2012

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

AN INVESTIGATION INTO AUDIO CONFERENCING ON SMART PHONES

Jui Sun

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2012

Approved by

Advisory Committee

Dr. Bryan Reinicke	Dr. Laurie Patterson

Dr. Ron Vetter, Chair

Abstract

This paper describes an approach to building an audio conferencing application for Android smart phones. As the need for audio conferencing systems grow, the smart phone becomes a viable platform for developing conferencing applications. We have implemented a centralized audio conferencing model and developed a client application which was deployed on Android-based smart phones. Experiments for battery consumption and packet delay were designed and carried out to evaluate the usability of the application. The smart phones were not affected by the application under low traffic conditions; however, the application did consume twice as much battery life under heavy traffic conditions. The results for delay testing showed that increasing the number of participants, and thereby introducing multiple audio streams, resulted in longer packet average delays. Throughout the development process, problems involving software/hardware diversification and audio signal processing were uncovered and potential solutions were proposed. The paper provides valuable information for developing VOIP applications on smart phones and can direct future development of audio conferencing systems.

Table of Contents

Tables

Figures

Chapter 1: Introduction

Voice-over-IP (VOIP) audio conferencing systems are increasingly becoming an important application on the Internet. VOIP introduces a possible solution for the long distance multi-people communication problem. As the need for voice conferencing systems continues to grow, they are being applied to many areas of business as well as in academic and social circles. VOIP systems are gaining more acceptance as the quality of service and surrounding network environment improves. A highly attractive scenario is when users can participate in a conference meeting, using their smart phones, without having to physically be present.

A smart phone is a portable handheld device with the capability of a personal computer and traditional cell phone. Smart phones are now technically capable of delivering sufficient performance for rich multimedia applications and audio communication; therefore deploying a high quality VOIP conferencing system in smart phones is now possible. Deploying an audio conferencing system in smart phones provides a new opportunity for making life more convenient for people all over the world. Although there are many products available in the marketplace, only a few of these products provide an audio conferencing service on smart phones. The lack of hardware and software resources on many older cell phone models is the primary reason for the limited availability of high quality audio conferencing systems on mobile phones.

The purpose of this research is to explore how a simple and extensible audio conferencing system for smart phones can be designed and implemented. This paper includes all of the fundamental components of how to construct an audio conferencing system for Android-based mobile phones. In addition, two experiments were designed to examine the usability of the system. The experiments examined limited battery/energy

use and measured application quality of service via delay testing.

The rest of the paper is organized as follows. Chapter 2 introduces related work and the design principles for a smart phone based audio conferencing systems. The overall methodology and system architecture is discussed in Chapter 3. Chapter 4 discusses the results of the experiments and some lessons learned. Finally, Chapter 5 provides conclusions and discusses future work that can improve audio conferencing systems on smart phones.

Chapter 2: Literature Review and Analysis

Voice over IP (VOIP)

In 1995, VocalTec developed internet-based telephony software for the personal computer. It was the first software system that compressed an analog voice signal into a packet for transmission over the Internet. In 1996, the ITU Telecommunication Standardization Sector (ITU-T) defined the first version of the H.323 standard. Because the Internet was a bandwidth constrained environment, only a few companies invested in the VOIP industry. In 2001, Yahoo Japan integrated the public switched telephone network (PSTN) and VOIP services, thereby providing a communication link between traditional telephone service and the Internet. In 2003, Skype was released and proved the reliability and quality of VoIP services in the marketplace. It convinced users of the capability and possibility of internet telephony [1].

VOIP is a technology that allows voice conversations to be conducted over the Internet. In general, voice signals are captured by microphone and the digitalized sounds are then compressed into a compact form by a codec. The sounds are encoded at the originating end, and sent over the Internet to a receiver. The receiving end decodes the packets and plays back the sound over speakers [4]. Since VOIP allows at least two way communication, it is a great deal more complicated. VOIP uses two types of Internet protocols in order to achieve end-to-end communication functionality: Signaling Control Protocol and Media Transport Protocol. Signaling Control Protocol, or Call Signaling Protocol, is used to establish and manage building and terminating connections between users. This type of protocol regulates the approach of searching for the correct target user, building connections, and processing data based on each user's processing capabilities. SIP (Session Initiation Protocol), H.323, and MGCP (Media Gateway Control Protocol)

are instances of a Signaling Control Protocol. The Media Transport Protocol (e.g. RTP and RTSP) is used to facilitate the transfer of digitalized media data after connection is built [1]. In addition, management protocols and other types of support protocols are also used in VOIP applications.

There are several advantages to using a VOIP software application. First, it costs less than PSTN. In the same VOIP service environment, no matter how far two or more end users are, the system cost is the same and oftentimes it is even free! Clients do not need to maintain different accounts for different locations. Moreover, the integration of VOIP and PSTN provide the capability to connect clients to traditional phone users as well. Although, the application of VOIP strengthens human communications, several system requirements should also be noted: quality of service, reliability, compatibility, security, and manageability. The increasing demand for reliable and available network services is becoming a concern on the Internet. Since VOIP transfers data over an IP network, any problems experienced by the network can cause performance issues of the VOIP application. Thus, this paper designs an experiment to measure the service quality of the proposed smart phone VOIP application to ensure user acceptance. More on this aspect of the design is discussed in chapter 4.

Session Initiation Protocol

The Session Initiation Protocol (SIP) is an ASCII-based, application-layer control protocol that can be used to establish, maintain, and terminate calls between endpoints [2]. It uses HTTP and SMTP concepts. It transfers users' information by text, such as IP address, ports, media ability, and codecs. The message is in plaintext; hence the receiver can realize the sender's message without decoding it [1]. SIP allows call information to be carried across networks, and provides the ability to manage connections between users.

In general, a SIP application should possess the following capabilities [2]:

- Name translation and user location.
- Feature negotiation.
- Establishes a session between the originating and target end point.
- Handles the transfer and termination of calls.

Once the connection is established, the software implements other protocols in order to achieve functionality.

A peer in a session is called the user agent. From the functionality standpoint, a user agent can be classified as either user agent client (UAC) or user agent server (UAS). A UAC, or Caller, initiates the request. A UAS, or Callee, receives the request and returns the user's information. A SIP's endpoint is typically capable to act as either a UAC or UAS [1]. From an architecture standpoint, SIP is composed of two components: clients and a server. The clients includes phone and gateway, and based on different responsibilities, the server can be a proxy server, a redirect server, a register server, a location server, a media server, a media delay server, and a Back-to-Back user agent [1, 2].

Figure 1 introduces a simple direct call peer-to-peer SIP model. It establishes a session without any proxy server. In this case, John wants to call Mary. John's machine is a user agent client and Mary's machine is a user agent server. John's machine calls the target by complete Universal Resource Identifier (URI). The machine then sends an "INVITE" plaintext to Mary's machine (UAS). Mary's machine returns messages appropriately ("100 Trying" and "100 Ringing"). After John's machine sends an ACK back to Mary's, two machines transfer data through RTP/RTCP protocol. If any user agent knows other SIP device's IP address or domain name, it can process a SIP direct

call.

In this project we aim to provide a simple audio conference application, where every mobile device acts as an UAC. The Role of UAS would be taken by a PC. In addition, mobile devices sends INVITE messages direct to UAS without any proxy server.

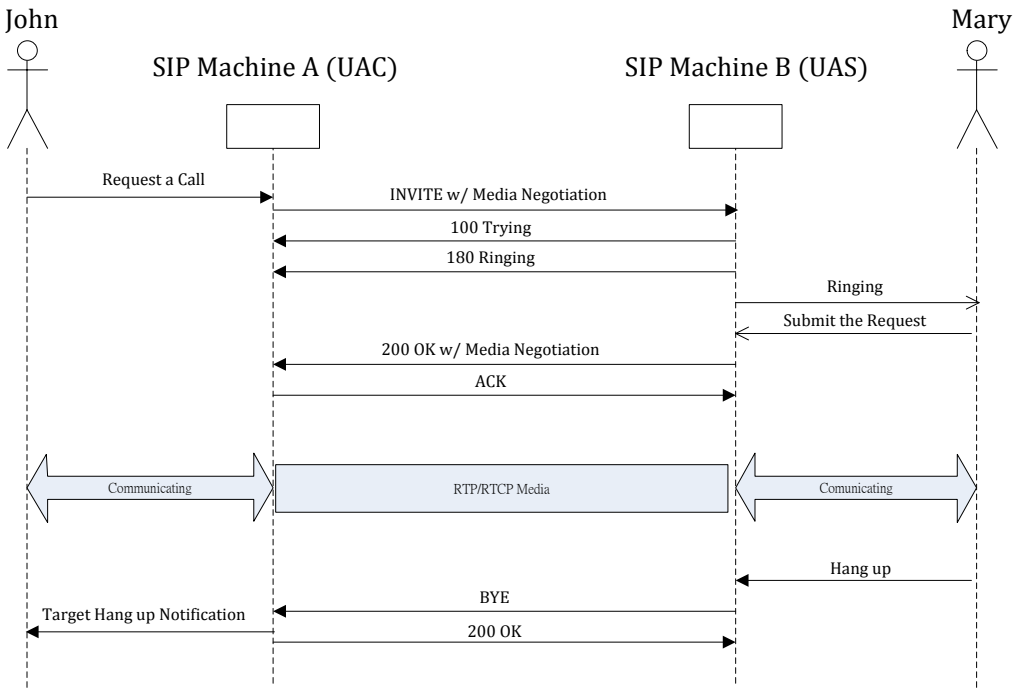


Figure 1: SIP Direct Call Model.

Codec

The word codec is a portmanteau of "compressor-decompressor". It is a mechanism of encoding/decoding a digital data stream or signal. The output is used for transmission, storage, encryption, and playback. It is widely used in multimedia [14]. With computers, there are many different types of codecs for handling sound or "audio". In hardware, a codec refers to the device that encodes analog audio and decodes the corresponding digital signal. In software, an audio codec refers to an algorithm that compresses and decompresses the digital audio data according to a given audio file

format [15]. According to the production format of an audio codec, it can be classified by different types: lossless and lossy. Lossless codecs try to maintain the original audio information, while lossy codecs trade some information to achieve other requirements. In a VOIP environment, real-time transferring is an important requirement. VOIP's codec is one application of speech coding, which is a lossy codec [16].

In year 1996, the G.279 Audio Codec was designed [1]. It needed only eight Kbytes bandwidth to achieve the quality of PSTN. Since then, several codecs, including ones for video, were designed for real-time transferring purposes. Before Google's Android 2.3, audio codecs were widely used in many VOIP applications (see Table 1). These codecs were generally thought of as acceptable for providing toll quality speech under real-time transmission [8].

Table 1

Third-party VOIP Application Supported Codec

Application Name	Support Codec
Sipdroid	Speex, G722, G711, GSM
LinPhone	Speex, G711, GSM, ILBC
SipAgent	Speex, G711, GSM
Kapanga	Speex, G.711, G.722, G.733, G.726, G.728, G.729, AMR, GSM, iLBC
fring	G.711, GSM
aSip	G.711, GSM

Audio Conferencing

Audio conferencing software, in the marketplace, commonly uses the client-

server architecture. Most server products run as a dedicated server, rather than as peer-to-peer. The TeamSpeak product allows users to install the server on their own machine. The service provider provides a location server for IP and DNS lookup. Raidcall manages servers by itself, but provides user client software. The user does not need to know detailed information, such as the server address. In addition, it extends its capability with social networking. It brings entertainment elements into a classic audio conferencing system.

According to the connection approach, conferences can be group as “Centralized Conferencing” and “Distributed Conferencing” [1]. Centralized conferencing (Figure 2) require a focus server. The focus server connects with clients independently, and upon receiving data from one client, it delivers the information to the remaining clients.

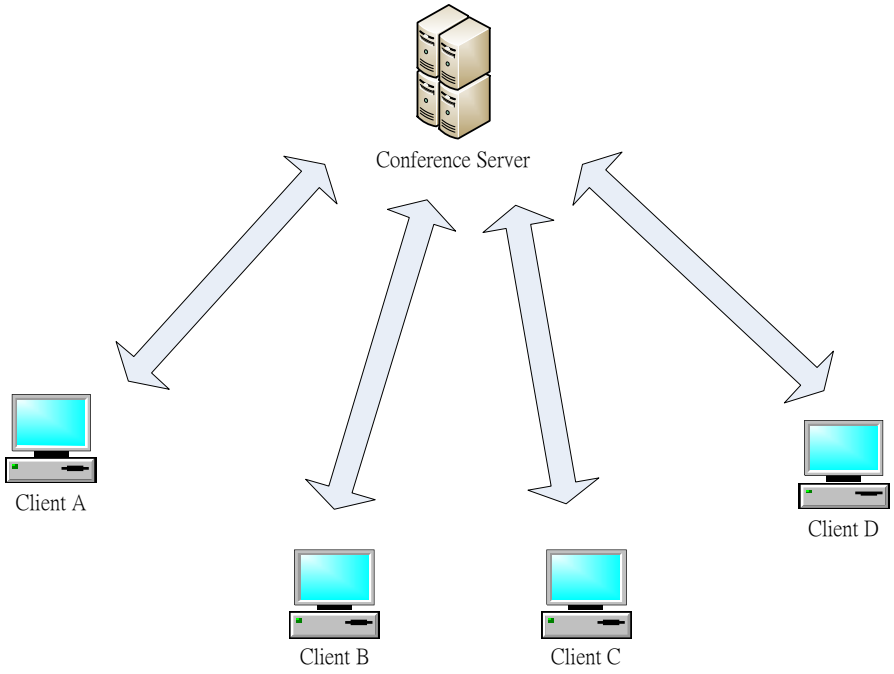


Figure 2: Centralized Conferencing Model

Distributed conferencing can be approached in two different manners: multicast and full mesh. When establishing a connection, multicast conferencing would use a

multicast IP address as its target. Unlike multicast, full mesh conferencing (Figure 3) establishes sessions with every other computer. In this case, it conforms to a peer-to-peer network architecture.

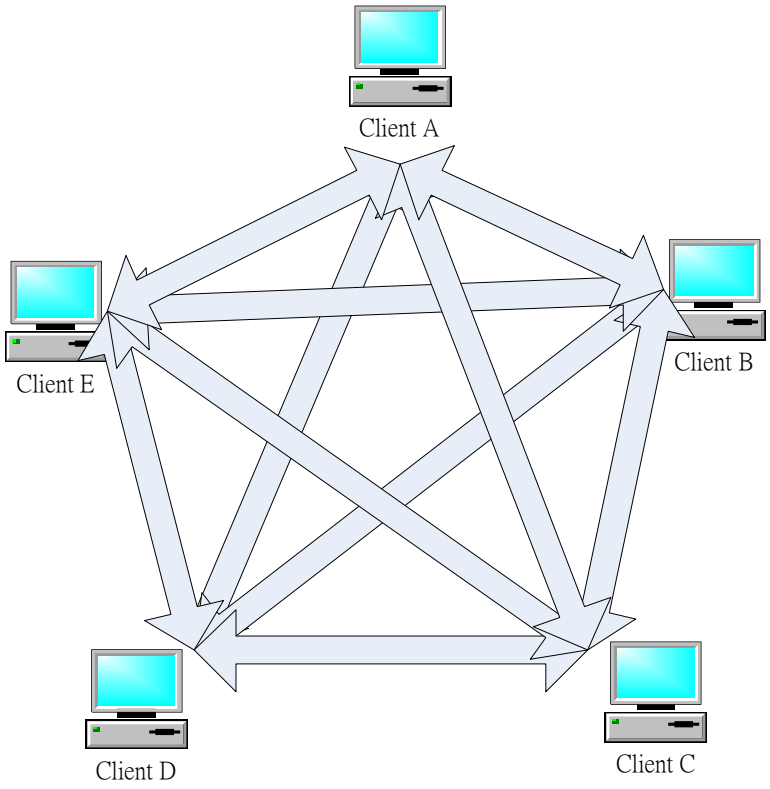


Figure 3: Full Mesh Distributed Conferencing Model

Energy Management

One of the most critical issues in smart phone application design is the management of energy consumption. Smart phones integrate the functionality of computer and mobile phone into one device; however, whereas a personal computer requires a continuous energy supply via a plugged-in power supply unit, a smart phone relies on its mobile battery. A high energy consumption application can be a heavy burden on the phone's battery and hence the number available hours for voice calling.

While a VOIP system communicates through a Wi-Fi network, the associate interface is active. Energy keeps getting consumed even if there is no data transfer. When

an application is running, program size, algorithms, and other programming factors also influence the overall operating time. Instead of increasing battery lifetime, some related work simply deals with mobile device energy consumption via software. Agarwal and Chandra presented a wakeup mechanism to solve the waste of energy by system idling [7] whereas Naeem and Namboodiri proposed an adaptive algorithm to switch codecs based on remaining battery life [6].

Generally speaking, longer operating hours represents higher usability. Because audio conferencing requires that the software be continually active, an experiment was designed to determine how long the software in mobile device can be run without running down the battery too far. More details on this aspect of the design can be found in Chapter 4.

Chapter 3: Methodology

This research project explores audio conferencing application on smart phones. The designed application allows users to install and setup their conference environment, without the use of third-party servers.

System Components

User Client

A user client program was designed and deployed in several Android-based smart phones. The program acts as a caller, or a user agent client. It established the connection by session initiation protocol. It has the following capabilities:

- Send the SIP request.
- Establish and maintain a connection between server and itself.
- Send and receive the audio stream.
- Terminate the connection with server.

In practice, the project uses SIP stack's interface and classes under the Android system. The program does not permit any incoming SIP request. As a result, it can never act a UAS.

Focus Server

A focus server is a central node of the conferencing network. All clients should transfer audio stream through this central node. The server has the following capabilities:

- Receive and respond the SIP request.
- Establish and maintain the connection between clients and itself.
- Manage anticipants.
- Clarify the incoming audio stream.
- Send an audio stream to correct endpoints.

- Receive the client termination request, and disconnect certain clients.

This project does not deploy a focus server in a smart phone device due to energy consumption concerns. Rather, the focus server is programmed and deployed on a personal computer under the Java environment. The server establishes and manages the session with every connected client, and controls all the incoming and outgoing data. It will have much more additional processes than the user client component.

System Architecture

This project uses a two-tier client-server architecture (Figure 4). The user acts with the clients' interfaces directly. Clients communicate with the server through the Internet. The server's IP address is the intended location of every client. The server includes the following modules:

- ChatServer: the server's main module which initiates the SIPEngine module and ChatHandler module.
- SIPEngine: a module which listens to client calls. Once it receives a client SIP call, it would create a specific SIPListener instances for the client and wait for the next SIP call.
- SIPListener: a module handles SIP massages with a specific client target. The listener will close if it receives a BYE message from its related client. This module also adds the client to or removes the client from the member list.
- Member: a class stores all conference participants' information.
- ChatHandler: a module mainly receives audio data from connected clients list and determines which target it should forward to.

The client includes following modules:

- AConPortableMain: the main module which generates a graphical user interface (GUI). This GUI asks user for server's information. It then creates a SIPEngineClient instance for further actions.

- SIPEngineClient: a module which sends a SIP call, terminating request, and interacts with all other SIP events. Once it builds the session between itself and the server, it then connects to the server's audio port by calling UDPSocket.
- UDPSocket: a module handles with audio data transferring. It includes method to communicate audio streams with the server.
- ChatHandler: the module of GUI which allows user to talk to server. User can turn on and off the talking threads. The model initiates InComePacket and OutComePacket and creates threads, respectively. Additional functionalities include volume adjustments and the method called from SIPEngineClient module to leave the conference room.
- InComePacket: a listener which listens to the incoming packets. Once it receives a packet it will push that data into the buffer waiting for playing.
- OutComePacket: a thread class which reads the data from the microphone's buffer and sends the data to the server.

From model standpoint, the system implements a centralized conferencing model, which is comprised of a server and all clients (participants). Figure 5 introduces the model and all possible message flow. The numbers label the outgoing flow and possible incoming flow for every smart phone.

System Operating Mechanism

The system uses a simplified SIP message to establish, manage, and terminate sessions. Figures 6, 7, and 8 introduce the actual mechanism for joining and leaving the conference system. Figure 6 presents a user client which intends to participate in the system. In this case, no other participants are currently in the system. Client A sends an INVITE request to the server. The server checks the register information, and sends 200 OK back to the client. After the client sends ACK to the Server, these two endpoints can start transferring audio streams.

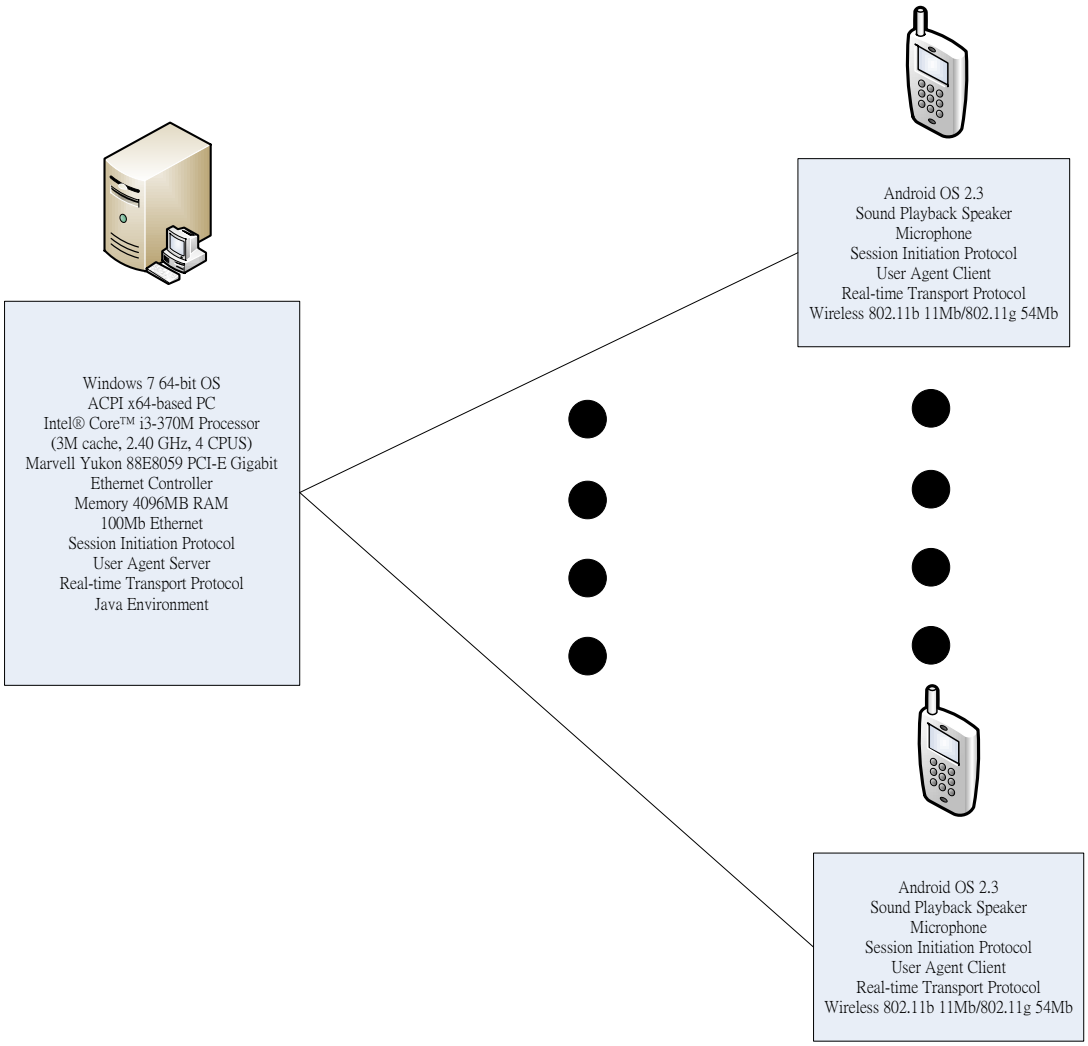


Figure 4: Two-tier Client-Server Architecture

Figure 7 gives an instance of another user who wants to attend the conference. It sends the same request, and the server sends the same response back to build the connection. Once the connection is built, it can transfer the audio stream between the client and server.

Figure 8 shows an example of terminating a connection. The Client user first sends a BYE message. Once the server receives the message, it sends back an ACK message and closes the connection. It also manages the member list and notifies conference members about the leaving client's message.

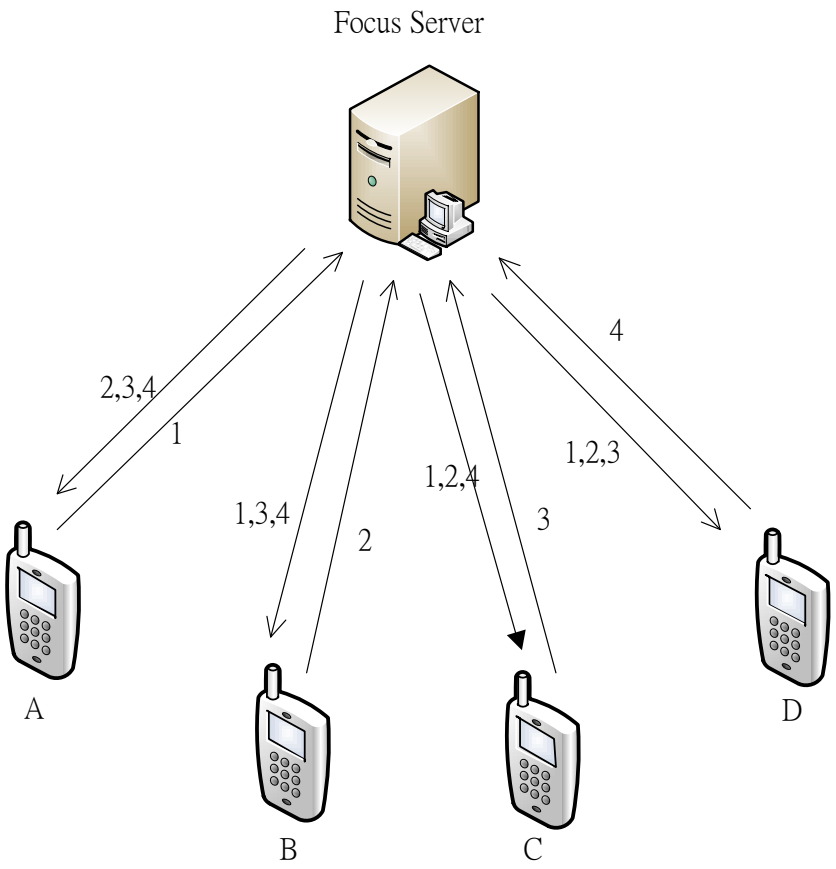


Figure 5: Centralized Message Flow Model

Figure 9 shows the architecture of system's modules. Two ports in the server are opened. The SIP message is transferred through TCP port, and the server creates different SIPListener objects for every connected client. Audio streams are transferred through UDP port. One object of server's ChatHandler is created to handle all traffic of audio stream. The Server's ChatHandler uses First-in-First-out (FIFO) algorithm to forward incoming packets. Figure 9 also shows how the modules associate with one another.

Appendix C shows the UML sequence diagram of the server. When the ChatServer is executed, the server runs a thread of the SIPEngine and a thread of the ChatHandler. The thread of SIPEngine opens a TCP port to listen to the network and waits for a SIP request. Once it receives any TCP connect from the specific TCP port, it starts a thread of SIPListener. If the SIP message is successfully created, then it adds the

client information into the member list. If the listener's connection fails or receives a "BYE" message, then it removes the member from the member list and terminates.

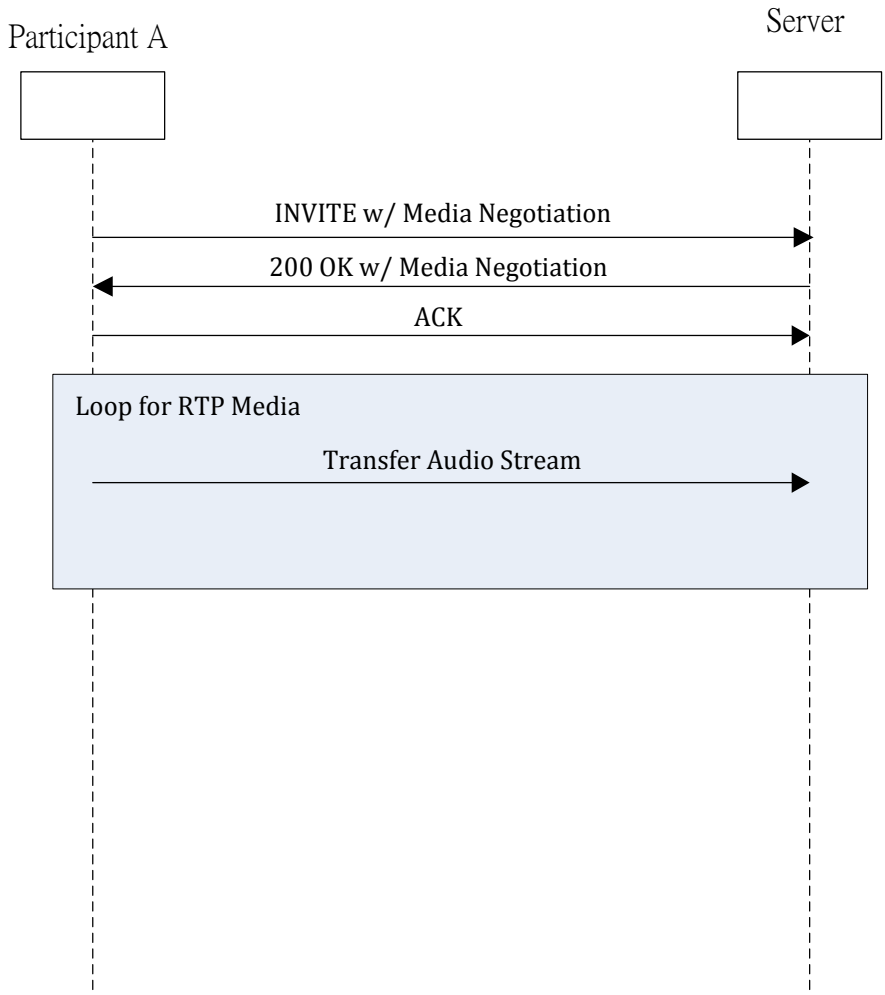


Figure 6: The process of the first client joining the conference

The client's UML sequence diagram is shown in Appendix D. When a user hit the "connect" button on the GUI, the main class will call the connect method of the instance of SIPEngineClient. Once the SIP connection is built, the object of the created SIPEngineClient calls the connect method of UDPSocket. The client then connects to the server's UDP port and returns true if it is connected. If the return connection value is true, the main GUI creates a ChatHandler UI and switches to it. Next, InComePacket's thread

is started and it plays the received audio data from server. Three buttons are shown in the GUI, which are Record, Stop, and Exit. Record generates a thread of OutComePacket, which pulls data from the microphone buffer and sends the audio data through UDPSocket. The Stop button stops the recording thread and theExit button closes related objects and returns to the main GUI.

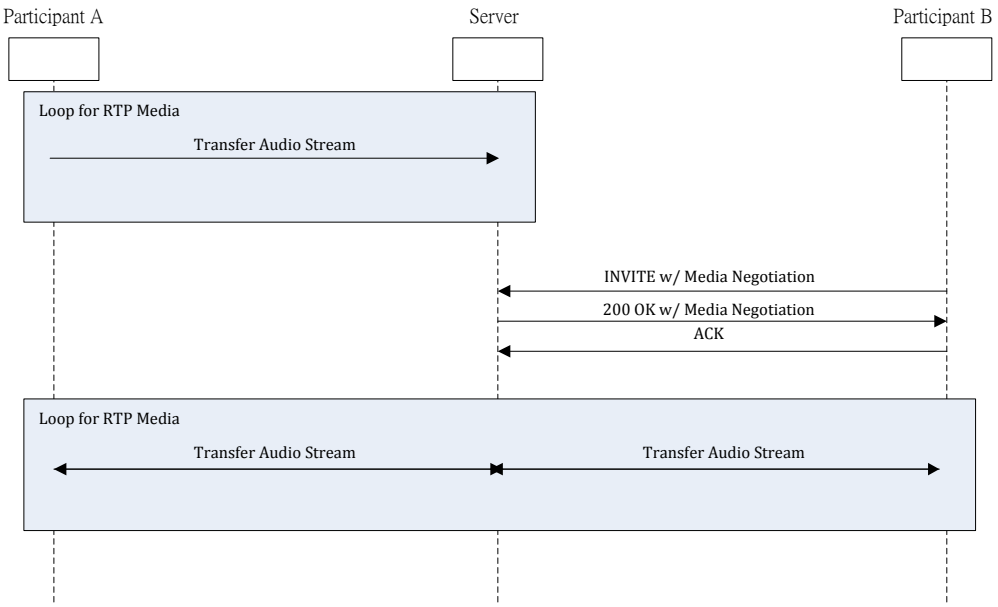


Figure 7: The process of the second client joining the conference

Designed Experiments

Experimentation Environment

Table 2 and Figure 4 illustrate the experiment's clients and servers of the audio conferencing system. The "Number" column in Table 2 distinguishes different smart phone. "Smart phone B" stands for smart phone number B in Table 2. This representation will be used when describing both the experiments and the results.

The VOIP system uses same audio settings in both power consumption and delay testing experiments. The client program will read 1024 Bytes from microphone's buffer

and sends to the server. The specification of audio data is configured as followed:

- Audio format: PCM 16 bit
- Channel configuration: Mono
- Sample rate: 8000 Hertz

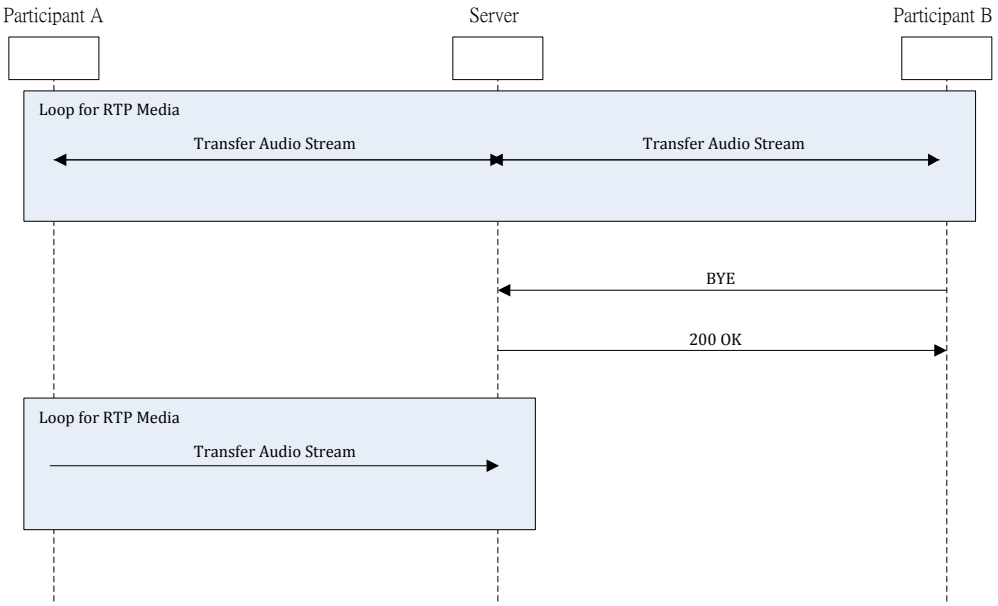


Figure 8: The process of a client leaving the conference.

Power Consumption

In order to evaluate the usability of the conferencing system, an experiment evaluating power consumption was designed. Measurements were taken to test how long it takes for the system to decrease a certain level of smart phones' battery life. Experiments included three conditions:

- Without system (VOIP application) running: measure the power consumption duration without running the audio conferencing system.
- Without data transferring: connect the client program to the server without audio data transferring.

- Heavy data transferring: connect to the server and keep transferring data during measurements. Speakers and microphones are turned on for all devices.

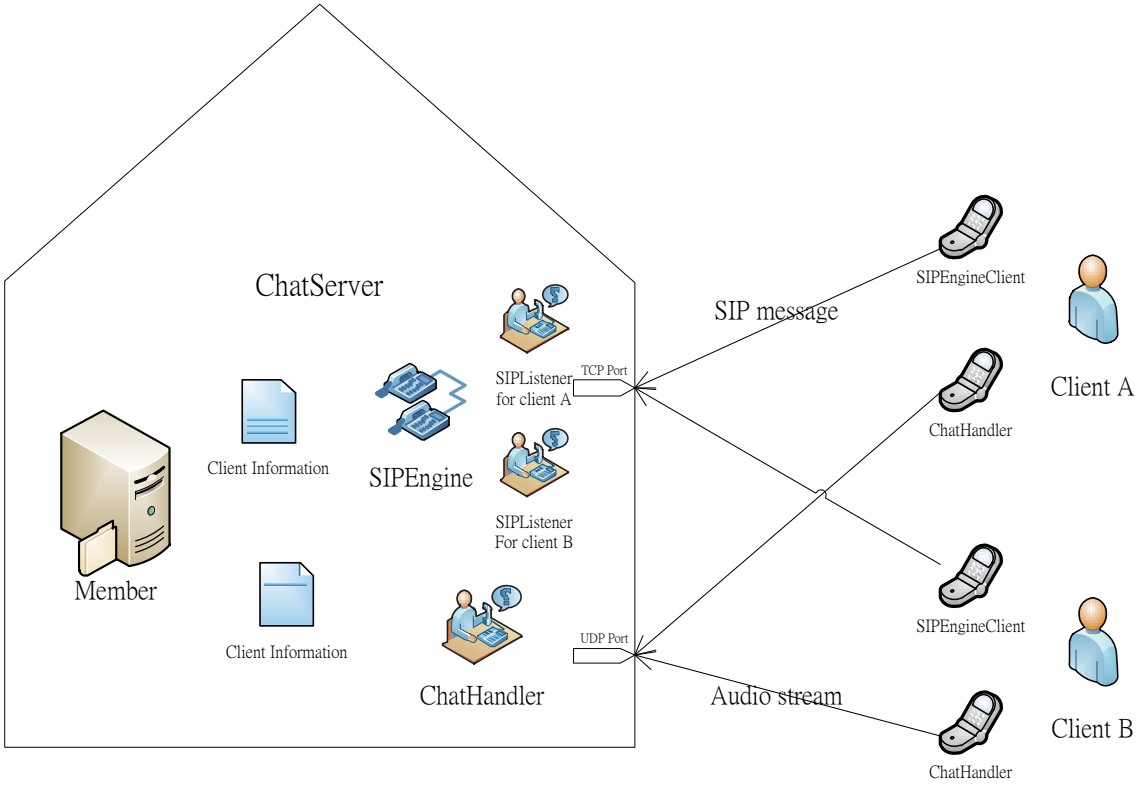


Figure 9: System architecture of different modules

In order to make every experiment more consistent, smart phone screens are turned on while testing. The battery life was scaled to 100 levels, with each level representing one percent of battery life. The experiment measured the elapse time between battery levels of 90% and 85%. Smart phones B, C, D, and E from Table 2 were used as experimental devices. Each smart phone measure was taken three times in three different conditions, respectively. Based on time consumption problem, sample times are three for every condition. Results and discussion are highlighted in the next chapter.

Delay Testing

A conferencing system can be classified as a real-time multimedia system and it

depends on audience perception of the delay of audio stream. An audio event with a huge delay time would result in low system accessibility and usability. In order to test the system's performance, we implemented a delay testing experiment to test the delay time for a specific audio packet. A low delay result represents high quality in a real-time conferencing system. In contrast, a high delay audio stream may cause a negative impression to users.

Table 2

Experimental Smart Phones Specification

Number	Model Number	CPU	RAM	Android Version
A	Xperia Play	1 GHz Scorpion ARMv7 processor	512MB	2.3.4
B	DROID2	ARMv7 Processor rev 2 (V7I)	512MB	2.3.7
C	DROID2	ARMv7 Processor rev 2 (V7I)	512MB	2.3.7
D	SCHI500	Samsung-Intrinsity S5PC110 RISC Application Processor	512MB	2.3.4
E	SCHI500	Samsung-Intrinsity S5PC110 RISC Application Processor	512MB	2.1-update1

In this system implementation, an analog signal is captured by the smart phone's microphone. The resulting audio data is stored in a buffer and the system waits for the application to read the data. The program reads a specific size of audio data from the buffer, places the data in a packet, and sends the packet to the server. The server forwards the packet to the target smart phone. Once the target smart phone receives the data packet, it extracts the data from the buffer and writes into the audio track. The digital signal is then converted to analog sound and played through the phone's speaker.

This experiment measures the elapsed time of a packet between two events: the data read from the buffer of microphone and the same data being received by the server.

A specific header was added to every packet for testing. The header was 5 bytes long, includes 1 byte for the device number and 4 bytes for a time stamp value. The elapsed time was calculated as the current system time minus the time stamp's value.

Table 3

Experimental router

Model Name	Standards	Speed
NETGEAR Wireless-G RouterWGR614	IEEE 802.11 b/g	up to 54 Mbps

All tests were measured under two different environments: single source audio stream network and multi-sources audio streams. The single source audio stream network transfers only the tested subject's audio stream. In contrast, all devices try to transfer audio data at the same time in multi-sources audio stream network. In general, once the server receives a data packet, it forwards the packet to every participated client, excluding the packet sender. The delay testing experiment sent every packet to the tested smart phone, whether if it was the packet's owner or not. In addition, the tested subject was last in order of the server's forwarding targets.

Smart phone A is the test subject in this experiment. Smart phone B, C, D, and E are participants in the testing network. For single source audio stream network, 100 and 1000 packets are collected and measured, respectively. For multi-source audio stream network, 100 packets are collected and measured. The sample time was set to 5 for each condition. Results and related discussion is highlighted in the next chapter.

Chapter 4: Results and Lessons Learned

Battery Consumption

Figure 10 and Appendix E show the results of battery consumption under different conditions. The "change" column in Appendix E.2 and E.3 represents the results as compared with Appendix E.1 in percentage terms.

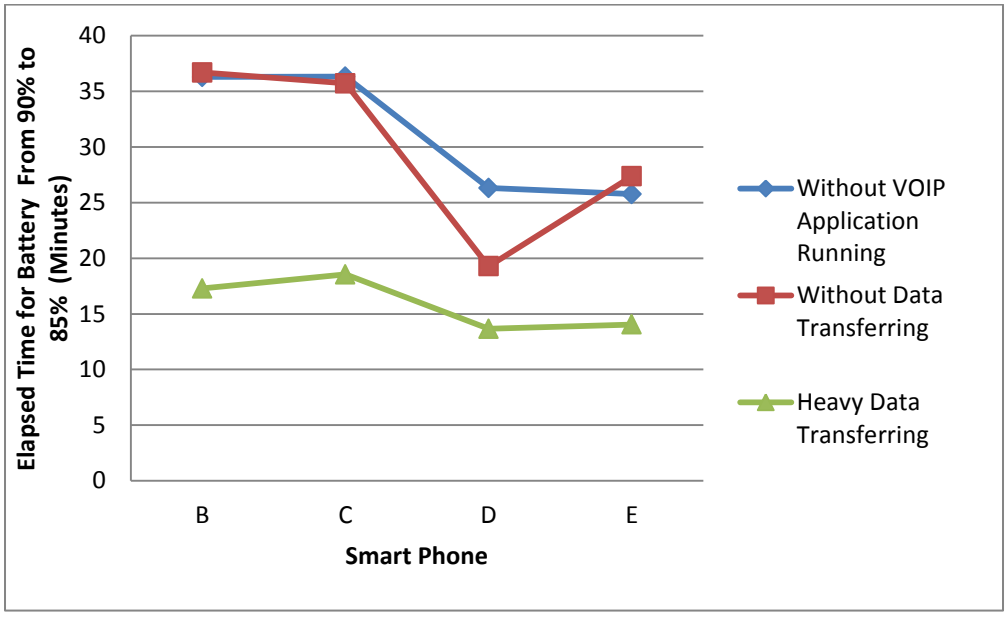


Figure 10: Elapsed time to consume 5 percent of battery life

In the condition of "Without Data Transferring", it is observed that it has similar performance to the condition where the VOIP application is not running. This result explains that the system has low battery usage without any audio data transferring. When there are no audio packets being sent between the server and clients, the client program only maintains a TCP connection and a UDP port waiting for incoming packets. The result shows that any participant in a "quiet" audio conferencing system is not affected much from a power consumption aspect.

Under the condition of heavy data traffic, the client system on the smart phone

requires heavy workload of processing audio data and interacting with the speaker and microphone. The result shows the elapsed time to consume five percent of the battery life decreases in half (-48.7152% on average), or more than twice the battery usage than is normal. If every level has same power intensity, then the smart phone can run the VOIP application for 5.29 hours (on average).

For a conference event which requires long hours of participation and a high number of clients interacting, the system would have low accessibility (or high power consumption). In this environment, the system provides an online streaming audio recorder and audio player. From appendix E.2, if the recorder and player are not functioning, it has similar performance with the first case. In other words, by decreasing the usage we can improve the overall performance of battery life usage. A possible solution to improve the system includes reducing the amount of received data and implementing a push-to-talk technique.

Delay Testing

The purpose of this experiment was to test the system's multi-user processing capability. The test measures the additional delay when a new participant joins the conference room. If the delay grows in a large range or grows exponentially, the system becomes less accessible when the number of users increases. Delay testing results are shown in Figures 11, 12, and 13.

Under all conditions, results show a tendency of increasing delay with a growing number of participants. Under a single source audio stream network, only the test subject's audio stream can be transferred through network. From experimental results, we can see that the maximum amount of additional delay for a new participant is 4.112 milliseconds for 100 packets sample size. For 1000 packets sample size, the maximum

delay time for a new joining client is 198.776 milliseconds.

To add a new member into the member list requires an additional target for the server to handle. The new overhead to add a new member does not grow exponentially, as it only adds a few milliseconds. Under multi-sources audio streams network, the delay time grows to more than 2.5 seconds. The overhead of having a new participant is 381.542 milliseconds (on average). This result illustrates that if more clients try to talk in the same time, a significant delay occurs in the system.

Once the sample size is increased to 1000 packets, the delay time grows more than twenty times the 100 packet cases. This significant change is caused by system data processing speed. In the implemented system, both the client and server program implement FIFO as the audio data processing policy. If the packet receiver's reading speed is not fast enough, more and more incoming data will stays in the buffer waiting for reading. The waiting time increases if the output speed is slower than input speed. The data collecting time for 1000 packets is greater than 100 packets. By comparing the result of 100 packets and 1000 packets, it shows that as the collecting time is increasing, the delay for individual packet is also increasing. The experiment of multi-sources audio stream network increases the data flows of traffic between clients and server, and results in greater amounts of incoming packets in unit time for the client. The increased delay illustrates that the client program does not have enough speed reading the incoming packets.

A design defect causes the reading speed problem. The program is designed to receive a packet and write the data from the packet into the audio track. These two events occur in order. That is, in order to receive a new packet, the thread has to wait the program calls the write() method to write the buffer to the track. Calling this method

causes additional overhead and decreases the speed of reading packets from the socket. The design of the program should call these two methods in different threads. The program should have a thread receiving packets and a thread writing the data to the buffer. By implementing these two threads, receiving data and writing track becomes two independent events and should increase the speed of reading data.

More detailed experiment results are shown in Appendix F.

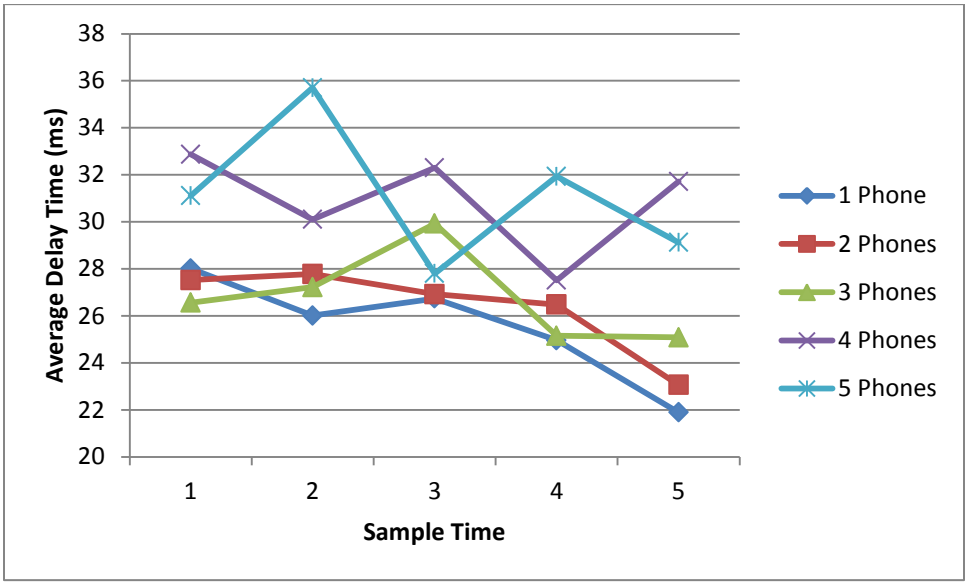


Figure 11: Average delay time of single source audio stream network (100 Packets)

Lesson Learned

From the architecture standpoint, a client program and a server program both needed to be developed and implemented for this project. In addition, both programs needed to have the capability to build connections and transfer audio streams. The sender of the stream must notify the receiver with the specification of the audio stream. These two requirements can be solved by implementing a signaling control protocol and a media transport protocol. In this implementation, we used the Session Initiation Protocol and Real-time Transport Protocol. We focused on building four elements: a centralized

conferencing server, a mobile phone client, classes which implement SIP, and classes which implement RTP.

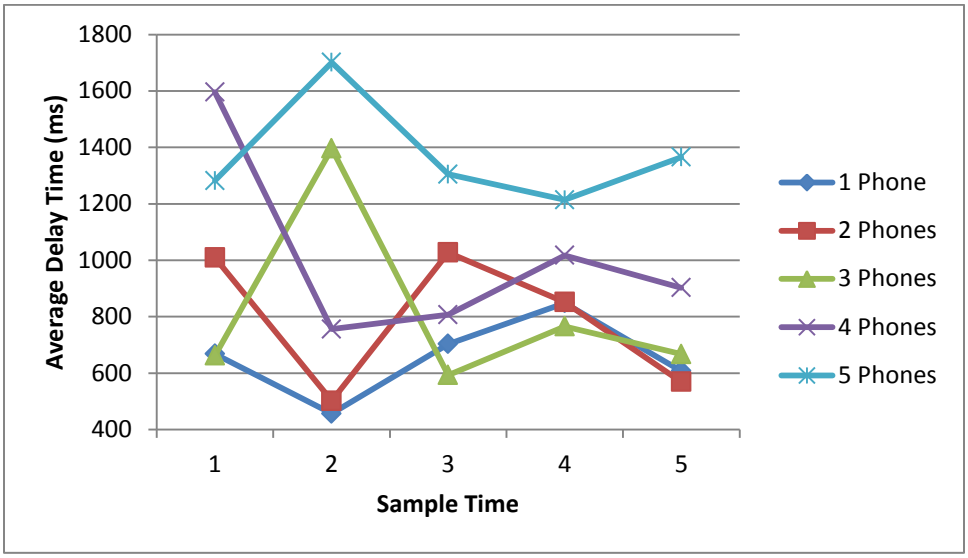


Figure 12: Average delay time of *single source audio stream network (1000 Packets)*

Different possible solutions are developed to achieve the goals of the project. The solutions were designed based on the Android and Java environments. Each solution has its advantage and disadvantages. Relative topics of problems and lessons learned are discussed in following sections.

Solutions and Relative Problems for Client Program

For client programs, three different solutions are proposed to solve the problems encountered. The first solution is using the library in android.net.sip package (API level 9). This package provides access to SIP functionality. By using the SIP API, it is easy to manage profiles, register to servers, make SIP calls, listen to SIP calls, and even transfer RTP streams after connection is built. It adds convenience of designing a client program without knowing any detailed information of SIP headers. Relative interface for acquiring server information and interacting with the user would have to be designed to complete the client program.

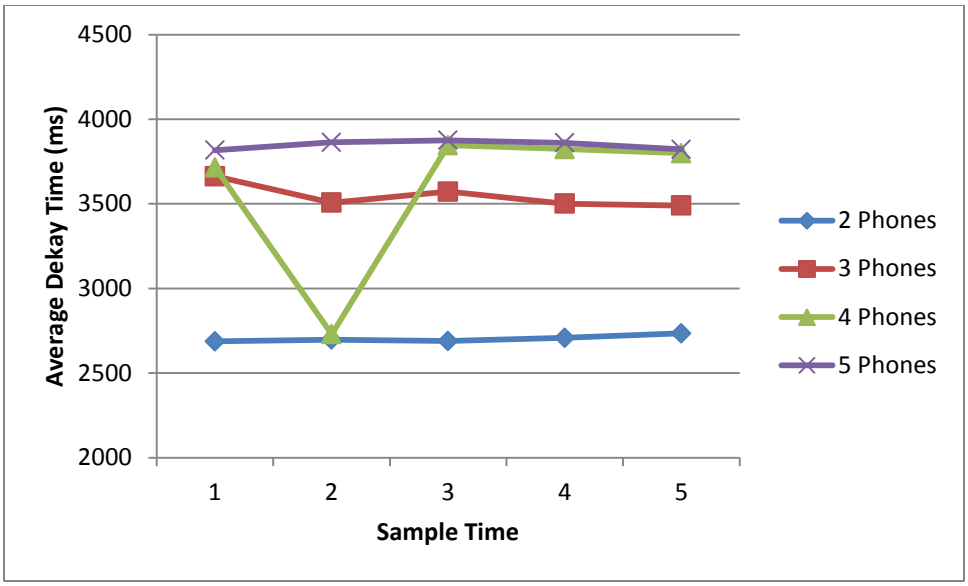


Figure 13: Average delay time of multi-sources audio stream network

There are two advantages of using the SIP API. Since the API follows the SIP standard, the message send from the machine can be easily accepted by other VOIP server which also uses SIP as its signaling control protocol. The client program of this project can therefore be designed to be compatible with any other SIP products. In other words, the server should strictly follow the SIP standards. Any misuse of headers can lead session connection problems. Another advantage is that the detailed information of audio stream transferring is hidden. The decision of audio channel, voice sample rate, voice sample size, and audio stream codec are all made in lower layer of Android architecture. The designer hence can ignore the process of designing audio I/O. The disadvantage is that it is more complicated to design an experiment with different audio configuration such as measuring the performance of different codecs.

The second solution involves using the customized SIP with the library in android.net.rtp package (API level 12). The customized SIP can omit some unnecessary headers for a SIP direct call. The provided API for RTP allows applications to manage

on-demand or interactive data streaming. In particular, applications that provide VOIP, push-to-talk, conferencing, and audio streaming can use these APIs to initiate sessions and transmit or receive data streams over available network. By using the RTP API, some configuration such as transferring codec can be set up. In addition, the class of android.net.rtp.AudioGroup provides an audio hub for the speaker, the microphone, and audio streams. It provides logical turned on or off for each of these components and audio mixing for all members in the audio group. It allows an application manages different audio stream in different groups. A full mesh distributing conferencing system (figure 3) can be designed by building connections to each individual devices and join them all in the same audio group.

Another problem involves Android operating system versions. Android is an open source operating system and the Android devices are very diverse. Different devices do not support the same API. An API level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform, and each device supports different API levels based on machine's system distribution version. Table 4 introduces different Android platform version and the supported minimum API level.

SIP stack requires minimum of API level 9 and RTP stack requires minimum of API level 11. Any smart phone's operating system under Android 2.3 cannot use the SIP stack in general approach (or Android 3.0.x for RTP stack). The choice of using these two stacks may lead the loss of market for lower level platform users. Because this project did not have enough resources to support the restricted APIs in the beginning of the project, using SIP and RTP were designed but not tested and implemented for the final system.

Table 4

The API Level Supported by Each Version of the Android Platform

Platform Version	API Level	VERSION_CODE
Android 4.0.3	15	ICE_CREAM_SANDWICH_MR1
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

From reference [18]

The third approach was to use available open source code, which was carefully considered for the project. There are several VOIP applications on the market and some of them are open source. Studying and researching their code and documents carefully could be very beneficial. Although this approach increases the complexity of developing a system, it also increases flexibility. The system programmer would need to handle data read from the machine, the buffer use for microphone and speaker, and the packet transferring through the internet. In addition, the programmer can import third party libraries. There are several libraries out there for SIP, RTP and audio signal processing. Making use of different libraries can meet the same requirements of using Android framework APIs.

Some problems result from using open source software as well. First, some projects do not have well organized documentation. Lack of good documentation decreases the readability of a large complex system. Second, most audio signal processing libraries are native codes. Using native code does not result in an automatic performance increase. A large corpus of C/C++ code increases application complexity. Additional time is also required to research how to use the native code and how to embed the native code into an application package file by using Android NDK toolset.

Hardware diversification naturally increases programming difficulty. Different parameters results in different output based on different models of smart phones. A higher sample rate for a PCM 16 bit mono audio configuration on older smart phones may lead bad sampling errors. The minimum buffer size required for the relative Android audio record object (android.media.AudioRecord) may also differ. Even if the configurations are all the same, the sound quality of different smart phones are different. Because of the problem of diversification, a usable configuration for all devices is very limited. Testing the audio parameters on different devices is required, and therefore increases the amount of effort spent developing and testing code.

Solutions and Relative Problems for Server Program

The project implemented a centralized conferencing model. Whenever the server receives an audio packet from a client, it determines packet's targets and forwards the packet to its targets. The fundamental concept of the server is simple, but there is a critical problem needs to be concerned with. If a server does not have any mechanism of audio mixing, then the audio stream cannot be send concurrently. For instance, if there are three different audio stream sends from different clients, each of the streams is comprised of five packets ($A_1 \sim A_5, B_1 \sim B_5, and\ C_1 \sim C_5$). Each packet includes audio data

which can be played for n milliseconds. The server then determine client Das these three streams final destination. In this case, client D will receive $5 \times 3 = 15$ packets. As a result, client D needs at least 15n milliseconds to play all the data it received. If each stream was sent from its source concurrently, then in theory, client D should only need 5 milliseconds to play the received audio data. As more and more clients participate in transferring audio streams, this phenomenon grows more severe. From the *Delay Testing* section, we saw that if multi-sources packets are transferred concurrently, the client program will have a significant delay. If the server can mix different packets and reduce the overall amount of transferring sending packets, the performance of delay can be considerably improved. Careful design and finding an adequate solution for audio mixing are necessary requirements to improve overall system performance.

System Testing

The project adopted a simplified incremental development approach. For every requirement, analysis and design should be done before implementing prototype. System integration happened if the latest developing module cannot be tested individually. System testing occurs before next requirement's analysis and design proceed. If problems are encountered during testing phase, additional requirements would be generated. The unsolved problems are labeled "unsolved" and become a new requirement.

The project used trial and error to test the system and resolve system failures. After implementing a module, if the testing module or system involves user interface, simulation of different user activities would be implemented as testing parameters. If the module involves smart phones, deploy the system in different devices and observe the performance after executing the system. If a failure occurs, debugging tools are used for tracing bugs and solving problems. Debugging tools included Wireshark for monitoring

the server's network traffic and Dalvik Debug Monitor Server (DDMS) for smart phone debuggers.

System Performance

The system has good performance when only one client is speaking at a time. From a human perception stand point, the sound is fluent and continues uninterrupted. Smart phones are designed for communications, hence the recording device, which is the smart phone's microphone, should meet a minimum recording capability as the audio quality is good. Because of the implementation of SIP, the server can clearly understand the information from clients, and understand who is in the conference room. From storage perspective, the client program requires about 316 kilobytes. The system's minimum required API level is 3; hence compatible with most Android smart phones.

While multiple clients speak concurrently, audio streams are broken into fragments. The problem becomes more serious if the system has more clients in the conference room. Human language can still be recognized, but background noise such as music is hard to understand. Because the program does not handle with sound noise and echo problem, headphone is recommended to use while running the system. In addition, the system is seriously affected by the internet environment. Ports availability and router's signal's stability also affect the system's accessibility.

Project Improvement

From the developer's experience, several suggestions are proposed for improvement. Suggestions are made to improve the system performance and solve the problems which occur while developing.

Resource management is an important component of a project. As the problems illustrated in the *Solutions and Relative Problems for Client Program* section, some

solution requires a minimum operating system's platform level. In this project, only one machine satisfies the minimum API level. The lack of resources causes the difficulty of developing and testing the system. Those solutions were both frozen after understanding it is impossible to have enough machines to deploy and test the conferencing system. Before any development starts, a developer should not only focus on what resource possesses the largest market, but also requires him/her to examine what resources they currently have. Since the project did not have a well-planned process of resource management, additional time was spent determining possible solution's approaches and/or to acquire enough machines which met the minimum API level.

Development difficulties caused by the smart phone's diversity also led to some critical errors. A configuration performs well in one smart phone model may not perform well in other models. When this problem occurs, debugging complexity increases, and additional research is required to figure out if it was hardware/software supported problem or a programming error.

Audio signal processing experience would also have helped greatly during the design of the audio conferencing system. Additional experience could be gained from doing more research, including researching open source projects that involve audio processing, working in a multimedia lab, and/or cooperating with other developers. All of these experiences would help to clarify the problems as well as to determine the direction of effective solutions. For instance, different audio data arrival times causes problems, therefore how to synchronize audio data from different sources becomes one important requirement for projects such as this.

In conclusion, to improve upon the audio conferencing project, three additional areas should have been the focus of more study at the beginning of the project. First,

more time should have been spent to examine and evaluate the available smart phone hardware and software that was available. Second, a particular hardware platform should have been chosen as the standard developing platform. For example, one should choose a smart phone model which was most prevalent in their inventory. An alternative solution would be to select a smart phone model with less diverse set of operating systems, such as the iPhone. The advantage of using iPhone is that it has less hardware and software variation. Third, developers should have considerable audio processing experience before starting a project that deals with audio processing. Understanding how to mix multiple audio streams can help the programmer to decide whether to develop the algorithm from scratch and/or use code from an existing library.

Chapter 5: Conclusions and Future Work

This paper introduces an approach to deploying audio conferencing systems in smart phones and implements a VOIP application. The project selects approaches from numerous sources and integrates them together. Experiments were designed to measure the usability of the system and solutions and related issues are discussed providing potential approaches to improve the project.

This paper's contributions are as follows:

- Introduce essential VOIP protocols
- Illustrate system architectures
- Evaluate system performance
- Expose issues/obstacles and present solutions

In order to build the audio conferencing system, this project adopts Session Initiation Protocol (SIP) to initiate and manage the connections. To minimize the work load in smart phones, the system chooses centralized conferencing model as its fundamental system architecture. Message flows and audio processing decision are made by the server. The smart phone's client program mainly focuses on sending SIP request, playing the received audio streams and sending the recorded audio data.

When designing a real-time smart phone application, two elements need to be carefully considered. Battery consumption determines the systems use of time and affects users willing to use the system. The designed experiments show both the consuming speed of a busy network environment and the speed of an idle network environment. Experiment results illustrate the limitation of using a small battery life in a smart phone for a high interactive conferencing environment.

Delay time of audio stream is another important element. If only a single source

audio stream is transferred at a time, sounds are consequent and the quality does not decay over time. When the system has a busy and complicated traffic, sounds are broken into fragments and are hard to be recognized.

Hardware diversification creates an obstacle and expands the developing life time. Resource management is necessary started in early phase of the project. Throughout the course of this project, solutions were abandoned because of the lack of resources. Different models of smart phone have different performance of same codes. Every prototype of modules should be tested in different models of smart phones. The testing time increases and parameters are adjusted frequently.

A system architecture was chosen and a timeline for software development activities was established. In practice, each module exceeded the expected development time due to numerous encountered problems. Problems such as finding a balance setting of all smart phones needed to be solved as well as problems such as sound mixing.

In future work, audio processing techniques will be designed and integrated into the system and a different codec will be adopted to accelerate the data transferring rate. Some modules will be redesigned to improve the system's performance. In addition, new system architecture of different modules will be designed and implemented (figure 14). Instead of having only one audio handler and one port for all clients, the system creates an identical port and handler for every client. This architecture allows the server to read packets from different clients concurrently. An audio mixer module will also be designed to synchronized different streams. More sophisticated conferencing system will be designed and be available for academic, personal, and business use.

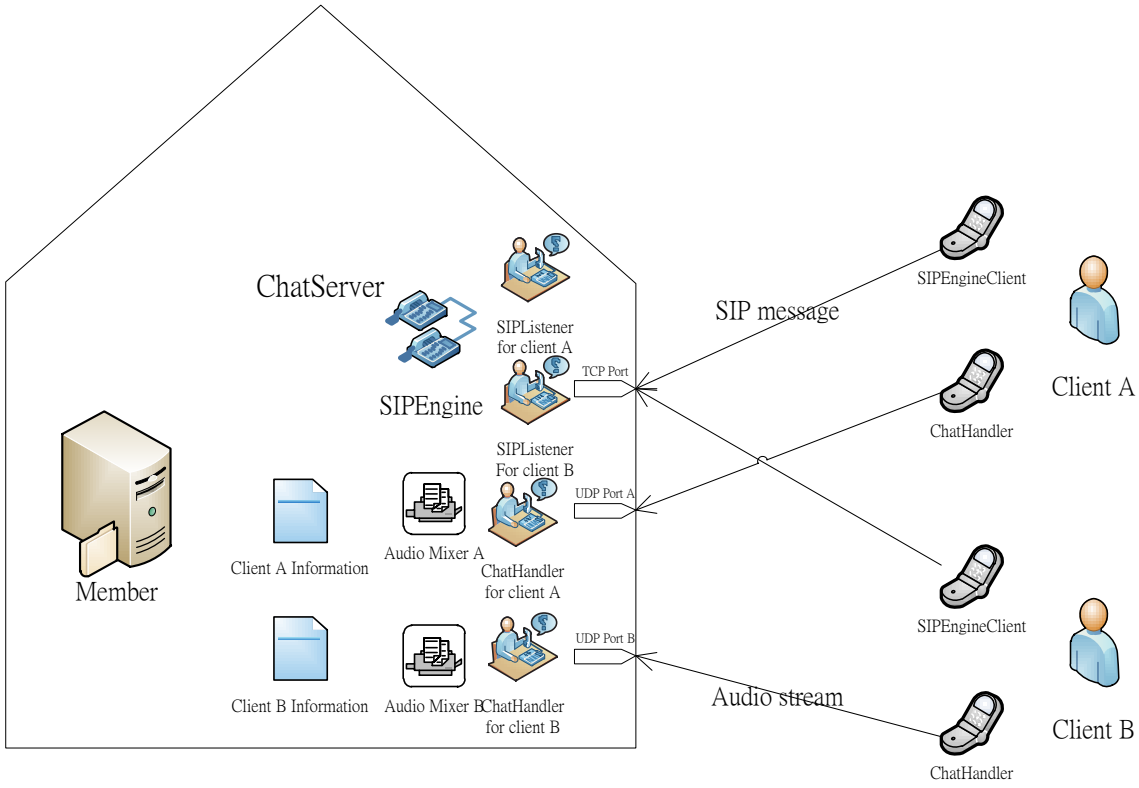


Figure 14: Future system architecture of different modules

References

[1] W.K. Jia, “Session-Initiation Protocol Methodology Handbook Second Edition”, Kings Information, Taipei, TW, 2008.

[2] “Overview of SIP”, http://www.cisco.com/en/US/docs/ios/12_3/sip/configuration/guide/chapter0.html, CISCO.

[3] T. KOISTINEN, “Protocol overview: RTP and RTCP”, Nokia Telecommunications, 2000.

[4] “Introduction to Voice over IP (VOIP)”, http://www.tech-pro.net/voice-over-ip.html, Tech-Pro.net, 2012.

[5] K. Singh, H. Schulzrinne, “Peer-to-peer internet telephony using SIP”, Proceedings of the international workshop on Network and operating systems support for digital audio and video, ACM, June 2005.

[6] M. Naeem, V. Namboodiri, and R. Pendse, “Energy implication of various VOIP codecs in portable devices”, In Local Computer Networks (LCN), 2010 IEEE 35th Conference on, pages 196-199, oct. 2010.

[7] Y. Agarwal , R. Chandra , A. Wolman , P. Bahl , K. Chin , and R. Gupta, “Wireless wakeups revisited: energy management for voip over wi-fi smartphones”, Proceedings of the 5th international conference on Mobile systems, applications and services, June 11-13, 2007, San Juan, Puerto Rico.

[8] J. Light, “Performance Analysis of Audio Codecs over Real-Time Transmission Protocol (RTP) for Voice Services over Internet Protocol”. Computing & Informatics, 2006. ICOCI '06. International Conference on, pages 1-8, IEEE, June 2006.

[9] H. L. Cycon , T. C. Schmidt , G. Hege , M. Waahlisch , D. Marpe , and M. Palkow, “Peer-to-peer videoconferencing with H.264 software codec for mobiles”, Proceedings of the 2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks, p.1-6, June 23-26, 2008

[10] H. Pereira, M. Curado, and P. Carvalho, “QoS in Real Time Data Transmission”, Department of Informatics Engineering, University of Coimbra, 2005

[11] Wu, W., et al., “Service Oriented Architecture for VoIP conferencing”, International Journal of Communication Systems, 2006.

[12] J.H. Dai and L.F. Jiang, “Principle and Application of VOIP SIP”, Scholars Books Co, Ltd, Taipei, TW, 2005.

[13] C, Ling, “Android Application Development Third Edition”, GOTOP Information Inc., Taipei, TW, 2011.

[14] “Codec”, http://en.wikipedia.org/wiki/Codec, Wikipedia.

[15] “Audio codec”, http://en.wikipedia.org/wiki/Audio_codec, Wikipedia.

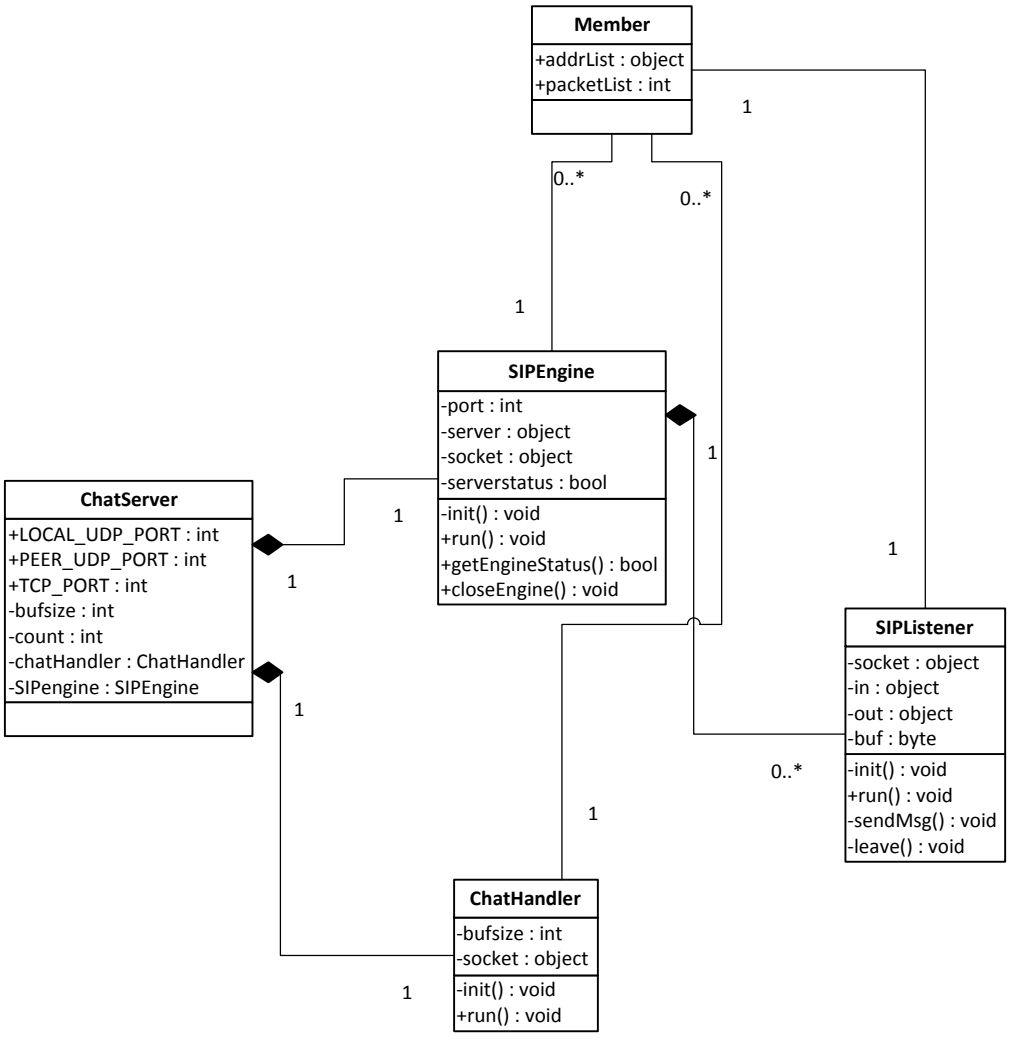
[16] “Speech coding”, http://en.wikipedia.org/wiki/Speech_coding, Wikipedia.

[17] A. Watson, “Evaluating real-time multimedia audio and video quality”, 1997.

[18] “Android API levels”, http://developer.android.com/guide/appendix/api-levels.html#level12, Android Developers.

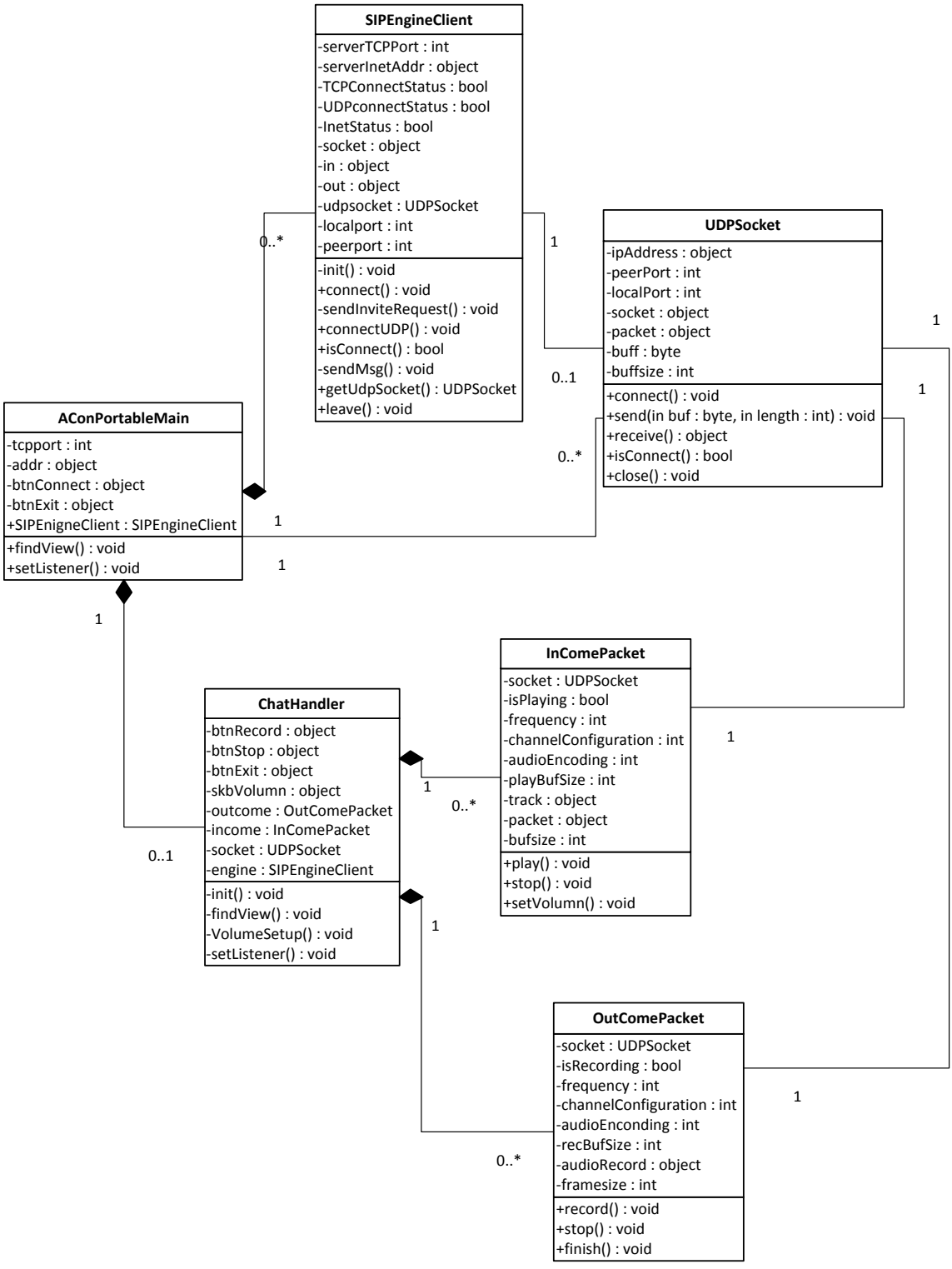
Appendix A

Class Diagram (Server)



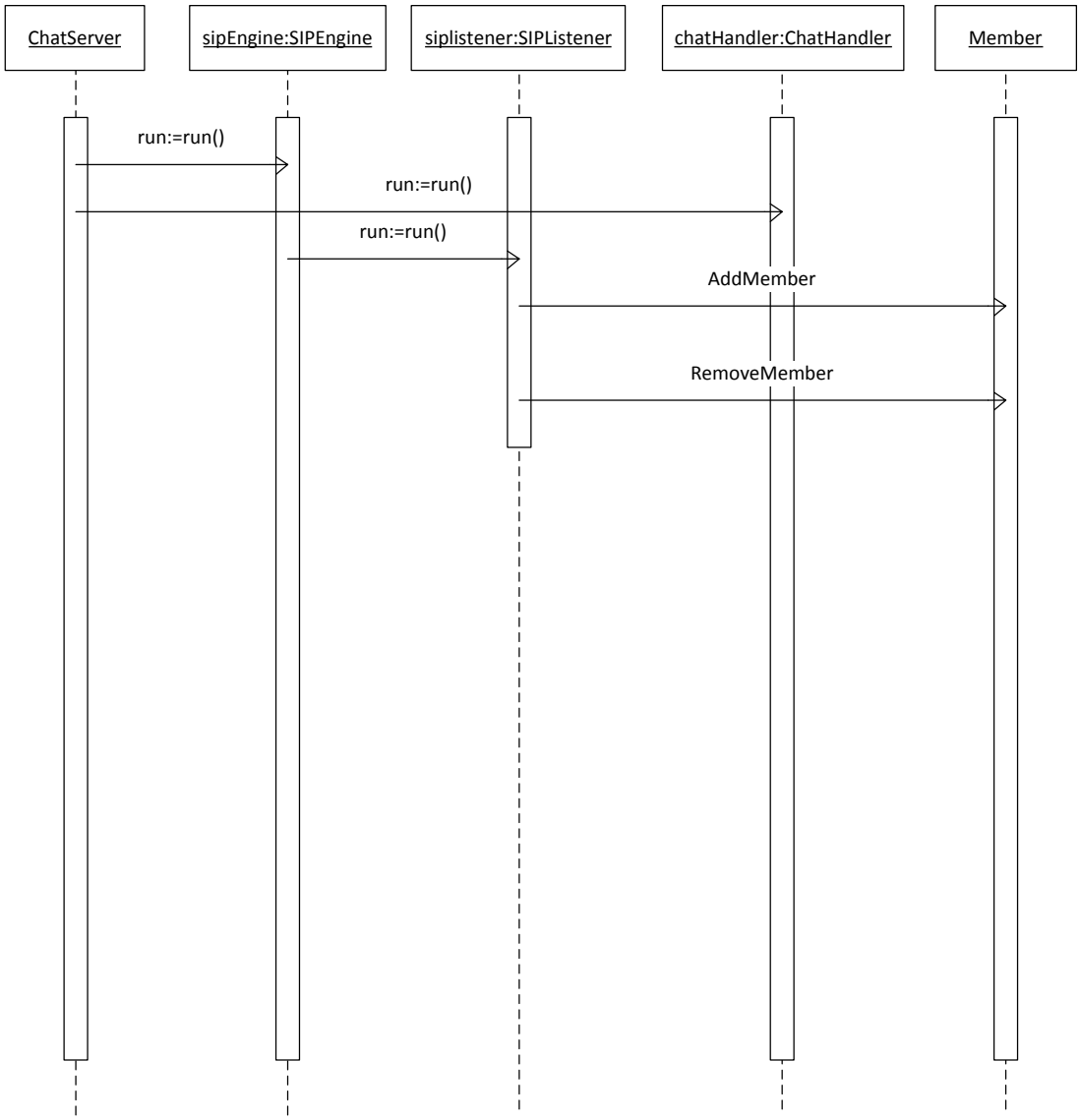
Appendix B

Class Diagram (Client)



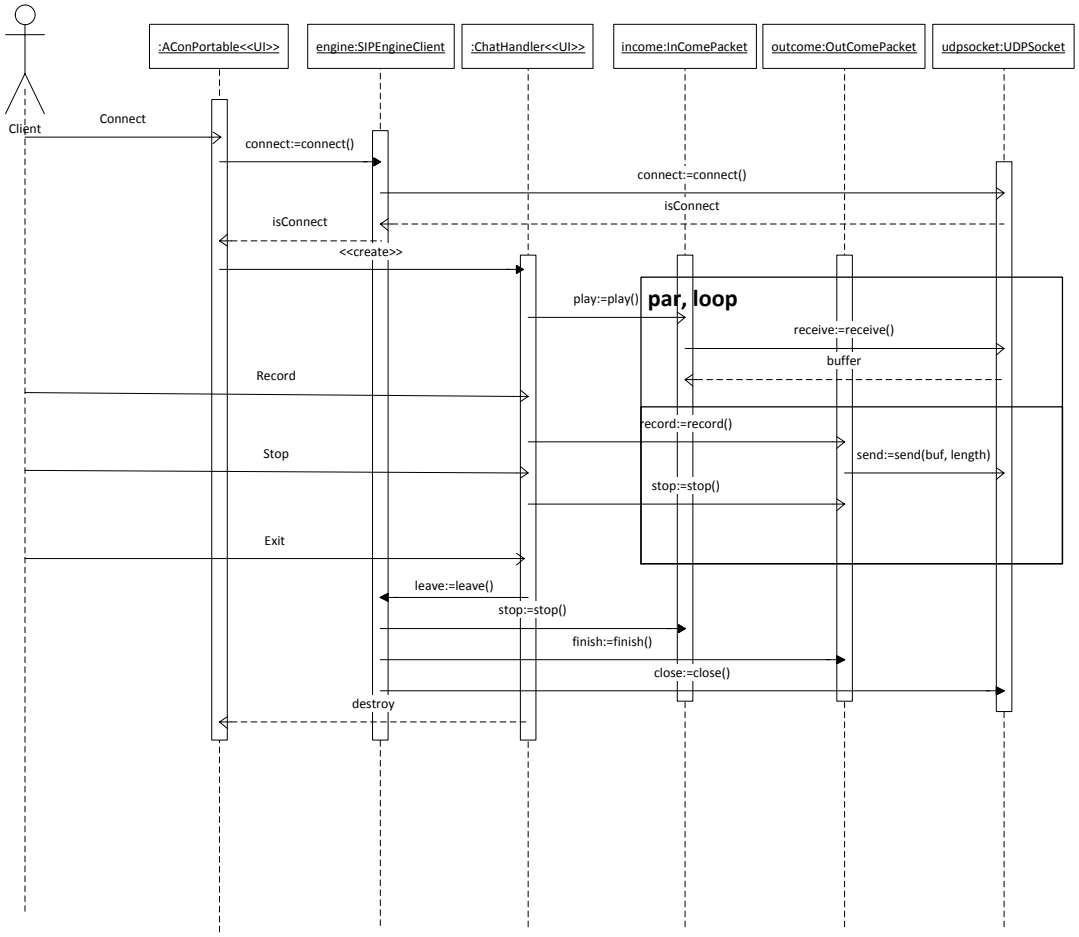
Appendix C

Sequence Diagram (Server)



Appendix D

Sequence Diagram (Client)



Appendix E

Power Consumption Experiment Result

Smart Phone \ Sample Time	1	2	3	Average Time
B	38.12995	38.15978	32.57527	36.28833
C	38.78988	37.65495	32.5057	36.31684
D	27.09058	30.44268	21.39608	26.30978
E	32.02523	17.1045	28.16995	25.76656

E.1 *Consumption time without VOIP application running (Minutes)*

Smart Phone \ Sample Time	1	2	3	Average	Change (%)
B	36.92553	36.51733	36.64685	36.69657	+1.124987
C	36.28982	36.44618	34.43167	35.72256	-1.6364
D	20.4067	18.3972	19.06218	19.28869	-26.6862
E	25.03367	27.16705	29.9	27.36691	+6.210935

E.2 *Consumption time without data transferring (Minutes)*

Smart Phone \ Sample Time	1	2	3	Average	Change (%)
B	19.77675	16.04188	16.0418	17.28681	-52.3626
C	18.7033	18.87768	18.05847	18.54648	-48.9315
D	13.03647	13.89308	14.05847	13.66267	-48.07
E	19.36782	2.029467	20.73355	14.04361	-45.4968

E.3 *Consumption time of heavy data transferring network (Minutes)*

Appendix F

Delay Testing Result

Smart Phones / Sample Times	1	2	3	4	5
1	28.01	27.52	26.56	32.87	31.11
2	26.01	27.78	27.22	30.1	35.7
3	26.74	26.92	29.92	32.3	27.8
4	24.96	26.48	25.15	27.51	31.93
5	21.89	23.06	25.08	31.71	29.12
Average Elapsed Time	25.522	26.352	26.786	30.898	31.132

F.1 S*ingle source audio stream network (100 Packets)*

Smart Phones / Sample Times	1	2	3	4	5
1	667.86	1009.11	662.1	1595.71	1282.04
2	456.43	502.04	1396.23	755.84	1701.36
3	703.38	1027.7	593.12	806.77	1304.78
4	848.05	851.61	765.73	1017.14	1214.2
5	608.12	569.42	667.22	902.82	1365.16
Average Elapsed Time	656.768	791.976	816.88	1015.656	1373.508

F.2 S*ingle source audio stream network (1000 Packets)*

Smart Phones / Sample Times	1	2	3	4	5
1	28.01	2686.73	3662.44	3715.51	3817
2	26.01	2697.57	3506.82	2729.15	3863.35
3	26.74	2688.82	3572.08	3846.2	3875.5
4	24.96	2707.65	3500.95	3823.44	3860.06
5	21.89	2734.09	3489.86	3798.67	3822.08
Average Elapsed Time	25.522	2702.972	3546.43	3582.594	3847.598

F.3 Multi-*sources audio stream network (100 Packets)*