# Voice over IP over GPRS

HOMAYOUN DERAKHSHANNO

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2008

COS/CCS 2008-09

# Voice over IP over GPRS

**Homayoun Derakhshanno**

**In partial fulfillment**

**of the requirements for the**

**Master of Science in**

**Internetworking**

Advisor and Examiner: Professor G.Q. Maguire Jr.
Department of Communication Systems
School of Information and Communication Technology
Royal Institute of Technology (Kungliga Tekniska Högskolan, KTH)

**Stockholm, April 30th 2008**

## Abstract

The Voice over IP (VoIP) technology has become prevalent today due to its lower cost than traditional telephony and its ability to support new value-added services. Additionally, the increasing availability of wireless internet access has led to research studies examining the combination of wireless network access with voice over IP. With the widespread availability of advanced mobile phones and Pocket PCs, the need for VoIP applications on these mobile platforms is tangible. To enable this, we need to evaluate the current wireless access technologies to see if they can support the necessary traffic and implement software to offer these VoIP services to users.

In order to easily implement an IP-based service on GSM technology, we should use the GPRS service provided by the GSM operators. In this thesis, we evaluate Voice over IP service over GPRS in terms of feasibility and quality. Following this we ported a locally developed VoIP program to a Pocket PC (with GSM SIM-card support) which runs Microsoft's Windows Mobile in order to provide suitable software as needed to offer the service from such a portable device.



*Keywords: VoIP, GSM, GPRS, WLAN, handoff, delay, synchronization*



## Sammanfattning

VoIP tekniken har blivit en rådande teknik numera på grund av dess lägre kostnader och mervärdestjänster som erbjuds jämfört med traditional telefoni. Samtidigt som tendensen mot mer tillgänglig trådlöst internet har underlättat och därmed driver mera studier inom dessa områden. Den allt mer utbredda användningen av avancerade mobiltelefoner och handdatorer numera har lett till ökat behov av att använda VoIP tekniken för dessa mobila utrustningar är alltmer kännbar. För att möjliggöra användadet av VoIP tekniken så behöver vi först och främst utvärdera dagens existerande teknologier för att stödja iden och för det andra måste vi kunna implementera en mjukvara vilket kan erbjuda olika typer av tjänster för slutanvändaren.

För att kunna använda en IP-baserad tjänst på GSM teknologin så måste vi använda oss utan GPRS tjänster som tillhandahålls av GSM opratörer. I detta examens arbete kommer vi att utvärdera VoIP tjänster på GPRS när det gäller kvalitet och möjligheter. Därefter kommer vi att Portning en VoIP mjukvara till en handdator (utrustad med GSM sim-kort) vilket har windows Mobile operativsystemet som erbjuder en rad olika tjänster.

## Acknowledgements

First, I will give my greatest thank to my supervisor Professor Gerald Q. Maguire Jr. for giving me the opportunity and supporting me in this work. I would not have been able to finish this thesis without him. He was a source of passion and inspiration for me during this study. Many thanks to my industrial advisor Dr. Hamid-reza Rabiee and also Dr. Mohammad Karampour who tried to help me a lot.

Special thanks to my parents for their ongoing support. Thank you my beloved Mitra, for always giving me assurance especially when I was really disappointed. Undoubtedly, I could only complete this thesis with their support and encouragement.

I would like to thank my friend, Saeed Mohammadi with his helpful camaraderie and fruitful discussions that gave me extensive information. Frankly, I must say thank you very much. Also, I should thank some of my friends or colleagues for their helping in various ways who are: Erik Eliasson, Arash Behgoo, Behzad Akbari, Mohammad Haghighi, Mehrshad Sharifirad, Nima Moghaddam, and Pooya Dehghani. I hope that anyone helped me whom I missed naming will forgive me.

This work has been financially supported by Iran Telecommunication Research Center (ITRC) which is gratefully acknowledged.

Acronyms

| | |
|---|---|
| 2G | 2nd Generation |
| 2.5G | 2nd Generation Plus |
| 3G | 3rd Generation |
| 3GPP | 3rd Generation Partnership Project |
| AP | Access Point |
| BSC | Base Station Controller |
| BSS | Base Station Subsystem |
| BTS | Base Transceiver Station |
| CAC | Call Admission Control |
| CBR | Constant Bit Rate |
| CDMA | Code Division Multiple Access |
| COA | Care of Address |
| CODEC | Coder-Decoder |
| CS | Coding Scheme |
| EDGE | Enhanced Data rates for GSM Evolution |
| EGPRS | Enhanced GPRS |
| FA | Foreign Agent |
| FDMA | Frequency Division Multiple Access |
| FEC | Forward Error Correction |
| GGSN | Gateway GPRS Support Node |
| GSM | Global System for Mobiles |
| GPRS | General Packet Radio service |
| GUI | Graphical User Interface |
| HA | Home Agent |
| HTTP | Hyper-Text Transfer Protocol |
| IEEE | Institute of Electrical and Electronic Engineers |
| IETF | Internet Engineering Task Force |
| IP | Internet Protocol |
| LLC | Logical Link Control |
| MAC | Media Access Control |

| | |
|---|---|
| MIME | Multipurpose Internet Mail Extension |
| NAT | Network Address Translation |
| NIC | Network Interface Card |
| NTP | Network Time Protocol |
| PDA | Personal Digital Assistant |
| PDP | Packet Data Protocol |
| PDU | Packet Data unit |
| PSTN | Public-Switched Telephone Network |
| QoS | Quality of Service |
| RFC | Request For Command |
| RLC | Radio Link Control |
| RTP | Real-time Transfer Protocol |
| RTCP | Real-time Transfer Control Protocol |
| RTSP | Real Time Streaming Protocol |
| SDP | Session Description Protocol |
| SGSN | Serving GPRS Support node |
| SIP | Session Initiation Protocol |
| SMS | Short Message Service |
| SMTP | Simple Mail Transfer Protocol |
| SNDCP | Sub Network Dependent Convergence Protocol |
| STUN | Simple Traversal of UDP through NATs |
| TCP | Transmission Control Protocol |
| TDMA | Time Division Multiple Access |
| UDP | User Datagram Protocol |
| UMTS | Universal Mobile Telephone System |
| URI | Uniform Resource Indicator |
| UTC | Coordinated Universal Time |
| VBR | Variable Bit Rate |
| VoIP | Voice over Internet Protocol |
| WAN | Wide Area Network |
| WLAN | Wireless Local Area Network |

# Table of Contents

**ABSTRACT** ........ I

**ACKNOWLEDGEMENTS** ........ II

**TABLE OF CONTENTS** ........ V

**TABLE OF FIGURES:** ........ VII

**1. INTRODUCTION** ........ 1

1.1 Background ........ 1
1.2 Thesis Overview ........ 2
1.3 Project in Detail ........ 2
*1.3.1 Idea* ........ *2*
*1.3.2 An example usage scenario* ........ *2*
*1.3.3 Equipment Requirements* ........ *3*
*1.3.4 Planned Experiments* ........ *4*

**2. TECHNOLOGIES AND PROTOCOLS INVOLVED** ........ 6

2.1 VoIP ........ 6
*2.1.1 SIP* ........ *6*
*2.1.2 SDP* ........ *10*
*2.1.3 RTP* ........ *10*
*2.1.4 NTP* ........ *12*
*2.1.5 VoIP and NAT problem* ........ *13*
2.2 GSM & GPRS ........ 16
*2.2.1 GSM History* ........ *16*
*2.2.2 GPRS Adoption around the World* ........ *17*
*2.2.3 GPRS as part of the evolution of GSM* ........ *17*
*2.2.4 GPRS Data Services and Infrastructure* ........ *18*
*2.2.5 Frequency and Coverage* ........ *19*
*2.2.6 Capacity and Dimensioning for Growth* ........ *20*
*2.2.7 GPRS Network Optimization* ........ *22*
*2.2.8 Reliability, Jitter, and Latency* ........ *23*
2.3 E-model ........ 24

**3. VOICE QUALITY OVER GPRS** ........ 26

3.1 VoIP in details ........ 26
*3.1.1 VoIP CODECs* ........ *26*
*3.1.2 IP header compression* ........ *28*
3.2 A deeper look at GPRS ........ 29
*3.2.1 GSM physical and link layer channel* ........ *29*
*3.2.2 GPRS layers* ........ *30*
3.3 VoIP QoS in GPRS and WLAN ........ 31
*3.3.1 Designing a QoS protocol for real-time services* ........ *31*
*3.3.2 QoS mechanisms for VoIP over WLAN* ........ *32*
*3.3.3 GPRS QoS and delay sources* ........ *34*

**4. MEASUREMENTS** ........ 36

4.1 Time Synchronization ........ 36
*4.1.1 One-Way Delay Measurement* ........ *36*
*4.1.2 The Application* ........ *37*
*4.1.3 Our Experiments* ........ *39*

4.2 VoIP over GPRS Experiments: ........ 42
*4.2.1 One-way measurement* ........ *42*
*4.2.2 Round-trip measurement* ........ *42*

**5. WINDOWS MOBILE AND MINISIP ........ 48**

5.1. Windows Mobile ........ 48
5.2. Minisip ........ 49
*5.2.1. Porting Minisip to Windows Mobile* ........ *49*
*5.2.2. Visual Studio 2005 Settings for Debugging Minisip* ........ *51*
*5.2.3. Windows mobile Emulator Settings* ........ *56*

**6. CONCLUSIONS AND FUTURE WORK ........ 58**

**REFERENCES ........ 60**

**APPENDIX: ........ 65**

A1. Synchronizing the time with a GPS receiver (Server Part) ........ 65
A2. Synchronizing the time with a GPS receiver (Client Part) ........ 73
A3. Sending and receiving RTP packets on Windows Mobile 5.0 ........ 78
A4. Sending and receiving RTP packets on Windows XP (Vista) ........ 81
A5. Minisip GUI in Windows Mobile 5.0 ........ 82

# Table of Figures:

Figure 1: Topology of the test-bed to be used for the Experiment ..... 5
Figure 2: SIP messages involved in initiating and terminating a session ..... 8
Figure 3: RTP packet ..... 10
Figure 4: RTP, RTCP, and RTSP ..... 12
Figure 5: Components in GPRS network (basic schema without all connections) ..... 18
Figure 6: Concentric cells of coverage in GPRS ..... 20
Figure 7: Voice over IP implementation ..... 28
Figure 8: GPRS Protocol stack ..... 34
Figure 9: Clock offset between two end-parties during sync ..... 40
Figure 10: Clock offset between two end-parties after sync ..... 41
Figure 11: GUI of the application located in the GPRS node ..... 43
Figure 12: Round-trip delay of G.711voice CODEC over GPRS ..... 44
Figure 13: Round-trip delay of G.723 voice CODEC over GPRS ..... 45
Figure 14: Upload Coding Scheme used as a fraction of the measurement time ..... 46
Figure 15: Download Coding Scheme used as a fraction of the measurement time ..... 46
Figure 16: Minisip Visual Studio settings/ Property tab/General ..... 51
Figure 17: Minisip Visual Studio settings/ Property tab/Debugging ..... 52
Figure 18: Minisip Visual Studio settings/ Property tab/Deployment ..... 52
Figure 19: Minisip Visual Studio settings/ C++/preprocessor ..... 53
Figure 20: Minisip Visual Studio settings/ Linker tab/ Additional Dependencies ..... 54
Figure 21: Minisip Visual Studio settings/ Linker tab/Advanced ..... 54
Figure 22: Minisip Visual Studio settings/ options/ VC++ Include directories ..... 56
Figure 23: Minisip Visual Studio settings/ options/ VC++ Library directories ..... 56

Table 1: Coding Parameters for GPRS Coding Scheme ..... 19
Table 2: Relation of Id with one-way delay ..... 25
Table 3: VoIP CODECs Bandwidth Requirement ..... 27

# 1. Introduction

## 1.1 Background

The importance and rapid spread of Voice over IP (VoIP) around the world is well known. The number of VoIP users is growing day by day and increasing varieties of users are utilizing VoIP services. It seems that, PC-to-PC users, PC to phone users, phone-to-phone users, etc., have all found out that VoIP is a suitable alternative for the traditional telephony service that they had previously used. VoIP has become a technology that no one can stop or ignore, because it is so widespread and the number of companies that provide this service are increasing rapidly. Some software/hardware platforms offer both Instant Messaging (IM) and VoIP; this combination is especially interesting for young users. However, some software/hardware offers only VoIP which is usually utilized as a PC-to-PC, PC-to-phone, phone-to-PC, or phone-to-phone type of service.

Given this market growth, using the wide area wireless cellular networks (such as the GSM[*] network) as the underlying network for VoIP is interesting as this would enable provisioning of VoIP servers even to mobile users. Although the GSM network growth in this last decade is significant, the evolution toward 3G[†] wide-area cellular systems (such as UMTS) based on packet-switching of IP traffic suggest that such networks might be suitable for VoIP service(s). Unfortunately, the existing GSM network infrastructure in many countries cannot support a direct transition to 3G cellular networks, thus GSM's 2.5G offers a transition mechanism. One of the additions to circuit-switched GSM in the transition to 2.5G networks was the addition of GPRS[‡] as a packet-switched extension to GSM technology. This clearly offers an interesting potential base for VoIP over GPRS as an alternative to VoIP over a GSM circuit-switched channel. GPRS offers the advantages of packet switching allowing discontinues media streams and enables a nearly constant connectivity – while only incurring costs for carrying actual traffic.

VoIP over GPRS may offer an alternative to circuit-switched GSM network usage and create a new business opportunity for GSM operators; while simultaneously reducing the cost of voice calls for GSM users. In addition, since new models of mobile handsets often have a built-in WLAN interface, users can make use of this alternative connectivity to carry their VoIP traffic when they are at work or home (for example, via local WLAN coverage). Being able to carry VoIP traffic over these local WLANs reduces the load on the GSM operators network - effectively giving them increased capacity, without requiring the installation of additional

[*] Global System for Mobile Communication: a mobile network that was launched in Europe and now is available in most countries in the world.
[†] Third Generation mobile network
[‡] General Packet Radio Service is a packet switched service over the GSM network.

infrastructure, thus it is vital to evaluate the quality of voice over GPRS and to create a model for handoff from WLAN to GPRS and vice versa.

## 1.2 Thesis Overview

In this thesis, an open-source SIP User Agent, minisip, has been ported to a Pocket-PC specifically the HTC "i-mate" (model: k-jam) which runs Microsoft's Windows Mobile 5.0 (see chapter 5). Following this, measurements of VoIP over GPRS were made (see chapter 4). Finally some conclusions and future works are presented (see chapters 6).

## 1.3 Project in Detail

The project began with a simple idea (evaluating VoIP over GPRS) – see section 1.3.1. Based upon this idea a usage scenario was envisioned which describes how a VoIP over GPRS service number be used – see section 1.3.2. Next the hardware and software needed to experiment with this scenario are described – see section 1.3.3. Finally the test environment and test plan is outlined –see section 1.3.4. Following this the current chapter introduces the relevant protocols and software which are used for the rest of the thesis.

### 1.3.1 Idea

The initial idea was to send voice as IP packets over a GPRS network. When GPRS was first introduced the delay, due to GPRS interleaving and processing made the additional delay too high. However, more recent extensions to GPRS (specifically as part of EGPRS) have significantly reduced these delays. A key element of the project is to determine if it is now both feasible and practical to send voice as packets over GPRS. If so, then the thesis should also characterize the performance of such a Voice over GPRS system.

In addition, Voice over IP traffic could be sent over a circuit switched GSM channel, if the delay over GPRS is occasionally too large (for example, when EGPRS is not locally supported), thus offering a fall back possibility which can be transparent to the users as far as functionality goes, but might have significantly higher cost. Voice over IP over a circuit-switched connection will offer an additional case for comparison of system performance.

### 1.3.2 An example usage scenario

You are in your office building and you make a VoIP call as you start to leave your office at 17:25. Since you are in a building which has excellent WLAN coverage, your device initiates the call using SIP over the WLAN and sends the RTP traffic (see section 2.1.3) to the called party over the WLAN. After a couple of minutes, you leave the building and arrive at the parking lot, there you get into your car. Somewhere between the building and your car, you lost WLAN connectivity. In order to maintain the call there are couple of choices:

1. Activate a GPRS PDP context and send your IP packets via the cellular interface, rather than the WLAN interface
2. Set up a circuit switched[§] call, but send IP packets over this connection
3. Set up a circuit switched call and bridge the VoIP call to/from the cellular network
4. Drop the call and set up a new call to the called party
5. Drop the call
6. Always set up the call via the cellular interface

The above choices assume that your phone has both WLAN and wide area cellular interfaces. This will be a working assumption throughout the remainder of the thesis. Considering the above choices, it seems that:

- Choice 6 looks unnecessarily expensive.
- Choice 5 looks like a bad choice as the user is likely to be unhappy with this.
- Choice 4 at first sight looks almost as bad, but it doesn't happen very often so the user might accept it.
- Choice 3 seems appealing, but requires additional infrastructure (the conference bridge) hence increasing your costs, and if the call is being terminated at a fixed line phone you might have to worry about echo cancellation.
- Choices 1 and 2 are the most appealing since you might actually be able to do a soft handoff if you can anticipate the loss of WLAN connectivity or trade off this with slightly higher wide area cellular costs.
- Choice 2 is no more expensive for the user per minute than choice 3, but costs less for the operator and might avoid problems with echoes.
- Choice 1 is more efficient that choice 2, since you only have to send/receive traffic when you actually have traffic to send (or receive). Activating a PDP context is faster than making a circuit switched call, so if there is a hard handoff the interruption might be shorter.

Given this scenario, the importance of evaluating Voice over IP over GPRS service is clear. In the following sections, the phases of the project are explained in detail.

### 1.3.3 Equipment Requirements

The minimum equipment requirements for doing the necessary experiments with the scenario outline above are:

[§] Circuit-switched channel means establishing a fixed bandwidth channel between two nodes before user may communicate. But in Packet-switched networks, data packets are routed between nodes over several data links shared with other traffic resulting in optimizing the bandwidth usage but sacrificing speed and quality guarantee.

- At least one mobile client (preferably two - so we can have two mobile parties);
- At least one fixed computer (for both development work, and to serve as a fixed network attached end party);
- A SIP user agent for each mobile device and computer;
- Access to GPRS, GSM, and WLAN networks (and accounts for using these networks); measurement software/hardware such as Wireshark[**];
- Access to suitable time sources (for example, NTP or GPS);
- Statistical analysis software (such as Splus[††], R[‡‡], or other similar software);

In order to have a SIP client on the mobile device, I ported an existing client to the device. In this project, I ported Minisip (see section 5.4.1) to a HTC Pocket-PC called the "i-mate" (specifically the "k-jam" model) which runs Microsoft's Windows Mobile 5.0 as its operating system and has GSM, GPRS/EGPRS, Bluetooth, and a built-in Wi-Fi (WLAN) interface; additionally it supports EDGE (which was expected to be interesting for further studies). For synchronizing the clocks with actual time in our experiments, we will use two Evermore GPS receivers. One of these GPS receivers is communicating via a USB port to the fixed part of the experimental equipment specifically, a laptop and the other GPS receiver has a Bluetooth communication port which enables it to be used with the Pocket-PC (i.e., the mobile device).

### 1.3.4 Planned Experiments

The planned experiments are briefly described below. The aim is to carry out a systematic test of the basic feasibility, followed by more detailed measurements to characterize the VoIP service under different conditions:

As shown in figure 1, traffic is sent over the GPRS network and it is desirable that it be monitored at different points of the network. These measurements are used to extract parameters that will be used for quality evaluation. The link quality of the wide area wireless link is manipulated by an attenuator in order to emulate the situation in which the mobile station is near or far from a BTS. A faraday cage is used to prevent any unwanted signals or noise that might affect with our experimental environment.

The points that are marked as the traffic monitoring points require the collaboration of the relevant organizations. Unfortunately, it was not possible to carry out the proposed experiments as planned, therefore I simply sent traffic over a GPRS link and with the end-party attached via

---

[**] http://www.wireshark.org/, a packet sniffer application for network analyzing
[††] http://www.insightful.com/products/splus/, a statistical application as analytical tool
[‡‡] http://www.r-project.org/, R is a free software environment for statistical computing and graphics

the internet, this end party was able to receive the packets and estimate the delay. The experimental configuration actually used for this thesis in described in section 4.2.

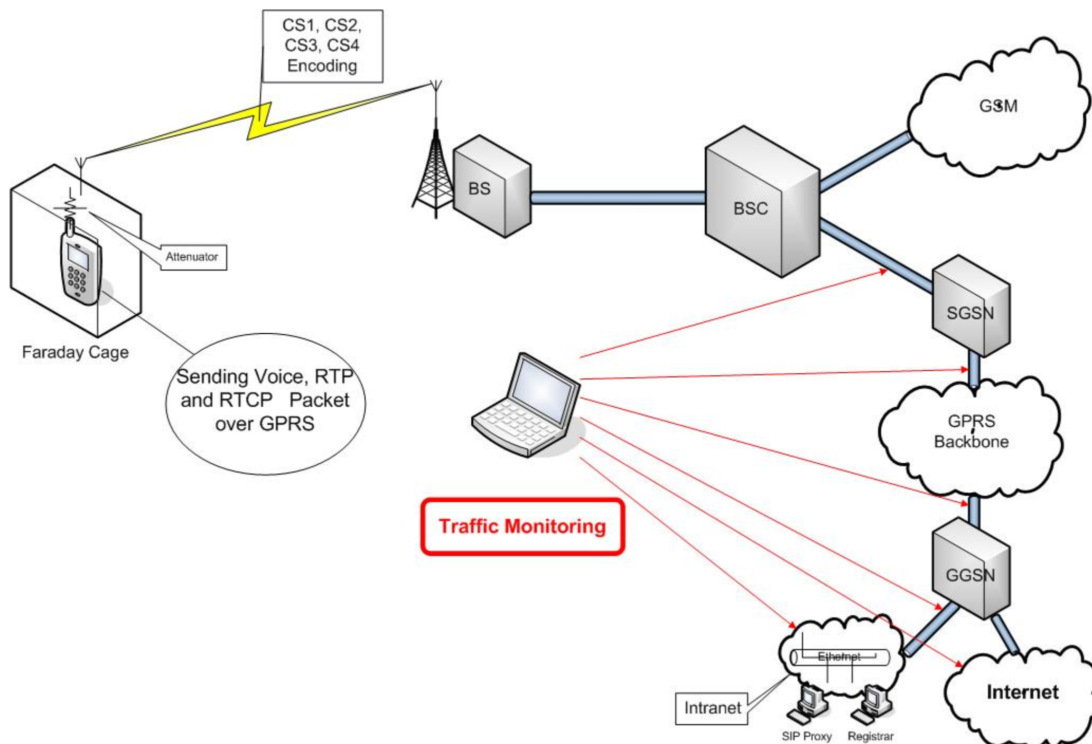


**Figure 1: Topology of the test-bed to be used for the Experiment**

When sending traffic over the GPRS network, there are two choices for the traffic to be used:

- Real voice sources- for example (a) a pre-recorded call (as this will include speech pauses) and (b) actual humans speaking to each other (as this will examine the impact of delays and losses on the actual user interactions)
- Synthetic traffic- for example using a traffic generator that simply sends an RTP packet every 20ms (these packets of course have to have proper time stamps and sequence numbers; an advantage is that any pattern of speech activity can be simulated later)

The second choice was utilized in the experiments; these are described in more detail in section 4.2.

There is a major difference between implementing the SIP UA in the handset and having it in a separate computer that is attached to the phone via a serial, USB, or Bluetooth connection. This is because these serial connections generally implement PPP which has its own flow control, adds its own queuing delays, adds in some cases alternating simplex use of a shared connection, etc. Therefore, to avoid all of these aspects complicating our measurements; in this project I will port the software for the SIP UA directly to the handset itself.

# 2. Technologies and Protocols involved

## 2.1 VoIP

Today, nearly free long-distance calling is not unusual. Several programs and companies offer long-distance voice services utilizing voice over IP (VoIP) technology. In this market, the combination of lower costs and the new features offered by these VoIP programs plays a significant role in the choice by many users to replace traditional telephony with VoIP [1].

IP-based telephony is also called packet telephony. IP telephony works over an IP network such as internet, and enables users to send pictures, video, and text in addition to voice packets over an IP network. With the advent of PDAs, advanced cellular mobile phones, and Pocket PCs, using a PC for making VoIP calls is unnecessary, rather the equipment has VoIP software which is compatible and working on these devices [1].

There are two major protocols that have been utilized for VoIP: H.323 and SIP. H.323 was developed by ITU in 1997 and was widely used in PC-to-Phone and Video-Conferencing applications. In 1999, a simple protocol called SIP was created by IETF to provide a distributed architecture to developers to create VoIP applications, which can have a number of advanced features [1].

The following sections describe the basics of SIP and relevant protocols that are important for this thesis. Citations to additional sources are given for the reader that wishes additional background or further details.

### 2.1.1 SIP

Session Initiation Protocol (SIP) is a signaling protocol used for establishing, modifying, and terminating sessions between users. SIP is a text-based protocol (like HTTP and SMTP). It is used to initiate an end-to-end interactive session between users. It is described in RFC 3261. Its simplicity is based on reusing existing IP protocols and the end-to-end design principle of the internet architecture. However, SIP only addresses the signaling part of H.323; while RTP is used by both H.323 and SIP to carry the media during the session [2]. As described in the SIP standard (RFC 3261), SIP is an application−layer control protocol with the ability to manage multimedia sessions, such as Internet telephone calls, which makes this protocol suitable for its use in VoIP [3].

**SIP Components**

In RFC 3261, the required components needed for a SIP-based network are defined. However, in practice several of these components are combined [1].

***SIP User Agents***

Telephony devices are implemented as User Agents (UAs) in the SIP protocol [1, 3]. UAs consist of UACs (User Agent Clients) and UASs (User Agent Servers). The UAC is the only component that is allowed to send a request in a SIP network. The UASs are servers that can receive and respond to requests.

UAs can be implemented in hardware such as an IP phone set or as software (a softphone). Making calls directly between User Agents is possible without any additional software components.

***SIP Servers***

RFC 3261 also defines four types of SIP servers [1, 2]:

Location Server — To identify the location of the callee (called party), utilized by Redirect Server or a Proxy server;

Proxy Server — An intermediate program that behaves as both a server and a client for making requests on behalf of other clients. It interprets and if necessary rewrites the SIP requests before forwarding them to the servers which are closer to destination;

Redirect Server — A server that receives SIP requests and maps the address into zero or new addresses and directs the client to an alternate URI[§§];

Registrar Server — Accepts Register requests and is typically collocated with a Proxy or Redirect server. It saves the location information of the party and updates the Location sever.

SIP performs five tasks regarding the establishment and termination of communications sessions [3]:

User location — Determining the destination end system.

User availability — Determining the willingness of the call party to accept a call to UA

User capabilities — Negotiation of the session's parameters

Session setup — Establish the session

[§§] URI: Uniform Resource Indicator is defined in RFC 2543, 3261; it is similar to email address: user@domain

Session management    Modification or termination of the session

Note that SIP does not provide services, but rather provides an environment that can be used to implement these services. SIP works with either IPv4 or IPv6.

SIP utilizes an *offerer/answerer* model, in which the caller represents the *offerer* and the called party represents the *answerer.* Using an example we investigate the SIP structure in detail below.

In this example, one user (the offerer, referred to as "Alice") calls another user (the answerer, referred to as "Bob") using his SIP identity, a type of *Uniform Resource Identifier* (URI), called a SIP URI. This SIP URI is similar to an email address and consists of the user's name (or some string identifier) and host identifier (for example alice@kth.se). Alice sends a request called an INVITE to Bob's SIP provider's server (in this case su.se SIP proxy) by her proxy (in this case kth.se SIP proxy). If Bob accepts the call, the media session is established directly between the SIP UAs of Alice and Bob.

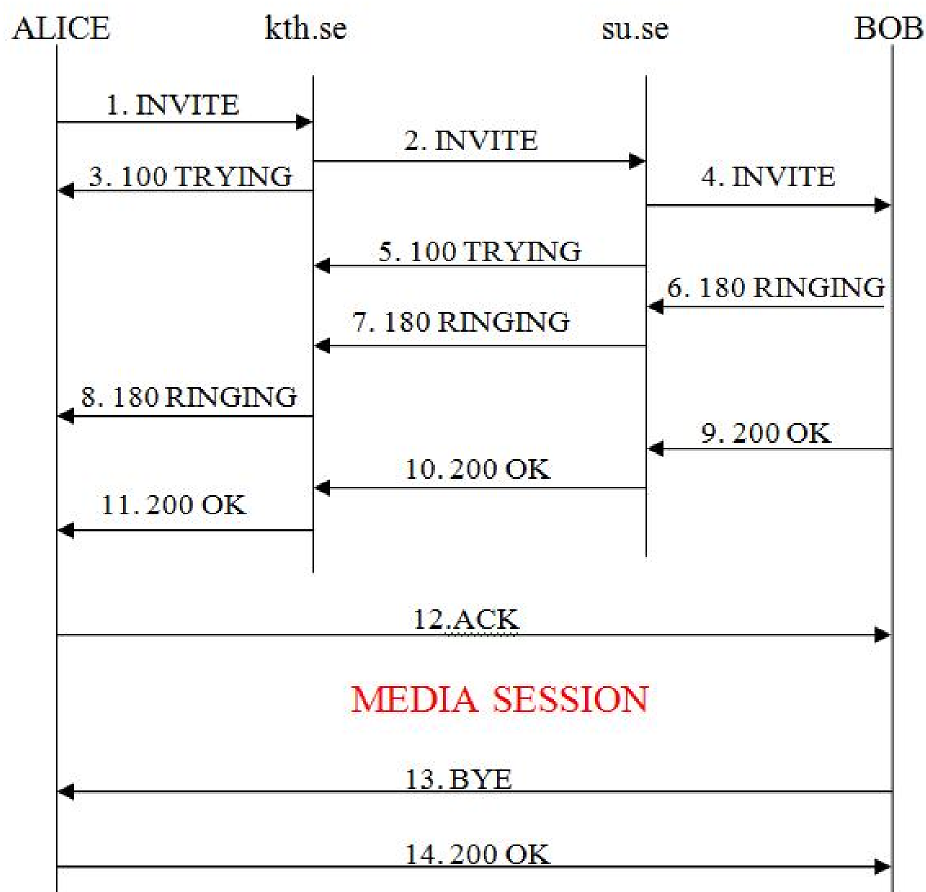


**Figure 2: SIP messages involved in initiating and terminating a session [2]**

Another important issue regarding SIP is the registration of the user with their SIP registrar server. When a SIP−based device (*User Agent*) goes online, it should register with a SIP Registration Server (*Registrar*) by sending a SIP REGISTER message. Registration is for a period of time; this associates the user's URI with the address of one or more of the user's SIP UAs, this information can be subsequently used to find the IP address where the user can be contacted [3].

The main functions of SIP in a VoIP context are:
- Registering a user with their SIP provider
- Inviting one or more users to join a session
- Negotiating the terms and conditions of a session
- Terminating sessions

One of the most important methods in SIP is the **INVITE** method, which is used to establish a session between participants (these participants have previously registered with their respective SIP provider's Registrars). As an example, the following paragraph shows the first INVITE message from the example shown in figure 2 [3]:

*INVITE sip:bob@su.se SIP/2.0*
*Via: SIP/2.0/UDP pc33.kth.se;branch=z9hG4bK776asdhds*
*Max−Forwards: 70*
*To: Bob <sip:bob@su.se>*
*From: Alice <sip:alice@kth.se>;tag=19283874*
*Call−ID: a84b4c76e66710@pc33.kth.se*
*CSeq: 314159 INVITE*
*Contact: <sip:alice@pc33.kth.se>*
*Content−Type: application/sdp*
*Content−Length: 142*
*(Alice's SDP not shown)*

The first line specifies the method, in this case an INVITE; which is followed by the remainder of the INVITE message header. **Via** contains the address at which Alice expects to receive a response to her request. The branch parameter identifies a specific transaction. **To** contains a display name and the SIP URI to which the request was directed. **From** identifies the originator of the request SIP URI. The tag parameter is used for identification purposes. **Call−ID** is a globally unique identifier for the call. **CSeq** is an integer used as a command sequence number. **Contact** is a SIP URI that represents a direct route to contact Alice. **Max−Forwards** stipulates the maximum number of hops to the destination. **Content−Type** describes the message body which is not shown here. **Content−Length** defines the length of this message body. The details of the established session are not overtly described by SIP and are carried in the SIP message body encoded using the *Session Description Protocol* (SDP) [3].

Another important SIP method is **REGISTER** which is used to register a UA's address with a system (via a SIP Registration Server or Registrar). A device must register in order to provide location information for incoming calls, otherwise the user cannot be reached at this UA. Other SIP methods include the **ACK** method which confirms that the client has received a final response to an INVITE request; and the **BYE** method which indicates that the user wants to terminate a session. This latter message may be sent by either the originator of the call or the receiver; while the **CANCEL** message **c**ancels a previous request message [3].

There are many different responses to the aforementioned methods these are divided into six different categories:

| | |
|---|---|
| **1xx Responses** | Informational Responses (e.g. 180 Ringing and 100 Trying) |
| **2xx Responses** | Successful Responses (e.g. 200 OK) |
| **3xx Responses** | Redirection Responses (e.g. 302 Moved Temporarily) |
| **4xx Responses** | Request Failure Responses (e.g. 404 Not Found) |
| **5xx Responses** | Server Failure Responses (e.g. 503 Service Unavailable) |
| **6xx Responses** | Global Failure Responses (e.g. 600 Busy Everywhere) |

### 2.1.2 SDP

The Session Description Protocol (SDP) is a text based protocol describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation or re-negotiation. In a conference session, SDP conveys information concerning the proposed session to each of the recipients. It is just a format for encoding a session description and does not incorporate a specific transport protocol [4]. SDP messages are encoded using MIME[***] and attached as a message body in SIP messages [2].

### 2.1.3 RTP

The Real-time Transport Protocol (RTP) defines a standard packet format for delivering audio, video, timed text, and other media over the Internet [5]. It was developed by the Audio-Video Transport Working Group of the IETF as RFC 1889 and RFC 3550. It uses a connectionless transport protocol, usually UDP. A speech frame carried by RTP in a packet is shown in figure 3:

Figure 3: RTP packet [2]

[***] Multipurpose Internet Mail Extensions (MIME) is an Internet Standard that extends the format of e-mail to support text in character sets other than US-ASCII, non-text attachments, multi-part message bodies, and header information in non-ASCII character sets. A large proportion of e-mail is transmitted via SMTP in MIME format [6].

RTP provides end-to-end network transport functions that are suitable for applications transmitting real-time data, such as audio, video, simulation data, etc. over multicast or unicast transport services. RTP does not deal with resource reservation and nor does it guarantee quality-of-service for real-time services. This data transport is paired with a control protocol (RTCP) to allow monitoring of the data delivery, while being scalable to large (very large) multicast networks. RTCP provides the ability to monitor the quality of service by conveying information about the participants in an on-going session. RTP and RTCP are designed to be independent of the underlying transport and network layers [7].

Each RTP packet includes a payload type identification, sequence number, timestamp, and a payload. Applications typically run RTP on top of UDP make use of UDP’s multiplexing and checksum services; both protocols contribute to the transport protocol’s functionality. RTP supports data transfer to multiple destinations using multicast distribution if this functionality is provided by the underlying network [7].

RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees, but relies on lower-layer services. It does not guarantee delivery or prevent out-of-order delivery, and does not assume that the underlying network is reliable or delivers packets in sequence. Hence, the sequence numbers included in each RTP packet allow the receiver to reconstruct the sender's packet sequence. Additionally, the sequence numbers might be used to determine the proper location of a packet, for example in video decoding, without necessarily decoding packets in sequence [7].

As said before, RTP can carry any type of data with real-time characteristics, however, call setup and tear-down is usually (but not necessarily) done by the SIP protocol. RTP does not utilize fixed standard TCP or UDP ports; but rather RTP uses a dynamic port range. This makes it difficult for RTP to traverse firewalls. In order to get around this problem, it is often necessary to set up a STUN (see section 2.1.5.1) server [5].

Although, RTP was originally designed as a multicast protocol, it has been applied in many unicast applications. Today, it is frequently used in streaming media systems (for example RTSP[†††]) as well as "video conferencing" and "push to talk" systems (for example H.323 or SIP), making it the technical foundation of Voice over IP media delivery. Applications using RTP are generally not extremely sensitive to packet loss, but typically are sensitive to delay; therefore UDP is a better choice than TCP as an underlying transport protocol for such delay sensitive applications [5].

[†††] Real-Time Streaming Protocol: it is defined in RFC 2326 and used for remote media playback control

<table>
<tr><td>Audio, Video Application</td><td colspan="3">Signaling and Control</td><td colspan="2">Streaming Application</td></tr>
<tr><td>Audio, Video CODECs</td><td rowspan="2">RTCP</td><td colspan="2">SDP</td><td colspan="2">CODECs</td></tr>
<tr><td>RTP</td><td colspan="2">SIP</td><td colspan="2">RTSP</td></tr>
<tr><td colspan="3">UDP</td><td colspan="2">TCP</td><td>UDP</td></tr>
<tr><td colspan="6">IP</td></tr>
</table>

**Figure 4: RTP, RTCP, and RTSP [2]**

With regard to real-time delivery of media using IP packets, we should consider two aspects: Order and Time; in order to reproduce the speech after transmission, each RTP packet contains a timestamp and a sequence number (the initial sequence number is chosen randomly) [2]. Because of the delay (due to Encoding, Packetization, Propagation, Switching, Receiving, Decoding, and Playing) which we call the "Mouth-to-Ear Delay" there is a play out buffer in the final receiver in order to hide the delay variations (jitter) by adding additional delay called the play out delay. Also, for recovering the lost packets a technique such as Forward Error Correction (FEC) which adds some redundant data to the packets as error detection bits may be used. FEC enables the receiver to detect and correct errors without need to retransmit the original data or informing the sender about the error [2].

Since the timestamps of each RTP stream starts with a random value, RTP uses NTP (Network Time Protocol) timestamps in order to synchronize multiple streams.

### 2.1.4 NTP

The Network Time Protocol (NTP) is a protocol for clock synchronization of computer systems over packet-switched, variable-latency data networks. NTP uses UDP port 123 as its transport layer. It is designed specifically to resist the effects of variable latency. It is an old protocol that has been commonplace since 1985 (RFC[‡‡‡] 958) and is still being developed (RFC 1059, 1119, 1305).

NTP uses the UTC time scale and end systems can usually maintain time synchronization to within 10 milliseconds (0.01 s) over the Internet, and can achieve an accuracy of 200 microseconds (0.0002 s) in LANs under ideal conditions [8].

[‡‡‡] For further details about each RFC go to the IETF organization's " Request For Comment" working group web site through http://www.ietf.org/rfc.html

The NTP Unix daemon is a process that runs continuously on a machine that supports NTP; all recent versions of the Linux and Solaris operating systems have this support. Also, Microsoft's Windows 2000/XP has the ability to synchronize the computer's clock to a NTP server [8].

NTP uses a hierarchical system of "clock strata". These stratum levels specify the (logical) distance from the reference clock and its accuracy [8].

- Stratum 0: Devices such as a atomic (cesium, rubidium) clocks, GPS receivers or other radio clocks. Stratum-0 devices are not attached to the network; instead they are locally connected to computers (e.g. via an RS-232 connection using a Pulse per second signal, USB, or other direct connection).

- Stratum 1: Computers attached to Stratum 0 devices. They operate as servers for timing requests from Stratum 2 servers via NTP. They are called "time Servers".

- Stratum 2: Computers that send NTP requests to Stratum 1 servers. A Stratum 2 computer utilizes a number of Stratum 1 servers and uses the NTP algorithm to reach the best data sample, dropping any Stratum 1 servers that seem to be wrong.

- Stratum 3 and higher: These computers employ exactly the same NTP functions of data sampling as Stratum 2, and can act as servers for higher strata, potentially up to 16 levels.

NTP uses 64-bit timestamps consisting of a 32-bit second part and a 32-bit fractional second part with an epoch of January 1, 1900. Thus, NTP has a time scale of $2^{32}$ seconds (136 years) and a theoretical resolution of $2^{-32}$ seconds (0.233 nanoseconds). Future versions of NTP will extend the time representation to 128 bits (64 bits for the second and 64 bits for the fractional second)[8].

NTP accuracy depends strongly on the precision of the local-clock hardware and strict control of device and process latencies. In addition, the computer should adjust its logical-clock time and frequency in response to corrections calculated based upon NTP [9].

After some explanations about the protocols that are relevant to this thesis, we will investigate the other part of the project which concerns GPRS in section 2.2. However, we first must examine some problems which SIP and RTP will encounter when used in practice.

### 2.1.5 VoIP and NAT problem

Firewalls and network address translation devices (NATs) are located at the edge of almost all networks. While today's firewalls are able to dynamically open and close ports as required by

VoIP signaling protocols such as SIP, they are still ineffective at supporting incoming media flows. Unfortunately, NATs prevent two-way voice and multimedia communication, because the private IP addresses and ports inserted by the end client devices (IP phones) in the SDP payload are not routable to/from public networks.

The role of a firewall is to protect the inside network from being accessed by unauthorized sources from outside the firewall, and it operates through blocking traffic based on three parameters: the source IP address, the destination IP address, and the traffic type. Firewalls also consider the direction of traffic flow. Generally, incoming traffic from un-trusted nodes in the public (globally routable) network is allowed in, if initiated from a device within the (inside) trusted private domain. SIP-based communication is based on unsolicited incoming calls, from a wide range of unknown sources. However, most network administrators are hesitant to change their policies to allow unrestricted two-way communication because of the potentially serious security attacks this could enable [10].

Because NAT translates IP addresses and port numbers in the packet headers from within a private address range into public addresses, this causes problems when traffic flows from a private to public network. Each device in the private network has its own private IP address and when the traffic (a media stream, for example) is sent to a device on the public network, this traffic flow will be dynamically assigned a specific public IP address port number combination by the NAT. The NAT maintains a "table" that links private addresses and port numbers to the public port numbers and IP addresses. These table entries can only be initiated by outgoing traffic [10] or by using a protocol which can manipulate these tables (such as Universal Plug and Play[§§§]).

Moreover, the end-to-end SIP messaging between clients which contain SDP will contain the private IP addresses and ports that the clients (User Agents) utilize for the media flows; however, these IP addresses are non-routable in public networks. Note that this issue also applies to other signaling protocols such as H.323. There are several methods to solve the aforementioned problems, next we will investigate some of them.

#### 2.1.5.1 Simple Traversal of UDP Through Network Address Translators (STUN)

The STUN protocol enables a SIP client to discover whether it is behind a NAT and to determine the type of NAT. Although STUN has been very attractive to users, it suffers from a flaw, as it will only work with some types of NATs. In fact, it doesn't work with the type of NAT most commonly used in corporate networks. A symmetric NAT creates a mapping based on source IP address and port number **as well as the destination IP address and port number**.

[§§§] Universal Plug-and-Play (UPnP) is a set of computer network protocols allowing devices to connect seamlessly and to simplify the implementation of networks.

Another problem is that STUN does not support TCP based SIP devices in spite of the SIP RFC 3261 specification. STUN defines a special server (STUN server) located in the public address space to inform a STUN-enabled SIP client in the private address space of the NAT's public IP address and the port that is being used for this particular session. A STUN-enabled client sends an exploratory message to the external STUN server to determine the receive ports in order to subsequently specify these values in an SDP message. The STUN server examines the incoming message and informs the client which public IP address and ports were used by the NAT. These are then used in subsequent call establishment messages. Note that the STUN server is simply used by the SIP UA to learn about its external IP address and port and this STUN server is not located in the signaling or media data flows [10]. Actually, once the outgoing port was mapped for the STUN server traffic, any traffic from any part of the network, with any source IP address, is able to use the mapping in the reverse direction and can reach the receive port on the client which is a major security hole. However, while the traffic can reach the node, the firewall or other filters at the node can reject the traffic based upon its source port.

As mentioned previously, the problem of symmetric NAT is because the destination VoIP client's IP address is different from the IP address of the STUN server. This means that the NAT will create a new mapping using a **different** port for outgoing traffic, which in turn means that the information contained in the call establishment messages is incorrect and the call attempt may fail. The same problem occurs for incoming traffic. The IETF has proposed another mechanism called TURN[****] – that is designed to solve the media traversal issue for symmetric NATs. TURN relies on a server that is inserted in the media and signaling path. This TURN server usually is located either in the customer's DMZ or in the Service Provider's network; however it can be located everywhere in publicly routable internet. A TURN-enabled SIP client sends an exploratory packet to the TURN server, which responds with the public IP address and port used by the NAT, this information will be used for a subsequent media session. This information is used in the SIP call establishment messages and for subsequent media streams. The advantage of this approach is that there is no change in the destination address seen by the NAT and, thus, a symmetric NAT can be used [10].

#### 2.1.5.2 Application Layer Gateway (ALG)

This technique relies on an enhanced Firewall/NAT called an Application Layer Gateway (ALG) that understands the signaling messages and their relationship with the resulting media flows. The ALG processes the signaling and media streams in order to find the public IP addresses and ports being used. Basically a NAT with built-in ALG can re-write the IP address and port number information within the SIP messages and can maintain address-bindings until the session terminates.

[****] Traversal Using Relays around NAT (TURN), for further details go to http://tools.ietf.org/html/draft-ietf-behave-turn-05

#### 2.1.5.3 Tunnel Techniques

By tunneling both media and signaling through the Firewall/NAT nodes located in the public IP address space we can proxy the communication between the internal and external address spaces. This method requires a new server within the private network and another in the public network; then we create a tunnel between them, subsequently all the SIP traffic is sent via this tunnel, thus allowing the VoIP system to make both outgoing calls and receive incoming calls. The unencrypted tunnel which is usually used could be problematic due to attacks from outside; but in our case, since we know exactly which node to communicate with, we can use an encrypted tunnel.

## 2.2 GSM & GPRS

GPRS is of a larger subsystem of a GSM system. First, we look at GSM in depth.

### 2.2.1 GSM History

GSM[††††] is a wireless wide-area communication technology. The mobility support is based on MAP (Mobile Application Protocol) and an air interface using Time Division Multiple Access (TDMA). In 1984, European countries decided to dedicate spectrum for GSM in the 900 MHz band.

The phase 1 GSM Core specification was completed by 1997, but development has continued with a stepwise migration to phase 2, 2.5, and now 3G. This latter stage is managed by the 3rd Generation Partnership Project (3GPP); which is responsible for ongoing evolution of the standard. GSM has had a rapid growth in numbers of both operators and customers; in 1992, there were 13 GSM networks offering service in Europe, by 1995, GSM service was offered in 69 countries in three frequency bands (900, 1800, and 1900 MHz) with more than 12 million customers. In 2000, there were more than 400 million GSM subscribers and at the end of 2003 more than 1 billion subscribers [11].

GSM Phase 2 introduced non-voice services such as Call waiting, Call hold, Call conferencing, etc; it also introduced support for High-Speed Circuit Switched Data (HSCSD). In 1999, The GSM Association released Phase 2+ which provided internetworking with 1800 and 1900 MHz in addition to the 900 MHz frequency band. Adapting IP technology to GSM was initially resisted, but became essential later. The European Telecommunications Standards Institute (ETSI) began to work on the GPRS specification in the mid-1990s. Transmitting data via a packet protocol mode without negatively impacting the circuit switched services was the main goal of the ETSI GPRS specification. Packet transfer is well suited to bursty data transmission, but does not guarantee quality of service. In addition, packet based service was thought to be only for delay tolerant applications, hence GPRS gives priority to circuit-switched traffic and

[††††] GSM was stand for Groupe Spécial Mobile, but it was re-branded in 1992 as Global System for Mobile communication; to indicate it is not just a pan-European wireless telecommunication system [11].

initially offered only "best-effort" quality of service. Due to error correction, the throughput may vary due to network conditions, hence affecting the performance which the user will experience at a given location under specific network conditions. On the positive side, GPRS is designed to be "always-on" which is beneficial for bursty data transmission as it avoids the long setup times for circuit-switched calls [11].

### 2.2.2 GPRS Adoption around the World

Just like GSM, GPRS quickly became widespread. In 2000, the first GPRS network was launched in England ($O_2$). Shortly afterward, T-Mobile launched this service in Germany and it quickly spread throughout Europe. By 2003, GPRS was offered by more than 200 network operators in ~50 countries. Thus, one-third of the countries offering GSM adopted GPRS within approximately two years [11].

### 2.2.3 GPRS as part of the evolution of GSM

GPRS is also known as GSM 2.5G. This later name makes obvious that GPRS does not replace GSM; but it is an evolution of GSM. Its purpose is to improve data transmission in mobile telephony systems. A GPRS phone offers better data services and in some cases can simultaneously offer GSM circuit-switched services [11].

Via a circuit-switched call, the maximum throughput is 9.05 kbps, but GPRS’s throughput can theoretically reach up to 171.2 kbps [12], but in practice limited to 50 kbps. In circuit-switched GSM, the operator charges the customer based upon the call’s duration (i.e., per minute), but the introduction of GPRS changes this business model allowing the operator to charge the user based upon the number of packets (or bytes) that have been transmitted (i.e., per kilobyte). Setting up, a new circuit-switched call takes about 10 seconds, but GPRS is thought of as "always-on". Although, in reality it is not actually always on, as there is the need to both set up a Packet Data Protocol (PDP)[‡‡‡‡] context and time needed to allocate resources, but these operations are sufficiently faster than the 10 seconds required for a circuit-switched call setup that it can be considered “always on” [11]. Details of GPRS will be presented in section 2.2.4.

GPRS data transmission enables icons, photos, images, music, and videos to be sent or received within a (hopefully) acceptable time. GPRS enabled video streaming had been impossible with single channel circuit-switched GSM technology because of its low throughput; however, multiple circuit-switched channels could be bundled together using HSCSD to support video streaming. An important feature of GPRS was the ability to check for new E-mails. Furthermore,

‡‡‡‡ The PDP (e.g. IP, X.25, or Frame Relay) context is a data structure present on both the SGSN and the GGSN which contains the subscriber's session information when the subscriber has an active session. For further information go to http://www.freepatentsonline.com/EP1351528.html or http://en.wikipedia.org/wiki/PDP_context#PDP_Context

GPRS made internet access available at a reasonable speed to customers with a PDA, Pocket PC, laptop, or other appropriate device [11].

It is considerable to notice the change in operators' business model with respect to revenue sources. The trends which are important to note are declining of voice as a source of revenue and the increase in messaging and data traffic. Thus data services are viewed as an important new source of revenue for GSM network operators.

### 2.2.4 GPRS Data Services and Infrastructure

As noted previously, GPRS was built on the GSM network infrastructure to transport data using packet-switching. It uses the radio interface in a different way that for circuit-switched GSM. By sending packet data as data frames to gateways, then onwards to data networks such as the internet, GPRS provides a convenient extension of the internet to mobile devices.

When adding GPRS service to the existing circuit switch network two new types of nodes have to be added, these are called GPRS support nodes (GSNs): the serving GSN (SGSN) and the gateway GSN (GGSN). Each of them will be described in detail in the following sections.

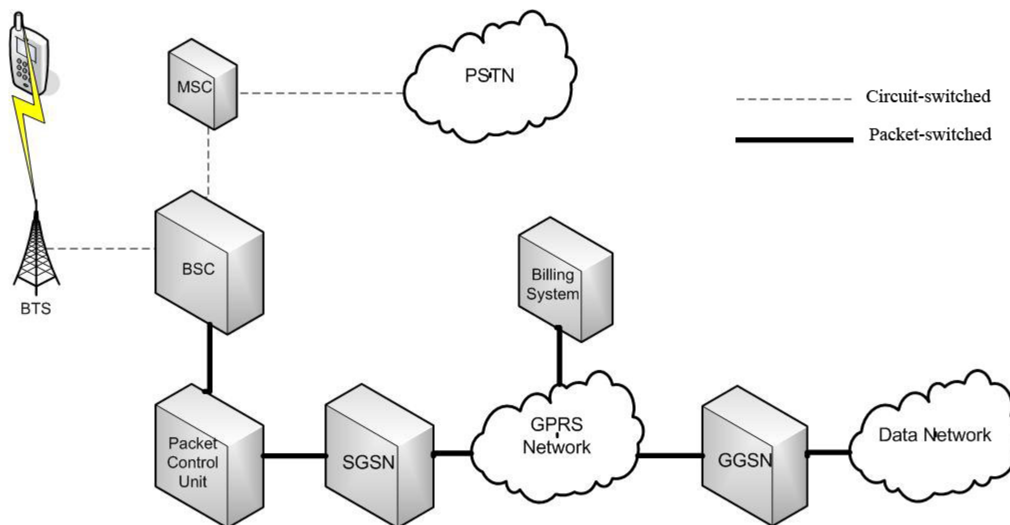


Figure 5: Components in GPRS network (basic schema without all connections) [11]

#### *2.2.4.1 SGSN*

When data arrives at the access network's Base Station Subsystem (BSS), it is bundled by the Packet Control Unit (PCU) and forwarded to the SGSN. The SGSN operates as a router: forwarding the packets toward their network destination or receiving packets and forwarding them to the handset [11]. Authentication and authorization of the handset is also done by the

SGSN [12]. Since the handset (for packet access) is a mobile station its location may change, thus mobility management (i.e., supporting handoffs) is an important task that is also performed by the SGSN. In addition, since the SGSN is the end point of packet-switched communication (as seen by the operator's internal packet-switched core network), it is responsible for some of the tasks which the BSS performs for the circuit-switched network, such as encryption and compression. Moreover, it supports the billing process by collecting charging information [11].

#### 2.2.4.2 GGSN

The GGSN acts as a gateway to the outside world for the GPRS network. When the Mobile Station (MS) moves in and out of the routing area of a given SGSN, it connects to a new (local) SGSN. However, this may change the IP address of the node and this change should be hidden from the exterior network; therefore, the GGSN provides a fixed address for this MS to the external network, but remaps this external (globally routable) IP address to the appropriate internal IP address. This task is known as the "anchor function" in GPRS [11]. One can also view the GGSN as a mobile IP home agent.

### 2.2.5 Frequency and Coverage

Although GPRS is carried via the physical channels of GSM, it employs dedicated packet-based logical channels. Additionally, it can use different coding schemes, the most common four coding schemes are: CS1, CS2, CS3, and CS4. These coding schemes use redundant bits to protect the transmitted data. CS1 has the most redundancy, whereas CS4 has no redundant bits; but does utilize interleaving. Therefore, CS1 provides the lowest throughput and conversely CS4 the highest. Unfortunately, we cannot use CS4 all the time due to the requirements on the MS's received signal power. The mobile station utilizes the received power to selects the appropriate coding scheme.

**Table 1: Coding Parameters for GPRS Coding Scheme [11, 12]**

| *Channel Name* | *Code Rate* | *Radio Interface Rate per Time Slot (kbps)* |
|---|---|---|
| ***CS1*** | 0.53 | 9.05 |
| ***CS2*** | 0.66 | 13.4 |
| ***CS3*** | 0.8 | 15.6 |
| ***CS4*** | 1 | 21.4 |

Since each coding scheme has its own carrier to noise ratio (C/N) requirements, the coverage area for each of them is different. The best signal quality is provided closed to the antenna, here users can use CS4; while in a concentric ring fashion, as the distance increases, the user is limited to lower throughput as they must use more redundancy due to the lower signal quality - eventually using CS1 far from the antenna [11]. This is shown in figure 6.

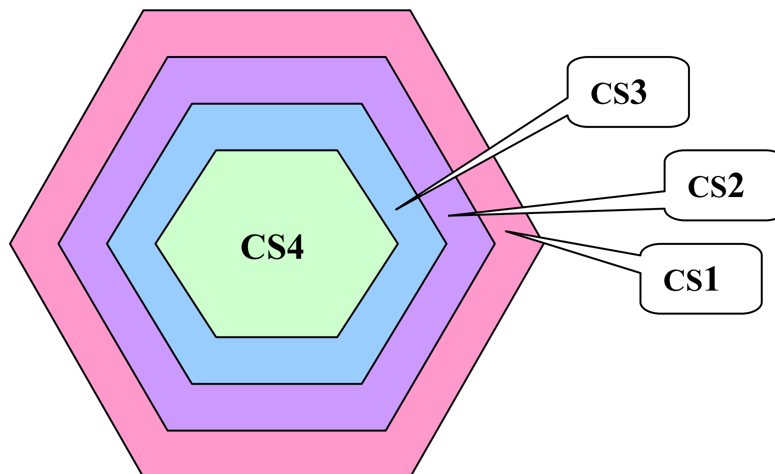


Figure 6: Concentric cells of coverage in GPRS [11]

Using CS1, you can provide coverage to 95% of the cell, but if you want higher rates everywhere, you need to split the cell by installing more antennas and thus use additional frequencies. As the available frequencies which a single operator can use are limited, the optimal assignment of frequencies becomes a graph color problem.

### 2.2.6 Capacity and Dimensioning for Growth

In GSM, as in the other radio networks, there are limited radio resources that can be used to support multiple users. When we add GPRS to the existing GSM network, these radio resources should be shared between circuit-switched calls and packet-switched data. Adding packet-switched services to the existing network without allocating new spectrum could affect negatively affect the voice capacity by increasing the blocking probability within each cell or require shrinking the cell service area (hence requiring the installation of new base stations). Alternatively, dedicating resources to the circuit-switch service will reduce the maximum throughput for GPRS users [11].

Fortunately, GPRS can efficiently utilize the unused radio resources of the GSM network to transmit data packets. When a GSM network is working with blocking probability of 2 percent, the channel load average is only 60-80 percent of the total cell capacity. Therefore, GPRS can

use on average the 20-40 percent of idle channels to transmit data **without** negative effects on the voice capacity of the cell [11].

In GSM, the (logical) channel for voice or circuit-switched transmission is called a traffic channel (TCH) and in GPRS, the logical channel for packet-switched traffic is called a packet data channel (PDCH). These PDCHs are shared between users in the cell. Each cell that supports GPRS should allocate some resources to both TCHs and PDCHs [11].

Assigning TCHs and PDCHs is done dynamically based upon demand. Physical resources are allocated to GPRS when there is a need for packet transmission. If there is no need, then no resources need to be allocated to GPRS. The maximum number of PDCHs with different numbers of timeslots allocated to a MS at a time within a carrier is limited to eight (which represent the complete resources of a carrier or channel). Thus, many GPRS users can receive service in a cell by sharing the available bandwidth. The number of users is dependent upon the applications used and the amount of traffic transmitted or received by each of these users [11].

#### *2.2.6.1 Packet-Switched Data Traffic Dimensioning*

To reach the theoretical maximum GPRS data transmission rate, which is 171.2 kbps, means that a single user would occupy all eight timeslots (in GSM) and would communicate without any error protection. However, an operator may or may not want to allocate this much capacity to a single user. In addition, early GPRS terminals supported only a limited number of timeslots (generally one to three) due to limitations on the power amplifier of the handset. Thus, the available bandwidth for a GPRS user is generally restricted to much less than the theoretical maximum rate. Increased data rates are offered via GSM evolution (EDGE) or 3G W-CDMA. [11].

#### *2.2.6.2 Core Network Dimensioning*

Separating of packet-switched and circuit-switched traffic in a GSM network is performed in the base station controller (BSC). The traffic load over the link between the BTS and BSC is increased due to added packet data traffic - which is significant because the data rate in the minimally coded GPRS frames is much higher than the bit rate for a normal voice coded frame. After splitting the traffic, packet data is sent to the SGSN through a fast Ethernet interface (called Gb) and the circuit-switched data is sent to the MSC. The other new interfaces in the GPRS network follow dimensioning rules for data networks. In a GSM network, if the overlay network (GPRS) is capacity limited, then capacity planning is based upon an estimation of the "active" and "stand by" users within the coverage area of each BSS; if the overlay network is coverage limited, then the SGSN capacity will be the main dimensioning parameter for the Gb interface. Some additional new entities are added to the existing GSM network, such as the

Packet Control Unit (PCU) and changes to the billing system; along with some new interfaces such as Gs and Gr [11].

### 2.2.7 GPRS Network Optimization

The optimization of a GPRS network is performed in three steps: GSM network optimization, the addition of value-added services, and the integration of these services into the GSM network.

GSM network optimization primary concerns the radio interface, coverage, the core network, and other dimensioning issues. Good coverage is the most important optimization for GPRS. Since coverage does not mean that the average distance to an antenna is short, rather good coverage can be provided when there are lots of small cells on average and they are located where the majority of users are. The throughput of a GPRS session is strongly dependent on the coding scheme used to protect the data. When a MS starts a GPRS session, it always begins by using the CS1 coding scheme regardless of its position in the cell and the signal to noise ratio. Afterwards, based upon the measurement reports exchanged with the base station, if the C/I ratio[§§§§] exceeds the C/I threshold for the use of CS2, and then the MS automatically switches to CS2. The same procedure occurs for switching to CS3 and CS4 [11].

As mentioned before, CS4 uses no redundant information, hence it offers no protection, thus it has maximum throughput (of these four schemes). However, areas with weak signal coverage force the mobile station to use additional redundant data as protection and therefore the mobile experiences lower throughput.

Dimensioning is also an element of GSM optimization since the voice and data users share the network's resources. Although GPRS uses dynamic resource allocation in order to support specific QoS requirements for both circuit and packet transmission, the correct amount of resources should be maintained in each cell. Therefore, there may be a period of time when there are not sufficient resources for GPRS in a cell – which will lead to reduced GPRS performance. The solution to this requires that the operator redimension the resources allocated to this cell.

The second step in optimization is to ensure that a packet data session starts and continues with the expected performance of a similar session over a wired IP network. All the main QoS parameters should be checked in this phase. The main issue here concerns the user's expectation, if the user gets much worse performance than they expect they are unlikely to continue to use the service [11].

The final optimization step involves examining the overall network performance with an emphasis on the cooperation between the GSM and GPRS network. In this thesis we assume that

---

[§§§§] Carrier-to-Interference (C/I) ratio is the ratio of power in a Radio Frequency (RF) carrier to the interference power in the channel. A high C/I ratio yields better quality in communication.

the operator has performed all of the steps necessary to optimize their network and that the network provides the best GPRS performance that it is capable of.

### 2.2.8 Reliability, Jitter, and Latency

QoS support in GPRS is minimal, but it is possible to ensure the integrity of received data through the implementation of two reliable modes of operation: Radio Link Control (RLC) Acknowledged and Logical Link Control (LLC) acknowledged. RLC acknowledged mode is used to ensure that the data received by/from the MS is without error by default. LLC acknowledged mode is an optional feature which ensures that all LLC frames are received without error. However, use of LLC decreases throughput because the correct receipt of all LLC frames requires each LLC frame to be acknowledged [13].

Latency is the time taken for data packets to pass over the GPRS bearer, normally expressed as a round-trip time. In GPRS there are a number of factors contributing to the overall latency which include: the Mobile Station (MS), radio resource procedures, the effective data throughput, and the GPRS core network nodes [13].

MS delay is the time taken by the MS to process an IP datagram and request radio resources. This delay is usually less than approximately 100ms. The delay depends on details of the specific MS. Radio resource procedures are the major source of delay in GPRS. In order for the MS to be capable of sending or receiving data, a radio resource known as a Temporary Block Flow (TBF) must be made available to the user. If no TBF is established, then the MS and network must exchange signaling messages to establish a TBF. The time taken to successfully attain an active TBF depends on the availability of radio resources and is different for the uplink and downlink directions, thus it may cause a delay of hundreds of milliseconds. If a TBF is currently active, then the MS may use it - thus minimizing this portion of the delay. Once established, the TBF will generally remain active for as long as there are LLC frames to transmit. [13].

Effective data throughput (over-the-air delay) is the rate at which user data is transmitted between the MS and the SGSN over an active TBF. This transmission delay is directly related to the size of the IP datagram being sent. Smaller packets experience less delay. This delay is reduced when multiple timeslots are used. Since the packets must be mapped onto the resources available in each timeslot, there is a relation between a IP packet size and coding used – this affects the number of timeslots used to send a given sized IP datagram. Additionally, as a whole timeslot is always used, this means that making too short an IP datagram simply wastes space in the last timeslots, hence it is important to use an appropriate choice of maximum transfer unit (MTU) in combination with the current coding scheme. The effective throughput is also dependent on the number of re-transmissions resulting from the RLC Block Error Rate (BLER). RLC BLER displays the percentage block error rate for downlink RLC. The downlink RLC

BLER is made when the TBF is open and calculated over a 1 second or a 150 bits block window (whichever is reached first). BLER is computed as the percentage of blocks with bad CRC[*****] over the reporting period.

Core network delay occurs as packets transit the core network from the SGSN to the GGSN (and the reverse). These nodes act as IP routers and hence will have a relatively low impact on the overall latency. However, under high load conditions the transit delay may increase due to the contention for the link resource with other traffic.

## 2.3 E-model

The quality of voice that is important to us for evaluation. However, this quality is a parameter dependent on many factors and is somewhat ambiguous to measure. Thus, we use the ITU-T E-model. The ITU-T E-model is an analytical model of voice quality for hybrid circuit-switched and packet-switched network. It is based on the calculation of an R-factor that ranges from 0 to 100, to describe the quality of a voice signal. The R-factor is related to MOS[†††††] (Mean of Opinion Score) as follows [14]:

For $R<0$: $MOS=1$
For $R>100$: $MOS=4.5$
For $0<R<100$: $MOS=1+ 0.035\,R+ 7* 10^{-6}\,R\,(R-60)(100-R)$

This Equation can be reduced to:
$R= 94.2 - I_d - I_{ef}$

In which $I_d$ is the impairment associated with the mouth-to-ear delay of the path and $I_{ef}$ is an equipment impairment factor associated with the losses within the gateway CODECs [14].

In the paper "Voice over IP Performance Monitoring" [15] by Cole and Rosenbluth, the above equation was investigated leading to table 2. This table shows the values for the delay component for selected values of the one way delay.

As can be seen, if the above data is plottted, a knee in the curve occurs at a delay of 177 msec. Thus, for one-way delays, to having a natural voice quality, the delay should be less than 177 msec, because after that the $I_d$ value increases at a high rate [15].

[*****] A cyclic redundancy check (CRC) is a type of function that takes as input a data stream of any length and produces as output a value of a certain fixed size. A CRC can be used as a checksum to detect accidental alteration of data during transmission or storage.
[†††††] A MOS ranges from 0 to 5, where 5 is excellent, 4 is good, 3 is fair, 2 is poor, and 1 is bad.

Table 2: Relation of Id with one-way delay [15]

| *One-way Delay (msec)* | $I_d$ |
|---|---|
| ***0*** | 0 |
| ***25*** | 0.9 |
| ***50*** | 1.5 |
| ***75*** | 2.1 |
| ***100*** | 2.6 |
| ***125*** | 3.1 |
| ***150*** | 3.7 |
| ***175*** | 5.0 |
| ***200*** | 7.4 |
| ***225*** | 10.6 |
| ***250*** | 14.1 |
| ***275*** | 17.4 |
| ***300*** | 20.6 |
| ***325*** | 23.5 |
| ***350*** | 26.2 |
| ***375*** | 28.7 |
| ***400*** | 31.0 |

A modification of E-model is described in the recent paper called "An E-Model Implementation for Speech Quality Evaluation in VoIP Systems" [16]. This paper modifies the E-model to:

*For R < 6.5: MOS = 1*
*For 6.5 < R < 100:*
$MOS = 1 + 0.035 R + 7 * 10^{-6} R (R-60) (100-R)$
*For R > 100: MOS = 4.5*

These modifications were made due to some shortcomings in the ITU-T E-model. Note that in case of end-to-end VoIP there are not CODECs in the gateways – since the parties agree upon a mutually supported CODEC. Hence, $I_{ef}$ is zero. The focus in this thesis will be explicitly **only on the measurement of the path delay**.

# 3. Voice Quality over GPRS

In this chapter, we investigate the GPRS network and voice quality issues using recent papers and articles in this area.

## 3.1 VoIP in details

### 3.1.1 VoIP CODECs

The packet sizes used in VoIP transmissions are in the range of 120–320 bytes (plus packet header bits). This packet size range is based upon voice packetization (10–40 milliseconds) which is encoded by different voice CODECs (Coder/Decoder). For instance, using the G.711 μ-law CODEC and a 40 milliseconds voice packetization time, the packet size is 320 bytes (bit rate of CODEC (64 Kbps)*packet size (40 ms)/8=320 bytes). Moreover, the maximum bit rate of each of CODECs is different. G.723 and G.729a need 6.3kbps and 8kbps bandwidth respectively versus G.711 which generates data at a rate of 64kbps. The availability of more bandwidth allows a higher bit rate which enables lower compression and hence better sound quality [17]. While this is very obvious at very low bit rates, the behavior is not linear – this is especially true at higher bit rates and with more complex encodings. However, in this thesis we will focus on bit rates of 64kbps and lower, as these are feasible to carry GPRS.

Some of the factors that specify the required bandwidth for a voice over IP application are: CODEC and sample period, IP header size, transmission medium, and silence suppression. Primarily, it is the CODEC which determines the actual amount of bandwidth needed for a given sampling rate. The IP/UDP/RTP header adds a fixed overhead of 40 octets per packet (i.e., RTP: 8 bytes, UDP: 12 bytes, IP: 20 bytes). Each transmission medium, be it Ethernet or radio link adds its own headers, framing, and checksums. Some CODECs exploit silence suppression which can reduce the required bandwidth by up to 50 percent [18].

The CODEC is responsible for converting the analogue waveform to a digital form (and at the receiver doing the reverse). Each CODEC samples the waveform at specific intervals (based upon the sampling rate) which is typically in 8,000 times a second. These samples are accumulated for a fixed period to create a frame of data. A packetizing period (also called the audio frame duration) of 20 ms is common, but 30ms (for G.723.1 CODEC) and 10ms (for G.729.a period) are also used [18].

We can say that the main characteristics of a CODEC are number of bits produced per second, the sampling rate, and the packetization period. Given this information we can calculate the size of the IP frame. For example, a G.711 CODEC packetizes 20 ms worth of audio which generates 50 datagrams per second. As G.711 transmits 64000 bits per second, hence each frame will

contain 64000/50 = 1280 bits which is 160 octets of audio payload [18] To this payload we have to add the overhead of the IP header, UDP header, RTP header, and any link layer framing.

Most IP phones simply place each audio frame into its own packet, but some implementation can put more than one audio frame in each packet, e.g. the G.729a CODEC utilizes a 10 ms audio frame and generates a small frame of only 10 bytes. Therefore, putting two such audio frames in each IP packet is more efficient and decreases the packet transmission overhead without significantly increasing the end-to-end latency. Choosing the number of audio frames to place in each IP packet is a trade-off, because the high latency of long audio frames affects the perceived quality of the call. While the shorter the audio frame, the better the perceived quality of the call, but the greater bandwidth usage because of each packet's header overhead [18].

As an example, the payload for the G.711 CODEC and 20 ms audio frame length calculated results in a 160 octets payload, while the IP header adds 40 octets. Hence, 200 octets (1600 bits) are sent 50 times a second resulting in a 80,000 bps total rate. This is the required network layer bandwidth to transport this CODEC's output using Voice over IP, in addition to this we must also take into account the physical transmission medium overhead as well [18]. Note that the choice of VOIP packet size (in ms) affects packet loss rate, latency, bandwidth, and congestion especially over WAN or cellular links. The larger the packet size, the lower the protocol overhead is, but the greater the delay and the greater the packet loss caused by a given rate of bit errors.

Table 3: VoIP CODECs Bandwidth Requirement [18]

| CODEC | Bandwidth | Sample Period | Frame Size | Frame/ Packet | Ethernet Bandwidth |
|---|---|---|---|---|---|
| **G.711 (PCM)** | 64 kbps | 20 ms | 160 | 1 | 95.2 kbps |
| **G.723.1A (ACELP)** | 5.3 kbps | 30 ms | 20 | 1 | 26.1 kbps |
| **G.723.1A (MP-MLQ)** | 6.4 kbps | 30 ms | 24 | 1 | 27.2 kbps |
| **G.726 (ADPCM)** | 32 kbps | 20 ms | 80 | 1 | 63.2 kbps |
| **G.728 (LD-CELP)** | 16 kbps | 2.5 ms | 5 | 4 | 78.4 kbps |
| **G.729A (CS-CELP)** | 8 kbps | 10 ms | 10 | 2 | 39.2 kbps |

G.711 uses linear pulse code modulation (PCM). This is an **uncompressed** audio format. This encoding was used in this thesis project. Here each audio sample was digitized into 16 bits using an analog to digital convertor, this generates 128 Kbps of data (16 bits/sample*8000 samples/s). Next these samples were μ-law coded, hence the voice samples are compressed to 8-bit samples, thus reducing the stream's bandwidth to 64 Kbps. At the receiver's side, the voice samples are

retrieved from the RTP packets, decompressed (in the case of compressed samples) and placed into the play-out buffer of the sound card, and finally played back [19]. [Figure 7]

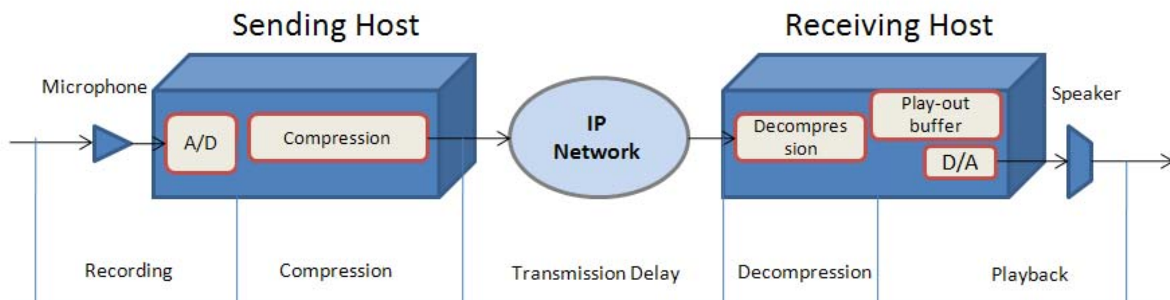


Figure 7: Voice over IP implementation [19]

The total end-to-end delay between the source and destination consists of different delay components: recording, compression, transmission delay (includes packetization and network delays), decompression, and playback [19].

### 3.1.2 IP header compression

The large headers due to the network and transport protocols, especially for IPv6 add significant overhead; consequently, significant channel bandwidth will be utilized transporting header information. Header compression is possible due to a high degree of correlation between the header fields of consecutive packets. Header compression methods work through a shared state called context at both ends of the link, this state includes the uncompressed version of the last sent header. Therefore, only unpredictable changes need to be transferred. Static header information is transmitted only once, while dynamic information – i.e., the changes from the previous packet- are transmitted whenever changes occur by using only a few bits. When packets are lost over the link, the synchronization of decompressor context will be ruined, hence decompression of subsequent headers will be unsuccessful. Thus, in addition to methods for initializing and updating the context, header compression schemes must also have mechanisms for detection of incorrect decompression and implement ways to repair an invalidated decompressor context.

Since IP telephony services typically use RTP, we can take the advantage of the fact that the IP/UDP/RTP headers can be compressed from 40 or 60 octets down to as little as 2 octets by using the CRTP (Compressed RTP) header compression protocol [20]. To repair an invalidated context, CRTP utilizes signaling messages for the decompressor to request a context update from the compressor. Hence, the link round trip time affects the delay of updating the decompressor context. Unfortunately, while the context is invalid, all of the received packets will be discarded,

thus a long delay in updating the context will generate a large number of lost packets. On lossy links with long round trip times, such as cellular links, CRTP does not perform well [20]. In other word, IP telephony in cellular systems has problems using CRTP in terms of speech quality due to the error-proneness and long round trip times of cellular links [20].

An alternative is ROCCO (Robust Checksum-based Compression) [20] another header compression algorithm, which claims to solve this problem by making the IP/UDP/RTP header compression more tolerant to packet loss. Headers are compressed down to 1 octet per packet. This kind of compression is possible because ROCCO utilizes compression profiles which use knowledge about packet contents for different types of traffic. ROCCO depends on local decompressor repair of the context by adding a CRC covering the original uncompressed header in the compressed header. With the aid of this mechanism, the decompressor can perform several reconstruction attempts to find the correct header. It has been shown [20] that ROCCO considerably reduces the negative effect of high packet loss and long round trip times on header compression performance. Larzon et al. [20] compared ROCCO to CRTP over cellular links and showed that CRTP packet loss is 6 times greater than ROCCO, causing significantly decreased speech quality.

## 3.2 A deeper look at GPRS

### 3.2.1 GSM physical and link layer channel

GSM uses 200 kHz frequency channels consisting of eight TDMA channels by dividing each frequency channel into eight time slots. Each time slot of a TDMA frame lasts for a duration of 156.25 bit times and, if used, contains a data burst [24]. Within a GSM cell, there are one or more carrier frequencies and over each carrier a TDMA frame of $Tf$ = 60/13ms is defined, consisting 8 slots of 15/26 ms each. Each bearer channel is defined by a slot in the TDMA frame, and its related carrier frequency.

Since some channels are allocated for signaling, each carrier frequency offers 6 to 8 channels for the transmission of user information, depending on the cell's configuration; a cell can provide two different services: telephony and data transfer. Telephony service relies on the usual circuit-based GSM service; and data packet transfer follows the GPRS standard, using the same radio resources as deployed for circuit-switched telephony [25].

Based on the GSM service provider's policies, different channel allocation strategies can be deployed for simultaneous delivery of telephony and data transfer services. The typical allocation policy is called *voice priority*: which is based upon the primary role of the circuit-switched telephony service (because this is currently the main source of revenues). Telephone calls are set up while channels are available in the cell; therefore, data packets can be transmitted

only in the timeslots or channels which are not used by voice connections. Another approach to resource allocation, called *hybrid,* may be applied when the telecommunication service provider expects that GPRS users are a few in number, hence allocating a static channel reservation would result in inefficient use of radio resources. In this case, there may be no GPRS user for long time intervals, thus mechanisms to detect the presence of active GPRS users and to only reserve channels for data traffic when there are active users can be used [25].

A wide area cellular telecommunications system typically supports user mobility, i.e. users can roam from a cell to a neighboring cell during an active voice call and the system must execute a handover procedure which transfers the call from the channel in the old cell to a channel in the new cell without interrupting the communication. However, if no channel is available in the new cell, then the call is will be blocked and hence terminated. Unfortunately for GPRS users, since the duration of a data transfer is assumed to be small, the possibility of roaming is often neglected by the operators [25].

### 3.2.2 GPRS layers

The communication between the mobile station (MS) and the GPRS network is handled by the physical- and data-link-layer functionalities. The physical layer functions consist of modulation, demodulation, channel coding/decoding, etc. The data-link layer includes two sub-layers: logical link control (LLC) and radio link control/medium-access control (RLC/MAC) [27].

LLC Layer is operating between the MS and the SGSN and provides a logical link between them [28]. The functions of the LLC layer includes link-level flow control and ciphering. Packet data units (PDUs) from higher layers (IP layer) are divided into LLC frames with variable sizes. Practically, the LLC layer may work either in acknowledged mode or in unacknowledged mode. In unacknowledged mode, the LLC layer is not responsible for the recovery of erroneous LLC frames and the either are erroneously received or are passed on to the higher layers. In the acknowledged mode of operation, the LLC layer utilizes an ARQ[‡‡‡‡‡] mechanism to retransmit erroneous LLC frames. A frame-check sequence (FCS) is provided in each LLC frame to detect LLC frame errors [28, 29].

The RLC layer is provided below the LLC layer and above the MAC layer [30]. The RLC peers are at the MS and the base station system (BSS). The RLC layer segments each LLC frame into several RLC data blocks and each of them resides in four time slots, irrespective of the channel coding scheme used. Coding schemes CS-1, CS-2, CS-3, and CS-4 are defined respectively with rate 1/2, 2/3, 3/4, and 1 (no coding) [30]. For example, in CS-1, each RLC block consists of 181 information bits, 40 block-check-sequence (BCS) bits, and 7 tail/control bits [27]. Moreover, it

[‡‡‡‡‡] Automatic Repeat Request is an error control mechanism achieved by retransmission of erroneous data packets

should be noted that the basic data unit transferred over a GPRS Packet Data Channel (PDCH) is an RLC block. It is transmitted during one block period, which is a sequence of four timeslots on a PDCH. The RLC data block consists of a RLC Header, RLC Data field, and spare bits [31].

## 3.3 VoIP QoS in GPRS and WLAN

Agreement upon a common protocol (in our case IP/UDP/RTP for media and SIP for signaling) is only the first step in unifying VoIP over different types of links. The main challenge in the case of GPRS and WLAN is ensuring the QoS required for real-time services in such an all-IP network. Actually when it comes to real-time services rather just web browsing, we need some improvements in order to achieve guaranteed QoS for real-time applications. As the VoIP industry is evolving, some improvements have been made in wired networks, but the main challenge is to allow real-time applications over packet switched mobile networks, such as GPRS or EDGE [32]. As mentioned before, the major factors affecting QoS of a voice call over a packet switched system are: delay, packet inter-arrival time, and loss rate, out of order packets, packet overhead, and network efficiency [33].

### 3.3.1 Designing a QoS protocol for real-time services

The design of new protocols with QoS support will enhance the quality of real-time voice communications. However, these protocols still cannot guarantee delivery within a definite time because they may use an over-subscribed network path or an unreliable network medium (such as a wireless link with interference). For example, IPv6 handles and prioritize the traffic flows of real-time services differently, but still does not guarantee link reliability [34]. Definitely, unless admission control is employed to guarantee bandwidth, packet loss will be unavoidable when periodic data has to be received under time constraints. Because using an end-to-end protocol which automatically retransmits lost or delayed packets on request by the receiver is not practical in interactive voice communications when the one-way delay should be under 200ms and bandwidth is limited; hence VoIP systems can make use of loss-concealment schemes that utilize redundancies in voice data to reconstruct lost data from that which was successfully received. However, the design of such schemes is difficult because coding algorithms introduce dependencies in a compressed stream. In this case, loss concealments should be applied not only to a lost frame, but also to all subsequent dependent frames that cannot be decoded even if they are received correctly [34].

In traditional telephony, a call admission control (CAC) mechanism is invoked when the number of call attempts surpasses the link capacity, thus subsequent requests for setting up new calls will be rejected in order to ensure that the system can provide QoS for all of the calls in progress. Most current IP networks do not have any CAC mechanism and simply offer best-effort services which means that, new traffic may keep entering the network even beyond the network’s capacity limit, as a result causing increased packet loss and potentially significant delay increases

for both existing and new flows [35]. However, the advantage is that communication is still possible – rather than simply being blocked as it would be in the traditional telephony system.

To prevent such QoS problems, a CAC mechanism should be introduced in VoIP networks in order to make sure that adequate resources are available to provide the QoS needed for both new and existing calls after a new call has been admitted. Current VoIP systems have become aware of the importance of call admission control to provide QoS guarantees. Current VoIP systems have used several types of CAC mechanisms but none of them are able to provide end-to-end QoS guarantee because of imperfectly applying and supporting the CAC mechanism [35].

### 3.3.2 QoS mechanisms for VoIP over WLAN

In order to provide QoS for Voice over WLAN, two QoS mechanisms are useful: call admission control and distributed channel access control. Call admission control usually runs at the access point (AP) and on a per call timescale. The call admission decision is applied in such a way that it is feasible (via arbitrary distributed channel access) to provide QoS for all existing and newly requested voice stream. Distributed channel access control runs at the mobile stations and on a packet scheduling timescale. Each mobile node competes for a shared channels before transmission. To provide a better QoS level for real-time services, many approaches allocate priorities to prioritize real-time traffic over non-realtime traffic. Enhanced Distributed Channel Access (EDCA) defined in IEEE 802.11e [36] supports four access categories, each of which provides differentiated channel access by varying the inter-frame spacing, and the initial and maximum window sizes for back off procedures [37].

While a lot of studies have been done in the area of QoS provisioning in WLANs, two issues are still not resolved. Firstly, supposing distributed channel access control is applied, then only service differentiation can be offered, thus we still have no QoS guarantee. Secondly, most of the existing schemes (including 802.11e EDCA) necessitate upgrading or replacement of hardware in both DCF[§§§§§]-based APs and mobile stations, this makes its implementation impractical. Some researchers [38] have proposed a dual queue strategy, which runs at the MAC layer and does not require modification of the existing hardware. However, this approach still does not provide QoS guarantees for VoIP flows because it is fundamentally still a best effort scheme and does not consider the global traffic conditions [37].

Some studies have been done to find out how we can support better QoS than differentiated services for VoIP calls with the broadly deployed DCF, while achieving high channel utilization. These studies, such as [37], prove that in the IEEE DCF protocol, the strict delay and delay jitter requirements can be statistically guaranteed if the instantaneous total traffic contending for the channel can be kept below the network's capacity. If this is true, then the probability of a collision is small, and the MAC delay is short enough to support voice calls. If the arriving

[§§§§§] Distributed Coordination Function (DCF) is the fundamental MAC technique of the IEEE 802.11 wireless LAN standard.

traffic exceeds the link's capacity, then WLAN enters saturation, causing a significant increase in delay, thus decreasing the throughput. In other words, if the arriving traffic is less than the capacity, then channel capacity is wasted, but guarantees are possible. However, in order for the network to operate at the optimum point we require an efficient control algorithm to regulate input traffic [37]. Although, these studies are valuable, the problem can be avoided by simply over dimensioning the network as well.

Also, a novel call admission and rate control scheme has been proposed by Zhai to provide statistical QoS guarantees for VoIP [37]. This mechanism adjusts voice traffic to efficiently manage the medium contention among voice sources. The rate control mechanism regulates non-voice traffic to control its effect on the performance of voice traffic. An advantage of this scheme is that, it runs on top of the 802.11 MAC protocol, thus, no modification to the existing MAC controller is required.

It has been proved by Zhai that there is an optimal operating point for IEEE 802.11 DCF [37]. This operating point depends on the amount of arriving traffic and at this point the MAC protocol can optimally meet the QoS requirements of real-time traffic while simultaneously achieving maximal channel utilization. If the arriving traffic is beyond the threshold, then the WLAN immediately enters the saturation state, and the collision probability becomes very high. Hence, a significant increase in delay and a decrease in throughput occurs. On the other hand, if the arriving traffic is less than this threshold, then the collision probability is small, but channel bandwidth is wasted [37].

The standard Distributed Co-ordination Function (DCF) provides a fair share of the network resources to all stations. In this configuration all stations listen to the medium to make sure it is safe to transmit. If the medium is busy, then the stations wait for a random period and then try to transmit again. This is Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) usually referred to as "Listen before talk". It is obvious that this scheme is not appropriate for voice data as there is no guarantee that the station will be able to transmit and consequently there may be bursts of losses in conversations [39]. However, at low loads the probability of not being able to transmit within the allowed jitter period is small; hence in practice systems work without special QoS support as long as the load is low.

The Point Coordinated Function (PCF) is part of the IEEE 802.11 access control protocol, which resides above the DCF standard. It provides contention free service and guarantees for each PCF station a percentage of available bandwidth. PCF is usually used with multimedia applications where the data is to be used or processed in real time [39].

In 2001, the IEEE 802.11 committee developed an enhanced 802.11 standard to provide better QoS, and avoid the previously mentioned problems. The resulting 802.11e standard builds upon

the ideas underlying PCF and is compatible with the basic 802.11 standard, but provides a Hybrid Coordination Function (HCF) that replaces DCF and PCF [40].

Within HCF, there are two access mechanisms, the Enhanced Distributed Channel Access (EDCA) and HCF Controlled Channel Access (HCCA). HCF encapsulates and expands the functions of the legacy DCF and PCF. Through the combined work of the IEEE 802.11i working groups it would appear that 802.11 WLAN technologies are evolving to provide the underlying services necessary to allow the native use of VoIP on WLANs [39].

### 3.3.3 GPRS QoS and delay sources

In Internet the transmission delay can vary between milliseconds and infinity (a lost packet) depending on the geographical distance, number of hops, and network congestion. Additionally, delays in the Internet have large variations and depend on many factors which make analysis of delays very difficult. Moreover, the voice CODEC used by the SIP or H.323 application adds encoding, decoding, and processing delays. Moreover, in order to smooth out delay variations, playback buffering is needed which also adds additional delay [41].

The main delay sources in a GPRS network are packetization, transmission, and buffering delays. Additional delay due to queuing before transmission and also processing delays in the terminal, BTS, BSC, SGSN, and GGSN are sometimes considerable (of the order of magnitude of tens of milliseconds each). As described earlier (see section 3.1), transmission delay over the air interface depends on the packet size. If the RTP packet consists of multiple voice frames, the packet size increases and subsequently the transmission delay will increase. This causes additional delay at the receiving end, because all the RLC blocks of an LLC frame must be received before the LLC payload can be passed to the SNDCP (Subnet Dependency Convergence Protocol) layer. Only after the whole RTP packet is received, voice frames can be input to the decoder. Assuming high capacity links in the cellular network infrastructure, the transmission over Gb and Gn interfaces causes only a very low delay and can be neglected [41].

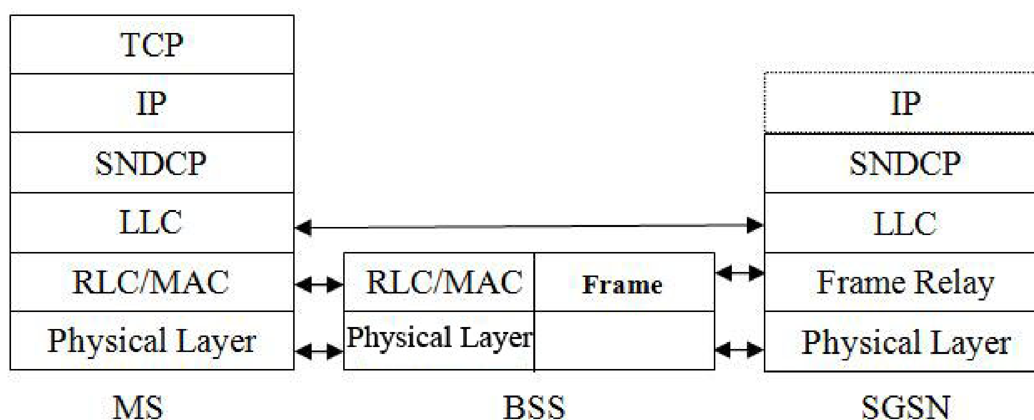


Figure 8: GPRS Protocol stack [42]

In Figure 8 the GPRS transmission plane protocol stack is shown. In order to support GPRS traffic, a cell may allocate resources on one or more physical channels. The physical channels shared by the GPRS MSs, are taken from the common pool of physical channels available in the cell. The BSS is the controller for allocation of these channels. In the MAC layer, an MS initiates a packet transfer by making a Packet Channel Request on Packet Random Access Channel (PRACH). The network responds on the Packet Access Granted Channel (PAGCH) [42].

The throughput of a radio network is essentially dependent on the Carrier-to-Interference Ratio (C/I) distribution within the cells. For voice services (in GSM), it is sufficient to have a certain C/I value in order to get sufficient speech quality; If the actual C/I value is higher than required, then no further speech improvement in terms of capacity enhancement can be expected. However, the situation for EDGE and GPRS is more complex due to the introduction of different coding schemes. In general we can say that the higher the C/I, the better the efficiency, i.e., increasing C/I (GPRS: up to 25 dB, EDGE: up to 35 dB) results in better throughput [43]. It has been proved by Hallman that one of the possible ways to increase C/I is planning the cell frequency in such a way as to have a high reuse factor, because this results in very high C/I values in the network [43]. Simulation for GPRS has proven that a very good C/I situation occurs in the networks with a reuse factor of 24, which results in high throughput values per time slot [43].

# 4. Measurements

In this section, we investigate the delay in different scenarios. The one-way delay measurement is enabled by synchronizing the equipment used. To accomplish this, we synchronize the both parties using GPS, then send packets over the GPRS link to the other party. Based upon the timestamps and their known relationship to GPS time, we can directly calculate the delay.

## 4.1 Time Synchronization

### 4.1.1 One-Way Delay Measurement

Accurate measurements and analysis of network characteristics are essential for ensuring robust network performance and enabling high quality services. Analysis of such data plays a vital role in the design and evolution of the network; and is essential to ensure user satisfaction [44]. One of the most important network performance characteristics is delay, as it strongly affects the performance of network services such as voice and video. Delay measurements are common in such environments. Moreover, continuous monitoring of delay is essential in many applications in order to ensure conformity with critical delay constraints [45].

In many cases, the path from a source to a destination may differ from the path from the destination back to the source [45]. Even when the two paths are symmetric, in quality-of-service (QoS) enabled networks, provisioning in one direction may be different from provisioning in the reverse direction, thus the QoS guarantees differ [44]. Fortunately, performance of many applications depends mostly on the delays in one direction [45]. For example, a streaming application's performance depends primarily on the characteristics of the path from the source to the destination [45] (assuming that the source is not prevented from further transmission because of high delay in the reverse channel). Also, a file transfer may depend more on the performance in the direction that data flows, rather than the direction in which acknowledgements travel [44] (However, here is an example where the failure to provide timely acknowledgements will limit the rate at which the window is opened and will delay recovery from congestion control.). Finally, for voice and video conferencing each unidirectional path needs to provide timely delivery. Consequently, the capability to measure or estimate one-way delays is very important [45].

The main difficulty in measuring one-way delays is due to the difference in clock offsets of each of the parties involved in the measurement. Fortunately, the ability to synchronize to a global clock such as GPS or NTP makes one-way measurements quite simple. A node can send a probe packet containing a time stamp to the destination. When this destination receives the packet, it adds its own time stamp to this packet. The difference between these two time stamps is a one-way delay. Obviously, this one-way measurement equals the corresponding one-way delay **only** if the clocks of the two nodes are synchronized. Otherwise, the one-way measurement combines

the corresponding **one-way delay and the clock offset** between the nodes (in the case of unsynchronized systems the later) is unknown [45].

Outdoors the Global Positioning Systems (GPS) can provide accurate time synchronization between network nodes. The Network Time Protocol (NTP) is the current internet standard for synchronizing clocks -with respect to Universal Time-Coordinated (UTC) [46]. NTP is widely deployed in the Internet to synchronize distributed clocks to each other or to a time server having an accurate clock. Since NTP estimates the clock offset and skew based on one-way delay measurement across the network, the accuracy of synchronization depends on the stability and symmetry of network paths between two hosts [47]. NTP measures roundtrip delays and uses a halving procedure to estimate the clock offsets.

However, there are some novel synchronization protocols based on NTP messages that provides better accuracy by optimizing a global cost function, but all these clock synchronization procedures work accurately only when the delay is symmetric [45]. Accurate, reliable time-stamping which is convenient and inexpensive is needed in many applications including real-time network applications and measurements of network characteristics. Recently the TSC register, which counts CPU cycles in popular PC architectures, was proposed as the basis of a new software clock which performs as well as more expensive GPS alternatives [48], but while this can provide a stable local time reference – it does not (by itself) enable different nodes to synchronize their clocks.

Unlike one-way delay, round-trip delay measurements are simple to make and they are accurate because the same clock is used for all the delay measurements. Therefore, a common approach used for estimating one-way delay is to measure round-trip delays and halve them. This requires not only that the route between source and destination be the same, but also that the traffic loads and QoS configurations in both directions should be the same which are generally not true [45]. Thus this method of estimating one-way delay is not very accurate.

### 4.1.2 The Application

In order to determine the one-way delay in our measurements, we have chosen to use GPS because this will enable us to use a cellular phone and either another cellular phone or a fixed PC. However, it does limit our experiments as the GPS receiver's antenna has to be located in such a way that it has sufficient signal strength for the receiver to extract the time – even though these devices might be far from each other and connected via a variety of networks. As we want to measure the delay between several nodes between the source and the destination, it is better to directly measure the one-way delay instead of the round trip delay. Some applications to measure the synchronization accuracy were developed for this project.

Our first application consists of several independent portable classes written in C++ (using Visual Studio 2005). In the following paragraphs the structure of our application along with some results of the experiments which were performed are explained. The focus of these experiments was to determine the ability to synchronize the clock of the local device to the GPS clock.

The Sync application package has a client-server structure (the code is included in the Appendices A1 and A2). In this package the client and server transmit simple UDP packets between them to determine their clock offset relative to each other. Each packet consists of a sequence number and some timestamps with a total length of 100 bytes.

The procedure is as follows: The client sends a UDP packet to the server containing a sequence number and a timestamp when the packet was sent. When the server receives the packet, it adds its own timestamp and immediately sends its reply back to the client. The client inserts its own receiving timestamp and logs the resulting data into a log file. By utilizing these three timestamps the client derives two parameters: t1 and t2; each of which represents the transmission delay in one direction between the client and the server (respectively). Obviously, t1 and t2 are not the actual delays as they are relative to the timestamps, but by adding them together, the round trip time is calculated. Now that we are able to measure are the delays relative to these timestamps, the next step is to synchronize the two time bases (see section 4.1.3)

#### 4.1.2.1 Client-side

The client-side application consists of 3 classes: CSendSock, CRecvSock, and CGPSDevice.

The CSendSock class creates and sends UDP packets at a predefined time interval to a specified destination. It marks the packet with the sending timestamp while transmitting it. Note that it does not wait to receive a response, but simply sends packets at predefined intervals (i.e. at fixed time intervals).

The CRecvSock listens on a specified network interface and port number for packets and adds a received timestamp to them. Based upon this, it calculates t1, t2, and the round-trip time (RTT), and logs the results in a log file.

The CGPSDevice class utilizes the COM port and the user specified baud rate for the GPS receiver and reads the received data buffer as soon as it is available. Because of the low baud rate (4800 bps) it takes hundreds of milliseconds for the port to receive the complete data. Initially, I did not understand this received delay - which causes a time offset between the GPS clock and real-time. So, this class waits 300 milliseconds (actually because the biggest GPS message size is 81 bytes receiving it takes 135ms at 4800bps baud rate, but because of some errors in the devices used or the way of performing our measurements, we found by trial and error that 300ms is the

maximum time) in order to get the latest message from the GPS device via the serial communication port. This class records the time when the first character was received and subsequently corrects for serial receiving delay.

#### *4.1.2.2 Server-side*

The Server application consists of two major classes: CGPSDevice and CServSock. The first class was explained above, the second one is similar to CRecvSock, but simply listens on a specified port and adds a timestamp to the packets which are received and sends a response back to the client. Note that CRecvSock does not calculate or compute any parameters.

### 4.1.3 Our Experiments

In a previous thesis regarding delay measurements [49], we found no details on measuring the accuracy and stability of the PC clock after synchronization, therefore we decided to perform several experiments in order to know to what extent we can trust the synchronization for use in our later measurements; these experiments are based upon modifying our application to learn the real offset between the two computers' clocks and to synchronizing them. The most successful experiments include the following:

1. *Synchronize each computer's clock with the GPS timing every 5 seconds, then measure the clock offset between them*: Synchronizing the clock with a GPS receiver is an actually difficult to do with the devices which were used (see section 1.3.3). As it turns out these devices were not appropriate for accurate experimental tests. We encountered excessive problems due to the number of times that there were unexpected disconnect events between the GPS devices and computers (specially between the Pocket PC based handheld computers and the Bluetooth attached GPS receiver). Also, the GPS connection with the satellite was unexpectedly lost a lot of the times. As a result the experiments were time consuming and required a lot of effort.

Figure 9 shows the result of these measurements. The offset between the clocks of these two parties is around 600ms, more precisely from a statistically point of view, the median difference is 612ms and the standard deviation is around 472ms over the measurements time of 40 minutes. The result shows that the clock offset between two parties - which should be near zero based upon the GPS synchronizing- is somehow constant around 600ms (with some spurts to 1600ms which seems to be outcome of some measurement error) but the high deviation (~600±100) which is distinguishable by the chart is considerable and seems to be related to PC clock instability not to measurement error.

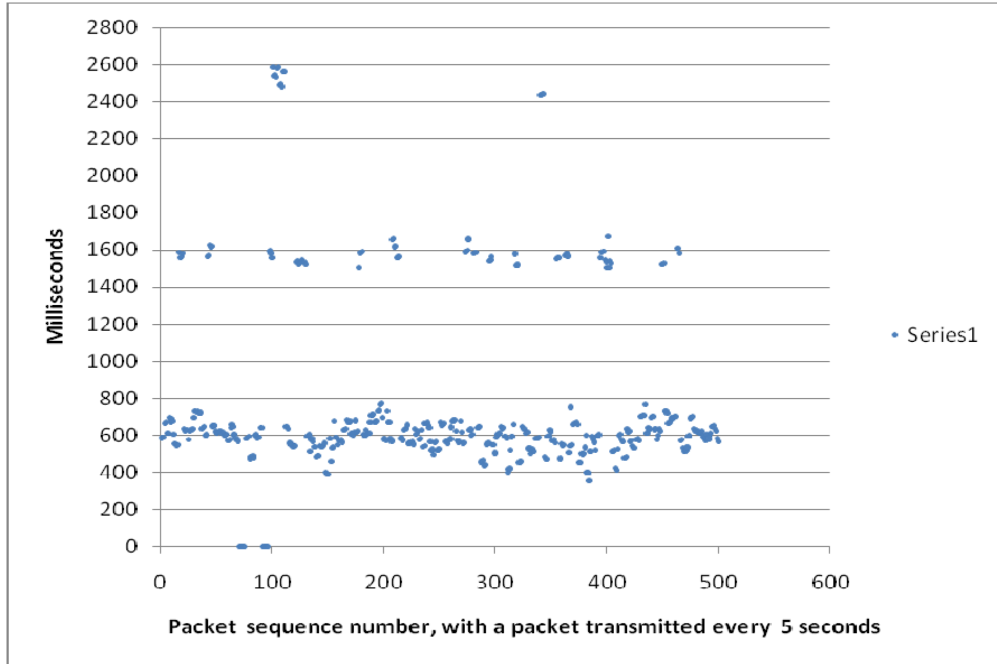


Figure 9: Clock offset between two end-parties during sync

2. *Measuring the clock offset between two computers to find out if this difference is stable or not (i.e., measuring the stability of a PC's clock)*: In this experiment we sent packets every 2 seconds to the server-side application. After several experiments we have found that the offset between the two clocks was growing. This is shown in figure 10. After transmitting about 1600 packets, i.e., 3200 seconds, the offset between the two computers' clock increased by about 200 milliseconds which means we can not assume that the PC clock is stable (actually the chart shows t1 which is the transmission delay between two parties based upon two different timestamps while the RTT is around 3ms in the performed experiment). The median is 1135ms and the standard deviation is around 82ms in these measurements. The result is very different from the results reported by A. Karapantelakis in [50]. This difference may come from the measurement's error relative to equipment or method of measurements.

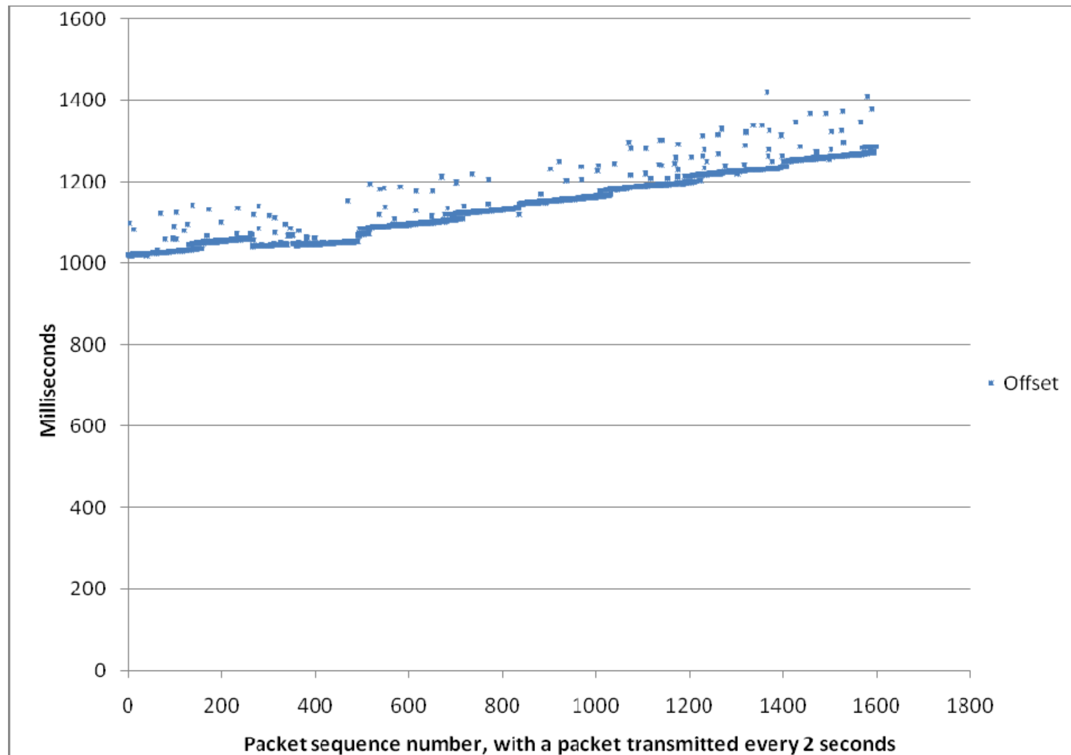


**Figure 10: Clock offset between two end-parties after sync**

3. *Finding the computer's clock offset with a NTP server to determine its accuracy stability*: After synchronizing the computer's clock with a NTP stratum 1 Server which is in Turkey (by clicking "Update Now" in the "Internet Time" tab in Windows XP), we performed this test using the following command line [49]:

*C:\>w32tm /monitor /computers:tr.pool.ntp.org*

The result was acceptable and the clock offset was near zero (~2ms) at first, but this offset increased gradually and exceeded 100ms within a short period of time (10 minutes). While we observed the traffic with applications such as Ethereal (now known as Wireshark[******]) but we observed only a slow drift, thus we did not performed experiments, which might have been performed, such as synchronizing two parties with NTP and then measuring the clock offset between them by our application. We did not perform these additional experiments because it was already clear that we could not count on the computer's clock to remain stable in the computers that we were using. The clock stability is valid only if we re-synchronize frequently and the measurements do not last very long (less than 120 seconds).

[******] http://www.wireshark.org/, a packet sniffer application for network analyzing

## 4.2 VoIP over GPRS Experiments:

One of the main goals of this thesis is to evaluate Voice over IP performance over GPRS. This section describes experiments in an actual GPRS network (in this case the GPRS network operated by MTN-Irancell Company).

### 4.2.1 One-way measurement

As mentioned in the section 4.1, we have done experiments to understand if and how to synchronize the clock of the two end-parties running on two different computers. In order to provide synchronization, we modified one of the applications in order to run it on Microsoft's Windows Mobile 5 (see Appendix A3 for details of this ported application). However, after running this application, we learned that the timestamps of the Windows Mobile powered device do not have a valid millisecond component, thus the millisecond part of the time is always zero. In order to access the actual system time we needed to work with the SYSTEMTIME structure in computers running Microsoft's Windows, but we found that Microsoft's Windows Mobile does **not** support such a structure.

After a lot of web surfing, we found in Microsoft's MSDN (Microsoft Developer Network) that: "The wMilliseconds value in the SYSTEMTIME structure is always zero (0) when you use the **GetSystemTime** function on a x86 Windows CE PC-based (CEPC) reference platform" [51]. Although our Pocket PC is running Windows CE based operating system it is not on an x86 (it is "Texas Instruments' OMAP" based architecture), but it seems that the same statement seems to be true with this type of machine as well.

Therefore, we could not perform one-way measurement experiments using the earlier methods, hence we simply perform a round-trip delay measurement.

### 4.2.2 Round-trip measurement

One of the things which we wanted to learn was the current GPRS coding scheme being used during the experiments. Since we were working with a Microsoft's Windows-based device, we could only access the information which Windows Mobile 5 makes available. After a lot of web surfing, we were unable to find any function that would directly reveal the current GPRS coding scheme. Therefore, we did experiments without knowing the coding scheme used (but we know from the operator that it is either CS1 or CS2, since the MTN-Irancell network only uses these coding schemes).

A set of experiments were conducted in two modes: (1) sending RTP packets at 10ms intervals with 20 byte payloads to simulate a G.723 CODEC and (2) sending RTP packets at 20ms intervals with 160 byte payloads to simulate a G.711 CODEC. Additionally, since the GPRS

node is behind a NAT, it should initiate a process to open a port in order for the traffic to flow in both directions (see section 2.1.5).

In the application, one part of the application is located in the GPRS node which initiates the sending process; the other part of the application is running in a node on the (public) internet listening to a specific port. The part of the application in the GPRS attached node controls the process (see figure 11), but the data analysis and creating of a log file are done in the fixed machine (the other party in the experiment). In these experiments, after activating the GPRS (and acquiring a new IP address), we determine the private IP address of the GPRS node by using the Microsoft Network Analyzer for Windows Mobile software in order to put this IP address in the relevant field of our application (because the sending process must tell the operating system which network interface and IP address it should use when sending) , one packet to initiate communication with for initiating to the other machine using its global IP address and the port number of a open listening port on the fixed device attached to the internet. Now, the application running on the fixed device has the global IP address and the port number of the application running on the GPRS node and can start communication. Then, the fixed machine sends the specified number of packets at a specified interval (that the GPRS node has indicated) to the GPRS node. The GPRS node simply sends a copy of the packet back to the IP address and port which had sent it (i.e., it acts as a echo service). After sending the specified number of packet from the fixed machine, the GPRS node waits to receive the last packet, but if it is lost, then after 10s (timeout) it terminates automatically and informs the other party about the timeout in order that the application can close its log file (see Appendix A3 for the source code for the client and server parts of this application).

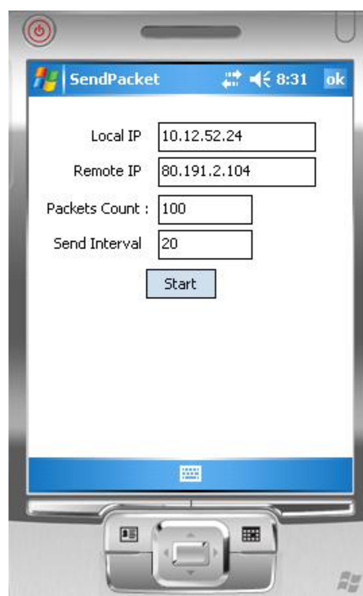


Figure 11: GUI of the application located in the GPRS node

The application inserts timestamp in each packet when sending and receiving -- in order to be able to calculate the round-trip time. Additionally, it saves the relevant timestamps for each sequence number in a log file in order to subsequently calculate the round-trip time for each of them. In our experiments, we sent 1000 packets in each test session because when we tried to send large amounts of data (i.e., over 1000 packets) the packet loss rate rapidly increased (to over 70% lost packets) -- this prevented us from acquiring a significant number of samples for analysis. Therefore, we chose 1000 packets as the number of samples in each session for our experiments; this enabled us to performance at least some delay measurements. We repeated this experiment ten times at different hours of the day in order to measure the average performance of the data channel and to calculate the average round-trip delay. These results are shown in figure 12 and 13.

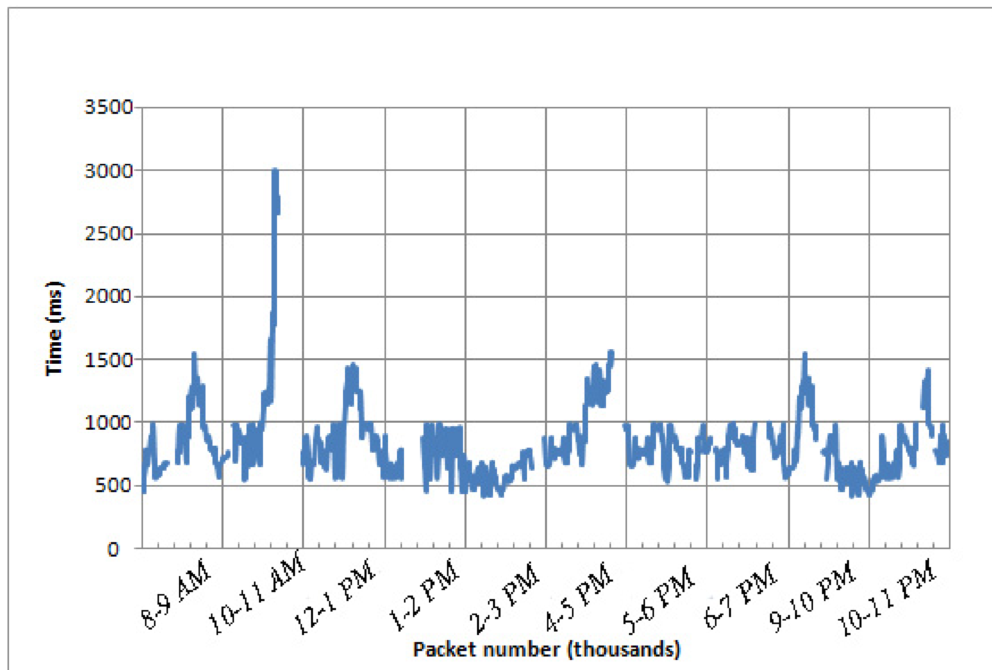


Figure 12: Round-trip delay of G.711voice CODEC over GPRS

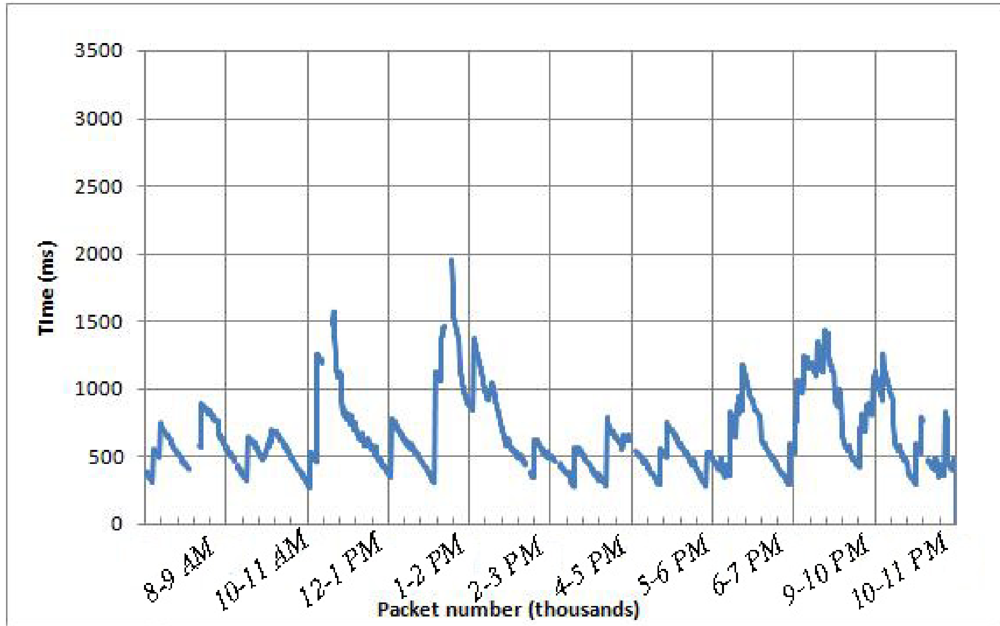


**Figure 13: Round-trip delay of G.723 voice CODEC over GPRS**

As is obvious in the figures, G.723 generally has a lower delay than G.711 CODEC. For G.723 the median round-trip delay is 569ms and the standard deviation is 281ms (square root of variance) across all of the experiments. For G.711 the median is 781ms and the standard deviation is 289ms.

Additionally, we did some experiments using Ericsson's TEMS software to find out more about the characteristics of the link during these experiments. In order to use TEMS, we attached a SonyEricsson phone to a computer using a USB interface. We changed the GPRS part of the application to run it under Microsoft's Windows (see Appendix A4) on this computer (a Dell D830 Latitude model laptop computer running Microsoft's Windows XP operating system). In figures 13 and 14 you can see the output of the TEMS Investigation software. Figures 13 and 14 indicate that almost all of the time the uplink utilized CS2 coding, while the downlink used CS2 ~90% of the time and CS1 ~10% of the time.

In the case of G.711, the packet loss rate is greater than for G.723. The reason can be the difference in packet sending intervals which is 20ms for G.711 and 30ms for G.723. When we increased the intervals between packets e.g., up to 50ms or 60ms, then the packet loss rate decreased very quickly and we found out that the interval time between sending packets is a very important factor for packet loss rate.

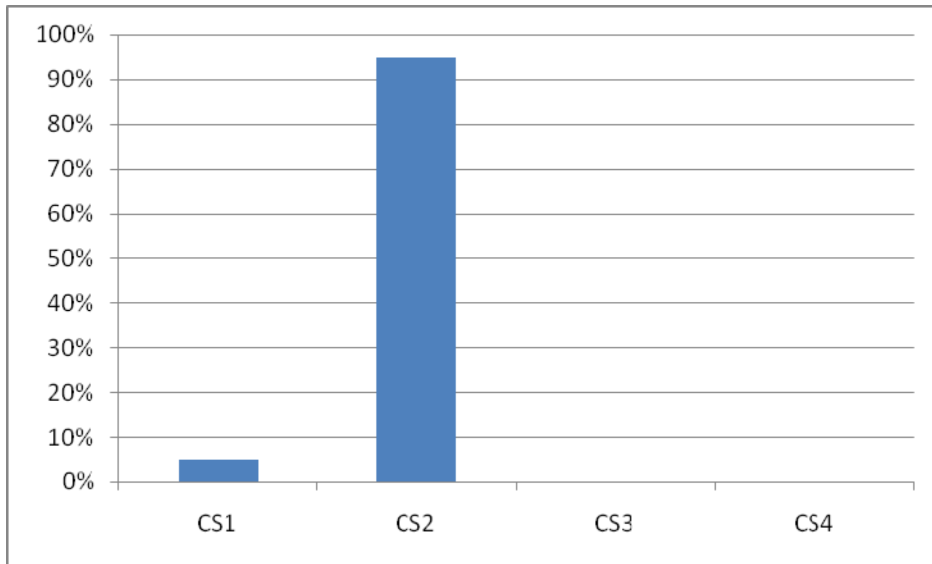

**Figure 14: Upload Coding Scheme used as a fraction of the measurement time**

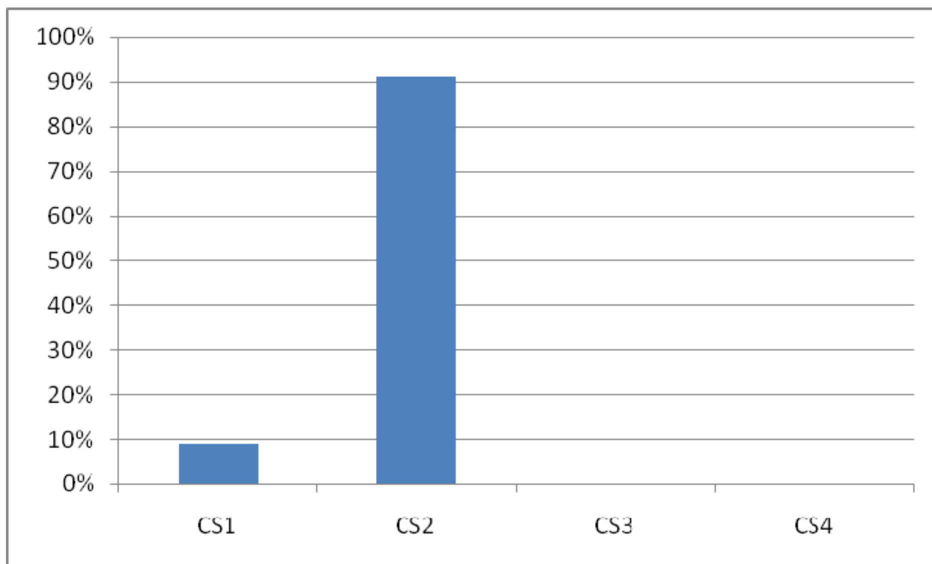

**Figure 15: Download Coding Scheme used as a fraction of the measurement time**

The median for the G.723 is 569ms for round-trip delay i.e., 285ms for one-way which sounds good for interactive voice calls but not good enough. Actually, since the standard deviation is high (289ms) and according to the mean (654ms), we can **not** assert that the one-way delay is around 285ms on average. About G.711, the median is 781ms which is high and for one-way is will be around 390ms which is not appropriate for VoIP. Again, since the standard deviation is 281ms while the mean is 825ms, we can **not** assert that this is an accurate measurement of the amount of delay. As is clearly shown in the figures, we can see that the delay varies as a function of the time of day. We see that during the working hours the delay is higher; one might assume

that this is because of the heavy GMS traffic during these hours, and during the night the delay decreases because the traffic load is lower. Since the TEMS results show that almost all of the traffic has utilized the CS2 coding scheme, we can say that measured delays are strongly linked to the CS2 coding scheme. Potentially, if the operator were to offer other coding schemes, then perhaps by using CS3 or CS4, one might attain better results in terms of one-way delay, if these delays were comparable to the lowest measured delay – then GPRS might be suitable as a carrier for VoIP traffic.

# 5. Windows Mobile and Minisip

## 5.1. Windows Mobile

Microsoft's Windows Mobile powered devices are of three types: Pocket PC, Pocket PC Phone Edition, and Smartphone. A Microsoft's Windows Mobile powered device is a handheld device executing a version of the Windows Mobile platform. This software enables users to make phone calls, check e-mails, schedule daily tasks, manage contacts, browse the Internet, send and receive text messages, read and compose Microsoft Word Mobile files, make Microsoft Excel Mobile charts, and view Microsoft PowerPoint presentations [52]. There are number of different configurations of Pocket PCs, many come with Wi-Fi (IEEE 802.11 WLAN) enabling the device to connect to the Internet when being in a WLAN hotspot. Users can compose e-mail messages and send them by synchronizing with a desktop computer or wirelessly when connected via a WLAN hotspot [52].

Windows Mobile 5.0 is one of the latest versions of the Windows Mobile platform (The latest version is Windows Mobile 6.0). Previous versions (in order of more recent to earliest) include: Windows Mobile 2003 Second Edition, Windows Mobile 2003, Pocket PC 2002, Smartphone 2002, and Pocket PC 2000. Currently, the vast majority of the Windows Mobile installed base is comprised of Windows Mobile 2003 Second Edition and Windows Mobile 5.0 powered devices [52].

Microsoft's Visual Studio 2005 is the recommended application development environment for Windows Mobile application development. When considering the development of an application's for the Windows Mobile platform, it is important to consider the application requirements and target devices in order to choose whether to use managed, unmanaged (native) or server-side code [52]. These three different approaches to application are described below:

**Managed code** is used with the .NET Compact Framework for user interface-centric applications that require rapid application development. You should also use managed code if easy access to Web services or data held in a SQL Server or SQL Server Mobile/Everywhere Edition database is required. Using the .NET Compact Framework, you can use Visual Basic .NET or C# to access a subset of the.NET Framework libraries, as well as the managed APIs [52]. In other words, Managed Code is what Visual Basic .NET and C# compilers create. It compiles to Intermediate Language (IL), not to machine code that could run directly on the computer. The IL is kept in a file called an assembly, along with metadata that describes the classes, methods, and attributes (such as security requirements) of the code that has been created[††††††].

[††††††] For further reading go to http://www.developer.com/net/cplus/article.php/2197621

**Native code** is used for high performance, if direct hardware access is needed, or if you require the smallest footprint. Native code is compiled directly to machine code - the machine where you compiled it. For native development, Visual C++ can be used to access Windows Mobile's native APIs (Application Programming Interface), as well as the Win32[‡‡‡‡‡‡] and MFC[§§§§§§] (Microsoft Foundation Classes) frameworks [52].

**Server-side code** is used for a wide variety of devices with a single code base and where there is guaranteed data bandwidth and connectivity to the device. With ASP.NET[*******], you can easily build browser-based Web applications that support multiple Web browsers, mark-up languages, and screen sizes/resolutions [52].

Whether you are developing using native, managed, or server-side code, Visual Studio 2005 can be used for development of applications for Windows Mobile 2003, Windows Mobile 2003 Second Edition, and Windows Mobile 5.0. In our case we used native code (Visual C++) and used Microsoft's Visual Studio 2005 as the development environment.

Visual Studio 2005 includes SDKs (Software Development Kit) and emulators for Windows Mobile 2003 and Windows Mobile 2003 Second Edition devices. To target Windows Mobile 5.0 devices, you need to download and install the Windows Mobile 5.0 SDK for Pocket PC and the Windows Mobile 5.0 SDK for Smartphone, which also include the relevant emulators [52].

## 5.2. Minisip

### 5.2.1. Porting Minisip to Windows Mobile

Porting Minisip [53] to Windows Mobile 5 was done in the Visual Studio environment using the device emulator. I ran and debugged the Text-UI (Text user-interface) version of Minisip on the Pocket PC 2003 emulator. This version is in **ppcgui** folder under the *branch* in the minisip SVN repository system. After solving some problems with compiling, we started working with the previously designed GUI (Graphical user-interface). One alternative was to make a wrapper for the .Net framework GUI, so that it could be connected to the pure C/C++ parts of the application. However, this did not seem to be a good solution, since the code of the previously designed GUI was not clear enough and contact with the author was not possible. Therefore, we created a new GUI for Windows Mobile version with MFC. Unfortunately, this did not work and we thought it

---

‡‡‡‡‡‡ Win32 is the 32-bit API for modern versions of Microsoft's Windows. The API consists of functions implemented, as with Win16, in system DLLs.

§§§§§§ MFC is a library that wraps portions of the Windows API in C++ classes, including functionality that enables them to use a default application framework.

******* ASP.NET is a web application framework developed by Microsoft that programmers can use to build dynamic web sites, web applications and web services. It is part of Microsoft's .NET platform and is the successor to Microsoft's Active Server Pages (ASP) technology.

is because of the use of MFC and some incompatibility with the main core. Next we tried to create a GUI using pure C/C++ code. The result was satisfactory.

Actually, since the Windows Mobile emulator is so slow while debugging such a heavy application (Windows Mobile emulator is installed as an add-on to Visual Studio), the Pocket PC 2003 emulator which is integrated in the Visual Studio 2005 is used with cross compilation using Visual Studio 2005—all running on a Dell D830 Latitude model laptop. After completing the task of porting, I cross-compiled and ran the ported code on the Windows Mobile emulator for a final test. Finally, I tested it on the actual Windows Mobile machine (Pocket PC i-mate k-jam).

In the Visual Studio you should set the switches and special settings for each of emulators in the main Solution (in Visual Studio parlance, a solution is a set of code files and other resources that are used to build an application). After porting to the Windows Mobile Emulator, the whole application compiled and built with no errors, but sometimes after the minisip starts running it crashes (with an "Access Violation" error) due to some problems in the main core of the application (the version of the application that I ported, had some problems in its main core).

The structure of the newly designed GUI is available from the Minisip project's web site (minisip.org) in the main solution called "Win32 GUI". Its structure is completely Objet-Oriented and uses a callback method to communicate with the Minisip core. This GUI has a very simple structure which can hopefully be easily extended and improved by other developers in the future.

In the main minisip "wmain" (minisip.cxx), I have added the definition of our GUI (called WIN32_UI) to instantiate it. Then in WIN32_UI, I have introduced the class CWinGui. In our added folder called "WIN 32GUI" (under the "minisip project" in the "minisip CE" solution in Visual Studio) we have the file called WinGui.cxx. In WinGui header file (WinGui.h) we have defined class CWinGui. It is inherited from class "CGui" which had been defined previously in minisip core file by previous developers.

The main file of our designed GUI is WinGui.cpp. All of the functions and user interface details have been defined in this file through a "Dialogue" (MainDlg). Finally, note that as said before every function in this GUI is maintained through a callback/handlecommand method (a technique that allows you to communicate with the minisip core without modifying or knowing how it works).

### 5.2.2. Visual Studio 2005 Settings for Debugging Minisip

The settings in the Visual Studio 2005 environment for debugging the Minisip project are important for further development because finding out how to correctly set the properties of the Solution is a very time consuming requiring extensive try and error.

When opening the Minisip Solution (after dowthe PPCGUI branch from the downloadable SVN repository from the minisip web site) in the Visual Studio, it consists of 6 projects [53]: 5 library projects (.dll) (Libmconsole, Libmutil, Libmikey, Libmsip, and Libmnetutil) and 1 executive project (.exe) called minisip. The properties of each of these projects should be set as follows (other settings are saved with the project, hence there is no need to configure other fields):

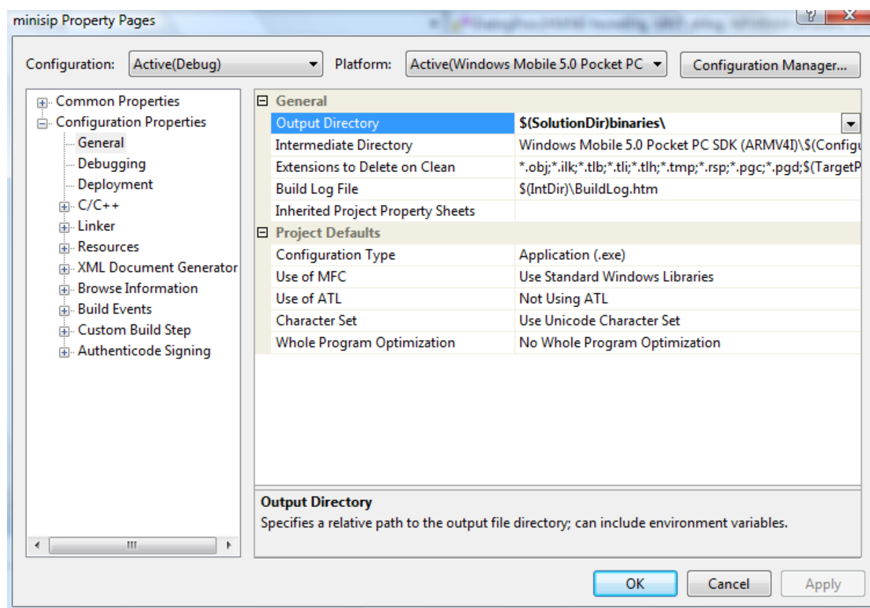


Figure 16: Minisip Visual Studio settings/ Property tab/General

As shown in the figure 16, the Character Set in the Project Defaults tab should be set as "Unicode"; this is necessary for the application to work in the Windows Mobile operating system. Also, the Output Directory and Configuration Type properties must be set appropriately (the configuration type for the other related projects should be Dynamic Library ".dll" so that they will be built as libraries which can be used by the main executable program).

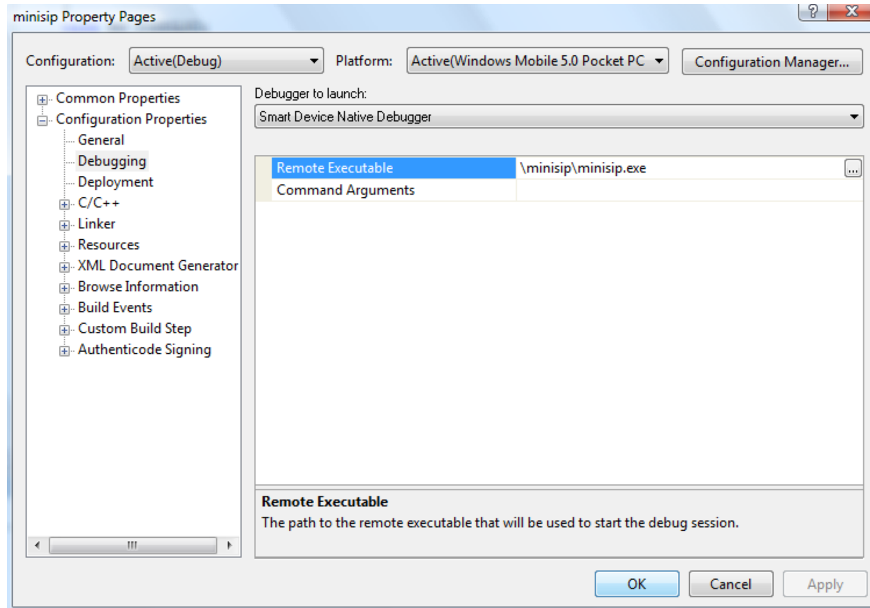


**Figure 17: Minisip Visual Studio settings/ Property tab/Debugging**

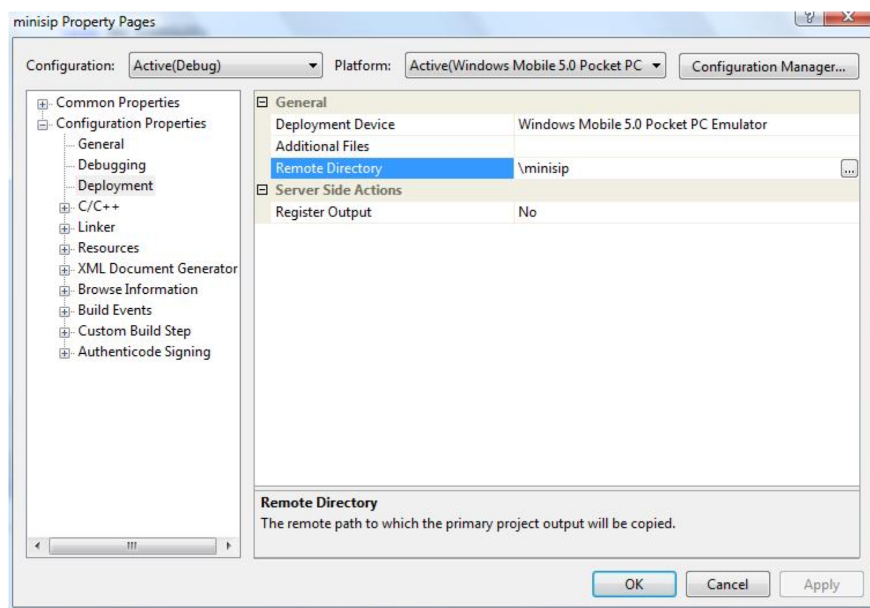


**Figure 18: Minisip Visual Studio settings/ Property tab/Deployment**

In the Debugging and Deployment tabs, ensure that the highlighted fields are the same as shown (i.e., the Remote Executable field should be set to “minisip.exe” along with its correct path). This indicates the directory and executable file that is to be created (see figures 17 and 18).

In the C/C++ tab, the most important configuration setting is defining the Preprocessor Definitions (see figure 19). Note that some new definitions are added and should be written exactly as follows: _W32_WCE, _DEBUG, UNDER_CE, $(PLATFORMDEFINES), WINCE,

DEBUG, _CONSOLE, $(ARCHFAM), WIN32_UI, _ARM_, ARM (these are found by trial and error since they are dependent on the core of the minisip application).

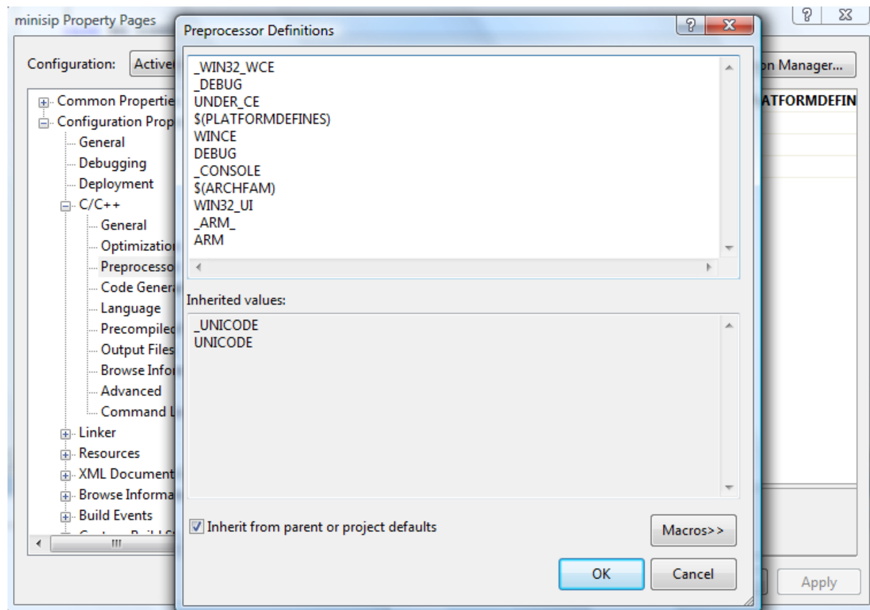


Figure 19: Minisip Visual Studio settings/ C++/preprocessor

The "runtime type info" field, in C/C++ language tab should be set as "Yes", note that for compatibility with Pocket PC 2003 you should add the library called "ccrtrtti.lib" to the path (\Program Files\Windows CE Tools\wce500\Windows Mobile 5.0 Pocket PC SDK\Lib) according to your installation path. In the case of Windows Pocket PC 2003, this file is included in the ARMV4 architecture SDK path, but in Windows Mobile 5.0 it is not included by default as this functionality exists as a built-in property; hence for to provide compatibility with both Windows Mobile versions (CE 2003 and 5.0) you should add "ccrtrtti.lib" in the libraries of Windows Mobile SDK (in order to create this library, you should run a program called "ritti.exe" available from Microsoft's website[†††††††]).

In the Linker tab, the Output file is set to ".exe" by default but for Input tab you should add some libraries in order to build the project (see figure 20) which are: wcecompatex.lib, libmutil.lib, libmnetutil.lib, libmikey.lib, libeay32.lib, ssleay32.lib, ws2.lib, ccrtrtti.lib, libmconsole.lib, portlib.lib, libmsip.lib.

[†††††††] For downloading the program go to http://support.microsoft.com/kb/830482

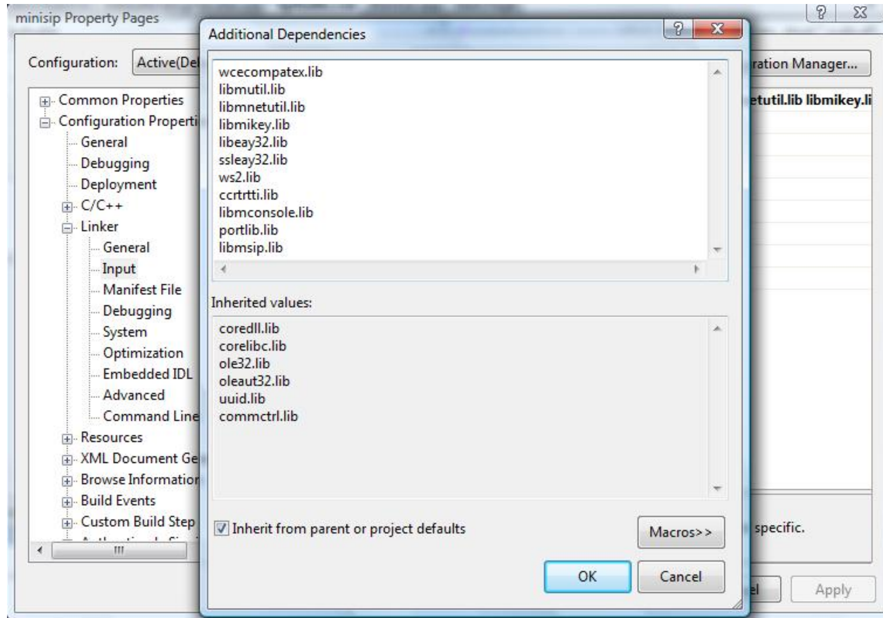

Figure 20: Minisip Visual Studio settings/ Linker tab/ Additional Dependencies

In the Advanced tab of the Linker for the target machine, you should choose Machine THUMB and not ARM, otherwise it does not work (this strange setting seems to be caused by a bug for Windows Mobile in Visual Studio 2005) (see figure 26).

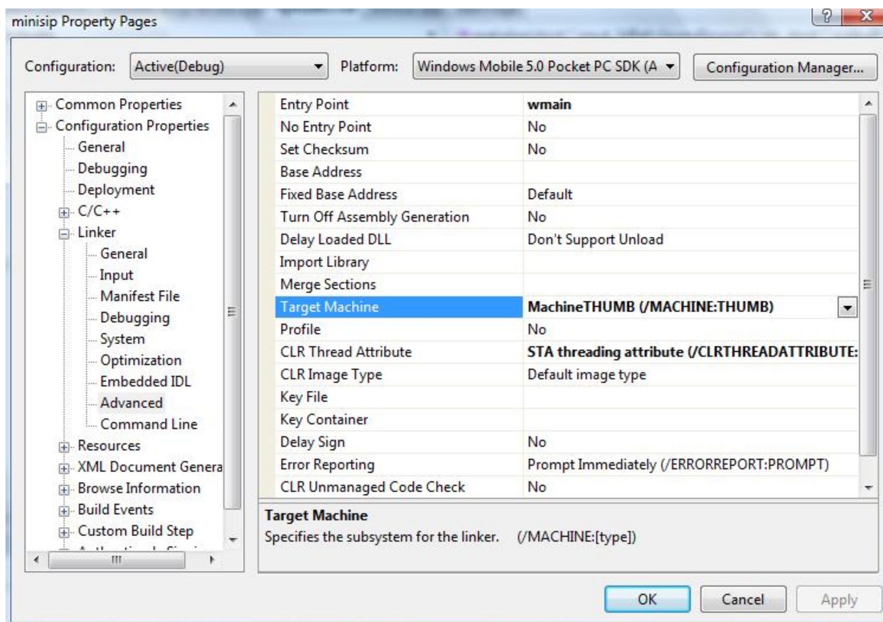

Figure 21: Minisip Visual Studio settings/ Linker tab/Advanced

Another important issue in this section is setting the dependencies of the “Projects” of the Minisip “Solution”. In the properties tab of the Solution, the Startup project should be set as

Minisip and in the Project Dependencies, the dependency of each of the projects should be set as follows:

libmconsole: depends on no other projects;
libmikey: depends on libmconsole, libmutil, and libmnetutil;
libmnetutil: depends on libmconsole and libmutil;
libmutil: depends on libmconsole;
libmsip: depends on all other projects except minisip;
minisip: depends on all of the projects;

Also in the Project Dependency of the Solution, note that in the Build Order you should set the order exactly as follows: libmconsole, libmutil, libmnetutil, libmikey, libmsip, and minisip. Finally, the last step is setting the tools\options in the Projects & Solutions; to do this - go to the VC++Directories and for platform Windows Mobile 5.0 in the Include Files and Library Files you should add the files that are shown in figures 22 and 23. The exact things that should be existed there is as follows:

Include files:
$(VCInstallDir)ce\atlmfc\include
$(VInstallDir)SmartDevices\SDK\SQL Server\Mobile\v.30
$(SolutionDir)ex_libs\dojstream-20041222\stdinc
$(SolutionDir)ex_libs\openssl\openssl\inc32
$(SolutionDir)ex_libs\PortSDK\Include
$(SolutionDir)ex_libs\PortSDK\Include\st
$(SolutionDir)ex_libs\wcecompat\include
$(SolutionDir)ex_libs\dojstream-20041222
$(SolutionDir)Minisip-gui
$(SolutionDir)source\libminisip\minisip\minisip
$(SolutionDir)source\libminisip\minisip

Library files:
…\Windows CE Tools\wce500\Windows Mobile 5.0 Pocket PC SDK\lib\ARMV4I
$(VCInstallDir)ce\altmfc\lib\ARMV4I
$(VCInstallDir)ce\lib\ARMV4I
$(SolutionDir)ex_libs\openssl\openssl\out32dll_ARMV4_pda_console
$(SolutionDir)ex_libs\wcecompat\lib
$(SolutionDir)binaries
…\Minisip SVN\ppcgui_textui\ex_libs\PortSDK\lib\ARM

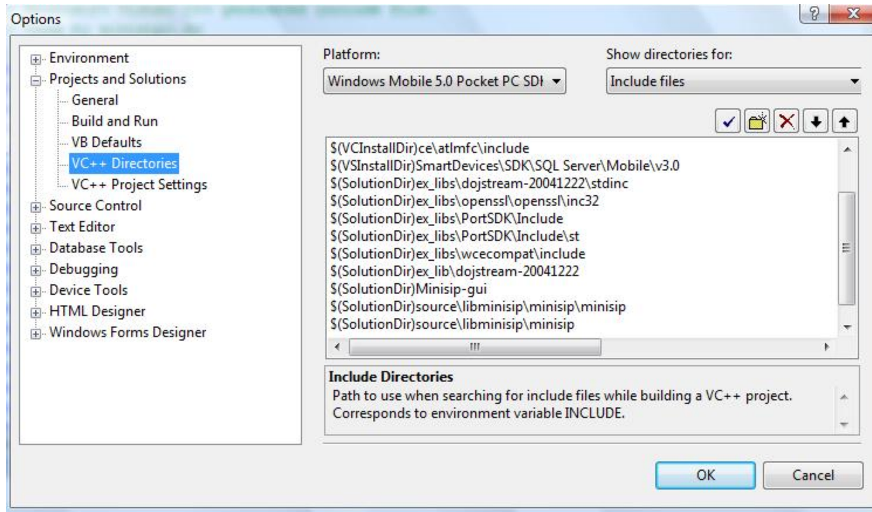


Figure 22: Minisip Visual Studio settings/ options/ VC++ Include directories

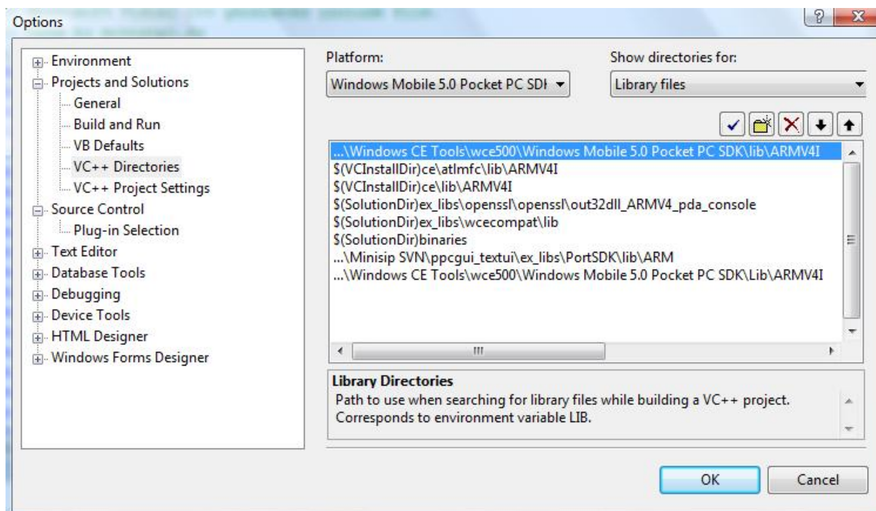


Figure 23: Minisip Visual Studio settings/ options/ VC++ Library directories

Now, everything is ready for compiling and building the solution. After building it, you can start to run or develop the application with no errors. In case of developing other versions of minisip, the aforementioned configuration of Visual Studio can be applicable and useful to consider.

### 5.2.3. Windows mobile Emulator Settings

Since the emulator for Windows Mobile is not integrated in Visual Studio 2005, you should download and install the latest version of it from Microsoft website. After installing this Emulator, you should install Pocketconsole (since by default Windows Mobile does not have any console), then create a share folder in the virtual disk storage of your emulator. Now you can

copy the needed .dll files, executable, and configuration files into this folder; then copy them from this location to the /windows path in the emulator.

The necessary .dll files include five basic .dll files (related to essential libraries; these are: libmikey.dll, libmconsole.dll, libmnetutil.dll, libmutil.dll, libmsip.dll) and two other files: ssleay32.dll and libeay32.dll which are relevant to "SSL" directory of the Minisip core and are available from \ppcgui_textui\ex_libs\openssl\openssl\out32dll_ARMV4_pda_console; and another .dll file named "portd.dll" available from \ppcgui_textui\ex_libs\PortSDK\bin\ARM.

Also, you should place the minisip.conf and minisip.addr files in the /root directory of the emulator; they can be copied into the disk storage folder relevant to the emulator for convenience (actually if you do not copy these files to the folder and later want to directly transfer them to the emulator, you will encounter a problem because to copy files to the emulator you should use the "ActiveSync" program which causes the IP address of the emulator's network card to be lost, hence minisip will not work and you must reset the emulator to get a new IP address).

# 6. Conclusions and Future Work

In this master's thesis, we evaluated the possibility of Voice over IP over a GPRS link and also ported an open-source software package for Voice over IP, called Minisip, to Microsoft's Windows Mobile operating system on a Pocket PC. We investigated the VoIP and GPRS parameters by looking at several recent papers prior to implementation.

As we wanted to measure one-way delays in our experiments, we developed some applications to synchronizing the devices with a GPS receiver's sense of time. We performed some experiments to assure ourselves that the two end parties of the experiment are synchronized. However, we found out that Windows Mobile does not provide us with millisecond time resolution, therefore measuring one-way delay was impossible. Additionally, synchronizing the parties does not make sense and is not necessary as we could only measure the round-trip delay in our later experiments.

We developed an application to send and receive RTP packets over a GPRS link running both on Microsoft's Windows Mobile and Windows XP (or Vista) i.e., we created two versions of the application for either running directly on a Windows Mobile powered device, or on a Windows XP (or Vista) machine connected to a cellular phone (such as a SonyEricsson TEMS). We performed some experiments to measure the delay of voice packets over a GPRS link during different hours of a day and we analyzed these results.

Additionally, we ported an open-source Voice over IP software package to the Windows Mobile platform. The version that we worked on was an olded version and we did not take care to make it easy to integrate the changes from the main trunk into this version (or vice versa). A procedure of porting more recent versions of Minisip has been defined. Since this thesis was done abroad, closed personal contact with the original authors of Minisip was not possible. The limited success of this part of the thesis project was due to this limited communication and because as I was new to Microsoft's Visual Studio environment and had limited experience porting such application.

The code which was developed as part of this thesis is in the Appendix, thus future work on each part of the thesis is feasible. Using the time synchronization applications, someone else can perform experiments using GPS receivers in order to get very accurate timing results. Since such GPS applications can not be easily found on the Internet, this object-oriented code can help future researchers to develop their own application according to their needs.

The application to send and receive RTP packets is a good starting point for more experiments utilizing different coding schemes of GPRS, or EDGE link. Since the network and GPRS operator offered little collaboration with us during our experiments, further experiments (as described in the scenario in the section 1.3.4) could be done. Since most of the papers relevant to

this area utilized only simulation, there is a good possibility to build upon this application, for those who want to have actual experimental results.

The Minisip section of the thesis report (chapter 5.2) includes many figures showing how you should configure Visual Studio environment in order to compile the application. With the aid of this section and the provided code, the Minisip project could be updated to port the latest versions of Minisip to Windows Moile 5.0 (or 6.0). A desirable future functionality is to automatically generate an executable file for Windows Mobile each time a new version appears in the trunk. This should be done as part of a thesis in the future, in order to always have an updated version of Minisip which can run on Windows Mobile. Since Windows Mobile is the default operating system run on the Pocket PC, this improvement is of considerable practical importance.

## References

[1] L. Dang, C. Jenning, and D. Kelly, "Practical VoIP", Orielly & Associates, USA, 2002

[2] G.Q. Maguire Jr., "Practical voice over IP" Lecture notes, Department of Communication Systems, Royal Institute of Technology (KTH), Stockholm, 2006

[3] Israel M. Caballero, "Secure Mobile voice over IP", masters thesis, June 2003, Stockholm, available at ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/030626-Israel_Abad_Caballero-final-report.pdf

[4] M. Handley, V. Jacobson, "SDP: Session Description Protocol", RFC 2327, www.ietf.org, April 1998

[5] Wikipedia RTP, http://en.wikipedia.org/wiki/RTP available at 2008-03-04

[6] Wikipedia MIME, http://en.wikipedia.org/wiki/MIME available at 2008-03-04

[7] R. Fredrick et al., "RTP: A Transport Protocol for Real-Time Application", RFC 3550, www.ietf.org, July 2003

[8] Wikipedia NTP, http://en.wikipedia.org/wiki/NTP available at 2008-03-04

[9] David L. Mills, "Network Time protocol (version 3)", RFC 1305, www.ietf.org, March 1992

[10] White Papers of Newport Networks Ltd., "NAT Traversal for Multimedia over IP", 2006 available at http://www.newport-networks.com/cust-docs/33-NAT-Traversal.pdf

[11] J. Hoffman, "GPRS Demystified", McGraw-Hill Telecom, USA, 2003

[12] G.Q. Maguire Jr., "Mobile & Wireless Architectires" Lecture notes, Department of Communication Systems, Royal Institute of Technology (KTH), Stockholm, 2006

[13] GSM GPRS, http://www.cellular.co.za/gprs.htm available at 2008-01-19

[14] ITU-T Recommendation G.107, "The E-model, a computational model for use in transmission planning", Dec 1998

[15] R.G. Cole and, J.H. Rosenbluth, "Voice over IP performance monitoring" AT&T Laboratories, Middletown, NJ., ACM SIGCOMM Computer Communication Review, Volume 31 , Issue 2 (April 2001), pages: 9 – 24, 2001

[16] L. Carvalho, et al., "An E-Model Implementation for Speech Quality Evaluation in VoIP Systems", Proceedings 10th IEEE Symposium on Volume Computers and Communications, 2005. ISCC 2005., Issue, 27-30 June 2005 Page(s): 933 - 938

[17] John H. Mock, et al., "A voice over IP solution for mobile radio interoperability", In Proceeding of the 2002 IEEE 56th Vehicular Technology Conference (VTC 2002), Volume 3, Issue , 2002 Page(s): 1338 - 1341 vol.3

[18] White Papers of Newport Networks Ltd., "NAT Traversal for Multimedia over IP", 2005 available at http://www.newport-networks.com/cust-docs/52-VoIP-Bandwidth.pdf

[19] S.Zeadally, et al., "Voice over IP in Intranet and Internet Environment", Communications, IEE Proceedings,25 June 2004, Volume: 151, Issue 3, page(s): 263- 269, ISSN: 1350-2425

[20] Lars-Ake Larzon, et al., "Efficient Transport of Voice over IP over Cellular links", Global Telecommunications Conference, (GLOBECOM'00), IEEE Volume 3, Issue, 2000 Page(s):1669 - 1676 vol.3

[21] "Telephone Transmission quality objective measuring apparatus", Artificial conversational speech, ITU-T Recommendation P.59, Mar. 1993 available at http://www.itu.int/rec/T-REC-P.59-199303-I/en at 2008-01-19

[22] P. Progtong, et al., "Analysis of speech data rate in voice over IP conversation", 2004 IEEE Region 10 Conference Thailand, TENCON, Nov. 2004, Volume: A, page(s): 143- 146 Vol. 1, ISBN: 0-7803-8560-8

[23] C. Moorhead, et al., "Resource Allocation Schemes to provide QoS for VoIP over GPRS", IEEE First International Conference on 3G Mobile Communication Technologies, (Conf. Publ. No. 471) , year: 2000, Page(s):138 – 142

[24] Xiaoyan Fang et al., "Analyzing Packet Delay across a GSM/GPRS Network", University of California Davis, Technical report. CSE-2003-3 Computer Science Department, January 2003, available at http://whitepapers.zdnet.co.uk/0,1000000651,260090597p,00.htm at 2008-01-19

[25] M. A. Marsan, et al., "Packet Delay Analysis in GPRS Systems", INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE, Volume 2, Issue , 30 March-3 April 2003 Page(s): 970 - 978 vol.2

[26] B. Rohani and M. Abaii, "QoS Augmentation in GPRS via Adaptive Half-RateSpeech Coding", The IEEE 9th Asia-Pacific Conference on Communication, (APCC 2003), Volume 1, Issue , 21-24 Sept. 2003 Page(s): 138 - 142 Vol.1, Malaysia

[27] K. Premkumar and A. Chockalingam, "Performance Analysis of RLC/MAC and LLC Layers in a GPRS Protocol Stack", IEEE Transactions on Vehicular Technology, Volume 53, Issue 5, Sept. 2004 Page(s): 1531 – 1546

[28] Digital cellular telecommunications system (phase 2+); GPRS; MS-SGSN logical link control (LLC) layer specification (GSM 04.64). ETSI, TC-SMG GPRS ad hoc, Sophia Antipolis, France. Available: http://www.etsi.org/

[29] "ARQ Protocol with packet-based reliability level setting", European Patent EP0996248 available at http://pmcg-consultancy.co.uk/courses/gprs_operations.htm at 2008-01-19

[30] Digital cellular telecommunications system (phase 2+); GPRS; MS-BSS radio link control/medium access control (RLC/MAC) protocol (GSM 04.64). ETSI, TC-SMG GPRS ad hoc, Sophia Antipolis, France. [Online]. Available: http://www.etsi.org/

[31] M. Mahdavi and R. Tafazolli, "Analysis of Integrated voice and data for GPRS", IEEE First International Conference on 3G Mobile Communication Technologies, 2000. (Conf. Publ. No. 471) Volume , Issue , 2000 Page(s):436 - 440

[32] Andreas Schieder, et al., "Enhanced Voice over IP Support in GPRS and EGPRS", Wireless Communications and Networking Conference, (WCNC, 2000) IEEE Volume 2, Issue, 2000 Page(s):803 - 808 vol.2

[33] D. Su, et al., "Investigating Factors Influencing QoS of Internet Phone" IEEE International Conference on Multimedia Computing and Systems, 1999. Volume 1, Issue , Jul 1999 Page(s):308 - 313 vol.1

[34] B. W. Wah and Dong Lin, "LSP-Based Multiple-Description Coding for Real-Time Low Bit-Rate Voice Over IP" IEEE Transactions on Multimedia, Volume 7, Issue 1, Feb. 2005 Page(s): 167 – 178

[35] Shengquan Wang, et al., "Design and Implementation of QoS-Provisioning System for Voice over IP", IEEE Transactions on Parallel and Distributed Systems Volume 17, Issue 3, March 2006 Page(s): 276 - 288

[36] “IEEE Standard for Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Medium Access Control (MAC) Enhancements for Quality of Service (QoS),” IEEE 802.11e/D13.0, July 2005.

[37] H. Zhai, et al., “Providing Statistical QoS Guarantee for Voice over IP in the IEEE 802.11 Wireless LANs”, Wireless Communications, IEEE. Volume 13, Issue 1, Feb. 2006 Page(s): 36 - 43

[38] J. Yu, and et al., “Enhancement of VoIP over IEEE 802.11 WLAN via Dual Queue Strategy”, IEEE International Conference on Communications, 2004, Volume 6, Issue , 20-24 June 2004 Page(s): 3706 - 3711 Vol.6

[39] M. Adda, et al, “Voice over IP on Wireless Networks”, IEEE Information and Communication Technologies, 2006. ICTTA '06, 24-28 April 2006 Volume: 2, page(s): 2742 – 2747

[40] A. Shepard, “Hybrid change Makes WLAN QoS Come to Life”, Apr. 2004, available at http://www.us.design-reuse.com/articles/7742/hybrid-change-makes-wlan-qos-come-to-life.html at 2008-01-19

[41] J. Parantainen and S. Hamiti, Nokia Research Center, “Delay analysis for IP speech over GPRS”, IEEE VTS 50th Vehicular Technology Conference, (VTC 1999) - Fall., Volume 2, Issue , 1999 Page(s):829 - 833 vol.2

[42] Qingguo Shen, “Performance of VoIP over GPRS”, International Conference on Advanced Information Networking and Applications, (IEEE AINA 2003), 17th Volume , Issue , 27-29 March 2003 Page(s): 611 – 614

[43] E. Hallmann and R. Helmchen, “Investigations on the Throughput in EDGE- and GPRS-Radio Networks”, Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd Volume 4, Issue , 2001 Page(s):2823 - 2827 vol.4

[44] O. Gurewitz and M. Sidi, “Estimating one-way delays from cyclic-path delay measurements,” INFOCOM 2001, Proceeding of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE Volume 2, 22-26 April 2001 Page(s):1038 - 1044 vol.2

[45] O. Gurewitz and I. Cidon, M. Sidi, “One-way delay estimation using network-wide measurements,” Information Theory, IEEE Transactions on Volume 52, Issue 6, June 2006 Page(s):2710 - 2724

[46] D. L. Mills, “Improved algorithms for synchronizing computer network clocks,” IEEE/ACM Transaction on Networking, vol. 3, no. 3, pp. 245–254, Jun. 1995.

[47] M. Tsuru and T. Takine, “Estimation of clock offset from one-way delay measurement on asymmetric paths,” IEEE 2002. Proceedings. 2002 Symposium on Applications and the Internet (SAINT) Workshops, 28 Jan.-1 Feb. 2002 Page(s):126 – 133

[48] D. Veitch, S. Babu, and A. Pasztor, “Robust synchronization of software clocks across the internet,” in Proc. Internet Measurement Conf.,Taormina, Italy, Oct. 2004, pp. 219–232.

[49] F. Mayer, “Adding NTP and RTCP to a SIP user agent”, Masters thesis, Department of Communication Systems, Royal Institute of Technology (KTH),, Stockholm, 2006, available at ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/060620-Franz-Mayer-with-cover.pdf

[50] A. Karapantelakis, “A Mobile SIP Client: From the user inrterface design to evaluation of synchronized playout from multiple SIP user agents”, Masters thesis, Department of Communication Systems, Royal Institute of Technology (KTH), 2007, Stockholm, available at http://web.it.kth.se/~maguire/DEGREE-PROJECT-REPORTS/070227-Athanasios_Karapantelakis-thesis_final-with-cover.pdf

[51] “GetsystemTime Milliseconds are zero on CEPC, available at http://support.microsoft.com/kb/260419 at 2008-01-19

[52] “Windows Mobile 5.0 Developer Resource Kit” downloadable from http://www.microsoft.com/downloads/details.aspx?familyid=3baa5b7d-04c1-4ec2-83dc-61b21ec5fe57&displaylang=en

[53] Minisip, www.minisip.org available at 2008-03-20

# Appendix:

## A1. Synchronizing the time with a GPS receiver (Server Part)

The excerpt of the relevant source code:

**File name: SyncServer.cpp**

```
/ SyncServer.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "SyncServer.h"
#include "ServSock.h"
#include "GPSDevice.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// The one and only application object

CWinApp theApp;

using namespace std;

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
	int nRetCode = 0;

	// initialize MFC and print and error on failure
	if(!AfxSocketInit())
		printf("Error initializing sockets.\n");

	if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
	{
		// TODO: change error code to suit your needs
		_tprintf(_T("Fatal Error: MFC initialization failed\n"));
		nRetCode = 1;
	}
	else
	{
	}
	printf("Initializing ...\n");
	printf("Enter GPS communication port : ");
	char *com_port = new char[4];
	char *local_ip = new char[15];
	char *remote_ip= new char[15];
	int udp_port;

	scanf("%s", com_port, 4);
	printf("Enter local IP address : ");
	scanf("%s", local_ip, 15);
```

```
	printf("Enter remote IP address : ");
	scanf("%s", remote_ip, 15);
	printf("Enter local/remote udp port : ");
	scanf("%d", &udp_port);
	//CGPSDevice gpsDevice(com_port, 4800);
	//gpsDevice.StartSync(5);
	CServSock *servSock = new CServSock(local_ip, remote_ip, udp_port);
	BOOL bRet;
	MSG msg;
	while( (bRet = GetMessage( &msg, NULL, 0, 0 )) != 0 )
	{
		//printf("Message %d\n", msg.message);
		if (bRet == -1 )
		{
			// handle the error and possibly exit
		}
		else
		{
			if(msg.message == WM_QUIT)
			{
				return 0;
			}
			TranslateMessage( &msg );
			DispatchMessage( &msg );
		}
	}

	return nRetCode;
}
```

**File name: ServSock.h**

```
#pragma once

CString GetDateTime();
double GetSecs(CString tm);
class CServSock : public CAsyncSocket
{
public:
	char *m_local_ip;
	char *m_remote_ip;
	int m_udp_port;
	CServSock(char *local_ip, char *remote_ip, int udp_port)
	{
		m_local_ip = new char[strlen(local_ip)];
		m_remote_ip = new char[strlen(remote_ip)];

		strcpy(m_local_ip, local_ip);
		strcpy(m_remote_ip, remote_ip);
		m_udp_port = udp_port;
		int bStat = Create(m_udp_port, SOCK_DGRAM, FD_READ, m_local_ip);
		if(!bStat)
		{
			printf("Error initiating Socket.\n\n");
			int err = GetLastError();
```

```
                return;
            }
            else
            {
                printf("Socket successfully initiated.\n\n");
            }
        }
        void OnReceive(int nErrorCode)
        {
            static int i=0;
            i++;
            char lpBuff[512];
            int nRead;
            DWORD nWritten = 0;
            nRead = Receive(lpBuff, sizeof(lpBuff));
            lpBuff[nRead] = 0;
            CString szTemp(lpBuff);
            CString now = GetDateTime();
            /////////////////////////
            LARGE_INTEGER tick;
            LARGE_INTEGER freq;

            QueryPerformanceFrequency(&freq);
            QueryPerformanceCounter(&tick);
            float t3 = (float)tick.QuadPart / freq.QuadPart;

            szTemp.AppendFormat(" %.4f", t3);
            //szTemp.AppendFormat(" %s", now);

            double t1 = GetSecs(szTemp.Mid(5, 12));
            double t2 = GetSecs(szTemp.Mid(18, 12));

            switch (nRead)
            {
            case 0:
                Close();
                break;
            case SOCKET_ERROR:
                if (GetLastError() != WSAEWOULDBLOCK)
                {
                    printf("Error in socket.\n");
                    int err = GetLastError();
                    Close();
                }
                break;
            default:
                printf("%s - Delay : %2.3f\n", szTemp, t2 - t1);
                SendTo(szTemp, szTemp.GetLength(), m_udp_port,
m_remote_ip);
            }
            CAsyncSocket::OnReceive(nErrorCode);
        }
        ~CServSock(void)
        {
            delete m_local_ip;
            delete m_remote_ip;
        }
```

```
};
double GetSecs(CString tm)
{
	int hr = atoi(tm.Mid(0, 2));
	int min = atoi(tm.Mid(3, 2));
	double sec = atof(tm.Mid(6, 6));
	double res = (hr * 3600) + (min * 60) + sec;
	return res;
}
CString GetDateTime()
{
	SYSTEMTIME st;
	CString timeStr;
	GetSystemTime(&st);
	timeStr.Format("%.2d:%.2d:%.2d.%.3d",
		st.wHour, st.wMinute, st.wSecond, st.wMilliseconds);
	return timeStr;
};
```

**File name: GPSDevice.h**

```
#define _LOGGING
#pragma once
CString strDate, strTime;
SYSTEMTIME *utcTime;
char port[4];
int baudRate;
HANDLE hCom;
UINT GPSThreadProc( LPVOID pParam );
bool InitComm(void);

class CGPSDevice
{
public:
	CGPSDevice(char *Port, int BaudRate)
	{
		strcpy(port, Port);
		baudRate = BaudRate;
	}
	int m_nInterval;
	void StartSync(int nInterval)//nInterval in seconds
	{
		m_nInterval = nInterval;
		if(InitComm())
			AfxBeginThread(GPSThreadProc, &m_nInterval);
	}
	void EndSync(void)
	{
		CloseHandle(hCom);
	}

public:
	~CGPSDevice(void)
	{
		CloseHandle(hCom);
```

```
        }
        SYSTEMTIME* getTime()
        {
                return utcTime;
        }
};
bool InitComm(void)
{
        OVERLAPPED o;
        BOOL fSuccess;
        printf("Openning %s at %d ...\n\n", port, baudRate);
        hCom = CreateFile(port, GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
        if(hCom == INVALID_HANDLE_VALUE)
        {
                printf("GPSDevice error : Error opening communication
port.\n\n");
                return false;
        }
        DCB dcb;
        GetCommState(hCom, &dcb);
        switch (baudRate)
        {
        case 4800:
                dcb.BaudRate = CBR_4800;
                break;
        case 9600:
                dcb.BaudRate = CBR_9600;
                break;
        }
        dcb.EvtChar = '$';
        SetCommState(hCom, &dcb);
        COMMTIMEOUTS to;
        GetCommTimeouts(hCom, &to);
        to.ReadIntervalTimeout = 0;
        SetCommTimeouts(hCom, &to);
        fSuccess = SetCommMask(hCom, EV_RXCHAR);// EV_RXCHAR EV_RXFLAG EV_DSR
EV_BREAK
        if(!fSuccess)
        {
                printf("GPSDevice error : Error setting port settings.\n");
                return false;
        }
        printf("Port ""%s"" opend at %d baud rate.\n\n", port, baudRate);
        ZeroMemory(&o, sizeof(o));
        o.hEvent = CreateEvent(NULL, false, false, NULL);
        PurgeComm(hCom, PURGE_RXABORT | PURGE_TXABORT);
        char *lpBuff = new char[1024];
        return true;
};
int SysTime2MilSec(SYSTEMTIME *tm)
{
        int res;
        res = tm->wHour * 3600000;
        res += tm->wMinute * 60000;
        res += tm->wSecond * 1000;
        res += tm->wMilliseconds;
        return res;
```

```
}

void AddMil2SysTime(SYSTEMTIME &tm, int offset)
{
	tm.wMilliseconds += offset;
	if(tm.wMilliseconds > 999)
	{
		tm.wSecond  += tm.wMilliseconds / 1000;
		tm.wMilliseconds = tm.wMilliseconds % 1000;
		if(tm.wSecond > 59)
		{
			tm.wMinute += tm.wSecond / 60;
			tm.wSecond = tm.wSecond % 60;
			if(tm.wMinute > 59)
			{
				tm.wHour += tm.wMinute / 60;
				tm.wMinute = tm.wMinute % 60;
			}
		}
	}
	return;

}

int MilDif(SYSTEMTIME *tm1, SYSTEMTIME *tm2)
{
	return ( SysTime2MilSec(tm1) - SysTime2MilSec(tm2) );
}
int Wait(int nInterval)
{
    HANDLE hTimer = NULL;
    LARGE_INTEGER liDueTime;

    liDueTime.QuadPart = - nInterval * 1000 * 10;

    // Create a waitable timer.
    hTimer = CreateWaitableTimer(NULL, TRUE, "WaitableTimer");
    if (!hTimer)
    {
        printf("CreateWaitableTimer failed (%d)\n", GetLastError());
        return 1;
    }

    //printf("Waiting for 10 seconds...\n");

    // Set a timer to wait for 10 seconds.
    if (!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, 0))
    {
        printf("SetWaitableTimer failed (%d)\n", GetLastError());
        return 2;
    }

    // Wait for the timer.

    if (WaitForSingleObject(hTimer, INFINITE) != WAIT_OBJECT_0)
        printf("WaitForSingleObject failed (%d)\n", GetLastError());
    else
```

```
			;//printf("Timer was signaled.\n");

}

UINT GPSThreadProc( LPVOID pParam )
{
	char lpBuff[1024];
	DWORD dwEvtMask;
	int index;
	int tick_count = 9999;
	CString date, time;
	SYSTEMTIME gpsTime;
	utcTime = new SYSTEMTIME();
	SYSTEMTIME *t1 = new SYSTEMTIME();
	SYSTEMTIME *t2 = new SYSTEMTIME();
	printf("System time is synchronizing every %d seconds ...\n\n",
*(int*)pParam);
	int res;
#ifdef _LOGGING
	OFSTRUCT of;
	HANDLE h = (HANDLE)OpenFile("c:\\nema.txt", &of, OF_CREATE);
#endif
	int time_tag1;
	int time_tag2;
	int offset;
	GetSystemTime(&gpsTime);
	int flag = 0;
	while(res = WaitCommEvent(hCom, &dwEvtMask, NULL))
	{
		DWORD bResult;
		DWORD bytesRead;
		ZeroMemory(lpBuff, sizeof(lpBuff));
		bytesRead = 0;
		bResult = ReadFile(hCom, lpBuff, sizeof(lpBuff), &bytesRead,
NULL);
		if(bytesRead == 1)
		{
			time_tag1 = GetTickCount();
			Wait(300);
			continue;
		}

		if(tick_count < *(int*)pParam)
		{
			tick_count++;
			continue;
		}
		else
		{
			tick_count = 0;
		}

#ifdef _LOGGING
		DWORD bytesWritten;
		WriteFile(h, lpBuff, bytesRead, &bytesWritten, NULL);
		CString ind;
		ind.Format("\n****\n");
```

```
            WriteFile(h, ind, ind.GetLength(), &bytesWritten, NULL);
#endif


            int err = GetLastError();
            CString buff(lpBuff);

            index = buff.Find("GPGGA", 0);
            if(index<0)
            {
                  printf("Error finding $GPGGA or $GPRMC statement ...\n");
                  continue;
            }

            CString gprmc = buff.Tokenize("\n", index);

            index = 6;
            int token = 0;
            CString tok;

            time = gprmc.Tokenize(",", index);

            int dotIndex = time.Find('.');
            int len = time.GetLength();
            CString t = time.Mid(0, dotIndex);
            CString mi = time.Mid(dotIndex + 1, len - dotIndex);

#ifdef _LOGGING
            //printf("Time : %s\n", time);
#endif
            gpsTime.wMilliseconds = atoi(mi.GetBuffer());

            if(t.GetLength() == 5)
            {
                  gpsTime.wHour = atoi(t.Mid(0, 1));
                  gpsTime.wMinute = atoi(t.Mid(1, 2));
                  gpsTime.wSecond = atoi(t.Mid(3, 2));
            }
            else
            {
                  gpsTime.wHour = atoi(t.Mid(0, 2));
                  gpsTime.wMinute = atoi(t.Mid(2, 2));
                  gpsTime.wSecond = atoi(t.Mid(4, 2));
            }
            printf("offset = %d\n", GetTickCount()-time_tag1);
            time_tag2 = GetTickCount();
            offset = time_tag2 - time_tag1;
            AddMil2SysTime(gpsTime, offset);
            SetSystemTime(&gpsTime);
            //SYSTEMTIME local;
            //GetSystemTime(&local);
            //int dif = MilDif(&local, &gpsTime);
            //printf("Dif : %d\n", dif);
            err = GetLastError();
      }  // while
#ifdef _LOGGING
      CloseHandle(h);
```

```
#endif
	int err;
	if(res == 0)
	{
		err = GetLastError();
	}
	delete utcTime;
	return 0;	// thread completed successfully
};
```

## A2. Synchronizing the time with a GPS receiver (Client Part)

The excerpt of the relevant source code:

**File name: SyncClient.cpp**

```
#include "stdafx.h"
#include "SyncClient.h"
#include "GPSDevice.h"
#include "SendSock.h"
#include "RecvSock.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

CWinApp theApp;

using namespace std;

int _tmain(int argc, TCHAR* argv[], TCHAR* envp[])
{
	int nRetCode = 0;
	if (!AfxWinInit(::GetModuleHandle(NULL), NULL, ::GetCommandLine(), 0))
	{
		_tprintf(_T("Fatal Error: MFC initialization failed\n"));
		nRetCode = 1;
	}
	else
	{
		AfxSocketInit();
	}
	printf_s("Initializing ...\n");
	//Initiating GPS usage
	char *com_port = new char[4];
	char *local_ip = new char[15];
	char *remote_ip = new char[15];
	int udp_port;

	printf_s("Enter GPS device communication port : ");
	scanf("%s", com_port);
```

```
	printf_s("Enter local IP address : ");
	scanf("%15s", local_ip);
	printf_s("Enter local/remote UDP port : ");
	scanf("%d", &udp_port);

	printf_s("Enter remote IP address : ");
	scanf("%15s", remote_ip);

	CGPSDevice gpsDevice(com_port, 4800);
	gpsDevice.StartSync(1);

	//Initiating packet delivary

	CRecvSock recvSock(local_ip, udp_port, NULL);

	sendSock = new CSendSock(remote_ip, udp_port, 2000);

	printf("\n\n");

	BOOL bRet;
	MSG msg;
	while( (bRet = GetMessage(&msg, NULL, 0, 0)) != 0 )
	{
		if (bRet == -1 )
		{
		}
		else
		{
			TranslateMessage( &msg );
			DispatchMessage( &msg );
		}
	}
	return nRetCode;
};
```

**File name: RecvSock.cpp**

```
#include "StdAfx.h"
#include "RecvSock.h"

CString GetDateTime();
double GetSecs(CString tm);
CRecvSock::CRecvSock(char *LocalIP, int Port, OnRecvCallbackType
OnRecvCallback)
{
	CAsyncSocket::CAsyncSocket();
	int bStat = Create(Port, SOCK_DGRAM, FD_READ | FD_WRITE, LocalIP);
	if(!bStat)
	{
		printf("Error creating Socket.\n");
		int err = GetLastError();
		return;
	}
```

```
	m_hLogFile = (HFILE)CreateFile("c:\\log.txt",
		GENERIC_WRITE,
		FILE_SHARE_WRITE,
		NULL,
		CREATE_ALWAYS,
		FILE_ATTRIBUTE_NORMAL, 0);
	if(m_hLogFile==HFILE_ERROR)
	{
		printf("Error creating log file.\n");
		return;
	}
	m_OnRecvCallback = OnRecvCallback;
}

CRecvSock::~CRecvSock(void)
{
}

void CRecvSock::OnReceive(int nErrorCode)
{
	static int i=0;
	i++;
	char lpBuff[1024];
	int nRead;
	DWORD nWritten = 0;

	nRead = Receive(lpBuff, 1024);

	switch (nRead)
	{
	case 0:
		Close();
		break;
	case SOCKET_ERROR:
		if (GetLastError() != WSAEWOULDBLOCK)
		{
			printf("Error occurred\n\n");
			Close();
		}
		break;
	default:
		lpBuff[nRead] = 0;
		CString szTemp(lpBuff);
		CString utc = GetDateTime();
		szTemp.AppendFormat(" %s", utc);
		CString temp;
		double t1 = GetSecs(szTemp.Mid(5, 12));
		double t2 = GetSecs(szTemp.Mid(18, 12));
		double t3 = GetSecs(szTemp.Mid(31, 12));

		double offset1 = t2 - t1;
		double offset2 = t3 - t2;
		double rtt = t3 - t1;

		CString display_data;
		display_data.Format("%s t1:%2.3f t2:%2.3f RTT:%2.3f\n", szTemp,
offset1, offset2, rtt);
```

```
            printf("%s", display_data);
            WriteFile((HANDLE)m_hLogFile, display_data,
display_data.GetLength() , &nWritten, NULL);
            if(m_OnRecvCallback != NULL)
            {
                  m_OnRecvCallback(szTemp);
            }
      }
      CAsyncSocket::OnReceive(nErrorCode);
}
// 24:12:13.999
double GetSecs(CString tm)
{
      int hr = atoi(tm.Mid(0, 2));
      int min = atoi(tm.Mid(3, 2));
      double sec = atof(tm.Mid(6, 6));
      double res = (hr * 3600) + (min * 60) + sec;
      return res;
}

CString GetDateTime()
{
      SYSTEMTIME st;
      CString timeStr;
      GetSystemTime(&st);
      //DATE and TIME
      //timeStr.Format("%.2d:%.2d:%.2d.%.3d-%.2d/%.2d/%4d",
      //    st.wHour, st.wMinute, st.wSecond, st.wMilliseconds,
      //    st.wDay, st.wMonth, st.wYear);

      //TIME Only
      timeStr.Format("%.2d:%.2d:%.2d.%.3d",
            st.wHour, st.wMinute, st.wSecond, st.wMilliseconds);
      return timeStr;
};
```

**File name: SendSock.h**

```
#pragma once
#include "StdAfx.h"
#include "RecvSock.h"
char *remote_ip;
int remote_port;
void CALLBACK SendTimerProc(HWND hwnd, UINT uMsg, UINT_PTR idEvent, DWORD
dwTime);
class CSendSock : public CAsyncSocket
{
public:
      CSendSock(char *RemoteIP , int RemotePort = 0, int nInterval = 1000)
      {
            static int i = 1;
            int res = Create(0, SOCK_DGRAM, FD_WRITE, NULL);
            remote_ip = new char[strlen(RemoteIP)];
            strcpy(remote_ip, RemoteIP);
            remote_port = RemotePort;
```

```
		CAsyncSocket::CAsyncSocket();
		HANDLE timerID = (HANDLE)SetTimer(NULL, 0, nInterval,
SendTimerProc);
		printf("Start sending packets every %d milliseconds.\n",
nInterval);
		int err = GetLastError();
	}
public:

	~CSendSock(void)
	{
		KillTimer(0, 1);
	}
};

CSendSock *sendSock;

void CALLBACK SendTimerProc(HWND hwnd, UINT uMsg, UINT_PTR idEvent, DWORD
dwTime)
{
	static int i = 0;
	i++;
	if(i == 999) i = 1;
	printf("#%-3d ->", i);

	CString buff;
	buff.Format("#%-3d %s", i, GetDateTime());
	int res = sendSock->SendToEx(buff, buff.GetLength(), remote_port,
remote_ip);
	int err;
	if(res == SOCKET_ERROR)
	{
		err = GetLastError();
	}
	return;
};
```

**File name: RecvSock.h**

```
#pragma once

//public class void OnReceive(int nErrorCode);
typedef void (CALLBACK* OnRecvCallbackType)(CString);
CString GetDateTime();
class CRecvSock: public CAsyncSocket
{
public:

	virtual void OnReceive(int nErrorCode);
	CRecvSock(char *LocalIP, int Port, OnRecvCallbackType OnRecvCallback);

protected:
	OnRecvCallbackType m_OnRecvCallback;
	HFILE m_hLogFile;
public:
```

```
	~CRecvSock(void);
};
```

## A3. Sending and receiving RTP packets on Windows Mobile 5.0

The excerpt of the relevant source code:

**File name: Send.cpp**

```
// Send.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Send.h"
#include "SendDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CSendApp

BEGIN_MESSAGE_MAP(CSendApp, CWinApp)
	ON_BN_CLICKED(IDC_BUTTON1, &CSendApp::OnBnClickedButton1)
END_MESSAGE_MAP()


// CSendApp construction
CSendApp::CSendApp()
	: CWinApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}


// The one and only CSendApp object
CSendApp theApp;

// CSendApp initialization

BOOL CSendApp::InitInstance()
{
#if defined(WIN32_PLATFORM_PSPC) || defined(WIN32_PLATFORM_WFSP)
    // SHInitExtraControls should be called once during your application's
initialization to initialize any
    // of the Windows Mobile specific controls such as CAPEDIT and SIPPREF.
    SHInitExtraControls();
#endif // WIN32_PLATFORM_PSPC || WIN32_PLATFORM_WFSP
```

```
	if (!AfxSocketInit())
	{
		AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
		return FALSE;
	}

	// Standard initialization
	// If you are not using these features and wish to reduce the size
	// of your final executable, you should remove from the following
	// the specific initialization routines you do not need
	// Change the registry key under which our settings are stored
	// TODO: You should modify this string to be something appropriate
	// such as the name of your company or organization
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	CSendDlg dlg;
	m_pMainWnd = &dlg;
	INT_PTR nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}

void CSendApp::OnBnClickedButton1()
{
}
```

**File name: send.h**

```
// Send.h : main header file for the PROJECT_NAME application
//

#pragma once

#ifndef __AFXWIN_H__
	#error "include 'stdafx.h' before including this file for PCH"
#endif

#ifdef POCKETPC2003_UI_MODEL
#include "resourceppc.h"
#endif

// CSendApp:
// See Send.cpp for the implementation of this class
//

class CSendApp : public CWinApp
```

```
{
public:
	CSendApp();

// Overrides
public:
	virtual BOOL InitInstance();

// Implementation

	DECLARE_MESSAGE_MAP()
	afx_msg void OnBnClickedButton1();
};

extern CSendApp theApp;
```

**File name: CERcvSock.h** // making the socket

```
#pragma once
#include "stdafx.h"
#include "afxsock.h"

class CCERcvSock : public CAsyncSocket
{
public:

	CCERcvSock(CString LocalIP)
	{
		Create(UDP_PORT, SOCK_DGRAM, FD_READ | FD_WRITE , LocalIP);
		CAsyncSocket::CAsyncSocket();
	}
	void OnReceive(int nErrorCode)
	{
		static bool firstTime = true;
		char *lpBuff = new char[255];
		ZeroMemory(lpBuff, 255);
		CString senderAddr;
		UINT iSenderPort;
		int nRead = ReceiveFrom(lpBuff, 255, senderAddr, iSenderPort);
		lpBuff[nRead]=0;
		ASSERT(nRead!=SOCKET_ERROR);
		if(nRead!=SOCKET_ERROR)
		{
			int nWrite = SendTo(lpBuff, nRead, UDP_PORT, senderAddr);
			ASSERT(nWrite!=SOCKET_ERROR);
		}
		if(lpBuff=="STOP")
		{
			delete this;
		}
		delete lpBuff;

	}
```

```
	~CCERcvSock(void)
	{
	}
};
```

## A4. Sending and receiving RTP packets on Windows XP (Vista)

The excerpt of the relevant source code:

**File name: SendW32.cpp**

```
// SendW32.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "SendW32.h"
#include "SendW32Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CSendW32App

BEGIN_MESSAGE_MAP(CSendW32App, CWinApp)
	ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()


// CSendW32App construction

CSendW32App::CSendW32App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}


// The one and only CSendW32App object

CSendW32App theApp;


// CSendW32App initialization

BOOL CSendW32App::InitInstance()
{
	CWinApp::InitInstance();

	if (!AfxSocketInit())
	{
		AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
		return FALSE;
	}
```

```
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	// of your final executable, you should remove from the following
	// the specific initialization routines you do not need
	// Change the registry key under which our settings are stored
	// TODO: You should modify this string to be something appropriate
	// such as the name of your company or organization
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));

	CSendW32Dlg dlg;
	m_pMainWnd = &dlg;
	INT_PTR nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

## A5. Minisip GUI in Windows Mobile 5.0

The excerpt from the source code:

**File name: WinGUI.cpp**

```
#include "WinGui.h"
#include "resource.h"

CWinGui::CWinGui(void)
{
}

CWinGui::~CWinGui(void)
{
}

HWND CWinGui::CreateMainDlg(void)
{
	HMODULE hModule = GetModuleHandle(NULL);
	hMainDlg = CreateDialog(hModule, MAKEINTRESOURCE(IDD_MAINDLG), NULL,
DialogProc);
	ASSERT(hMainDlg != NULL);
	return hMainDlg;
```

```
}
void CWinGui::setSipSoftPhoneConfiguration(MRef<SipSoftPhoneConfiguration *>
sipphoneconfig)
{
	sipConfig = *sipphoneconfig;
	return;
}
void CWinGui::setContactDb(MRef<ContactDb *> contactDb)
{
}
void CWinGui::handleCommand(CommandString command)
{
	string stat = command.getOp();

	if(stat == SipCommandString::incoming_available)
	{
		callId = command.getDestinationId();
		stat.append(" from ");
		stat.append(command.getParam());
	}
	if(stat == SipCommandString::incoming_im)
	{
		stat = "IM from " + command.getParam2() + ":" +
command.getParam();
	}

	newStatus(stat);
}
bool CWinGui::configDialog( MRef<SipSoftPhoneConfiguration *> conf )
{
	return true;
}

void CWinGui::run()
{
	CreateMainDlg();
	MSG msg;
	while (GetMessage (&msg, NULL, 0, 0))
	{
		if(msg.message == WM_CLOSE)
		{
			return;
		}
		TranslateMessage (&msg);
		DispatchMessage (&msg);
	}
	return;
}

BOOL CALLBACK CWinGui::DialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
	switch(uMsg)
	{
	case WM_COMMAND:
		if(HIWORD(wParam) == BN_CLICKED)
		{
```

```
                  switch(LOWORD(wParam))
                  {
                  case(IDC_CALL):
                        {
                              HWND hDest = GetDlgItem(hwndDlg, IDC_DEST);
                              WCHAR *lpBuff = new WCHAR[16];
                              GetWindowText(hDest, (LPWSTR)lpBuff, 16);
                              string *dest = toString(lpBuff);
                              callback->guicb_doInvite(*dest);
                              delete lpBuff;
                              break;
                        }
                  case(IDC_HANGUP):
                        {
                              CommandString
command(callId,SipCommandString::hang_up);
                              callback->guicb_handleCommand(command);
                              break;
                        }
                  case(IDC_ANSWER):
                        {
                              CommandString
command(callId,SipCommandString::accept_invite);
                              callback->guicb_handleCommand(command);
                              break;
                        }
                  case(IDC_REFUSE):
                        {
                              CommandString
command(callId,SipCommandString::reject_invite);
                              callback->guicb_handleCommand(command);
                              break;
                        }
                  case(IDC_IM):
                        {
                              WCHAR *lpMes = new WCHAR[50];
                              WCHAR *lpDest = new WCHAR[20];

                              GetDlgItemText(hwndDlg, IDC_IMTEXT, lpMes, 50);
                              GetDlgItemText(hwndDlg, IDC_DEST, lpDest, 20);
                              string *s1 = toString(lpMes);
                              string *s2 = toString(lpDest);
                              CommandString command("",
SipCommandString::outgoing_im, *s1, *s2);
                              callback->guicb_handleCommand(command);
                              SetDlgItemText(hwndDlg, IDC_IMTEXT, _T(""));
                              break;
                        }
                  case(IDC_CONFIG):
                        {
                              CSipConfDlg *confDlg = new
CSipConfDlg(sipConfig);
                              confDlg->doModal();
                              delete confDlg;
                              break;
                        }
                  case(IDC_REGISTER):
```

```
                        {
                              CommandString command("",
SipCommandString::proxy_register);
                              callback->guicb_handleCommand(command);
                              break;
                        }
                  }
            }
            break;
      }
      return DefWindowProc(hwndDlg, uMsg, wParam, lParam);
}
void CWinGui::newStatus(string stat)
{
      WCHAR *lpStr = new WCHAR[500];

      GetDlgItemText(hMainDlg, IDC_LOG, lpStr, 500);
      if(wcslen(lpStr) == 0)
            wsprintf(lpStr, L"%s", toUnicode(stat));
      else
            wsprintf(lpStr, L"%s\r\n%s", lpStr, toUnicode(stat));

      SetDlgItemText(hMainDlg, IDC_LOG, lpStr);
      return;
}

WCHAR* CWinGui::toUnicode(string str)
{
      WCHAR *lpBuff = new WCHAR[str.size()];
      MultiByteToWideChar(CP_ACP, MB_COMPOSITE, str.c_str(), str.size(),
lpBuff, str.size());
      return lpBuff;
}

string *CWinGui::toString(WCHAR *lpStr)
{
      char *cStr = new char[wcslen(lpStr)];
      WideCharToMultiByte(CP_ACP, WC_COMPOSITECHECK, lpStr, wcslen(lpStr),
cStr, wcslen(lpStr), NULL, NULL);
      return new string(cStr);

}
void CWinGui::setCallback(GuiCallback *callback){
      this->callback=callback;
}
```

**File name: SIPConfDlg.cpp**

```
#include "SipConfDlg.h"

CSipConfDlg::CSipConfDlg(SipSoftPhoneConfiguration *conf)
{
      sipConf = conf;
}
CSipConfDlg::~CSipConfDlg(void)
```

```
{
}
void CSipConfDlg::doModal(void)
{
	createConfDlg();
	//readSipConfig();
}
void CSipConfDlg::createConfDlg(void)
{
	HMODULE hModule = GetModuleHandle(NULL);
	hConfDlg = (HWND)DialogBox(
		hModule,
		MAKEINTRESOURCE(IDD_CONFDLG),
		NULL,
		CSipConfDlg::dialogProc);
	//ASSERT(hConfDlg != NULL);
}
BOOL CALLBACK CSipConfDlg::dialogProc(HWND hwndDlg, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
	static bool firstTime = true;
	switch(uMsg)
	{
	case WM_COMMAND:
		if(HIWORD(wParam) == BN_CLICKED)
		{
			switch(LOWORD(wParam))
			{
			case(IDC_OK):
				{
					applySipConfig();
					firstTime = true;
					EndDialog(hConfDlg, 0);
					break;
				}
			case(IDC_CANCEL):
				{
					firstTime = true;
					EndDialog(hConfDlg, 0);
					break;
				}
			}
		}
		break;
	}
	BOOL res = DefWindowProc(hwndDlg, uMsg, wParam, lParam);
	if(firstTime  && uMsg == WM_PAINT)
	{
		firstTime = false;
		hConfDlg = hwndDlg;
		readSipConfig();
	}
	return res;
}
WCHAR* CSipConfDlg::toUnicode(string str)
{
	WCHAR *lpBuff = new WCHAR[str.size()];
```

```
	MultiByteToWideChar(CP_ACP, MB_COMPOSITE, str.c_str(), str.size(),
lpBuff, str.size());
	return lpBuff;
}

string* CSipConfDlg::toString(WCHAR *lpStr)
{
	char *cStr = new char[wcslen(lpStr)];
	WideCharToMultiByte(CP_ACP, WC_COMPOSITECHECK, lpStr, wcslen(lpStr),
cStr, wcslen(lpStr), NULL, NULL);
	return new string(cStr);

}
void CSipConfDlg::applySipConfig(void)
{
	SipIdentity *ident = **sipConf->identities.begin();
	SipProxy* sipProxy = *ident->getSipProxy();

	//Set SIP Proxy and UDP port
	string *strProxy = toString(getDlgField(IDC_PROXY));
	int nPort = _wtoi(getDlgField(IDC_PORT));
	sipProxy->setProxy(*strProxy, nPort);
	delete(strProxy);
	//Set User Pass
	string *strUser = toString(getDlgField(IDC_USER));
	sipProxy->sipProxyUsername= *strUser;
	delete(strUser);

	string *strPass = toString(getDlgField(IDC_PASS));
	sipProxy->sipProxyPassword= *strPass;
	delete(strUser);

	//Set STUN
	string *strStun = toString(getDlgField(IDC_STUN));
	sipConf->stunDomain = *strStun;
	delete(strStun);
	sipConf->save();

}
WCHAR* CSipConfDlg::getDlgField(int IID)
{
	WCHAR* str = new WCHAR[100];
	GetDlgItemText(hConfDlg, IID, str, 100);
	return str;
}

void CSipConfDlg::setDlgField(int IID, WCHAR* value)
{
	SetDlgItemText(hConfDlg, IID, value);
	return;
}
void CSipConfDlg::readSipConfig(void)
{
	SipIdentity* ident = **sipConf->identities.begin();
	SipProxy* sipProxy = *ident->getSipProxy();

	setDlgField(IDC_PROXY, toUnicode(sipProxy->sipProxyAddressString));
```

```
        WCHAR strPort[3];
        _itow(sipProxy->sipProxyPort, strPort, 10);
        setDlgField(IDC_PORT, strPort);

        setDlgField(IDC_USER, toUnicode(sipProxy->sipProxyUsername));
        setDlgField(IDC_USER, toUnicode(sipProxy->sipProxyPassword));
        setDlgField(IDC_STUN, toUnicode(sipConf->stunDomain));

        return;
}
```

COS/CCS 2008-09

www.kth.se