Voice over IP in a resource constrained environment

ALI NESH-NASH

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2006

COS/CCS 2006-6

Voice over IP in a resource constrained environment

12th March 2006

Ali Nesh-Nash

Advisor and Examiner: Gerald Q. Maguire Jr.

Royal Institute of Technology (KTH)
Stockholm, Sweden

.

Abstract

Today, the telecommunication world is focused on mobility. This is popular because since the 1990s most people have integrated their mobile phones into their life. A new factor is the rise of the voice over IP(VoIP) technology, with VoIP over Wireless LANs (WLANs) as the clear next growth area for mobile communications.

The purpose of this thesis was to understand how to save power based upon changing when some operations are performed in a VoIP client. In order to do this, we decided to port minisip to an HP iPAQ 5500 Personal Digital Assistant (PDA), in order to explore some of the issues of running such a client on a PDA - due to its constraints with regard to storage, processing power, and battery power. Minisip is a SIP open source user agent running on Linux and Windows.

This thesis builds upon earlier theses which showed that minisip can offer a secure communications platform with the latest functions which are desired in a mobile personal VoIP system. However, most of these earlier theses utilized desktop, laptop, or server based system, i.e., with few resources constrains. The focus of this thesis was to examine the case of a highly constrained user platform such as an iPAQ.

Abstrakt

Dagens telekommunikationssystem fokuserar på mobilitet. Detta har blivit populärt under 90-talet då mobilitet blev naturligt integrerad i människans vardagliga liv i form av exempelvis mobiltelefoner. Voice over IP (VoIP) har blivit en stor del av dagen teknik där trådlösa system Wireless LANs (WLANs) har blivit en större del av mobilkommunikation.

Målet med denna rapport är att förstå hur strömförbrukningen kan minimeras genom att utföra vissa operationer med hjälp av en VoIP-klient. För att åstadkomma detta porterade vi minisip, en SIP agent som är baserad på öppen källkod och körs på Linux och Windows, till en HP iPAQ 5500, en så kallad Personal Digital Assistant (PDA). Vi valde PDAn för att kunna utforska de begränsningar den medför i form av lagringsutrymme, processorkapacitet, och batteri.

Denna rapport bygger vidare på tidigare rapporter som visar att minisip kan erbjuda en säker kommunikationsplattform med de senaste funktionerna som önskas i mobila VoIP-system. De flesta av dessa tidigare rapporter baseras på system med få begränsningar rörande resurser såsom stationära- eller bärbara datorer samt serverbaserade system. Denna rapports fokus är att utforska detta fall i en miljö med större begränsningar på resurser som till exempel en iPAQ.

Table of contents

CHAPTER 1: MOTIVATION AND PROBLEM DESCRIPTION

We begin with a description of the hardware and software used in this project, then we describe the motivation for porting this software to this hardware.

1.1. Mobile computing and communications

Today, Personal Digital Assistant (PDA) evolved from keeping schedules, calendars and address book information to a phone, a general purpose computer, an mp3 player… All in a single device! No wonder that Microsoft and others strongly believe that this is the next area of the IT market to explode.

iPAQ

In this project we have used the HP iPAQ 5550 (shown in figure 1 and described in table 1) as our mobile device. For more information about the HP iPAQ 5550 see [1] and [2].

Figure 1: HP iPAQ h5500 Pocket PC . Picture appears with the permission of HP

Specifications of the HP iPAQ 5550:

OS	**Microsoft ® Windows ® Mobile™ Pocket PC 2003 Premium**
Processor	**400 MHz Intel® XScale™ technology-based**
Memory	**128 MB SDRAM, 48 MB Flash ROM Memory**
Weight	**138 x 84 x 15.9 mm**
Battery	**Removable/rechargeable 1250 mAh Lithium-ion polymer battery**
Dimensions	**206.8 g**

Table 1: Specification of the HP iPAQ 5550

1.1.2. Minisip

Minisip[3] is an open source SIP softphone developed by students and faculty at Royal Institute of Technology (KTH) in Stockholm, Sweden. The most attractive features are related to security. It implements TLS for securing the signaling, as well as the MIKEY protocol and SRTP (for key management and voice encryption respectively). Other features are: video support, Push to talk [4], it runs on ARM platforms (such as the iPAQ under Linux), It has several interesting features, such as instant messaging, seamless multi-party calls, and support for multiple SIP accounts. MIKEY is an important part of minisip implementation and will be examined in section 2.1.

1.1.3. Minisip on iPAQ

As handheld devices become more and more popular, software vendors start to target these platforms. For example, Skype has already shipped a windows CE compliant version. Minisip has the attractiveness of being based on open protocols like SIP, thus providing the necessary transparency to the end users. For a comparison between minisip and Skype, read Arranz's thesis [5]. Moreover, as VoIP is rapidly replacing telephony (both fix and mobile porting) porting to iPAQ seems to be a natural step for a VoIP software. Additionally, as an open source project minisip enables developers to add the features that they want; while also enabling those who are security conscious to analyze their VoIP client.

A dream that's becoming reality little by little is nearly free telephony. VoIP is converting traditional telecommunications into a more open market. In the 1990s fixed telephony was overtaken by the immense success of GSM. This was driven (in part) because today's world is mobile, people move and would like their world to move with them. Taking into consideration the significant capabilities of the handheld personal computing and communication devices, this world seems to center on personal handsets (be the mobile phones or PDAs). Moreover, the need to communicate is a constant in human nature, but the key of success of minisip is not only the low cost (often the incremental cost is nearly zero), but also the image of transparency it displays: an open source project based on open standards.

Minisip has already been ported to iPAQ under Linux[6], but unfortunately Linux has little power management for this platform. This is one of the reasons why porting minisip to Windows CE is so valuable. Additionally, given the large numbers of users who want to run other applications that run on the standard software that is pre-installed at the factory - these users could potentially run minisip as just another application. Finally, Minisip has proven to utilize only limited resources and power consumption so it is an attractive application for such platforms.

CHAPTER 2: BACKGROUND

The purpose of this chapter is to provide the necessary information to understand this thesis work. After presenting the working environment and the Microsoft Windows CE architecture, I will focus on minisip to provide an overview of the software, focusing mainly describing on its security features.

2.1. Voice over IP

Minisip is fundamentally a voice over IP (VoIP) user agent (i.e., an application program supporting a voIP user). Thus, I will present the basics of VoIP focusing on the protocols minisip uses. I will particularly comment upon a specific security method: Diffie-Hellmann.

During most of the last century, there was only one telephony network: the Public Switched Telephony Network. With the introduction of packet networks and the explosion of Internet, an infinity of possibilities opened for telephony. Although people initially believed that the non-reliability of packet networks and that routing - packets can follow different paths and thus arrive in inverse order- would keep telephony over IP as a mere dream.

Nowadays, due to the increase of bandwidth, the delays are reduced and the quality of an average voIP call may soon become better than a classical telephony one. Moreover the low cost of VoIP is another asset that makes it very attractive to users. Finally, VoIP offers a new panel of possibilities such as video/audio multi-conferencing, file transfer, instant messaging…

2.1.1. RTP/RTCP

In order to make a call, every user has to transmit data (received from a microphone connected to an Analog to Digital Converter(ACD)- coded, packetized…) to the other party - who receives it, decodes, etc,… before passing it to a Digital to Analog Converter (DAC). In this section we examine one protocol used to send packets containing audio (or video) samples across the network.

The Real Time Protocol (RTP)[7]provides end-to-end network transport functions suitable for applications having real time requirements such as audio or video. The monitoring of this uses the Real Time Control protocol. The design of RTP and RTCP is independent of the underlying layers (i.e. the transport, network, link and physical layers). However, RTP and RTCP are typically using UDP [8] (i.e., RTP is encapsulated in UDP packets), although they can be on top of any network protocol. The choice of UDP is because it is not connection oriented and the lack of reliability is often acceptable (because of the human perception limits). Because the source of packets is typically a voice codec which is running at a fixed upper rate, there is no need for flow control.

Furthermore, flow control would be incompatible with the bounded delay requirements of real time applications.

In order to have a clear view of RTP and RTCP in the protocol stack we are using, refer to figure 2 below.

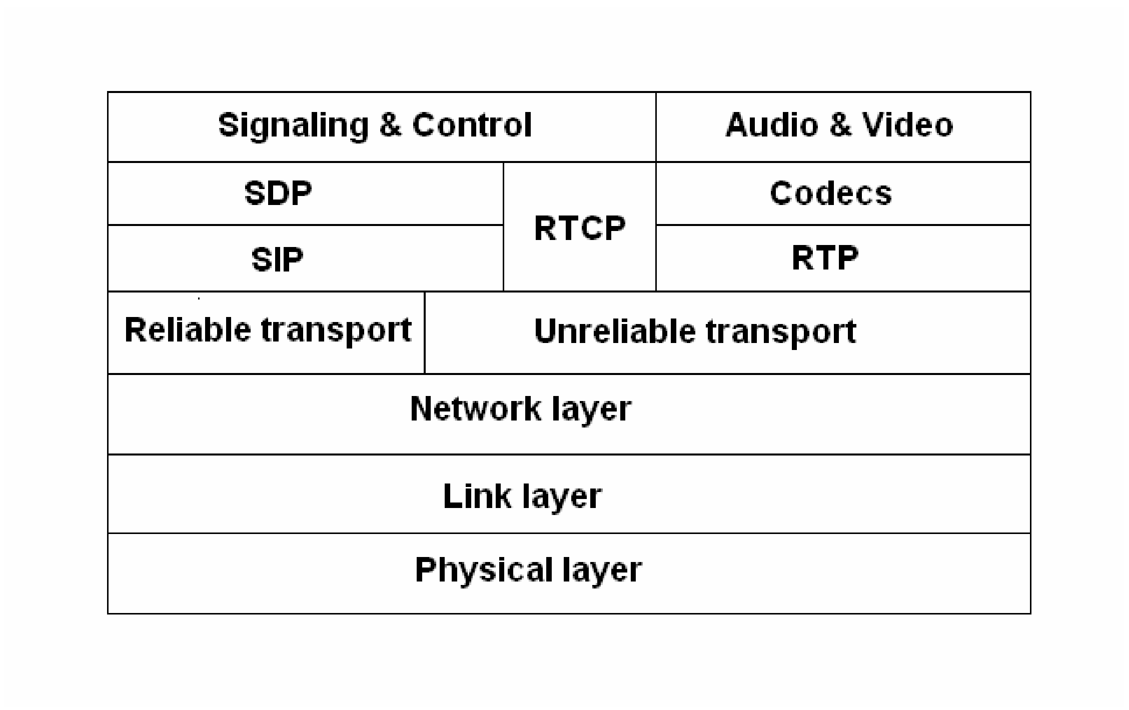


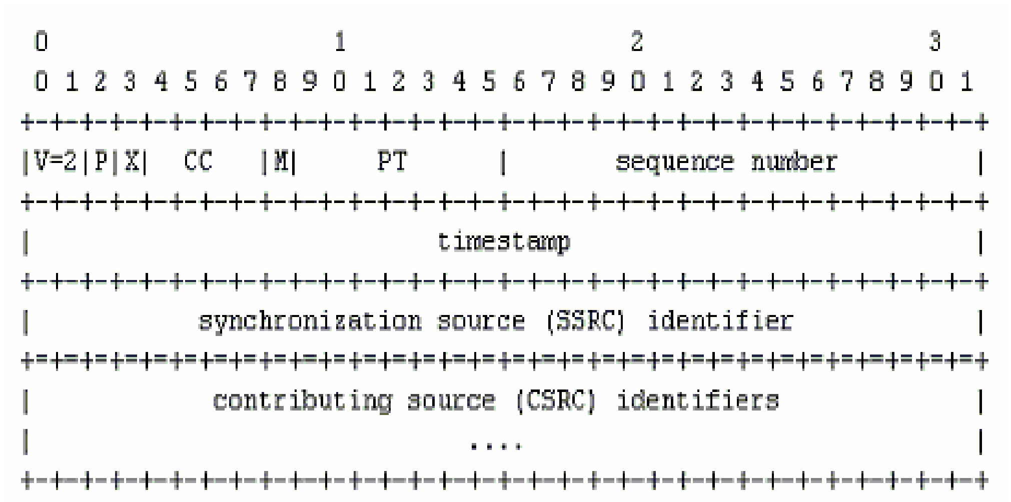
Figure 2: The VoIP protocol stack

As with other types of packets, RTP and RTCP packets are divided into headers and payloads. The most relevant information in the header of the RTP packet is:

1. The timestamp, which allows us to create an accurate reproduction of the input signal, as a receiver must know when the packet was produced. The receiver converts the data back to analog form and plays each sample at (almost exactly) the same temporal relationship to the surrounding samples as occurred in the input.

2. The sequence number so that data can be played in the same order as it was recorded and to detect missing or silence suppressed packets.

Figure 3 illustrates the format of the RTP packet.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: The RTP header structure

2.1.2. SIP, a signaling protocol for VoIP

The Session Initiation Protocol [9], is a signaling protocol for Internet conferencing, telephony, presence, event notification, and instant messaging. SIP was developed within the IETF MMUSIC (Multiparty Multimedia Session Control) working group.
SIP is a peer-to-peer protocol. The peers in a session are called User Agents.

SIP is a simple, ASCII-based protocol that uses requests and responses to establish communication among the various entities needed to ultimately establish a conference between two or more end points.

SIP's goal was initially to provide the basic signaling features necessary for a call: dialing a number, ring tone, hearing ring back tones or a busy signal, and completing a call. In this aspect, SIP looks like very similar to the Signaling System 7 protocol used by the PSTN, but SIP also provides other more advanced features:

- SIP allows for the establishment of a user's current location (i.e. translating from a user's name to their current network address).
- SIP provides for feature negotiation so that all of the participants in a session can agree on the features to be supported to enable communication among them.
- SIP is a mechanism for call management - for example adding, dropping, or transferring participants.
- SIP allows for changing features of a session while it is in progress.

SIP defines a set of signaling messages:

- INVITE is used by a user to request a call to another user
- ACK is used by the receiver to accept the call
- BYE is used to end a call
- OPTIONS is used to request a client's capabilities

- CANCEL is used to cancel a request
- REGISTER is used by a client to register with its registrar

Below is a typical session establishment. Once the SIP session is established, then RTP packets can be exchanged between the session participants (as shown in figure 2). (after the ACK).

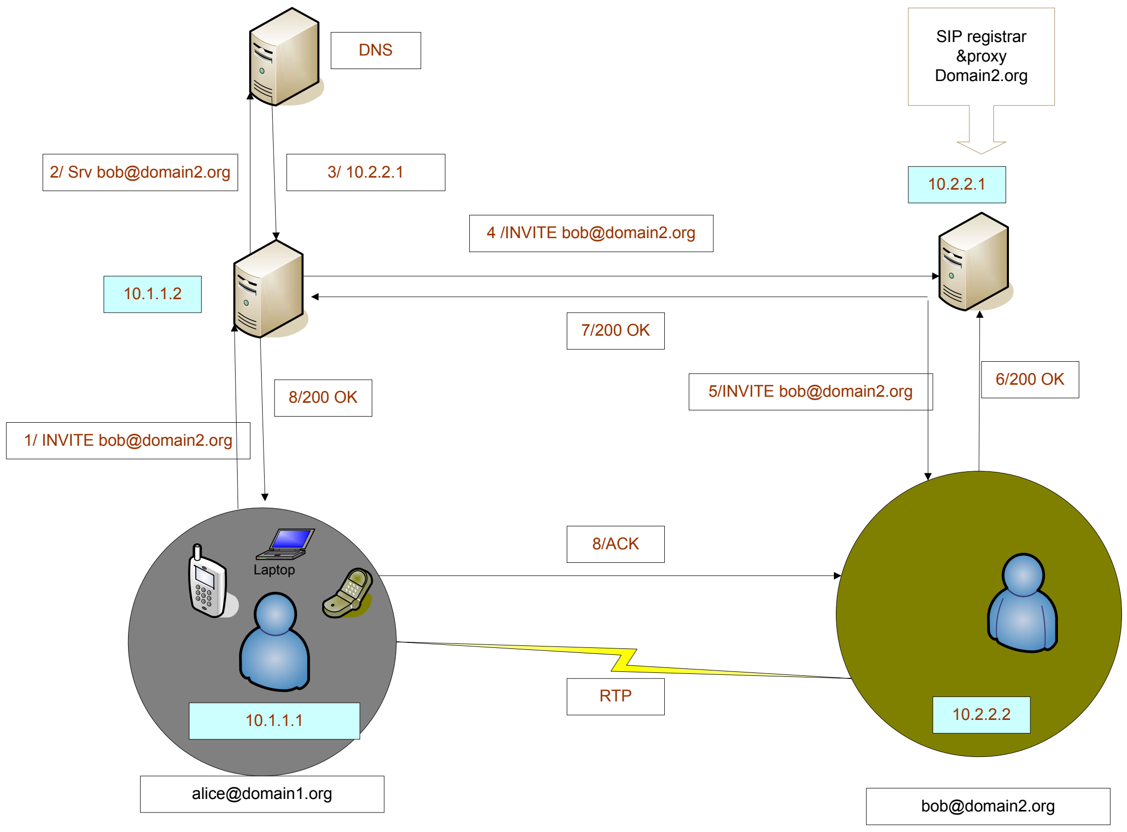


Figure 4: A typical session establishment in SIP

2.1.2.1 MIKEY

"MIKEY" stands for Multimedia Internet KEYing. MIKEY [10] is a key agreement protocol specially designed for protected multimedia exchanges. it supports three different authentication methods: Public Shared based, Public Key based, Diffie-Hellmann Based. We will focus on Diffie-Hellman, since it's the only method that provides perfect forward secrecy among these three methods.

Diffie-Hellmann:

The Diffie-Hellman protocol is a key agreement protocol (as shown in figure 5). It could be used by SIP to establish a key in order to encrypt subsequent data(such as control and media packets). Two parties, initiator and responder that initially do not share a secret construct a shared secret key:

- The initiator chooses a secret random value x and computes its Diffie-Hellmann public value (g^x) over a defined Diffie-Hellman group with generator g. It then sends the resulting DH public value, along with other information, in its Diffie-Hellman payload. The message is signed in order to prevent man-in-middle attacks.

- The responder, after checking the received message, sends a message similar to the initiator's one, with its own Diffie-Hellman public value and signature. The exchange provides both parties with the key material (g^{x*y}), which can not be calculated by an attacker intercepting g power x and g power y.

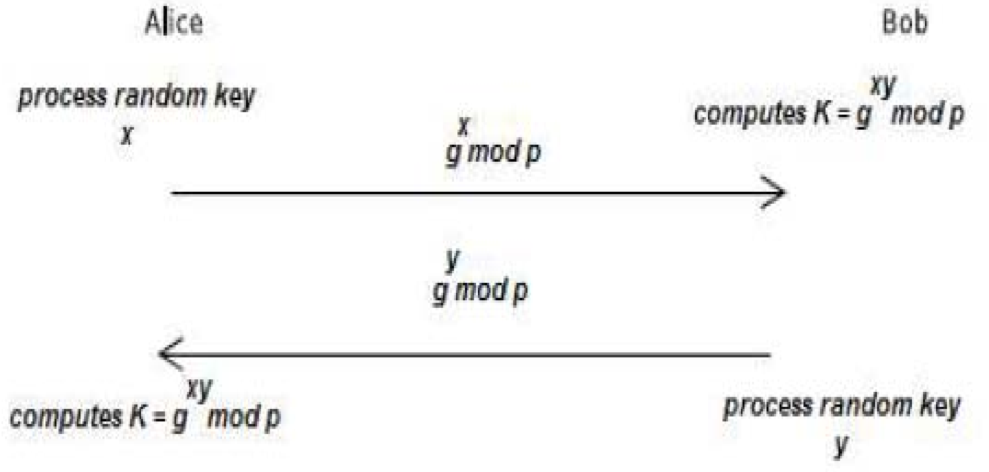


Figure 5: The Diffie-Hellmann method

Note that the generator g and the prime number p are public. Thus, mathematically, Diffie-Hellmann is based on the assumption that if p is a prime number, given g^x mod p, it is hard to compute x.

Among the advantages offered by this method is the symetric key contribution (i.e., both parties contribue to the key). Moreover, the key generation and a subsequent call could be completely decorrelated in time. For example, these half keys could be generated when the phone is charging (hence shifting the consumption of power from when the device is operating on batteries to when the device is operating from main power). We may even think of pre-computing a pool of (half-)keys. Generating a random or pseudo-random value is expensive for a processor, generating these values in a short amount of time is even more power consuming; while generating random values over long periods of time requires less power.

The principal condition for mobility is a suitable power source. Today's mobile device has a very limited source power (generally via a battery). As shown in the following graph, battery life decreases linearly as a function of the number of UDP packets sent and the time the system is operating. We can see that doing these operations leads to a battery life time of 3 hours, which is clearly not satisfactory when we consider that such a device would be used throughout a working day (therefore, 8-10 hours lifetime would be more satisfactory).

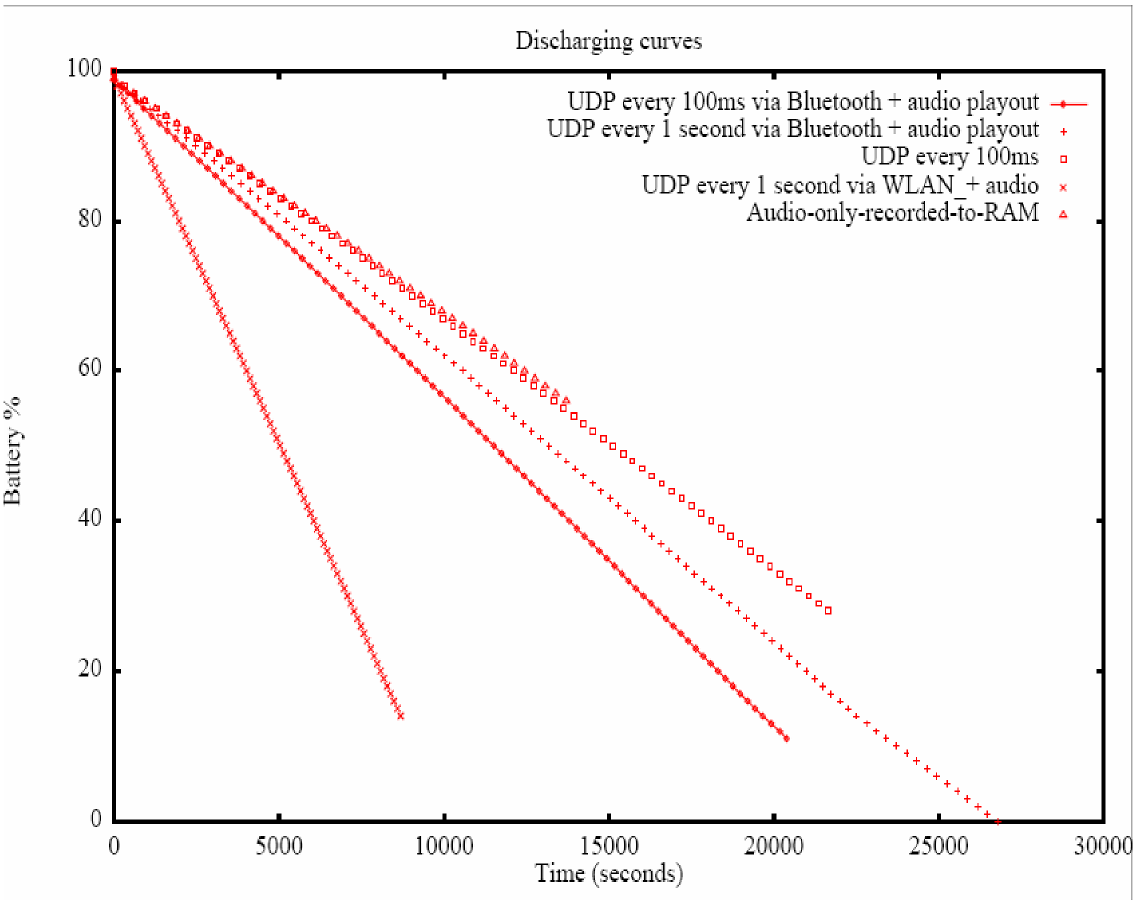


Figure 6: Battery discharge while playing audio and sending UDP packets. The figure is from G Q Maguire Jr. and it is used with his permission.

To increase the mobile device's batteries lifetime, an obvious solution is to limit the number of packets sent. However, it is no simply the sending of packets which is costly, but rather the whole process of communication, which includes generating the packets and listening for other traffic.

Next, we will explicitly consider when the Diffie-Hellman computation needed for generating a key for a new SIP session can be done. Figure 7 shows that the key - generation simply needs to be done before the indicated operations.

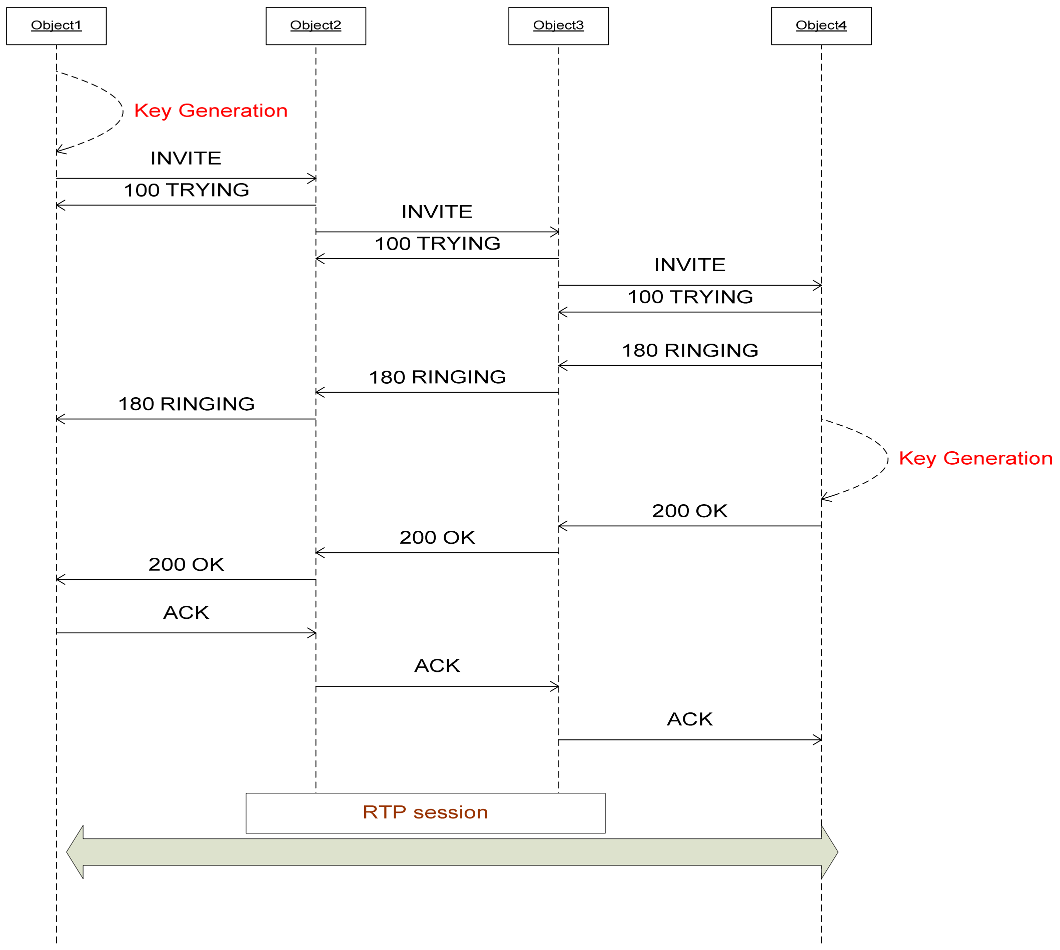


Figure 7: A scenario of SIP with Diffie-Hellmann

We note that the "key generation" process can be done by each party well before the session establishment, by using the pre-computation of random (half-)keys (as noted above - this could be while re-charging). Hence, removing this costs from the critical path of call establishment , with the benefit of the call setup still being secure but also faster (the call setup delay is reduced because the key generation phase of the key exchange does not have to take place at the time of a session establishment) and using less battery power!

Thus, battery lifetime could be increased if mobile devices would pre-compute and even pre-transfer information when energy is readily available (for example, when the mobile phone is charging). Kiratiwintakorn has proven that slower computations require less energy, since we have eliminated the problem of delay by utilizing pre-computation[11], we can even implement the pre-computation in such a way that it computes slower.

2.2. Working environment

2.2.1. Target platform: windows CE

Windows CE is a version of Microsoft's Windows operating systems designed for use in embedded systems that are very small and require real-time behavior. Windows CE is extremely customizable and targets several different processors. There is a tool called Platform Builder which lets you create a custom CE build for a specific device. The Pocket PC system is a custom CE build combined with device drivers for the relatively standardized Pocket PC hardware. The HP iPAQ 5550 supports Windows CE 4.2 and lower.

Windows CE, win32: a problem of subsetting

The Windows CE Application Programming Interfaces (APIs) are a subset of win32 APIs, and some of what is supported has reduced features set (fewer arguments for example). This means that porting an application from Windows CE to Windows is quite straightforward but porting in the reverse direction is not. When porting minisip to Windows CE, the primary issue has been the reduced functionality of certain APIs.

Another problem that we faced to is that there're significant limitations in exception handling. Windows CE does not support the normal C++ try/catch exception handling, one must use the win32 _try/_except and _try/_finaly. This issue will not be treated further in this master thesis but it remains as an impportant future work.

In his thesis [6], Billien notes that there's an issue with the minisip port to embedded Linux because Linux has very limited power management. Windows CE has a better energy management than Linux although Windows performs better in this area, Microsoft still declares: "Windows CE devices may have very limited energy resources […] Programs should be written to minimize energy consumption as much as possible." [12]

Another difference between the typical C++ environment and the Windows CE environment is the use of Unicode for all text strings. Not only does this mean that you need to include tchar.h, but you also need to use the TEXT macro for string literals (for example, TEXT("Your Text")). Special attention should be given to this difference: Generally, Windows applications use char which has a memory size of one byte whereas Unicode has 2 bytes.

Windows CE is a multithreaded operating system, but unfortunately semaphores are not supported. When porting an application to Windows CE, we will need to modify it to use other ways of coordination between threads. I have used in my tests critical sections for this purpose.

Finally, there're also memory limitations that we should take into consideration when writing our applications, as PDAs generally have much more limited amounts of memory, as it can be seen on table 1.

2.2.2. The .NET compact Framework

The **.NET** Framework created by Microsoft is a software development platform focused on rapid application development, platform independence and network transparency. .NET is Microsoft's strategic initiative for server and desktop development for the next decade. According to Microsoft, .NET includes many technologies that are designed to facilitate rapid development of Internet and intranet applications[13]. Microsoft .NET compact framework is a subset of Microsoft .NET targeting devices using the Windows CE operating system.

2.2.3. Embedded Visual C++

The compact framework version of 2003 doesn't support native C++, so we decided to use Embedded Visual C++. Later, following the release of Microsoft Visual Studio 2005, we moved to this Interactive Development Environment (IDE), experiencing far fewer porting issues.

The Microsoft eMbedded Visual C++ 4.0 tool delivers a complete desktop development environment for creating applications and system components for Windows CE .NET-powered devices. eMbedded Visual C++ targets small device, whereas Visual Studio 2005 is a generic platform. In order to target small devices, we need to choose "Smart Device Application" when creating a new project.

2.3. Minisip architecture

Minisip builds on 4 libraries (libmutil, libmnetutil, libmikey and libmsip) and one application layer directory (minisip). They are related to each other as shown in figure 8.

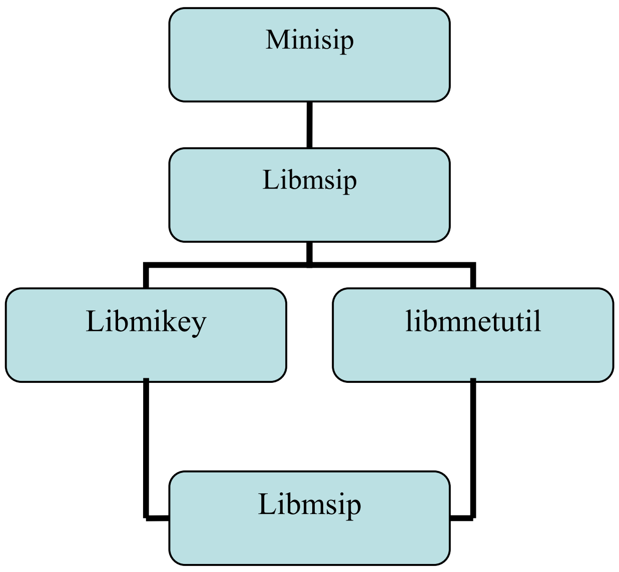


Figure 8: Minisip architecture

Libmutil:	A library providing convenient C++ utilities (AES encryption, HMAC-SHA1, certificate handling, memory object management, ...).
Libmnetutil:	A library providing convenient C++ network utilities (UDP/TCP/TLS sockets, IP addresses).
Libmikey:	A C++ library implementing the Multimedia Internet KEYing protocol.
Libmsip:	A C++ library implementing the SIP protocol.
Minisip:	The SIP user agent itself as an application layer.

Minisip is implemented following the Model/Vue/Controler design (MVC) methodology. This provides greater flexibility, allowing easier further development since all the modules are independent from each other.

For a more complete description of the minisip architecture, please refer to the minisip documentation [14].

CHAPTER 3: PORTING

3.1. Needed libraries

Since Windows CE lacks many of the necessary libraries, we needed to utilize a number of additional libraries. We included openSSL, PocketConsole (a simple console interface - as we did not initially implement a Graphical User Interface (GUI)), and a library to implement the usual C++ iostream. Each of these is described below.

3.1.1. OpenSSL

The OpenSSL Project [15] is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) protocols as well as a full-strength general purpose cryptography library.

In order to implement its additional security features, which allow end-to-end authentication, and protection of the media stream.minisip uses openSSL.
Porting minisip to iPAQ needs a port of openSSL. Fortunately, there's already a port. For more information, about the port of openSSL to iPAQ, refer to [16].

3.1.2. PocketConsole

Microsoft Windows CE operating system based PDAs do not implement text console. PocketConsole is an implementation of console driver allowing to have the text console available on Windows mobile. This patch allows using a console in the PDA running openSSL directly as can be done in a Windows XP operating System.

The minisip version running on windows uses the console for user interaction. Porting involves the most software reuse possible, thus having a console in the Pocket PC was useful. However, no console device is available by default on Pocket PC or Pocket PC 2002 platforms. Fortunately PocketConsole[17] is a software, which provides such a console device for Pocket PCs.

3.1.3. PortSDK

Port SDK is the PocketConsole Runtime Software Development Kit. It provides the following features:

- low / high level console API functions (show function list)

    - color support
    - unicode support
    - multiple screen buffers
    - installable CTRL-C handler
- extension of the coredll c-runtime library (show function list)
- Giuseppe Govi's STL port for Windows CE [18]
- Unix like SIGINT handling

Port SDK consists of the libraries for ARM, MIPS, SH3, x86 targets and the corresponding header files.

Also included in Port SDK is the application composer Pozard, the PocketConsole Application Wizard. Pozard integrates into the embedded VisualC++ IDE as a new application wizard: we use this to create pocket console applications.

3.1.4. Dojstream

Dojstream is a C++ iostream compatible library. It provides basic (file) IO for Microsoft's Embedded Visual Studio compilers, which lack an iostream library. The library was designed to coexist with the partial STL library for CE by Giuseppe Govi (see above).

3.1.5 Other Libraries

The Run-time Type Information (rtti): is a library that provides exception handling and dynamic cast support for C++.

wcecompat: As suggested by its name, the wcecompat this library tries to afford compatibility for Windows applications. It plugs the holes in the Embedded Visual C++ and Visual Studio 2005, making them more compatible with ANSI C++.

3.2. Coding issues

One source of problems is coding errors - here the porting was valuable because different compilers are stricter or more relaxed about different code than others. Therefore, using different compilers exposed errors which had been in the code for sometime, but had not been detected. This is despite the fact that some of the code has been running on several different platforms for several years.

The following are some samples of issues I encountered:

3.2.1.Build

On EVC++: when building a project, it doesn't compile just the necessary file. It also builds files that are part of the project, even if they are not used. The solution is to create a tree of the used files and to exclude the unused ones from the project.

3.2.2. Scope

The following error was very strange, and we were astonished to find it:

```
for (int i=0; i<10;i++){
        //do something
        if (condition){
                break;
        }
}

//Followed by the use of the variable i out of its local scope
i = i%5;
```

Although it should clearly generate a compiler error, surprisingly it "cleanly" compiles on g++ and EVC++ compilers.

A similar problem result when using the switch statement:

```
ObjectType myObject;
swicth(condition){

        myObject = new objectType();

        Case(condition1):
                myObject.method();
                break;
}
```

This Code generates an error on EVC++. EVC ++ assumes myObject instance is not affected, whereas the object instantiation is done indeed by
myObject = new objectType();

3.2.3. Polymorphism

Initially, we faced many problems when trying to use polymorphism. This important feature of object oriented languages was not resolved. Using RTTI solved most of the problems, but not all. For the rest, we avoided using dynamic casts.
Another issue we had with RTTI is that we were constrained to use the "/GR" compiler flag, which enabled RTTI in the environment. This was not needed on Visual Studio IDE (RTTI was used by default if linking against it).

3.2.4. Other problems

Exceptions

As noted in section 2.1.1., Windows CE exception handling is fundamentally different than the Visual C++ one. Thus, I need to disable exceptions in order to avoid related errors. I used the compiler flaf "/GX(-)", to disable exception handling (otherwise I could not build it).

Environment variables

Windows CE doesn't support environment variables. The ported openSSL libraries call environment variables to process timestamps, random values, keys…the solution used was hardcoding these values, but this may reduce the portability of the software. Thus, in the future we may need to store some environment variable in the configuration file of minisip.

GUID

Short for Globally Unique Identifier. It is a unique 128-bit number that is produced by the Windows operating systems or by some Windows applications to identify a particular component, application, file, database entry, and/or user. For instance, a Web site may generate a GUID and assign it to a user's browser to record and track the session. The GUID was not recognized unless we included "initguid.h". However the version of the code running on Windows and Linux didn't need this include.

Linking problems

Most link problems we faced were solved by linking against the libraries listed above. But sometimes, we didn't manage to resolve the link problems (Although we checked that the symbols were exported correctly). In this particular case, we decided to use a not so elegant solution: we imported the files exporting the "unresolved symbols" to our project.

3.3. The low level sound API

Windows provides DirectSound[19] as a high level API to play and record sound. Unfortunately, Pocket PC doesn't provide "dsound.lib". This is very confusing since the Microsoft Developer Network (MSDN) documentation suggests that DirectSound is part of the Windows CE platforms. A simple test for using this dynamically loadable library (dll) as a delay loaded one shows that this library is not present on Windows CE 4.2. This fact was confirmed by searching for this file using Active Sync (setting the "show hidden files" option).

An extensive web search for "dsound.lib" or "dsound.dll" targeting Pocket PC 2003 platforms didn't discover any instances. We finally contacted some Microsoft developers of the Pocket PC team who confirmed that this library was not available.

Thus, in this case software reuse was not possible and I had to write a new sound module using an available API on Windows CE 4.2 (note that windows CE 5.0 and higher provide "dsound.lib", but we could not use this OS since the HP iPAQ doesn't support it).

3.3.1. The waveform API

The waveForm API provides a way of playing and recording data with two sub APIs : waveOut and waveIn. There're many tutorials on the web indicating how to use this APIs[20] but very few deal with real time recording and playing - as most examples assume that you are recording to a file or playing a prerecorded file.

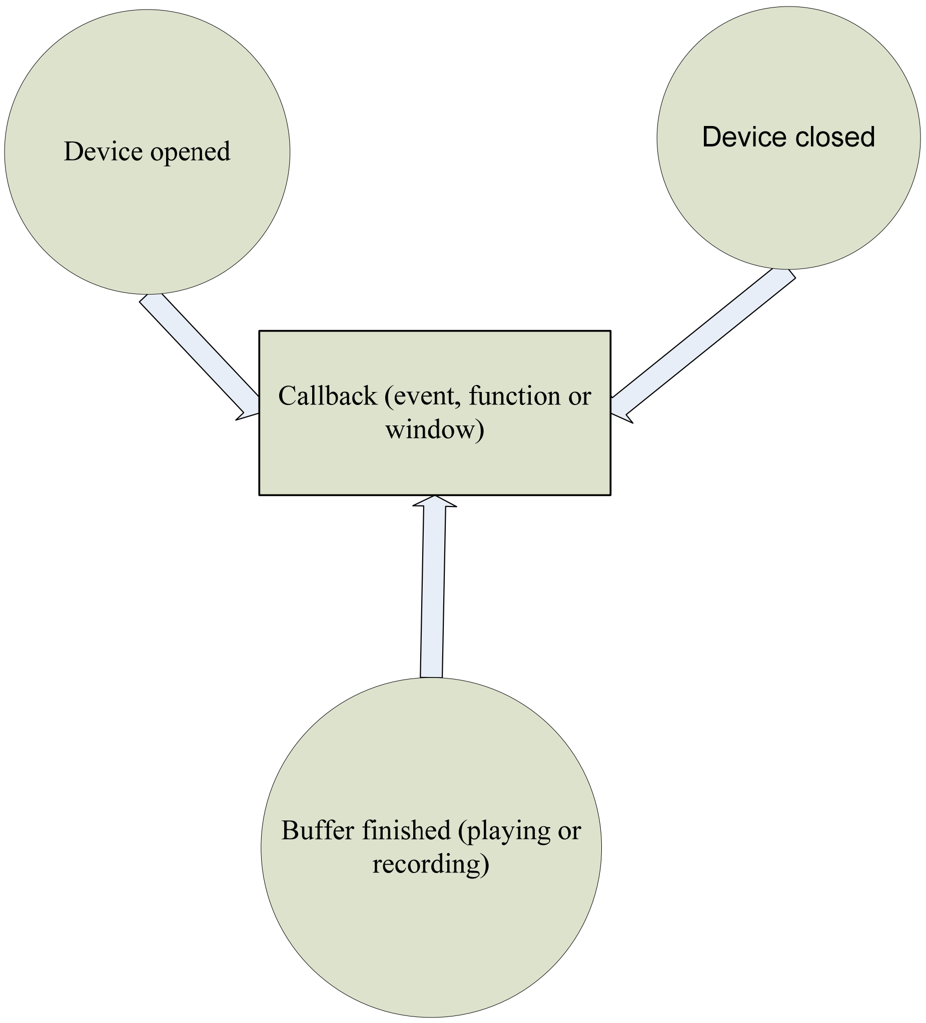

Device opened
Device closed
Callback (event, function or window)
Buffer finished (playing or recording)

Figure 9: the waveform API logic

Following is a brief overview of the basic API logic (illustrated in figure 9).

The callback is triggered every time one of these particular events happens:

-Device opened: this can be through the waveOutOpen or waveInOpen function.

-Device close: waveOutClose or waveInClose

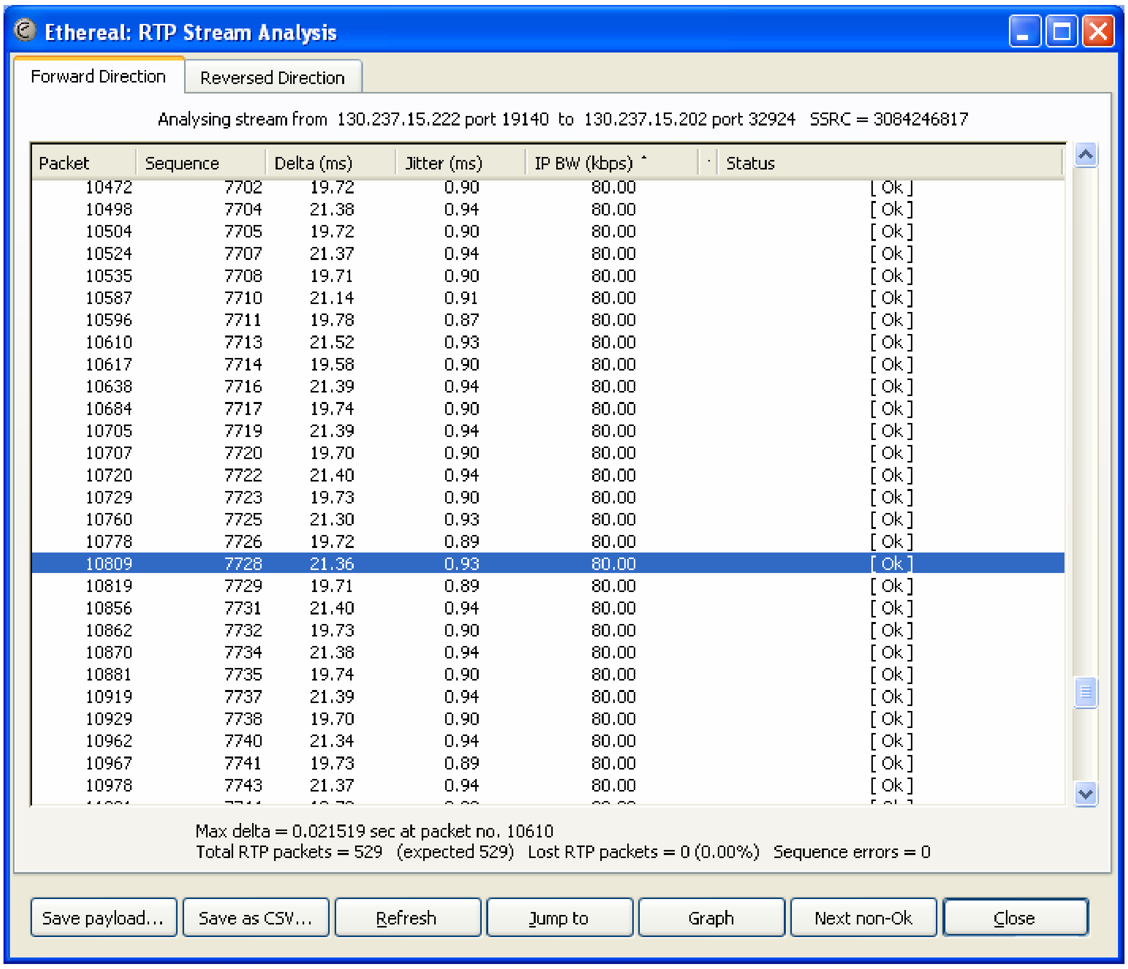
-Buffer finished playing/recording: the buffers are enqueued using waveInAddBuffer for recording and waveOutWrite for playing. This event happens when the audio device finishes reading or writing into the buffer.

Note that it's also possible to use the API without callback when specifying CALLBACK_NULL as a parameter of waveOutOpen/waveInOpen. But then, our application wouldn't have any information of the state of the audio device, with the risk of losing the synchronization.

Using the API

A simplistic approach was to play and send buffers as soon as they were received (in minisip an RTP payload corresponds to 20 milliseconds of sound). This is an erroneous method since our application relies on the network delivering the packets at the expected time. The playback is consuming data from the current packet, if the next packet arrived exactly as our device finishes playing data from the current buffer, playback will continue uninterrupted. However, if the next packet is delayed, playback must be paused until the next packet arrives and we may hear gaps. Thus, playback cannot start when packet first arrives.

In figure 10, we can see the jitter for an RTP communication captured with Ethereal[21].



Analysing stream from 130.237.15.222 port 19140 to 130.237.15.202 port 32924 SSRC = 3084246817

Packet	Sequence	Delta (ms)	Jitter (ms)	IP BW (kbps)	Status
10472	7702	19.72	0.90	80.00	[Ok]
10498	7704	21.38	0.94	80.00	[Ok]
10504	7705	19.72	0.90	80.00	[Ok]
10524	7707	21.37	0.94	80.00	[Ok]
10535	7708	19.71	0.90	80.00	[Ok]
10587	7710	21.14	0.91	80.00	[Ok]
10596	7711	19.78	0.87	80.00	[Ok]
10610	7713	21.52	0.93	80.00	[Ok]
10617	7714	19.58	0.90	80.00	[Ok]
10638	7716	21.39	0.94	80.00	[Ok]
10684	7717	19.74	0.90	80.00	[Ok]
10705	7719	21.39	0.94	80.00	[Ok]
10707	7720	19.70	0.90	80.00	[Ok]
10720	7722	21.40	0.94	80.00	[Ok]
10729	7723	19.73	0.90	80.00	[Ok]
10760	7725	21.30	0.93	80.00	[Ok]
10778	7726	19.72	0.89	80.00	[Ok]
10809	7728	21.36	0.93	80.00	[Ok]
10819	7729	19.71	0.89	80.00	[Ok]
10856	7731	21.40	0.94	80.00	[Ok]
10862	7732	19.73	0.90	80.00	[Ok]
10870	7734	21.38	0.94	80.00	[Ok]
10881	7735	19.74	0.90	80.00	[Ok]
10919	7737	21.39	0.94	80.00	[Ok]
10929	7738	19.70	0.90	80.00	[Ok]
10962	7740	21.34	0.94	80.00	[Ok]
10967	7741	19.73	0.89	80.00	[Ok]
10978	7743	21.37	0.94	80.00	[Ok]

Max delta = 0.021519 sec at packet no. 10610
Total RTP packets = 529 (expected 529) Lost RTP packets = 0 (0.00%) Sequence errors = 0

Figure 10: observing jitter with Ethereal

The network introduces this variation in delay (Jitter). To avoid having gaps in the output for each delay; the receiver should not start playback when the first data arrives. Instead, incoming data is placed in a buffer known as playback or jitter buffer.

Note that for waveIn, there's no need to buffer since we don't need to add delay, we can simply read blocks of samples and pass them to the minisip code. Figure 11 illustrates the flow of data in the application. The application supplies a number of data buffers to the audio capture device (waveIn). The device fills each buffer with digitized data and returns it to the application. The application inserts the filled buffer into the transmission queue, and after the data is output to the network, the application returns the empty buffer to the waveIn device, starting the process all over again.

At the playback computer, packets of audio data are received from the network into buffers taken from the "Free List". The application inserts the filled buffers into the Playback List. The Playback List is used to assemble the buffers in the proper order needed for playback. After ordering the buffers, the application hands the buffers to the playback device, waveOut. After playing the data in the buffers, waveOut returns the buffers to the application for reinsertion into the Free List.

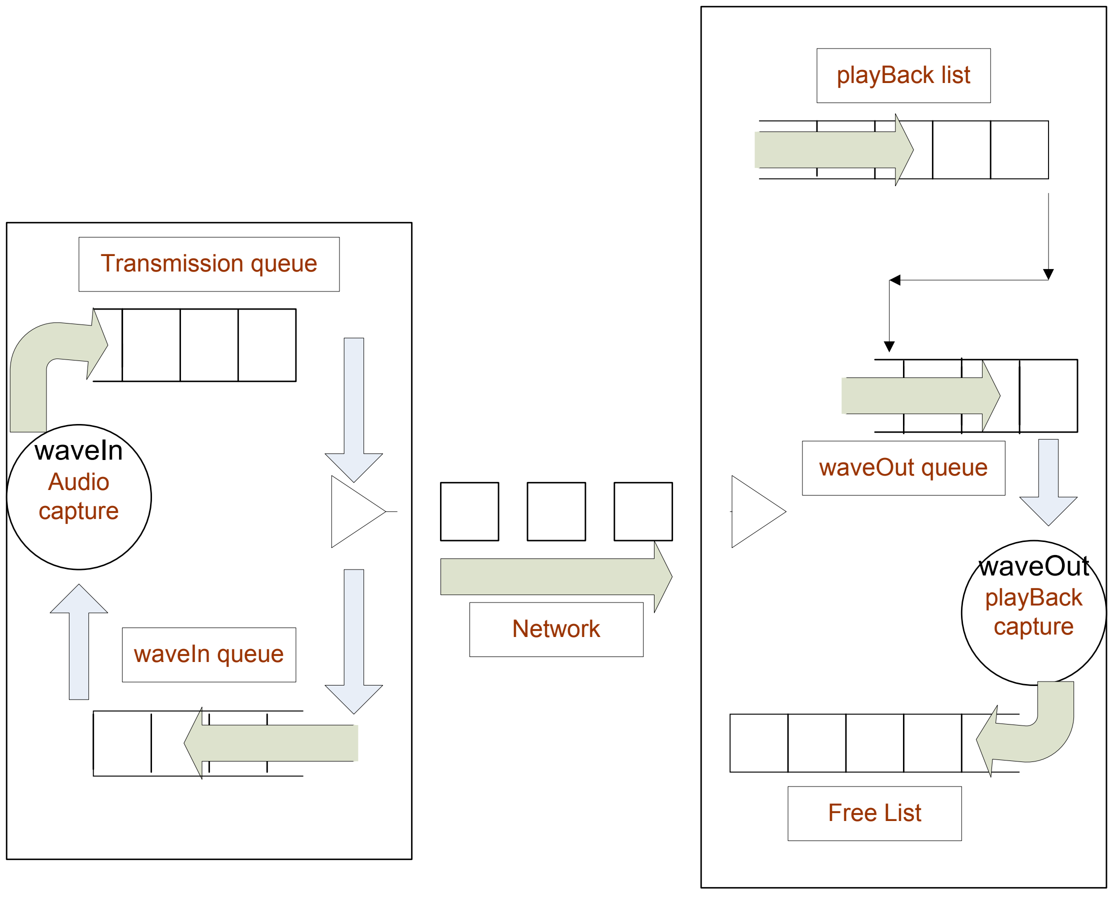


Figure 11: Data Flow in our VoIP application

For an implementation of playback, see the Appendix A.

After implementing this code, there were still some synchronization issues. In fact, when making very short measures of times when the thread calls write (in order to play sound) method, we noticed that these calls were not regularly spaced.

The following function processes determines the difference in time between two consecutive RTP packets arrival. Thus, the thread RTPReceiver is not synchronized and doesn't play at a regular pace. Because Windows CE is a real time system, we should be able to fix this.

```
int getJitter(){
        static unsigned short millisec=0;
        static int count=1;
        static int average=0;
        int res;

        SYSTEMTIME st;
        GetSystemTime(&st);
        if (st.wMilliseconds-millisec<0)
                res= 1000-millisec+st.wMilliseconds;
        else
                res= st.wMilliseconds-millisec;
        millisec=st.wMilliseconds;
        return res;
}
```

Below are some results that may be relevant:

Packet number	Differerence of time from packet i-1 in milliseconds
1	1
2	15
3	1
4	17
5	15
6	1
7	0
8	1
9	14
10	16

Table 2: Thread synchronization issue

In “Streaming Real Time Audio” [22], the author uses 2 lists, one contains the free buffers and the other one the enqueued buffers that are full of digital sound that needs to be sent to the DAC.

CHAPTER 4: EVALUATION

This project required many choices that appeared to be crucial for the project path.

4.1. Managed versus unmanaged code

Managed code is code that has its execution controlled by the .NET Framework Common Language Runtime (CLR). Before the code is run, the intermediate Language in which the code has been converted to is compiled into native executable code. Because the code is compiled in a managed way (by the CLR), there're some guarantees and advantages such as garbage collection, type safety...Although there's a performance penalty as compared to unmanaged code.

On the other hand, unmanaged code provides no guarantees. Unmanaged code compiles to binary files, so the system cannot know what the application is doing, but runs faster. Thus, it is particularly important to notice that while most of the code is in native C++, porting this code in order to make it "managed" would require a lot of effort and the benefits of this process for this particular thesis are doubtful. Minisip has been developed in C++ for two main reasons:

- Object oriented design provides some advantages such as code reuse and maintenance
- C++ performance in comparison with other languages that run on virtual machines (say Java Virtual Machine or CLR).

Because of this second reason, it seems reasonable not to port the code into managed C++, although it can be interesting to consider this possibility in a future in order to make the code compatible with all platforms.

4.2. The IDE

Once the previous decision was made, we had to start porting minisip. Thus, we needed a compiler for Windows CE platforms. There're several compilers for Windows CE. I naturally thought about Visual Studio IDE but VS 2003 unfortunately didn't provide a compiler for native C++ targeting windows CE. After that, I naively chosed Embedded Visual C++, which we used until Visual Studio 2005 was available. But by this time, there was only one module left for to port (minisip itself). VS 2005 appeared to be more flexible than EVC++ and its compiler gave us fewer warnings and required fewer fixes.
This was particularly frustrating knowing that there were minor differences with VS2005Beta (concerning the compact framework features). Thus, the choice of the compiler represented a significant drawback for this project.

4.3. Open Source software

Free and Open Source software is great but it sometimes lacks of reliability. Minisip builds on openSSL in order to manage the basic cryptography functions needed. When starting the project, there was a new port of openSSL to iPAQ. This was valuable considering that we didn't have to implement this module so it saved a considerable time. Later, when building upon this library, problems with this module appeared (i.e., its use of environment variables). Moreover, we found out that the latest version shipped would not always compile.

4.4. The planning

The porting was a necessary step to optimize Diffie-Hellman method on iPAQ, since this optimization wouldn't make sense in a desktop were energy is not a scarce resource, but its cost were clearly underestimated. Afterwards, I think I could not estimate its cost earlier since I was not aware of the complexity of the software; if I could have made an accurate estimation of its cost, I may have had considered starting the porting process earlier.

The porting method

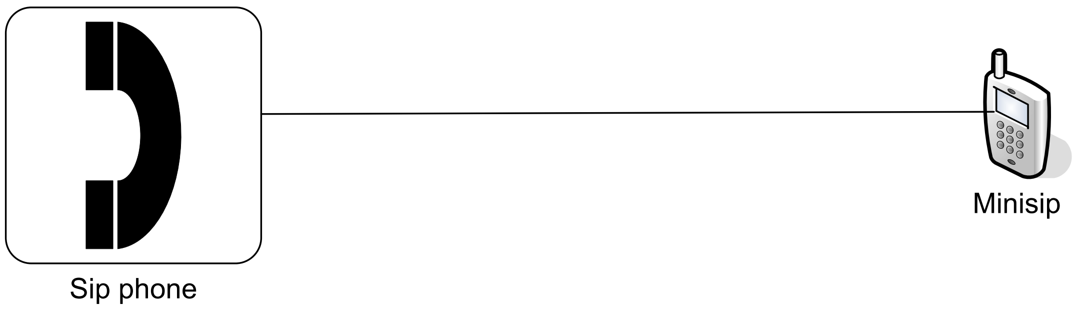
The porting method was quite straightforward:

- creating the module on the IDE.
- importing all the files into the project.
- setting the project configuration using the win 32 makefile settings: precompiled definitions, linked libraries.
- build
- incrementally correction of errors.
- check the platform dependencies with "dependency walker"[23]
- deployment and test.

4.5. Test

The tests involved testing our application on Windows and then on iPAQ (due to the subsetting paradigm we've discussed earlier), because debugging an application on the desktop is much easier than doing so on the iPAQ.

The main test was testing the sound API. The testbed is described below:



We used a VoIP SIP phone Cisco IP Phone 7940 Series. This phone was calling the minisip user agent (by this way, we didn't have to register our software).

Other tests: due to the gaps in the voice, we create a sinusoidal sound which had the sampling frequency we used.

4.6. Results from testing

When porting, some modules such as libmnetutil were provided by their developers with some test programs. These tests allowed gaining confidence concerning the correctness of the results. Otherwise, we were debugging the application and checking it had the expected behavior. Finally, on Windows, Minisip was running without main concerns.

The sound quality was qualitatively tested. Initially it had a lot of gaps and the sound was inaudible, with really poor quality. After adding the buffering to our scheme, the quality improved substantially, and the voice was rather clear but with average sound quality, thus there are still some improvements to make.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1. Conclusion

The main contribution of this thesis was porting most of minisip to both the Pocket PC and to Windows platforms in general. This porting process included designing and writing code to provide audio input and output using an audio low level API, as this API is available on both platform. In addition, I investigated lowering the energy consumption of computing random values for use in a Diffie-Helman exchange, as energy savings are particularly important for the Pocket PC. This design also could reduce the run-time delay at the time of an INVITE, thus reducing the delay cost of security for the user(s).

The implementation and evaluation remain for future work. While the ambition of this thesis was greater than what was completed, the results are significant, especially given the bounded time of this project. Already others are building upon this code.

In addition, I have gained some valuable insights (both theoretical and practical) in this project:

- Learning VoIP technology: RTP, SIP and RTCP.
- Learning a new language: C++
- I was faced with a real life project - with all the related difficulties, for example the lack of documentation and support.
- Debugging and importance of software design.

5.2. Future work

Obviously, minisip needs to be a user friendly software with a fancy graphical user interface (GUI). The console is far from providing this. In the appendix B, there's a brief tutorial about how to call Dynamic Link Libraries from C# code, avoiding the limitation of P/Invoke. C# GUIs are easy to implement (using simple drag and drop operations). Moreover, the decrease of performance due to this in not relevant since it only deals with the GUI side of the application and not the core network logic.

Another interesting porting project would be to port Minisip to .NET: managed code benefits from the advantages of the managed code (safety, garbage collection, portability) and the richness of the .NET libraries but once again there's a sensitive decrease of performance depending on the APIs used. Portability issues would be almost solved (except for the APIs used) since .NET uses CLR which has an open specification and is available on all popular platforms.

Work has already begun on the implementation and evaluation of the energy and time savings based upon when the random numbers and partial Diffie-Hellman computation is done as per the design proposed in this thesis.

APPENDIX A

The “jitterControl” method waits until it has some buffers queued in the “playList” to start playing them in a loop, thus because there’re no interruptions while playing them, we may not hear the little gaps…

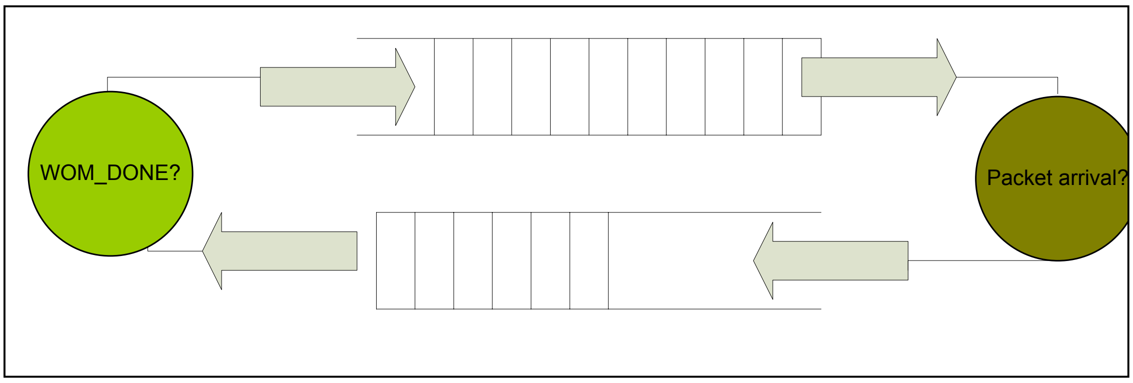


Figure: the playback logic.

```
#include <windows.h>
#include <iostream>
#include <mmsystem.h>
#include <list>

#define BLOCK_SIZE 1920
#define NUM_BLOCKS 25
#define THRESHOLD 10        //delay threshold
#define PLAYBACK_THRESHOLD 5 // Playback threshold

typedef unsigned char  byte_t;


/* I use two buffer lists: -freeList containing free buffers
                           -playList: containing all the buffers queued in oredr to be played

WOM_OPEN is sent to waveOutProc when the device is opened using the
waveOutOpen function..
*/
```

```
typedef unsigned char  byte_t;


class CRecvBuffer {

public:

        WAVEHDR m_WaveHeader;

        MMRESULT Prepare(HWAVEOUT m_hWaveOut) {
                ZeroMemory(&m_WaveHeader, sizeof(WAVEHDR));
                m_WaveHeader.dwBufferLength = BLOCK_SIZE;
                m_WaveHeader.lpData = (char *)malloc(BLOCK_SIZE);
                return waveOutPrepareHeader(m_hWaveOut,
&m_WaveHeader,sizeof(WAVEHDR));
        }
        MMRESULT Unprepare(HWAVEOUT m_hWaveOut) {
                return waveOutUnprepareHeader(m_hWaveOut, &m_WaveHeader,
                        sizeof(m_WaveHeader));
        }
        MMRESULT Add(HWAVEOUT m_hWaveOut) {
                if(waveOutWrite(m_hWaveOut,
&m_WaveHeader,sizeof(WAVEHDR))==MMSYSERR_NOERROR){
                        printf("writing to device\n");
                        //exit(1);
                }
                return MMSYSERR_NOERROR;
        }

};

typedef std::list<CRecvBuffer*> CRecvBufL;


class MyAPI{

public:
        static MyAPI* instance;
        HWAVEOUT device;
        CRecvBuffer m_aOutBlocks[NUM_BLOCKS*2];
        CRecvBufL playList;          //List of playback buffers
        CRecvBufL freeList;          // List of free recv buffers
        bool m_fDelay;               // In delay mode?
        int count;                   // items in playback queue
        bool waveOutClosing;         // Stopping playback?
```

```
private:



          MyAPI(){
          }


public:





          static MyAPI*  getInstance(){
                    if (instance==NULL){
                              instance= new MyAPI();
                    }
                    return instance;
          }






void openDevice(){
                                        HWAVEOUT
          waveOutDevice;
                    WAVEFORMATEX                                        waveOutputFormat;
                    ZeroMemory( &waveOutputFormat, sizeof(WAVEFORMATEX) );
                    waveOutputFormat.wFormatTag = WAVE_FORMAT_PCM;
                    waveOutputFormat.nChannels = 1;
                    waveOutputFormat.nSamplesPerSec = 48000;
                    waveOutputFormat.nAvgBytesPerSec=48000*1*2;
                    waveOutputFormat.nBlockAlign=2;  // number of bytes in smallest
"atomic unit" - nChannels*wBitsPerSample/8
                    waveOutputFormat.wBitsPerSample=16;

                    waveOutOpen(&waveOutDevice, WAVE_MAPPER,
&waveOutputFormat,(DWORD_PTR)MyAPI::waveOutProc,(DWORD_PTR)NULL,CAL
LBACK_FUNCTION);
                    MyAPI* myApi = MyAPI::getInstance();



          }
```

```
void jitterControl(){

        CRecvBuffer *pBuffer;
        if (m_fDelay) {
                if (playList.size() >= THRESHOLD){
                        // Start playback if enough buffers received
                        printf("m_lpPlayBufs.size() >= THRESHOLD\n");
                        m_fDelay = false;
                        for (int i = 0; i < THRESHOLD; i++) {
                                pBuffer = playList.front();
                                printf("preparing and adding\n");
                                pBuffer->Prepare(device);
                                pBuffer->Add(device);
                                count++;
                        }
                }
                return;
        }
        if(count==0){
                                // Start delay mode if we run out of buffers
                m_fDelay = true;
                return;
        }
        for (;;) { // Playback as many as possible without gaps
                printf("playing back in loop\n");
                if (playList.empty()){
                        printf("m_lpPlayBufs.empty()\n");
                        return;
                }
                pBuffer = playList.front();
                pBuffer->Prepare(device);
                pBuffer->Add(device);
                count++;
                playList.pop_front();
                continue;
        break;
        }
}




void onBufferDone(){

                CRecvBuffer *bufferPlayed;
                bufferPlayed = (CRecvBuffer *)playList.front();
                playList.pop_front();
                freeList.push_back(bufferPlayed);
                if (!waveOutClosing)
                        jitterControl();

                else
                        waveOutClose(device);
        }



void onOpening(HWAVEOUT waveOut){
                MMRESULT err;
                device=waveOut;
                for (int i = 0; i < NUM_BLOCKS*2; i++) { // Setup free list
                        printf("creating blocks");
                        m_aOutBlocks[i] = *(new CRecvBuffer());
                        err=m_aOutBlocks[i].Prepare(waveOut);
                        switch (err){
                                case MMSYSERR_INVALHANDLE:
                                        printf("invalid handle \n");
                                case  MMSYSERR_NODRIVER:
                                        printf("MMSYSERR_NODRIVER \n");
                                case MMSYSERR_NOMEM:
                                        printf("MMSYSERR_NOMEM\n");
                        }
                        freeList.push_back(&(m_aOutBlocks[i]));
                }
        }




void static CALLBACK waveOutProc(HWAVEOUT hwo, UINT uMsg,DWORD_PTR
dwInstance, DWORD dwParam1,DWORD dwParam2){
                 //better make myAPI as a singleton
                MyAPI* myApi=MyAPI::getInstance();

                switch (uMsg){

                        case WOM_OPEN:
                                printf("device opened");
                                break;
                        case WOM_DONE:
                                myApi->onBufferDone();
                                break;
                }
        }
```

```
int write(byte_t *buffer, int nSamples){

                CRecvBuffer *myBuffer;

                if (freeList.empty()) //overflow
                        return nSamples;
                if (waveOutClosing)
                        return nSamples;
                myBuffer=freeList.front();
                for (int i=0; i<BLOCK_SIZE;i++){
                        myBuffer->m_WaveHeader.lpData[i]=buffer[i];
                }
                freeList.pop_front();
                playList.push_back(myBuffer);
                jitterControl();
                return nSamples;
        }

        void close(){
                waveOutClosing=true;
        }
};
```

APPENDIX B

In this part I will show how to build a DLL in C++ targeting windows CE platforms and use these DLL's exported functions in a managed language like C#.
Launch EVC or Visual Studio 2005.
-For VS 2005:
Click on File->New.
Go to project Tab
. Select Visual C++->Smart Device->"Win32 Smart Device Project"
Provide exampleDll as the project name.
Then on application Settings, select "DLL" for application type and "empty project" in additional options.
In the project settings-> preprocessor, add "MYDLL_EXPORTS" preprocessor command
-For EVC++:
Click on File->New, project creation wizard will come up.
Go to 'Projects' tab.
Select 'WCE Dynamic-Link Library'
Provide 'NativeFunctions' as the project name and select a directory where the projects need to be created.
Select Win32 (WCE x86) under CPU's.
Select 'A simple Windows CE DLL project' and click finish.
In the project settings-> preprocessor, add "MYDLL_EXPORTS" preprocessor command
Content of exampleDll.h:

```
#ifdef NATIVEDLL_EXPORTS
#define DLLEXPORT _declspec(dllexport)
#else
#define DLLEXPORT _declspec(dllimport)
#endif
#ifdef __cplusplus
extern "C" {
#endif
DLLEXPORT int function();
#ifdef __cplusplus
}//end extern "C"
#endif
```

Content of exampleDll.cpp:

```
#include <iostream>
#include "NativeFunctions.h"
int function()
{
return 0;
}
```

Then Build and deploy the project to a directory in pocket pc.

Using dlls from managed code(C#):

In Visual Studio 2005, select Smart Device Application template.
Add the following lines in the beginning of your class:

```
using System.Runtime.InteropServices;
[DllImport("exampleDll.dll")]
public static extern int function();
/*Drag and drop a button on the panel and go to a method “button1_click”, you can then
add the following:*/
int number = function();
System.Windows.Forms.MessageBox.Show(number.ToString());
```

Finaly, right click on your project name and add coredll.dll. and exampleDll.dll to the project(“add an existing item”).
Unfortunately, platform invoke cannot invoke exported classes. The solution is to wrap the class methods in functions:
Example: assume we have a class:

```
Class MyClass{
MyClass(){}
~MyClass(){}
int method(){ return 1;}
}
```

See the code above for an example of wrapping a class:

```
#define MYWRAPPER _declspec(dllexport)
extern "C" {
MYWRAPPER int wrapperFunction();
}//end extern "C"
int wrapperFunction()
{
int returnValue = 0;
MyClass * instance = new MyClass ();
returnValue = instance->function();
delete instance;
return returnValue;
}
```

Then just change:

```
[DllImport("exampleDll.dll")]
public static extern int function();
```

to

```
[DllImport("WrapperDll.dll")]
public static extern int wrapperFunction(); /*and use this function as we already did
earlier */
MyAPI* MyAPI::instance=NULL;
```

References:

[1] Gerald Q. Maguire Jr.'s notes on using the HP iPAQ h5550.
http://web.it.kth.se/~maguire/ipaq-notes.html Latest update: November 29, 2005

[2] HP iPAQ h5500 Pocket PC series.
http://h20000.www2.hp.com/bizsupport/TechSupport/DocumentIndex.jsp?locale=en_US&contentType=SupportManual&docIndexId=3124&prodTypeId=215348&prodSeriesId=322916&lang=en&cc=us Latest update: June 26, 2005.

[3] Minisip.
www.minisip.org Latest update: March 11, 2006

[4] Florian Maurer, Push 2 Talk in Voice over IP decentralized.
http://push2talk.flohweb.ch Latest update: August 30, 2004

[5] Carlos Marco Arranz, "IP Telephony: Peer-to-peer versus SIP", M.S. Thesis, Royal Institute of Technology, June 13, 2005.
ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/050613-Carlos_Marco_Arranz_Thesis-with-cover.pdf

[6] Johan Billien, Key agreement for secure Voice over IP, M.S. Thesis, Royal Institute of Technology, December 15, 2003.
ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/031215-Johan-Bilien-report-final-with-cover.pdf

[7] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", IETF RFC 1889, January 1996.
http://www.ietf.org/rfc/rfc1889.txt

[8] J. Postel, "User Datagram Protocol", IETF RFC 768, 28 August 1980.
http://www.ietf.org/rfc/rfc0768.txt

[9] J. Rosenberg and H. Schulzrinne, "Reliability of Provisional Responses in the Session Initiation Protocol (SIP)", IETF RFC 3262, June 2002.
http://www.ietf.org/rfc/rfc3262.txt

[10] J. Arkko, V. Torvinen, G. Camarillo, A. Niemi, and T. Haukka, "Security Mechanism Agreement for the Session Initiation Protocol (SIP)", IETF RFC 3329, January 2003.
http://www.ietf.org/rfc/rfc3329.txt

[11] Phongsak Kiratiwintakorn, "Energy efficient security framework for wireless local area networks", M.S. Electrical Engineering, University of Kansas, April 2000.
http://etd.library.pitt.edu/ETD/available/etd-04052005-141245/unrestricted/phongsak.kiratiwintakorn-20050620.pdf

[12] Porting Windows 95-based Programs to Windows CE.
http://www.microsoft.com/technet/prodtechnol/wce/deploy/porting.mspx
Last read: March 12, 2006

[13] http://encyclopedia.thefreedictionary.com/Microsoft+.NET
Last read: March 12, 2006

[14] Minisip documentation.
http://www.minisip.org/doc/ Latest update: June 06, 2006

[15] OpenSSL.
http://www.openssl.org/ Latest update: December 12, 2004

[16] openSSL port to pocket pc.
http://www.it.uc3m.es/pervasive/wce_lite_compat/ Last read: March 12, 2006

[17] Pocket Console.
http://www.symbolictools.de/public/pocketconsole/developers/portsdk.htm
Last update: 2003

[18] Giuseppe Govi, STL for eMbedded Visual C++ - Windows CE.
http://www.syncdata.it/stlce/index.html Latest update: November 4, 2001

[19] Direct Sound Tutorial.
http://www.freevbcode.com/ShowCode.Asp?ID=1082 Last read: March 12, 2006

[20] Waveform-Audio interface.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_waveform_audio_interface.asp Last read: March 12, 2006

[21] Ethereal - The world's most popular network protocol analyzer home page.
http://www.ethereal.com/ Latest update: February 14, 2006

[22] Yogi Dandass, Streaming Real-Time Audio.
http://www.cse.msstate.edu/~yogi/pubs/StreamingRTAudio.pdf

[23] Dependency Walker.
www.dependencywalker.com/ Latest update: October 04, 2005

COS/CCS 2006-6