Creating an Audio Conferencing Application on Android Smart Phones

Jui Sun
js9207@uncw.edu

Ron Vetter
vetterr@uncw.edu
Department of Computer Science

Bryan Reinicke
reinickeb@uncw.edu
Information Systems and Operations Management

University of North Carolina Wilmington
Wilmington, NC 28403, USA

Abstract

This paper describes an approach to building an audio conferencing application for Android smart phones. As the need for audio conferencing systems grows and smart phone market penetration has increased, the smart phone has become 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 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 also 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, specifically on the Android platform, and can direct future development of audio conferencing systems.

Keywords: Android Development, Mobile Development, VOIP, client-server architecture

1. INTRODUCTION

Voice over IP (VOIP) audio conferencing systems are increasingly becoming an important application on the Internet (Freese, 2005). VOIP introduces a possible low cost solution for long distance multi-people communication problems (Jaiswal and Raghav, 2004). As the need for voice conferencing systems continues to grow, these systems are being applied to many areas of business, as well as in academic and social circles (Gilson and Xia, 2007). VOIP systems are gaining more acceptance as the software and the quality of service and surrounding network environment improves (Park, 2010). A highly attractive scenario combines VoIP with the

expanding use of smartphones (comScore, 2012), and would allow users to participate in a conference meeting, without having to physically be present or incurring charges for the minutes used on their cell phones.

A smart phone is a portable handheld device with the capability of a personal computer and traditional cell phone rolled into one. 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 a VOIP 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 paper is to explore how a simple and extensible audio conferencing system for smart phones can be designed and implemented. The 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. Section 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 section 3. Section 4 discusses the experimental design, and section 5 discusses the results and lessons learned. Finally, section 6 provides conclusions, and discusses future work that could improve audio conferencing systems on smart phones.

2. BACKGROUND AND RELATED WORK

Voice over IP (VOIP) was first introduced in 1991 when Speak Freely developed internet-based telephony software for the personal computer (Tech-Pro, 2012). In 1996, the ITU Telecommunication Standardization Sector (ITU-T) defined the first version of the H.323 standard (International Telecommunications Union, 1996). Because the Internet was a bandwidth constrained environment, 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, which convinced users of the capability and possibility of internet telephony (Jia, 2008).

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 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 (Jia, 2008). In addition, management protocols and other types of support protocols are also used in VOIP applications.

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 (CISCO, 2012) using 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 (Jia, 2008). 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 (CISCO, 2012):

- 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.

User Agent Clients and Servers

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 able to act as either a UAC or UAS (Jia, 2008). 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 (Jia, 2008; CISCO, 2012).

Figure 1 can be found in Appendix A, and 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 UAC and Mary's machine is a UAS. John's machine calls the target via 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, the 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.

Codecs

A Codec is the method used to encode and decode a digital stream or signal, and there are several in widespread use in multimedia (Isnardi, Fielder, Goldman and Todd, 2006). One of the first things that needed to be determined was which Codec would be used to encode and transmit voice data for this project. In general, Codecs can be defined as lossless or lossy (Wikipedia, 2012a; Wikipedia, 2012b). Lossless codecs try to maintain the original audio information, while lossy codecs trade some information to achieve other requirements. There are a number of different Codecs available which can provide toll quality speech under real-time transmission (Light, 2006), which can be seen in table 1 in Appendix A. All of these were available prior to the release of Android OS version 2.3. It should be noted that while, in many respects, codecs for speech present a simpler signal than other audio codecs (Kroon, 1995), this does not make them simple.

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 grouped as "Centralized Conferencing" and "Distributed Conferencing" (Jia, 2008). Centralized conferencing (Figure 2, Appendix A) 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.

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, a smart phone relies on its mobile battery. If an application is a burden on the phone's battery, it decreases the time for voice calling.

While a VOIP system communicates through a Wi-Fi network, the associate interface is active. Energy is consumed even if no data is transferred. When an application is running, program size, algorithms, and other programming factors influence battery consumption. As a software developer, it is impossible to increase the battery size on a mobile device, thus requiring this to be managed via software. To address this problem, Agarwal, Chandra et al. presented a wakeup mechanism to solve the waste of energy by system idling

(Agarwal, Chandra, Wolman, Bahl, Chin, Gupta, 2007) while Naeem et al proposed an adaptive algorithm to switch codecs based on remaining battery life (Naeem, Namboodiri, Pensi, 2010).

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 developed can be run.

3. METHODS AND SYSTEMS ARCHITECTURE

This project was developed using the Eclipse IDE and included the development of two software components.

The first component was the user client, which was deployed on several Android based smart phones (see table 2 in Appendix A). The program acts as a caller, or UAC. It established the connection by SIP and can:

- Send the SIP request.
- Establish and maintain a connection to the server.
- 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 and, as a result, cannot act a UAS.

The second component was the focus server. A focus server is the central node of the conferencing network. All clients transfer the audio streams through this central node. The server has the following capabilities:

- Receive and respond to SIP requests.
- Establish and maintain the connection between clients and itself.
- Manage participants.
- Clarify the incoming audio stream.
- Send an audio stream to the correct endpoints.
- Receive the client termination request, and disconnect clients.

We did not implement the focus server on a smart phone due to energy consumption concerns. Rather, the focus server was deployed on a personal computer running in a Java environment.

System Architecture

This system implemented a two-tier client-server architecture (shown in Figure 3, Appendix A). Clients communicate with the server through the Internet and 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: listens to client calls. Once it receives a client SIP call, it creates a specific SIPListener instances for the client and waits for the next SIP call.
- SIPListener: 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 which stores all conference participants' information.
- ChatHandler: receives audio data from the connected clients list and determines which target it should forward to.

The client includes the following modules:

- AConPortableMain: this generates the graphical user interface (GUI). This GUI asks users for the server's information and then creates a SIPEngineClient instance for further actions.
- SIPEngineClient: 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: handles audio data transferring. It includes the method to communicate audio streams with the server.
- ChatHandler: the part of the GUI which allows users to talk to the server. The 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 the 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 to await play back.
- OutComePacket: a thread class which reads the data from the microphone's buffer and sends the data to the server.

From a model standpoint, the system implements a centralized conferencing model, which is comprised of a server and all clients (participants). Figure 4 (Appendix A) introduces the model and the 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 5, 6, and 7 (Appendix A) introduce the actual mechanism for joining and leaving the conference system.

Figure 5 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 6 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 7 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 8 shows the architecture of the system's modules. The SIP message is transferred through the TCP port, and the server creates different SIPListener objects for every connected client. Audio streams are transferred through the UDP port. One object of the server's ChatHandler is created to handle all audio stream traffic. The Server's ChatHandler uses a First-in-First-out (FIFO) algorithm to forward incoming packets. Figure 8 also shows how the modules associate with one another.

4. EXPERIMENTAL DESIGN

In order to test this application environment, two tests were carried out. One to test the applications ability to transmit data and the time lag associated with this transmission and a second to test the impact of the application on the mobile devices batteries.
Table 2 and Figure 4 illustrate the clients and server for the experiment. The "Device" column in Table 2 distinguishes the different smart phones, which are labeled correspondingly in figure 3. This representation will be used when describing both the experiments and the results.

For the experiment we used the same audio settings in both power consumption and delay testing. The client program read 1024 Bytes from the microphone's buffer and sent it to the server. The audio data was configured as follows:

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

Power Consumption

In order to evaluate the usability of the conferencing system, an experiment evaluating power consumption was designed. Measurements were taken to determine how long it took the system to decrease the smart phones' battery life from 90% to 85%. This increment was chosen as a sample as a matter of practicality to meet time constraints. In order to make every experiment more consistent, smart phone screens were turned on while testing.

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.

Smart phones B, C, D, and E from Table 2 were used as experimental devices. Each smart phone measure was taken three times in the three different conditions. Smart phone A was not used in these tests, as it was the primary phone for one of the authors.

Delay Testing

A conferencing system can be classified as a real-time multimedia system, and as such the perceived quality of the system depends in part on audience perception of audio delay. An audio event with a huge delay would result in low system accessibility and usability. In order to test the system's performance, we implemented a delay testing experiment to determine the delay time for a specific audio packet.

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 amount of audio data from the buffer (in this case 1024 bytes), 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 it to the audio track. The digital signal is then converted to analog sound and played through the phone's speaker.

This experiment measured the elapsed time of the packet between two events: the data read from the buffer of the microphone and the same data being received by the server. A specific 5 byte header was added to every packet for testing, including 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.

Single vs. Multiple Streams

All tests were measured under two different environments: single source audio stream and multi-sources audio streams. The single source audio stream transfers only the tested subject's audio stream. In contrast, all devices try to transfer audio data at the same time in a multi-sources audio stream. In general, once the server receives a data packet, it forwards the packet to every participating client, excluding the packet sender. The delay testing experiment sent every packet to the tested smart phone, whether 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 was the test subject in this experiment. Smart phones B, C, D, and E were participants in the testing network. For the single source audio stream network, 100 and 1000 packets were collected and measured, respectively. For the multi-source audio stream network, 100 packets were collected and measured. Each condition was tested 5 times.

5. RESULTS AND LESSONS LEARNED

For this system, two separate tests were conducted to determine the impact of the system on battery life, and latency of the audio traffic over a network. These results are presented separately, followed by the lessons learned in the process of building this system.

Battery Consumption

Figure 9 in Appendix A provides a graphical representation of the results from the battery consumption test. As the figure shows, when the application is running, but not transferring data, there is very little impact on battery performance. This is not unexpected, as the client application only maintains a TCP connection with the server, and an open UDP port for incoming packets.

Under conditions of heavy data, there is a noticeable impact on battery life. Based on the results shown in figure 9, the application requires roughly half the time to reduce the available battery life from 90% to 85%. Based on this, we can project that a phone running this application would drain it's battery it a little less than 5.3 hours. This assumes constant traffic levels, and could obviously vary based on other factors.

Delay Testing

The purpose of this experiment was to test the system's multi-user processing capability. The test measured the additional delay when a new participant joins the conference room, and was run under three conditions.

The first condition was with a single audio stream and a sample size of 100 packets for the audio stream. As expected, the delay time to

add a participant grew as the number of conference call participants grew, though the average was still very low. One other factor illustrated by this test was that network traffic at different times appears to have an impact on the delays. As this was not part of the experiment, data was not collected on this factor, though it would likely explain the variation in the results. Results from this test are shown in figure 10 in Appendix A.

In the second condition, the sample size was increased from 100 to 1,000 packets of data. Once the sample size was increased to 1,000 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 system, both the client and server program implemented FIFO as the audio data processing policy. If the packet receiver's reading speed is too slow, more and more incoming data will remain in the buffer. Once again, the congestion on the network itself likely played a role in the test results. The results from this test can be seen in figure 11 in Appendix A.

The final test condition was with a sample size of 100 packets, but with multiple sources of the audio stream, rather than a single source. Under these conditions, the average delay ballooned to over 2.5 seconds, and went as high as 3.9 seconds. This delay would be noticeable to participants in the audio conference. Complete results are shown in figure 12.

A design defect caused the reading speed problem, particularly in the final test condition. The program is designed to receive a packet and write that data into the audio track. These two events occur sequentially, so in order to receive a new packet, the thread has to wait for the program to call 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. By implementing two threads, receiving data and writing tracks become two independent events which should increase the speed of reading data.

Lessons Learned

One of the primary issues with the development of this system revolved around issues with the Android OS itself. To delve into this area, some explanation is required.

One of the primary difficulties in the development for the client application came from implementing the SIP functionality. These could have been resolved by using either the library provided in the android.net sip package, or the library provided in the android.net.rtp package. Using these libraries would have eased development, as this would have negated the need to design the low level audio I/O.

The reason these were not implemented is that they require different API levels (9 and 12 respectively). Different API levels are not available to every version of Android. The API levels available to different versions of Android are presented in table 3 in Appendix A. One of the goals for this project was to implement a system that would be available on multiple versions of Android, which required working at a significantly lower API level. This is because the majority of Android devices do not, and cannot, run the latest version of the Android OS. In fact, when the discovery of the libraries was made, an attempt was made to upgrade the OS on the devices used in this experiment. However, this was not possible – the devices would not support newer versions of the operating system.

There were also problems resulting from using open source libraries for the server and client software developed. Specifically, some of the open source projects are not well documented, which makes them difficult to implement, especially when interoperability is required. A great deal of the open source software was created in C/C++, which requires additional time to research how this native code operates, and how it must be embedded using the Android NDK toolset.

Another problem was hardware diversification. This project used multiple devices to reflect the fact that there are multiple hardware vendors that produce Android handsets. However, each hardware vender has different settings for their audio devices. Also, the minimum buffer size required for the relative Android audio record object (android.media.AudioRecord) may differ. Even if the configurations are all the same, the sound quality on different smart phones varies. Because of this, 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.

On the server side, the biggest hurdle was audio mixing for multiple streams. If a server does not have any mechanism for audio mixing, then the audio stream cannot be sent concurrently. For instance, assume three different audio stream sends from different clients, with each of the streams comprised of five packets (A_1~A_5, B_1~B_5, and C_1~C_5). Each packet includes audio data which can be played for n milliseconds. The server then determines client D to be these three streams final destination. In this case, client D will receive 5 ×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 5n 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, we saw that if multi-source packets are transferred concurrently, the client program will have a significant delay. If the server can mix different packets and reduce the overall number of sending packets, the delay can be considerably improved. Careful design for audio mixing is necessary to improve overall system performance.

6. CONCLUSIONS AND FUTURE WORK

This project undertook the development of an audio conferencing solution based on a server side and a client side built on the Android platform. While there could certainly be demand for applications such as this, the development of such a system was more difficult than anticipated. When developing audio solutions for multiple streams, it is necessary to implement a server side solution for audio mixing. Also, when implementing an Android application, the developer must make a choice. Either they can develop an application that will work on the majority of Android devices by working at a low API level, or they can simplify the problem by working at a higher level API. One problem of course is that to develop for a higher level API, the developer must have access to devices capable of running more recent versions of Android, and must accept that if they wish to sell their application, they will be selling to a more limited market.

A possible future study would also involve looking more carefully at the impact of other network traffic on VoIP calls. While there has certainly been research done on this area (Vaiapury, Nagarajan, & Jain, 2009; Mahani, Kavian, Naderi, & Rashvand, 2011), it was not the intent of this study to examine this in particular. It would likely be worth testing the impact of delays on VOIP over smart phones to see if there are quality differences, or if network congestion impacts any of the other measured factors mentioned in this study.

While there are challenges to developing this type of application, we believe that the demand for this type of application will continue to expand. As the use of smart phones becomes more and more widespread, and users become more comfortable with the idea of using VOIP, there will be increased pressure on the development community to develop this type of solution.

7. REFERENCES

Agarwal, Y., Chandra, R., Wolman, A., Bahl, P., Chin, K., & Gupta, R. (2007). 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, San Juan, Puerto Rico.

Android Developers, Android API Levels, Retrieved June 18, 2012, from http://developer.android.com/guide/appendix/api-levels.html#level12.

CISCO, Overview of SIP. Retrieved June 18, 2012, from http://www.cisco.com/en/US/docs/ios/12_3/sip/configuration/guide/chapter0.html.

comScore (2012). comScore Reports July 2012 U.S. Mobile Subscriber Market Share. Retrieved September 14, 2012 from http://www.comscore.com/Press_Events/Press_Releases/2012/9/comScore_Reports_July_2012_U.S._Mobile_Subscriber_Market_Share

Freese, D.D. (2005). Voice over Internet Protocol. *Saturday Evening Post*, (277:1), 52-55.

Gilson, C.L., Xia, R. (2007). Spanning the Pacific Ocean through Voice-over Internet Protocol Chat with the Hadley School for the Blind. *Journal of Visual Impairment and Blindness*, (101:4), 232-236.

International Telecommunication Union (1996). Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service. Recommendation H.323 (11/96). Retrieved September 14, 2012 from http://www.itu.int/rec/T-REC-H.323/ .

Isnardi, M.A, Fielder, L.D., Goldman, M.S., Todd, C.C. (2006). ATSC Video and Audio Coding. *Proceedings of the IEEE,* (94:1), 60-76.

Jaiswal, M.P., Raghav, B. (2004). Cost-Quality based consumer perception analysis of voice over internet protocol (VoIP) in India. *Internet Research*, (14:1), 95-102.

Jia, W.K. (2008). Session-Initiation Protocol Methodology Handbook Second Edition, Kings Information, Taipei, TW.

Kroon, P. (1995). Evaluation of speech coders, in Speech Coding and Synthesis. Elsevier Science, Amsterdam.

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

Mahani, A., Kavian, Y.S., Naderi, M., Rashvand, H.F. (2011). Heavy-tail and voice over internet protocol traffic: queuing analysis for performance evaluation. *IET Communications,* (5:18), 2736-2743.

Naeem, M., Namboodiri, V., & Pendse, R., (2010). Energy implication of various VOIP codecs in portable devices, Local Computer Networks (LCN), 2010 IEEE 35th Conference on, pages 196-199, Oct. 2010.

Park, Namkee (2010). Adoption and Use of Computer-Based Voice Over Internet Protocol Phone Service: Toward and Integrated Model. *Journal of Communications*, (60:1), 40-72.

Tech-Pro.net, 2012 Introduction to Voice over IP (VOIP), Retrieved June 18, 2012, from http://www.tech-pro.net/voice-over-ip.html.

Vaiapury, K., Nagarajan, M., Malmurugan, J., Kumar, S. (2009). Ambience-based voice over internet protocol quality testing model. *IETE Journal of Research*, (55:5), 212-217.

Wikipedia (2012a), Codec, Retrieved June 18, 2012, from http://en.wikipedia.org/wiki/Codec.

Wikipedia (2012b), Audio Codec, Retrieved June 18, 2012, from http://en.wikipedia.org/wiki/Audio_codec.

ISSN: 2167-1508
v5 n2220

Appendix A: Figures and Tables

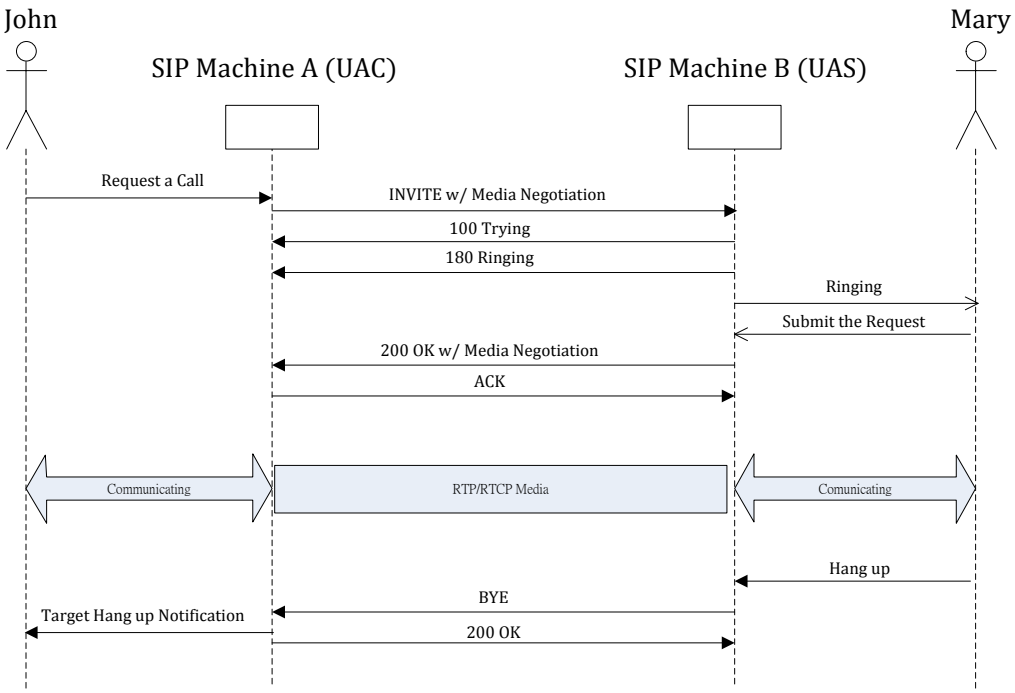

Figure 1: SIP Direct Call Model.

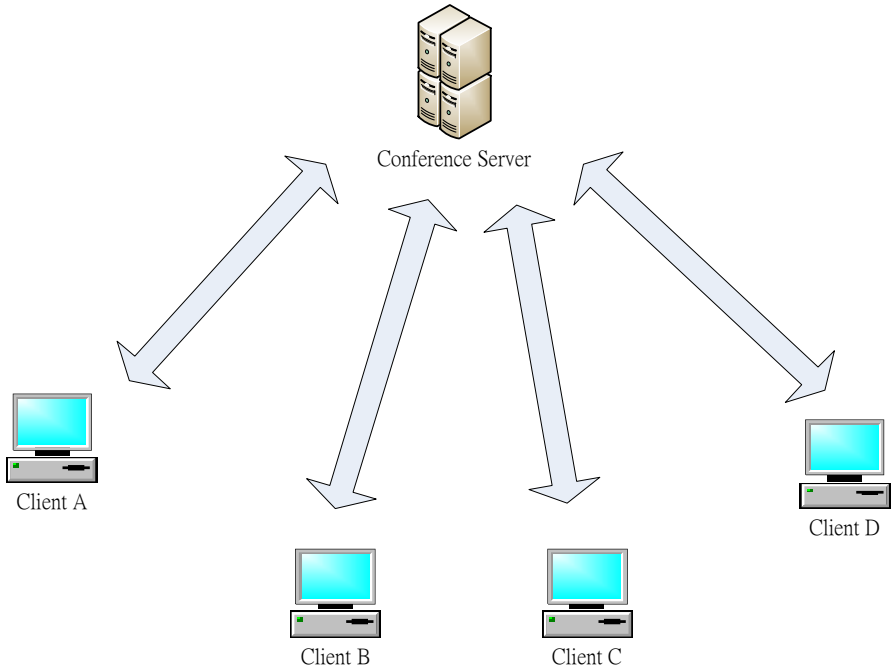

Figure 2: Centralized Conferencing Model

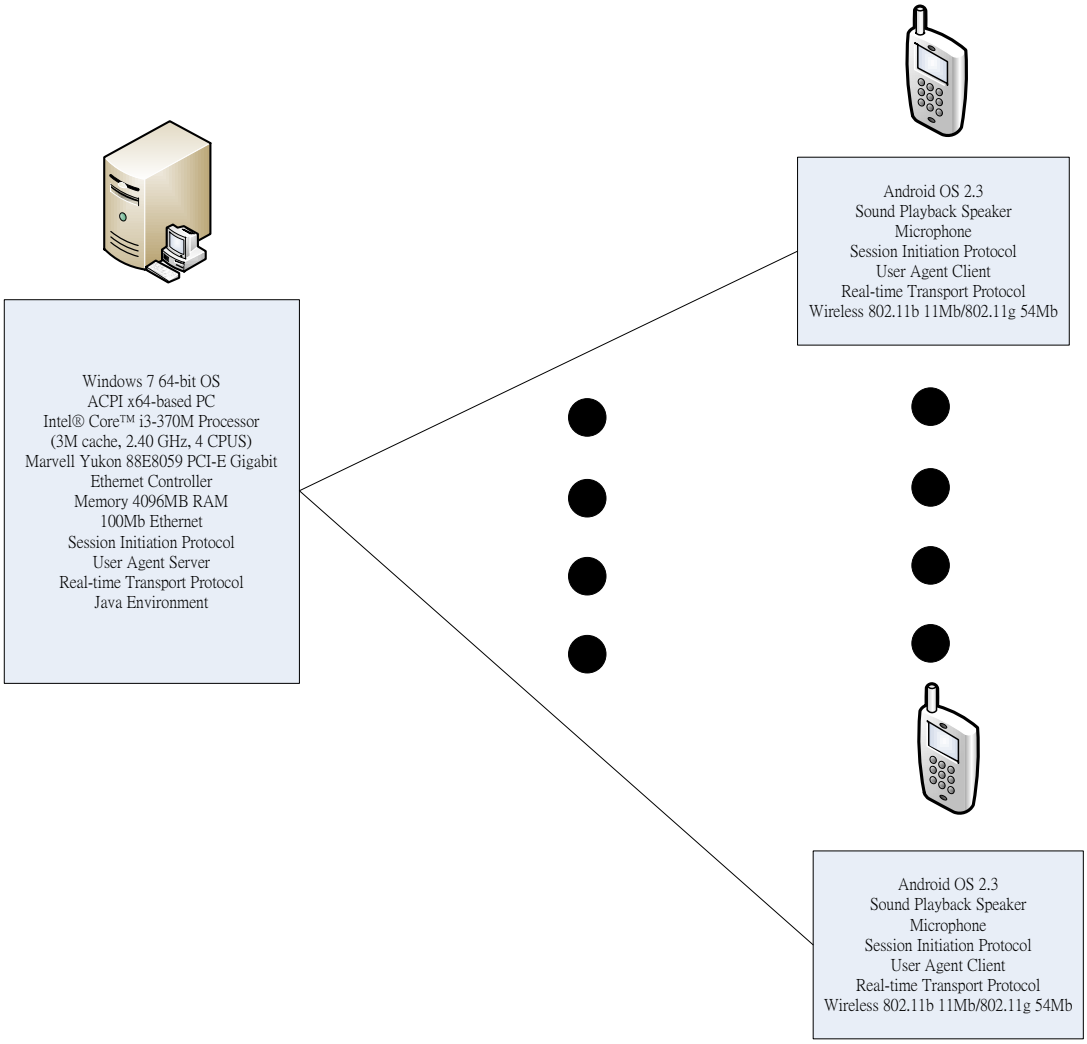

Figure 3: Two-Tier Client Server Architecture

ISSN: 2167-1508
v5 n2220

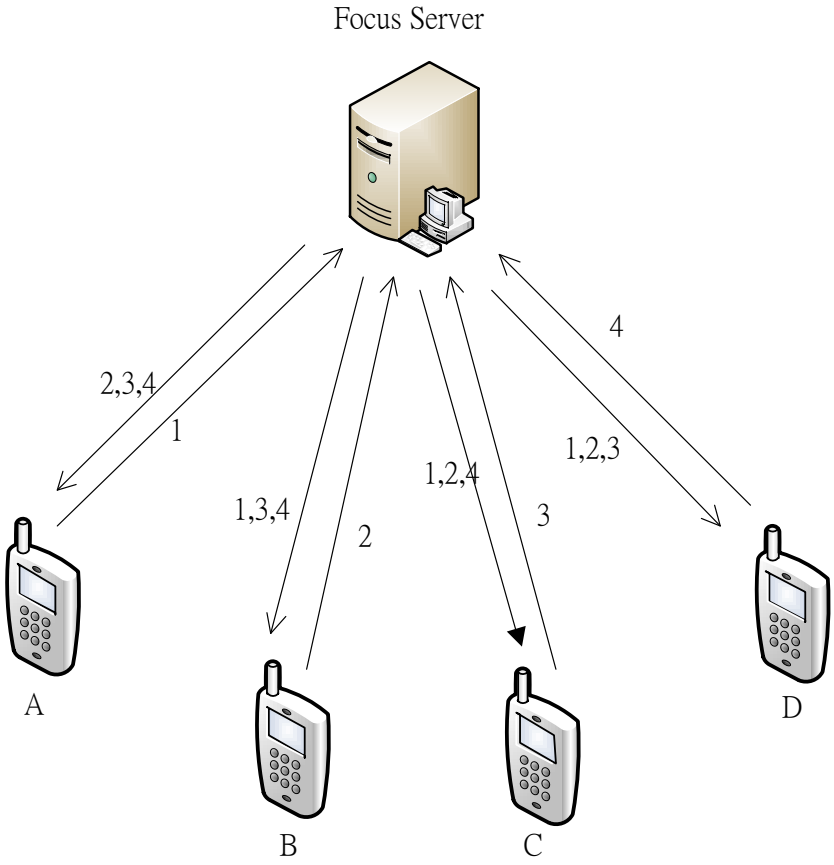

Figure 4: Centralized Message Flow Model

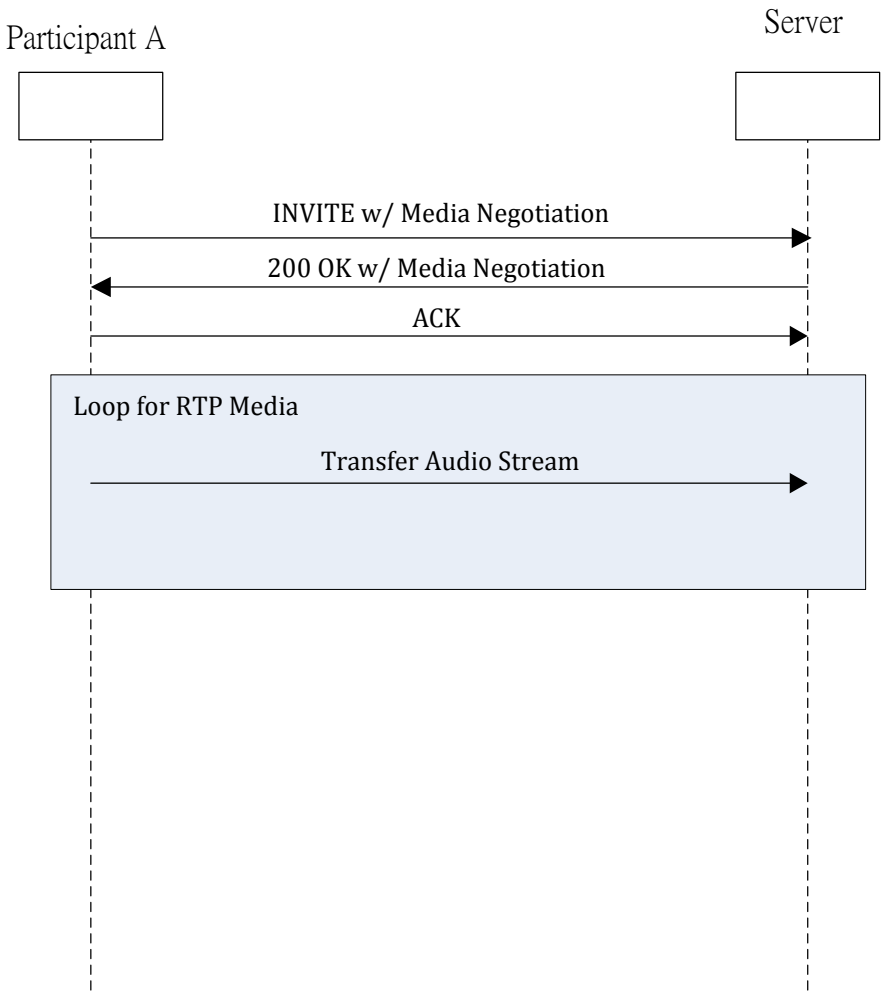

Figure 5: First client joining the conference

ISSN: 2167-1508
v5 n2220

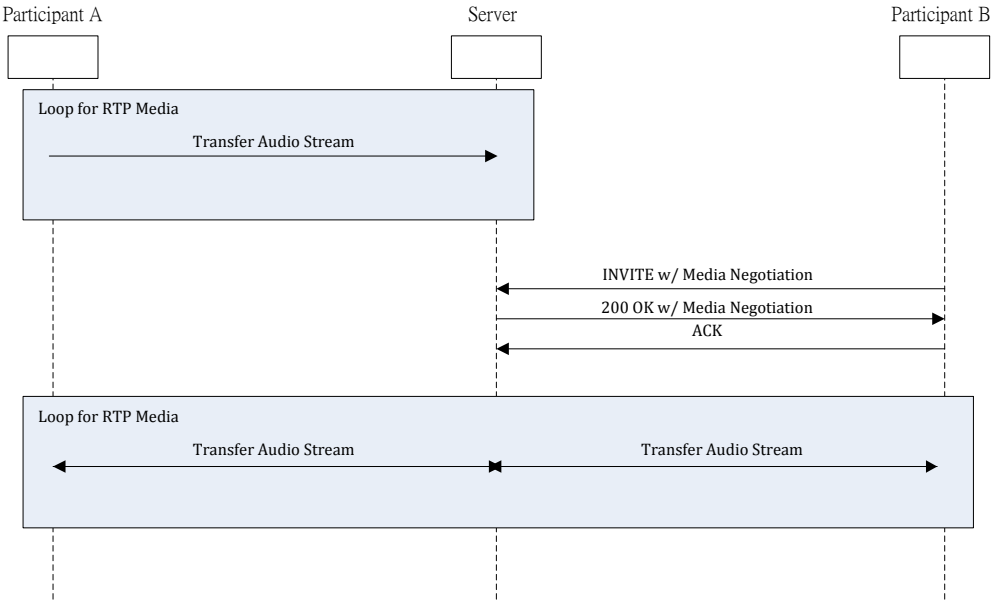

Figure 6: Second client joining the conference

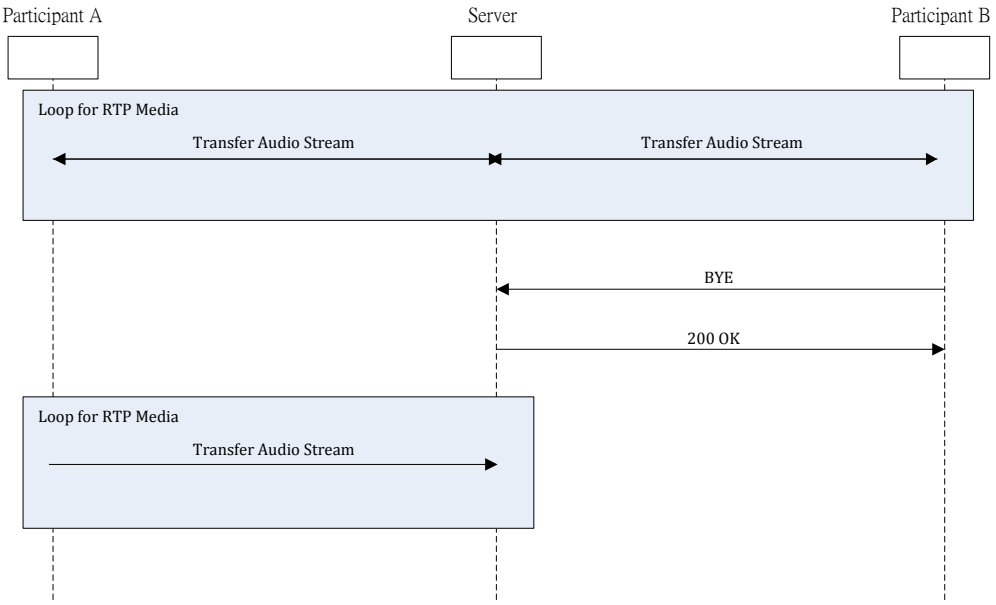

Figure 7: Client leaving the conference

ISSN: 2167-1508
v5 n2220

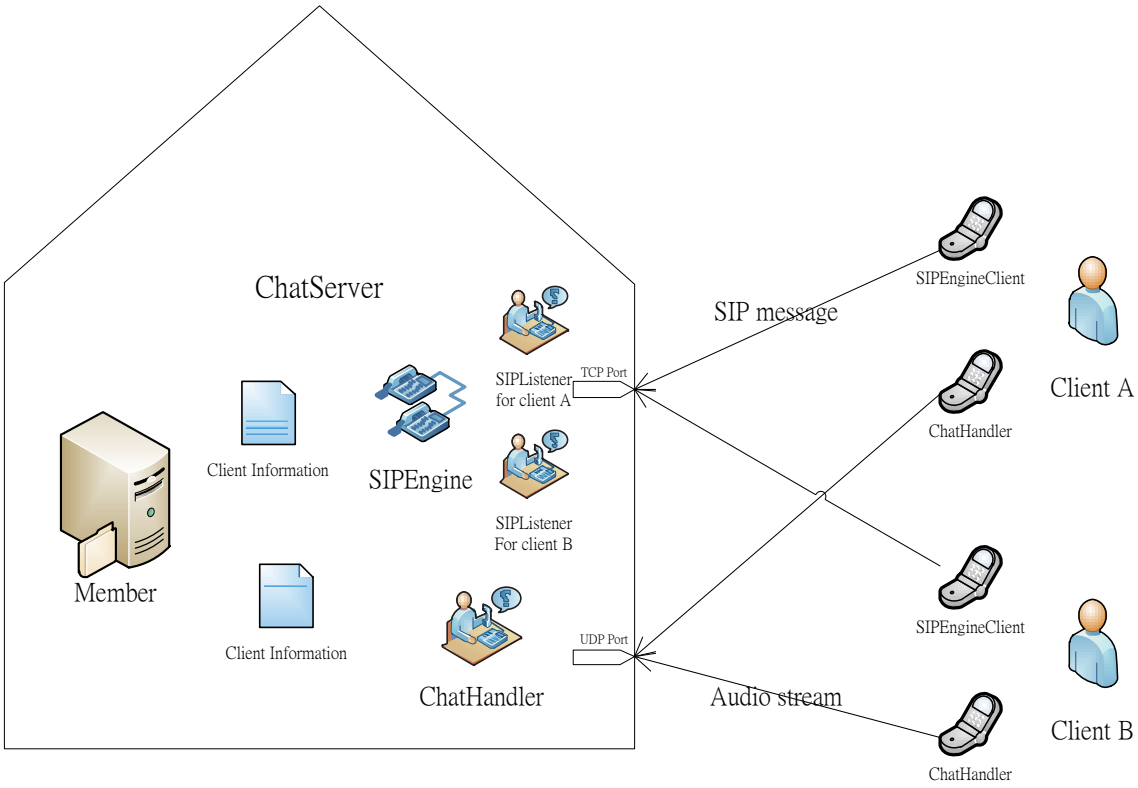

Figure 8: System architecture

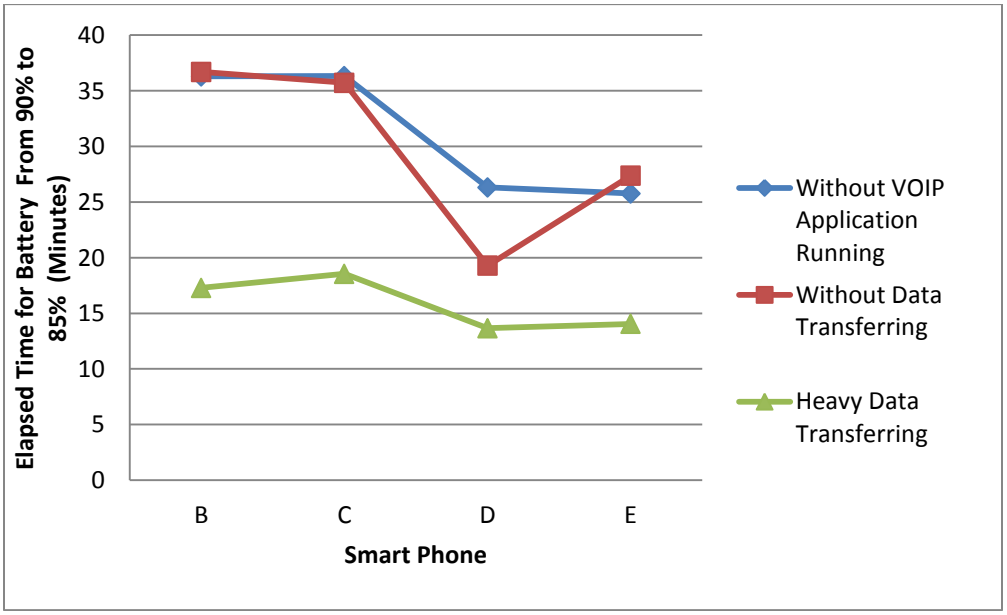

Figure 9: Elapsed time to consume 5% of battery life

ISSN: 2167-1508
v5 n2220

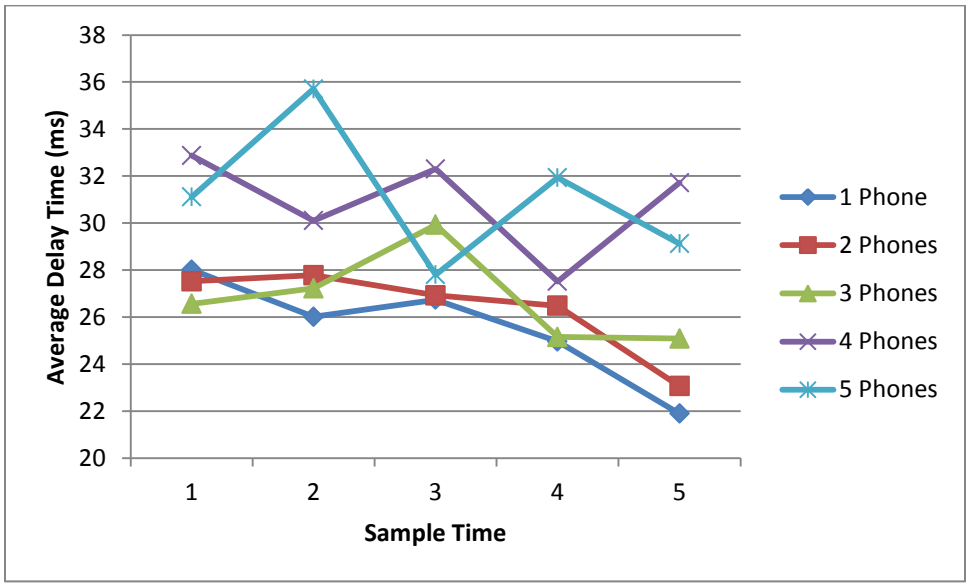

Figure 10: Average delay time of single source audio stream network (100 Packets)

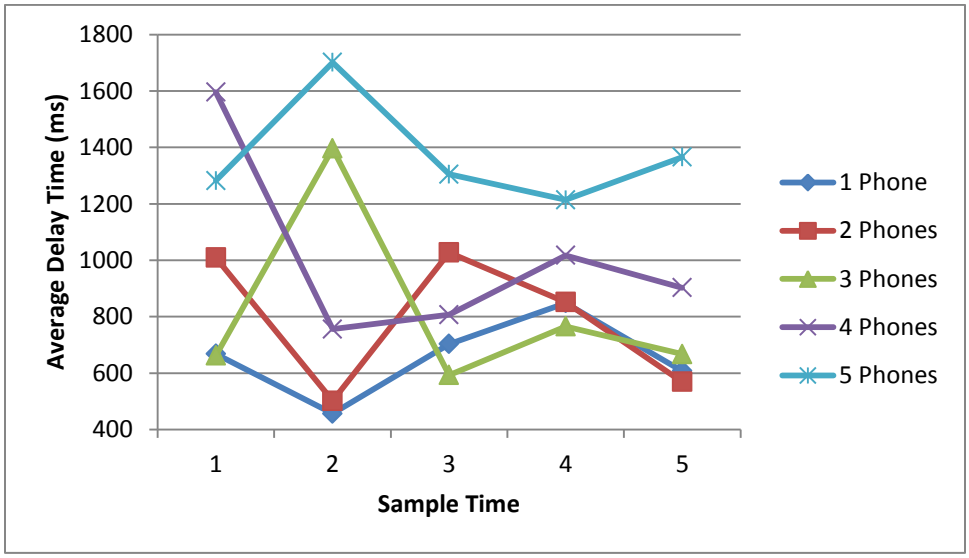

Figure 11: Average delay time of single source audio stream network (1000 Packets)

ISSN: 2167-1508
v5 n2220

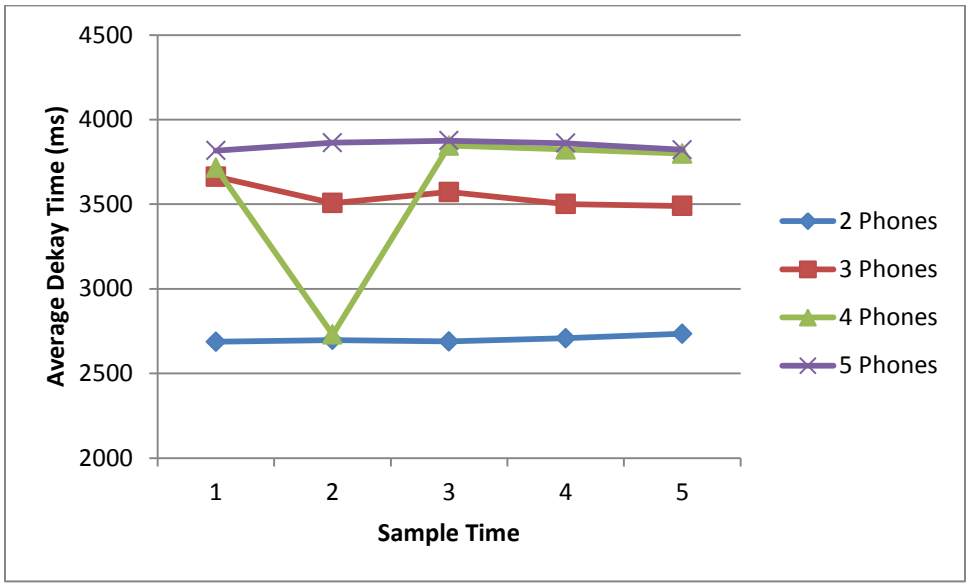

Figure 12: Average delay time of multi-sources audio stream network

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

Table 1*: Third-party VOIP Application Supported Codec*

Device	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

Table 2: *Experimental Smart Phones Specification*

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	ÉCLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_1
Android 1.0	1	BASE

Table 3: API Level Supported by each version of the Android OS (Android.com, 2012)