Voice over Internet Protocol on Mobile Devices

Guo Fang Mao*, Alex Talevski†, Elizabeth Chang‡
*†‡DEBI Institute, Curtin University, GPO Box U1987 Perth Western Australia 6845
Email:(*dean.mao,†alex.talevski,‡elizabeth.chang)@cbs.curtin.edu.au

Abstract

Voice over Internet Protocol (VoIP) is a way to carry out a telephone conversation over a data network. VoIP products promise converged telecommunications and data services that are cheaper, more versatile and provide good voice quality as compared to traditional offerings. Although VoIP is widely used, VoIP on mobile devices is still in its infancy. Currently, there are a number of VoIP solutions for mobile phones. However, VoIP solutions developed using Java 2 Platform Micro Edition(J2ME) are not available. Java based solutions are widely compatible with many devices. In this paper, strong focus has been granted to cross-device compatibility through the use of the widely supported J2ME framework. The implementation details of VoIP client using J2ME are illustrated.
Key words: *VoIP, RTP, SIP, J2ME, GPRS, Asterisk, PBX, UDP, WiFi, ExpressTalk.*

I. Introduction

To date, the Public Switched Telephone Network (PSTN) has been used to conduct telephone calls over a wired network. With the development of computing technology, Voice over Internet Protocol (VoIP) has been established as an alternative to traditional telephony networks. VoIP allows telephone conversations to take place over a data packet-switched networks like the Internet.

VoIP products promise converged telecommunications and data services that are cheaper, more versatile and provide improved voice quality as compared to traditional offerings. Although VoIP is widely used, VoIP on mobile devices is still in its infancy.

The second section of this paper details the VoIP on mobile devices solution. Section three provides a proposed solution and architectural overview. Section four details the system implementation. Section five demonstrates and evaluates the implementation outcomes. The last section concludes this paper and recommends some future work.

II. Background and Related Work

VoIP is the digitalizing of voice using an analog to digital converter(ADC), sending this data through a data network and the reassembing of this data to form the original analog format using a digital to analog converter(DAC). VoIP is made of two parts, signaling and data transport. The VoIP signaling function can be performed using protocols such as SIP[1], H.323[3] and MGCP[4]. Data transport can be performed by The Real-time Transport Protocol(RTP)[2]. This protocol is used to deliver voice data during conversation.

A. Session Initiation Protocol (SIP)

The Session Initiation Protocol[1](SIP) is an application-layer control signaling protocol. SIP is used to create, modify, and terminate multimedia sessions or conferences such as Internet Telephone calls. The SIP message format is similar to the Hyper Text Transfer Protocol(HTTP) message format[1].

Two main components in SIP are user agent(UA) and servers:

- **UAs:** are regarded as a client that can send the request and response together. This includes a user agent client and a user agent server.
- **Servers:** are used to receive requests from clients for servicing and sending responses back to the clients. The servers are typed as redirect server, proxy server and registrar.

B. Real-time Transport Protocol (RTP)

Once signaling functions are implemented, voice data needs to be transmitted between clients. Real-time Transport Protocol(RTP) is viewed as the most powerful protocol to deliver multimedia packets in a session. RTP is defined by Internet Engineering Task Force(IETF). It consists of two parts:

- **RTP:** is used to carry voice data
- **RTCP:** is used to monitor the quality of services and information about the participants who are in a sessions

C. Voice Codecs

In VoIP, there are many different audio codecs. The bandwidth required during a VoIP conversation naturally depends on the codec. Table I describes the audio codec supported in VoIP. The G.711 codec is used on most telephony systems all over the world. The G.729 codec provides the best voice quality. However, due to the native support of G.711 by mobile devices, it is more suitable to use G.711.

Codec	Sampling Rate	Bandwidth	Payload size
G.711	8KHz	64Kbps	20ms
G.722	16KHz	48,48,64Kbps	30ms
G.723.1	8KHz	5.3,6.3Kbps	30ms
G.726	8KHz	24,42,40Kbps	20ms
G.728	unknown	16Kbps	
G.729	8KHz	8Kbps	20ms
GSM	8KHz	13Kbps	
iLBC	unknown	13.33Kbps	30ms
iLBC	unknown	15Kbps	20ms

TABLE I
AUDIO CODECS SUPPORTED IN VOIP[5]

	Symbian	J2ME	BREW
Foundation	C++	Java	C++
Learning Curve	Difficult	Average	Difficult
Emulator	Free	Free	Limited
Debuggers available	Good on latest version	Excellent	Need Payment
Development Tool Cost	Varies(free tools available)	Free	Expensive
Cross-Platform Deployment	Compile per target	Average	CDMA handsets only
Developer Community and Support	Extensive	Extensive	Limited
Market penetration	Extensive	Extensive	In few countries
SIP/RTP Support	Yes/Yes but complicated	Yes/No	Unkown

TABLE II
COMPARISON OF MOBILE PLATFORM DEVELOPMENT

D. Mobile Application Development Environment

There are a number of integrated development environments, languages, frameworks and libraries that can be used to develop the solution proposed in this paper. Table II details the differences between the most popular mobile development environments. J2ME is chosen as the development tool in this paper. Although RTP is not supported on J2ME, implementation of RTP on J2ME framework is one of key features of this paper.

E. Existing Related Work

Although there are no VoIP solutions developed for J2ME enabled mobile devices, some solutions exist:

1) Burdet et. al. have developed a VoIP solution for mobile device using the Symbian operating system trough bluetooth[6]. SIP, SDP and RTP features operate on a Symbian emulator. The work of Symbian on mobile phone shows that VoIP can be performed over a Symbian Mobile Phone. The proposal illustrated in this work is different to the ideas illustrated in this paper.
2) Vikram Goyal experimented with streaming content data using JAVA ME[7]. An existing audio file on Darwin server is requested by a client through RTP protocol. A custom data source class, source stream class and RTSP class are created to handle the incoming RTP packets. However, this work does not follow RTP standard. It also does not mention how to deliver and play voice data.

III. Proposed Architectureal Framework

A. Proposed SIP Framework

SIP performs signaling functions in a session. According to the SIP specification[1], the sequence for making a SIP call is illustrated in Figure 1. In this paper, SIP functions are implemented based on the sequence in Figure 1.

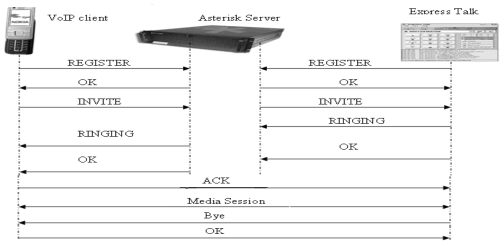


Fig. 1. VoIP Client SIP Message Sequence

As signaling functions are provided, SDP is used to negotiate media session description between clients. The media session description includes encoding format, payload type and sampling rate of voice packets. Also, IP address and port number for receiving voice packets is included.

B. Proposed RTP Framework

After both parties reach the same media session description, the RTP protocol is used to deliver voice data. Figure 2 shows the details of RTP Framework that is used as a basis for the implementation. The implementation carries out the functions illustrated in Figure 2.

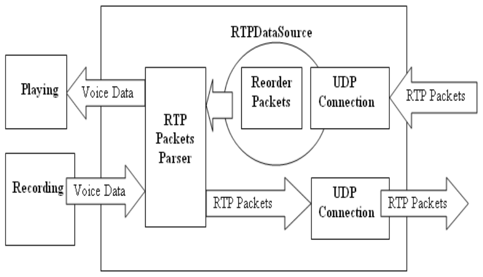


Fig. 2. RTP Framework

C. Expected Results

Initiating a call using SIP is expected as the SIP API is available on J2ME. The implementation of RTP on

J2ME framework is essential to deliver voice data. The client should be able to talk with a existing VoIP desktop client in a wireless(WiFi) network such as ExpressTalk to prove its compability with VoIP standards. In addition, the processor utilisation, memory consumption, storage size and bandwidth cost should be able to fit on a mobile devices.

IV. Implementation of Proposed Framework

In this section, the implementation of the proposed architecture is going to be discussed. Requirements for setting up development environment is presented. SIP for J2ME is implemented to initiate VoIP calls on mobile emulator. Moreover, Session Description Protocol(SDP) is used for negotiating media session description. RTP is used to deliver voice data in this implementation.

A. Requirements of the Proposed Architectural Framework

A number of additional softwares and tools were used for implementation and evaluation. They performed different functions during the implementation process. They are outlined as follows:

- **Asterisk** is an open source Private Branch Exchange(PBX) that gives you real-time connectivity for both PSTN and VoIP networks. It also functions as a SIP proxy.
- **Sip-midp** is a nokia mobile phone emulator.
- **ExpressTalk** is a third-party VoIP solution to test whether the mobile phone client can communicate with existing third party VoIP software that complies to the VoIP protocol.
- **Ethereal** is a network protocol analyzer that is used to capture network packets during transmission
- **J2ME** is used to develop the VoIP application.

B. SIP for J2ME

JSR 180 is an optional package that supports basic SIP API on J2ME. It can run on devices with limited memory. There are six SIP request methods are explained as following in SIP specification[1]: REGISTER, INVITE, ACK, CANCEL, BYE, OPTIONS. Two main requests are introduced in the following:

1) REGISTRATION: registration is the first essential part for starting a session. A SIP[1] registration message is generated and sent to Asterisk server. According to SIP specification[1], the transactions between client and Asterisk server are illustrated in Figure 3.

2) INVITATION: The invitation is the second step that a VoIP call needs to do after registration. According to SIP API, SIP Invitation Message is generated as follows:

```
INVITE sip:212@203.94.167.26
To: Dean1 <sip:212@203.94.167.26>
From: Dean2 <sip:213@203.94.167.26>;
      tag=1928301774
Call-ID: a84b4c76e66710
```

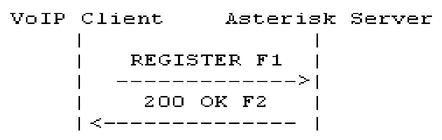


Fig. 3. SIP Client Registration

```
CSeq: 314159 INVITE
Contact: <sip:dean2@203.94.167.28:1728>
Content-Type: application/sdp
Content-Length: 142
```

C. SDP on J2ME

When a client wants to communicate with another client, the session description details must be received by both parties. Then, the callee and caller will negotiate to choose a common media for voice packets. Session Description Protocol(SDP) is used to describe the media details for clients. The functions of SDP :

- Define where the voice data should be sent such as the port number and IP address
- Define the codec to be used by other party.
- Identify the channel and sample rate of voice data.

The following is one of the SDP message example according to SDP specification[8].

```
String SDP = "" +
 "v=0\n"+
 "o=dean1 0 0 IN IP4 203.94.167.28 \n" +
 "s=-\n" +
 "c=IN IP4 203.94.167.28 \n" +
 "t=0 0\n" +
 "m=audio 10086 RTP/AVP 8\n" +
 "a=rtpmap:0 PCMU/8000\r\n";
```

D. RTP on J2ME

As J2ME does not support UDP rather than RTP, it is necessary to implement the RTP features. It is widely known that RTP packets house in UDP packets. According to RTP specification[2], the following features are required:

- Constructing/Extracting RTP packets from UDP packets
- Receiving/Sending UDP packets
- Streaming and playing voice data

1) Open Connection for Receiving: Syntax and coding of UDP Connection are as follows:

```
{protocol}://[{host}]:[{port}]
```

```
String c = datagram://:10086;
DatagramConnection dc_socket =
(DatagramConnection)Connector.open(c);
```

The above code specifies a UDP connection on port 10086.

2) Receiving Packets: After connection is open, receiving UDP packets code is available as follows:

```
byte[] fullPkt = new byte[length];
Datagram packet =
dc_socket.newDatagram(fullPkt, length);
dc_socket.receive(packet);
```

The above code means:

- Creates a byte array to store packets
- Declares a datagram packet using the byte array created above
- Receives UDP packet and puts them into the datagram packet created before. The receive method in the datagram connection object is always waitting for UDP packets to arrive.

3) RTP Packets Extraction: Once receiving UDP packets, RTP packets should be extracted from UDP packets. The following code extracts RTP packets from UDP packets.

```
byte[] rtp_packet = packet.getData();
```

There are 12 bytes RTP header in the rtp_packet in the above example. However, the player only needs the media data. Therefore, the first 12 bytes in RTP packet must be disregarded when playing the data.

4) Custom DataSource for streaming: The function of custom DataSource is to receive media data and return the data to the player after data is requested. J2ME JSR135 API shows that a DataSource provides the methods for a Player to access the input data from SourceStream. SourceStream provides the methods to allow a player to read data for processing. Based on the Sun Porting Guide[9], a custom DataSource and SourceStream for customized input protocol must be implemented before creating the Player object.

5) Player: J2ME provides three methods to create a Player for playing back media.

- **Create from a media locator** is used to create a Player to handle the media identified by the locator.
- **Create from a DataSource** is used to handle an application-defined protocol.
- **Create from an InputStream** can be used to interface with other Java API's. This does not provide the necessary random seeking functionality.

As shown before, Creating Player from a DataSource. Then, the code for creating player using DataSource is as follows.

```
//data source is rtp_data_source
Player rtp_play =
Manager.createPlayer(rtp_data_source);
rtp_play.start();
```

If the player is started, it requests media data from DataSource. The experiments show that the player firstly requests headers of media data to identify the details of media data. Therefore, Custom DataSource should return the header of media data to player at begining and then followed by media data(voice data).

6) Recording: Once the local client gets the remote media session description, the client needs to send voice data to destination. According to the API, an example to create a Player for capturing live media data in Manager class is like:

```
Player p = Manager.createPlayer
("capture://audio?encoding=pcm&rate=8000
&bits=8&channels=1");
```

According to the experiment, the final total size for recording one second is about 8000 bytes plus the header size. The header size depends on the encoding format. For instance, the PCM format header size is 44.

7) Constructing RTP Packets: The following shows the steps about how to construct RTP packets based on audio data array and put RTP packets into UDP packets.

- Get the audio data array.
- Get ride of the header of raw audio data. The size of header depends on the encoding format
- Split the original audio data array as some new small arrays.
- Attach the RTP header into each small arrays, then these become RTP packets.
- Create UDP packets based on RTP packets.

Experiments show that the audio size array is about 64.352 kbps(352 bits is the header for raw voice data). After dropping the header, the rest of bytes(about 8000 bytes) in the array is the pure audio data. Assuming that each packet contains 160 bytes audio data, 8000 bytes audio data can be splitted as 50 RTP packets. And the RTP header has to be attached into each RTP packet.

8) Open Connection for Sending Packets: Connection for sending voice packet needs the details of remote IP address. The connection can be created based on SDP message resided in SIP message. A part of code is as follow:

```
String c =
   datagram://203.94.167.27:10086;
//203.94.167.27 is the remote IP
DatagramConnection dc_socket =
(DatagramConnection)Connector.open(c);
```

Then, the UDP packets can be created based on the RTP packets. The code for creating and sending UDP packets is as follow:

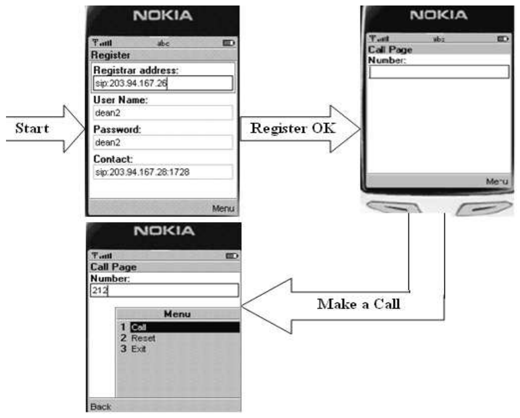
```
Datagram d =
    dc_socket.newDatagram(rtpPacket,172);
dc.send(d);
```

V. Demonstration and Evaluation

Because the experiment is a live conversation, it is impossible to formally demonstrate the client in this paper. In this section, some experimental screenshots are provided.

A. Demonstration

The following are demonstration of the implemented VoIP client.



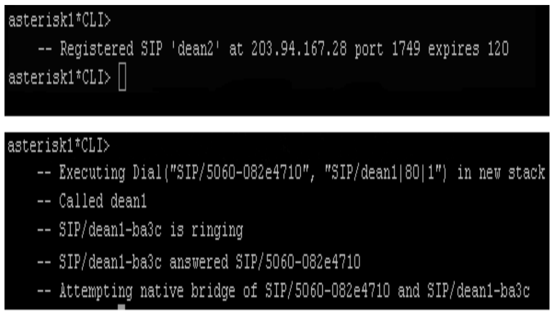
Registrar address:
sip:203.94.167.26
User Name:
dean2
Password:
dean2
Contact:
sip:203.94.167.28:1728

Call Page
Number:

Call Page
Number:
212
Menu
1 Call
2 Reset
3 Exit

Fig. 4. The Demonstration of Making a Call

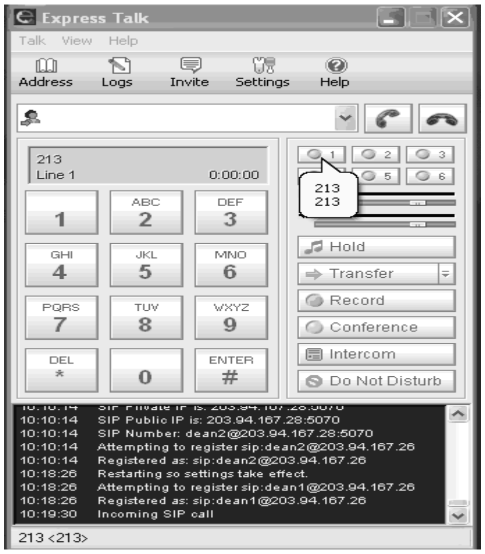
Figure 4 provides screenshots of the experiment for making a call. Figure 5 shows the details of Asterisk server when the clients register and make a call. Figure 6 shows that the third party client(ExpressTalk) received the call from the client implemented in this paper.



```
asterisk1*CLI>
   -- Registered SIP 'dean2' at 203.94.167.28 port 1749 expires 120
asterisk1*CLI>
```

```
asterisk1*CLI>
   -- Executing Dial("SIP/5060-082e4710", "SIP/dean1|80|1") in new stack
   -- Called dean1
   -- SIP/dean1-ba3c is ringing
   -- SIP/dean1-ba3c answered SIP/5060-082e4710
   -- Attempting native bridge of SIP/5060-082e4710 and SIP/dean1-ba3c
```

Fig. 5. The Demonstration of Asterisk Server



213
Line 1 0:00:00

213
213

```
10:10:14    SIP Public IP is: 203.94.167.28:5070
10:10:14    SIP Number: dean2@203.94.167.28:5070
10:10:14    Attempting to register sip:dean2@203.94.167.26
10:10:14    Registered as: sip:dean2@203.94.167.26
10:18:26    Restarting so settings take effect.
10:18:26    Attempting to register sip:dean1@203.94.167.26
10:18:26    Registered as: sip:dean1@203.94.167.26
10:19:30    Incoming SIP call
```

213 <213>

Fig. 6. ExpressTalk Answered Incoming Call

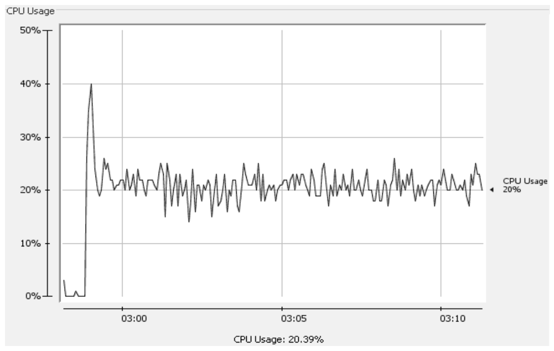


Fig. 7. Processor Utilisation when Receiving and Playing RTP Packets

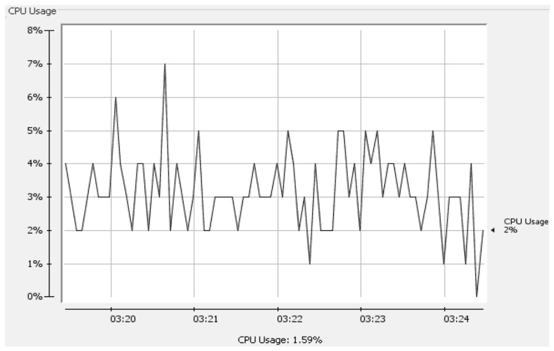


Fig. 8. Processor Utilisation when Recording and Sending RTP Packets

B. Evaluation

In this subsection, the processor utilisation, memory usage, storage size and network bandwidth required by the implemented VoIP client are evaluated.

1) Processor Utilisation: Processor utilisation determines whether the implemented VoIP client is suitable to actual devices. Figure 7 illustrates the processor usage of the VoIP client when receiving and playing RTP packets. Figure 8 illustrates the processor usage of the VoIP client when sending RTP packets. The average processor usage is around 20% for receiving and playing voice data and around 4% for the recording and sending voice data. These processor usage is considered to be reasonably acceptable.

2) Memory usage: In the implementation, the maximum memory size allocated is around 2MB. Figure9 shows the memory usage when receiving and playing RTP packets. Garbage collection is performed to flush the memory once it achieves the peak of memory.

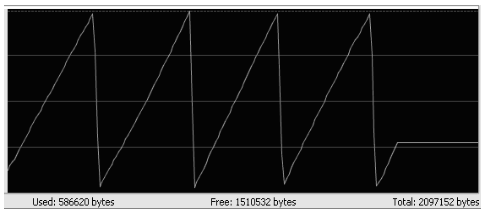
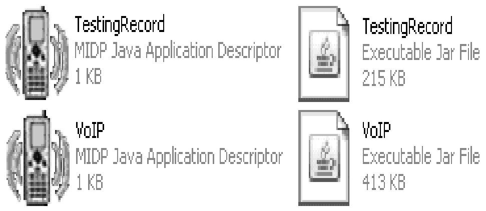


Fig. 9. The Memory Usage for Sending and Receiving Packets

3) *Storage Size:* The application size is a key issue for deploying applications on actual devices. The application size of SIP and SDP functions is around 413k as illustrated in Figure 10. The size for RTP function is 215k. They are all quite suitable for micro devices.

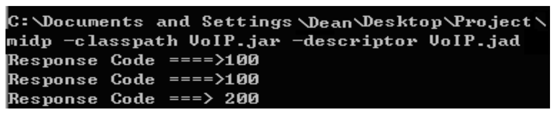


TestingRecord MIDP Java Application Descriptor 1 KB	TestingRecord Executable Jar File 215 KB
VoIP MIDP Java Application Descriptor 1 KB	VoIP Executable Jar File 413 KB

Fig. 10. Storage Size of Implemented VoIP Application

4) *Network Bandwidth:* RTP packets constructed by the VoIP client need to travel through GPRS, 3G or higher bandwidth network. In theory, the bandwidth of GPRS is around 171.2kbps. The size of audio data recorded is 64kbps in implementation. It indicates that the solution performs sufficiently on a typically GPRS network.

5) *SIP:* Evaluating the SIP function is used to track the requests and responds between the client and Asterisk server. Figure 11 shows the response status code from



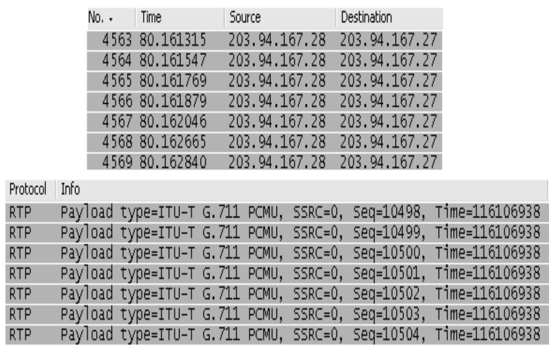
```
C:\Documents and Settings\Dean\Desktop\Project\
midp -classpath VoIP.jar -descriptor VoIP.jad
Response Code ====>100
Response Code ====>100
Response Code ===> 200
```

Fig. 11. Responds From Asterisk Server

Asterisk server. The meaning of status code number in SIP specification[1] is listed below:

- 100: TRYING – request received, continuing to process the request.
- 200: OK – success, the action was successfully received, understood and accepted.

6) *RTP:* Ethereal software can be used to evaluate RTP packets by catching packets on the ExpressTalk side. The result from Ethereal is in Figure 12. It illustrates that the packets are recognized as standard RTP packets.



No.	Time	Source	Destination
4563	80.161315	203.94.167.28	203.94.167.27
4564	80.161547	203.94.167.28	203.94.167.27
4565	80.161769	203.94.167.28	203.94.167.27
4566	80.161879	203.94.167.28	203.94.167.27
4567	80.162046	203.94.167.28	203.94.167.27
4568	80.162665	203.94.167.28	203.94.167.27
4569	80.162840	203.94.167.28	203.94.167.27

Protocol	Info
RTP	Payload type=ITU-T G.711 PCMU, SSRC=0, Seq=10498, Time=116106938
RTP	Payload type=ITU-T G.711 PCMU, SSRC=0, Seq=10499, Time=116106938
RTP	Payload type=ITU-T G.711 PCMU, SSRC=0, Seq=10500, Time=116106938
RTP	Payload type=ITU-T G.711 PCMU, SSRC=0, Seq=10501, Time=116106938
RTP	Payload type=ITU-T G.711 PCMU, SSRC=0, Seq=10502, Time=116106938
RTP	Payload type=ITU-T G.711 PCMU, SSRC=0, Seq=10503, Time=116106938
RTP	Payload type=ITU-T G.711 PCMU, SSRC=0, Seq=10504, Time=116106938

Fig. 12. Packets Sent by Implemented VoIP Client

VI. Conclusion and Future Work

In this paper, the background and related work of VoIP on mobile devices were discussed. The proposed architecture and implementation were also detailed. The implemented client was also demonstrated and evaluated.

This paper proves that a VoIP client can be developed using J2ME followed the standard. and deployed on a mobile phone with the necessary features. The features of the implemented client are suitable for mobile devices.

Although the implemented client is compatiable with the VoIP standard, the client is not implemented completely. Therefore, some future work inlcudes:

- Implementing RTCP in RTP
- Creating SDP message dynamically
- Implementing Duplex conversation

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, R. Sparks, M. Handley, E.Schooler. *SIP: Session Initiation Protocol.* The Internet Engineering Task Force, June 2002.
[2] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. *RTP: A Transport Protocol for Real-Time Application.* Network Working Group, July 2003.
[3] W. Muller, H. Glasmann, J. Kellerer. *Service Development and Deployment in H.323 and SIP.* IEEE, 2001.
[4] F. Andreasen, B. Foster. *Media Gateway Control Protocol (MGCP) Version 1.0, pages 5-33.* Network Working Group, January 2003.
[5] Ozvoip. VoIP Codecs. viewed 18 April 2006[ONLINE] http://compare.ozvoip.com/codecs.php.
[6] L. Burdet, P.Stuedi, A. Frei and G. Alonso. *Symphone: Design and implementation of a VoIP peer for Symbian Mobile phones using bluetooth and SIP.* The ACM Ditial Library, 2006.
[7] V. Goyal. *Experiments in Streaming Content in Java ME. Sun Microsystems*, viewed 06 Septerber 2006[ONLINE] http://today.java.net/pub/a/today/2006/08/22/experiments-in-streaming-java-me.html?page=1, 2006.
[8] M. Handley, V. Jacobson. *SDP: Session Description Protocol.* Network Working Group, 1998.
[9] Inc Sun Microsystem. Porting Guide Java 2 Platform, Micro Edition. *Technical Report: Mobile Media API*, June 2002.