-time Transport Protocol  
6. Bluetooth  

**Chapter III: Methodology**

1. Overview  
2. Selection  
3. Implementation Methodology  

**Chapter IV: Requirements Analysis**

1. Aim  
2. Objective  
3. Requirements  
4. Use Case  

**Chapter V: Professional, legal, ethical, and social issues**

1. Professional Issues  
2. Legal Issues  
3. Ethical Issues  
4. Social Issues  

**Chapter VI: Project Plan**

1. Gantt Chart  
2. Task Analysis  
3. Evaluation Plan  
4. Risk Analysis  

**Chapter VII: Design**

1. Interaction Design
2. UI Design 38
3. Function Of Design 40

Chapter VIII: Implementation 42
1. Project Milestones 42
2. Implementation Steps 42

Chapter IX: Evaluation 48
1. Before Evaluation 48
2. Usability Evaluation 49
3. Functionality Evaluation 54

Chapter X: Conclusions 60
1. Features And Limitations 60
2. Future Work 60

References 62

Appendix 67
Source Code 67
Chapter I: Introduction

1. Overview

Nowadays smartphones are becoming more and more related to people’s lives. The sales of smartphones have beaten the sales of feature phones for the first time in 2013, says Gartner (2014). The same source observes that Android has captured most market share in the smartphones market. Its market share was 78.4% in 2013 and keeps growing. This indicates that developing application for Android will serve the greatest percentage of people in the world. Therefore this project has chosen Android as its development platform.

Comparing smartphones with feature phones, various applications make smartphones popular and useful, especially those that meet customers’ real needs. Communication is a popular activity and is also hot in application markets. Famous apps include Whatsapp, Skype, Facebook messenger, Line, Snapchat, Vine, WeChat, etc. A notable recent event is Facebook’s decision to buy Whatsapp for 19 billion dollars, Business Insider (2014) observes:

“Whatsapp has more than 450 million active users, and reached that number faster than any other company in history...Every day, more than a million people install the app and start chatting...With only 32 engineers, one WhatsApp developer supports 14 million active users, a ratio unheard of in the industry.”

It indicates not only that real time communication apps meet the deep desire of people, it also shows that the software industry is a technology-intensive industry, where a great product can rapidly make lots of profits.

A lot of people spend their time and money on communication with friends, families and strangers with smartphones everyday. This is not only due to the good mobility of smartphones, but also because creative usage of smartphones makes communication more
convenient as well. People can text, talk or send video to others with just a few seconds effort using a smartphone. The connections among people have never been closer and the world has become smaller than before. Among so many communication methods, audio talk is a convenient and natural way to engage in a conversation.

Customer can communicate with others via audio in three main ways: voice network, data network and wifi. On one hand, voice and data network are almost universally supported in cities and developed countries, so people there can sent audio information readily while wifi capability is only available in limited zones. On the other hand, wifi usually costs nothing while customers need to pay per minute or per byte for voice and data network access. After Android 4.0 (Ice Cream Sandwich), Google provided wifi with a peer-to-peer API (Application Programming Interface) for Android devices. Wifi peer-to-peer can “discover and connect to other devices when each device supports Wi-Fi P2P, then communicate over a speedy connection across distances much longer than a Bluetooth connection”, says Android Developers (2014).

This project aims to design and implement an application that can provide high quality peer to peer audio chat among smartphones via local ad hoc networking. It can be exercised in some highly confidential places like office buildings, factories, some public zones like classrooms and residence halls and also some moving environments like chat between vehicles on a road or in the country. It can provide a free confidential talk service to customer without using any other equipment.
Chapter II: Literature Review

1. Overview

This literature review aims at providing a solid theoretical foundation and approach for the design of the project. So this part will explain background knowledge concerning this project step by step, from the operating system to typical technology used in this project. This part will be divided into following sections:

- Android Platform
- Ad hoc Network
- Wifi Direct
- RTP and RTPS
- Blue Tooth

2. Android Platform

As the world’s most widely used smartphone operating system, “Android is a software toolkit for mobile phones, created by Google and the Open Handset Alliance”, says Burnette (2009). Android is so popular for two reasons: On the market side, Google makes Android an open source operating system and provides it to everyone and every company to use or modify even without their permission; On the application side, Android is based on Linux and most of its applications are written in the Java language which is familiar to millions of software developers. Therefore Android gets support from almost every smartphone company and impresses customers with its various applications.

Although Android runs on Linux, it is not a typical Linux system. Perneel, Fayyad-Kazan and Timmerman (2012) explain that “Android is not a specification, or a distribution in the traditional Linux sense. It’s not a collection of replaceable components. Android is a
chunk of software that you port to a device”. That is because Android uses a special VM (Virtual Machine) called Dalvik to emulate compiled Java codes. David Ehringer (2010) summarizes the situation as follows:

“Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included “dx” tool. The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.”

As each application runs in its own process, it has its own user ID and only accesses limited services of the operating system with permission. This design protects the data and privacy of the customer.

To coordinate an application written in Java and machine codes that drive hardware, the Android operating system is set up in four layers as shown in Figure 2.1.

![Android System Architecture](Android_Developers_2007.png)

Figure 2.1 Android System Architecture (Android Developers 2007)
The application layer supports interaction between the user and the device. It contains applications originating with the operating system and other applications downloaded from electronic markets. The application framework layer contains useful services and libraries which are used to build the content of applications, and form a GUI (Graphical User Interface) to control the display, control the notification of events, etc. The libraries layer provides native libraries and services like graphical APIs and search engines. At the bottom is the Linux kernel layer. It is the lowest layer and the foundation of the Android operating system. The Linux kernel layer consists of drivers of various hardware and includes power management for the device.

The application developed by this project will call managers in the application framework to initialize and run the program, and to establish robust connections with other devices. So the connection part is a focal point of this project.

3. Ad Hoc Network

“An ad-hoc network is the cooperative engagement of a collection of mobile nodes without the required intervention of any centralized access point or existing infrastructure” (Perkins and Royer, 1999).

![An Ad hoc Network Of Mobile Nodes](Figure 2.2 An Ad hoc Network Of Mobile Nodes (Charles E. Perkins, 2000))
Ad hoc network consist of several mobile nodes. They set up short period wireless communication via their own wifi network rather than using an AP (Access Point) to assign IP addresses or control communication protocols. In an ad hoc network, each node is not only a host but is also a router, passing packets to their destinations. It is different from a traditional network host which does not usually perform routing functions.

In a typical wifi network using wireless access points, a mobile node gets connected and transmits data generally using IP broadcast technology. Ma Weidong, Li Youping (2005) have pointed out that in traditional IPv4 communication, data dissemination and data broadcast use the UDP protocol, which is fast but unreliable. To avoid this issue, the 802.11 protocol can be used as an alternative.

I. Mobile Nodes In a Traditional Wifi Network

IEEE has standardized the 802.11 protocol for Wireless Local Area Networks (Bianchi, G. 2000). It divides into IEEE 802.11-1997 to IEEE 802.11ah. These 11 versions specify the physical layer and Medium Access Control (MAC) layer protocols used (Kuang and Williamson 2002, July). For a smartphone to use IP broadcast technology involves several steps (Lee and Choi 2008):

A. Network Discovery

A mobile connects to the network by scanning the channel first. It will scan all the possible channels until it find a valid signal. Once it finds a valid signal, it will synchronize itself with the physical layer. After successfully getting synchronization between the mobile and the physical layer, “the mobile will attempt to acquire the channel control parameters for the downlink and uplink in the MAC.” says Lee and Choi (2008).
B. Network Initialization
Once a mobile has discovered a network, it will try to establish context in the network environment by network initialization. Firstly, the mobile need to do initial ranging, make sure that the timing offset and power adjustments are correct for the mobile. Then after an initial ranging procedure, the mobile will negotiate basic capabilities. In order to make a secure connection, the mobile also engages in authorization and key exchange. Finally, the mobile is registered and establishes a connection.

C. Connection Setup
The connection setup is relatively simple after the registration step. It only needs to exchange DSx-REQ/DSx-RSR messages.

D. Nonconnected State
There are two modes of nonconnected state: one is sleep mode and another is idle mode. The mobile will enter sleep mode when inactive after a period of time. The main purpose of sleep mode is to save the battery power of the mobile. It there is a need, the mobile will wake up immediately. Idle mode means quit from the network. If the mobile wants to connect to this network again, it will need to register once more.

E. Paging
“Paging is the concept of sending out a message on a general broadcast channel that all mobiles are expected to be listening to” (Lee and Choi 2008). Paging’s main job is to broadcast messages and update locations for mobiles.

F. Mobility
Mobility has two states: Nonconnected-State Mobility and Connected-State Mobility-Handover. When a mobile moves from one base to another one in sleep mode or idle mode, it will enter Nonconnected-state mobility. When a mobile is in sleep mode, it has to register with the new network. When the mobile is in idle mode, it has to wait for
messages sent from the paging group from new network to update the mobile’s location and let it in.

G. Maintenance
When connected to a network, a mobile need to maintain its link to ensure it can transmit/receive data correctly. It uses synchronization and periodic ranging to make sure the timing and frequency is consistent with the network.

II. Mobile Node In Ad Hoc Network
Ad hoc network has a range of routing protocols: DSDV, TORA, DSR and AODV.

A. DSDV Protocol
Perkins and Bhagwat (1994) explained DSDV (Destination-Sequenced Distance Vector) Protocol, it allows a collection of mobile nodes to afford all nodes a path along which data can be exchanged while relying on no base station or access point. In an ad hoc network, the essential problem is find a shortest way to reach a destination via different nodes. Each node in this network has its neighbours and a routing table which starts with the destination station. A routing table consist of destination, next hop, sequence number, etc. Its structure is displayed in Figure 2.3.

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
<th>Metric</th>
<th>Sequence Number</th>
<th>Install</th>
<th>Flags</th>
<th>Stable_data</th>
</tr>
</thead>
<tbody>
<tr>
<td>MH&lt;sub&gt;1&lt;/sub&gt;</td>
<td>MH&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2</td>
<td>S406.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>T001.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>Ptr1.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MH&lt;sub&gt;2&lt;/sub&gt;</td>
<td>MH&lt;sub&gt;2&lt;/sub&gt;</td>
<td>1</td>
<td>S128.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>T001.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>Ptr1.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MH&lt;sub&gt;3&lt;/sub&gt;</td>
<td>MH&lt;sub&gt;2&lt;/sub&gt;</td>
<td>2</td>
<td>S564.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>T001.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>Ptr1.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MH&lt;sub&gt;4&lt;/sub&gt;</td>
<td>MH&lt;sub&gt;4&lt;/sub&gt;</td>
<td>0</td>
<td>S710.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>T001.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>Ptr1.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MH&lt;sub&gt;5&lt;/sub&gt;</td>
<td>MH&lt;sub&gt;6&lt;/sub&gt;</td>
<td>2</td>
<td>S392.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>T002.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>Ptr1.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MH&lt;sub&gt;6&lt;/sub&gt;</td>
<td>MH&lt;sub&gt;6&lt;/sub&gt;</td>
<td>1</td>
<td>S076.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>T001.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>Ptr1.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MH&lt;sub&gt;7&lt;/sub&gt;</td>
<td>MH&lt;sub&gt;6&lt;/sub&gt;</td>
<td>2</td>
<td>S128.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>T002.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>Ptr1.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>MH&lt;sub&gt;8&lt;/sub&gt;</td>
<td>MH&lt;sub&gt;6&lt;/sub&gt;</td>
<td>3</td>
<td>S050.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>T002.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td>Ptr1.M&lt;sub&gt;H&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.3 Structure Of Routing Table (Perkins and Bhagwat 1994)

When a node receives packets, it searches for their destination in its routing table then finds the preferred neighbour node for transmitting the packets to. The reason why DSDV can establish a shortest path between a start node and a destination node is that each node
not only transmit data packets but also advertises its own routing table to all its neighbours. When a node receives an updated routing table with a significant new sequence number, it replaces its old routing table with the updated one. To ensure that all the nodes have the latest routing table of the current network, the advertisements between nodes must be made frequently enough. The advantage of DSDV is that it keeps almost the same routing table between nodes, and is suitable for an ad hoc network with a low number of nodes. But the disadvantage is obvious, DSDV requires a high frequency of updates for all the nodes. It will consume lots of power and significant bandwidth doing this. And once the topology of network changes, it needs to produce new sequence numbers, therefore DSDV is not suitable for a rapidly changed network.

B. TORA
The TORA (Temporally Ordered Routing Algorithm) is a “highly adaptive loop-free distributed routing algorithm based on the concept of link reversal” (Park and Corson 1997). “The key design concept of TORA is the localization of control messages of a very small set of nodes near the occurrence of a topological change” (Royer and Toh 1999). TORA defines a area for each node. Within each area, there are several nodes. Each node only needs to know the topology of its area and to do updates within the area. Therefore, the whole ad hoc network may be large, but updates only happen in the small area where topological change occurs. Figure 2.4 displays the process of (a) route creation and (b) route maintenance. Unlike DSDV, TORA use a metric with a direction to establish a connection between nodes in the processes of route creation and route maintenance. The direction is based on the relative data flow of neighbour nodes from the original node to the destination node. When a node loses its connection with the original next step node, it can reverse the direction to find a possible approach to the destination, just as Figure 2.4(b) indicates.
C. DSR

The DSR (Dynamic Source Routing) protocol is an on-demand routing protocol. Johnson, Maltz and Broch (2001) regard it as “a simple and efficient routing protocol designed specifically for use in multi-hop wireless ad hoc networks of mobile nodes”. On-demand means the routing table only need to react when the current route has been changed. This
design improves efficiency and saves energy especially in a wide area with multiple nodes. Another character of DSR is that a network using the DSR protocol can be self-organizing and self-configuring without extra network administration and other infrastructures.

DSR has two important parts: Route Discovery and Route Maintenance.

When a node $S$ wants to establish a connection to node $D$ which $S$ does not know, it searches its Route Cache first. If there is an existing route, then it sends a data packet via the route recorded in Route Cache. But if there is no existing route, Route Discovery will be started. As the example indicates in Figure 2.5, node $A$ wants to send a packet to node $E$. It broadcasts its Route Request to its neighbour with a unique request id, start node, destination node and a list of nodes where the Route Request has been processed.

![Figure 2.5 Route Discovery example (Johnson, Maltz and Broch 2001)](image)

When a Route Request reaches the destination node $E$, $E$ will send a Route Reply message to source node $A$. $A$ caches this route and sends a packet to $E$ via this route.

![Figure 2.6 Route Maintenance example (Johnson, Maltz and Broch 2001)](image)

Route Maintenance is used to make the route integrated and available. When the original route cannot deliver a packet to its destination, the node nearest to the break point will send a Route Error message back to the source node immediately, stating a link failure...
between the current node and the next node. In Figure 2.6, the link between C and D is unavailable, so a Route Error message is sent from C to A via the reversal route. A receives the error message, deletes previous route from Route Cache, then starts to search for a backup route from Route Cache. If there is an alternative route to node E, A will replace the old route with this new one and send packets via the new route. Otherwise, A has to start a new Route Discovery process to find a new accessible way to reach destination node E.

D. AODV

AODV (Ad-hoc On Demand Distance Vector Routing) is a kind of ad hoc network routing protocol like a combination of DSR and DSDV. Each node plays a role as router and the route is generated by demand. There is also some enhancement of AODV:

First, AODV uses symmetric links between neighbour nodes rather than broadcasting a routing table to all the nodes in the same ad hoc network like DSDV. Second, AODV’s routing table only stores the best route by optimizing the response time for every establishment of a new link between a current node and neighbour nodes. After optimization, the routing table stores the best route, the shortest and least nodes between the source and the destination. Third, AODV uses a sequence number counter instead of a route cache. The benefit of using AODV is that AODV is highly adaptively to ad hoc networks with many nodes. But the drawback is also obvious: AODV will cause heavy network congestion especially in high node density areas.

![Diagram of Reverse Path Formation and Forward Path Formation](image-url)
AODV implements route discovery by broadcasting a route request (RREQ). A source node broadcasts a RREQ with its source address, source sequence number, broadcast id, destination address, destination sequence number and hop counter. Each node that receives RREQ will increase the hop counter by 1 and then broadcast it to its neighbours unless the RREQ reaches the destination node. And the destination node sends a route reply (RREP) back to the source node to indicate the accessibility of the route.

4. Wifi Direct

With the development of technology, Wifi Direct (Wi-Fi Alliance 2011) has been developed from the original 802.11 architecture. It allows users to have Peer to Peer connectivity without any other network or Access Point (AP) when they only have two devices with wifi function because the devices play the role either of AP or of client in this ad hoc network. Camps-Mur, Garcia-Saavedra and Serrano (2013) sum it up thus: “Wifi Direct devices, formally known as P2P Devices, communicate by establishing P2P Groups, which are functionally equivalent to traditional WiFi infrastructure networks”.

Figure 2.8 Example of Wifi direct supported topologies and use cases (Camps-Mur, Garcia-Saavedra and Serrano, 2013)
Figure 2.8 shows the example of wifi direct topologies. Devices can get 3G data or a local WLAN connection and wifi direct connection concurrently. Moreover a P2P Group can have a number of devices. One is called P2P GO, others are called P2P Client. In a pair of P2P devices, the device which act as AP is called P2P GO, short for P2P Group Owner. Another is a P2P Client, like a device in a traditional wireless network. These two form a cluster of devices, called a P2P Group. However P2P GO and P2P Client are not unalterable. Two devices can decide on their own role by a GO Negotiation phase as soon as they discover each other. A Go Negotiation message consist of a P2P Device Info, Operational Channel and Intent value (Asadi and Mancuso 2013, November). After a three way handshake of Negotiation, the two devices will determine who is the P2P GO and who is the P2P Client by the Intent value: the higher one becomes the P2P GO, otherwise it is a P2P Client. There is also a tie-breaker process to avoid conflict with same Intent value of two devices by setting a random time at the beginning of GO Negotiation. A P2P GO announces itself to P2P Clients by beacons, just like a traditional AP does, therefore each device in a P2P Group will discover its role. As an AP, an important duty is to allocate an IP address to a P2P Client. Hence, the P2P GO needs to deploy a DHCP (Dynamic Host Configuration Protocol) service to offer appropriate IP addresses to P2P Clients. Once a P2P Group has formed, any other P2P device can join this P2P Group just like joining a traditional wifi network. Although the roles of P2P Go and P2P Client are not unchangeable, the role of P2P GO can not be transferred within a P2P Group. That means, if a P2P GO leaves its P2P Group, the whole Group will be abandoned and have to start the GO Negotiation phase again among the rest of the devices.

A. Discovery of Devices

Device discovery starts with a normal wifi scan with the purpose of finding existing P2P Groups. At the same time, a Discovery algorithm is used to control the search process. First, a P2P device picks a network social channel from the 2.4 GHz band as its Listen channel. Then the P2P device “alternates between two states: a search state, in which the device performs active scanning by sending Probe Requests in each of the social channels;
and a *listen* state, in which the device listens for Probe Requests in its listen channel to respond with Probe Responses” (Camps-Mur, Garcia-Saavedra and Serrano 2013). If one P2P device receives a Probe Response, that means there is another P2P device around that wishes to establish a connection. Then the two P2P devices will start a P2P GO phase. The whole process can be found in the top of Figure 2.9.

![Figure 2.9 Frame Of Three P2P Group Formation Types](image)

**B. Establish Connection**

In a GO Negotiation phase, there is a three-way handshake, called GO Negotiation Request, GO Negotiation Response and GO Negotiation Confirmation. The device that receives a Probe Response sends a GO Negotiation Request first along with a number called the GO Intent value. Another P2P device receives the GO Negotiation Request,
replies with a GO Negotiation Response and, of course, another GO Intent value. Two devices commit to the device which has the highest GO Intent value becoming the P2P GO. After this step, the previous device needs to send a GO Negotiation Confirmation to confirm the result. When Go Negotiation finishes, each device know its role in the GO Group. The GO then starts to act as beacon and assigns an IP address to the GO Client using DHCP.

There are also two alternative ways to establish connections among P2P devices: P2P autonomous group formation and P2P persistent group formation as shown in Figure 2.9.

The feature of P2P autonomous group formation is that the entire P2P Group is created by only one P2P device from the beginning. Therefore, the only P2P device naturally becomes the P2P GO. Other P2P devices join this P2P Group by using a normal wifi scan. The advantage of this method is simple and quick. No GO Negotiation step is involved, and there is no alternation between states, which is useful for quickly establishing a connection between a limited number of P2P devices.

P2P persistent group formation takes the same steps as standard group formation except that it replaces GO Negotiation with an Invitation Procedure. The Invitation Procedure is a two-way handshake. It can initialize a GO Group in a short period of time.

**C. Security**

Security is an essential issue in almost every system, especially systems which provide communication services. In Wifi Direct, P2P devices establish a secure connection by WPS (Wifi Protected Setup). “WPS is an optional certification program based on technology designed to ease set up of security-enabled Wi-Fi networks in home and small office environments” (Wi-Fi Alliance 2006). It is based on WPA-2, AES (Advanced Encryption Standard) and PSK (Pre-Shared Key) to provide a secure environment for devices.
Moreover a user only needs to enter a pin or just simply push a button. The secure connection will then be setup immediately.

D. Group Conversation

When multiple devices have joined one wifi direct group, every device can send packets to its neighbour nodes. Therefore, one node can broadcast packets to all the other nodes within its range by flooding. Delivered packets may be transited through different nodes, broadcast by every node to all neighbours except the one which send packets to it. The advantage of flooding is that it is simple to implement. Its mainly disadvantage is that it uses bandwidth wastefully. There are better solutions for broadcasting packets in a wifi direct group, Khabbazian, Blake and Bhargava (2012) designed a new algorithm to perform better broadcast in ad hoc network. They proved that "having relative position information can greatly simplify the problem of reducing the total number of selected nodes using the static approach." The costing of flooding can be reduced by setting up a correct constant approximation factor using position data.

E. Presence Service

Comparing to a traditional presence service process which stores client state data in a server, a wifi direct group does not have a server to store or manage data. Therefore to implement a presence service, one approach is that one device needs to broadcast its state at a high frequency. The body of broadcast packets can be as simple as a few characters representing several states of the device.

5. Real-time Transport Protocol

It is challenging to provide multimedia services in ad hoc networks, says Mao, Bushmith, Narayanan and Panwar (2006). The most difficult part is to find a reliable existing path from source to destination in the real environment which is filled with uncertain conditions like weak signal, external interference, etc. In this situation, using a robust protocol to support multimedia transmission is necessary and important. One solution is
to use the Real-time Transport Protocol, also known as RTP. “RTP provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services” (Schulzrinne 1996). Nowadays, RTP is not only a basic framework supporting various multimedia applications, but is also a standard specification for transferring data such as audio and video via the Internet.

In fact, RTP is made up with two parts: RTP Data Transfer Protocol and RTP Control Protocol (RTCP).

A. RTP Data Transfer Protocol
RTP packets are sent from source to destination via a RTP session. In a RTP session on the Internet, the address of the source, consists of a network IP address and a port, called the Transport address. Data is also used to distinguish different RTP sessions, because different RTP sessions have different destination transport addresses. A RTP packet usually has a RTP fixed header field including a sequence number, timestamp, SSRC (Synchronization Source) identifier etc. This is an important part of a RTP session. It determines not only this session’s identification but also the data type and other important aspects that are involved. Beside fixed head fields, there is a list of contributing sources (CSRC). Mattila (2003) explains that,“Contributing source (CSRC) is stream of RTP packets that has contributed to the combined stream produced by an RTP mixer”. The list of CSRC records all the input stream sources and mixes them before delivering them to the destination. But sometimes, this list could be empty. The payload data is carried after the CSRC list. It contains the data restricted by header field, while waiting for delivery to the destination.

B. RTP Control Protocol
The RTCP (Real-Time Control Protocol) has evolved from periodic transmission. It carries RTCP packets, starting with a fixed header field which is similar to RTP data packets.
RTCP packets have an amount of information that indicates the packet type. RTCP is mainly used to achieve four purposes:

- It should reflect the effect of data distribution. This is an important purpose of RTCP.
- It should provide a special identifier for each RTP source, so that audio and video data can be synchronized with correct timestamps.
- It should control the right order of RTP and RTCP sending packets. The previous two functions are based on a good order of transmission of RTP and RTCP packets.
- It should not interfere with the whole process too much.

C. Real Time Streaming Protocol

The Real Time Streaming Protocol (RTSP) is not a part of RTP. It is another network control protocol used to associate streaming media servers with clients. According to Schulzrinne (1998), “RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video”. That means RTSP not only plays an important role when controlling the delivery of streaming data but also providing operations to control multimedia content. Moreover, there are two advantages of RTSP:

- A program can track a RTSP packets by its pointer or URL instead of by a whole multimedia file. This advantage is useful for low bandwidth environments where the transmit speed is at a low level.
- RTSP packets can be accessed by any RTSP based media player, says Singh and Schulzrinne (2000). This advantage indicates that RTSP can be used widely as an application level control protocol.

6. Bluetooth

There is an alternative kind of ad hoc network — Bluetooth. Bluetooth is a technology that connects devices with a short range radio link. It has almost become a standard equipment of various devices, from smartphones to computers. Bluetooth is widely used especially after the Bluetooth specification version 4.0, also known as BLE (Bluetooth Low
Energy), was issued. The biggest difference between BLE and previous Bluetooth versions is that BLE significantly decreases the consumption of energy (Gomez, Oller and Paradells 2012). For this reason, Apple Company improved BLE and invented iBeacon technology. It has deployed this technology in its stores and many other shopping malls recently. Google supports BLE since Android 4.0. These developments indicate that Bluetooth technology is getting increased attention nowadays.

A. Architecture Overview

Bluetooth using a special topology architecture called piconets to organize a group of Bluetooth devices. The main control unit in a Bluetooth group is called the master device. It can control up to eight slave devices. Moreover a master device in one group can be a slave device in another.

![Figure 2.10 Topology Of Bluetooth (Sairam et al. 2002)](image)

Figure 2.10 Topology Of Bluetooth (Sairam et al. 2002)
B. Connection
Unlike the wifi direct mechanism, Bluetooth devices can establish a connection immediately they are in the range of other devices. L2CAP (Logical link control and adaptation protocol) contributes to it.

“The logical link control and adaptation protocol (L2CAP) specification can be viewed as Bluetooth’s link layer. Usually, L2CAP and layers above it are implemented in software. L2CAP delivers packets received from higher layers to the other end of the link” says Bhagwat (2001).

C. Service Discovery Protocol
SDP (Service Discovery Protocol) provides a mechanism to allocate services to linked devices. A device has three roles in a group: SDP Client, SDP Server or both.
SDP Servers offer services while SDP Clients receive services. The SDP service contains different information, such as permission for an action and allocation of resources. All the services will be recorded by the SDP server. SDP is specially optimized for process services. It does not have the permission to access other fields.

The advantage of Bluetooth is low cost, battery saving and quick start. The disadvantage of Bluetooth is short range. The maximum effective transmit distance of Bluetooth is usually rather less than one hundred meters which makes Bluetooth not suitable for playing a main role in this project. However due to the advantages of Bluetooth, it is a good auxiliary function for short range confidential conversations.
Chapter III: Methodology

1. Overview
Methodology is used in software development mainly to provide a framework for the whole software engineering process. A good and suitable methodology define the tasks that need to be accomplished at each stage of development and establishes the role played by the tasks in the whole development process.

2. Selection
Among various methodologies, the Iterative Model (Jacobson et al. 1999) of software development has been chosen as the development methodology for this project. This is for three reasons:

1. The Iterative Model is rapid. It can provide a prototype within a short period of time;
2. The Iterative Model allows there to be a focus on improving the prototype with extra requirements and getting clear feedback from clients which can help to develop a correct application as well as reduce the risk of wasting time to develop unnecessary functions.
3. The Iterative Model has high flexibility. It’s easy for maintaining and modifying software.

3. Implementation Methodology
The Iterative Model is a cyclic software development process. This model does not require a whole detailed development plan at the initial stage. Each iteration is created by requirement analysis, framework design, implementation deployment, application testing and evaluation, just like the Waterfall Methodology. But the difference is, in the Iterative Model, these processes will be repeated with each new iteration from the previous iteration. Each new iteration refines the program until the final product is complete.
Therefore, in this project, the first iteration will be quickly completed with basic functionality. After testing and evaluation, experience from the first iteration will be used in order to make a new plan with more detailed features. The process will be repeated again and again until the application meets its requirements.
Chapter IV: Requirements Analysis

1. Aim
The aim of this project is to develop a peer to peer walkie talkie chat application for mobile computers. There are no extra requirements for this application except that devices support wifi direct technology. No matter whether it is used in the desert, at sea or in other more usual environments, the application should work fine.

2. Objective
The objective of this project is to develop a Java application for Android 4.0+ platforms that enables peers to discover each other and engage in two way real time voice communication over wifi direct wireless networks.

3. Requirements
Requirements spell out the detailed constraints that need to be satisfied to implement a project. Requirements can be divided into Functional and Non-Functional Requirements.

A. Functional Requirements
Functional Requirements specify the functions and services of a program. They determine the standards of needed properties, specify the behavior of implementation classes and detailed the functions of generated objects. In this project, Functional Requirements include:
• A device running this application must be able to find another such device within the range of wifi. This is a basic necessity of the whole project and is also a problem throughout its entire development due to unpredictable characteristics of real environments.
• Application must be able to establish reliable connections between devices with wifi direct technology.
• Application must be able to capture audio, encode/decode audio, transmit data and play audio.

• Application should minimize delay when transmitting data via wifi direct and process audio speedily.

• Application must establish two data channels for emitting and receiving data at the same time.

B. Non-Functional Requirements

Non-Functional Requirements mean the qualities or constraints of a project other than those derived from its functions and services. In this project, the Non-Functional Requirements include:

• Usability: Application should have an interface that is convenient and pleasant for users to interact with. This is an important aspect because a user friendly application must be simple and easy to use, otherwise it will not be used for long.

• Compatibility: Application should be compatible with Android version 4.0 or later.

• Efficiency: Application should respond to user activity quickly and correctly and consume system resources modestly.

• Extensibility: Good extensibility is necessary for maintenance. Application should have good extensibility so that a programmer can add, delete or modify functions in the program easily.

• Noise tolerance: A slight distortion of sound passed by the application is acceptable. Major distortions would be fatal for this project.

• Portability: Since Google only supplied wifi direct APIs with Android 4.0, the application derived from this project should be compatible for Android version 4.0 or later.

• Security: Audio chat could be encrypted to prevent eavesdropping. User privacy would be assured by this measure. Access to the service could also be constrained to authenticated registered users.
• Stability: Application must work reliably and robustly. No tendency for the program to crash can be tolerated.

• Testability: Application must have good testability. The programmer should be able to test the code carefully and catch bugs.

4. Use Case

![Use Case Diagram]

Figure 4.1 Use Case

Main Success Scenario:

1. Two users turn on application.

2. One user finds and establishes a connection with another user by scanning and choosing an available node.

3. User presses button to start talking and presses another button to stop talking.
Alternatives:
1a. User who forgets to turn on the application will not be found among available node list.
2a. User can not find an available node.
2b. User can not get connected with an available node.
3a. Application fails to capture user’s audio.
3b. Application fails to play audio.
Chapter V: Professional, legal, ethical, and social issues

1. Professional Issues

This project will adopt best practice, and will be developed as a professional software application. It requires highly clear and readable source code, efficient performance and secure usage for users. A major professional issue for this project is to create robust connections between devices with minimal delay. Therefore, use of a good audio compression algorithm will be a prerequisite of professional performance. In addition, this project will use open standards as opposed to proprietary technology. This will tend to improve the quality of its outcomes.

2. Legal Issues

According to the Data Protection Act (DPA) of the UK, no personal data should be insecurely stored anywhere on devices. During program run time, no identifiable data about users will appear. And with regard to the Computer Misuse Act, this application will not access a user’s system or modify or delete their files without their authorisation. All the behavior of this application will conform with UK privacy protection and computer access laws.

3. Ethical Issues

To avoid any potential for masquerading for criminal purposes using this facility, a warning will be given when describing this app in electronic market places, before a user can download this app, to advise users not to totally trust strangers who are using this application. To discourage cyberbullying, an option will be provided to record the current conversation when having a chat, so that a victim can use audio records to support criminal proceedings. And there is also an interface ethical issue. A suitably designed interface will be provided to maintain a user’s privacy and to provide convenient and easy to understand interaction with the user.
4. Social Issues

This application is a kind of social product. If users like this application, use of it may grow rapidly among potential users. Therefore addiction to its usage is a potential issue. To avoid this happening, this application will provide in a window reminders to users to track how long they talk and quit the program for an essential break after lengthy periods.

Table 5.1 Stakeholder Analysis

<table>
<thead>
<tr>
<th>Stakeholder</th>
<th>Role</th>
<th>Impact of the project</th>
<th>Be impacted by the project</th>
<th>Issues raised by the project</th>
<th>How to improve the project</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>User of the app</td>
<td>Provide feedback to help improve the app</td>
<td>Get a free chat app</td>
<td>No</td>
<td>Offer feedback to developer</td>
</tr>
<tr>
<td>Developer</td>
<td>Developer of the app</td>
<td>Develop the whole project</td>
<td>Get degree for complete project</td>
<td>Maintain and improve the app</td>
<td>Receive feedback and improve the app</td>
</tr>
<tr>
<td>Supervisor</td>
<td>Supervisor of developer</td>
<td>Supervise developer</td>
<td>Assist developer to get a degree</td>
<td>No</td>
<td>Give advice to developer</td>
</tr>
</tbody>
</table>

Table 5.1 Stakeholder Analysis
Chapter VI: Project Plan

1. Gantt Chart

![Gantt Chart Image]

Figure 6.1 Gantt Chart

2. Task Analysis

A. Iteration 1

The first iteration mainly aims at building an available prototype in a short period of time. Therefore it will start with an analysis of requirements for the first iteration. After that, two days will be spent on implementing connections between devices via wifi direct. This step will use example open source programs provided by Google. Then a whole week will be spent on adding audio functions to the application to achieve the basic functionality for this project. After that four days will be allocated to testing and evaluation.
**B. Iteration 2**
The second iteration will enhance the first prototype. After a re-analysis of requirements, two weeks will be used to improve the application, mainly focusing on stabilizing the program and connection.

**C. Iteration 3**
The third iteration aims at keeping the program stable and reducing delay. If necessary, some other features can be considered and added later when analyzing requirements for this iteration.

**D. Iteration 4**
The fourth iteration should be the final application. Therefore when this iteration is conducted, debugging will be the major work.

**E. Final Stage**
The final stage is used to write the dissertation and prepare for the demonstration and poster presentation. Additionally, two weeks will be available for slippage in case something goes wrong.

**3. Evaluation Plan**
This project will be evaluated by assembling between 6 and 10 subjects in pairs, briefing them and giving them a number of tasks to perform using the software involving mutual discovery and communication between them. This will be followed up by administering a questionnaire to these users. The questionnaire addresses various aspects of this application such as system stability, voice quality, delay time, usability etc. Users will be able to select response options which are closest to how they feel about it and give their advice in reply to some open questions that ask for answer in relation to various aspects of the application’s functionality and usability.
4. Risk Analysis

<table>
<thead>
<tr>
<th>Risk</th>
<th>Impact</th>
<th>Likelihood</th>
<th>Mitigation Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unanticipated technical difficulty delaying project</td>
<td>Medium</td>
<td>Low</td>
<td>Carefully analysis of technical issues in advance</td>
</tr>
<tr>
<td>Devices can not establish connection</td>
<td>High</td>
<td>Low</td>
<td>Carefully design connection part, modify interface code if necessary</td>
</tr>
<tr>
<td>No voice can be captured/played</td>
<td>High</td>
<td>Low</td>
<td>Use more compatible ways to drive mike and speaker</td>
</tr>
<tr>
<td>Device can not be found by another device</td>
<td>High</td>
<td>Medium</td>
<td>Check conflict between applications</td>
</tr>
</tbody>
</table>

Table 6.1 Risk Analysis Table
Chapter VII: Design

1. Interaction Design

- In the MainActivity screen, pressing a special image button will start discovering of available wifi p2p peers;
- After discovery is begun, available nearby peers will be listed with corresponding function buttons provided to start communication with them;
- And in the AudioActivity screen, users can choose to mute the audio, make audio louder by speaker or stop the conversation by pressing corresponding buttons.

2. UI Design

A. MainActivity

The MainActivity layout provides the initial view of whole application, it is designed to be simple and clear. The top small part of screen is used to place an avatar icon which also acts as a start button for discovery of wifi p2p peers. The rest of the screen lists available peers. When discovery has not been initialized, no peers are shown.
Once the discovery has been initiated, the application should find nearby available peers and add the device name to the list of MainActivity along with the peer’s status and a connect button. If there are multiple peers, the list should scroll up and down to let user pick suitable peers. The user can press connect buttons to get connected with other peers. Pressing other places in the block of other devices will get information showing the device name about to connect.

In addition, the application will perform an animation with the avatar icon when the list is scrolling. For instance, the size of icon will get smaller and disappear if the user scrolls down and the icon will grow from nothing to normal size by scrolling up.

**Figure 7.3 Initialized Discovery**

B. AudioActivity

The AudioActivity screen deals with events which occur during audio conversations. Since most of the processes are running in the background, the user interface of this activity is succinct. The layout only has three buttons for different functions:

- The speaker button starts audio output from the speaker;
- The pause button mutes audio from both the mike and the speaker;
- The stop button quits the current activity and returns the user to the MainActivity screen.

**Figure 7.4 AudioActivity Layout**
3. Function Of Design

The architecture of the system consists of five parts:

- Audio Record: uses APIs provided by Android to capture audio data from the microphone;
- Audio Encoder: encodes raw audio data to reduce the data size for transmission;
- Data Transmission: transfers encoded data packets through the wifi direct connection;
- Audio Decoder: decodes data packets to recover raw audio data;
- Audio Play: uses APIs provided by Android to play audio data.

Figure 7.5 System Architecture (Belda 2012)

A. Audio Record

Android provides the default API MediaRecorder and AudioRecord classes for recording audio data from the microphone. The application presented in this dissertation uses the AudioRecord class to capture audio. The MediaRecorder class records and stores raw audio at the same time while the AudioRecord class only records audio data and provides the data for transmission.

B. Audio Encoder

Raw PCM (Pulse-Code Modulation) audio data is large and has some redundancy. It takes extra time to send, which is a major reason for delay in a real-time application. Although Android provides an AudioEncoder API for encoding audio data, it is only for audio data which is recorded and stored in device. Thus for the application presented in this project, the AudioEncoder API is not suitable.
The application of this project has chosen a third party codec named Speex to encode/decode raw audio data.

Speex is an open source audio codec written in C/C++. The advantage of Speex is that using Speex to encode/decode audio data should be more efficient than using Java libraries. The drawback is, Android runs Java apps on the Dalvik virtual machine which doesn’t support C/C++ libraries directly. To solve this compatibility issues, the Speex library needs to be called through JNI (Java Native Interface) code.

C. Data Transmission
To connect devices by wifi direct technology, Android provides a WifiP2pManager class to control wifi p2p communication. When it is used, a BroadcastReceiver class needs to be created for receiving different intents. Once wifi direct devices have successfully found each other and established a connection, the audio data packet can be sent via sockets. In this project, the socket type employed is datagram sockets, which uses UDP (User Datagram Protocol) for transmission. The wifi direct devices get/receive socket data through Multicast, which is more efficient than unicast and cost less bandwidth than broadcast.

D. Audio Decoder
Decoding audio data is the converse process to encoding data. The application calls Speex library through JNI to decode audio data and recover audio as raw data.

E. Audio Play
Once the raw audio data is decoded, the application can use the example Android API AudioTrack to play audio from the speaker.

If the AudioRecord, AudioPlay and Data transmission threads are running at the same time, then the application can implement a real time audio conversation service.
Chapter VIII: Implementation

1. Project Milestones

According to the Iterative Model methodology, the milestones of this project can be set out as follows:

1. The application calls example wifi direct APIs to establish a wifi direct connection;
2. Both record and play functions are added to the application;
3. Data sending and receiving is enabled by socket communication functions;
4. Encode/Decode audio data is achieved with the Speex library;
5. The application is customised and performance is optimized.

2. Implementation Steps

The application was developed using Android Studio, the new IDE (Integrated Development Environment) for the Android platform. Android Studio is recommended by Google and was announced at the Google I/O conference of 2013. It is based on IntelliJ IDEA, and gets support from Google. Many new features and improvements over the Eclipse ADT make Android Studio suitable development software for this project.

A. Establish Wifi Direct Connection

To start wifi direct, first of all, a WifiP2pManager class needs to get an instance in the activity. Once the WifiP2pManager has been initialized, it can register a broadcast receiver to handle different Wifi Direct intents.

To discover available peers, WifiP2pManager class provides a discoverPeers() method to detect available peers that are in range. This method need to contain two results: onSuccess() and onFailure(). If available peers are found within the range, the application will notify user about success. If not, the application will return a error code to the programming console.
Once the discovery process has found available peers, the application will broadcast a new intent, which can be received by the broadcast receiver. Once the application receives such an intent, it can request the available peers by the `requestPeers()` method, and register them with a `PeerListListener`.

After these steps, the `WifiP2pManager` class can finally connect available peers by calling the `connect()` method. If devices have successfully established connection, it will start an `AudioActivity` to implement audio communication functions. If not, the reason for the error will be shown on the console.

B. Add Record/Play Functions

This application uses the `AudioRecorder` and `AudioPlay` APIs to capture and play audio. First of all, the application needs to set an `AudioManager` to control the volume stream. Secondly, the audio formats of `AudioRecorder` and `AudioPlay` need to be set up. The formats are required to be the same, otherwise, the device can not play the audio correctly. Starting audio recording/playing is simple and straightforward. It only needs the `startRecording()` and `play()` methods to be called. In this application, audio recording and playing are running at the same time to implement real-time audio communication. Therefore the two processes are running in two different threads which are started concurrently.

C. Send/Receive Data via Sockets

This application transmits data by using a `MulticastSocket`. Different devices send data packets to the same broadcast IP address and fetch other devices’ packet from that address. The application allocates the same ip address and port for all the devices. Then the application registers the current device with the multi-broadcast group, therefore all the devices registered in the same group can publish or fetch the same data packets.
D. Encode/Decode Audio Data

According to the decision made in the Design part, the application uses the Speex codec to encode/decode raw audio data. Because Speex is a C/C++ library (Speex actually has Java edition, this project chose the C/C++ edition for efficiency’s sake), the application can only use the Speex library via JNI.

First of all, the source code of Speex needs to be downloaded from the Speex official website: http://www.speex.org/downloads/. Then a JNI folder must be created in the project workspace of the Android application. After that, all the files of libspeex and include directories are required to be copied to the JNI folder which is created in the last step. Then a new Android.mk file should be added in the JNI folder. The Android.mk file is the file used to specify the source code for compiler system.

Besides the Android.mk file, an Application.mk file also needs to be added into the JNI folder. The Application.mk file is created to specify the module (static library or dynamic library) that is essential in the application.

Furthermore, there is a speex_config_types.h file which adds definitions of the format for encoding and decoding audio data. Without these definitions, the application will not be able to call valid methods in C files.

Another important thing is to create a JNI wrapper class speex_jni.cpp which is used to call C function code in Speex. The speex_jni.cpp file is a C++ wrapper. It not only provides a way to use C code to operate objects but also specifies related attributes for every basic data type. The encode frame size, decode frame size and the implementation for each member function is configured in the speex_jni.cpp file. It acts like a bridge which connects the C library with the Java application.
After that, a Speex Java class is created to provide an operable object for Java application. This class imports packages in Speex folder and makes a connection with the JNI wrapper class and Java application. This Speex Java class sets the compression quality for audio data, and calls corresponding methods in the JNI wrapper class by using methods declared with the native keyword.

The final step, for the Windows operating system, .so (shared object) file are generated through cygwin program. This project are developed by the Android Studio Mac edition. Therefore the .so file will automatically be created after compiling the project in the command line.

The Speex codec is now ready for service. According to the Speex Java class, the Speex codec can be initialized by creating a new Speex object instance and calling the init() method. Once the Speex codec has been initialized, encoding and decoding audio data will be processed by simply call encode() and decode() method:

E. Decoration And Optimization
The main decoration in the application presented in this project is the UI (User Interface) of MainActivity. The major part of the UI is an ExpandableListView. It can be expanded and collapsed while the header of the ExpandableListView is pinned. Moreover, the header can grow and shrink if the ExpandableListView is scrolling up and down. This ExpandableListView is modified from an open source project of singwhatiwanna (https://github.com/singwhatiwanna) in Github.

The effect of this ExpandableListView is made up with two views: StickyLayout and PinnedHeaderExpandableListView. The StickyLayout is a layout that contains a header which can grow and shrink. The PinnedHeaderExpandableListView is a special ExpandableListView which has a pinned header. The final effect in the UI is the combination of these two views.
• StickyLayout
The StickyLayout has two parts mHeader and mContent. As the design requirement, the Header should grow and shrink along with the sliding of finger. The mHeader is a header whose height can be dynamically changed. The mContent is not only a view which hold the content of PinnedHeaderExpandableListView, but also detects the direction of a user finger is sliding along. When the user moves his finger on the screen, the StickyLayout can calculate the distance on each coordinate axis, then set the suitable height for the mHeader. In this application, the height of mHeader will increase or decrease the exactly the same as the finger sliding on the Y coordinate axis. So far, the effect of the header is completed. The next step is to create a ListView to hold available peers which are detected by the Wifi P2p service.

• PinnedHeaderExpandableListView
This PinnedHeaderExpandableListView is inherited from ExpandableListView. The basic mechanism is to monitor the top item belongs to which group while scrolling, then draw the group name in the header of this view. It is easy to implement, but attention also needs to be paid to some problems:
1. Each item is a ChildView of the whole ListView. The group name of current item can be identified by looking up the position in the whole ListView.
2. Override the dispatchDraw() method to draw the group name in the header of ListView. The dispatchDraw() method is used to draw the child element. It is different from the onDraw() method which is used to draw itself. Because the background does not need to be redrawn, this application only needs to use the dispatchDraw() method to draw different group names in the header of the ListView.
3. Update header while scrolling. When the ListView is scrolling and two groups are getting close, the group name below the header will push the original header up and replace the header with its own group name. This effect is a little difficult to process. It is implemented by changing the layout’s group position.
4. The click event of header. In this ListView, the header is created by drawing and it is not a child element of the ListView. It can be regarded as a view drawn out of nowhere. It is hard to directly handle the click event for the header. Alternatively, this application judges whether the area of clicking is located inside of the header to decide how to deal with the click event.
Chapter IX: Evaluation

1. Before Evaluation

Unlike some other projects, an evaluation of this project can not be undertaken through the Internet. The application presented by this project is a real-time audio chatting Android application. It requires at least two people to test this application and give their feedback. It is hard to assemble enough volunteers to spend their precious time to take the survey and fill up the questionnaire. Therefore the questionnaire was designed carefully to keep close by related to the application and provide various options for participants. The survey was carried out through face to face meetings between the developer and experimental subjects. The developer explained every question asked by the volunteers and made sure all the volunteers understood every question. In addition, the developer had to be there with volunteers in case one of them did not know how to use this application. The advantage of this survey was saving time for both volunteers and the developer. The disadvantage was that the opinions of volunteers might get affected by the developer answering their questions. The volunteers were also more likely to give positive feedback in front of the developer. Explanations about how to use the application might give volunteers a wrong impression about the application’s usability.

The questionnaire (please see Appendix) contains two types of questions: closed questions and open questions. Each closed question has a range from one to nine representing the trend from the most negative to the most positive. It also has an extra NA option for each closed question to let the volunteers reply that the question is not applicable. The open questions appear in pairs. One is for positive side and another is for negative side, to keep a balance for the volunteers’ judgments.

The questionnaire mainly aims at collecting feedback about the usability and functionality of the application. There are four usability questions and seven functionality questions in the closed questions part. In the open questions part, there is one pair of usability
questions and two pairs of functionality questions. The ratio of usability questions to functionality questions in closed questions part is four to seven and the ratio in the open questions part is two to four. In total, the ratio of usability questions to functionality questions is six to eleven. It shows a coherent balance between the different types of questions.

![Chart 9.1 Ratio Of Questions Types]

2. Usability Evaluation

According to the questionnaire, there are four closed questions and two open questions about usability. Among the four closed questions, the ratio of usability question, UI question and other usability related question is one to one to two. In addition, the ratio of usability question to UI question in the open questions is one to one. The rationale for this ratio is that UI plays an important role in usability nowadays. Good UI design can clearly display the intents of developers and make its use easy for users to understand without or with some limited guidance. Moreover, an attractive UI provides an enjoyable user experience, which should make continued use of the application more attractive.
The survey was undertaken among ten people. Everyone answered each question and gave their opinions in the open questions part. Six out of ten people had no computer science education background. They were all asked about the meaning of some professional terms and descriptions, such as “interface”, “delay”, “robustly”. The developer of this project explained the concepts of all asked terms and descriptions objectively. The other four volunteers who had relevant computer science knowledge background did not ask similar questions. All the volunteers were able to easily use the application to find available peers, establish connections and start an audio chat session. The only step they queried about was how to start the discovery process. Eight out of ten people had this query. It indicated that there needed to be a more noticeable symbol or other obvious sign for the discovery button.

The table below shows the results of answers about usability of the application:

<table>
<thead>
<tr>
<th>Statement</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>NA</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>The application is easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7.5</td>
</tr>
<tr>
<td>The application has an attractive interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>8</td>
<td></td>
<td></td>
<td>8.8</td>
</tr>
<tr>
<td>The application is convenient for your life</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td>7.1</td>
</tr>
<tr>
<td>You will recommend this application to your friend</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>6</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>5.8</td>
</tr>
</tbody>
</table>

Table 9.1 Answers About Usability (The Higher The Better)

In the table above, the first question addresses usability. The second question is aimed at UI design. The other two questions address other related usability questions.

- According to the table above, the UI is the best part. This suggests that the application’s UI is well designed which maybe due to its use of a flat design and gradient Gaussian blur effects.
• The usability achieves the second highest score with equal votes between score seven and eight. The low variance implies that the experience of usability is similar for most volunteers. Therefore the score of usability can be regarded as an accurate grade in the survey. The average score of usability is 7.5. The score may be affected by the obscure discovery button which caused trouble for volunteers to use. Other settings and interaction were found satisfactory by the volunteers. In general, the usability of this application would seem to be adequate although button visibility needs to be improved.

• The third question asked users to rate the convenience of the application in practical use in life. Four different scores were given and different votes given for each score. This phenomenon shows that the application has various attractions for different people while the most people think it is slightly better than average. In other words, the score means that the application has scope to be made more convenient for practical use.

• The fourth question related to whether user might be prepared to use the application. Unfortunately, this issue got the lowest score. The reason may be that the application is not a mature application yet. It still needs new functions, less bugs, enhanced performance, etc.
In addition, two open questions addressed usability beyond the closed questions. This pie chart shows the best improvement that volunteers suggested:

According to the pie chart, some conclusions can be drawn:

- The most wanted improvement is bug fixed. This shows that the biggest problem for the application is bugs, which may also be a major reason for the reluctance to recommend the software. The detailed analysis of the bugs is given in the next section of the evaluation chapter.

- The second wanted improvement is to enhance current performance. This proves the application actually is not a mature application. It still needs to be optimized. The detailed analysis of performance enhancement can also be found in the next section.
The rest of the people (20%) thought new functions and adjusting the screen resolution for different devices were necessary. More functionality is always sought by users. Better adaptability to platform diversity is also commonly sought for most software.

The last open question of usability is about the design of the UI. The pie chart below shows the reason why the UI can get a high score in the survey.

The pie chart shows:
- Fashionable design style and suitable images are almost the determining factors to a small application like the application presented in this project. Fashionable design style caters for the aesthetic needs of the volunteers. The flat button design and gradient Gaussian blur effect which is the main design trend nowadays, bring an enjoyable experience to the volunteers. The image in the application is rounded to fit a current popular design trend. Moreover, the whole application uses an uniform color for different contents, which delivers a sense of unity.
• One of the ten volunteers chose the interaction effect as the best. This effect looks like animation, implemented by a special layout. The interaction provides a dynamic visual effect in a static application and confers a special feeling for users.

• It is noteworthy that ten per cent of people gave a comment of “None”. The reason may be that the volunteer found the UI style bland. Different people have different aesthetics, one UI style cannot satisfy everyone. Another possible reason is that the volunteer was tired of answering questions and thus wrote a “None” comment.

3. Functionality Evaluation

The functionality of the application is the major object of evaluation in the survey. Therefore, it contains seven closed questions and four open questions on functionality. Among the seven closed questions, the ratio of network questions to system questions to audio questions is three to two to two. The network questions are questions related to wifi direct ad hoc networking, and mainly concern the success rate of discovery and connection. If the application cannot even establish wifi p2p connection, the application will be a failure. The system questions investigate the fluency and robustness of the application. These aspects are not only highly related to the user experience, but are also important issues for a good application. The audio questions concern issues raised while chatting with this application. They address the aspects of audio quality and audio delay. For an audio application, the quality of audio is important. And furthermore, the delay of audio should not be too long in case users lose their patience.

<table>
<thead>
<tr>
<th>Statement</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>NA</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>The application finds peer devices easily and quickly</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td></td>
<td>7.5</td>
</tr>
<tr>
<td>The application works over a good range of distances</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td></td>
<td>7.0</td>
</tr>
</tbody>
</table>
In the table above, the first three questions are network questions, the fourth and fifth questions are system questions, the rest questions are audio questions.

According to the table above, there are some conclusions:

- The average scores of network questions are more than or equal to seven. This indicates that the wifi p2p function works well in this application. To be specific, the highest score is achieved by the first question which means the process of discovering available peer devices is fast and has a high success rate. The second highest score is 7.2, and belongs to the third question which is aimed at the running environment for the application. The high score of this environment question proves that the application can work without other data connection. That means the application can truly implement real time audio chat function with no extra wifi router or mobile data. It is a totally free application. The second question is answered by taking a series of distance tests. The tests concern distances from about 10 meter to more than 100 meter indoors and outdoors. The high score on the distance question proves one advantage of wifi direct technology is that the effective communication distance for wifi direct peers is much longer than for bluetooth.
• The system questions also achieved good results. The score of system fluency reaches 7.5. It means that the application has done a good job at controlling memory. Moreover, the high efficiency of the application is also a proof that using the C/C++ library as codec is a good choice, since the application can call C/C++ codes from JNI. The application gets an average score of 6.9 in the system robustness test. This test mainly addresses the crash rate of the program. Due to the short time of test, no crash occurred in the period of test. Therefore some of the volunteers answered this question with a subjective guess.

• The scores for the audio questions are unsatisfactory. The two audio questions only get scores of 3.5 and 3.6 which are the lowest scores among all aspects. This exposes major problems for the application. For the audio quality question, four out of ten people have chosen only a score of two for the application. It is because that lots of noise can be heard while talking with the application. Volunteers also experience a serious echo phenomenon when chatting indoors. And some volunteers also reported that the audio volume played by the phone speaker is too weak to hear. These problems are caused in the recording process. When the device’s microphone starts to record audio data, it also records the noise created by electric current. The echo comes from two sources: one is the result of playing audio data locally instead of only play it remotely, another is the real echo created in small room. Furthermore the application only uses the codec to encode/decode audio data. It does not use specific algorithms to get rid of background noise and eliminate echo. The main problem in audio delay is that the whole process of audio activity takes too much time. Using Java APIs to record/play audio is much slower than using the C library OpenSL ES which got Android support since Android 2.2. However, the MultiBroadcast process costs even more time. It broadcasts data packets to specific IP addresses and fetches data packets sent by other devices. This process is the main reason for the long delay in the application.
Besides the seven closed questions, there are two pairs of open questions. One pair aims at current performance, another pair seek opinions about future work.
The pie chart on the left represents the best effect, the pie chart on the right shows the function that most needs improvement. As the two pie charts display:

- The best part of the application is the UI. Half of the volunteers voted for the UI design. It suggests that the UI plays an important role in the application. Two out of ten people thought that the fact that the application can work without costing any fee is its most attractive feature. This means that a totally free application has a potential to attract a mass of users. Another twenty per cent people appreciated the performance of the application. And the rest of the people thought that the application has wide potential for usage. It indicates that users see the advantage in running applications which work under various environment and different conditions.

- Without doubt, the audio quality and audio delay are the primary issue that need to be improved. Users have high expectations for application performance and are sensitive to bugs or mistakes in an application. Ten per cent of people want to have a suitable application layout for various devices.

![Chart 9.7 Suggestions For Future Performance](image)

The pie chart on the left represents the functions that would be most useful to add. The right pie chart indicates the functions that might most readily be dropped.
According to the left pie chart, fifty per cent of people want a group talk function most. This function can be implemented by allocating different broadcast ip addresses for different groups. In this way, each group member can only receive the audio data packets sent within the same group. Furthermore, the different broadcast ip address mechanism can effectively prevent eavesdropping. So that is a way to group talk. The second most popular function asked for is messaging. It means people living in this age are not satisfied with a single communication pattern. They require more versatile tools to communicate with others. The rest of volunteers want a custom avatar to represent themselves. It reflects that personality expression is also a desirable feature for a mature application.

Because there are not so many functions in the current application, ninety per cent of volunteers did not want to drop any functions of the application. On the other hand, it means that every function of current application is essential. The rest (ten per cent) of volunteers did not like the speaker mode of audio activity. That may because there still is lots of noise in the audio data. If devices are switched into speaker mode, loud noise will be generated.
Chapter X: Conclusions

1. Features And Limitations

This dissertation has provided a way to create a real time chat Android application via wifi direct technology. The application has the following features:

• Using wifi direct technology. It can establish connections among available peer devices.
• Using multi broadcast. It can transmit audio data packets.
• Using the Speex library. It can encode/decode raw PCM data.
• It uses a special UI to hold elements and provide interaction.
• It implements two speaker modes for audio chatting.

The application has the following shortcomings:

• The quality of audio needs to be improved.
• The delay in delivering audio needs to be minimized.

2. Future Work

According to the questionnaires and evaluation chapter, there are several tasks that need to be completed in the future:

• Improving the audio quality. Audio quality is a major problem with this project. To improve it, the echo cancellation module and noise elimination module of the Speex library need to be implemented in this project.
• Minimizing delay. Delay in audio delivery is also an important problem. It takes a long time to hear response at the current stage. In the future, the OpenSL ES library could be used to capture/play audio.
• Adding a message function. This function is not difficult to implement. Since the audio data can be transmitted between peer devices, the text data can also be sent and received in the same way. The only thing that needs to be done is to create a window with update method to display messages.
• Adding a group talk function. As mentioned in the previous section, a group talk function could be realised by setting different broadcast ip addresses for different groups. Each group is defined by a different broadcast ip address. One user can create a group of his own, then send an invitation to other peer devices along with the unique broadcast ip address.

• Adding custom avatar function. To achieve a custom avatar function, one available solution is to replace current avatar with alternative avatars in a specific gallery content. When the user click the ImageButton, there will pop up a dialog window displaying various avatars for user to choose. The user could pick his favourite avatar by clicking, then the chosen avatar will replace the original one on the screen.

• Developing iOS (iPhone Operating System) version. Since the iOS 7 supports Multipeer Connectivity framework which provides services for discovering and sending data to nearby iOS devices using infrastructure wifi network, wifi direct and Bluetooth, it is possible to create an iOS application like the application presented in this dissertation.
References


Appendix

Source Code

AudioActivity:

```java
package leewin.macs.uk.ac.hw.witalk;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.net.UnknownHostException;

import leewin.macs.uk.ac.hw.witalk.R;

public class AudioActivity extends Activity {
    private AudioManager audioManager = null;
    private AudioRecord audioRecord = null;
    private AudioTrack audioTrack = null;
    private boolean isSpeaker = false;
    private boolean isPlaying = true;
    private boolean isRun = true;

    public AudioActivity() {
        super();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_audio);

        audioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
        audioRecord = new AudioRecord(audioManager, AudioFormat.CHANNEL_IN_MONO,
                                       AudioFormat.SAMPLE_SIZE_16BIT, AudioFormat.ENCODING_PCM_16BIT,
                                       AudioFormat.CHANNEL_IN_MONO, AudioFormat.TECHNOLOGY_PCM);
        audioTrack = new AudioTrack(AudioManager.TECHNOLOGY_PCM);

        // Set up the AudioRecord
        audioRecord.setAudioSessionId(-1); // sets the audio session ID
        audioRecord.startRecording();

        // Set up the AudioTrack
        audioTrack.setAudioSessionId(-1); // sets the audio session ID
        audioTrack.play();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_audio, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.item1:
                Log.d("AudioActivity", "Record was started/ended.");
                return true;
            case R.id.item2:
                Log.d("AudioActivity", "Recording ended.");
                audioRecord.stop();
                audioRecord.release();
                audioTrack.stop();
                audioTrack.release();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("AudioActivity", "Destroy.");
    }

    @Override
    public void onDestroyOptionsMenu() {
        super.onDestroyOptionsMenu();
    }

    @Override
    public void onAudioAvailable() {
        super.onAudioAvailable();
    }

    @Override
    public void onAudioStopped() {
        super.onAudioStopped();
    }

    @Override
    public void onAudioError(java.lang.Exception e) {
        super.onAudioError(e);
    }

    @Override
    public void onAudioCaptured(int numBytes) {
        super.onAudioCaptured(numBytes);
    }

    @Override
    public void onAudioBufferUnderflow() {
        super.onAudioBufferUnderflow();
    }

    @Override
    public void onAudioBufferOverflow() {
        super.onAudioBufferOverflow();
    }

    @Override
    public void onRecordedFrames(int numBytes) {
        super.onRecordedFrames(numBytes);
    }

    @Override
    public void onRecordedFrames(int numBytes, int start, int end) {
        super.onRecordedFrames(numBytes, start, end);
    }

    @Override
    public void onRecordedFrames(int numBytes, int start, int end, int frames)
```
private boolean micOn = true;
private MulticastSocket multicastSocket = null;
private static String BROADCAST_IP="224.0.0.1";
private static int BROADCAST_PORT = 8988;
private InetAddress inetAddress = null;
private Speex speex = new Speex();
private byte[] processedData = new byte[1024];
private short[] rawdata = new short[1024];
private String TAG = "AudioActivity";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_audio);
    try{
        inetAddress = InetAddress.getByName(BROADCAST_IP);
        multicastSocket = new MulticastSocket(BROADCAST_PORT);
        multicastSocket.setTimeToLive(1);
        multicastSocket.joinGroup(inetAddress);
    }catch (UnknownHostException uhe){
        Log.d(TAG, "Unknown host in Audio Activity");
    }catch (IOException ioe){
        Log.d(TAG,"IO exception in Audio Activity");
    }
    setVolumeControlStream(AudioManager.MODE_IN_COMMUNICATION);
    Intent intent = getIntent();
    String player_name = intent.getStringExtra("player_name");
    TextView playerNameTextView = (TextView) findViewById(R.id.player_name);
    playerNameTextView.setText(player_name);
    init();
    recordAndSend();
    receiveAndPlay();
}

private void init() {
    speex.init();
    int min = AudioRecord.getMinBufferSize(8000, AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT);
    audioRecord = new AudioRecord(MediaRecorder.AudioSource.VOICE_COMMUNICATION, 8000, AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT, min, AudioRecord.ENCODING_PCM_16BIT);
AudioFormat.ENCODING_PCM_16BIT, min);

int maxJitter = AudioTrack.getMinBufferSize(8000, AudioFormat.CHANNEL_OUT_MONO, AudioFormat.ENCODING_PCM_16BIT);

audioTrack = new AudioTrack(AudioManager.MODE_IN_COMMUNICATION, 8000, AudioFormat.CHANNEL_OUT_MONO, AudioFormat.ENCODING_PCM_16BIT, maxJitter, AudioTrack.MODE_STREAM);

private void recordAndSend()
{
    new Thread(new Runnable() {
        @Override
        public void run() {
            try{
                audioManager = (AudioManager) getSystemService(Context.AUDIO_SERVICE);
                audioManager.setMode(AudioManager.MODE_IN_COMMUNICATION);
                audioRecord.startRecording();
                while (isRun)
                {
                    if (micOn)
                    {
                        int readBytes = 0;
                        int len;
                        readBytes = audioRecord.read(rawdata,0,1024);
                        if (readBytes>0) {
                            len = speex.encode(rawdata, 0, processedData, 0);
                            byte[] encData = new byte[len + 1];
                            DatagramPacket packet = new DatagramPacket(encData,encData.length,inetAddress,BROADCAST_PORT);
                            multicastSocket.send(packet);
                            Log.i(TAG,"Start sending packet....");
                            }
                        }
                    }
                }catch (IOException ioe){
                    Log.d(TAG,"IO Exception in sending packet.");
                    }
                }).start();
            }
        }
    }

private void receiveAndPlay(){

new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            byte[] encData = new byte[1024];
            short[] decData = new short[256];
            audioTrack.play();
            while (isRun) {
                int dec = speex.decode(encData, decData, encData.length);
                if (dec > 0) {
                    audioTrack.write(decData, 0, dec);
                }
            }
        } catch (IOException ioe) {
            Log.d(TAG, "IO Exception in receiving packet.");
        }
    }
}).start();

public void modeChange(View view) {
    Button modeBtn = (Button) findViewById(R.id.btn_mode);
    if (isSpeaker == true) {
        audioManager.setSpeakerphoneOn(false);
        isSpeaker = false;
        modeBtn.setText(R.string.btn_mode_call);
    } else {
        audioManager.setSpeakerphoneOn(true);
        isSpeaker = true;
        modeBtn.setText(R.string.btn_mode_speaker);
    }
}

public void audioPlay(View view) {
    Button playBtn = (Button) findViewById(R.id.btn_play);
    DatagramPacket packet = new
if(isPlaying){
    audioRecord.stop();
    audioTrack.pause();
    isPlaying=false;
    playBtn.setText(R.string.btn_play);
}else{
    audioRecord.startRecording();
    audioTrack.play();
    isPlaying=true;
    playBtn.setText(R.string.btn_pause);
}
}

public void audioStop(View view){
    if (isRun){
        isRun = false;
        multicastSocket.disconnect();
        multicastSocket.close();
        startActivity(new Intent(this, MainActivity.class));
    }
}

DeviceActionListener:

package leewin.macs.uk.ac.hw.witalk;

import android.net.wifi.p2p.WifiP2pConfig;
import android.net.wifi.p2p.WifiP2pDevice;

/**
 * An interface-callback for the activity to listen to fragment interaction
 * events.
 */
public interface DeviceActionListener {
    void cancelDisconnect();
    void connect(WifiP2pConfig config);
}
void disconnect();

}

Group:

package leewin.macs.uk.ac.hw.witalk;

public class Group {

    private String title;

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
}

MainActivity:

package leewin.macs.uk.ac.hw.witalk;

import java.util.ArrayList;
import java.util.List;

import leewin.macs.uk.ac.hw.witalk.PinnedHeaderExpandableListView.OnHeaderUpdateListener;
import leewin.macs.uk.ac.hw.witalk.StickyLayout.OnGiveUpTouchEventEventListener;

import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.net.wifi.p2p.WifiP2pConfig;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pDeviceList;
import android.net.wifi.p2p.WifiP2pManager;
import android.os.Bundle;
import android.app.app.AppCompatActivity;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.widget.AbsListViewLayoutParams;
import android.widget.BaseExpandableListAdapter;
import android.widget.Button;
import android.widget.ExpandableListView;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends Activity implements
  ExpandableListView.OnChildClickListener,
  ExpandableListView.OnGroupClickListener,
  OnHeaderUpdateListener, OnGiveUpTouchEventListener,
  WifiP2pManager.ChannelListener, DeviceActionListener,
  WifiP2pManager.PeerListListener {

  public static final String TAG = "MainActivity";

  private PinnedHeaderExpandableListView expandableListView;
  private StickyLayout stickyLayout;
  private ArrayList<Group> groupList;
  private ArrayList<List<WifiP2pDevice>> childList;

  private MyexpandableListAdapter adapter;

  private ViewGroup mHeaderView;

  private ProgressBar mProgressBar;

  private WifiP2pManager manager;
  private ArrayList<WifiP2pDevice> peers;
  private final IntentFilter intentFilter = new IntentFilter();
  private WifiP2pManager.Channel channel;
  private boolean retryChannel = false;
  private boolean isWifiP2pEnabled = false;
  private BroadcastReceiver receiver = null;
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    expandableListView = (PinnedHeaderExpandableListView) findViewById(R.id.expandablelist);
    stickyLayout = (StickyLayout) findViewById(R.id.sticky_layout);
    mProgressBar = (GoogleProgressBar) findViewById(R.id.google_progress);

    intentFilter.addAction(WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION);
    intentFilter.addAction(WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION);

    manager = (WifiP2pManager) getSystemService(Context.WIFI_P2P_SERVICE);
    channel = manager.initialize(this, getMainLooper(), null);
    initData();

    adapter = new MyexpandableListAdapter(this);
    expandableListView.setAdapter(adapter);

    for (int i = 0, count = expandableListView.getCount(); i < count; i++) {
        expandableListView.expandGroup(i);
    }

    expandableListView.setOnHeaderUpdateListener(this);
    expandableListView.setOnChildClickListener(this);
    expandableListView.setOnGroupClickListener(this);
    stickyLayout.setOnGiveUpTouchEventListener(this);
}

/** register the BroadcastReceiver with the intent values to be matched */
@Override
public void onResume() {
    super.onResume();
    receiver = new WiFiDirectBroadcastReceiver(manager, channel, this);
    registerReceiver(receiver, intentFilter);
}
@Override
public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}

/**
 * @param isWifiP2pEnabled the isWifiP2pEnabled to set
 */
public void setIsWifiP2pEnabled(boolean isWifiP2pEnabled) {
    this.isWifiP2pEnabled = isWifiP2pEnabled;
}

/**
 * initData
 */
void initData() {
    groupList = new ArrayList<Group>();
    Group group = null;
    for (int i = 0; i < 1; i++) {
        group = new Group();
        group.setTitle("Available Peers: ");
        groupList.add(group);
    }
    childList = new ArrayList<List<WifiP2pDevice>>();
    peers = new ArrayList<WifiP2pDevice>();
}

@Override
public void onPeersAvailable(WifiP2pDeviceList peerList) {
    if (mProgressBar != null && mProgressBar.isActivated()) {
        mProgressBar.setVisibility(View.GONE);
    }
    childList.clear();
    peers.clear();
    peers.addAll(peerList.getDeviceList());
    childList.add(peers);
    adapter.notifyDataSetChanged();
}
if (peers.size() == 0) {
    Log.d(MainActivity.TAG, "No devices found");
    return;
}

public void refreshPeople(View view) {
    if (!isWifiP2pEnabled) {
        Toast.makeText(MainActivity.this, R.string.p2p_off_warning,
                   Toast.LENGTH_SHORT).show();
    }
    adapter.onInitiateDiscovery();
    manager.discoverPeers(channel, new WifiP2pManager.ActionListener() {

        @Override
        public void onSuccess() {
            Toast.makeText(MainActivity.this, "Discovery Initiated",
                    Toast.LENGTH_SHORT).show();
        }

        @Override
        public void onFailure(int reasonCode) {
            Toast.makeText(MainActivity.this, "Discovery Failed : " + reasonCode,
                    Toast.LENGTH_SHORT).show();
        };
    });
    adapter.notifyDataSetChanged();
}

@Override
public void connect(WifiP2pConfig config) {
    manager.connect(channel, config, new WifiP2pManager.ActionListener() {

        @Override
        public void onSuccess() {
            // WiFiDirectBroadcastReceiver will notify us. Ignore for now.
        }

        @Override
        public void onFailure(int reasonCode) {

        }
    });
}
public void onFailure(int reason) {
    Toast.makeText(MainActivity.this, "Connect failed. Retry.",
    Toast.LENGTH_SHORT).show();
}  
};
}

@Override
public void disconnect() {
    manager.removeGroup(channel, new WifiP2pManager.ActionListener() {

    @Override
    public void onFailure(int reasonCode) {
        Log.d(TAG, "Disconnect failed. Reason :");
    }

    @Override
    public void onSuccess() {
        Toast.makeText(MainActivity.this,"Disconnect successfully.",Toast.LENGTH_SHORT).show();
    }
}
}

@Override
public void onChannelDisconnected() {
    // we will try once more
    if (manager != null && !retryChannel) {
        Toast.makeText(this, "Channel lost. Trying again", Toast.LENGTH_LONG).show();
        adapter.resetData();
        retryChannel = true;
        manager.initialize(this, getMainLooper(), this);
    } else {
        Toast.makeText(this, "Severe! Channel is probably lost permanently. Try Disable/Re-Enable P2P.",
        Toast.LENGTH_LONG).show();
    }
}
@Override
public void cancelDisconnect() {
    if (manager != null) {
        if (adapter.getDevice() == null
            || adapter.getDevice().status == WifiP2pDevice.CONNECTED) {
            disconnect();
        } else if (adapter.getDevice().status == WifiP2pDevice.AVAILABLE
            || adapter.getDevice().status == WifiP2pDevice.INVITED) {
            manager.cancelConnect(channel, new WifiP2pManager.ActionListener() {
                @Override
                public void onSuccess() {
                    Toast.makeText(MainActivity.this, "Aborting connection",
                        Toast.LENGTH_SHORT).show();
                }

                @Override
                public void onFailure(int reasonCode) {
                    Toast.makeText(MainActivity.this, "Connect abort request failed. Reason Code: " + reasonCode, 
                        Toast.LENGTH_SHORT).show();
                }
            });
        }
    }
}

public void updateThisDevice(WifiP2pDevice p2pDevice) {
    WifiP2pDevice device = p2pDevice;
    peers.add(device);
    adapter.notifyDataSetChanged();
}

class MyexpandableListAdapter extends BaseExpandableListAdapter {
    private Context context;
    private LayoutInflater inflater;
    private WifiP2pDevice device;
public MyexpandableListAdapter(Context context) {
    this.context = context;
    inflater = LayoutInflater.from(context);
}

@Override
public int getGroupCount() {
    return groupList.size();
}

@Override
public int getChildrenCount(int groupPosition) {
    if (childList.size() == 0){
        return 0;
    } else return childList.get(groupPosition).size();
}

@Override
public Object getGroup(int groupPosition) {
    return groupList.get(groupPosition);
}

@Override
public Object getChild(int groupPosition, int childPosition) {
    return childList.get(groupPosition).get(childPosition);
}

@Override
public long getGroupId(int groupPosition) {
    return groupPosition;
}

@Override
public long getChildId(int groupPosition, int childPosition) {
    return childPosition;
}

@Override
public boolean hasStableIds() {

return true;

@Override
public View getGroupView(int groupPosition, boolean isExpanded,
View convertView, ViewGroup parent) {
GroupHolder groupHolder = null;
if (convertView == null) {
    groupHolder = new GroupHolder();
    convertView = inflater.inflate(R.layout.group, null);
    groupHolder.textView = (TextView) convertView
        .findViewById(R.id.group);
    groupHolder.imageView = (ImageView) convertView
        .findViewById(R.id.image);
    convertView.setTag(groupHolder);
} else {
    groupHolder = (GroupHolder) convertView.getTag();
}

    groupHolder.textView.setText(((Group) getGroup(groupPosition))
        .getTitle());
    if (isExpanded) // ture is Expanded or false is not isExpanded
        groupHolder.imageView.setImageResource(R.drawable.expanded);
    else
        groupHolder.imageView.setImageResource(R.drawable.collapse);
    return convertView;
}

@Override
public View getChildView(int groupPosition, int childPosition,
    boolean isLastChild, View convertView, ViewGroup parent) {
ChildHolder childHolder = null;
    final String player_name = ((WifiP2pDevice) getChild(groupPosition,
        childPosition)).deviceName;
    if (convertView == null) {
        childHolder = new ChildHolder();
        convertView = inflater.inflate(R.layout.child, null);

            childHolder.deviceName = (TextView) convertView
                .findViewById(R.id.deviceName);
            convertView.setTag(childHolder);

    } else {
        childHolder = (ChildHolder) convertView.getTag();
    }

    childHolder.deviceName.setText(player_name);
    if (isExpanded) // ture is Expanded or false is not isExpanded
        childHolder.imageView.setImageResource(R.drawable.expanded);
    else
        childHolder.imageView.setImageResource(R.drawable.collapse);
    return convertView;
}
findViewById(R.id.deviceName);
childHolder.deviceStatus = (TextView) convertView
.findViewById(R.id.deviceStatus);
childHolder.imageView = (ImageView) convertView
.findViewById(R.id.image);
Button button = (Button) convertView
.findViewById(R.id.btn_connect);
button.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(MainActivity.this,AudioActivity.class);
intent.putExtra("player_name", player_name);
startActivity(intent);
}
});
convertView.setTag(childHolder);
} else {
childHolder = (ChildHolder) convertView.getTag();
}

childHolder.deviceName.setText(((WifiP2pDevice) getChild(groupPosition,
childPosition)).deviceName);
childHolder.deviceStatus.setText(getDeviceStatus(((WifiP2pDevice) getChild(groupPosition,
childPosition)).status));
return convertView;
}

@Override
public boolean isChildSelectable(int groupPosition, int childPosition) {
return true;
}

public WifiP2pDevice getDevice() {
return device;
}

private String getDeviceStatus(int deviceStatus) {
Log.d(MainActivity.TAG, "Peer status :" + deviceStatus);
switch (deviceStatus) {
    case WifiP2pDevice.AVAILABLE:
        return "Available";
    case WifiP2pDevice.INVITED:
        return "Invited";
    case WifiP2pDevice.CONNECTED:
        return "Connected";
    case WifiP2pDevice.FAILED:
        return "Failed";
    case WifiP2pDevice.UNAVAILABLE:
        return "Unavailable";
    default:
        return "Unknown";
    }
}

public void resetData() {
    childList.clear();
    initData();
    notifyDataSetChanged();
}

public void onInitiateDiscovery() {
    if (mProgressBar != null && mProgressBar.isActivated()) {
        mProgressBar.setVisibility(View.GONE);
    }
    mProgressBar.setIndeterminateDrawable(new FoldingCirclesDrawable.Builder(MainActivity.this).build());
}

class GroupHolder {
    TextView textView;
    ImageView imageView;
}

class ChildHolder {
    TextView deviceName;
    TextView deviceStatus;
    ImageView imageView;
}
@Override
public boolean onGroupClick(final ExpandableListView parent, final View v, int groupPosition, final long id) {
    return false;
}

@Override
public boolean onChildClick(ExpandableListView parent, View v, int groupPosition, int childPosition, long id) {
    Toast.makeText(MainActivity.this, childList.get(groupPosition).get(childPosition).deviceName, Toast.LENGTH_SHORT).show();
    return false;
}

@Override
public View getPinnedHeader() {
    if (mHeaderView == null) {
        mHeaderView = (ViewGroup) getLayoutInflater().inflate(R.layout.group, null);
        mHeaderView.setLayoutParams(new LayoutParams(
            LayoutParams.MATCH_PARENT, LayoutParams.WRAP_CONTENT));
    }
    return mHeaderView;
}

@Override
public void updatePinnedHeader(int firstVisibleGroupPos) {
    Group firstVisibleGroup = (Group) adapter.getGroup(firstVisibleGroupPos);
    TextView textView = (TextView) getPinnedHeader().findViewById(R.id.group);
    textView.setText(firstVisibleGroup.getTitle());
}

@Override
public boolean giveUpTouchEvent(MotionEvent event) {
    if (expandableListView.getFirstVisiblePosition() == 0) {
        View view = expandableListView.getChildAt(0);
    }
}
if (view != null && view.getTop() >= 0) {
    return true;
}
return false;

PinnedHeaderExpandableListView:

package leewin.macs.uk.ac.hw.witalk;

/**
The MIT License (MIT)

Copyright (c) 2014 singwhatiwanna
https://github.com/singwhatiwanna
http://blog.csdn.net/singwhatiwanna

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
* /

import android.content.Context;
import android.graphics.Canvas;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.widget.AbsListView;
import android.widget.ExpandableListView;
import android.widget.AbsListView.OnScrollListener;

public class PinnedHeaderExpandableListView extends ExpandableListView implements OnScrollListener {
    private static final String TAG = "PinnedHeaderExpandableListView";

    public interface OnHeaderUpdateListener {
        /**
         * 采用单例模式返回同一个view对象即可
         * 注意：view必须要有LayoutParams
         */
        public View getPinnedHeader();

        public void updatePinnedHeader(int firstVisibleGroupPos);
    }

    private View mHeaderView;
    private int mHeaderWidth;
    private int mHeaderHeight;

    private OnScrollListener mScrollListener;
    private OnHeaderUpdateListener mHeaderUpdateListener;

    private boolean mActionDownHappened = false;

    public PinnedHeaderExpandableListView(Context context) {
        super(context);
        initView();
    }

    public PinnedHeaderExpandableListView(Context context, AttributeSet attrs) {
        super(context, attrs);
        initView();
    }

    public PinnedHeaderExpandableListView(Context context, AttributeSet attrs) {
        super(context, attrs);
        initView();
    }

    private void initView() {
        // Initialize the view and setup the listener
    }
}

DISSERTATION
public PinnedHeaderExpandableListView(Context context, AttributeSet attrs, int defStyle) {
  super(context, attrs, defStyle);
  initView();
}

private void initView() {
  setFadingEdgeLength(0);
  setOnScrollListener(this);
}

@Override
public void setOnScrollListener(OnScrollListener l) {
  if (l != this) {
    mScrollListener = l;
  }
  super.setOnScrollListener(this);
}

public void setOnHeaderUpdateListener(OnHeaderUpdateListener listener) {
  mHeaderUpdateListener = listener;
  if (listener == null) {
    return;
  }
  mHeaderView = listener.getPinnedHeader();
  int firstVisiblePos = getFirstVisiblePosition();
  int firstVisibleGroupPos = getPackedPositionGroup(getExpandableListPosition(firstVisiblePos));
  listener.updatePinnedHeader(firstVisibleGroupPos);
  requestLayout();
  postInvalidate();
}

@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
  super.onMeasure(widthMeasureSpec, heightMeasureSpec);
  if (mHeaderView == null) {
    return;
  }
  measureChild(mHeaderView, widthMeasureSpec, heightMeasureSpec);
}
mHeaderWidth = mHeaderView.getMeasuredWidth();
mHeaderHeight = mHeaderView.getMeasuredHeight();
}

@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
    super.onLayout(changed, l, t, r, b);
    if (mHeaderView == null) {
        return;
    }
    mHeaderView.layout(0, 0, mHeaderWidth, mHeaderHeight);
}

@Override
protected void dispatchDraw(Canvas canvas) {
    super.dispatchDraw(canvas);
    if (mHeaderView != null) {
        drawChild(canvas, mHeaderView, getDrawingTime());
    }
}

@Override
public boolean dispatchTouchEvent(MotionEvent ev) {
    int x = (int) ev.getX();
    int y = (int) ev.getY();
    Log.d(TAG, "dispatchTouchEvent");
    int pos = pointToPosition(x, y);
    if (y >= mHeaderView.getTop() && y <= mHeaderView.getBottom()) {
        if (ev.getAction() == MotionEvent.ACTION_DOWN) {
            mActionDownHappened = true;
        } else if (ev.getAction() == MotionEvent.ACTION_UP) {
            int groupPosition = getPackedPositionGroup(getExpandableListPosition(pos));
            if (groupPosition != INVALID_POSITION && mActionDownHappened) {
                if (isGroupExpanded(groupPosition)) {
                    collapseGroup(groupPosition);
                } else {
                    expandGroup(groupPosition);
                }
                mActionDownHappened = false;
            }
        }
    }
    return false;
}
protected void refreshHeader() {
    if (mHeaderView == null) {
        return;
    }
    int firstVisiblePos = getFirstVisiblePosition();
    int pos = firstVisiblePos + 1;
    int firstVisibleGroupPos = getPackedPositionGroup(getExpandableListPosition(firstVisiblePos));
    int group = getPackedPositionGroup(getExpandableListPosition(pos));

    if (group == firstVisibleGroupPos + 1) {
        View view = getChildAt(1);
        if (view.getTop() <= mHeaderHeight) {
            int delta = mHeaderHeight - view.getTop();
            mHeaderView.layout(0, -delta, mHeaderWidth, mHeaderHeight - delta);
        } else {
            mHeaderView.layout(0, 0, mHeaderWidth, mHeaderHeight);
        }
    } else {
        mHeaderView.layout(0, 0, mHeaderWidth, mHeaderHeight);
    }

    if (mHeaderUpdateListener != null) {
        mHeaderUpdateListener.updatePinnedHeader(firstVisibleGroupPos);
    }
}

@Override
public void onScrollStateChanged(AbsListView view, int scrollState) {
    if (mHeaderView != null && scrollState == SCROLL_STATE_IDLE) {
        int firstVisiblePos = getFirstVisiblePosition();
        if (firstVisiblePos == 0) {
            mHeaderView.layout(0, 0, mHeaderWidth, mHeaderHeight);
        }
    }
}

if (mScrollListener != null) {
    mScrollListener.onScrollStateChanged(view, scrollState);
}

@Override
public void onScroll(AbsListView view, int firstVisibleItem,
    int visibleItemCount, int totalItemCount) {
    if (totalItemCount > 0) {
        refreshHeader();
    } else if (mScrollListener != null) {
        mScrollListener.onScroll(view, firstVisibleItem, visibleItemCount, totalItemCount);
    }
}

Speex:

package leewin.macs.uk.ac.hw.witalk;

import android.util.Log;

public class Speex {
    /* quality
     * 1 : 4kbps (very noticeable artifacts, usually intelligible)
     * 2 : 6kbps (very noticeable artifacts, good intelligibility)
     * 4 : 8kbps (noticeable artifacts sometimes)
     * 6 : 11kbps (artifacts usually only noticeable with headphones)
     * 8 : 15kbps (artifacts not usually noticeable)
     */
    private static final int DEFAULT_COMPRESSION = 8;
    private String TAG = "Speex";

    Speex() {
    }

    public void init() {
        load();
open(DEFAULT_COMPRESSION);
Log.d(TAG,"Speex opened");
}

private void load() {
  try {
    System.loadLibrary("speex");
  } catch (Throwable e) {
    e.printStackTrace();
  }
}

public native int open(int compression);
public native int getFrameSize();
public native int decode(byte encoded[], short lin[], int size);
public native int encode(short lin[], int offset, byte encoded[], int size);
public native void close();
}

StickyLayout:

package leewin.macs.uk.ac.hw.witalk;

/**
The MIT License (MIT)

Copyright (c) 2014 singwhatiwanna
https://github.com/singwhatiwanna
http://blog.csdn.net/singwhatiwanna

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```java
import java.util.NoSuchElementException;

import android.annotation.TargetApi;
import android.content.Context;
import android.os.Build;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewConfiguration;
import android.view.ViewGroup;
import android.widget.LinearLayout;

public class StickyLayout extends LinearLayout {
    private static final String TAG = "StickyLayout";

    public interface OnGiveUpTouchEventListener {
        public boolean giveUpTouchEvent(MotionEvent event);
    }

    private View mHeader;
    private View mContent;
    private OnGiveUpTouchEventListener mGiveUpTouchEventListener;

    // header的高度 单位: px
    private int mOriginalHeaderHeight;
    private int mHeaderHeight;

    private int mStatus = STATUS_EXPANDED;
    public static final int STATUS_EXPANDED = 1;
```
public static final int STATUS_COLLAPSED = 2;

private int mTouchSlop;

// 分别记录上次滑动的坐标
private int mLastX = 0;
private int mLastY = 0;

// 分别记录上次滑动的坐标(onInterceptTouchEvent)
private int mLastXIntercept = 0;
private int mLastYIntercept = 0;

// 用来控制滑动角度，仅当角度a满足如下条件才进行滑动：tan a = deltaX / deltaY > 2
private static final int TAN = 2;

public StickyLayout(Context context) {
    super(context);
}

public StickyLayout(Context context, AttributeSet attrs) {
    super(context, attrs);
}

@TargetApi(Build.VERSION_CODES.HONEYCOMB)
public StickyLayout(Context context, AttributeSet attrs, int defStyle) {
    super(context, attrs, defStyle);
}

@Override
public void onWindowFocusChanged(boolean hasWindowFocus) {
    super.onWindowFocusChanged(hasWindowFocus);
    if (hasWindowFocus && (mHeader == null || mContent == null)) {
        initData();
    }
}

private void initData() {
    int headerId = getResources().getIdentifier("header", "id", getContext().getPackageName());
    int contentId = getResources().getIdentifier("content", "id", getContext().getPackageName());
    if (headerId != 0 && contentId != 0) {
        // Here comes the code to initialize the sticky layout.
    }
}
mHeader = findViewById(headerId);
mContent = findViewById(contentId);
mOriginalHeaderHeight = mHeader.getMeasuredHeight();
mHeaderHeight = mOriginalHeaderHeight;
mTouchSlop = ViewConfiguration.get(getContext()).getScaledTouchSlop();
Log.d(TAG, "mTouchSlop = " + mTouchSlop);
}

throw new NoSuchElementException("Did your view with "header" or "content" exist?");

public void setOnGiveUpTouchEventEventListener(OnGiveUpTouchEventEventListener l) {  
mGiveUpTouchEventEventListener = l;  
}

@Override
public boolean onInterceptTouchEvent(MotionEvent event) {  
int intercepted = 0;

int x = (int) event.getX();
int y = (int) event.getY();

switch (event.getAction()) {

case MotionEvent.ACTION_DOWN: {
    mLastXIntercept = x;
    mLastYIntercept = y;
    mLastX = x;
    mLastY = y;
    intercepted = 0;
    break;
}

case MotionEvent.ACTION_MOVE: {
    int deltaX = x - mLastXIntercept;
    int deltaY = y - mLastYIntercept;
    if (mStatus == STATUS_EXPANDED && deltaY <= -mTouchSlop) {  
        intercepted = 1;
    } else if (mGiveUpTouchEventEventListener != null) {
        if (mGiveUpTouchEventEventListener.giveUpTouchEvent(event) && deltaY >= mTouchSlop) {  
            intercepted = 1;
        }
    }
    break;
}

}
break;

}
case MotionEvent.ACTION_UP: {
    intercepted = 0;
    mLXIntercept = mLLastYIntercept = 0;
    break;
}
default:
    break;

Log.d(TAG, "intercepted=" + intercepted);
return intercepted != 0;


@Override
public boolean onTouchEvent(MotionEvent event) {
    int x = (int) event.getX();
    int y = (int) event.getY();
    Log.d(TAG, "x=" + x + "  y=" + y + "  mlastY=" + mLastY);
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN: {
            break;
        }
        case MotionEvent.ACTION_MOVE: {
            int deltaX = x - mLLastX;
            int deltaY = y - mLLastY;
            Log.d(TAG, "mHeaderHeight=" + mHeaderHeight + "  deltaY=" + deltaY + "  mlastY=" + mLastY);
            mHeaderHeight += deltaY;
            setHeaderHeight(mHeaderHeight);
            break;
        }
        case MotionEvent.ACTION_UP: {
            // 这里做了下判断, 当松开手的时候, 会自动向两边滑动, 具体向哪边滑, 要看当前所处的位置
            int destHeight = 0;
            if (mHeaderHeight <= mOriginalHeaderHeight * 0.5) {
                destHeight = 0;
                mStatus = STATUS_COLLAPSED;
            } else {
                destHeight = mOriginalHeaderHeight;
            }
            break;
        }
    }
}
mStatus = STATUS_EXPANDED;
}

// 慢慢滑向终点
this.smoothSetHeaderHeight(mHeaderHeight, destHeight, 500);
break;
|
default:
break;
}
mLastX = x;
mLastY = y;
return true;
}

public void smoothSetHeaderHeight(final int from, final int to, long duration) {
final int frameCount = (int) (duration / 1000f * 30) + 1;
final float partation = (to - from) / (float) frameCount;
new Thread("Thread#smoothSetHeaderHeight") {

@Override
public void run() {
for (int i = 0; i < frameCount; i++) {
final int height;
if (i == frameCount - 1) {
    height = to;
} else {
    height = (int) (from + partation * i);
}
post(new Runnable() {
    public void run() {
        setHeaderHeight(height);
    }
});
try {
    sleep(10);
} catch (InterruptedException e) {
    e.printStackTrace();
}
}
}
private void setHeaderHeight(int height) {
    Log.d(TAG, "setHeaderHeight height=" + height);
    if (height < 0) {
        height = 0;
    } else if (height > mOriginalHeaderHeight) {
        height = mOriginalHeaderHeight;
    } else if (mHeaderHeight != height || true) {
        mHeaderHeight = height;
        mHeader.getLayoutParams().height = mHeaderHeight;
        mHeader.requestLayout();
    }
}

WiFiDirectBroadcastReceiver:

/*
 * Copyright (C) 2011 The Android Open Source Project
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package leewin.macs.uk.ac.hw.witalk;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.net.NetworkInfo;
import android.net.wifi.p2p.WifiP2pDevice;
import android.net.wifi.p2p.WifiP2pManager;
import android.net.wifi.p2p.WifiP2pManager.Channel;
import android.net.wifi.p2p.WifiP2pManager.PeerListListener;
import android.util.Log;

/**
 * A BroadcastReceiver that notifies of important wifi p2p events.
 */
public class WiFiDirectBroadcastReceiver extends BroadcastReceiver {

    private WifiP2pManager manager;
    private Channel channel;
    private MainActivity mainActivity;

    /**
     * @param manager WifiP2pManager system service
     * @param channel Wifi p2p channel
     * @param mainActivity activity associated with the receiver
     */
    public WiFiDirectBroadcastReceiver(WifiP2pManager manager, Channel channel, MainActivity mainActivity) {
        super();
        this.manager = manager;
        this.channel = channel;
        this.mainActivity = mainActivity;
    }

    /*
     * (non-Javadoc)
     * @see android.content.BroadcastReceiver#onReceive(android.content.Context,
     * android.content.Intent)
     */
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
            /*
             * (non-Javadoc)
             * @see android.content.BroadcastReceiver#onReceive(android.content.Context,
             * android.content.Intent)
             */
            @Override
            public void void onReceive(Context context, Intent intent) {
                String action = intent.getAction();
                if (WifiP2pManager.WIFI_P2P_STATE_CHANGED_ACTION.equals(action)) {
/ UI update to indicate wifi p2p status.
int state = intent.getIntExtra(WifiP2pManager.EXTRA_WIFI_STATE, -1);
if (state == WifiP2pManager.WIFI_P2P_STATE_ENABLED) {
    // Wifi Direct mode is enabled
    mainActivity.setIsWifiP2pEnabled(true);
} else {
    mainActivity.setIsWifiP2pEnabled(false);
    mainActivity.initData();
}
Log.d(MainActivity.TAG, "P2P state changed - " + state);
} else if (WifiP2pManager.WIFI_P2P_PEERS_CHANGED_ACTION.equals(action)) {

    // request available peers from the wifi p2p manager. This is an
    // asynchronous call and the calling activity is notified with a
    // callback on PeerListListener.onPeersAvailable()
    if (manager != null) {
        manager.requestPeers(channel, mainActivity);
        Log.d(MainActivity.TAG, "Wifi P2P Requested");
    }
    Log.d(MainActivity.TAG, "P2P peers changed");
} else if (WifiP2pManager.WIFI_P2P_CONNECTION_CHANGED_ACTION.equals(action)) {

    if (manager == null) {
        return;
    }
} else if (WifiP2pManager.WIFI_P2P_THIS_DEVICE_CHANGED_ACTION.equals(action)) {
    mainActivity.updateThisDevice((WifiP2pDevice) intent.getParcelableExtra(
        WifiP2pManager.EXTRA_WIFI_P2P_DEVICE));
}