IP Telephony: Peer-to-peer versus SIP

CARLOS MARCO ARRANZ

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2005

IMIT/LCN 2005-13

IP telephony: Peer-to-Peer versus SIP

Master of Science Thesis Performed at
Wireless@KTH, June 2005

Student: Carlos Marco Arranz

Supervisor and Examiner: Prof. Gerald Q. Maguire Jr.

School of Information and Communication Technology (ICT)
Royal Institute of Technology (KTH)
Stockholm, Sweden

Abstract

In recent years dramatic technology developments have exploited the development of better transmission media and allowed for broad internet penetration. This in turn has fostered the growth of IP telephony calls, i.e., Voice over IP (VoIP).

New VoIP products are introduced almost daily, each seeking an opportunity in the market. Some of these products are free - thus putting pressure on other vendors. A good example of a commercial VoIP product is Skype. It is possibly the most important one as it has gained more than 3 millions users in approximately 2 years time. In contrast, Minisip is a non-commercial implementation of SIP developed by students at KTH. These programs are based on different architectures. While Skype is said to be based on a peer-to-peer protocol, Minisip utilized the Session Initiation Protocol (SIP) protocol.

The aim of this thesis was to evaluate these two VoIP programs not only in terms of development, but also in terms of the quality of service and user perceived voice quality. The study of efficiency, usability, and installation of both are also in the scope of this thesis. The devices used for the evaluation included a HP iPAQ 5550, two PCs running in RedHat Linux 9, and a laptop running Microsoft Windows XP.

Sammanfattning

På senare år har den dramatiska teknikutvecklingen exploaterat utvecklingen av bättre överföringsmedia samt möjliggjort för en bred Internetpenetrering. Det i sin tur har gynnat ökningen av telefonsamtal med IP-telefoner, d.v.s. Voice over IP (VoIP).

Nya VoIP produkter introduceras nästan dagligen och varje produkt söker sin möjlighet på marknaden. Vissa av dessa produkter är gratis och sätter alltså press på andra försäljarna. Ett bra exempel på en VoIP produkt är Skype. Det är möjligtvis den viktigaste produkten då den har fått tre miljoner användare på ungefär två års tid. I kontrast till detta finns Minisip som är en icke kommersiell implementation av SIP, utvecklad av studenter från KTH. Dessa program är baserade på olika arkitekturer. Medan Skype är baserat på ett peer-to-peer protokoll, utnyttjar Minisip protokollet Session Initiation Protocol (SIP) Protocol.

Syftet med denna avhandling var att evaluera dessa två VoIP program, inte bara i termer av utveckling utan också i termer av ”quality of service” och hur användaren uppfattar röstkvaliteten. Studien innefattar också effektivitet, användarvänlighet och installation av de båda programmen. Enheterna som användes under denna evaluering var en HP iPAQ 5550, två pc:s med Linux Red Hat 9 samt en bärbar pc med Windows XP.

Acknowledgment

There are many people who made this work possible. First of all, I am highly thankful to my advisor Gerald Q. Maguire Jr. for his time, his advice and valuable ideas during this thesis and also for giving me the opportunity and resources needed to pursue this thesis. I would also like to thank Johan Billien and Erik Eliasson for their time helping me to solve problems with Minisip.

I would like to extend my thanks to my family, my girlfriend Oya Yilmaz and all my friends for their support and encouragements throughout my studies. I could not have done it without you, I am really grateful to your help.

Thanks as well to everyone else in the Wireless@KTH department that have helped me in anyway.

Table of contents

1 Introduction

In the last years, an incredible development has happened in the different technological fields. Transmission mediums such as optical fiber have improved data transmissions providing both higher speed and lower noise. This rapid evolution in security and quality has allowed telephony over the network to become a reality, so that, better telephone service is available for users. However, this technological development has happened in a short period of time and has generated a huge hole in the market driven by Voice over IP which a lot of companies are trying to take advantage of.

Perhaps the VoIP product with the greatest development and best known (currently) is Skype. This product, in just two years, has spread all over the world with the advertisement "make calls for free". In fact, more than 3.5 million users use Skype, more than 100 million downloads have been done, and these numbers increase every day.

However, at the same time, new open source VoIP products are being introduced to provide services to clients without cost. An example of these is Minisip, an SIP agent which provides videoconferencing, telephone conferencing, or instantaneous message services in a efficient and secure way.

In this thesis, I compare both VoIP products' features, user interface, and performance in order to explain the current situation in both products. I also assess the quality of service provided to the user, not only in terms of quality of voice perceived, but also the support given to the user. I also measure which program is more efficient in the voice transfer. These programs are based on different architectures. While Skype is based on a peer-to-peer architecture (section 4.5), Minisip utilizes the SIP protocol (section 4.3)

The comparison of the features permits me to describe the services each program provides and on what operating systems the application runs so that services common to both programs can be compared and the differences can be emphasized.

The comparison of the user interface begins with a description of the support for the user, the development of applications in each program, as well as the usability of the program and quality of voice perceived by the user. To study this quality, a graphic user interface (GUI) was created to hide the program interface from users in order to obtain objective evaluations.

Finally, measurements are performed and the programs´ behaviours are monitored using the Ethereal program, from this we can determine the efficiency of both programs by knowing which messages correspond to what information. This necessitated a study of the Skype protocol at network level.

2 Prior work

The present thesis provides a performance comparison between two VoIP packages: Minisip [38] and Skype [21].

2.1 Minisip

Minisip is a prototype of a VoIP product based on a SIP agent initially written by Erik Eliasson at the Royal Institute of Technology (KTH). This prototype is constantly being improved by various master thesis students and others so that, it enables communication between clients providing:

- A “phone” call
- Instantant messaging
- Videoconferencing

Underneath this communication service, security for signaling and media is provided by an implementation of the MIKEY and SRTP protocols [26], [77] in C++. This is a result of the Johan Bilien´s and Israel Abad Caballero´s master theses.

Other students are studying alternative data encoding, allowing different quality and efficiency, spatial audio [45], etc.

2.2 Skype API

As the comparison was to include the quality perceived by the user when making calls, the initial idea was to implement a graphical user interface (GUI) to hide which program was being used to make calls so that the user does not know the program carrying his call. However, currently there is no Windows version for Minisip, so only the connection with Skype was implemented. To establish the connection between our interface and Skype, the Skype API provided in [21] was modified.

That API is built based on simple text messages that are sent back and forth between Skype and our device [70]. Such a API permits the following types of transfers:

<u>Skype to Device</u>	<u>Device to Skype</u>
- Status commands	- Initiating searches
- Search results	- Getting parameter values
- Notifications	- Setting parameter values
	- Making calls
	- Sending messages
	- Opening dialogs

Moreover, in [70] an example of this API is available, where the commands are enter via a console. I use this example to understand the structure of these messages. Based on this understanding and I build our interface.

2.3 Skype Protocol

Finally, a study at the packet level of both programs was done to learn the traffic generated for each program when sending the different commands. By comparing common services such as call establish, tear down of the call, media traffic, etc. We will examine which product is more efficient.

Unfortunately the Skype protocol is not opened, so the study entitled "The Skype Protocol Analysis" performed by S.Baset at the Department of Computer Science at the Columbia University [63] was considered the initial source of information for this comparison. In this report, Skype´s operation is divided into several functions; then each one of them is studied. These functions are:

- Startup
- Login
- User search
- Call establish/Teardown
- Media transfer and codecs
- Conferencing

I expand upon this earlier study of the Skype protocol (see Appendix A). This study will clarify several points related to the architecture and the encoding used by Skype.

3 Problem Statement

This master thesis can be considered as another step in the evolution of the Minisip product developed by Erik Eliasson, Johan Bilien and Jon-Olov Vatn and was intended to compare it with one of the most important VoIP products nowadays, Skype. An introduction of both programs is provided in section 4.6.

For this thesis I utilized four devices: an HP iPAQ h5550 PDA running Microsoft Pocket PC operating system, an AMD 2800+ laptop running the operating system Microsoft Windows XP, and two PCs running Redhat Linux.

The environment chosen for my development was Microsoft´s Embedded Visual C++ 4.0. This was used to implement the graphic user interface (GUI) for the Pocket PC and Microsoft´s Visual Studio 6.0 to implement the GUI for the laptop. Moreover the connection between my graphic interface and Skype used the API provided by Skype. This is briefly explained in Appendix B.

The codes for both applications and the different simulations and measures performed using Ethereal are not included in this document, but rather provided on a separate CD.

4 Background

4.1 Voice over IP

Today the Internet and mobile telephony are the two most important areas in telecommunications. Both have had huge growth in the number of users in recent years.

IP technology appears to be a substitute for conventional telephony due to, mainly, lower customer prices. However, traditional operators offering local and long-distance calls could decrease the price of these calls - in order to have a similar cost, i.e. similar costs for both circuit-switched and Voice over IP calls. In such a situation, the economical advantage of VoIP would be lost, but other features of VoIP will still favor its use, e.g. for multimedia traffic, easy creation of new services, control of call routing by the user's PC, etc.

Today, there are two main motivations for Internet telephony:

- Cost reduction
- Easy integration of (additional) services

However, some problems must be solved to increase the popularity of VoIP technology. These problems appear as a consequence of the design of the Internet for transporting data and because many vendors do not follow the relevant standards. As it is explained in [1] and [2], these aspects are:

- Quality of voice: As stated above, the Internet was designed to transport data without considering real time services, i.e. it simply provides a best effort service. Voice communication will generally be considered acceptable when the delay is less than 150 ms and when the loss rate is less than 10% [3]. Additional techniques can be used to improve the perceived voice quality such as: Echo Cancellation, Packet Prioritization (e.g. giving higher priority to voice packets), or Forward Error Correction.

- Interoperability: In a public network environment it is necessary to have some level of interoperability, thus products from different companies should be able to interoperate with each other - if VoIP is to become wide-spread.

- Security: The information being communicated by the users and the information to setup calls must be protected in order to provide some level of security - since it is easy to capture and analyze network packets.

- Integration with Public Switched Telephone Network (PSTN): As both PSTN and VoIP will need to operate side by side for several years, thus some solution should be developed to integrate PSTN and VoIP.

- Scalability: Currently major efforts are being made to provide better quality in calls at lower costs by exploiting the high penetration of home broadband. Moreover, VoIP systems need to be flexible enough to grow to a very large user market and to allow a mix of private and public services.

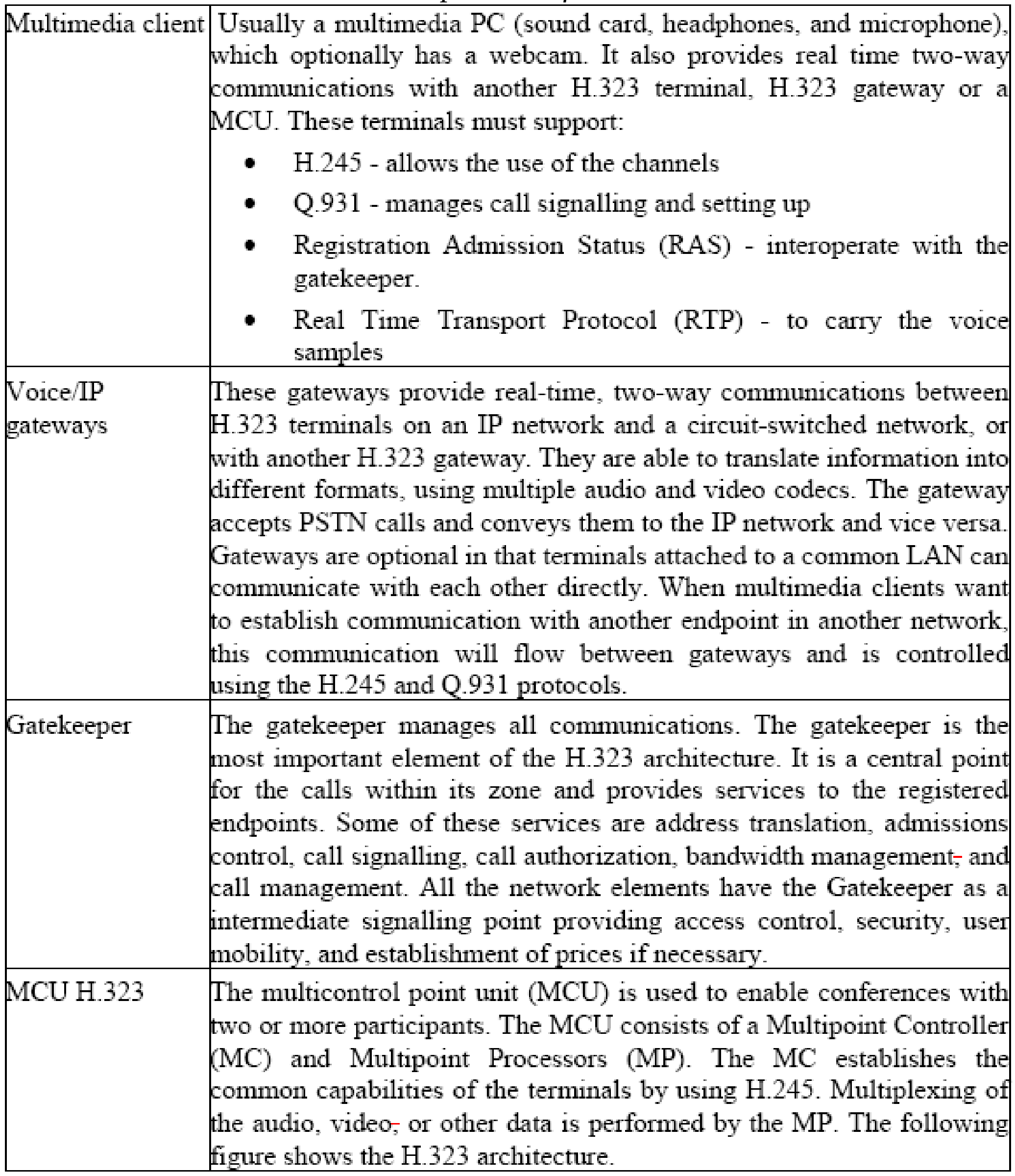
4.2 H.232 Standard

H.323 [4] is a standard developed by the International Telecommunications Union - Telecommunication section (ITU-T). It is utilized in many VoIP products. It defines the technical requirements to provide VoIP in LANs without considering Quality of Service (QoS). Although it was created to support multimedia conferencing over LANs, it is now used for Voice over IP communications. Products based on the H.323 standard should interoperate.

4.2.1 Components of H.323

In [5], [6], and [7] VoIP networks built according to H.323 consist of four fundamental elements:

Table 1. Components of H.323



Multimedia client	Usually a multimedia PC (sound card, headphones, and microphone), which optionally has a webcam. It also provides real time two-way communications with another H.323 terminal, H.323 gateway or a MCU. These terminals must support: • H.245 - allows the use of the channels • Q.931 - manages call signalling and setting up • Registration Admission Status (RAS) - interoperate with the gatekeeper. • Real Time Transport Protocol (RTP) - to carry the voice samples
Voice/IP gateways	These gateways provide real-time, two-way communications between H.323 terminals on an IP network and a circuit-switched network, or with another H.323 gateway. They are able to translate information into different formats, using multiple audio and video codecs. The gateway accepts PSTN calls and conveys them to the IP network and vice versa. Gateways are optional in that terminals attached to a common LAN can communicate with each other directly. When multimedia clients want to establish communication with another endpoint in another network, this communication will flow between gateways and is controlled using the H.245 and Q.931 protocols.
Gatekeeper	The gatekeeper manages all communications. The gatekeeper is the most important element of the H.323 architecture. It is a central point for the calls within its zone and provides services to the registered endpoints. Some of these services are address translation, admissions control, call signalling, call authorization, bandwidth management~~,~~ and call management. All the network elements have the Gatekeeper as a intermediate signalling point providing access control, security, user mobility, and establishment of prices if necessary.
MCU H.323	The multicontrol point unit (MCU) is used to enable conferences with two or more participants. The MCU consists of a Multipoint Controller (MC) and Multipoint Processors (MP). The MC establishes the common capabilities of the terminals by using H.245. Multiplexing of the audio, video~~,~~ or other data is performed by the MP. The following figure shows the H.323 architecture.

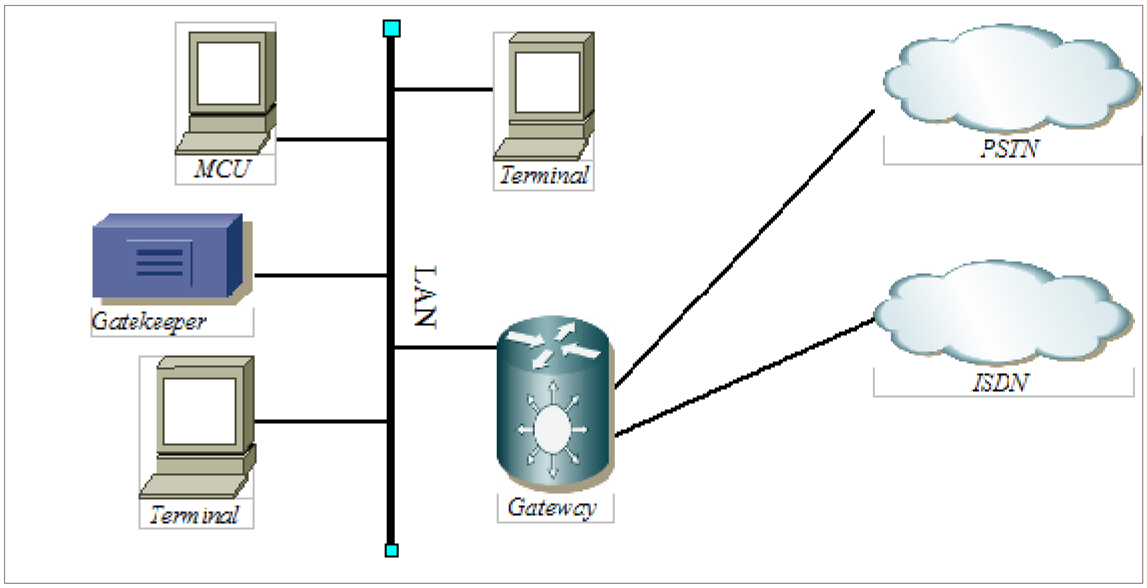


Figure 1. H.323 architecture

4.2.2 H.323 Stack Protocol

The H.323 protocol stack is presented in the figure below. Data, control, and signaling information are transmitted using the Transmission Control Protocol (TCP) whereas audio and registration packets use the User Datagram protocol (UDP).

Data	Control and Signalling		Audio and video	Registration
T.120	H.225.0 call signalling	H.245 Conference Control	RTP/ RTCP	H.225.0 RAS
	TCP		UDP	
		Network Layer		
		Link Layer		
		Physical Layer		

Figure 2. H.323 protocol stack

4.2.3 Control and Signaling

H.323 provides three control protocols:

- Signaling for call control is provided by H.225.0/Q.931 call signaling
- Call establishment from a source to a receiver (host) is performed by H.225.0 RAS
- Media streams will be negotiated once the call is set up with the H.245 protocol

H.225.0 RAS

H.225.0 RAS is the protocol between endpoints (terminals and gateways) and gatekeepers and it is used to perform registration, admission control, bandwidth changes, status, and disengage procedures between terminals [8]. A RAS channel is used to exchange RAS messages over UDP.

H.225.0 Call Signaling

H.225.0 call signaling [8] is a protocol used to establish connections between H.323 endpoints. In order to do that, H.225 protocol messages are exchanged on the call-signaling channel, using TCP port 1720, between two endpoints. This port initiates the Q.931 call control messages with the purpose of connecting, maintaining, and disconnecting calls.

When a gateway is present in the network zone, H.225.0 call setup messages are exchanged either via Direct Call Signaling or Gatekeeper-Routed Call Signaling (GKRCS). The gatekeeper selects the method during the RAS admission message exchange. If there is no gatekeeper, H.225 messages are exchanged directly between the endpoints.

H.245 Media and Conference Control

H.245 [10] is a control signaling protocol inside the H.323 architecture which performs the exchange of end-to-end H.245 messages between two endpoints once they have established communication. The H.245 control messages flow over H.245 control channels and include the information necessary to exchange terminals capabilities and to open and close logical channels.

Once the connection is set up using the call signaling procedure, the H.245 call control protocol is used to determine the call media type and establish the media flow, before the call is established. It also manages the call after it has been established.

4.3 Session Initiation Protocol (SIP)

IETF's SIP [11] standard is used for the establishment, modification, and termination of VoIP connections. This protocol operates at the application layer, and it can create, modify, and terminate sessions with one or more participants. It uses a client-server architecture similar to HTTP [9], where the client generates and sends requests to the server. The server receives and processes the requests and then replies to the client. The complete process is called "transaction".

SIP has two messages, `INVITE` and `ACK`, which are used to open a reliable channel. Messages exchanged over this channel control the call. SIP makes very few assumptions about the underlying transport protocol. SIP can utilize: TCP, TLS, UDP, or SCTP. SIP's `INVITE` also permits user mobility via re-`INVITE`s.

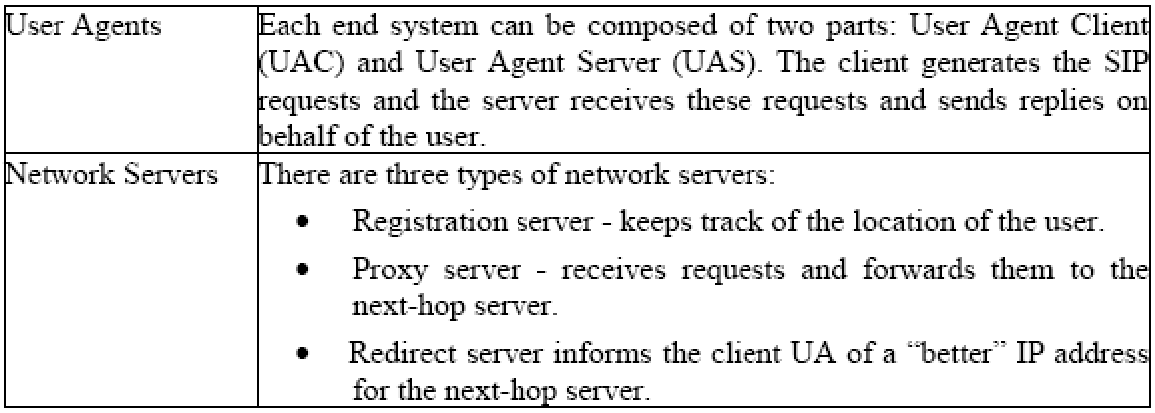
SIP uses the Session Description Protocol (SDP) to perform the negotiation of the codecs and their parameters. This allows the participants to agree upon the set of multimedia types to be used.

SIP provides a number of services, some of these are [11]:

- User location, determining the end-point to be used for the communication,
- Call establishment: making a call and configuring its parameters,
- Determining user availability (including consideration of user preferences), and
- Call management: transfer and termination of calls.

4.3.1 SIP Components

SIP is built upon two different types of components: user agents and network servers.



User Agents	Each end system can be composed of two parts: User Agent Client (UAC) and User Agent Server (UAS). The client generates the SIP requests and the server receives these requests and sends replies on behalf of the user.
Network Servers	There are three types of network servers: • Registration server - keeps track of the location of the user. • Proxy server - receives requests and forwards them to the next-hop server. • Redirect server informs the client UA of a "better" IP address for the next-hop server.

4.3.2 SIP Messages

The messages SIP can use for communication between client and server are:

- `INVITE`: invites the user to join a session (call).
- `BYE`: ends the session.
- `ACK`: an acknowledgement
- `OPTIONS`: exchanges information related to the caller/callee capabilities.
- `REGISTER`: tells the SIP Registration Server the user's current location.
- `CANCEL`: cancels a SIP request.

4.3.3 Overview of SIP Functionary

Each SIP terminal is identified by one or more SIP addresses. When a user wants to make a SIP call, the first step is to find the location of the appropriate server and to send to that server the `INVITE` request. Two situations could occur:

- the caller is able to directly reach the callee user, or
- the caller needs to communicate with one or more Proxy and/or Redirect servers to reach the destination.

The Call ID field in the SIP message uniquely identifies a given call.

4.3.4 SIP Addressing

In the SIP architecture, each user is identified using one or more SIP Uniform Resource Identifier (URIs). These addresses are of the form: sip:username@domain. Note, that such a URI can identify a user or a group of users (or even terminals).

4.3.5 Locating a SIP Server

When a client wants to establish communication with another server, first it sends a message to the proxy with the SIP URI of the user to be called. Then the proxy performs a search of the registrar sever for that client, this is similar to locating a SMTP server for a given e-mail address. There are several ways to try to connect with the registrar server [16]:

- We know the IP address of the registrar associated with this SIP URI.
- The registrar's network address is designated by a DNS SRV record for the domain contained in the SIP URI.
- The SIP URI's domain part has an DNS A record or AAAA record of IPv6 in the case, which points to the registrar
- The SIP URI's domain, prepended with 'sip.' has a DNS A record or AAAA record which points to the registrar.

However, usually the client relies on its own SIP proxy server to perform the search.

4.3.6 SIP Transaction

Once the domain part of the URI is resolved, the client can send the request to the identified server. Such a request together with its reply constitutes a SIP transaction. The request and the replies could be sent over TLS, TCP, UDP, SCTP.

4.3.7 SIP Invitation

In a SIP invitation, two messages are sent: `INVITE` and `ACK`. The **`INVITE`** request, asks the callee if he wants to join the conversation. This request contains several parameters which permits the callee to participate and describes the proposed session. If the callee's client accepts the call, it will reply to the `INVITE` request by sending its proposed description of the session. After receiving this, if the caller wishes to establish such as session, then the caller sends to the callee an `ACK`.

4.3.8 Locating a User

The callee may change its position over time (i.e., user mobility is permitted) and the new location will be dynamically registered with the callee's Registration SIP server. When this SIP server is interrogated to learn the location of the callee, it returns a list of possible locations. Actually this list is created by a Location Sever which provides it to the SIP server However, this communication does **not** use SIP.

4.3.9 Modifying an Existing Session

The SIP protocol supports the modification of the parameters of an ongoing session. To do this, another INVITE message is sent with the **same** call identification, but with new parameters.

4.3.10 Sample SIP Session

This section will illustrate SIP's operation using a simple example. In this example, a SIP client sends an `INVITE` message to the proxy server indicating that the destination is the SIP user `torobravo7@sip1.it.kth.se`. The proxy server next obtains the IP address of the SIP server which handles or manages requests for the domain (e.g., sip1.it.kth.se). The proxy server communicates with the Location server to determine the next hop server. Actually, the Location server is not a SIP server, but maintains the registration information for each user.

The Proxy server forwards the `INVITE` request to the next hop server in order to learn its IP address. Finally the User Agent Server (UAS) in the destination is reached and it replies to the Proxy server. Now the Proxy server sends the reply to the client and the client responds with an `ACK` acknowledgement.

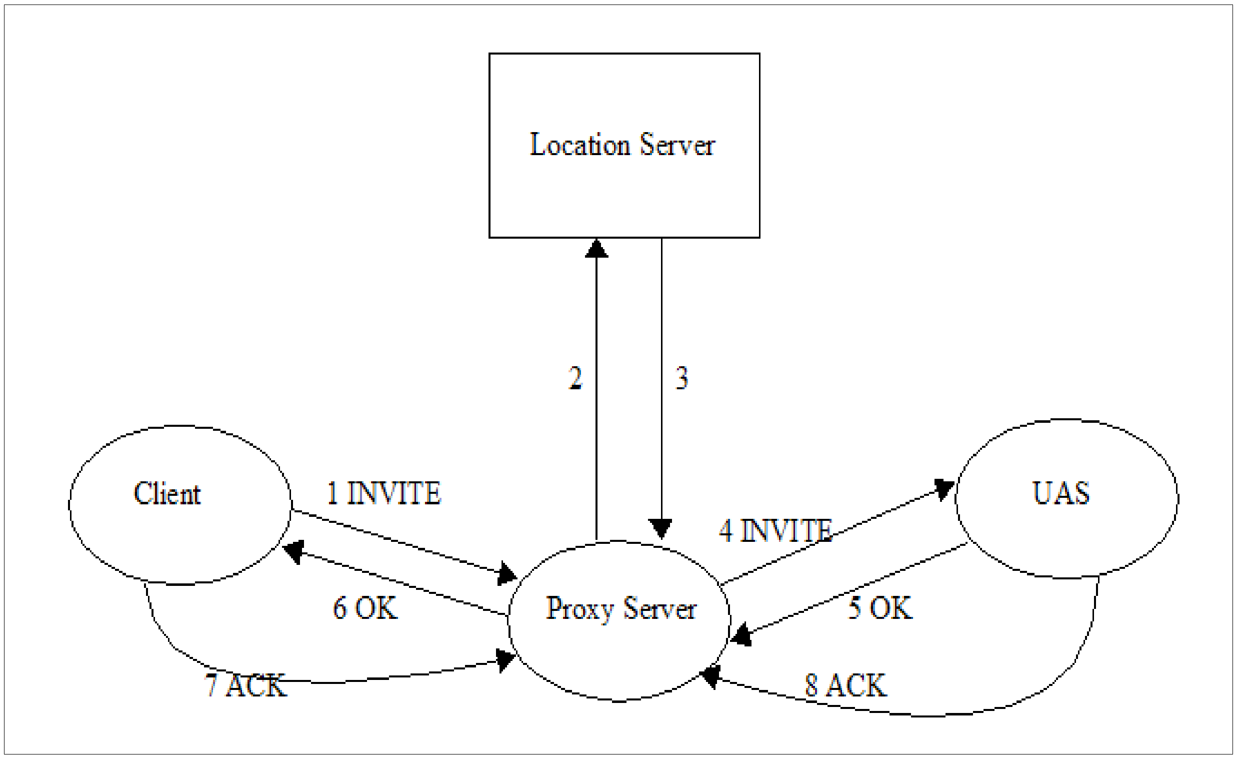


Figure 3. Example of operation in SIP

4.4 Comparison of H.323 with SIP

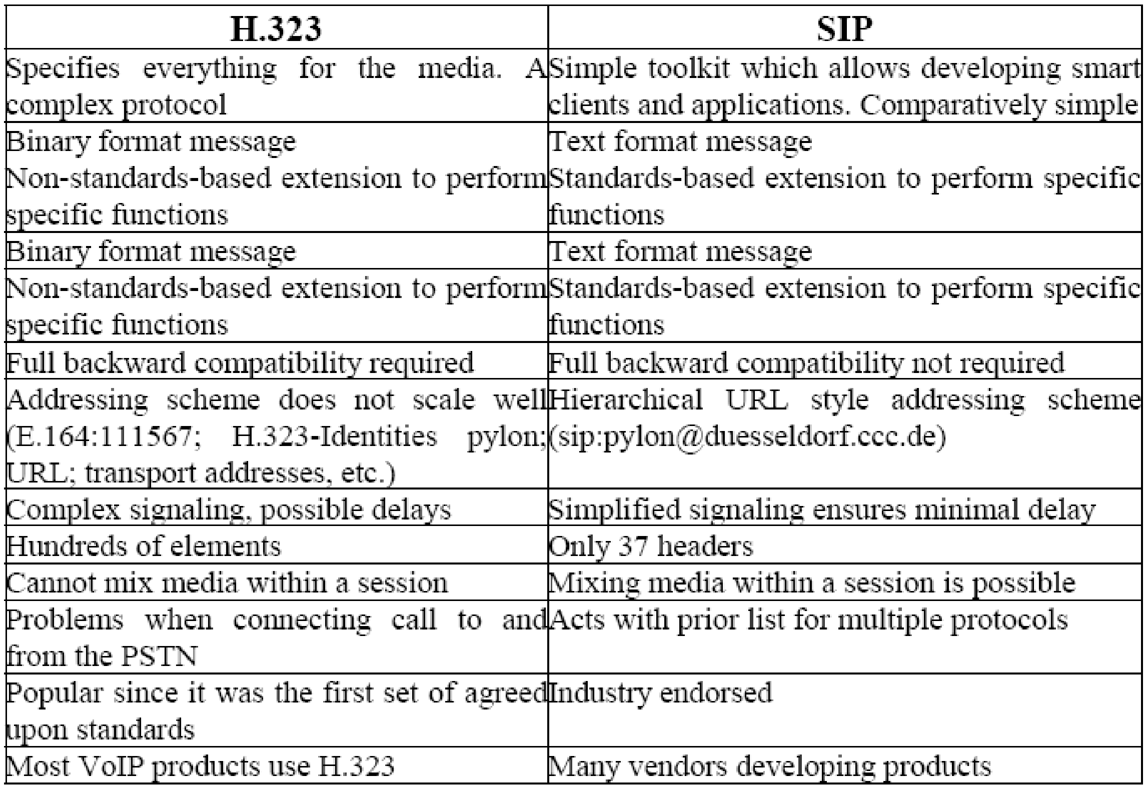
The developers of the SIP protocol believe that the H.323 protocol has high complexity and overhead. In contrast, SIP was designed to avoid complexity, thus SIP reuses many of the header fields, formats, error codes, and the authentication mechanism used in HTTP. Moreover, SIP defines only 37 headers, each one of them with a small number of values and parameters - whereas H.323 has hundreds of elements.

Another difference between both protocols is that H.323 uses a binary representation for its messages, based on ASN.1 - while in SIP the messages are encoded as text, just as for HTTP.

H.323 does not scale which is necessary for VoIP to be successful [1]. As was indicated previously; this is because the design is focused on LAN settings rather than on internet setting. H.323 has limited scalability regarding loop detection in complex multi-domain searches since it performs these by keeping message states. In contrast, SIP uses a loop detection method based on checking the message history which is saved inside the header fields themselves - thus avoiding the need to keep state.

SIP has the advantage of being supported by IETF, but H.323 currently has an advantage of a larger market. The comparison between the two is summarized in the following table:

Table 2: Comparison of H.323 and SIP



H.323	SIP
Specifies everything for the media. A complex protocol	Simple toolkit which allows developing smart clients and applications. Comparatively simple
Binary format message Non-standards-based extension to perform specific functions	Text format message Standards-based extension to perform specific functions
Binary format message	Text format message
Non-standards-based extension to perform specific functions	Standards-based extension to perform specific functions
Full backward compatibility required	Full backward compatibility not required
Addressing scheme does not scale well (E.164:111567; H.323-Identities pylon; URL; transport addresses, etc.)	Hierarchical URL style addressing scheme (sip:pylon@duesseldorf.ccc.de)
Complex signaling, possible delays	Simplified signaling ensures minimal delay
Hundreds of elements	Only 37 headers
Cannot mix media within a session	Mixing media within a session is possible
Problems when connecting call to and from the PSTN	Acts with prior list for multiple protocols
Popular since it was the first set of agreed upon standards	Industry endorsed
Most VoIP products use H.323	Many vendors developing products

4.4.1 Supporting Protocols

SIP protocol operates together with other protocols:

- Real-time Transport Protocol (RTP) & RTCP - used for transmission of data with real time properties.
- [Optionally] Resource Reservation Protocol (RSVP) - allows resources to be reserved.
- Session Description Protocol (SDP)

The H.323 protocol also utilizes RTP and RTCP. Current voice gateways utilize RTP and RTCP at a media gateway; while the signaling gateway communicates with the media gateway using the Media Gateway Access Protocol (MGCP) protocol. MGCP is able to work with both SIP and H.323.

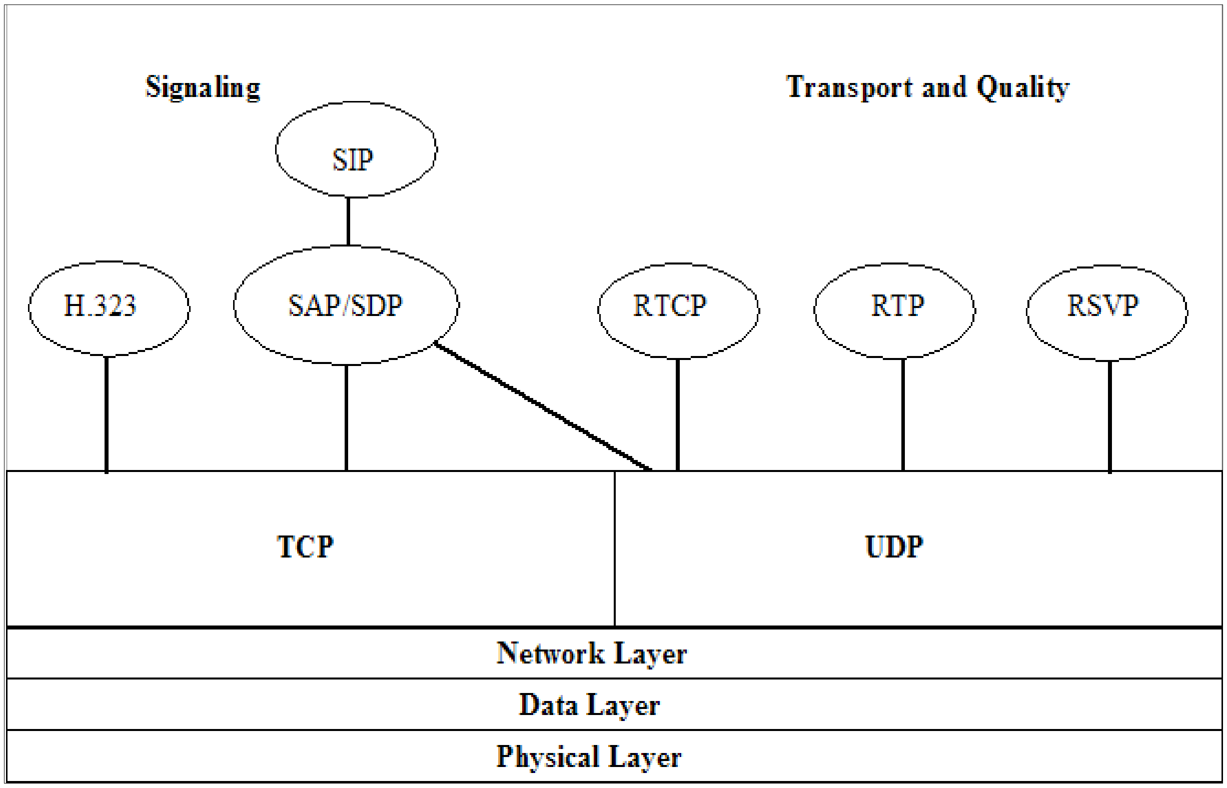


Figure 4. Protocol layering

4.4.2 RTP and RTCP

The Real-time Transport Protocol (RTP) [12], [13] is an Internet protocol which can be used to transmit data with real-time properties, such as audio or video. This protocol does **not** ensure real-time delivery of the multimedia data, but provides information to both order the data and detect data loss at the receiver. The RTP header also indicates to the receiver which codec was used to produce this packet.

RTP works together with a control protocol (RTCP), which allows the monitoring of data delivered in large multicast network. The receiver can inform the sender of the level of lost packets, jitter, and etc. while senders can indicate their identity and other information in RTCP packets. As a general rule, both RTP and RTCP are carried over UDP.

The RTP components include:

Sequence number	allows the detection of lost packets
Payload identification	describes the encoding of the multimedia data
Frame identification	indicates the limits of a frame (beginning and end)
Source identification	identifies the source of the frame
Intramedia synchronization	timestamps allow jitter detection within a frame and can be used in conjunction with a de-jitter buffer

The RTCP components are:

Quality of service feedback	indicates the number of lost packets, the round trip time, and jitter, so that the sources can modify their transmission rates, coding, etc.
Session control	indicates that a user is leaving a session by sending a BYE packet.
Identification	Can be used to provide the name, E-mail address, telephone number, … of the participant
Intermedia synchronization	Permits synchronizing separate audio and video streams

4.4.3 SDP

The SIP communications requires a protocol to describe the sessions. This protocol is SDP (Session Description Protocol) [15] which permits the exchange between users of information related to the type of data, the transport protocol to be used, or the port to utilize in the communication. The protocol has different fields to indicate this information [16]:

- Origin field: contains information about the user that initiates the session.
- Session name field: title for the session.
- Email address field: contains the IP address of the caller.
- Phone number field: contains the telephone number of the caller.
- Connection Data field: provides information about the network connection to establish the session.
- Media Announcement field: indicates the kind of data to be exchanged.
- Attribute field: gives additional properties for the session and for one of the streams.

The negotiation performed using SDP begins by sending an SDP offer (proposal) from one of the terminals (the SIP initiator) which can include the additional information noted above. The destination terminal will reply to this message describing its capacities in a 200 OK message.

4.4.4 RSVP

The Resource Reservation Protocol (RSVP) [14] can be utilized by terminals (host) to request a specific quality of service from the network. The protocol must also be used by all the routers along the path over which the media will flow. If all the routers are able to supply the requested resources, then they establish and maintain the state to support the requested service. As a result resources are reserved in every node on the path.

4.4.5 Peer-to-Peer Architecture

A peer-to-peer architecture, abbreviated as P2P, defines a network in which each node has the same capabilities and responsibilities [17]. It can be seen as the opposite of a client-server architecture, in which some node(s) focus on serving requests sent by others.

Peer-to-peer networks are widely used to share information such as video, audio, files, or other documents in a digital format. Specific nodes are not designated as servers, but rather all the nodes work as both servers and clients (often simultaneously).

There are a number of famous “peer-to-peer” networks, such as Napster [18] and Gnutella [19], that mix a client-server architecture with a peer-to-peer architecture. These networks utilize servers that inform peers about the addresses of other peers, thus providing better run-time performance. For example, Napster uses a centralized file list to improve its service, by improving the speed of searching.

4.4.6 Advantages of peer-to-peer networks

One of the most important advantages of peer-to-peer networks is that the bandwidth of all the clients can be used, thus the more users are (and hence clients), the higher the bandwidth available. In a client-server architecture all the clients share the bandwidth to and from the server, so the greater the number of clients is, the lower their share of the limited bandwidth is; thus in this setting the client generally has a lower average transfer rate.

4.4.7 Generations of peer-to-peer networks

First generation

The first generation of peer-to-peer networks was designed for sharing files and utilized a centralized file list. An example of this generation is Napster [18]. In this centralized peer-to-peer model, a user sends a request or search query to a centralized server indicating the information desired. Then, that server replies with a list which contains the peers that have the information. The client subsequently connects to one of these peers and download the desired file.

Second generation

Napster encountered major legal problems, and a new network called Gnutella [19] was introduced to avoid the problems caused by centralized nodes. In the Gnutella network all the nodes are equivalent, i.e. they all have the same roles even though some may have more memory, processing power, bandwidth, etc. than others. However, this new model suffered from several major bottlenecks as the former Napster users joined the new network. The result was the creation of yet another network called FastTrack [20], in which some nodes are more capable than others. In this new architecture nodes with greater storage capacity provide an index to other nodes, thus generating a tree structure. This dramatically increases the scalability. Gnutella uses the same solution; in fact, most current peer to peer networks utilize this structure since it makes it possible to create very large networks.

The second generation also utilizes the hash tables. Here several nodes are selected to index hashes which identify files. As a consequence, it is possible to perform searches faster.

Third generation

Examples of third generation networks are Freenet [71], I2P [72], GNUnet [73], or Entropy [74]. These networks have integrated anonymity features so that only known contacts will be able to access your computer. Moreover, each user can forward a request (and files) between friends, hence increasing the anonymity. The problem in these networks is the overhead caused by providing anonymity.

4.5 Skype and Minisip

Skype

Skype is a VoIP package based on a peer-to-peer architecture which diferenciates it from other VoIP products. The Skype program allows clients to make telephone calls between them for free, and even to call regular telephone numbers paying for using the SkypeOut service. In [21] and [22] it is stated that the Skype user directory is completely *decentralized* and *distributed* between the nodes in the network providing scalability of the Skype network while avoiding high costs. However, with the increasing number of Skype clients, some centralized elements were added. On April 15th, 2005 the number of downloads was ~100 millions and the number of users ~3.5 millions.

Skype can route calls through other peer-to-peer networks and is able to traverse NATs and firewalls. This is not true of most other VoIP programs. Moreover, the selection of computers to be used is done automatically; without the users having a choice to avoid the utilization of their resources by other Skype clients. This point has not been properly explained and it could be a "contradiction with the license agreement of facilitating the communication between the user and other Skype Software users" [51].

Minisip

Minisip implements a SIP user agent. It is being developed as a platform to explore services that are not available in the traditional telephony networks today [23].

The Minisip program currently available for Linux allows the user to exchange video, audio or text; while providing additional security features such as mutual authentication, encryption, and integrity of on-going calls, and encryption of the signaling (SIP over TLS)[24]. To provide this security the SRTP [25] and MIKEY [26] IETF standards were implemented and added to the program.

4.6 AES

AES [27] is a block cipher used as a standard for encryption by the government of the United States of America. Both software and hardware implementations are possible. The algorithm is designed to be fast and easy to implement without requiring a large memory. Currently, the AES standard is being used in many applications.

AES is sometimes called Rijndael, but really they are not exactly the same cipher algorithm. In the case of Rijndael, the size of the block is not fixed - whereas in AES blocks of 128 bytes are used. The size of the key can vary among 128, 192, and 256 bits. In the Rijndael algorithm the size of the block is a multiple of 32 over the range 128 to 256 bits.

The AES algorithm utilizes arrays of 4x4 bytes and each round consists of four steps:

1. **SubBytes**: Each byte in the array will be replaced by its entry in a fixed lookup table. This operation introduces the non-linearity in the cipher algorithm. Using a lookup table avoids runtime computation. The speed gain can be significant, since retrieving a value from memory is often faster than performing an (expensive) computation.

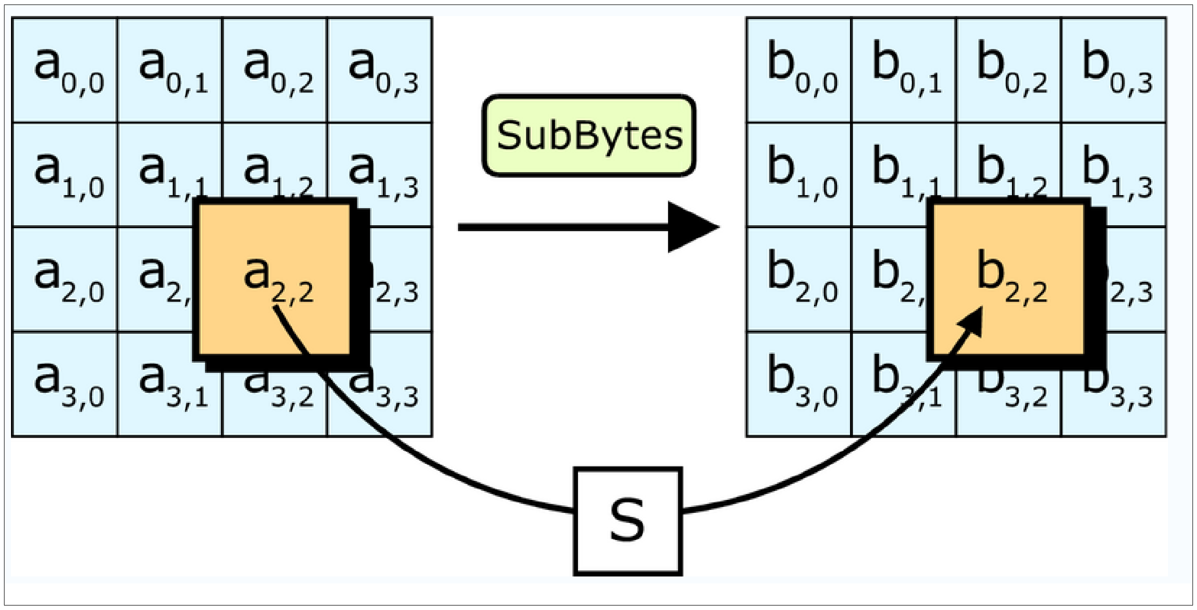

SubBytes
S
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 5. SubBytes operation

2. **ShiftRows**. In this step every row is shifted cyclically a certain number of steps. The number of steps changes among the rows. The operation can be better understood by examining the figure below:

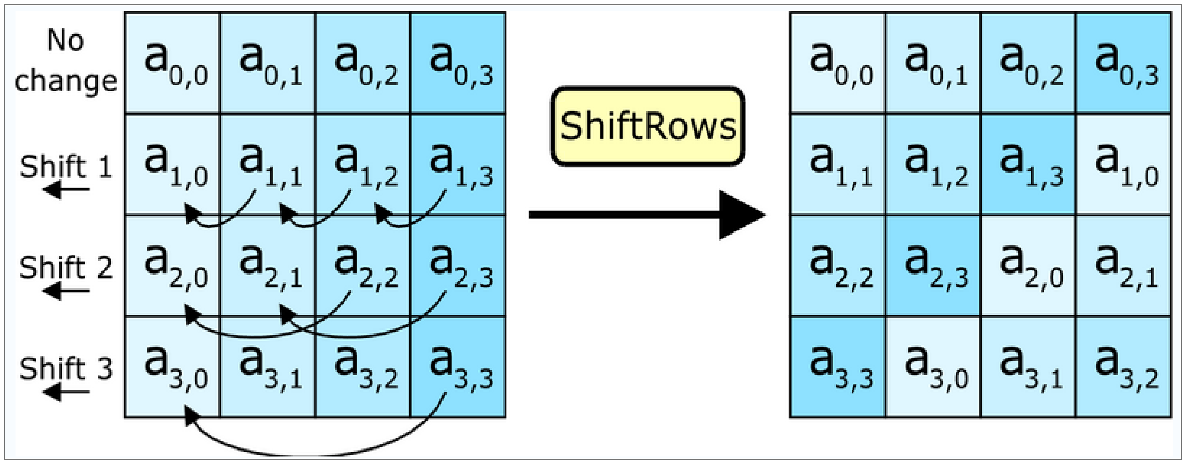

No change
Shift 1
Shift 2
Shift 3
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Figure 6. ShiftRows operation

3. **MixColumns**. Now the four bytes in every column are going to be passed thorough a linear transformation. This linear transformation consists of a multiplication by a fixed polynomial c(x).

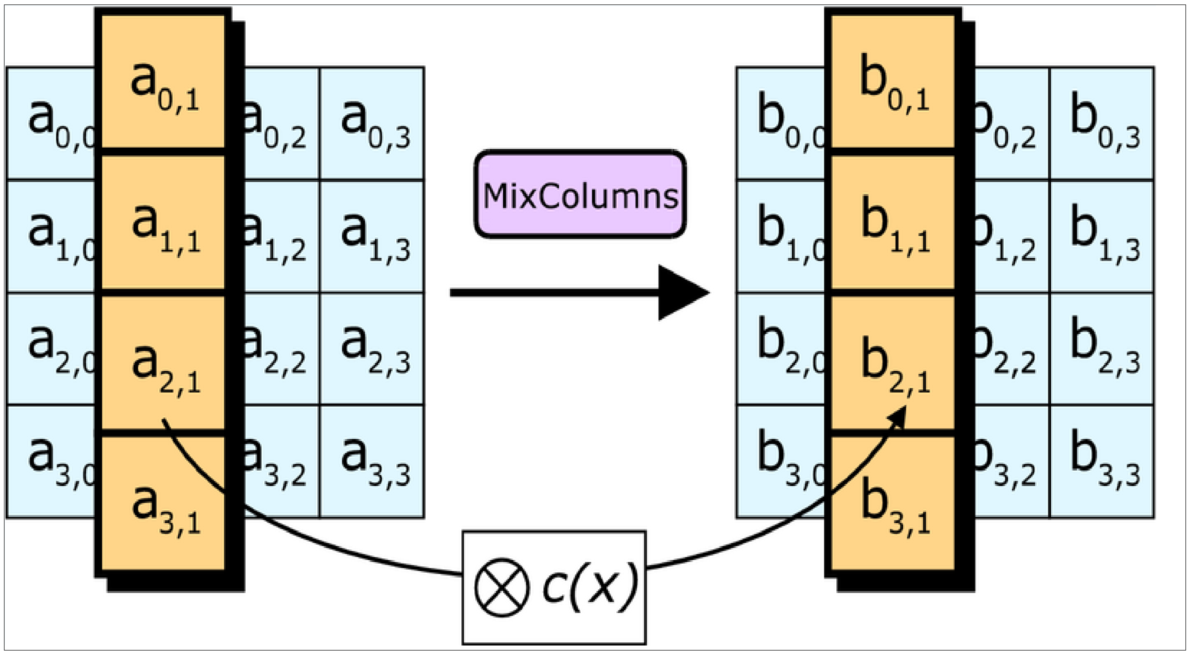

MixColumns
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$\otimes c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 7. Mix column operations

4. **AddRoundKey**. Each byte will be combined using a XOR (exclusive OR) operation with the round key. Each round key is obtained from the cipher key using a key schedule.

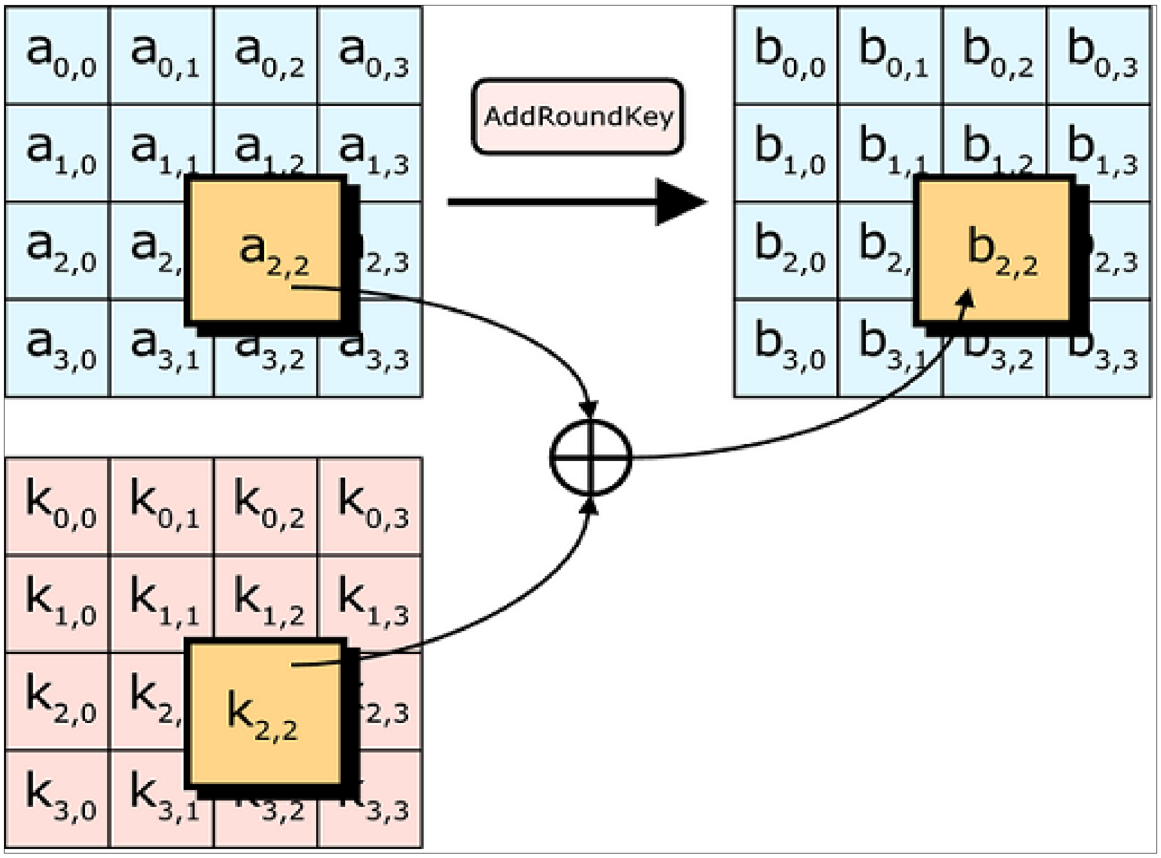


Figure 8. AddRoundKey operation

4.7 RSA (Rivest, Shamir, Adleman)

The RSA public key algorithm [28], [29] was created in 1978 by Rivest, Shamir, and Adleman, and it is currently the best known and most widely used cryptographic system. It is based on the difficulty of calculating factors of large numbers.

The best way to factor a number is to divide this number by 2, 3, … searching for an exact result which provides us with a factor of the number. If the original number is prime, that is, it can be divided only by 1 and itself. If, in addition to being prime, the number is large enough, the factorization process takes a really long time.

The RSA system creates keys using these steps:

1. Two large prime numbers are chosen, **P** and **Q** (each between 100 and 300 digits long).
2. Two numbers **n** and **m** are calculated as **n = P*Q** and **m= (P-1)*(Q-1)**.
3. Now, a new number we call **e** is selected such that there are factors in common with **m**.
4. The private key is calculated as **d = inverse (e) mod m**, where mod is the remainder of a division.

Once these steps are done, **n** is the public key and **d** is the private key. While **P**, **Q**, and **m** will be destroyed, **e** is publicized since it will be used later for encryption/decryption. The calculations of the keys are done in such a manner as to protect the private key. RSA is a computationally attractive function, since although the modular exponentiation is easy to perform, the inverse operation and the calculation of the roots modulo **m** are very difficult unless you know **e**, i.e. the private key).

RSA is widely used for public key systems. In part this has been because it is the fastest. Moreover, it presents all the advantages of an asymmetric system such as enabling digital signatures, but it is mainly used to implement confidentiality in symmetric systems, because they are faster than asymmetric ones. It can also be utilized in mixed systems to encrypt and send the symmetric key which will later be used to encrypt the communication. Some examples of this algorithm are presented in [30] and [31].

5 Feature Comparison between Skype and Minisip

In order to correctly compare these two VoIP approaches I begin by clearly describing each program.

5.1 Skype

5.1.1 Underlying Operating systems

One of the first to be noted which operating systems can be used under each of these programs. From the Skype website [21] we can easily see from the download choices which operating systems are supported. There are four different Skype versions depending on the OS. These are:

- Microsoft's Windows 2000 or XP,
- Apple Computer's Mac OS X,
- Linux, and
- Microsoft's Pocket PC

The requirements for each of them are described on this web page. For example, in order to use the Microsoft Windows version of the Skype software we need:

- a PC running Windows 2000 or XP,
- at least a 400 MHz processor,
- at least 128 MB RAM,
- a minimum of 15 MB free disk space,
- a Sound Card, speakers, and microphone, and
- an Internet connection of at least 33.6 kbps (either broadband or a dial-up)

The complete information about the requirements for each version can be found at [20].

5.1.1 Information Transfer

The main function of Skype is to make a simple voice call. If the called party also has a Skype client this call can be made without cost. These calls are encrypted - ensuring both security and privacy for the communication. Moreover, the user does not need to worry about neither the presence of a firewall or its configuration, nor about the routers or other network components. As Skype said says on its website, "it just works".

In addition to the calls to Skype clients, Skype also enables calls to regular telephone numbers, that is, the number of a PSTN (fixed or mobile) phone using their SkypeOut service. This provides connectivity to nearly any telephone in the world. It is important to note that the cost of the call does not depend on the location of the caller, but rather it depends on the location of the callee. For example, the cost of calling from London to London will be the same as calling from Russia to London. The tarrifs for making calls to PSTN numbers can be seen at [32].

Apart from the calls, Skype also offers instant messaging, enabling chatting with upto 48 other Skype users. Note that the maximum size of a message is only limited by the operating system. In general, this size will be between two and four Gigabytes. It is possible to send messages between different platforms - but for non-ASCII text there is no guarantee that the contents of the file will be meaningful.

5.1.3 Telephone Conference

Skype allows users to establish conferences between several users. With the current version of Skype it is possible to have conferences with upto five users. However, only the Skype client who initiated the conference can add new people to the conference. In fact, the Skype protocol handles the conferences by making this node the centralized node for the other participants in the conference. The scheme is shown in the following figure.

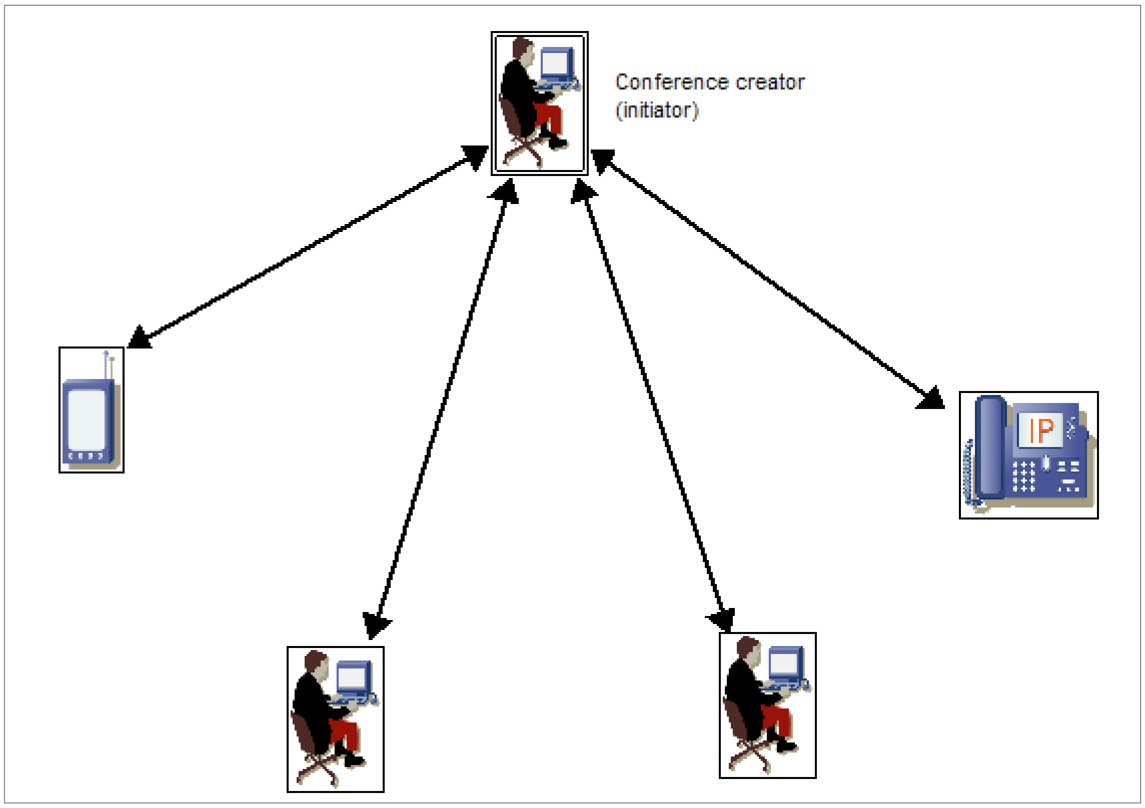


Figure 9. Conferences in Skype

Thus, the other users send to the conference creator their voice packets, where the audio streams are mixed and forwarded to the other users. Therefore we can see that the communication is not really peer to peer, but actually depends on one node and its resources in terms of bandwidth and computational power.

There are several aspects to be noted. First of all, the conferences can not only be between Skype clients, but may include users of the PSTN (including mobile networks), i.e., regular phones, and even with SIP phones in a close future. Of course, the calls will be billed separately: all based on where the callees are.

Secondly, conferences can include Skype clients running on different OS platforms. However, depending on the OS, this behavior may be different. For example, clients running on top of the Linux software are able to join a conference with other users as well as to see which users are taking part in it, but they cannot start the conference by themselves. For users using Skype running on a Pocket PC, the client can join conferences, but not start new ones. While, clients running Windows or Mac software can join and create conferences as well as control the users participating in these conferences.

5.1.4 Audio CODEC

Skype currently uses a broadband codec which permits the encoding of audio signals upto 16 kHz. Several studies indicate that the codec used by Skype is iLBC [33], [34]. An earlier study of the Skype protocol performed at Columbia University [63] indicated that the codec used by Skype could be iLBC, iSAC, or another unknown codec. However, the reality is that the codec could not be iLBC since it is actually a narrow band codec and also this codec fixes the size of the frame to one of the two possible sizes; nor iSAC because the range of output rates from this codec is from 10 to 32 kilobytes per second - while the transmission rate observed from Skype varies between 3 and 16 kilobytes per second. A study of the Skype protocol was performed in this thesis and details can be found in the Appendix A.

5.1.5 User Availability

Skype also has a Global User Directory consisting of a huge telephone list with all the Skype users [21]. This list can be used to search for other users (contacts) that we wish to communication with. It provides information such as the user's name, country of residence, birthday, or other data depending upon what the user decided to include in his login information.

In addition to searching for contacts, it is possible to add a given user to your own contact list and to send a message to this user (for example, asking if they are available for a call). In fact, Skype provides us a general indication of the state of the user - this is one of:

- offline
- online
- away
- busy
- not available
- do not disturb
- invisible

5.1.6 Information Security

Skype provides security for signaling and call contents. Skype uses two encryption algorithms: RSA [28], [29] and AES (Advanced Encryption Standard) [27]. Skype utilizes 1536 to 2048 bit RSA to negotiate symmetric AES keys at the beginning of the communication. Once the negotiation of the keys has been performed, the signaling and data messages are actively protected by encrypting them using the AES algorithm. The size of the keys used is 256 bits, so that the total number of possible keys is $1.1 * 10^{77}$. Moreover, users public keys are certified during the login process by one of the Skype servers [63].

Therefore, the information that every user sends, regardless of being a speech communication or an instant message, is protected. However, the user is not able to modify the parameters to provide a greater or a weaker level of security since the Skype protocol does not offer such a feature and it is a closed source protocol.

5.1.7 Other Features

Another Skype function is a robot to test your sound configuration. This robot helps you to have better quality in the calls and it is described at [21].

Skype also provides a link named "store" [21] where users can find information related to all the complementary products sold by the company. For example, it is possible to buy a handset or USB telephone.

Moreover, Skype offers users advanced services such as VoiceMail [35]. This service is available to registered Skype users who pay the additional fee charged for this service. This service allows users to create audio messages and send them as an e-mail to ones contacts. To send the messages, the VoiceMail service uses the "Standard Windows MAPI" [36] which is compatible with nearly all e-mail systems. The VoiceMail service subscription is 5 euros per 3 months, or 15 euros per year (plus 15% VAT in EU countries), or for free if you subscribe to the SkypeIn service. Although initially they advertised an automatic service to extend the contract for VoiceMail, Skype no longer provides this option. Moreover, if the user decides not to continue using Skype VoiceMail within the period of a month, Skype will return their money.

Another interesting advanced service, called SkypeIn, enables a user to register a Skype contact with a number of telephone which could be used to call Skype clients (without cost) or to call PSTN numbers or SIP numbers using the SkypeOut service (for which the caller would pay per call charges).

In addition to these services, Skype has announced it is working to enable users to send GSM short messages. Even more important is that this will be usable to/from the third generation of mobile telephones [37].

Now that the main features of Skype have been noted, I next give a similar description to Minisip [38].

5.2 Minisip

5.2.1 Operation systems

The Minisip program is currently only available on top of the Linux OS. It consists of a set of libraries to be downloaded. Binary packages exist for installation on the GNU Linux Debian, Red Hat Linux 9, Familiar Linux, or Mandrake cooker. However, it is noted on the website that some of these versions are not stable. It can be assumed that in the near future that a Windows software will be available. Unlike Skype, the full source code is available from the web site for those who wish to compile it from source, extend it, … .

There is no information about the minimum hardware requirements for proper operation of the software. The web site only indicates that the appropriate versions of the libraries are needed. However, it is necessary to have a full-duplex audio subsystem.

5.2.2 Exchange of information

Minisip allows user to make calls around the world without cost when the communication is between Minisip clients or between Minisip and another SIP client elsewhere on the Iiternet. Communication between two Minisip clients is encrypted to ensure the privacy of this communication. However if one of the users is not using Minisip (for example, a regular telephone via a voice gateway, or another SIP client), then encryption might not be available. This depends on the voice gateway and the other SIP client.

In addition to audio communication, Minisip also incorporates the ability to send messages, just as Skype; but in addition also includes support for video conferences. Additional functionality is planned, such as exploiting speech-to-text and text-to-speech to dramatically decrease the bandwidth necessary for a given quality, integration with local media players (as shown by Inmaculada Ranges Vacas [39] in her thesis), … .

In addition to single party telephone calls, Minisip also supports a push-to-talk mechanism; for details see Florian Maurer's project report [40]. The push to talk mechanism enables the establishment of group communication (such as conferences), but adds floor control. This mechanism allows the development of a wide variety of communication facilities such as:

- Instant Personal Talk: direct communication between users.
- Chat Group Talk: There are two different modalities for this kind of communication:
  - Open chat group. In this version of Chat Group Talk, each participant can invite new users to join the communication.
  - Restricted Chat Group. In this version, the creator of the group will be allowed to add or remove users to/from the group. Moreover, only those users within the contact list of the creator could join the communication.
- Instant Group Talk. Consist of the establishment of a push-to-talk session with a user of your contact list. Each user upon initiating communication can add a new contact to such a list.
- Ad hoc Instant Group Talk. This service allows direct communication between users without this infrastructure.
- Instant Personal Alert. This facility consist of sending to the user a message indicating to him that he must contact the sender of such a message.

5.2.3 Conferences

Conferences without floor control are just recently supported by Minisip [75]. This required either the design of a bridging/mixing mechanism for the multiple audio streams or a full mesh. The implementors chose the later and added acoustic echo suppression. Using the spatial audio facilities which Ignacio Sanchez Pardo introduced into Minisip is possible to listen to multiple audio streams at the same time - however, there is not yet a convenient control to indicate to which stream your outgoing audio should be directed, but in a conference it naturally goes to all but then source.

5.2.4 Videoconference

Minisip includes videoconference in order to provide better communication between the users of the system. This enables communication to be much more personal and efficient than audio only calls, since we can appreciate both voices and facial features (and perhaps even gestures).

Videoconferencing is useful between two friends, in fields such as remote education, internal enterprise communication, communication between different companies, remote consultation with doctors, etc. An important application area is "I see what you see" to enable a remote advisor to help a user. Typically videoconferencing implies expensive equipment as well as high costs of use, however, Minisip provides this service on-line over internet and only requires a web camera (whose cost is today rather low).

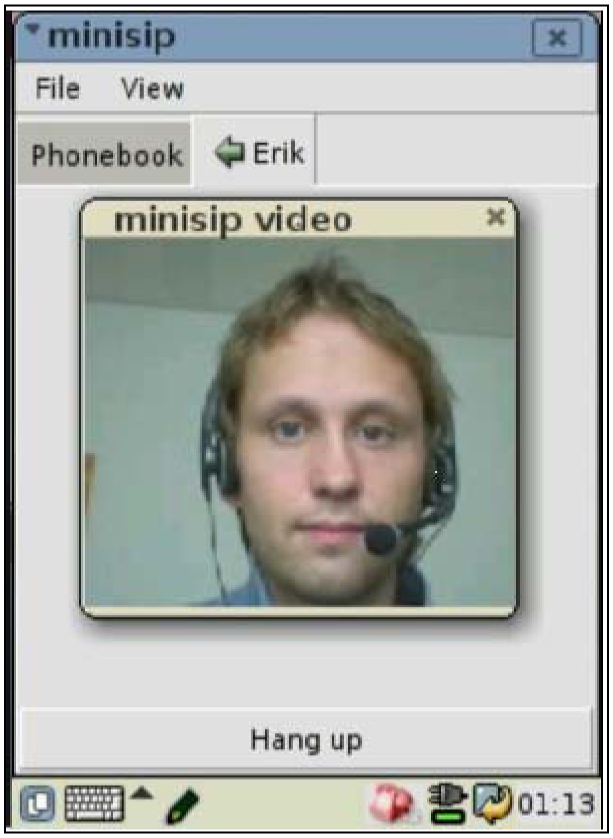


Figure 10. Videoconference in Minisip

5.2.5 Audio CODEC

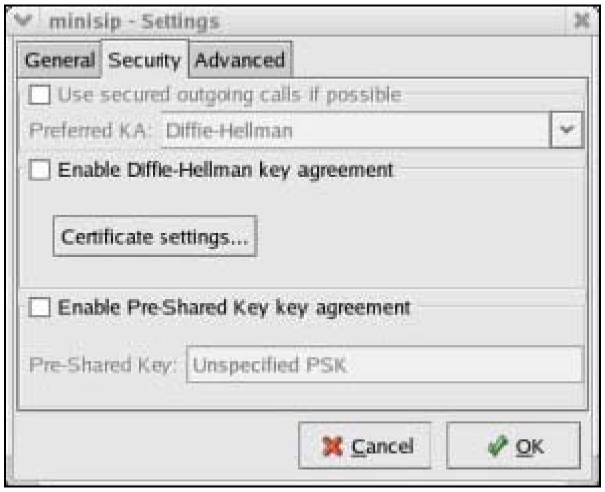
In the current version of Minisip there are many possible codecs: PCMu, iLBC, linear encoding with 16 bits and Stereo. Unfortunately, only PCMu and iLBC are available from the Graphic User Interface (GUI). Both codecs are narrowband codecs and assume that frequencies over 8 kHz will be filtered or removed. However, if the users have a broadband connection and use 16 bit linear stereo coding at up to 48K samples per second, the resulting quality can be even better than CD quality.

5.2.6 User Availability

Since Minisip is based upon SIP, all contacts can be kept at servers (similar to Skype) or the user can keep a local list of contacts at the device running Minisip. Unlike Skype, we do not need to send a message to the other party to add them to our list of contacts. Unfortunately, no display of the status of your contacts as is available in the Skype GUI. Of course since presence information is available via SIMPLE [41], the Minisip GUI could be extended to display it. In fact, the code is already in the GUI to enable this [77].

5.2.7 Information security

Minisip also supports encryption, but unlike Skype a Minisip user can select different levels of security via the GUI. Of course these different levels of security will change the traffic actually sent and hence the efficiency of the communication. Measurement of this will be presented in section 7.2. The interface to control these settings is shown below:



☐ Use secured outgoing calls if possible

Preferred KA: Diffie-Hellman

☐ Enable Diffie-Hellman key agreement

☐ Enable Pre-Shared Key key agreement

Pre-Shared Key: Unspecified PSK

Figure 11. Dialog to change the security properties

As shown above the level of security could be one of the following:

- enable secure outgoing calls **if** it is possible
- enable Diffie-Hellman [42] key agreement
- enable pre-shared key agreement

In the case of selecting Diffie-Hellman key agreement it is possible to change the certificate settings in the following dialog in which you can select: the private key to be used, the certificate to be used, and the authority of certification (CA) to be used.

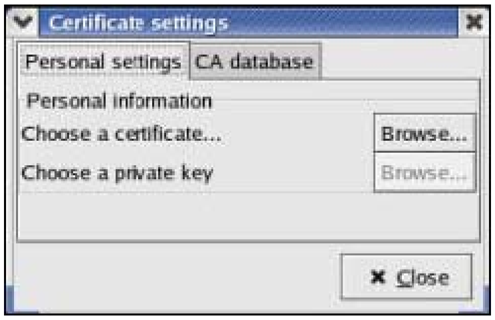


Figure 12. Advanced security options.

5.3 Functional Comparison of Skype and Minisip

This section presents a functional comparison of each of these two VoIP applications. An obvious difference is that Skype is a commercial product while Minisip is a research prototype.

Skype is available for several operating systems (Windows, Macintosh, Linux or Pocket PC) while Minisip is currently only available for Linux machines (but is being ported to Windows by Andreas Ångström in his thesis [43]). More importantly, the released Skype versions are stable and have clearly stated hardware and software requirements, while some of the Minisip software versions are not stable and there is not yet a clear statement of hardware and software requirements [38]. Not having a Windows version puts Minisip at a great disadvantage; hence the great need to port the program, as this represents the largest potential user group. The result will be the ability to establish secure calls between users running on top of different OSs.

With respect to basic communications both Skype and Minisip allow making telephone calls as well as sending messages. The possibility of chatting exists in both cases, but in the case of Minisip chatting is limited to ten clients, two per position, when Skype allows a communication with up to 48 users. Using either program it is possible to call PSTN (including mobile numbers) via a PSTN gateway. However, here Minisip has a dramatic advantage over Skype since you could utilize any SIP compatible PSTN gateway (hence potentially have lower costs); while if you use Skype you can only use their gateways.

It must be noted that Skype supports audio conferencing without using high bandwidth, i.e.the bandwidth used is between 3 and 16 KBps; while the bandwidth for Minisip depends on the users choice of codec. Since a voice PSTN call provides a bandwidth of ~8 kilobytes/s (in the best conditions), the communication with Skype is limited to this bandwidth. In Skype to Skype communications, a broadband codec is used - which codec is used can not be selected by the user; while Minisip supports both narrowband and broadband codecs as selected by the user. Depending on the codec selected in Minisip the comparison of the perceived voice quality should be better or worse than in Skype. If the codec used in Minisip is a narrowband codec, then the perceived voice quality of Skype is better (as expected).

Typically, a young adult perceives the range of frequencies from 20 to 20,000 Hz. Consequently, the sampling frequency of CD audio is chosen to be 44.1 KHz, which is more than double that of the highest frequency perceivable by most humans. Moreover, it is known that the energy in speech signals is concentrated in the low frequencies. In fact, there are studies which show that a speech signal can be filtered with a bandwidth of 10 KHz without affecting its perception [44]. Thus the quality provided by Skype is better than Minisip when it uses a narrowband, since Skype keeps all the necessary frequencies, while when Minisip uses a narrowband codec it filters out some of these frequencies. However, if the encoding used in Minisip is 16 bit per sample linear encoding, then the sampling frequency can be increased up to 48 K samples per second (in mono or stereo) - providing **much** better voice quality (although for most people the difference will not be noticed for speech - but can be noticed if the audio content is music). Consequently, the selection of the codec and its parameters will affect the quality of a telephone conversation; this can be evaluated by enabling users to rate the quality of their call when it ends.

There is a difference in how conferences are controlled, since in Skype conferences the outgoing packets are sent to the participants in the group by the party who started the call; while in Minisip there is either (a) explicit floor control using a push-to-talk mechanism, that is, the same mechanism used in walkie-talkie communications and the resulting outgoing packet is sent directly to all the participants - note that here there will only be one source sending at a time or (b) a full mesh – where each source sends to all the others. Therefore, in Skype is essential that the user starting the communication has the best computational and network resources; while in Minisip there no such a limitation.

Additionally, the push-to-talk floor control not only covers the conference service provided by Skype, but it introduces new advantages since it allows the user to classify the conferences depending on the users talking with (different conference modes permitting a better division of them) as well a taking part in several conferences simultaneously as was shown in [40]. Chat Group Talk provides the equivalent conferencing as Skype, since it allows communication with your current contacts and with new contacts. In fact the conferences that Skype can establish are really equivalent to those allowed via the Restricted Chat Group. The Open Chat Group service allows a greater freedom than thus, as it does not centralize the ability to incorporate new users in the conference as happens in Skype.

Additionally, the push-to-talk mechanism can setup Ad-Hoc communication in which Minisip clients could communicate without having any infrastructure. Hence allowing peer-to-peer communication!

Recently Minisip has incorporated a new service named “spatial audio for the mobile user” as a result of Ignacio Sanchez Pardo’s thesis [45]. This service allows the user to locate every incoming call in a virtual spatial position - this provides immediate knowledge to the user who can now separate the different communication streams when listening, based on the perceived direction from which the sound arrives.

Beyond the capacity to make voice calls, Minisip also provides a videoconference service. This service provides many possibilities to the user. And recently, it is possible to create a video conference with more than two users using a full mesh. Therefore, this has improved the service provided by other programs such as MSN messenger [46], CuSeeMe [47], or Netmeeting [48] which do not support videoconferencing for more than two users simultaneously.

Although both programs support instant messaging, Minisip client only supports messages with a number of users not higher than 10 Minisip users; whereas Skype supports a chat service with up to 48 users. In addition, Skype can send whole files as messages; while

Minisip does not yet have this functionality.

With regard to the security provided by both systems, Minisip enables the user to choose the desired level of security. However, I personally consider the security provided by Skype (using AES [27]) to be sufficient, thus I don't consider this functionality to be an important advantage for a normal user.

The final features which we will consider ("other features") are Voicemail and communications involving PSTN telephony. As VoiceMail is currently being introduced in both programs there is no clear advantage of Skype versus Minisip. However, significant change in functionality is the recent addition of SkypeIn - as until now a PSTN user could not call a Skype user (even though the reverse was possible), of course both are services which you must pay for. In the case of Minisip you do not have these problems because Minisip supports SIP telephony, thus you can use any SIP compatible PSTN gateway and there are numerous ways to register a telephone number to cause a call to it to be gatewayed to your SIP client. Providers of these kinds of service are companies such as: Digisip [49] or HotSIP [50].

6 User Interface Comparison

The structure of this comparison will be similar to the functional comparison above.

6.1 Skype

6.1.1 User Support

One of the most important aspects from the user's point of view, without any doubt, is the tools, documents, or sources that help us when we have some problem with the program. Here, Skype provides (through its website [21]) a complete set of solutions for the different sorts of problems that most users will encounter.

They provide a finder which finds the answers to user's questions easily and quickly - avoiding the need to search the website. In addition to the finder, they provide a number of other aids:

- Links to Frequently Asked Questions (FAQ).
- A database of answers classified depending on where the problem appears such us:
  - Use of SkypeOut,
  - File transfer,
  - General questions,
  - Etc.
- A tutorial which introduces and explains to the user how to use the program,

In addition to these sources of immediate answers, Skype has two additional ways of informing customers. The first of these is a list of news related to the use of the program, this currently mainly focuses on the SkypeOut service. This list is updated periodically and describes the new options introduced in the program - as well as known problems. The second source of information consists of a number of forums, currently fourteen forums, where the user can send questions or look for answers to previous questions - this requires selecting the appropriate forum. Questions can be answered by other participants in the forum or by the developers of the Skype software.

All of this information is the result of the collective effort and knowledge of the many workers employed by Skype. The extensive information makes the development of new applications easier and faster. It seems that Skype is paying attention to their customers. A consequence is the rapid development of a relatively new product that is now widely used. Of course this depends on the Skype partners and owners providing capital to maintain this structure, as well as supporting the ongoing development of new services.

6.1.1 Skype Support for Developers

Skype has not limited the forums to answering problems related to the operation, installation, or other aspects for the basic user, but has opened the door to anyone interested in developing their own applications building on one of the Skype products. These questions are answered by the personal in charge of developing Skype. Skype is conscious of the enormous potential for development outside the company and tries to make use of these efforts to improve its product and, as a result of it, to obtain greater economic benefits.

To support this, Skype segments the people interested in the development into 3 levels:

1. The first level, is oriented to all those interested in going beyond simply using the product and to develop new applications. At this level, Skype will help interested users by answering their questions without cost. One of the great opportunities introduced to this level of developers is the Skype Application Programming Interface (API). This API allows other applications to utilize the underlying Skype functions (however, currently this is limited to Windows and Linux applications running on a PC). As an example, I have evaluated the operation of this API by developing a console application. This is described in appendix B.
2. The second level of the support requires establishing a relation between the developer and Skype. In this case, the developer has to pay an annual fee of 1000 euros. In addition to the previous level it provides the following additional advantages:
    - Access to the upcoming Skype versions for testing;
    - Access to prototypes and examples of Skype services; and
    - Increased potential access to the Skype developers.
3. Finally, a third level is reserved for invited corporations and important partners. Obviously this level includes the features of the two previous levels, along with:
    - Direct access to Skype engineering team members;
    - Ability to provide input into Skype's development efforts and priorities; and
    - Strategic development support.

6.1.3 Software and Service Installation

The installation of the Skype product depends on the software selected:

- For the Windows software, you simply download to your computer the installation software and to follow the steps as indicated.
- For the Pocket PC software: Either (A) you are required to have ActiveSync [51] in your Personal Digital Assistant (PDA), but aside from this the process is quite similar to installing the Windows version of the software. Once you have downloaded the installer, you simply open the file, when you accept the displayed conditions, the Active Application Manager will pop up to do the rest of the work or (B) you can download a CAB file directly to your PDA.
- For the Macintosh software version, it is necessary to mount the downloaded disk image doing a double click on the "dmg" file and drag the Skype application from mounted disk image volume to Applications folder on your main volume as it is described on the Skype web page [21].
- Skype is available for several Linux platforms, specifically:
    - SuSe 9 and newer versions,
    - Mandrake cooker 10.1 and newer versions,
    - Fedora Core 3,
    - Debian,
    - RedHat.

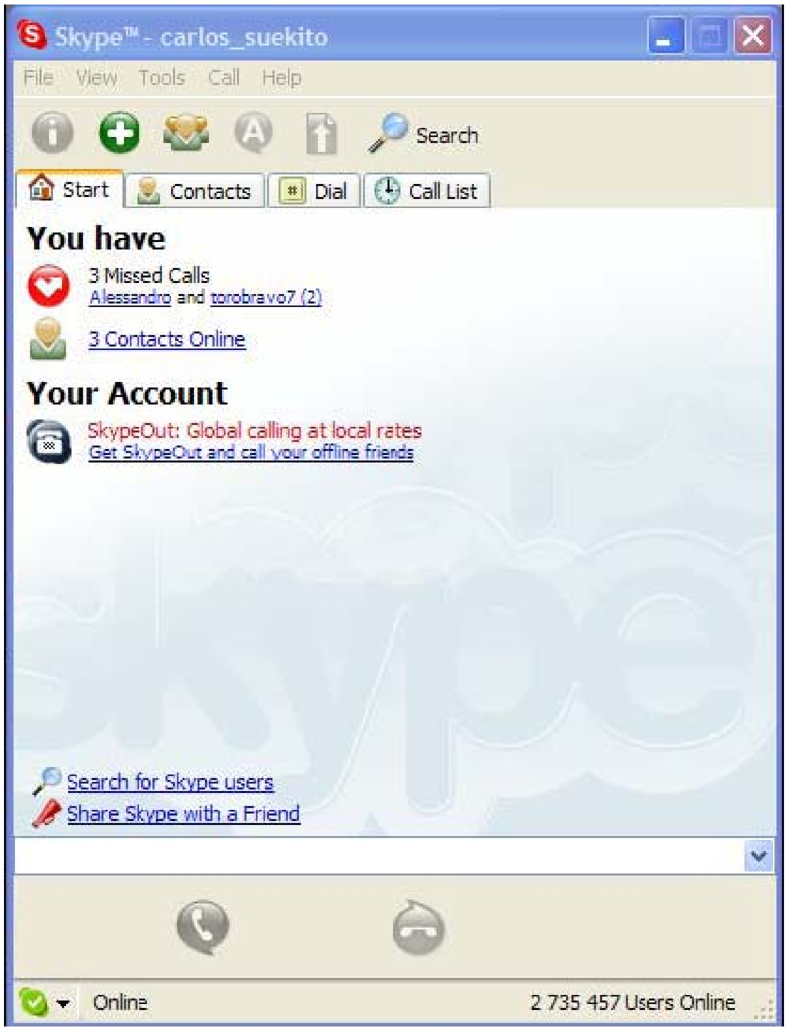
Depending on the Linux version there are three different installation methods:

- RPM version: as a superuser simply enter the command "`rpm -U skype-version.rpm`", where skype-version.rpm is the name of the file downloaded.
- Tar.bz2 version: simply enter the command: "`tar xjvf skype-version.tar.bz2`", where skype-version.tar.bz2 is the name of the file downloaded; in this case it is not necessary to be the superuser. The Skype software is unpacked to the current directory.
- Debian package: download the package and follow the instructions.

6.1.1 Using Skype

In this section we will briefly comment how to interact with the program. The primary focus is how to use the different options described earlier. Rather than describe each of the different versions of the software (since the differences are not particularly important), I will focus on the current version for Windows (2000 or XP).

The main Skype interface is shown in the following figure:



You have

3 Missed Calls

Alessandro and torobravo7 (2)

3 Contacts Online

Your Account

SkypeOut: Global calling at local rates

Get SkypeOut and call your offline friends

Search for Skype users

Share Skype with a Friend

Figure 13. Skype Interface for Windows XP or 2000 software.

The first tab available inside this Graphical User Interface (GUI) shows a summary of the information related to the current Skype contact, e.g. the number of contacts online in that moment, calls unanswered and the Skype users who made them, or a message indicating that there is some user who wants to establish a communication session with the user, but whose request has not (yet) been accepted.

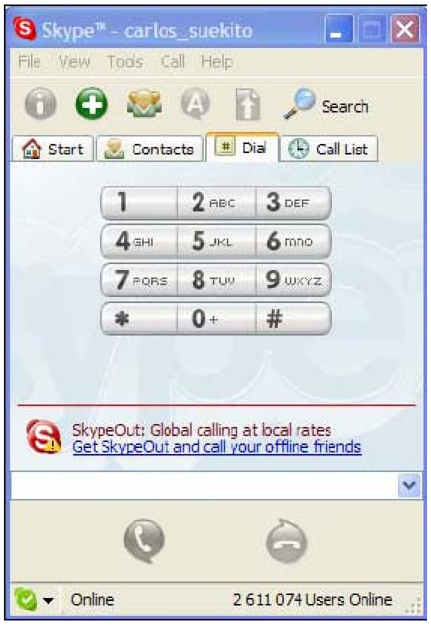
As noted earlier, Skype offers the possibility of making calls to telephones attached to the PSTN (or mobile networks). In order to be able to make these calls the user should have a positive credit with Skype, since the SkypeOut service is not free. If the user has bought credits to use with SkypeOut, this will be indicated (as show in the figure above).

If a Skype user is interested in knowing who his online contacts are, they simply selecting the second tab shown in the dialog box - which is labelled “Contacts”. One can also access this same information from inside the main menu.

Once the list of contact is shown, the Skype user might be interested in setting up a communication session with one or more of them (however, no more than five can be contacted currently; as noted earlier). The easiest way to establish a call is to select the appropriate user or users to be called and click over them. Once these contacts are selected pressing the “call” button, represented by a red button with a vertical telephone, the call will be initiated.

If the Skype user wants to send an instant message instead of making a call to the selected contacts, the user simply opens a new menu with a right mouse click. From this new menu, you simply select the appropriate option to send a message.

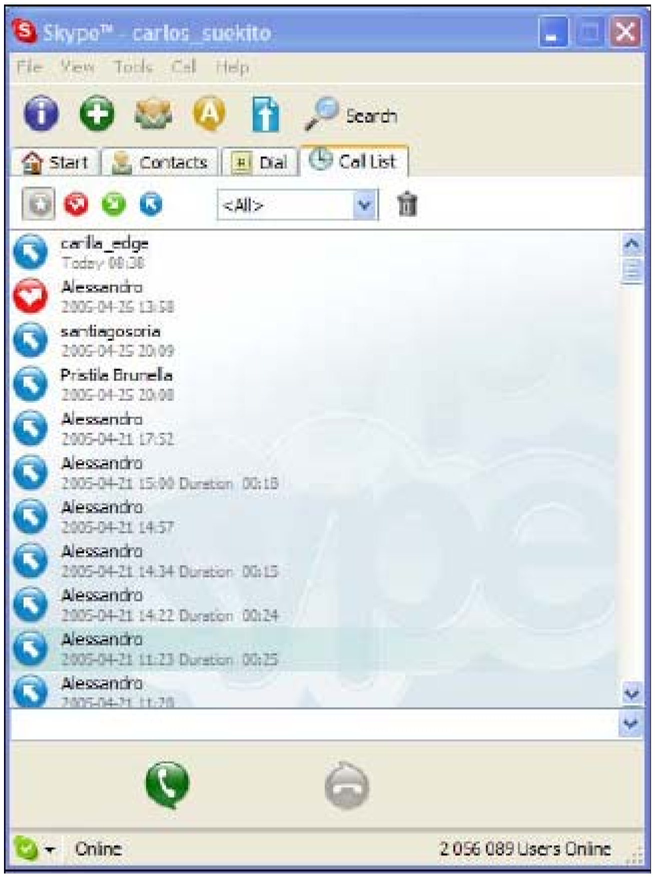
If the user wishes to make a call to a PSTN user whose number is not yet in the list of contacts, they can do so by using the “dial” tab. Invoking this “dial” tab displays a keypad which can be used to enter the number of the user to be called. Once the number has been dialed the “call” button has to be pressed and the call starts.



SkypeOut: Global calling at local rates
Get SkypeOut and call your offline friends
2 611 074 Users Online

Figure 14. Keypad in Skype

The last of the tabs presented in the main Skype window displays a list of calls sent and received indicating the date, the time, and the client which the call was from. An example is shown in the figure below:



carlla_edge
Today 08:38
Alessandro
2005-04-25 13:58
santiagosoria
2005-04-25 20:09
Pristile Brunella
2005-04-25 20:08
Alessandro
2005-04-21 17:52
Alessandro
2005-04-21 15:00 Duration 00:18
Alessandro
2005-04-21 14:57
Alessandro
2005-04-21 14:34 Duration 00:15
Alessandro
2005-04-21 14:22 Duration 00:24
Alessandro
2005-04-21 11:23 Duration 00:25
Alessandro
2005-04-21 11:28

Figure 15. Register of calls

Below I will explain the main buttons presented via the Skype GUI (although many of them perform functions which have been previously described). The Skype user often has different ways to do the same action, this redundancy increases the usability of the program.

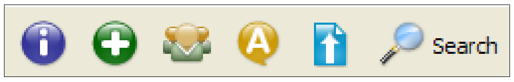


Figure 16. Skype Push-buttons

The first button shown in Figure 16, which looks like the letter "i" in a purple circle, allows the user to retrieve his or her user profile or the profile of a selected user, i.e. it retrieves the data available in the (Skype) Global User Directory.

The second button shown in Figure 16, which looks like a "plus-sign" in a green circle, allows the user to add a new contact to its list of contacts. After pressing this button, a new dialog will be opened so that the Skype name of the contact can be added. The next figure shows this dialog:

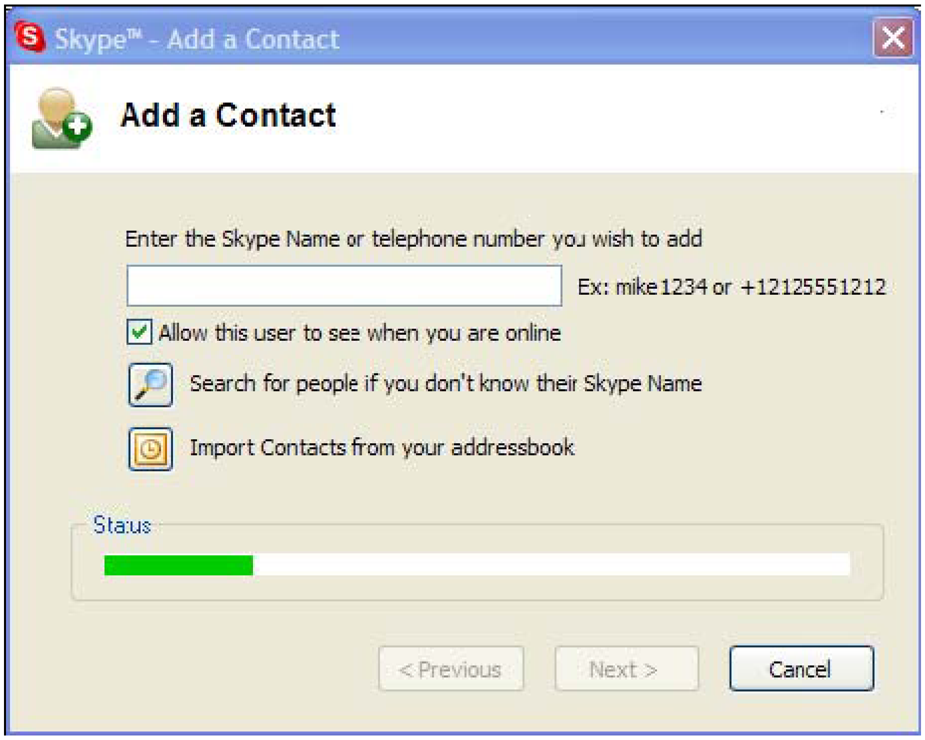

Skype™ - Add a Contact
Add a Contact
Enter the Skype Name or telephone number you wish to add
Ex: mike1234 or +12125551212
Allow this user to see when you are online
Search for people if you don't know their Skype Name
Import Contacts from your addressbook
Status
< Previous
Next >
Cancel

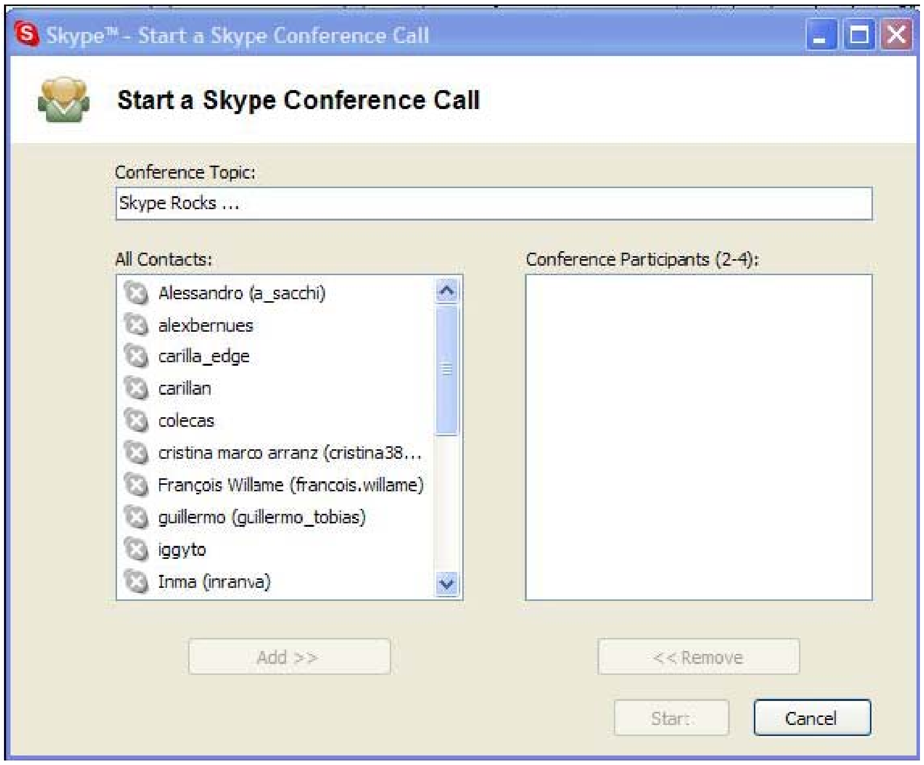
Figure 17. Add a Contact dialog

The third button (which is an icon of a person) and the fourth button (which looks like the letter "A" in a yellow circle) allow the user to establish a session or to send messages (as explained previously). These and other functions are also available via the main menu.

If the user wants to setup a conference using the dialog shown in Figure 18, it will be necessary to add each of the other parties one by one - by selecting and pressing the "Add" button. Once this is done, the Skype contacts are introduced to the conference list. As noted earlier this is limited to 4 users (for a total of 5 participants).

The dialog to establish a chat session is shown in the Figure 19. As can be seen, it is quite similar to other dialogs which are used to establish chat session in MSN messenger, NetMeeting, etc. The dialog consists of one editlist in which the received messages are displayed and a second editlist is used to introduce messages to be sent. It is possible to communicate with upto 48 users in a single session.

When pushing the fifth button (which looks like a page icon with a vertical), a new dialog is opened to allow the user to select the file they want to send. Once the file has been selected, the dialog displayed in Figure 20 is used to select the contact whom you wish to send it.


Skype™ - Start a Skype Conference Call

Start a Skype Conference Call

Conference Topic: Skype Rocks ...

All Contacts:
- Alessandro (a_sacchi)
- alexbernues
- carilla_edge
- carillan
- colecas
- cristina marco arranz (cristina38...
- François Willame (francois.willame)
- guillermo (guillermo_tobias)
- iggyto
- Inma (inranva)

Conference Participants (2-4):

Figure 18. Start a Skype Conference Call dialog

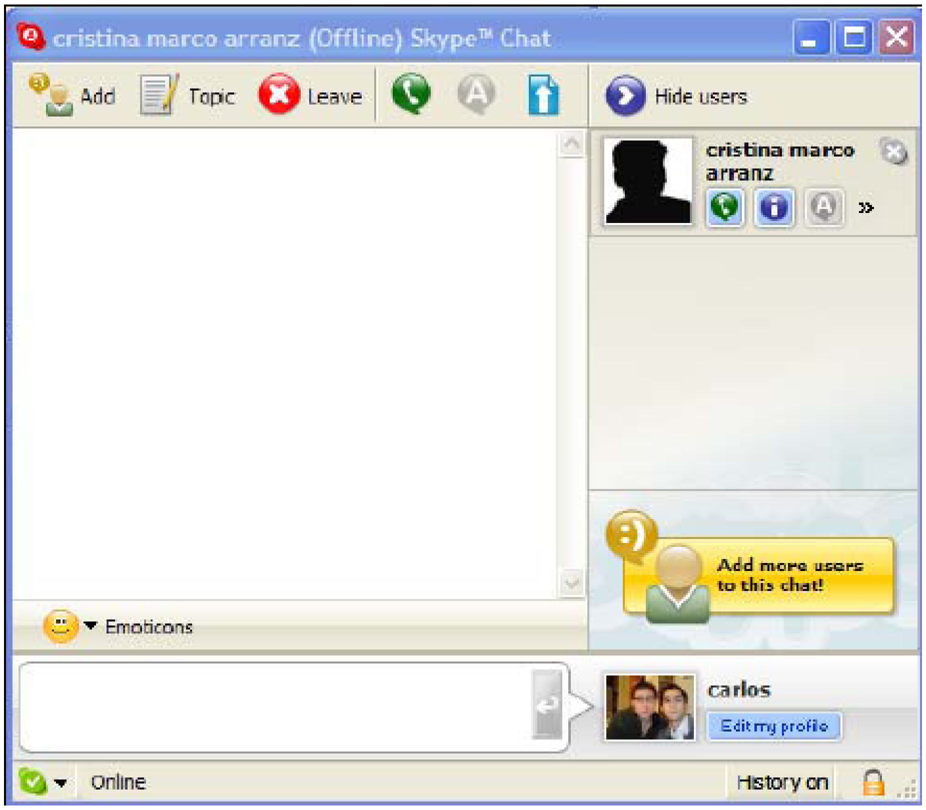
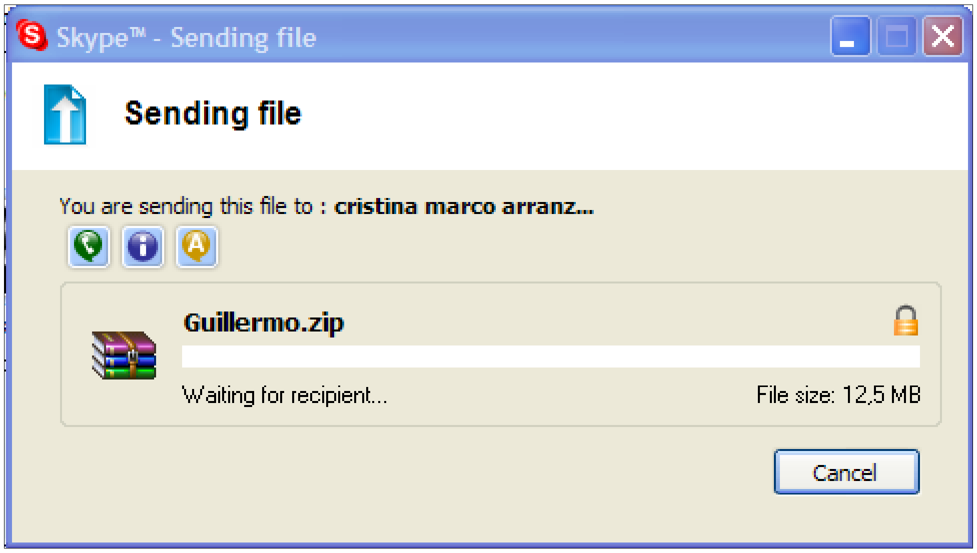

cristina marco arranz (Offline) Skype™ Chat

Figure 19. Skype Chat dialog



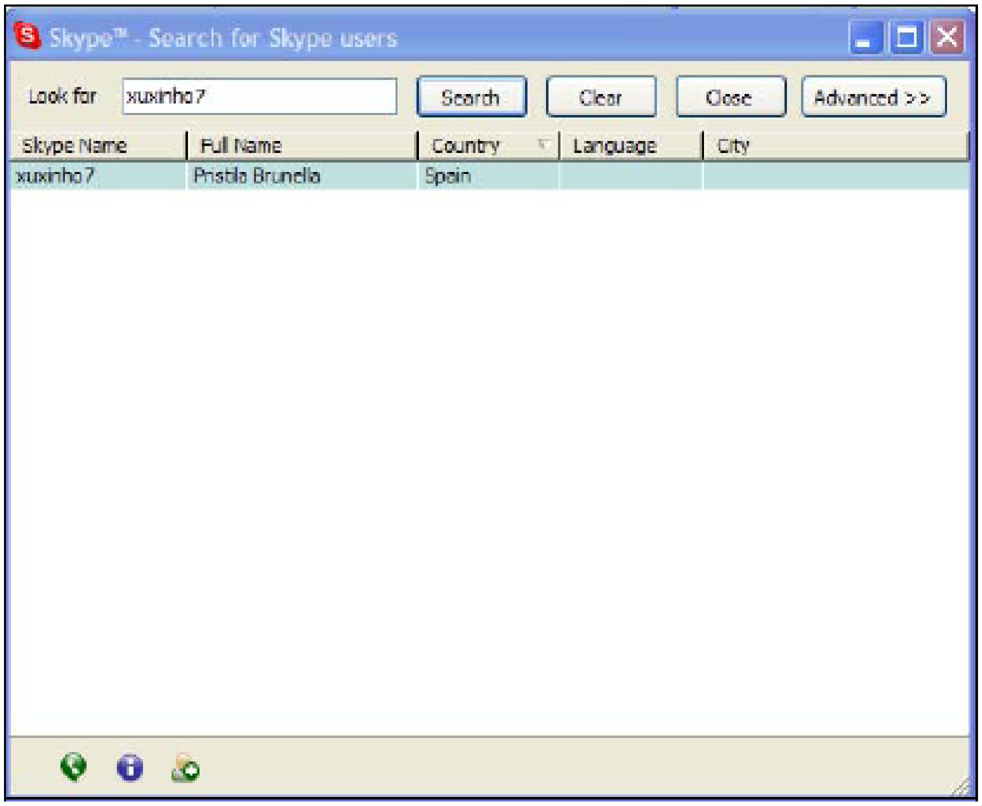
You are sending this file to : **cristina marco arranz...**

Guillermo.zip

Waiting for recipient... File size: 12,5 MB

Figure 20. Sending file dialog

Finally, the last button (which looks like a magnifying glass icon) is used to search for Skype contacts that are not in our own contact list. Once the button has been pressed, a new dialog is displayed to enter the name of the Skype contact, pushing the search button will start the actual search:



Look for: xuxinho7

Skype Name	Full Name	Country	Language	City
xuxinho7	Pristila Brunella	Spain		

Figure 21. Search for Skype users dialog

In addition to these functions, Skype provides additional functions via its main menu, for example: blocking users (to avoid communication with them), sending your contacts to another Skype client, or checking for an updated version of your Skype client.

6.2 Minisip

6.2.1 User Support

Minisip also has a website [38] which provides an introduction to Minisip. It is also provides information as to what operating systems are supported, as well as a list of publications about Minisip. The user will also find a link to download either pre-built binary releases along with the different libraries needed by the program and a brief explanation of which versions of the packages must be download or they can download the complete source code. Additional information related to installing Minisip will be explained below. In addition, the user can consult the master theses written by several KTH students.

One of the best features of Minisip support is that the source code is open. This allows other people to not only use this program and to work with it, find and fix bugs, but to add the services and features which they want. There are two e-mail lists, one to help users and the other developers. These lists facilitate contact between the developers and users. Information about joining these mailing lists is provided from the web site.

6.2.2 Extending and Building upon Minisip

Unlike Skype, Minisip is not a commercial product, but rather an experimental prototype for exploring issues in communications, security, etc. Additionally, since the source code for Minisip is available and the code has very liberal license terms (LGPL [52]) it can be studied and even extended by others. Because of this there is not a similar hierarchy for support as for Skype, but rather it is up to users and developers to extend it as they wish.

6.2.3 Installation

The installation of Minisip depends on which version of the software is to be installed (although as noted earlier the number of versions is fewer), specifically:

- The Windows version of the software, is not yet available
- There are several Linux versions available:

• Debian GNU/Linux
• Mandrake cooker 10.2 (unstable)
• Redhat Linux 9
• Familiar Linux

- For the Debian version, it is possible to access the binary packages in an APT [53] repository with:

```
deb http://www.minisip.org/debian binary
```

• Unfortunately, the version is only tested for the Intel x86 ("i386") architecture and is know to be unstable

- For Redhat 9, the binary packages can be downloaded directly from the available link, but it is also possible to download it from an APT-RPM repository:

```
rpm http://www.minisip.org redhat stable
```

• On RedHat systems an environment variable must be changed since there is an incompatibility with the threading library. To do that:

```
$ export LD_PRELOAD="/lib/libc.so.6 /lib/libpthread-0.10.so"
```

- For Familiar Linux the version is also unstable, but its binary packages can be accessed from:

```
src minisip http://www.minisip.org/ipkg/release/
```

- Binary packages are also available for Mandrake cooker in 10.2 version (unstable)

Once the binary packages have been downloaded, they can be built. The procedure to build each of the packages is shown at the web site. The build process to build the packages libmutil, libmnetutil, libmsip, and libmikey is the same:

- In most cases:

```
$ ./configure
$ make
```

- Then install:

```
# make install
```

- On many systems /usr/local/lib should be listed in /etc/ld.so.conf. If not, add it:

```
# echo '/usr/local/lib' >> /etc/ld.so.conf
# ldconfig
```

Once the above libraries are installed you can build Minisip:

```
$ cd minisip
$ ./configure
$ make
```

The executable is in a subfolder called Minisip and the binary package will be installed executing the command "`make install`". It must be noted when Minisip is started the first time a default configuration folder ".minisip.conf" is created

As note above on RedHat systems it is necessary to say:

```
$ export LD_PRELOAD="/lib/libc.so.6 /lib/libpthread-0.10.so"
$ minisip
```

due to the incompatible version of pthread that is used by default on Redhat 9.

For the Pocket PC software running in Familiar Linux, the procedure is mostly the same. In this case, there is not a dependency of the computer. It is just an executable to be downloaded to your computer.

6.2.4 Minisip User Interface

Minisip has several possible user interface faces, ranging from a textual interface to a GTK GUI. Here I will describe the operation of Minisip via the GTK GUI. The main window of the Minisip GUI is show below:

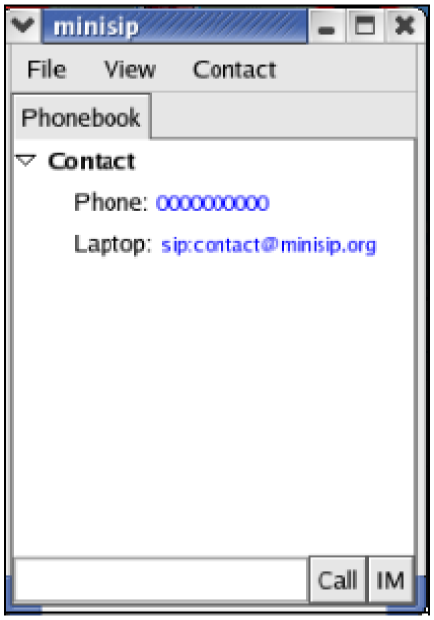


Figure 22. Main Window in the Minisip GUI.

The main window is composed of a main menu and only one tab. This tab shows the contacts of the current user, indicating their phone number, laptop URL, mobile phone number, etc. (depending on the contact information that has been added for this user). The figure above shows the window corresponding to when the program is first launched (i.e., the user does not yet have any contacts). This window shows a template for a URI (e.g., address) by default, so that the user can add new contacts in this form. It does not display a valid phone number least the user can choose it. Hence the telephone number is initialized to 00000000000.

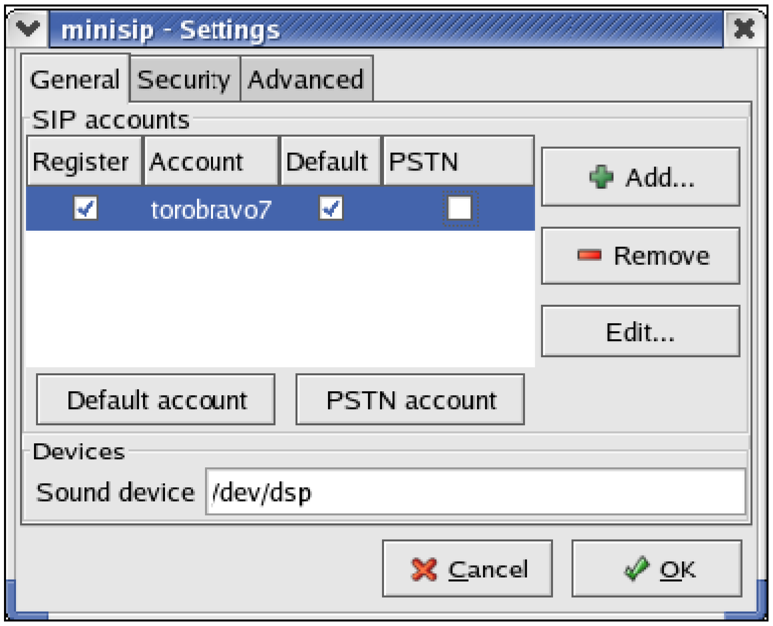
Using Minisip the user can make calls to Minisip, SIP, or PSTN contacts. For this reason the graphical interface incorporates an editlist in which the user will introduce the address or phone number of the contact. Then the user only has to press the call button to initiate a call.

Additionally, Minisip can send instant messages using the same editlist with the contact information (number or address) and pressing the "IM" (Instant Message) button. Then the message will be sent so to the user.

Of course if the call or the message cannot be sent, a warning message indicating the reason will be shown on the interface.

The main window in the interface only has these options so it would seem that functionality is limited. However, unlike the Skype product, the main menu of Minisip does not incorporate redundancy - you have to know how to reach the functionality you want. The main menu consists simply of three items: `File`, `View`, and `Contact`.

The file item has a submenu with three items: `Preferences`, `Certificates Settings`, and `Quit`. After selecting the preferences item the following dialog appears:



SIP accounts

Register	Account	Default	PSTN
✓	torobravo7	✓	☐

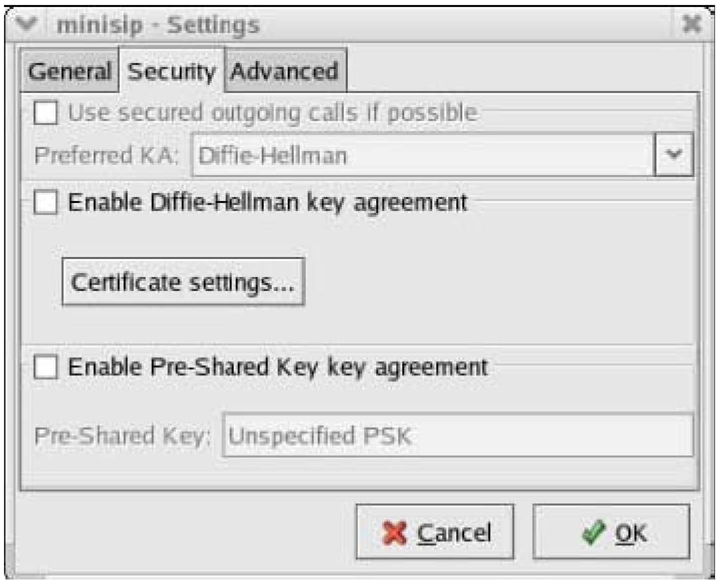
Devices

Sound device: /dev/dsp

Figure 23. Preferences submenu

This dialog shows in its first tab the general properties activated for this contact. Figure 23 shows the initial state of my "torobravo7" Minisip account - showing the user has a default configuration and that a PSTN account is not yet activated. To active it, simply press the PSTN account button and the radiobutton below the title PSTN will be marked. Via this dialog we can add or remove contacts using the "Add" or "Remove" buttons. The default account is related to the file created previously to use the first time the program is executed.

The second tab in this dialog contains the options related to security. As noted earlier Minisip allows the user to establish different security levels. The security dialog is show in the next figure:



☐ Use secured outgoing calls if possible

Preferred KA: Diffie-Hellman

☐ Enable Diffie-Hellman key agreement

☐ Enable Pre-Shared Key key agreement

Pre-Shared Key: Unspecified PSK

Figure 24. Security options in Minisip

The level of security for communication can be modified through this dialog. The user can select from Diffie-Hellman key agreement [42], [54], a Pre-Shared key [55], or using some security algorithm in an outgoing call only when it is possible. Here the user can select the authentication certificate [56] and the private key to use, as well as the Authority of certification [57]. By default these options are not activated. However, they will become available after pressing the Certificate settings button, at which point the following dialog will pop up:

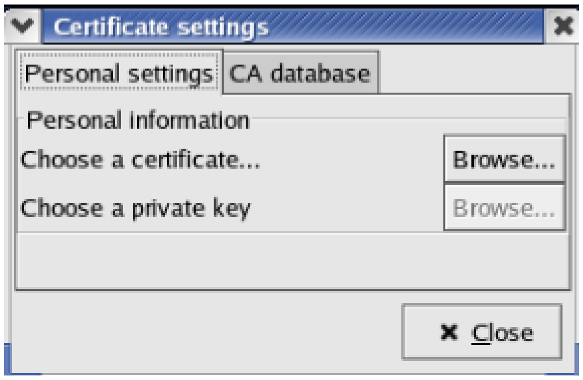
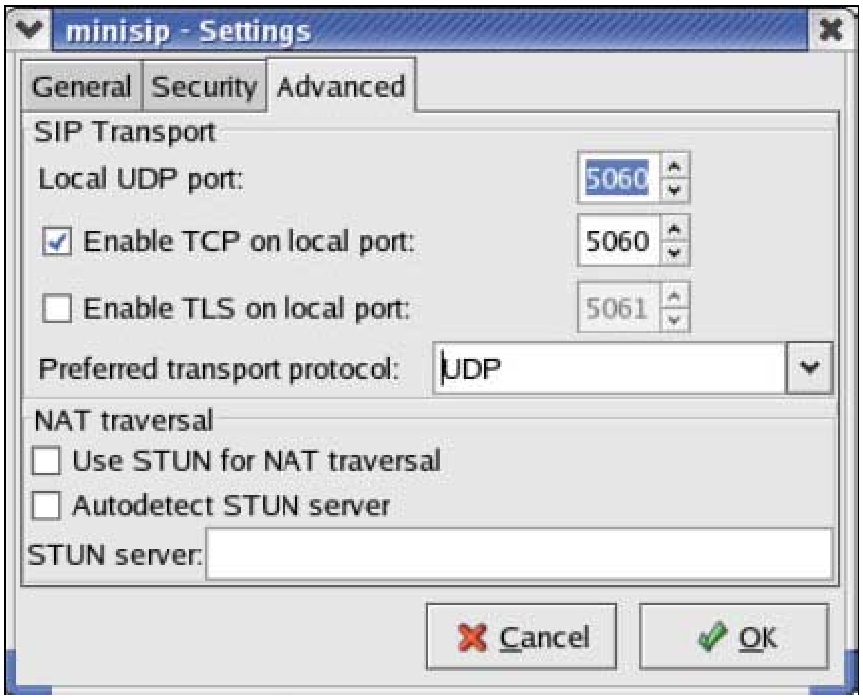


Figure 25. Certificate settings

Additional properties are managed by using the third tab titled “Advanced”. This tab opens a dialog with the advanced preferences, as shown in the figure below:



SIP Transport

Local UDP port: 5060

[✓] Enable TCP on local port: 5060

[ ] Enable TLS on local port: 5061

Preferred transport protocol: UDP

NAT traversal

[ ] Use STUN for NAT traversal

[ ] Autodetect STUN server

STUN server:

Figure 26. Advanced preferences for the communication in Minisip

This dialog enables the user to modify several important communication parameters. For example, the user is able to select which local port will be used for UDP, TCP, or Transport Security Layer (TLS) [58] traffic. By default UDP is used because it introduces less overhead (i.e. no confirmation or acknowledgement as would be provided by TCP nor security as provided by TLS). Moreover it is possible to active the STUN protocol [59] which is a lightweight protocol that allows applications to discover the presence and types of NATs and firewalls between the client and the public Internet, as well as determine the presence of STUN servers.

6.3 Common Graphical Interface

6.3.1 Design decisions

As it stated in the introduction, in this thesis two GUIs are created to hide the Minisip and Skype programs so that we are able to obtain objective results of the voice quality perceived. The first task was to determine the environment and the programming language to develop such a tool. In the “Problem Statement” section, it was noted that Microsoft Visual Studio 6.0 was selected for the laptop and Microsoft Embedded Visual C++ 4.0 for developing for the Pocket PC.

The selection of a programming environment depends on the programming language to be used. I considered that the most appropriate language to develop the graphical interfaces was C++ since Minisip was develop in this language, but it would have been possible to create the GUIs in another language like C# and to use a cross compiler to port and obtain compatibility.

The interface for the Pocket PC was the first program created and I planned to use Visual Studio .NET 2003, but although this program allows the user to work in C++, it does **not** provide the option to create a C++ program for the PDA. As a consequence, I chose Microsoft Embedded Visual C++ 4.0 for this development.

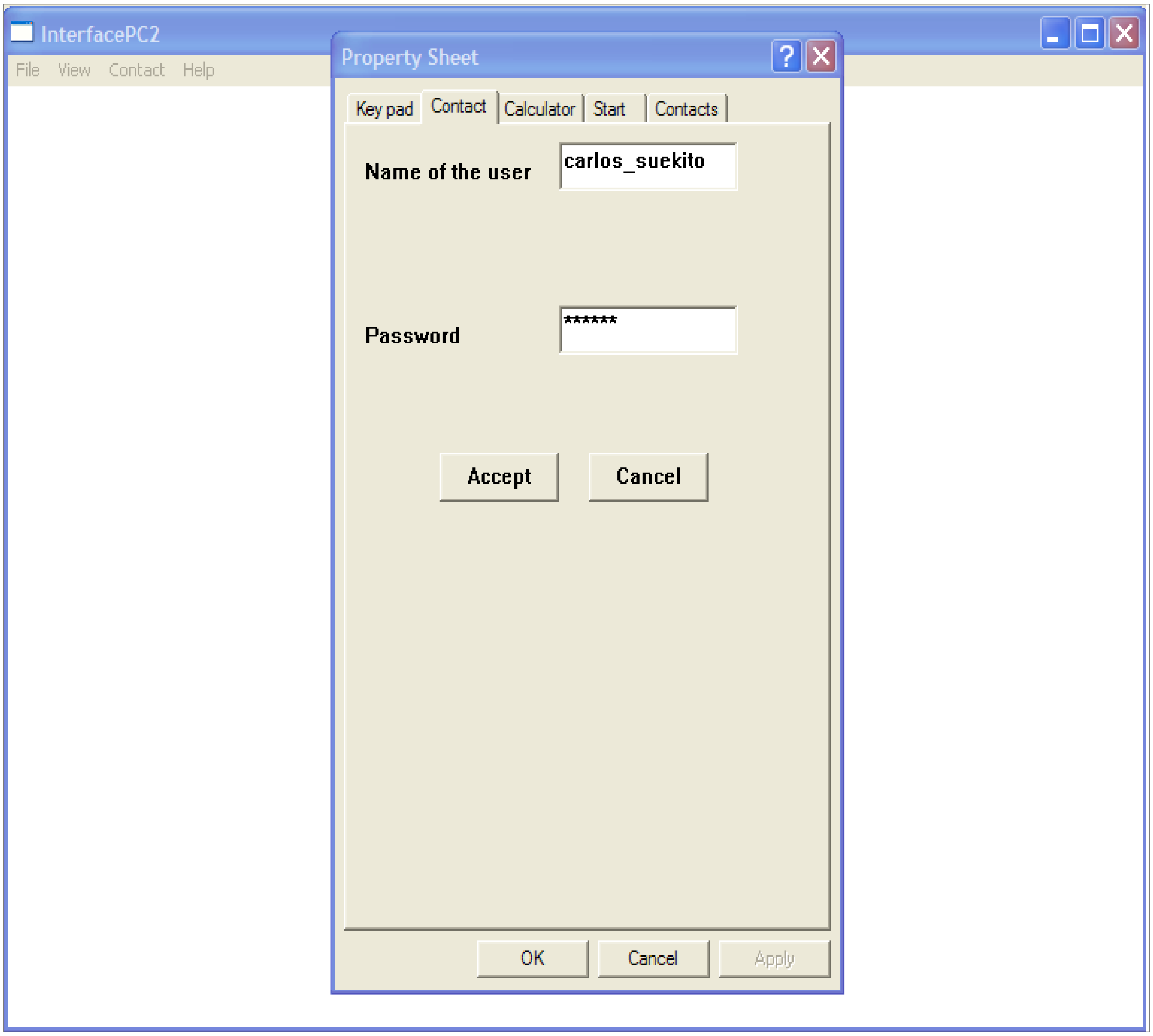
Once the GUI was created I tried to establish a connection with Skype in the Pocket PC. Unfortunately, I found that the current Skype software for Pocket PC does **not** allow other applications to connect with it. Thus it was not possible to connect the interface on the PDA to Skype. Because it was impossible to continue on the PDA I decided to translate the GUI available for the Pocket PC to a version for the laptop. Unfortunately, Microsoft´s Embedded Visual C++ is not completely compatible with Microsoft´s Visual Studio 6.0 so I had to introduce several modifications to the code.

6.3.2 Operation with the GUI

Besides comparing the support, development, and usability of both VoIP programs it is necessary to compare the quality of voice as perceived by the user when making calls. In order to perform an objective study I created a common GUI which hides from the user which program is being used, but allows the user to make calls. The goal was to examine the quality which the user perceives, hence allowing a comparison of Minisip with Skype. However, the situation is more complex that it might first seem, since Minisip users can change codec - resulting in different quality for each call.

During this thesis project I have created a similar GUI for both a (laptop) PC and for an HP iPAQ running Pocket PC. I will first explain the version for a (laptop) PC.

Since I have earlier shown the graphical interfaces in Minisip and Skype it was logical to implement a common interface similar to these two GUIs. For that reason, the common interface presents a main menu composed of four items: File, View, Contact, and Help. Inside the File item there is the option to invoke a dialog as shown in the figure below - called a property sheet:



Name of the user carlos_suekito

Password ******

Figure 27. Contact tab in common interface

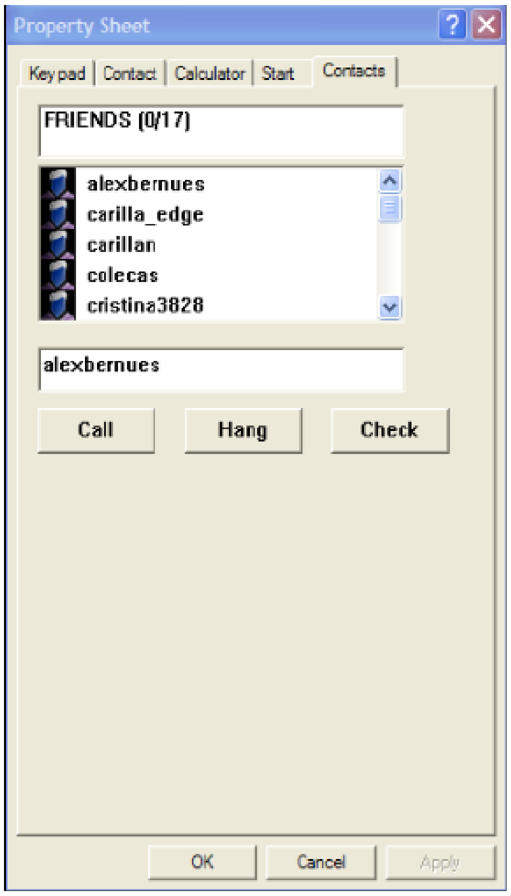
This dialog is composed of several tabs that I will explain in this section. The second tab shown in Figure 27 presents several editlist and fields in which the user can enter his or her own name and password before starting the experiment. Thus we can save in a file the name, password, and the result of the evaluation given for each user. It is also important to have this information both so that we know which user makes which call, but also so that we can invoke the underlying program as if it were being invoked by this user.

In the near future, we hope to be able to invoke both programs from this common interface. Unfortunately, this is not currently possible because the API provided by Skype does **not** allow us to perform the start up and login!

Once the user has written these data, they must press the "Accept" button in order to store the user's data and to initiate the experiment. Once they have pushed the "Accept" button, the interface will itself select which VoIP program is to be used for this communication session so that this user does not know which program underlies the common interface. This selection is performed randomly.

If the user decides not to continue, they can close the interface or start another registration by pressing the "Cancel" button.

Once the "Accept" button is pressed and the VoIP program has been selected, we can continue the experiment. The "contacts" tab now shows the contacts of the current user. It is shown in the next figure:



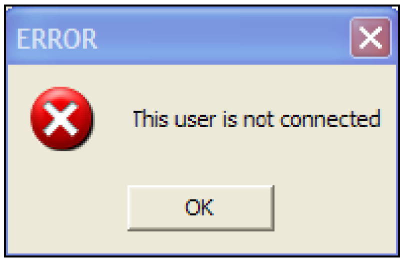
FRIENDS (0/17)

alexbernues
carilla_edge
carillan
colecas
cristina3828

alexbernues

Figure 28. Contact of the user.

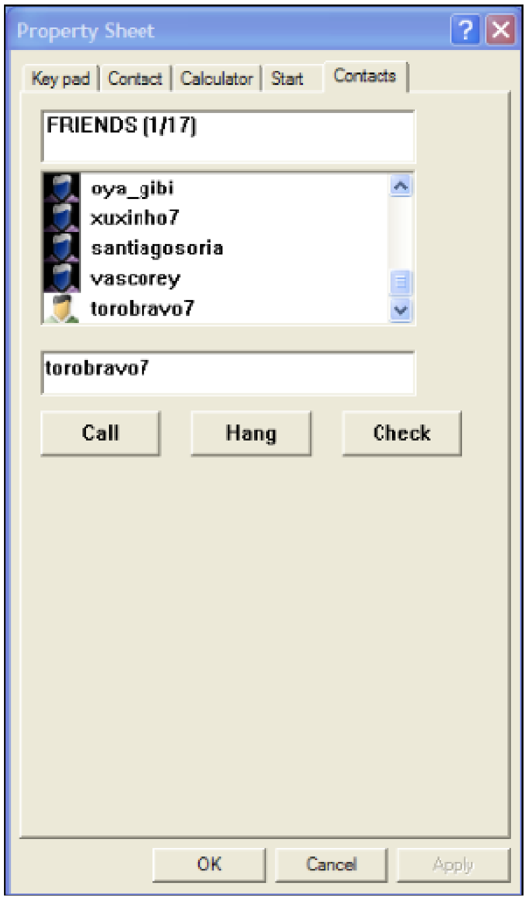
Initially all the contacts in the list are shown as if they were in an offline state. The reason for this is that Minisip does not indicate the current state of the user. When the user wants to make a call, the "Check" button must be pressed and at that time the program will check if the selected user is available (on-line) or not. Next, the "call" button can be pressed, then there are two choices: the contact is connected and a call is establish or the user is not available. In the later case, the following dialog will be shown:



This user is not connected

Figure 29. Dialog indicating the user is not available

If the user is available, the call will be initiated and the icon corresponding to that user will change its color. Moreover the position of this user will be located at the end of the list and the number of users connected will be increase in one.



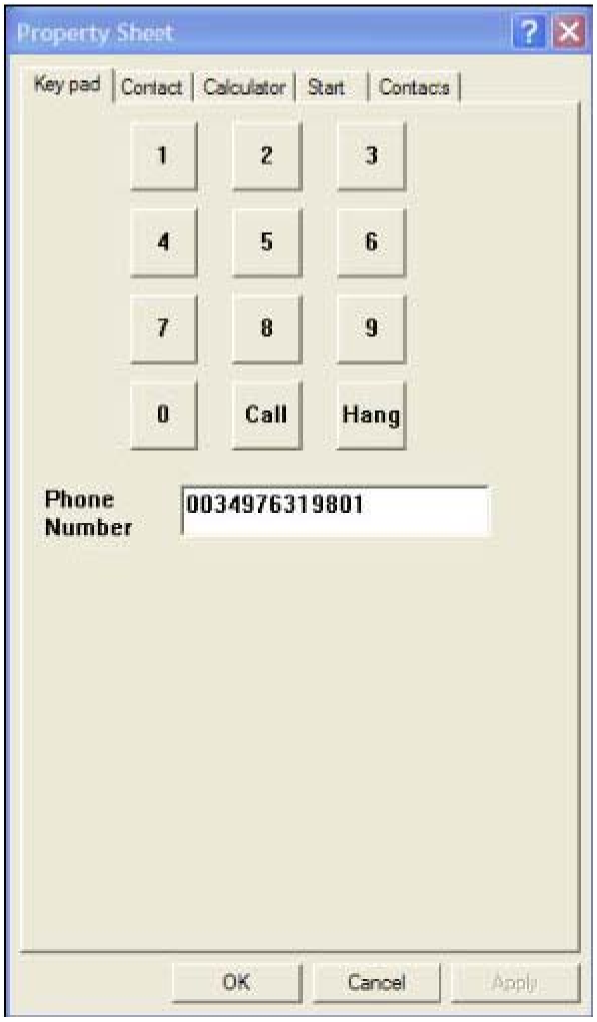
FRIENDS (1/17)

oya_gibi
xuxinho7
santiagosoria
vascorey
torobravo7

torobravo7

Figure 30. torobravo7 is available.

My common interface also introduces a keypad to make calls similar the one presented in Skype. This keypad allows the user to enter a PSTN or mobile number by directly entering the phone number or by entering a contact name or a SIP URI.



Phone Number 0034976319801

Figure 31. Dial in the interface

Using this keypad is as easy as dialing a telephone number and pressing the "call" button on a typical VoIP phone. To finish the call, simply press the "Hang" button. If the user wants to make call to a PSTN number, he or she must have positive credit in the SkypeOut service, otherwise the call cannot be made.

Both tabs, "keypad" and "contacts", allow establishing calls.

Once the user decides to terminate the communication session, they press the "Hang" or "Cancel" button, at this point a new dialog will pup up as in Figure 32. Here the user can evaluate the quality of the session with 1 being the worst quality and 5 the best quality. This evaluation will be stored in a file (along with the username and the password). Thus creating a log of the users who participated in the experiments along with their evaluations. Once the user enters their evaluation, the dialog will be closed and the program will be ready to start again.

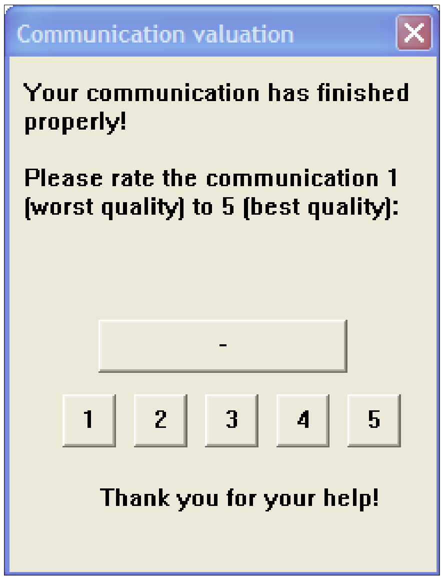

Communication valuation
Your communication has finished properly!
Please rate the communication 1 (worst quality) to 5 (best quality):
-
1
2
3
4
5
Thank you for your help!

Figure 32. Evaluation dialog

.

Additionally, I created a main menu which allows me to perform the same experiments without using the property sheet. Using the property sheet is easier and faster than using the main menu. However, the main menu could be used to introduce new features to permit compare both programs not only for speech communication, but also for instant messaging.

Once the common graphical interface was created, the connection with Skype and Minisip needed to be done. Currently only Skype provides such an API. The design and the limitations of this API influenced the design of my common interface. The same idea will be used for the connection with Minisip. However, unlike the connection with Skype which currently works, the connection with Minisip will not be possible until the Minisip Windows port is available. Unfortunately, this Windows port is not yet available, therefore experiments will be performed on different platforms to allow me to finish this comparison. Once the port is available, then experiments could be repeated on a single platform.

6.4 Comparison of The User Interface

Skype clearly provides much more extensive help and information to users and developers than does Minisip. While a number of theses have described their extension to Minisip – many of these theses only became available at the end of the thesis project. There is no corresponding on-going news as there is for Skype (although there are local weekly VoIP meetings and the minutes of these meetings are sent to the participants - they are not readily available from the Minisip web site).

In Minisip, because the source code is available, anyone can download the source code and change it in order to develop whatever service they wish and they can examine it to find and fix whatever bugs they encounter.

While there are two e-mail lists for Minisip (for users and the developers), my experience was that I got questions answered much more quickly from Skype. In fact, I was fortunately in having most of my questions concerning Skype answered by one of the Skype developers. One of the reasons for this is clearly the vast difference between a commercial product with a large development and support organization versus a group of graduate students and others simply interested in utilizing Minisip for their own purposes. These differences are consequence of these different goals of the two programs:

- Skype receives incomes from their partners in the levels 2 and 3 and in the third level the partners even provide technology as in the case of GlobalIPSound [60] which provides the codec.
- The knowledge and the availability of the code in Minisip allows developers to introduce new services without restrictions, that is, the possibilities are much greater than for Skype.

In terms of installation, it must be noted that in addition to having support for many more operating systems, that even when comparing the installation process for the different Linux versions, the installation process for Skype is faster and easier than in Minisip.

With respect to the usability of each program, both interfaces allow the user to work without problems since they are quite clear, but there are four points to comments upon:

1. The first point is that the choices Minisip provides to the user might be considered as more complex in the sense that the knowledge required of the user is greater than in the case of Skype. For example, if the user wants to turn on the Diffie-Hellman key agreement, he must first consider the advantages and disadvantages the activation will provide him.
2. The second point is that, unfortunately, the Minisip GUI is not being developed - thus new features have been included in the program, but a normal user is not able to access them through the GUI. Therefore, if the user wants to enjoy these new features, he has to search in the code for how to run them which is quite unfortunate. For example, how to enable 16 bit linear encoding.
3. The third point is that the user does not have to be concerned with firewalls, as the router configuration is determined by Skype. A new user only needs to create a new account, all the rest is done by the underlying Skype protocol, while in the Minisip in addition to creating an account, the Minisip user must also decide upon the proxy to use for their communications.
4. Finally, Minisip has not displayed the state of the user's contacts via the GUI. I think from the user's point of view that this information is quite interesting and it would be desirable to introduce this functionality in order to avoid situations in which the user tries to establish a conversation, but can not do so and does not know why it is not possible to do so.

After performing several calls with Skype and Minisip with PCMu and iLBC codecs we realized that the perceived voice quality in Skype is much better than in Minisip. Using iLBC the quality is improved over PCMu but still worse than Skype. In Skype the user feels that the callee is closer and the voice is clearer distinguishing perfectly the different letters, whereas in Minisip the voice is perceived as if it is coming from far and noisy. It was impossible to make calls with a 16-bit linear encoding since the last modifications in Minisip did not include this codec so the results are reasonable because the codec utilized in Skype is broadband while the two codecs in Minisip are narrowband.

7 Packet Level Comparison of Skype and Minisip

This comparison was performed by studying with Ethereal [62], the information sent during the operation of both programs. Thus we can calculate, for each of the main features to be considered, the amount of data sent, the bandwidth used, the time necessary to perform a given function, and even the entities involved in such communication. Given this information several metrics could be used to evaluate and compare the programs, including the difference due to the user's selection of the security level they wish.

7.1 Skype

7.1.1 Login

The decentralized architecture (peer to peer) of Skype results in a complex login process where an number of nodes participate and where the information to be sent and received by the user is significant.

This process mainly involves three entities or machines: the Skype client (SC), one or more supernodes (SN) which will establish a connection with the Skype client in order to perform the login, and one or more login servers. Actually, during the login process the Skype client establishes UDP sessions with several nodes, not only with the three nodes previously mentioned.

In order to login, the user must enter his username and password in the dialog provided in the Skype GUI. Once these data have been introduced, the Skype client starts to establish UDP connections with different nodes following several patterns. These nodes are called supernodes and the client must communicate with at least one of them to complete the login process. These nodes will reply to the client and then the client will establish a TCP session which will allow an exchange of information between both machines. A scheme showing that exchange of information is in the following Figure 33.

Once that exchange of information has happened, then the Skype client has the address of a login server (in fact, it is conjectured that the user has the address of more than one server login). In such a situation, the client will set up a connection with the login server to perform the authentication of the Skype user. The communication is shown in Figure 34.

At this point, the client has been authenticated and the login process will have finished if the Skype client is running over Windows. But if the client is running over Linux software, then I have found a couple of differences. First, the initial communications between the SC and SN are a bit different and second, and much more important, there is communication with a fourth node (also centralized). Such a communication is described in the following Figure 35.

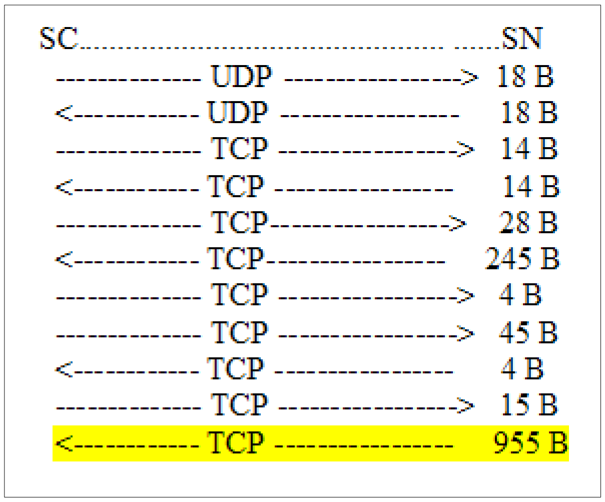

SC..SN
-------------- UDP ----------------> 18 B
<------------ UDP ----------------- 18 B
-------------- TCP ----------------> 14 B
<------------ TCP ----------------- 14 B
-------------- TCP---------------> 28 B
<------------ TCP---------------- 245 B
-------------- TCP ----------------> 4 B
-------------- TCP ----------------> 45 B
<------------ TCP ----------------- 4 B
-------------- TCP ----------------> 15 B
<------------ TCP ----------------- 955 B

Figure 33. Exchange of information between a Skype client (SC) and a supernode (SN) during the login. The amount of data sent in bytes

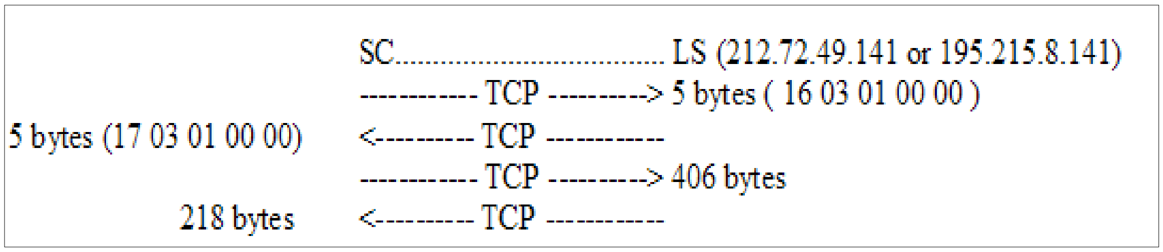

SC.................................... LS (212.72.49.141 or 195.215.8.141)
------------ TCP ---------> 5 bytes (16 03 01 00 00)
5 bytes (17 03 01 00 00) <--------- TCP -----------
------------ TCP ---------> 406 bytes
218 bytes <--------- TCP -----------

Figure 34. Communication between SC and a login server (LS)

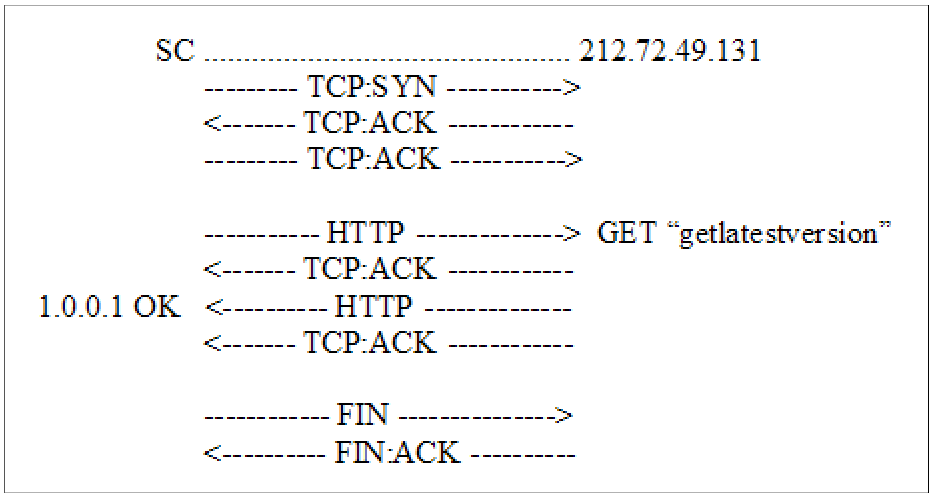
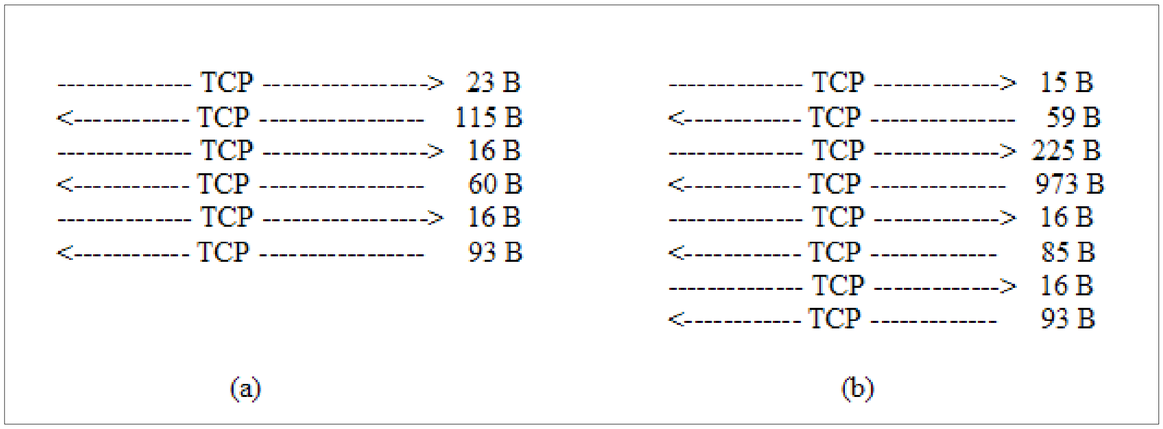

SC 212.72.49.131
--------- TCP:SYN ---------->
<------- TCP:ACK -----------
--------- TCP:ACK ---------->
----------- HTTP -------------> GET "getlatestversion"
<------- TCP:ACK -----------
1.0.0.1 OK <---------- HTTP -------------
<------- TCP:ACK -----------
------------ FIN -------------->
<---------- FIN:ACK ----------

Figure 35. Exchange of information with the "version server"

In my opinion the above communication is reminiscent of a previous version of Skype and will be eliminated in the future (as is explained in the appendix A). In fact, in a previous study of the Skype product [63] a similar communication pattern was found and it was argued that such a node is the login server. But in that case there was observed after the GET “getlastversion”, two messages were exchanged in which the authentication was done. However, in the new version of Skype this node only appears for linux software and without the authentication messages have disappeared.

It is also important to indicate the communication with the supernode (SN) does not end with the last transfer shown, but it continues until the client closes the session or that supernode is not available in which case a new SN will be found.

In fact, during the login process, we should see a transfer **after** the authentication has been done. In this transfer, between SC and SN, information related to the contacts of this user will be exchanged. To show that, two transfers of data are displayed below corresponding to the cases in which the client does **not** have contact in his contact list and when the client has 20 contacts.



```
-------------- TCP ----------------->  23 B        -------------- TCP ------------->  15 B
<------------ TCP -----------------   115 B        <------------ TCP ---------------   59 B
-------------- TCP ----------------->  16 B        -------------- TCP -------------> 225 B
<------------ TCP -----------------    60 B        <------------ TCP --------------   973 B
-------------- TCP ----------------->  16 B        -------------- TCP ------------->  16 B
<------------ TCP -----------------    93 B        <------------ TCP -------------    85 B
                                                   -------------- TCP ------------->  16 B
                                                   <------------ TCP -------------    93 B

                  (a)                                                (b)
```

Figure 36. Information about the contacts (a) no contacts, (b) 20 contacts

The number of nodes the user contacts initially is not fixed, but varies which does not allow me to calculate the bandwidth required. The minimum bandwidth in the case when the client only communicates with the supernode and the login server, and the user does not have any contacts is 3293 bytes (with some little variation in the transmission ± 5 bytes), but the time to perform all this communications varies between 1 or more seconds. Thus the average is not useful.

7.1.2 Call Establishment

With respect to the establishment of the call in Skype, three points are worth mentioning:

First of all, the captures revealed that the signaling utilizes TCP and UDP which is different from the old versions of the Skype product studied in [63].

The second aspect concerns whether the callee belongs to the buddy list or list of contacts, if not then the establishment of the call will include a search for that contact as well as the call establishment signaling. Finally the third aspect is the signaling shown in Figure 37.

In the communication between calleer and callee, messages are sent over UDP and TCP. In my opinion, TCP supports the basic signaling to setup the call whereas the UDP traffic is used to exchange some information which improves the communication beyond that of previous versions.

In the figure above, it is observed that the first transfers are carried over UDP. In particular I want to comment on the two messages between both users which are of sizes 103 and 41 bytes (respectively). These two messages carry the warning from the calleer to the callee indicating that the calleer is online and the message with 41 bytes is the reply. I can assume this because when a user perform the login process a message with 101 or 103 bytes is sent to each one of his contacts and the reply from each one of them contains 41 bytes.

As described in appendix A, the first transfers of information do not really correspond to the establishment process since otherwise the duration of the process would be longer than 5 or 6 seconds. As the time between pressing of the call button by the calleer and the moment at which the telephone rings is shorter than these values.

After these messages over UDP, a TCP session between both users is initiated and two messages in each direction are exchanged containing 14 bytes. Those two messages probably perform mutual authentication. The other transfers in the session setup cannot be explained because, unfortunately, I do not better knowledge about the proprietary Skype protocol.

However, the last two transfers, with message sizes of 16 (or 17) and 14 (or 13) bytes, correspond to the tear down of the call and the direction of these messages depends on the user who decides to tear down the call.

Finally, I have calculated the average delay of the call establishment (0,19 seconds) as well as the average of bytes transmitted (2723,75 bytes). With these two values it was obtained the average bandwidth used in the establishment and in the tear down processes can be calculated. These results are:

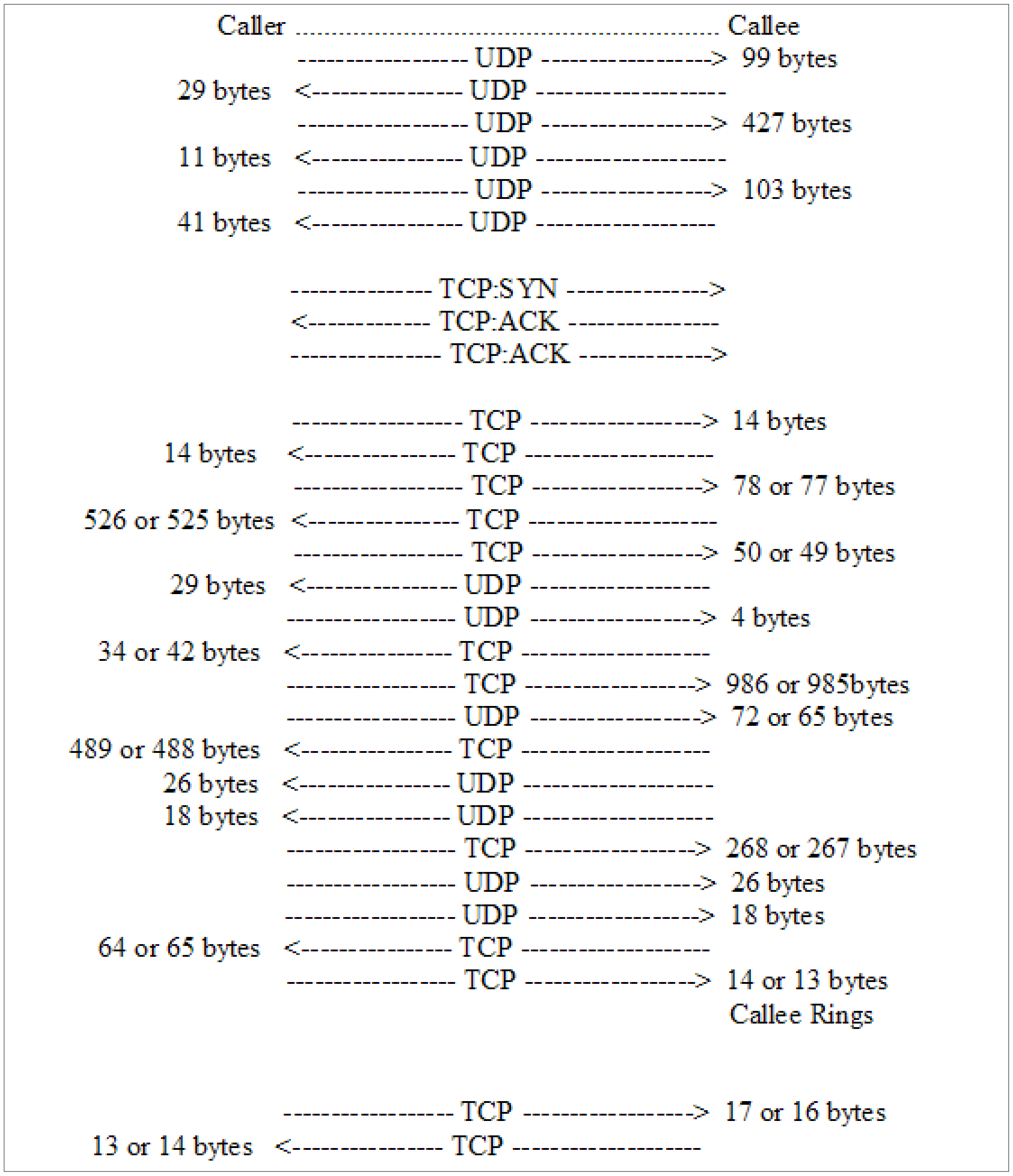
BWcall_establishment ≈ 114684 bps
BWcall_ending ≈ 1537 bps

The results obtained show that the bandwidth required for the signaling in the initialization of the call is almost five times higher than the value described in the previous measurements [63].

7.1.3 Media Transfer and Codecs

The different packet capture session enabled me to see that the voice is carried over UDP. In fact, the user of the Windows software can modify the port used to send the voice data (Linux and Pocket PC versions do not have this facility).

I have also found that the size of these packets is **not** fixed, and even more, it is not possible to discern a pattern in their variation. Moreover when the call is initialized but the user does **not** speak, the bandwidth does not decrease, this indicates that the current Skype version does not use a voice activity detection mechanism.



```
                 Caller ..................................................... Callee
                        ----------------- UDP ----------------> 99 bytes
               29 bytes <--------------- UDP ------------------
                        ----------------- UDP ----------------> 427 bytes
               11 bytes <--------------- UDP ------------------
                        ----------------- UDP ----------------> 103 bytes
               41 bytes <--------------- UDP ------------------

                        -------------- TCP:SYN -------------->
                        <------------ TCP:ACK ---------------
                        --------------- TCP:ACK ------------->

                        ----------------- TCP ----------------> 14 bytes
               14 bytes <--------------- TCP ------------------
                        ----------------- TCP ----------------> 78 or 77 bytes
        526 or 525 bytes <--------------- TCP ------------------
                        ----------------- TCP ----------------> 50 or 49 bytes
               29 bytes <--------------- UDP ------------------
                        ----------------- UDP ----------------> 4 bytes
           34 or 42 bytes <--------------- TCP ------------------
                        ----------------- TCP ----------------> 986 or 985bytes
                        ----------------- UDP ----------------> 72 or 65 bytes
        489 or 488 bytes <--------------- TCP ------------------
               26 bytes <--------------- UDP ------------------
               18 bytes <--------------- UDP ------------------
                        ----------------- TCP ----------------> 268 or 267 bytes
                        ----------------- UDP ----------------> 26 bytes
                        ----------------- UDP ----------------> 18 bytes
          64 or 65 bytes <--------------- TCP ------------------
                        ----------------- TCP ----------------> 14 or 13 bytes
                                                                Callee Rings


                        ----------------- TCP ----------------> 17 or 16 bytes
          13 or 14 bytes <--------------- TCP ------------------
```

Figure 37. Call establishment signaling

The bandwidth used in voice transmission was calculated taking intervals with a duration of approximately a second and calculating the number of bytes transmitted. Once those values were obtained, I could calculate the bandwidth. To remove the dependence on the specific temporal period considered, I utilized several periods of the same duration and finally the average the values yielding with ± bytes/sec. variance:

- With media traffic exchanged:

$BW_{mt} \approx$ 7892 Bytes/sec.

- Without media traffic exchanged (just silence):

$BW_{wmt} \approx$ 7175 Bytes /sec.

The comparison of these bandwidths supports the previous statement about the non-existence of voice activity detection mechanism in the current version of Skype. During voice communication, the bandwidth used varies from 3 to 16 kbps.

TCP traffic was also observed during the call. That traffic corresponds to messages with a size of 18 bytes from one user to another user and the respective acknowledgements (ACK). The interval between the sending of these messages in one direction and in the other varies. My hypothesis (see appendix A) is that these messages are used to inform the other user about the kind of information is going to be sent in the next UDP packets.

On average, the TCP traffic is:

- Caller to Callee, Callee to Caller:
T1 = 0,31 sec.
- Callee to Caller, Caller to Caller:
T1 = 0,68 sec.

With respect to the rate of sending TCP messages, the average value obtained was:

$BW_{tcp} \approx$ 40,74 Bytes/sec.

But, despite the values enumerated above, delay and jitter in these communications were also studied. To do that, first we obtained a measure of the delay introduced by the network running Skype and Ethereal in both terminals joining the communication and with the same NTP [78] clock, that is time reference:

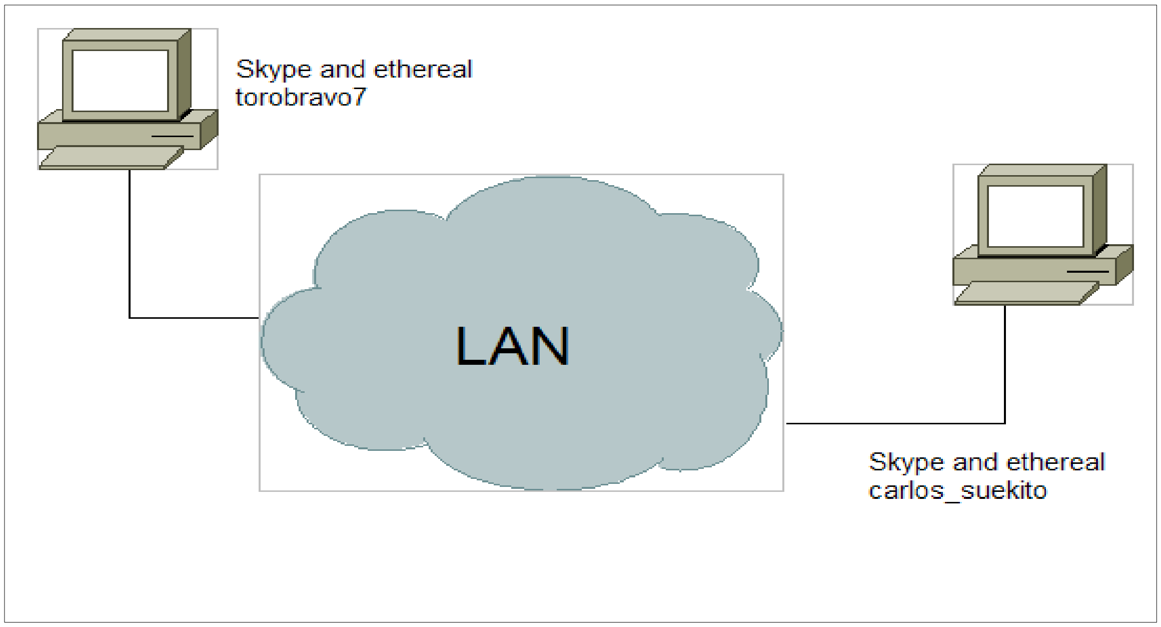


Figure 38. Scheme to calculate the delay and jitter in the network

In such a scheme previously presented, the values calculated in average for the delay and the jitter are:

t(delay) ≈ 0,4272 msec.
σ (jitter) ≈ 0,10771 msec.

Once we know the average of the delay and the variance of this value, we calculate the real delay following the next scheme:

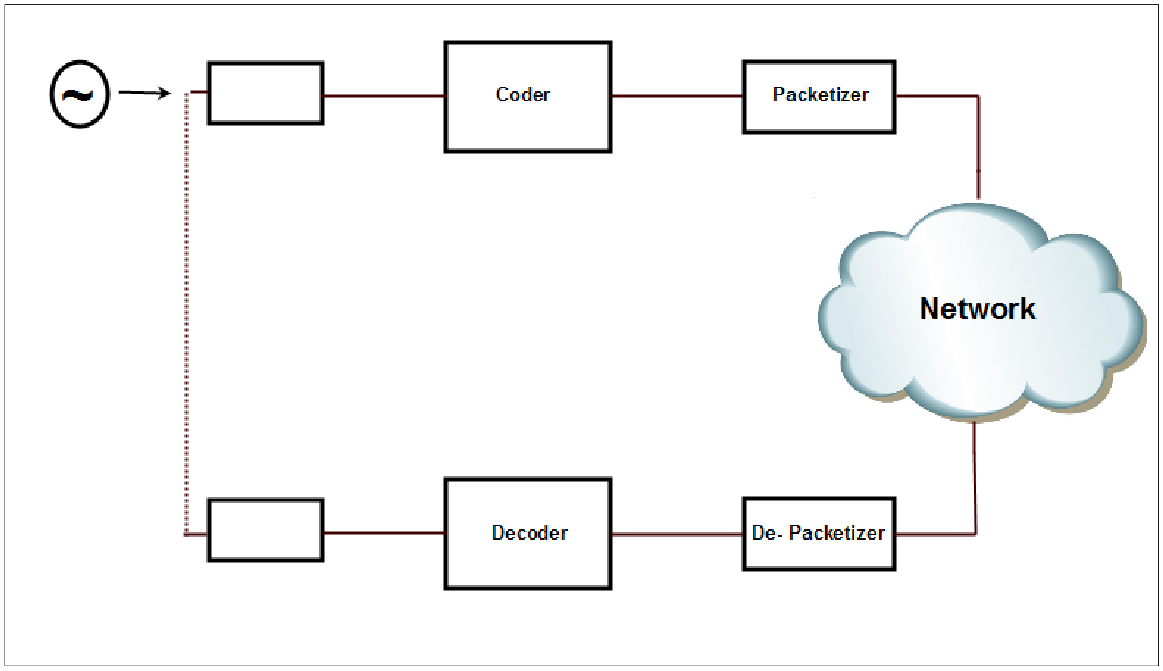


Figure 39. Scheme to calculate the delay and jitter

We introduce a signal, which can be a whistle, for example, and we record the conversation. Since the path is closed we are going to receive the same signal, with less power every T seconds. This value is the real delay to be measured. To obtain this period we can calculate the Fourier transform of the signal. Since the value of the delay introduced for the network is around 0,43 msec, the frequency corresponding to T will be really small.

The value of the real delay calculated following the explanation is T = 0,89 seconds, and the variation of this delay, that is the jitter is 0.014 seconds. As a consequence, the time to encode the information and to introduce in packets is the half of the difference between these two values, that is approximately 0,445 sec. Once we know that value, we can study the variation of such a delay due to the network.

Moreover of these values, the traffic and bandwidth used by the Skype user in an on-line state without using the program was measured. The results of several of captures are shown in the appendix A but was between 0 and 0.5 kbps.

With reference to the encoding used by Skype, I do **not** know exactly codec was used. However, it can be shown that Skype uses a broadband codec that works with frequencies up to 16000 Hz. which permits it to have a better quality of voice than in basic PSTN telephony. Although this was stated at Skype´s website [21], I confirmed this by performing an experiment based on recording the signal one of the participants receives. Then I process this signal to see the frequencies included. I used Matlab [64] to calculate the Fourier transform of the signal. I recorded the call with a sampling frequency of 22542 Hz. In the frequency domain I have centered the signal at 0 so it is symmetric at this frequency and this represents 0 Hz in the frequency domain (figures 41 and 42). In Figure 41 we can see that we have signal frequencies until almost 0,5, which means that this signal has frequencies until almost 11.25 KHz. As a consequence, we have shown that Skype use a broadband encoding.

In the Figure 40 and 41 below we can see the signal represented in the time domain and in the frequency domain.

The Figure 42 shows the power of the signal. As was stated in the section 5.3, the main part of this power is focused within the low frequencies.

It is also possible to establish that the CODEC used is not among those below:

- iLBC
- RCU
- Enhanced G.711
- iPCM_wb
- ISAC

Since some of them are narrowband codecs (e.g. iLBC, RCU, and Enhanced G.711), and the others do not provide the service features provided by Skype as is described in detail in appendix A.

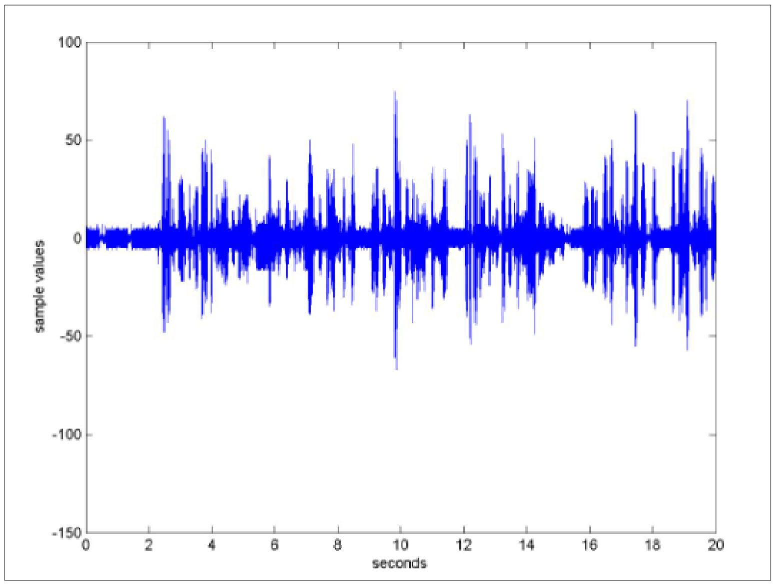


Figure 40 . Signal temporal domain.

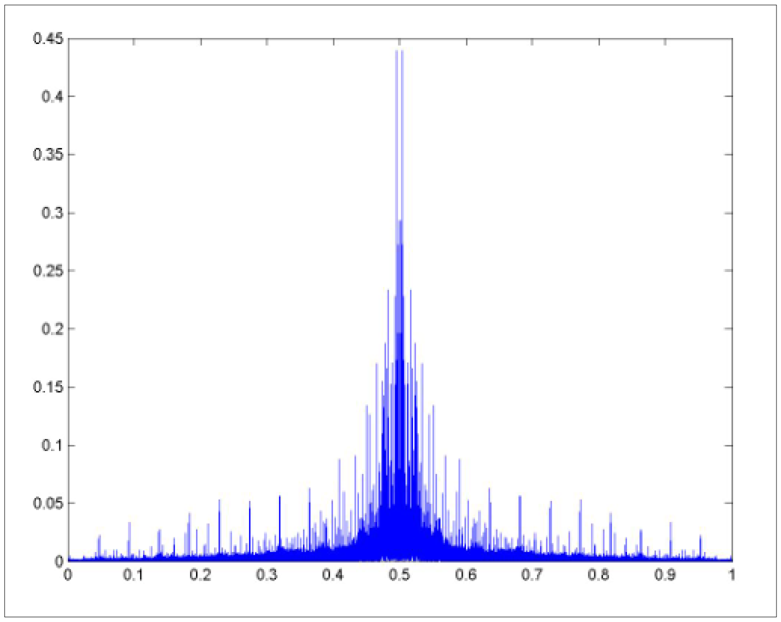


Figure 41. Signal in frequency domain with sampling frequency of 22 KHz.

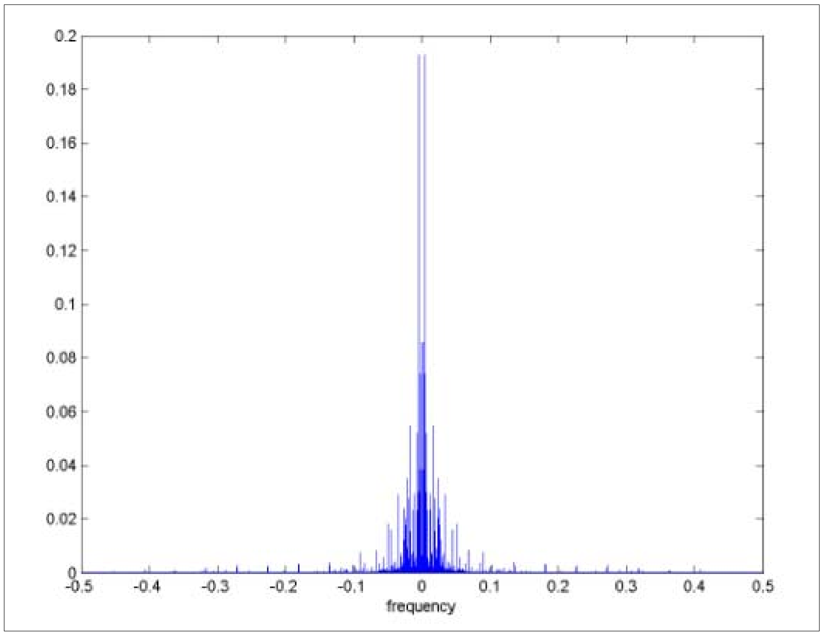


Figure 42. Signal power spectrum.

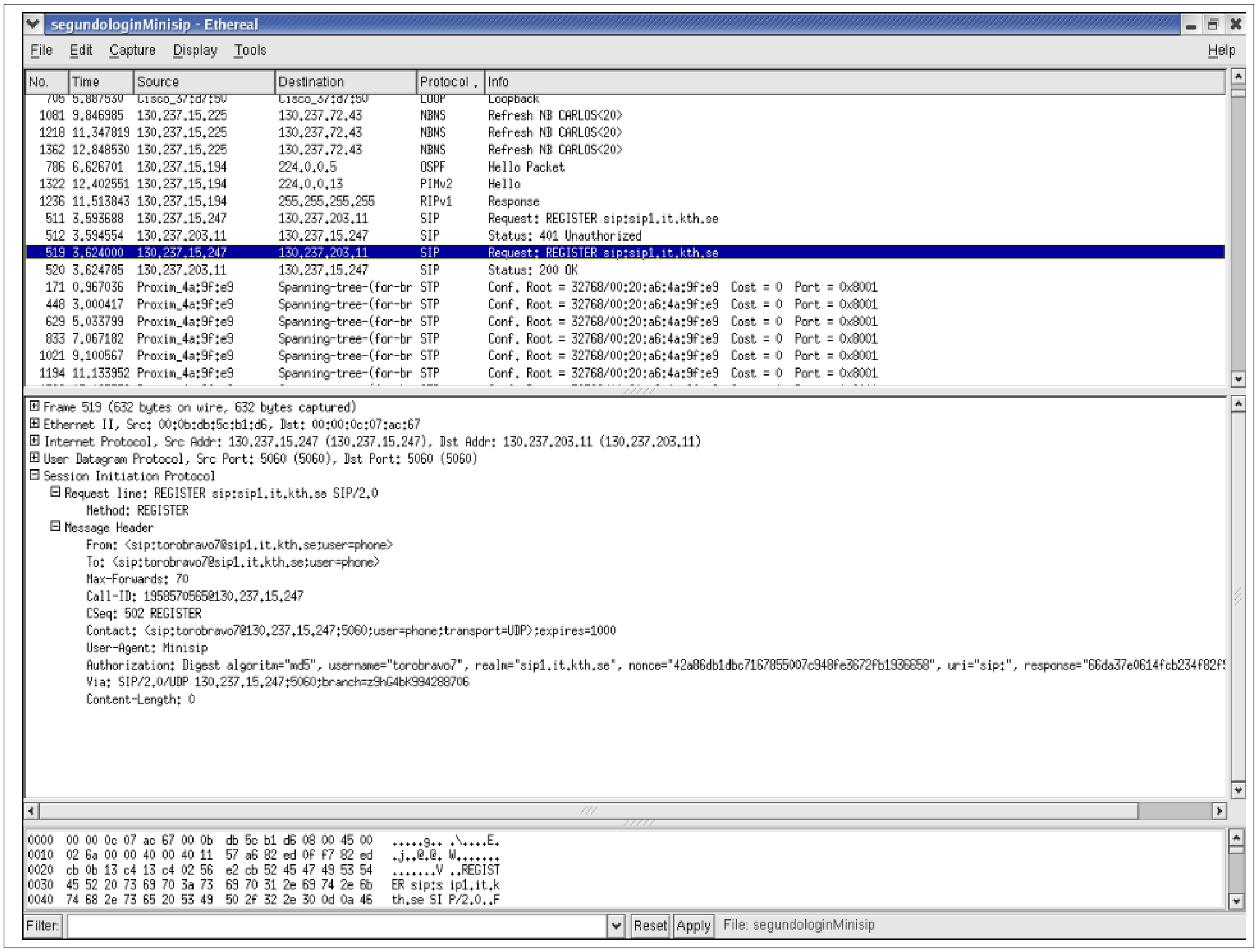
7.2 Minisip

7.2.1 Login

In the case of Minisip, the login is a easier process than in Skype. In fact, there are only four transferences to consider:

- Request to the ip address of the proxy server to the DNS server,
- Reply to this message,
- SIP REGISTER message
- Reply to the REGISTER message

In the following figure corresponding to a capture performed with Ethereal we can see the SIP transfers between the client and the Proxy server:



```
No.  Time      Source           Destination          Protocol  Info
 705 5.887530  Cisco_37:d7:50   Cisco_37:d7:50       LOOP      Loopback
1081 9.846985  130.237.15.225   130.237.72.43        NBNS      Refresh NB CARLOS<20>
1218 11.347819 130.237.15.225   130.237.72.43        NBNS      Refresh NB CARLOS<20>
1362 12.848530 130.237.15.225   130.237.72.43        NBNS      Refresh NB CARLOS<20>
 786 6.626701  130.237.15.194   224.0.0.5            OSPF      Hello Packet
1322 12.402551 130.237.15.194   224.0.0.13           PIMv2     Hello
1236 11.513843 130.237.15.194   255.255.255.255      RIPv1     Response
 511 3.593688  130.237.15.247   130.237.203.11       SIP       Request: REGISTER sip:sip1.it.kth.se
 512 3.594554  130.237.203.11   130.237.15.247       SIP       Status: 401 Unauthorized
 519 3.624000  130.237.15.247   130.237.203.11       SIP       Request: REGISTER sip:sip1.it.kth.se
 520 3.624785  130.237.203.11   130.237.15.247       SIP       Status: 200 OK
 171 0.967036  Proxim_4a:9f:e9  Spanning-tree-(for-br STP      Conf. Root = 32768/00:20:a6:4a:9f:e9  Cost = 0  Port = 0x8001
 448 3.000417  Proxim_4a:9f:e9  Spanning-tree-(for-br STP      Conf. Root = 32768/00:20:a6:4a:9f:e9  Cost = 0  Port = 0x8001
 629 5.033799  Proxim_4a:9f:e9  Spanning-tree-(for-br STP      Conf. Root = 32768/00:20:a6:4a:9f:e9  Cost = 0  Port = 0x8001
 833 7.067182  Proxim_4a:9f:e9  Spanning-tree-(for-br STP      Conf. Root = 32768/00:20:a6:4a:9f:e9  Cost = 0  Port = 0x8001
1021 9.100567  Proxim_4a:9f:e9  Spanning-tree-(for-br STP      Conf. Root = 32768/00:20:a6:4a:9f:e9  Cost = 0  Port = 0x8001
1194 11.133952 Proxim_4a:9f:e9  Spanning-tree-(for-br STP      Conf. Root = 32768/00:20:a6:4a:9f:e9  Cost = 0  Port = 0x8001
```

```
⊞ Frame 519 (632 bytes on wire, 632 bytes captured)
⊞ Ethernet II, Src: 00:0b:db:5c:b1:d6, Dst: 00:00:0c:07:ac:67
⊞ Internet Protocol, Src Addr: 130.237.15.247 (130.237.15.247), Dst Addr: 130.237.203.11 (130.237.203.11)
⊞ User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)
⊟ Session Initiation Protocol
   ⊟ Request line: REGISTER sip:sip1.it.kth.se SIP/2.0
        Method: REGISTER
   ⊟ Message Header
        From: <sip:torobravo7@sip1.it.kth.se;user=phone>
        To: <sip:torobravo7@sip1.it.kth.se;user=phone>
        Max-Forwards: 70
        Call-ID: 1958570565@130.237.15.247
        CSeq: 502 REGISTER
        Contact: <sip:torobravo7@130.237.15.247:5060;user=phone;transport=UDP>;expires=1000
        User-Agent: Minisip
        Authorization: Digest algoritm="md5", username="torobravo7", realm="sip1.it.kth.se", nonce="42a86db1dbc7167855007c948fe3672fb1936658", uri="sip:", response="66da37e0614fcb234f82f
        Via: SIP/2.0/UDP 130.237.15.247:5060;branch=z9hG4bK994288706
        Content-Length: 0
```

```
0000  00 00 0c 07 ac 67 00 0b  db 5c b1 d6 08 00 45 00   .....g.. .\....E.
0010  02 6a 00 00 40 00 40 11  57 a6 82 ed 0f f7 82 ed   .j..@.@. W.......
0020  cb 0b 13 c4 13 c4 02 56  e2 cb 52 45 47 49 53 54   .......V ..REGIST
0030  45 52 20 73 69 70 3a 73  69 70 31 2e 69 74 2e 6b   ER sip:s ip1.it.k
0040  74 68 2e 73 65 20 53 49  50 2f 32 2e 30 0d 0a 46   th.se SI P/2.0..F
```

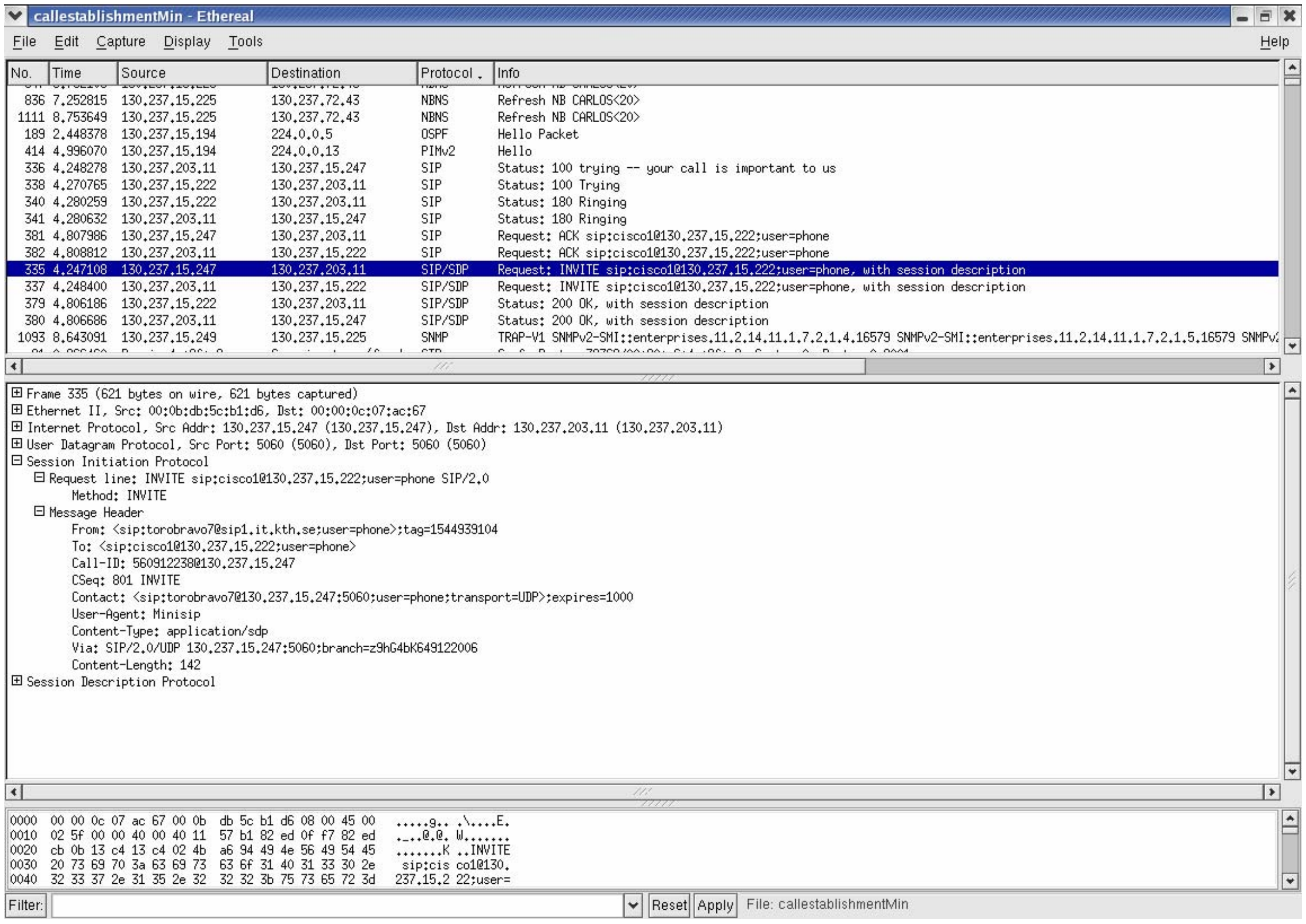
Figure 43. Capture of the login process in Minisip.

We can also see that the first REGISTER message receives a reply 401 Unauthorized and it is in the second request when the reply is the correct one. The explanation to this situation is that the first REGISTER message is sent to the registrar server without the username, the password and the hash. Then the registrar does not accept this message and the client sends another message including these parameters which is accepted by the server.

Looking the packets shown it is possible to calculate the time and the number of bytes sent to complete the registration or login. This time is 31.1 msec. approximately and the number of bytes transmitted is 2540 bytes.

7.2.2. Call establishment

When performing the call establishment is possible to check that the operation is the same that was described in the background section for the SIP protocol. The capture of these packets is shown in the figure below:



```
No.  Time     Source          Destination     Protocol  Info
 836 7.252815 130.237.15.225  130.237.72.43   NBNS      Refresh NB CARLOS<20>
1111 8.753649 130.237.15.225  130.237.72.43   NBNS      Refresh NB CARLOS<20>
 189 2.448378 130.237.15.194  224.0.0.5       OSPF      Hello Packet
 414 4.996070 130.237.15.194  224.0.0.13      PIMv2     Hello
 336 4.248278 130.237.203.11  130.237.15.247  SIP       Status: 100 trying -- your call is important to us
 338 4.270765 130.237.15.222  130.237.203.11  SIP       Status: 100 Trying
 340 4.280259 130.237.15.222  130.237.203.11  SIP       Status: 180 Ringing
 341 4.280632 130.237.203.11  130.237.15.247  SIP       Status: 180 Ringing
 381 4.807986 130.237.15.247  130.237.203.11  SIP       Request: ACK sip:cisco1@130.237.15.222;user=phone
 382 4.808812 130.237.203.11  130.237.15.222  SIP       Request: ACK sip:cisco1@130.237.15.222;user=phone
 335 4.247108 130.237.15.247  130.237.203.11  SIP/SDP   Request: INVITE sip:cisco1@130.237.15.222;user=phone, with session description
 337 4.248400 130.237.203.11  130.237.15.222  SIP/SDP   Request: INVITE sip:cisco1@130.237.15.222;user=phone, with session description
 379 4.806186 130.237.15.222  130.237.203.11  SIP/SDP   Status: 200 OK, with session description
 380 4.806686 130.237.203.11  130.237.15.247  SIP/SDP   Status: 200 OK, with session description
1093 8.643091 130.237.15.249  130.237.15.225  SNMP      TRAP-V1 SNMPv2-SMI::enterprises.11.2.14.11.1.7.2.1.4.16579 SNMPv2-SMI::enterprises.11.2.14.11.1.7.2.1.5.16579 SNMPv
```

```
⊞ Frame 335 (621 bytes on wire, 621 bytes captured)
⊞ Ethernet II, Src: 00:0b:db:5c:b1:d6, Dst: 00:00:0c:07:ac:67
⊞ Internet Protocol, Src Addr: 130.237.15.247 (130.237.15.247), Dst Addr: 130.237.203.11 (130.237.203.11)
⊞ User Datagram Protocol, Src Port: 5060 (5060), Dst Port: 5060 (5060)
⊟ Session Initiation Protocol
  ⊟ Request line: INVITE sip:cisco1@130.237.15.222;user=phone SIP/2.0
        Method: INVITE
  ⊟ Message Header
        From: <sip:torobravo7@sip1.it.kth.se;user=phone>;tag=1544939104
        To: <sip:cisco1@130.237.15.222;user=phone>
        Call-ID: 560912238@130.237.15.247
        CSeq: 801 INVITE
        Contact: <sip:torobravo7@130.237.15.247:5060;user=phone;transport=UDP>;expires=1000
        User-Agent: Minisip
        Content-Type: application/sdp
        Via: SIP/2.0/UDP 130.237.15.247:5060;branch=z9hG4bK649122006
        Content-Length: 142
⊞ Session Description Protocol
```

```
0000  00 00 0c 07 ac 67 00 0b  db 5c b1 d6 08 00 45 00   .....g.. .\....E.
0010  02 5f 00 00 40 00 40 11  57 b1 82 ed 0f f7 82 ed   ._..@.@. W.......
0020  cb 0b 13 c4 13 c4 02 4b  a6 94 49 4e 56 49 54 45   .......K ..INVITE
0030  20 73 69 70 3a 63 69 73  63 6f 31 40 31 33 30 2e    sip:cis co1@130.
0040  32 33 37 2e 31 35 2e 32  32 32 3b 75 73 65 72 3d   237.15.2 22;user=
```

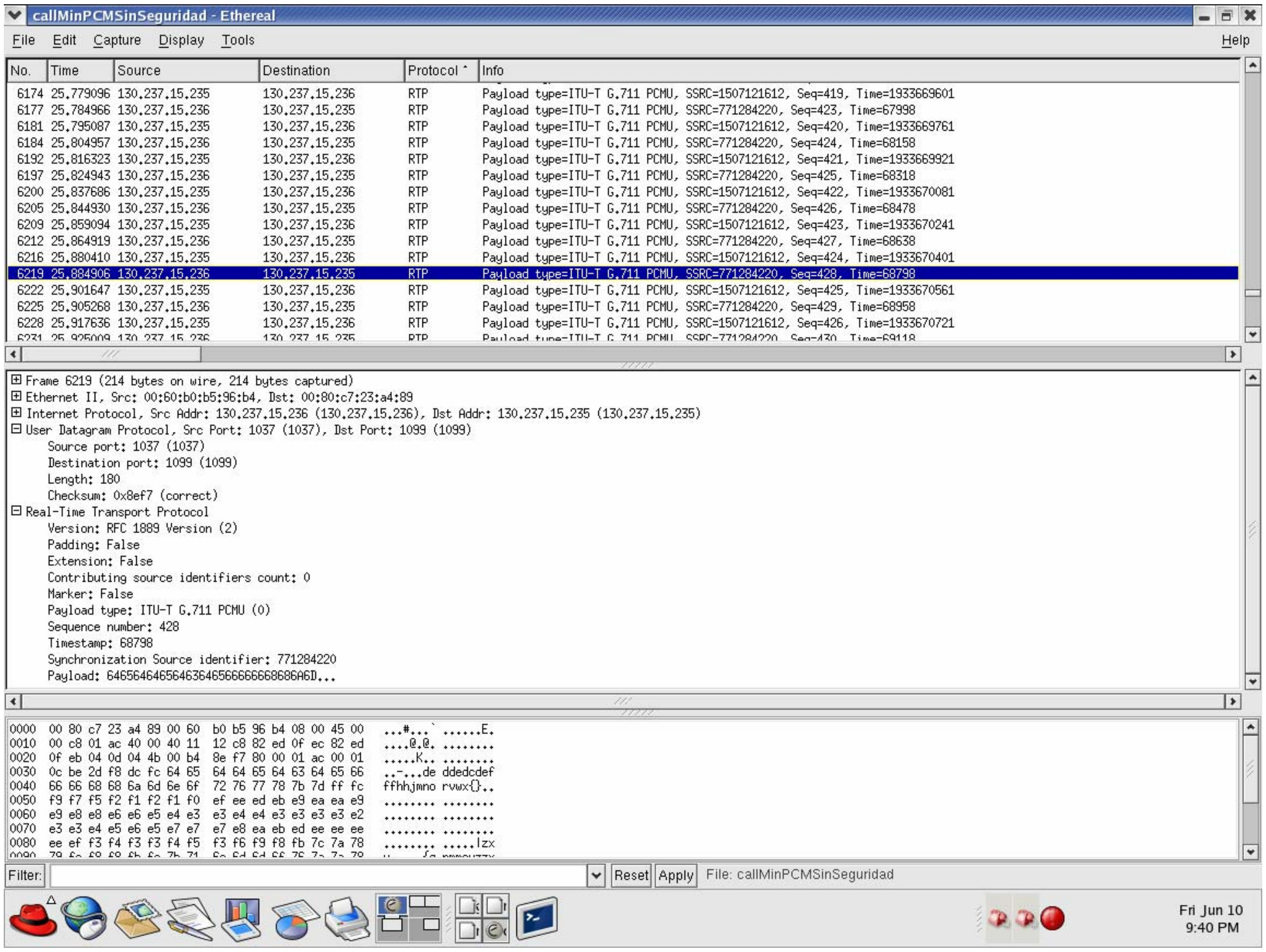
Figure 44. Capture of the call establishment in Minisip

From the previous packets it can be seen that the time required to complete the call establishment is 0,56 sec. approximately and the number of bytes is 3710.

7.2.3. Media transfer

The different packet capture session enabled me to see that the voice is carried over UDP and RTP. The user can select the port used to send the voice data directly from the GUI using the dialog shown in Figure 26.

In Minisip the size of the packets will depend on the encoding where are using and on the level the security. So it is necessary to consider these situations separately. I will start with the case of using the codec PCMu without security. One of the captures performed with this codec is shown in figure below:



callMinPCMSinSeguridad - Ethereal

```
No.   Time       Source          Destination     Protocol  Info
6174  25.779096  130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=1507121612, Seq=419, Time=1933669601
6177  25.784966  130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=771284220, Seq=423, Time=67998
6181  25.795087  130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=1507121612, Seq=420, Time=1933669761
6184  25.804957  130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=771284220, Seq=424, Time=68158
6192  25.816323  130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=1507121612, Seq=421, Time=1933669921
6197  25.824943  130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=771284220, Seq=425, Time=68318
6200  25.837686  130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=1507121612, Seq=422, Time=1933670081
6205  25.844930  130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=771284220, Seq=426, Time=68478
6209  25.859094  130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=1507121612, Seq=423, Time=1933670241
6212  25.864919  130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=771284220, Seq=427, Time=68638
6216  25.880410  130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=1507121612, Seq=424, Time=1933670401
6219  25.884906  130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=771284220, Seq=428, Time=68798
6222  25.901647  130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=1507121612, Seq=425, Time=1933670561
6225  25.905268  130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=771284220, Seq=429, Time=68958
6228  25.917636  130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=1507121612, Seq=426, Time=1933670721
```

```
⊞ Frame 6219 (214 bytes on wire, 214 bytes captured)
⊞ Ethernet II, Src: 00:60:b0:b5:96:b4, Dst: 00:80:c7:23:a4:89
⊞ Internet Protocol, Src Addr: 130.237.15.236 (130.237.15.236), Dst Addr: 130.237.15.235 (130.237.15.235)
⊟ User Datagram Protocol, Src Port: 1037 (1037), Dst Port: 1099 (1099)
    Source port: 1037 (1037)
    Destination port: 1099 (1099)
    Length: 180
    Checksum: 0x8ef7 (correct)
⊟ Real-Time Transport Protocol
    Version: RFC 1889 Version (2)
    Padding: False
    Extension: False
    Contributing source identifiers count: 0
    Marker: False
    Payload type: ITU-T G.711 PCMU (0)
    Sequence number: 428
    Timestamp: 68798
    Synchronization Source identifier: 771284220
    Payload: 6465646465646364656666668686A6D...
```

```
0000  00 80 c7 23 a4 89 00 60  b0 b5 96 b4 08 00 45 00   ...#...` ......E.
0010  00 c8 01 ac 40 00 40 11  12 c8 82 ed 0f ec 82 ed   ....@.@. ........
0020  0f eb 04 0d 04 4b 00 b4  8e f7 80 00 01 ac 00 01   .....K.. ........
0030  0c be 2d f8 dc fc 64 65  64 64 65 64 63 64 65 66   ..-...de ddedcdef
0040  66 66 68 68 6a 6d 6e 6f  72 76 77 78 7b 7d ff fc   ffhhjmno rvwx{}..
0050  f9 f7 f5 f2 f1 f2 f1 f0  ef ee ed eb e9 ea ea e9   ........ ........
0060  e9 e8 e8 e6 e6 e5 e4 e3  e3 e4 e4 e3 e3 e3 e3 e2   ........ ........
0070  e3 e3 e4 e5 e6 e5 e7 e7  e7 e8 ea eb ed ee ee ee   ........ ........
0080  ee ef f3 f4 f3 f3 f4 f5  f3 f6 f9 f8 fb 7c 7a 78   ........ .....|zx
```

Figure 45. Capture of the media traffic using PCMu without security

We can see that the size of the packet is 180 bytes for the RTP packets. However the real size of data is 168 bytes. Moreover if the users do **not** speak the size of the packets is the same so there is not a voice activity detection mechanism.

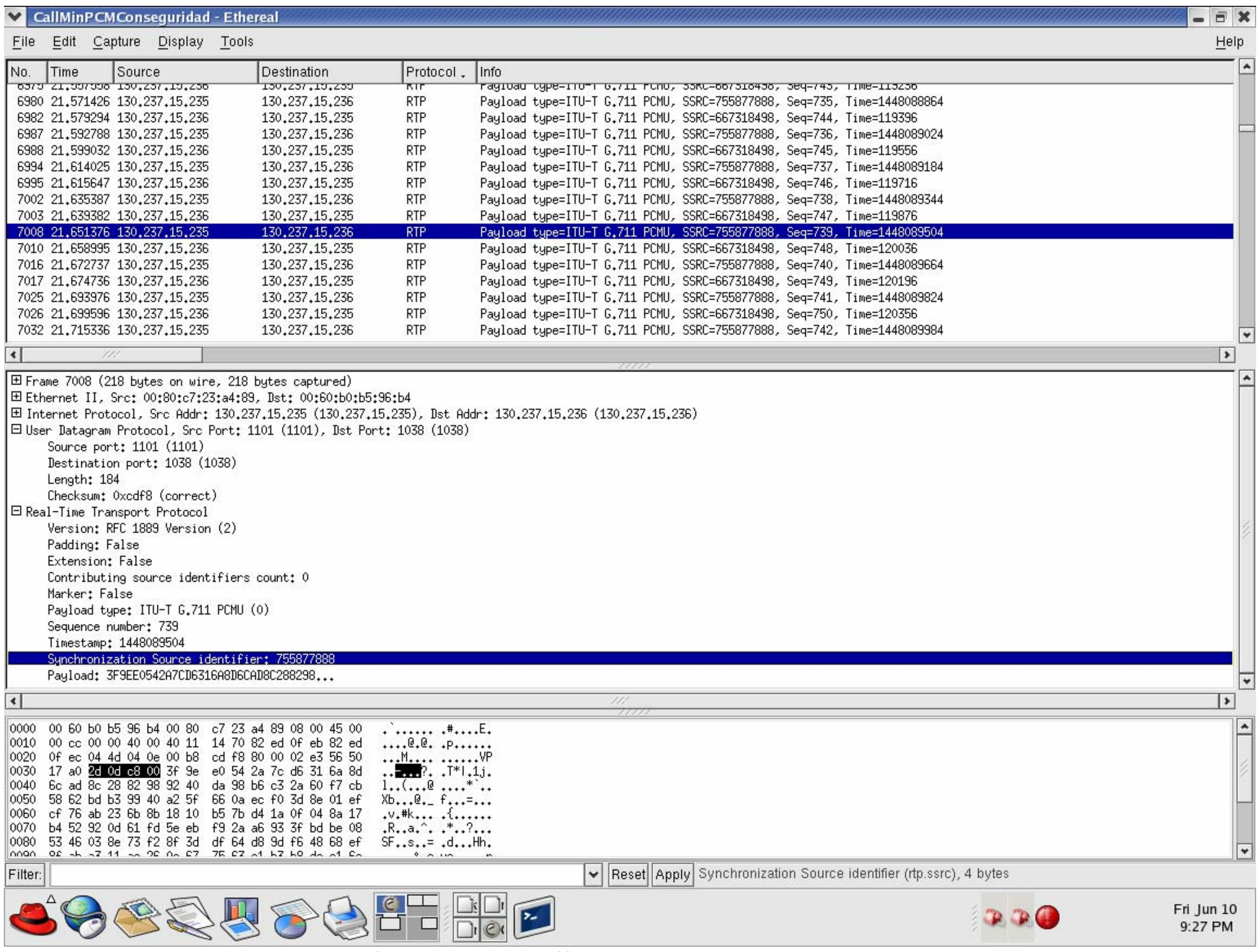
To calculate the bandwidth used in voice transmission we just need to measure the time between the sending of two packets and divide the size of the packet in bits by the time previously calculated. In average this value is 64000 bits per seconds since we are using PCMu encoding.

130.237.15.236 $\longrightarrow$ 130.237.15.235

Relative time to the first packet	Time difference between packets
25,804957	--
25,824943	0,019986
25,844930	0,019987
25,874919	0,019989
25,884906	0,019999
25,905268	0,020362

So we can see that the average time is approximately 20 msec.

If security is also included by enabling the Pre-Shared Key Agreement and using secure outgoing calls, we see that the size of the packets is increased by 4 bytes which is as expected [16]. The situation is seen in the following figure:


CallMinPCMConseguridad - Ethereal
File Edit Capture Display Tools
Help

```
No.  Time      Source          Destination     Protocol  Info
6980 21.571426 130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=755877888, Seq=735, Time=1448088864
6982 21.579294 130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=667318498, Seq=744, Time=119396
6987 21.592788 130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=755877888, Seq=736, Time=1448089024
6988 21.599032 130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=667318498, Seq=745, Time=119556
6994 21.614025 130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=755877888, Seq=737, Time=1448089184
6995 21.615647 130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=667318498, Seq=746, Time=119716
7002 21.635387 130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=755877888, Seq=738, Time=1448089344
7003 21.639382 130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=667318498, Seq=747, Time=119876
7008 21.651376 130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=755877888, Seq=739, Time=1448089504
7010 21.658995 130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=667318498, Seq=748, Time=120036
7016 21.672737 130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=755877888, Seq=740, Time=1448089664
7017 21.674736 130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=667318498, Seq=749, Time=120196
7025 21.693976 130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=755877888, Seq=741, Time=1448089824
7026 21.699596 130.237.15.236  130.237.15.235  RTP       Payload type=ITU-T G.711 PCMU, SSRC=667318498, Seq=750, Time=120356
7032 21.715336 130.237.15.235  130.237.15.236  RTP       Payload type=ITU-T G.711 PCMU, SSRC=755877888, Seq=742, Time=1448089984
```

```
⊞ Frame 7008 (218 bytes on wire, 218 bytes captured)
⊞ Ethernet II, Src: 00:80:c7:23:a4:89, Dst: 00:60:b0:b5:96:b4
⊞ Internet Protocol, Src Addr: 130.237.15.235 (130.237.15.235), Dst Addr: 130.237.15.236 (130.237.15.236)
⊟ User Datagram Protocol, Src Port: 1101 (1101), Dst Port: 1038 (1038)
    Source port: 1101 (1101)
    Destination port: 1038 (1038)
    Length: 184
    Checksum: 0xcdf8 (correct)
⊟ Real-Time Transport Protocol
    Version: RFC 1889 Version (2)
    Padding: False
    Extension: False
    Contributing source identifiers count: 0
    Marker: False
    Payload type: ITU-T G.711 PCMU (0)
    Sequence number: 739
    Timestamp: 1448089504
    Synchronization Source identifier: 755877888
    Payload: 3F9EE0542A7CD6316A8D6CAD8C288298...
```

```
0000  00 60 b0 b5 96 b4 00 80  c7 23 a4 89 08 00 45 00   .`...... .#....E.
0010  00 cc 00 00 40 00 40 11  14 70 82 ed 0f eb 82 ed   ....@.@. .p......
0020  0f ec 04 4d 04 0e 00 b8  cd f8 80 00 02 e3 56 50   ...M.... ......VP
0030  17 a0 2d 0d c8 00 3f 9e  e0 54 2a 7c d6 31 6a 8d   ..-...?. .T*|.1j.
0040  6c ad 8c 28 82 98 92 40  da 98 b6 c3 2a 60 f7 cb   l..(...@ ....*`..
0050  58 62 bd b3 99 40 a2 5f  66 0a ec f0 3d 8e 01 ef   Xb...@._ f...=...
0060  cf 76 ab 23 6b 8b 18 10  b5 7b d4 1a 0f 04 8a 17   .v.#k... .{......
0070  b4 52 92 0d 61 fd 5e eb  f9 2a a6 93 3f bd be 08   .R..a.^. .*..?...
0080  53 46 03 8e 73 f2 8f 3d  df 64 d8 9d f6 48 68 ef   SF..s..= .d...Hh.
```

Figure 46. Capture of the media traffic with PCMu and security

130.237.15.236 $\longrightarrow$ 130.237.15.235

Relative time to the first packet	Time difference between packets
21,579294	--
21,599032	0,019738
21,615647	0,016615
21,639382	0,02375
21,658995	0,019613
21,674736	0,015741

In this case data size is 172 bytes and the average time is same. And as in the previous case if the users do **not** speak the size of the packets is the same so there is not a voice activity detection mechanism.

Another codec is iLBC, and the results for this codec without security are shown in the table.

130.237.15.236 $\longrightarrow$ 130.237.15.235

Relative time to the first packet	Time difference between packets
10,496575	--
10,519311	0,022736
10,540923	0,021612
10,562036	0,021113
10,582647	0,020611
10,603510	0,020863

These values correspond to the capture shown in the Figure 47.

In this case we see that the average time continues being the same while the data size is 38 bytes RTP (or 50 bytes of UDP payload). In this situation the bandwidth is:

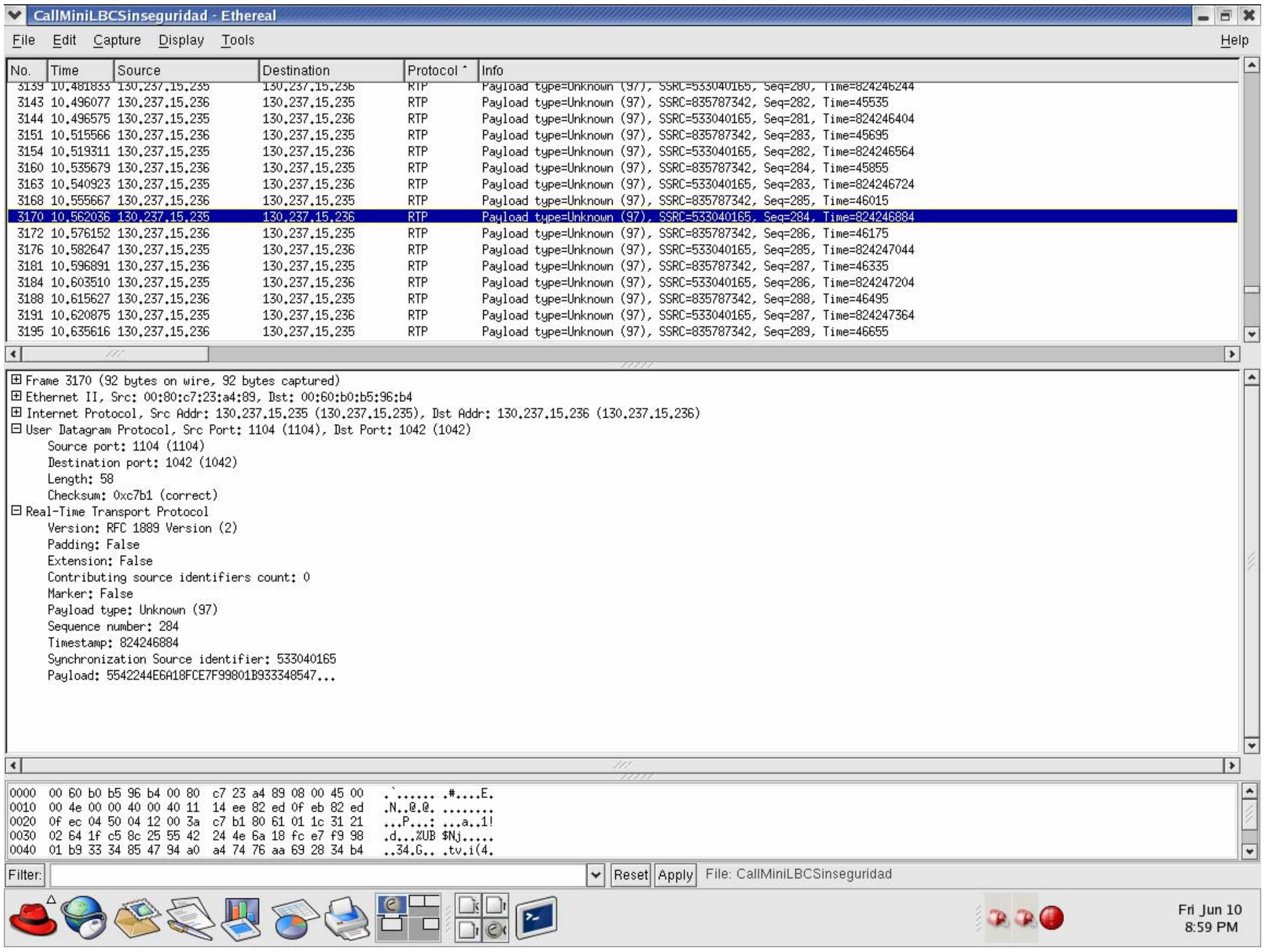
BW_{RTP} = 15200 bps.

BW_{UDP} = 20000 bps.

The last codec measurement is for iLBC including the same security that was also used with PCMu. The table showing the results is below.

130.237.15.236 $\longrightarrow$ 130.237.15.235

Relative time to the first packet	Time difference between packets
11,856495	--
11,876481	0,019986
11,896468	0,019987
11,916458	0,01999
11,936444	0,019986
11,956432	0,019988


CallMiniLBCSinseguridad - Ethereal
File Edit Capture Display Tools
Help
Filter:
Reset
Apply
File: CallMiniLBCSinseguridad
Fri Jun 10
8:59 PM

```
No.  Time      Source         Destination    Protocol Info
3139 10.481833 130.237.15.235 130.237.15.236 RTP      Payload type=Unknown (97), SSRC=533040165, Seq=280, Time=824246244
3143 10.496077 130.237.15.236 130.237.15.235 RTP      Payload type=Unknown (97), SSRC=835787342, Seq=282, Time=45535
3144 10.496575 130.237.15.235 130.237.15.236 RTP      Payload type=Unknown (97), SSRC=533040165, Seq=281, Time=824246404
3151 10.515566 130.237.15.236 130.237.15.235 RTP      Payload type=Unknown (97), SSRC=835787342, Seq=283, Time=45695
3154 10.519311 130.237.15.235 130.237.15.236 RTP      Payload type=Unknown (97), SSRC=533040165, Seq=282, Time=824246564
3160 10.535679 130.237.15.236 130.237.15.235 RTP      Payload type=Unknown (97), SSRC=835787342, Seq=284, Time=45855
3163 10.540923 130.237.15.235 130.237.15.236 RTP      Payload type=Unknown (97), SSRC=533040165, Seq=283, Time=824246724
3168 10.555667 130.237.15.236 130.237.15.235 RTP      Payload type=Unknown (97), SSRC=835787342, Seq=285, Time=46015
3170 10.562036 130.237.15.235 130.237.15.236 RTP      Payload type=Unknown (97), SSRC=533040165, Seq=284, Time=824246884
3172 10.576152 130.237.15.236 130.237.15.235 RTP      Payload type=Unknown (97), SSRC=835787342, Seq=286, Time=46175
3176 10.582647 130.237.15.235 130.237.15.236 RTP      Payload type=Unknown (97), SSRC=533040165, Seq=285, Time=824247044
3181 10.596891 130.237.15.236 130.237.15.235 RTP      Payload type=Unknown (97), SSRC=835787342, Seq=287, Time=46335
3184 10.603510 130.237.15.235 130.237.15.236 RTP      Payload type=Unknown (97), SSRC=533040165, Seq=286, Time=824247204
3188 10.615627 130.237.15.236 130.237.15.235 RTP      Payload type=Unknown (97), SSRC=835787342, Seq=288, Time=46495
3191 10.620875 130.237.15.235 130.237.15.236 RTP      Payload type=Unknown (97), SSRC=533040165, Seq=287, Time=824247364
3195 10.635616 130.237.15.236 130.237.15.235 RTP      Payload type=Unknown (97), SSRC=835787342, Seq=289, Time=46655
```

```
⊞ Frame 3170 (92 bytes on wire, 92 bytes captured)
⊞ Ethernet II, Src: 00:80:c7:23:a4:89, Dst: 00:60:b0:b5:96:b4
⊞ Internet Protocol, Src Addr: 130.237.15.235 (130.237.15.235), Dst Addr: 130.237.15.236 (130.237.15.236)
⊟ User Datagram Protocol, Src Port: 1104 (1104), Dst Port: 1042 (1042)
    Source port: 1104 (1104)
    Destination port: 1042 (1042)
    Length: 58
    Checksum: 0xc7b1 (correct)
⊟ Real-Time Transport Protocol
    Version: RFC 1889 Version (2)
    Padding: False
    Extension: False
    Contributing source identifiers count: 0
    Marker: False
    Payload type: Unknown (97)
    Sequence number: 284
    Timestamp: 824246884
    Synchronization Source identifier: 533040165
    Payload: 5542244E6A18FCE7F99801B933348547...
```

```
0000  00 60 b0 b5 96 b4 00 80  c7 23 a4 89 08 00 45 00   .`...... .#....E.
0010  00 4e 00 00 40 00 40 11  14 ee 82 ed 0f eb 82 ed   .N..@.@. ........
0020  0f ec 04 50 04 12 00 3a  c7 b1 80 61 01 1c 31 21   ...P...: ...a..1!
0030  02 64 1f c5 8c 25 55 42  24 4e 6a 18 fc e7 f9 98   .d...%UB $Nj.....
0040  01 b9 33 34 85 47 94 a0  a4 74 76 aa 69 28 34 b4   ..34.G.. .tv.i(4.
```

Figure 47. Media traffic corresponding to iLBC codec and without security

The capture enables to see that the packet size is 42 bytes for RTP and 54 for UDP. So, security adds 4 bytes in each packet.

The bandwidths are approximately:
BW_{RTP} = 16800 bps.
BW_{UDP} = 21600 bps.

Concerning the 16-bit linear encoding, I faced one problem. It is that the traffic capture could not be done since the last version of Minisip did not allow including this codec.

The next step would be the calculation of jitter and delay for the communications with different codecs. However, there was another problem, valid for all codecs. The communication in one direction presented a reasonable delay, whereas the delay in the other direction was huge. And this case was not able to be explained by the developers neither. That is why, I did not calculate the delay and jitter.

I also calculated the bandwidth utilized during the tear down of the calls for the four cases above. I found that regardless of security the traffic is the same for a given codec. This is as expected since SIP messages are not protected. The traffic values are in the table below.

Codec used	BYE message and 200 OK, Time
PCMu	275 bytes, 382 bytes, 0,016 sec
iLBC	500 bytes, 255 bytes, 0,016 sec

Finally I will say that capture sessions were perform to calculate the traffic sent for the Minisip users when there is not communication. All these captures showed that there is no traffic.

7.3. Packet level comparison

In the login process the minimum number of bytes that the user exchanges is 3293 and it increases depending on the number of contacts to be retrieved, whereas in the case of Minisip is 2540 bytes. We can see that there is an important difference in bytes.

Moreover, there is an important difference in the time needed to finish the login process. In the Skype is observed that this time varies from 6 to 9 seconds. This is due to the search of the supernodes and the information to be downloaded from them (buddylist and list of contacts). In the case of Minisip the login process is done in 31.1 msec.

With respect to the call establishment process there are also several differences. First, in Skype this process requires UDP and TCP traffic, whereas in Minisip we find only SIP traffic over UDP. Second, in Skype if the user to be contacted is in our list of contacts, the caller transmits in average 2723,75 bytes in a period of time of 0,19 seconds. However, if the callee is not in our list of contacts, then the time and the number of bytes increases since it is necessary to communicate with our supernode to perform a search of this contact. In the case of Minisip the data transferred is higher than in Skype and the time could be higher if the user to be called is in our list.

In reference to the media communication I see that in Skype the size packet is not fixed and it varies, whereas in Minisip for PCMu and iLBC is fixed. It is also observed that the bandwidth used in Minisip is lower when the audio is encoded with iLBC than in Skype (the half approximately), but if the codec selected is PCMu, then the bandwidth required in Minisip is the double. Then we can see that the bandwidth required depends on the codec selected and that if we are able to introduce the same codec in both programs the bandwidth obtained will be the same.

In the case of the tear down of the call two packets are sent in both programs. However, the number of bytes in the case of Skype is lower than in Minisip. In the case of Skype these two packets are carried over TCP whereas in Minisip correspond to the BYE and 200 OK packets. Besides, the time for the tear down the call is lower in Minisip than in Skype.

Additional results are that both programs do not include voice activity detection mechanisms, that the bandwidth utilized in Minisip when there is not communication is null while in Skype it varies from 0 to 0.5 kbps, and that including security in the communication adds four bytes.

8 Conclusions

The general conclusion in my master thesis is that there are two initial ideas or targets completely different in both products. These two ideas define and explain the other results and conclusions obtained in my work.

The main aim of Skype is to create a VoIP program which provide to the company incomes. In consequence, it is necessary to develop a product available to as many users as possible by providing software version of this product for the most important operating system, that is, Microsoft Windows, Microsoft Pocket Pc, Linux, and Macintosh. In addition, since the clients pay for a service (i.e. SkypeOut, SkypeIn, VoiceMail, ...), a complete information and support must be ready to gain the trust of the clients. Actually, this explains the different sources of information included in the Skype website as well as the complete definition of the hardware and software requirements and the steps to follow when installing the corresponding software version. It is also very important to create a solid program to avoid problems to the users.

Of course, it is desired and needed simple operation with the program. For this reason parameters like the codec to be used in the communication or the security level are fixed in the program. The current Skype version utilizes a broadband encoding of the audio so that the frequencies upper to 16 KHz. will be filtered and the security is provided by using two cryptographic algorithms. Initially, RSA is used to protect the negotiation of the 256-bit symmetric key and later the AES algorithm encrypts the rest of the communications.

Moreover, we can see that Skype focuses development in the communication via text, audio and also sending files. However they do not explore the field of the videoconference. In fact, other services announced in their website and in their journal continue in the field of the telephony. Basic knowledge of the operations of the market, especially the technological market, shows us that the development of a product for large number of clients requires a huge effort in terms of time and money. Furthermore, this enterprise wants to sell its product for sure, but can introduce it in the market, even, without being completely sure about its correct operation.

On the other hand Minisip is a prototype in continuous development. It pretends to study the new ways of communication and services that the IP telephony allows to emphasize the possible advantages and the problems. It also has the main source of developers in students performing their master thesis at KTH in Stockholm.

Therefore Minisip, at least by the moment, is not oriented to normal users, but it is focused in the investigation and development of new applications. This different aim permits to understand that Minisip is covering a wider number of communications means than Skype, including videoconference with one or more users as well as variation of the encoding of the audio, or different levels of security in the communication. It is also possible to vary the mechanism to control the phone conferences, the definition of different audio qualities by selecting different codecs in the different links of the communication (which is possible since we are using SIP). It also allows sending different flows of audio over the same link of and locating in different virtual spatial positions and even utilizing the audio-to-text and text-to-audio mechanisms decreasing dramatically the bandwidth for a given quality. Then the Minisip users have a greater number of parameters to vary so that it is possible to understand how these parameters affect the communication.

However, since Minisip is a prototype with an investigational target the structure of support to the clients does not have the structure of the Skype support, the number of operating systems in which it is available is lower (only Linux versions although the translation to Windows has been started). There is no definition of the hardware requirements and the only stable version is on Debian. As a consequence, the Minisip users must have a greater knowledge to understand the advantages of using one or the other options when establishing the communication and also to solve more complex problems without much information.

The following figure aggregates the results providing an easy way to compare the both soft wares.

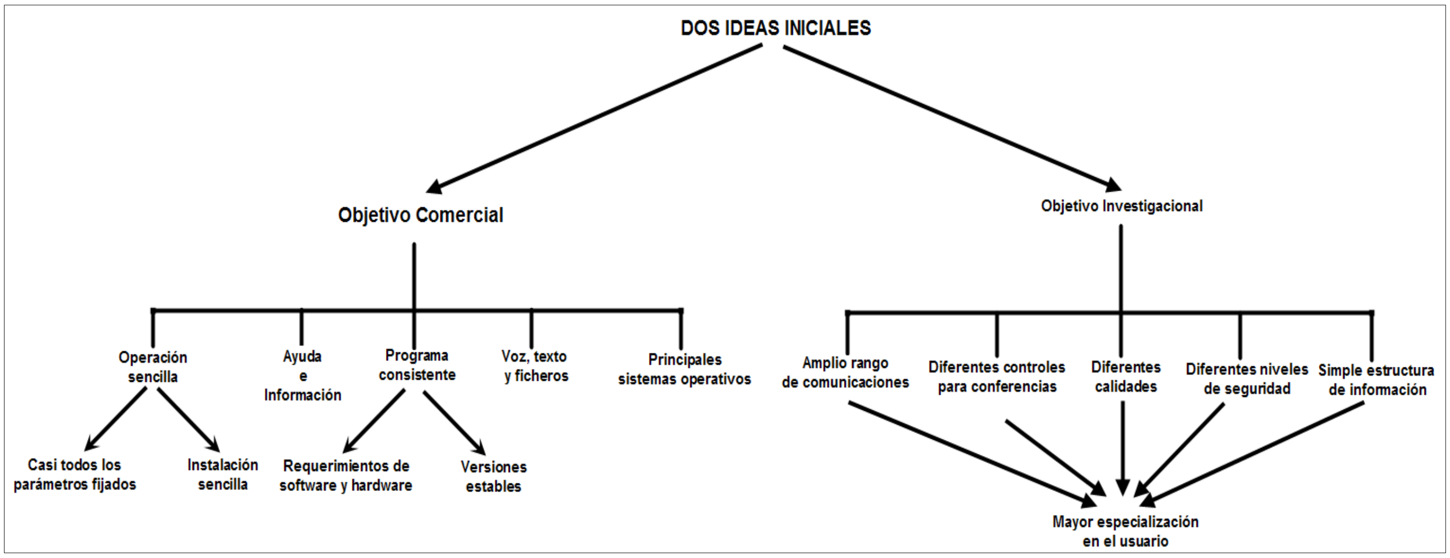

DOS IDEAS INICIALES
Objetivo Comercial
Objetivo Investigacional
Operación sencilla
Ayuda e Información
Programa consistente
Voz, texto y ficheros
Principales sistemas operativos
Casi todos los parámetros fijados
Instalación sencilla
Requerimientos de software y hardware
Versiones estables
Amplio rango de comunicaciones
Diferentes controles para conferencias
Diferentes calidades
Diferentes niveles de seguridad
Simple estructura de información
Mayor especialización en el usuario

Figure 48. Summary of the results

9 Future work

The work I consider interesting to do in the future is commented in the following points:

- First, it is necessary to finish the translation of Minisip providing a Windows version even compatible with Pocket PC. In that situation, the more people will be covered increasing the number of developers and improving the usability of the program.

- Once the Windows version is available it would be interesting to introduce a connection to Skype contacts in Minisip´s graphical interface. Then when the user wants to call one of his Skype contacts, Minisip will connect with Skype by using the Skype API (for Microsoft Windows and Linux) so that we could initiate the communication using the Skype resources.

- It is also interesting to perform the comparison of both VoIP programs over the same platform. Then it is necessary to connect our common graphical interface with the Windows version of Minisip using the textual interface available for Minisip. As a consequence the connection would be straight and it would be possible to compare the new results with the results obtained in this master thesis.

- Finally, the last work I suggest is to decrypt the Skype protocol by following the ideas indicated in this document in Appendix C. Then we will be able to know completely the messages exchanged between the client and the servers in Skype and to obtain advantages that could be introduced in Minisip. And even more, it could be possible to use Skype network resources.

10 References

[1] Rakesh Arora, “Voice over IP: Protocols and standards" http://www.cse.ohio-state.edu/~jain/ cis788-99/ftp/voip_protocols/index.html#intro. Last accessed on 2005.05.02.

[2] José Manuel Huidobro, “La telefonía sobre IP”, http://www.monografias.com/trabajos10/ tele/tele.shtml. Last accessed on 2005.05.02.

[3] Marco A. Escobedo and Michael L. Best, ”Convivo comminicator: An interface-adaptative VoIP systems for poor quality networks”, http://xenia.media.mit.edu/~marcoe/convivo_paper_2003.pdf. Last accessed on 2005.05.15.

[4] O. Levin, April 2003, ”RFC 3508-H.323 Uniform Resource Locator (URL) Scheme Registration”, http://www.faqs.org/rfcs/rfc3508.html. Last accessed on 2005.05.15.

[5] H.323. A premier on the H.323 Series Standard, http://www.packetizer.com/voip/h323/ papers/primer/. Last accessed on 2005.05.02.

[6] Videoconferencia, una herramienta para aumentar la proctividad en la empresa de hoy: H.323 o H.320, http://www.comunicaciones.unitronics.es/tecnologia/H.323.html. Last accessed on 2005.05.02.

[7] Jose Espinosa, “VoIP: Presente y futuro de las comunicaciones de voz”, http://www.aui.es/ biblio/libros/mi99/19voz_ip.htm, last accessed on 2005.05.02.

[8] H.225: Call Signaling and RAS in H.323 VOIP Architecture, http://www.javvin.com/ protocolH225.html, last accessed on 2005.05.02.

[9] Client/server architecture16.20.35 http://www.webopedia.com/TERM/C/client_server_architecture.htm. Last accessed on 2005.05.15.

[10] H.245: Control Protocol for Multimedia Communication http://www.javvin.com/ protocolH245.html, last accessed on 2005.05.02.

[11] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, “SIP: Session Initiation Protocol”, March 1999, 132 pages, IETF RFC 2543, ftp:// ftp.isi.edu/in-notes/rfc2543.txt

[12] Real time Transport Protocol, http://searchnetworking.techtarget.com/sDefinition/ 0,,sid7_gci812019,00.html, last accessed on 2005.05.05.

[13] Ville Hallivouri, Helsinki University of Technology, ”Real time Transport Protocol Security”, http://www.tml.hut.fi/Opinnot/Tik-110.501/2000/papers/hallivuori.pdf, last accessed on 2005.05.02.

[14] R.Braden, L.Zhang, S.Berson, S. Herzog, S. Jamin, September 1997, “Resource ReSerVation Protocol”, IETF RFC 2205, http://www.ietf.org/rfc/rfc2205.txt

[15] Session Description Protocol, http://www.javvin.com/protocolSDP.html, last accessed on 2005.05.04.

[16] Johan Bilien, “Key agreement for Secure Voice over IP”, Master Thesis, KTH, Stockholm, December 2005. ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/031215-Johan-Bilien-report-final-with-cover.pdf

[17] Peer to peer, http://en.wikipedia.org/wiki/Peer-to-peer, last accessed on 2005.05.08.

[18] Napster, http://www.napster.com/

[19] Gnutella, http://www.gnutella.com/. Last accessed on 2005.05.18.

[20] FastTrack, http://www.fasttrackdcn.net/ . Last accessed on 2005.05.18.

[21] Skype, www.skype.com , last accessed on 2005.04.26.
[22] Skype. http://en.wikipedia.org/wiki/Skype. Last accessed on 2005.05.10
[23] Open.Net at KTH, Open Comminication Network Architectures, http://www.nsrc.se/Open-brochure.pdf , last accessed on 2005.05.10
[24] Minisip, http://www.voip-info.org/wiki-minisip, last accessed on 2005.05.10.
[25] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman, March 2004, "The Secure Real-time Transport Protocol (SRTP)", IETF RFC, www.ietf.org
[26] J. Arkko, E. Carrara, F. Lindholm, M. Naslund, K. Norrman, August 2004, "MIKEY: Multimedia Internet KEYing", IETF RFC, www.ietf.org
[27] Advanced Encryption Standard, http://en.wikipedia.org/wiki/AES, last accessed on 2005.05.06.
[28] Tom Davis, "RSA Encryption", October 10, 2003, http://www.geometer.org/mathcircles/ RSA.pdf, last accessed on 2005.05.08.
[29] Francis Litterio,"The Mathematical Guts of RSA Encryption", http://world.std.com/~franl/ crypto/rsa-guts.html, last accessed on 2005.05.08.
[30] Francis Litterio, "An example of the RSA Encryption", http://world.std.com/~franl/crypto/ rsa-example.html, last accessed on 2005.05.08.
[31] Luciano Moreno, "Criptografía (X)", http://www.htmlweb.net/seguridad/cripto/ cripto_10.html, last accessed on 2005.05.05.
[32] SkypeOut Rates, http://www.skype.com/products/skypeout/rates/, last time accessed on 2005.04.26.
[33] iLBC White Paper, October 15, 2004, "iLBC – Designed For The Future", http://www.packetizer.com/codecs/ilbc/iLBC.WP.pdf. Last accessed on 2005.05.17.
[34] Olof Hagsand, "The application layer", NADA/KTH, olofh@nada, January 20, 2005, http://www.nada.kth.se/kurser/kth/2D1490/05/lectures/lecture4/applications.pdf. Last accessed on 2005.05.17.
[35] Handybits voicemail, http://www.handybits.com/voicemail.htm, last time accessed on 2005.03.21.
[36] Messaging Application Programming Interface, http://project.uet.itgo.com/mapi.htm, last time accessed on 2005.04.27
[37] Skype Journal, http://www.skypejournal.com/. Last accessed on 2005.05.18.
[38] Minisip, http://www.minisip.org, last time accessed on 2005.04.22.
[39] Inmaculada Rangel Vacas, "Context Aware and Adaptative Mobile Audio", Royal Institute of Technology (KTH), Semester Project performed during September 2004 - March 2005. ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/050418-Inmaculada-Rangel-Vacas.pdf. Last accessed on 2005.04.26.
[40] Florian Maurer, "Push-2-talk Decentralized", Royal Institute of Technology (KTH), Semester Project performed during May-August 2004. http://push2talk.flohweb.ch/
[41] B. Campbell, R. Mahy, C. Jenning, August 24, 2005, "The Message Session Relay Protocol", http://www.ietf.org/internet-drafts/draft-ietf-simple-message-sessions-10.txt. Last accessed on 2005.05.18.
[42] RSA Laboratories, What is Diffie Hellman?, 2004, http://www.rsasecurity.com/rsalabs/ node.asp?id=2248, Last accessed on 2005.04.27.
[43] Andreas Ångström, "Peer to peer vs. SIP", Royal Institute of Technolgy (KTH), Master thesis, in preparation.
[44] Jan Linden, April 12th, 2005. "VoIP: Better than PSTN?", http://www.globalipsound.com/solutions/solutions_whiteprs.php. Last accessed on 2005.05.18.

[45] Ignacio Sanchez Pardo, "Spatial Audio for the Mobile User", Master Thesis, KTH, Stockholm, February 2005. ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/050307-Ignacio_Sanchez_Pardo-with-cover.pdf
[46] Messenger, http://messenger.msn.es/. Last accessed 2005.04.26.
[47] CuSeeMe, www.cu-seeme.com. Last accessed 2005.04.26.
[48] Microsoft NetMeeting, http://www.microsoft.com/windows/netmeeting/. Last accessed 2005.04.26.
[49] Digisip – Bredbandstelefoni, www.digisip.com. Last accessed on 2005.05.18.
[50] HotSIP. http://www.hotsip.com/. Last accessed on 2005.05.18.
[51] ActiveSync. http://www.microsoft.com/windowsmobile/downloads/activesync38.mspx. Last accessed on 2005.04.29.
[52] "GNU LGPL", http://es.wikipedia.org/wiki/LGPL. Last accessed on 2005.05.18.
[53] Bruce Perens, Sven Rudolph, Igor Grobmanm, James Treacy, and Adam Di Carlo, 18 December 2002, "Installing Debian GNU/Linux 3.0 for Intel x.86", http://www.debian.org/releases/stable/i386/install last accessed 2005.05.02
[54] Key agreement protocol. http://en.wikipedia.org/wiki/Key_agreement. Last accessed 2005.05.02
[55] What is pre-shared key or shared secret?. http://kb.indiana.edu/data/aodl.html?cust=641840.37921.30 .Last accessed 2005.05.02
[56] Glossary. http://docs.sun.com/source/816-6393-10/Glossary.html. Last accessed 2005.05.02.
[57] Certification authority. http://www.atis.org/tg2k/_certification_authority.html. Last accessed 2005.05.02.
[58] Transport Layer Security. http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci557332,00.html. Last accessed on 2005.05.02.
[59] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", IETF RFC 3489, http://www.ietf.org/rfcs/rfcs3489.html . Last accessed 2005.05.02.
[60] GloblalIPSound. http://www.globalipsound.com , last accessed 2005.04.29.
[61] Ethereal. http://www.ethereal.com/, Last accessed on 2005.05.03
[62] Richard Sharpe, Ed Warnicke, and Ulf Lampimg, "Ethereal user´s guide V2.00 for Ethereal 0.10.5", NS Computer Software and Services P/L, 2004. Last accessed on 2005.04.04.
[63] S.Baset, "The Skype Procol Analysis", Department of Columbia Science, Columbia university, New York last accessed 2005.04.04.
[64] "Matlab 7.0.4. The Language of Technical Computing", http://www.mathworks.com/products/matlab/. Last accessed on 2005.05.16.
[65] Joe Barr, Monday October 25, 2004, "CLI Magic: Looking for strings", http://enterprise.linux.com/enterprise/04/10/21/1240220.shtml?tid=47&tid=89. Last accessed on 2005.05.18.
[66] Microsoft, http://msdn.microsoft.com/. Last accessed on 2005.05.19
[67] Jeff Prosise, "Programming Windows With Mfc", 2th Edition, Last accessed on 2005.05.19.
[68] Douglas Boiling, "Programming Microsoft Windows CE .NET", Third Edition. Last accessed on 2005.05.19.
[69] Charles Petzold, "Programming Windows, 5th Edition". Last accessed on 2005.05.19.
[70] Taavet Hinrikus, SKYPE API "Description of Skype API and how to use it",

http://skype.com/community/devzone/Skype%20API%20description%201.2.pdf. Last accessed on 2005.05.19

[71] Freenet, http://www.freenet.co.uk/. Last accessed on 2005.05.24

[72] I2P, www.i2p.net Last accessed on 2005.05.24

[73] GNUnet, http://gnunet.org/. Last accessed on 2005.05.24

[74] Entropy, http://entropy.stop1984.com/ Last accessed on 2005.05.30

[75] CSD group 15, http://csd.ssvl.kth.se/~csd2005-team15/. Last accessed 2005.05.30

[76] Developer list, http://lists.minisip.org/pipermail/minisip-devel/. Last accessed on 2005.05.24

[77] Israel M. Abad Caballero master thesis, "Secure Mobile Voice over IP", Royal Institute of Technology (KTH), ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/030626-Israel_Abad_Caballero-final-report.pdf. Last accessed on 2005.05.30

[78] Judah Levine, "Nist Internet Time service", http://tf.nist.gov/service/its.htm. Last accessed on 2005.05.30

[79] Bilge Cetin, Elena Martín, Guang Liang, Max Loubser, and Hanning Zhang", CSD group 14, http://csd.ssvl.kth.se/~csd2005-team14/. Last accessed 2005.05.30

Appendix A Study of the Skype protocol

A.1 Experiment setup

The experiments were performed with two versions of Skype: 1.2.0.37 and 1.1.0.79. Skype was installed on my laptop under Windows XP and on two more machines running RedHat Linux 9. The Windows machine was an AMD Athlon 2500 with 512MB memory while the others machines were Dell Power Server XX. Each machine has a 100 Mbps Ethernet card and the Windows machine also has a wireless network interface.

Experiments were performed in only one network situation in which all the Skype users were on at public IP addresses.

Ethereal was used to monitor the network traffic.

A.2 Functions

Skype functions can be classified as startup, login, user search, connection establishment/ tear down, media transfer, and presence messages. The following study will discuss each of them in detail.

A.2.1 Startup

In the study performed at Columbia University [63] concerning the operation of the Skype protocol, they ascerted that on normal startup, Skype client (SC) sends a HTTP 1.1 GET request to the Skype server (skype.com) to determine if a new version is available. They also indicated that the request contains the keyword 'getlatestversion'.

However, in the measurements performed I verified this behavior depends on the machine on which Skype client is running. When the Skype client runs on Linux the GET message over the HTTP is sent. In the case of running of Windows, in the measurements which were made this message never was sent.

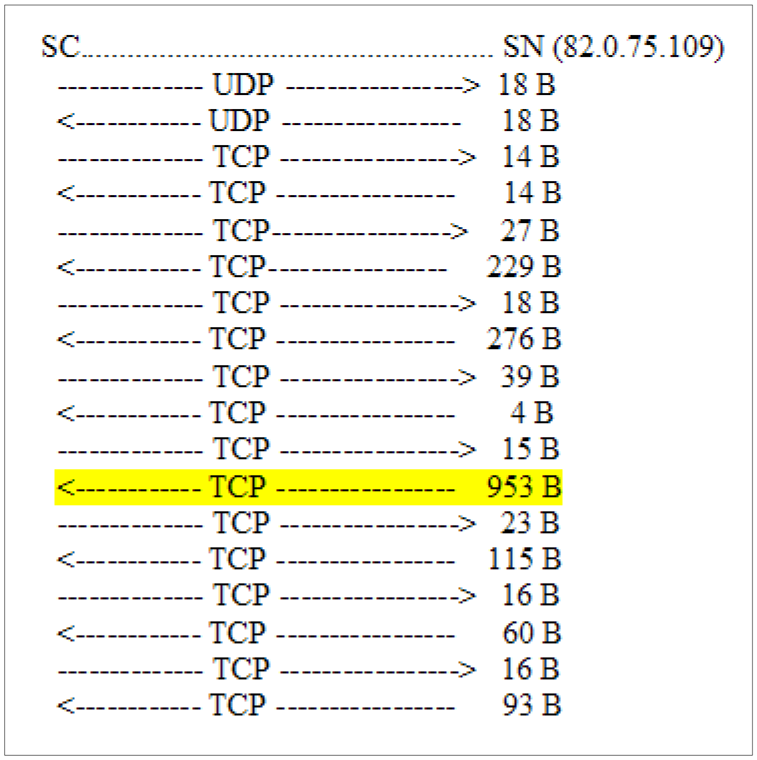
A.2.2 Login

The Baset [63] says “the login is the most critical function on the Skype operation. During this process SC advertises its presence to other peers and its buddies determine the type of NAT and firewall is behind, and discovers online Skype nodes with public IPs”.

It was verified that the Skype client creates a table after the first log to use later during the installation. This table is periodically updated with the IP address and port of the new peers. During subsequent logins, the SC sends UDP packets to the nodes stored in the table and it must receive a reply from at least one of them.

In any case, the Skype client must connect to a supernode to be able to login to the network. To do that, the Skype client (SC) initially sends UDP packets to different nodes, hopefully at least one of them will respond. Then the SC establishes a TCP connection with this supernode (SN), then they exchange some information over TCP which is assumed to be the address of one or more several login servers so that the client can try the connect with one of them.

The connection with the supernode follows one of two patterns depending on whether the Skype client is running on Windows or Linux. These are the patterns of exchanges that I found are in Figures 49 and 50.



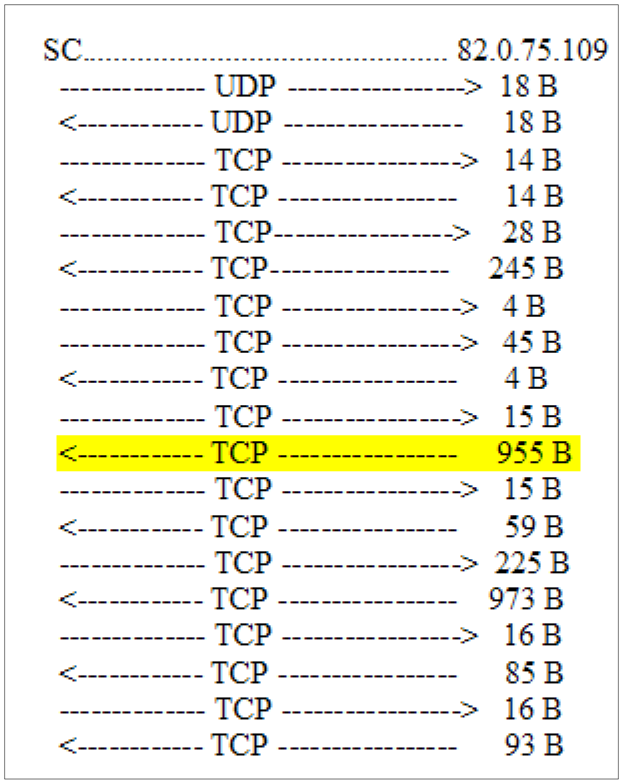
```
SC................................................ SN (82.0.75.109)
 ------------- UDP ---------------> 18 B
 <----------- UDP ----------------  18 B
 ------------- TCP ---------------> 14 B
 <----------- TCP ----------------  14 B
 ------------- TCP---------------> 27 B
 <----------- TCP----------------  229 B
 ------------- TCP ---------------> 18 B
 <----------- TCP ----------------  276 B
 ------------- TCP ---------------> 39 B
 <----------- TCP ----------------   4 B
 ------------- TCP ---------------> 15 B
 <----------- TCP ----------------  953 B
 ------------- TCP ---------------> 23 B
 <----------- TCP ----------------  115 B
 ------------- TCP ---------------> 16 B
 <----------- TCP ----------------  60 B
 ------------- TCP ---------------> 16 B
 <----------- TCP ----------------  93 B
```

Figure 49. Connection with the supernode in Linux

The emphasized frames show the last frame exchanged with the supernode (SN) before the TCP connection with the login server is made. It is possible to observe that there is a certain parallelism between both cases as was expected. But even more interesting is the difference in information exchanged between both these nodes (SC and SN) following this. Thus, in the case of the client running over Linux, the amount of information exchanged is much smaller. This is not because the client is running on Linux, but that accomplishment corresponds to the case of having a single contact in our list of contacts, whereas in the case of Windows we have approximately 20 contacts.

We are going to study in more detail the two transfer patterns with the goal of determining the basis for their differences. First, observing the last transference in both experiments I note that in both cases this transfer is equal and the explanation is these two transfers corresponds to the cancellation command done in both experiments from the caller before the call establishment was completed (i.e., a missed call). Second, we also can discard the previous transfer since it is due, if the user is using Linux, to the search for the callee

Skype user and, in the case of Windows, it is expected that the caller is informed about the path to follow in order to complete the call because the called Skype client was in the buddy list of the caller. In the first case, the Skype client did not have the client in his buddy list.



```
SC............................................ 82.0.75.109
 ------------- UDP ---------------> 18 B
 <------------ UDP ----------------  18 B
 ------------- TCP ---------------> 14 B
 <------------ TCP ----------------  14 B
 ------------- TCP---------------> 28 B
 <------------ TCP---------------- 245 B
 ------------- TCP ---------------> 4 B
 ------------- TCP ---------------> 45 B
 <------------ TCP ----------------  4 B
 ------------- TCP ---------------> 15 B
 <------------ TCP ---------------- 955 B
 ------------- TCP ---------------> 15 B
 <------------ TCP ----------------  59 B
 ------------- TCP ---------------> 225 B
 <------------ TCP ---------------- 973 B
 ------------- TCP ---------------> 16 B
 <------------ TCP ----------------  85 B
 ------------- TCP ---------------> 16 B
 <------------ TCP ----------------  93 B
```

Figure 50. Connection with the supernode in Windows

Thus, the real difference is in the remaining frames:

- Without the callee being in the buddy list of the caller:
- 23 Bytes sent.
- 115 Bytes received.
- With twenty users and the user in the buddy list:
- 240 Bytes sent.
- 1032 Bytes received.

It is conjectured that the Skype client receives these 1032 bytes containing the information about the users´ (contacts) who are connected (because the user will send to the connected contacts UDP messages some time later 101 bytes indicating he is online).

However, additional aspects were noticed. First of all, we must emphasize that the discovery of more than one centralized element in the Skype network can be interpreted as a modification of the previous structure. It was shown that whenever the users' login occurs, the connection with the supernode node is established as well as a connection with, at least, one other node which must provide the authentication information. The previous studies affirm that it is the server (with IP address 212.72.49.131 in the current version), which was called the login server, i.e., the machine which authenticates the new user. However, we see that the connection with this node is not present in all the logins (only in machines running Linux) so it **cannot** be the login's server.

Instead of connecting with that machine, a TCP connection was observed with one of the following three machines in all the measurements: 212.72.49.141, 195.215.8.141 or 212.72.49.142. In the packets captured in these measurements, these three centralized nodes correspond to servers, but it is very probable that even more servers exist (considering the number of Skype clients and that this number grows every day).

It also was observed than when the Skype client is running on Linux, the connection with 212.72.49.131 is established and the command GET is sent from the user to this node. In this situation we can assure that there are more than one centralized node in the network (which is logical as the number of users surpasses the 3 million users). Therefore, the network has evolved from a structure in which there was a single centralized element to a structure composed of several login servers in order to take care of the users which may contact them.

Once we have determined the node which must be contacted to provide authentication, we should study the information exchanged between the user and that node. The connection with the login server always occurs after the connection with the supernode was established. Therefore, I can propose that it is **within** the communication with the supernode that the Skype client obtains the address of the login server. It is also thought that redundancy is introduced in the communication between the Skype client (SC) and supernode (SN) which sends the addresses of several login server, because if it is not possible to connect with the first of them, then the next attempt could be done to the next in the list.

The traffic flow between the two nodes is shown in Figure 34. It is possible to appreciate how within these 406 bytes the SC sends to SN there are the two groups of 5 bytes exchanged previously with the difference that the last one of these 5 bytes in each group is replaced by another byte. For a given user the first 15 bytes in this 406 bytes are constant. Following with the same scheme in the 218 bytes of answer the four first bytes are always 17 03 01 00 and the last byte is replaced by another one.

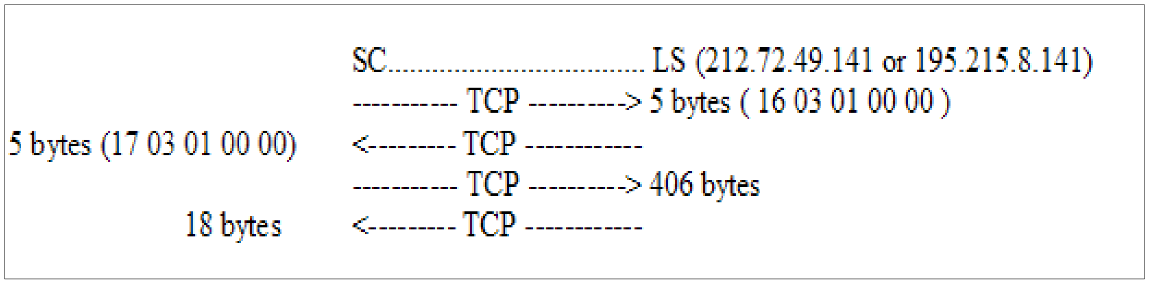

SC.................................. LS (212.72.49.141 or 195.215.8.141)
----------- TCP ---------> 5 bytes (16 03 01 00 00)
5 bytes (17 03 01 00 00) <--------- TCP ------------
----------- TCP ---------> 406 bytes
18 bytes <--------- TCP ------------

Figure 51. Traffic between SC and login server for an incorrect login

If the login process is not correct, i.e., introducing some incorrect data, then the transfer of information between these two terminals varies. Both packages of five bytes remain unchanged and continue to be exchanged. Additionally, the Skype client will send the 406 bytes to the server as many times as the login process is tried. Nevertheless, if login is not correct, then the login server will return a packet with just 18 bytes, rather than 218 bytes which again proves that this is the login process, Figure 51. Therefore, see a difference of 200 bytes between the cases of correct and incorrect login.

Additional, packet captures showing details can be found on the CD.

It must be noted that the TCP connection with the SN persists as long as the SN is alive. If the SN becomes unavailable, then a TCP connection will be started with another SN.

As I have already indicated previously, in some cases (specifically a user running on Linux machine) a HTTP connection is made to the IP address 212.72.49.131. Via this connection a message GET containing the word "getlatestversion" is sent.

In previous versions of the Skype program, the information exchanged with the machine follow this pattern:

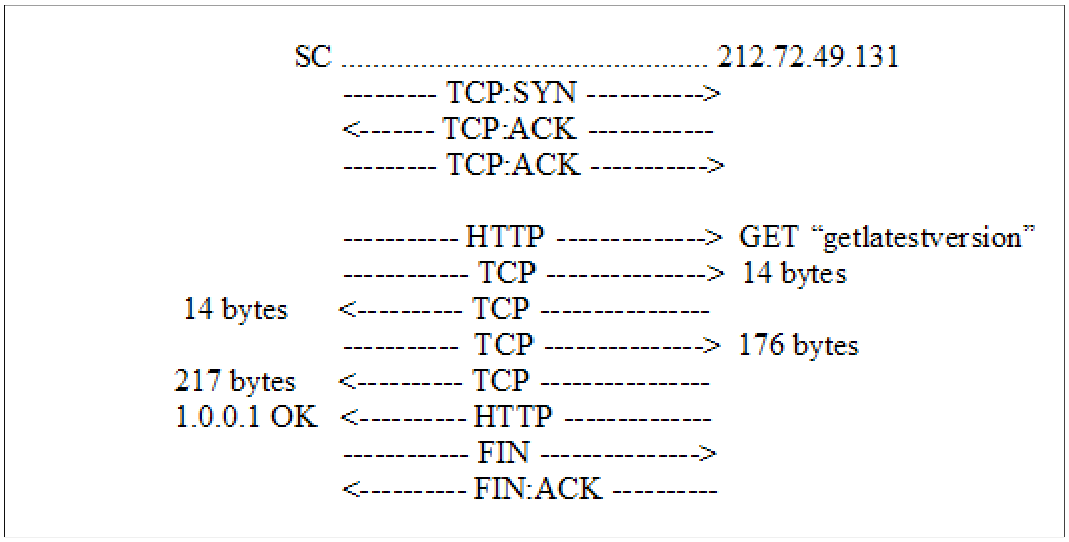


Figure 52. Connection with a second centralized node (Old version of Skype)

I have already seen this operation has been modified, so that a communication with other nodes of the network settles down which will make the authentication of the user before connection HTTP with 212.72.49.131 settles down. The current transference of information is as in Figure 35.

I believe that, the communication with this node constitutes a remnant of previous versions that will be eliminated in future versions. Note that the server only replies with an acknowledgement.

<u>Request:</u>

Hypertext Transfer Protocol
GET
http://ui.skype.com/ui/2/1.0.0.20/en/getlatestversion?ver=1.0.0.20&**uhash=be29c465765bf4cd4470b0ad662d3e39** HTTP/1.1\r\n

```
Request Method: GET
connection: Keep-Alive\r\n
host: ui.skype.com\r\n
\r\nHypertext Transfer Protocol
```

Reply:

```
HTTP/1.1 200 OK\r\n
Date: Fri, 08 Apr 2005 14:53:21 GMT\r\n
Server: Apache\r\n
X-Powered-By: PHP/4.3.9\r\n
Cache-control: no-cache, must revalidate\r\n
Pragma: no-cache\r\n
Expires: 0\r\n
Content-Length: 7\r\n
Connection: close\r\n
Content-Type: text/html; charset=utf-8\r\n
Content-Language: en\r\n
\r\n
Data (7 bytes)

0000  31 2e 30 2e 30 2e 31                    "1.0.0.1"
```

The parameter "uhash=be29c465765bf4cd4470b0ad662d3e39" is constant if the Skype user is fixed and it is the hash of the user ID, the password and the own value of the hash function. Moreover, the hash value has 32 hexadecimal numbers so that the number of bits is 32*4= 128 bits. Then, the hash function used could be MD5.

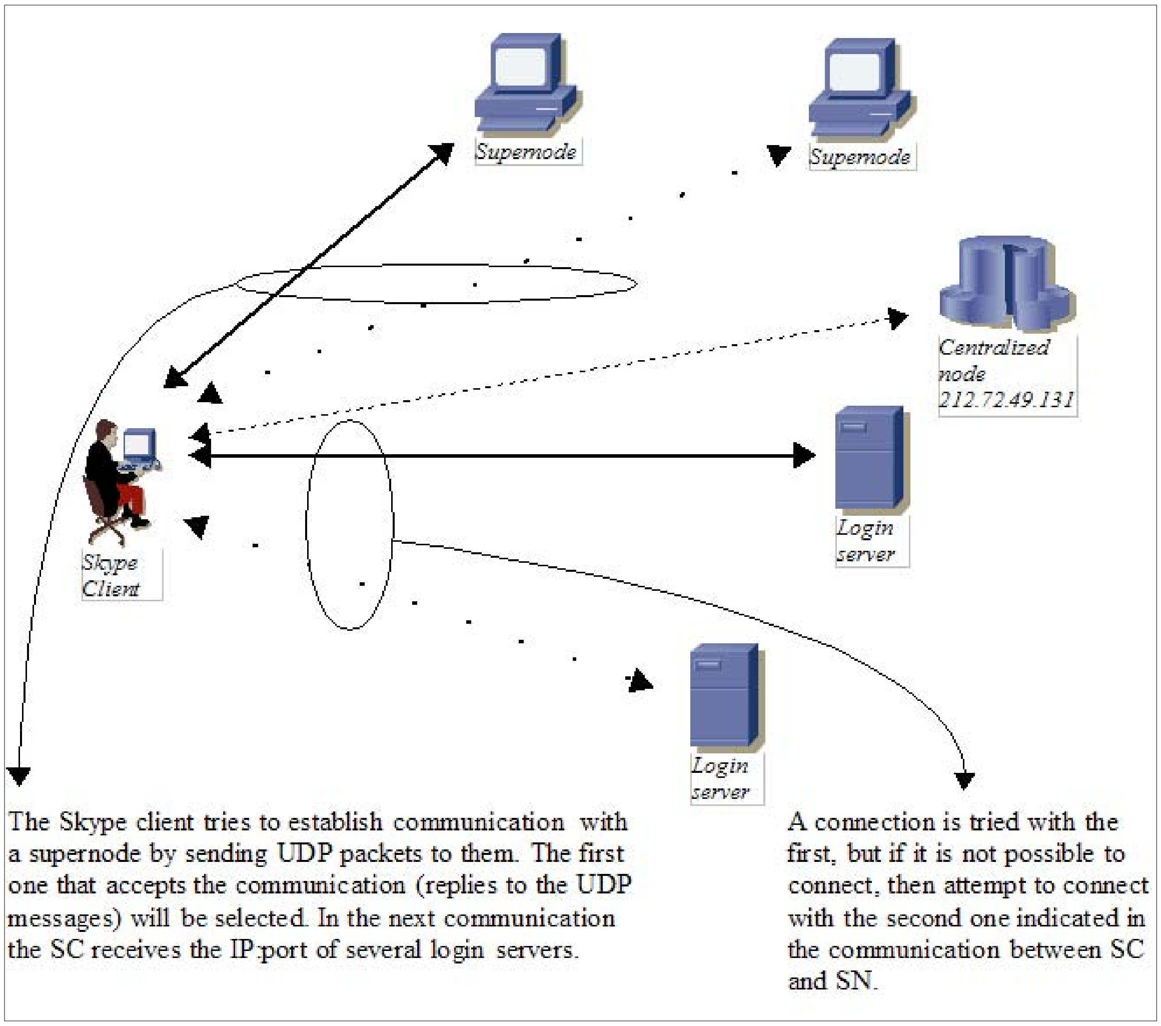
The Skype operations explained up to this point are shown in the Figure 53.

A.2.3 User Search

Skype uses its Global Index technology (GI) to search for a user. Skype also claims in [21] that the search is distributed and is guaranteed to find a user if they exist and have logged in the last 72 hours.

The messages in Skype are encrypted so while the login process is possible to understand, the search process is more difficult, because we do not know the details of the technique Skype is using.

In any case, Skype provide the user with a dialog box by which to search for a user. Once the start button is pressed, the behavior is similar to that which observed in the previous study [63]. In that study, the investigators showed that the Skype client sends a packet TCP to its SN (super node). This supernode SN responds by sending another packet of greater size than the previous one which contains the IP:port pairs of four possible nodes to be contacted. The situation was as in Figure 54.



The Skype client tries to establish communication with a supernode by sending UDP packets to them. The first one that accepts the communication (replies to the UDP messages) will be selected. In the next communication the SC receives the IP:port of several login servers.

A connection is tried with the first, but if it is not possible to connect, then attempt to connect with the second one indicated in the communication between SC and SN.

Figure 53. Skype scheme

```
           SC ............................................ SN
              --------------- TCP ---------------> 16 Bytes
52 Bytes <------------ TCP ---------------
              --------------- UDP -------------> 77 Bytes  N1
              --------------- UDP -------------> 77 Bytes  N2
              --------------- UDP -------------> 44 Bytes  N3
              --------------- UDP -------------> 44 Bytes  N4
```

F igure 54. Communication during the user search process (Old version)

Currently, the system is based on the same decentralized process to request addresses from the SN, followed by sending UDP packets later to the specified corresponding nodes using the indicated IP addresses and ports. Nevertheless, it was observed the number of nodes is not four in the previous study, but has increased -if necessary multiple queries are made to the super node.

The current situation is shown in the following Figures 55 and 56.

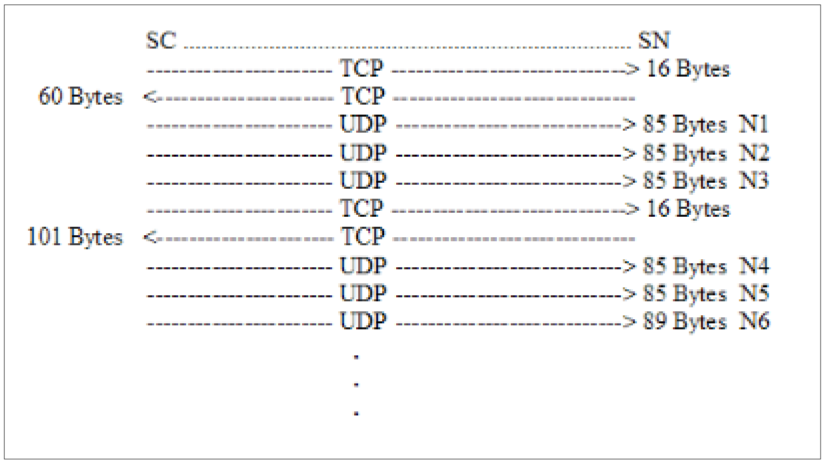


Figure 55. Current user search

If additional nodes must be queried, then new messages will be sent from the SC to the SN. The size of these replies alternates between 60 and 101 bytes. The size of the UDP packets sent by SC will be either 85 or 89 bytes, but the reply is always 29 bytes. The SC always sends several UDP packets with 85 bytes before sending packets with 89 bytes. I do not know what is the reason for 4 bytes-difference between the two kinds of packets. It was also observed in the subsequent captures that if the Skype user is not able to find the requested user or when that user is found, a message with 16 bytes is sent to the super node (SN) and it responds by sending another packet 93 bytes. It is not clear how this search terminates if it fails to find a user.

A.2.4. Call Establishment / Teardown

In this section we continue the comparison between the current operation of the program and the previous version.

The study of the previous version showed two important ideas:

- Call signaling is always carried over TCP.
- For users that are not present in the buddy list, call placement is essentially equal to user search plus call signaling.

Different measurements were made and they showed that the second aspect above was always true. Nevertheless, it is necessary study the existing transfers between the different involved terminals during the calls in order to be able to make confirm the previous study.

Considering online users with each one in the buddy list of the other, upon pressing the "call" button we observe the traffic in Figure 57.

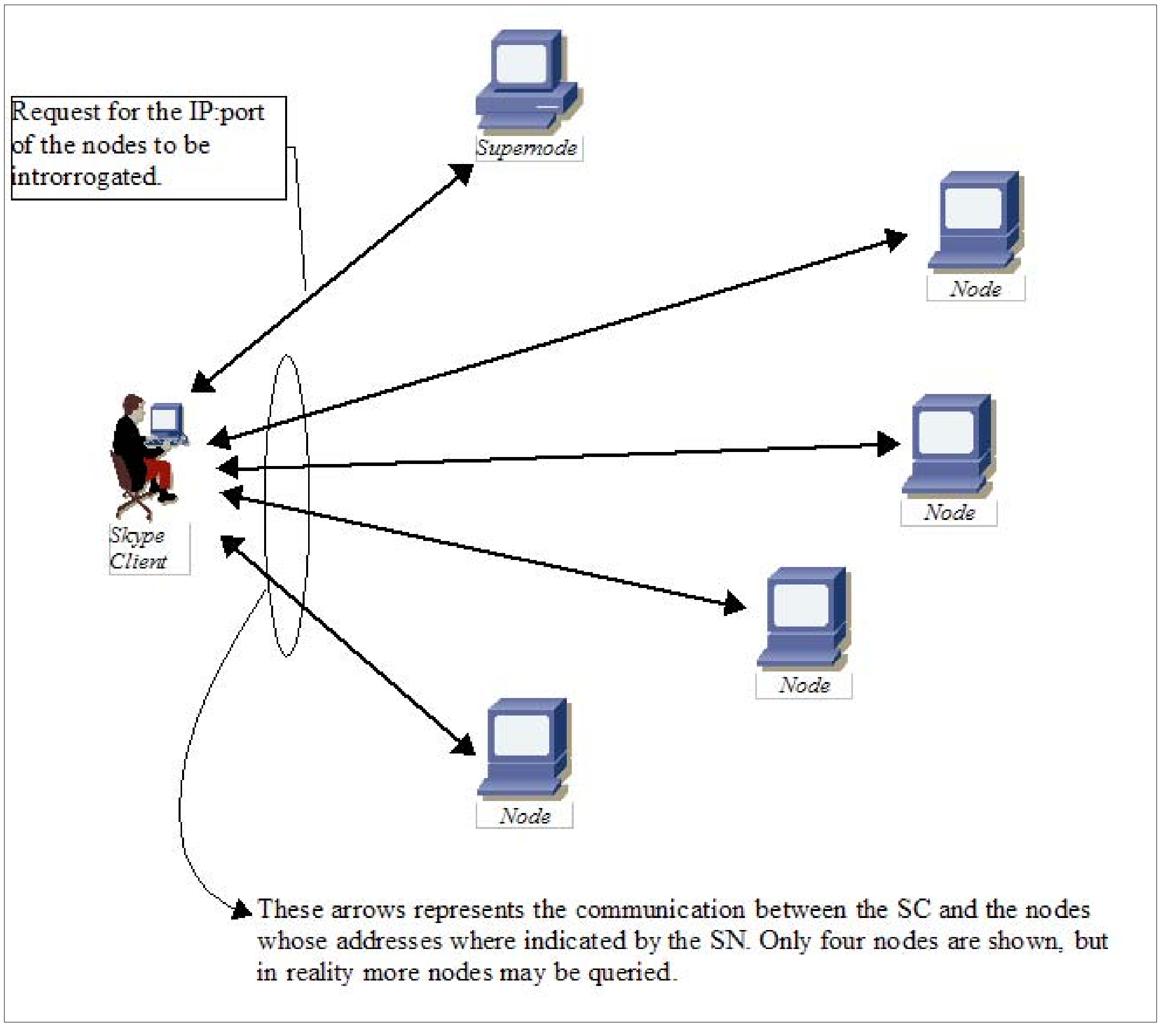

Request for the IP:port of the nodes to be introrrogated.
Supernode
Node
Node
Node
Node
Skype Client
These arrows represents the communication between the SC and the nodes whose addresses where indicated by the SN. Only four nodes are shown, but in reality more nodes may be queried.

Figure 56. Search scheme

As can be observed in the previous scheme as well as in the corresponding captures available in the additional CD, we see TCP traffic between the machines, but we also see UDP packets sent from one machine to another. I propose that the transfer of the minimum data for the call establishment will be sent via TCP whereas other information that can improve the communication with respect to previous versions is sent on UDP.

As commented previously, we can understand what is happening in some of the transfers. Thus, the two transfers with sizes of 103 and 41 bytes respectively are used to carry the warning message between the caller and callee users indicating the caller is online and the reply. I assume this because when the user performs the login process a message with 101 or 103 bytes is sent to each one of his contacts and the reply from each one of them contains 41 bytes. In fact, this transfer does not appear in all the captures in the same position because if the "warning message" was sent previously to your contacts then we already know that the user to be called is online.

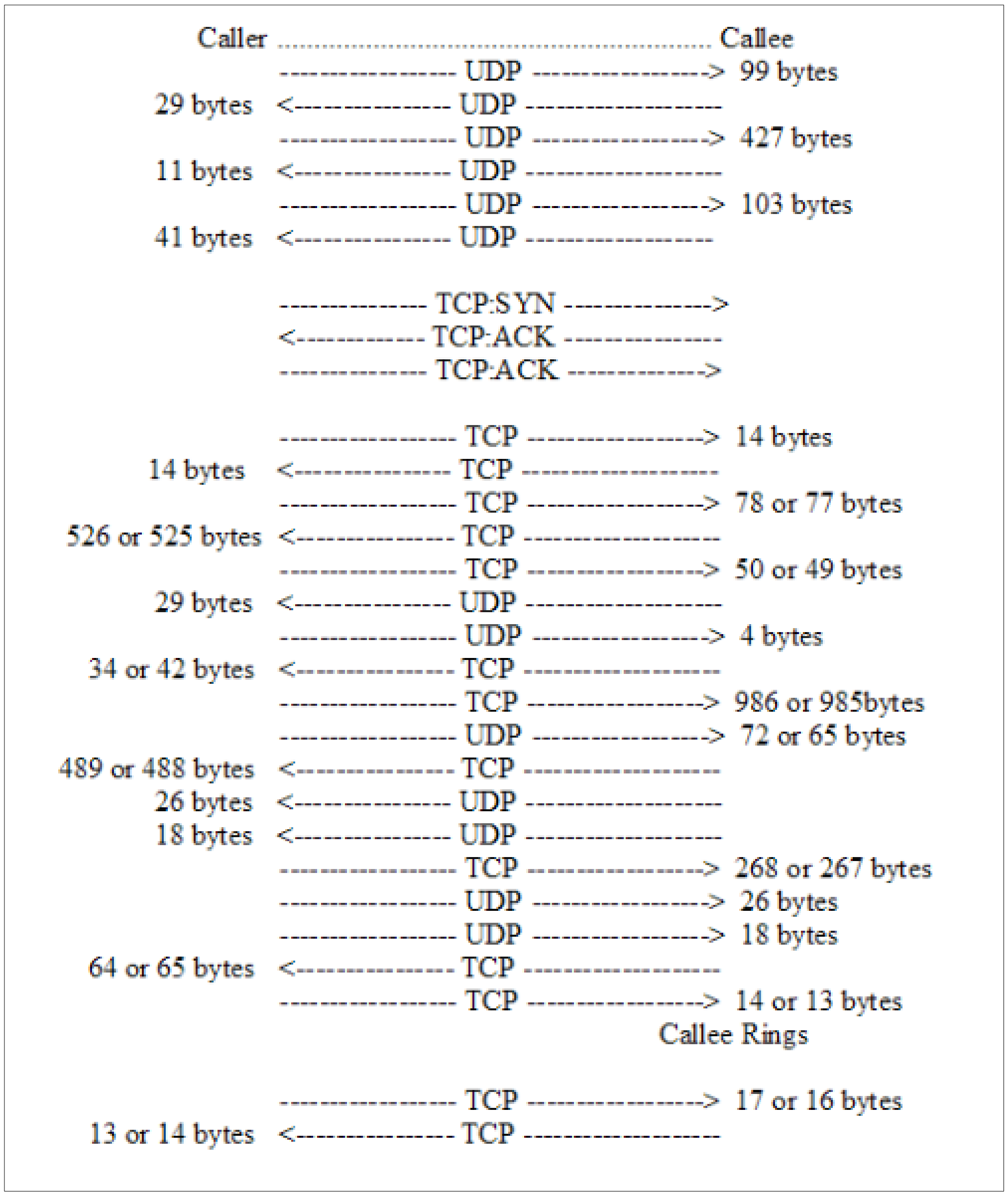

Caller .... Callee
------ UDP ------> 99 bytes
29 bytes <------ UDP ------
------ UDP ------> 427 bytes
11 bytes <------ UDP ------
------ UDP ------> 103 bytes
41 bytes <------ UDP ------
------ TCP:SYN ------>
<------ TCP:ACK ------
------ TCP:ACK ------>
------ TCP ------> 14 bytes
14 bytes <------ TCP ------
------ TCP ------> 78 or 77 bytes
526 or 525 bytes <------ TCP ------
------ TCP ------> 50 or 49 bytes
29 bytes <------ UDP ------
------ UDP ------> 4 bytes
34 or 42 bytes <------ TCP ------
------ TCP ------> 986 or 985bytes
------ UDP ------> 72 or 65 bytes
489 or 488 bytes <------ TCP ------
26 bytes <------ UDP ------
18 bytes <------ UDP ------
------ TCP ------> 268 or 267 bytes
------ UDP ------> 26 bytes
------ UDP ------> 18 bytes
64 or 65 bytes <------ TCP ------
------ TCP ------> 14 or 13 bytes
Callee Rings
------ TCP ------> 17 or 16 bytes
13 or 14 bytes <------ TCP ------

Figure 57. Call establishment

The timestamps in each packet that Ethereal provides permit us to state that the first transfers do not really correspond to the call establishment as it was thought initially since the duration of the process would be longer than 5 or 6 seconds. However, these messages are present in all our measurements.

Then the TCP connection is established by sending three messages and next, fourteen bytes in each direction are exchanged. As described in [63], these fourteen bytes perform the mutual authentification.

The other transfers cannot be explained since we do not know the Skype protocol. In any case in the scheme of the traffic between both terminals it is observed that the caller user always sends 4 bytes to the callee user. At first I thought about the possibility that these four bytes corresponded to a encrypted IP address. Nevertheless, in the successive captures it was verified the calls established in a same day present a first common byte. Also it was verified the last 12 bits of this transference are constant (1101 0000 0000) and as a result of this discovery I suppose that those four bytes constitute just a temporary information.

Finally it is shown two messages with 17 (or 16) and 13 (or 14) bytes sent between them. These two packets correspond to the end of the call and their directions depend on the user who decides to tear down the call. This situation agrees with the awaited one and indicated in the study made in the University of Columbia [63].

The packet captures performed permit us to determine the average delay of the call establishment process as well as the average of transmitted bytes number of transferred data.

Table 3. Call establishment transmission

Time (sec.)	***Data (bytes)***
0,203065 sec.	2710 bytes
0,195481 sec.	2765 bytes
0,206562 sec.	2710 bytes
0,1551889 sec.	2710 bytes

Average Time = 0,19007425 sec.
Average Data = 2723,75 bytes

With these two values the average bandwidth used in the establishment and tear down processes can be calculated. Thus the average bandwidth is:

BWcall_establishment ≈ 114639 bps

Once I have the packet captures I can also calculate the average delay of the tear down process as well as the average of transmitted bytes number of transferred data:

Table 4. Tear down traffic.

Time (sec.)	***Data (bytes)***
0,163027 sec.	31 bytes
0,1235464 sec.	29 bytes
0,188132 sec.	31 bytes
0,155089 sec.	30 bytes

Average time = 0,15744885 sec.
Average data = 30,25 bytes

Then the average bandwidth is: BWcall_ending ≈ 1537 bps

There was not information in [63] about bandwidth used in the tear down process but the average of bandwidth in the call establishment was 3 Kbytes/s. Then the current Skype version utilizes a bandwidth almost five times bigger than in the previous.

A.2.5 Media Transfer and Codecs

The different packet captures sessions enabled me to see that the voice traffic is sent over UDP packets (being possible to select the port to be used in the options of the Windows version of the software). As described in [63] the traffic used for media transfer was 5 kbps by sending packets of a size of 67 bytes, so it is between the limits indicated by Skype (3 – 16 kBps).

However, in the current Skype version I have found that the sizes of these packets are **not** fixed, and it is not possible to discern a pattern in their variation although the media traffic flows inside UDP packets.

Moreover, we can see that when the call is initialized but there is no voice flowing between users, the bandwidth does not decrease. This indicates the current Skype version does not utilize a voice activity detection mechanism.

The bandwidth used for media transmission in each communication was calculated taking intervals in the communications with duration of approximately a second and calculating the number of bytes transmitted. With these two values I could calculate the bandwidth. To remove the dependence on the specific temporal period considered, I utilized several periods of the same duration and the values obtained are:

- With media traffic exchanged:
BWmt ≈ 7892 Bytes/sec.
- Without media traffic exchanged (just silence):
BWwmt ≈ 7175 Bytes /sec.

These values present a variance since the packets size change from one to another measurement. However, the comparison of these two bandwidths supports the previous statement about the non-existence of a voice activity detection mechanism in the current version of Skype. These packets will be confort noise packets to make the communication more pleasant. In any case, the bandwidth used for voice transmission continues inside the limits indicated by Skype (3 – 16 kBps)

TCP traffic was also observed during the measurements. TCP packets with a size of 18 bytes and their acknowledgements were exchanged alternatively between both users joining the call. The interval between the sending of these messages in one direction and in the other varies. In my opinion, these messages are used to maintain a control on the call, indicating the type of information that will be sent in following UDP messages.

On average, the TCP traffic is:

- Caller to Callee, Callee to Caller:

T1 ≈ 0,31 sec.

- Callee to Caller, Caller to Caller:

T2 ≈ 0,68 sec.

With respect to the rate of sending TCP messages, the average value obtained was:

BWtcp ≈ 41 Bytes/sec.

In addition to the value of the bandwidth calculated above, delay and jitter in these packet capture sessions were measured following the scheme in Figure 38.

The delay was measured looking in both computers when the same packets are sent from one of the users and when is received in the other one. To calculate the jitter we have to calculate the variance of the interarrival time. However, this is only possible if we have a common time reference given by the NTP clock [78].

Table 5. Some measures to calcule the delay and jitter

delay
0,372223 msec
0,32456 msec
0,459283 msec
0,32456 msec
0,455634 msec
0,567456 msec
0,343253 msec
0,62114 msec
0,376654 msec
Average = 0,4272 msec
Jitter = 0,10771 msec

But, as it was commented in section 7.1.3, these values correspond to the delay introduced by the network. What we want to find is the time from the moment in which the user speaks to the moment in which the other user listen the voice. Then, the scheme we should follow is in Figure 39.

This noise can be some packets. If we record the communication and process it by calculating the Fourier Transform, we should be able to obtain the frequency of the occurrence of these packets.

However, it is possible to measure the delay in a easier way as was explained in section 7.1.3 with the Figure 39. Following that scheme, the result is shown in Figure 58.

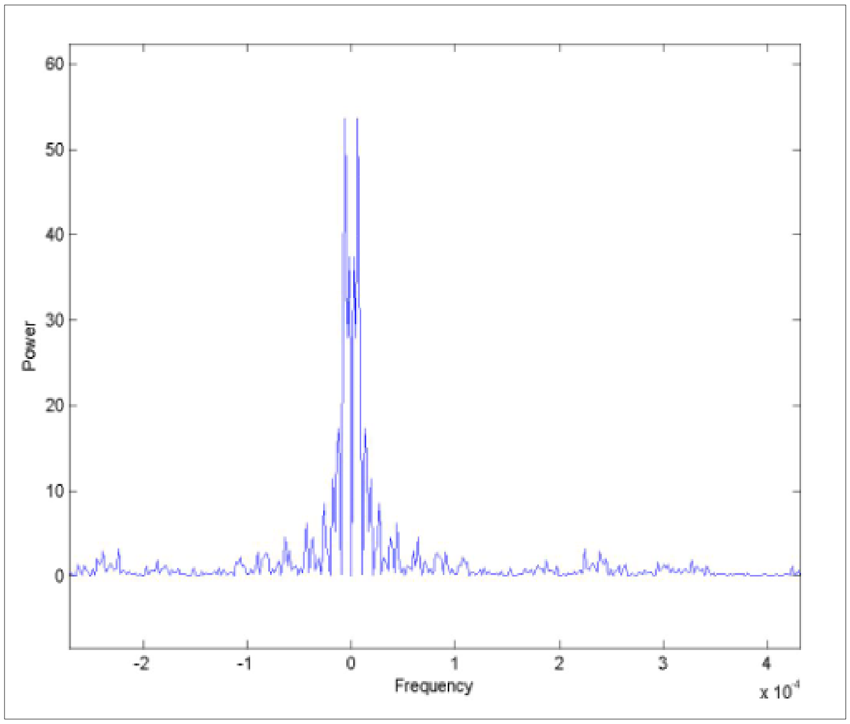


Figure 58. End to End delay in the frequency domain.

This figure corresponds to values of:
f_{sample}= 88223 Hz
f = $6*10^{-6}$
Delay =1,88 sec.

The average obtained in different experiments is:
Average Delay =0,89 sec.
Jitter = 0,014 sec.

Now we know the value of the delay and the jitter, and also what is the influence of the network. A future work to consider could be to simulate longer distances to know which the limit for the delay is.

Moreover, it is also interesting to know approximately the bandwidth used by Skype when the user is online, but when not making calls. In the Skype website [21] is assured the bandwidth used is between 0 and 0.5 Kbytes. Next I show the results corresponding to five simulations without voice in Table 6, that is, without making calls.

We can see now that when there is not voice, the bandwidth used is around 0 to 0.5 kilobytes per second.

With respect to CODECs, the study made by the University of Columbia revealed that codecs used could be three. These codecs are iLBC, iSAC or a third stranger. In this study it was also mentioned the GlobalIPSound Company displays Skype in its website to like its partner, in fact, it is also assumed the possibility that Skype uses its codecs.

Table 6. Traffic without making calls

Simulati	Bytes	Time	Bandwidth
1	1551	15,5472 sec.	≈ 99,82 Bps
2	2670	15,3779 sec	≈ 173,63 Bp
3	2340	15,2194 sec	≈ 153,75 Bp
4	2090	16´081 sec.	≈ 129,97 Bp
5	1673	14,7833 sec	≈ 113,17 Bp

The codecs mentioned in the [60] are:

- iLBC
- RCU
- Enhanced G.711
- iPCM_wb
- ISAC

Nevertheless, in [21] is stated the program uses broadband codecs, so that iLBC, RCU and Enhanced G.711 will not be used because they are narrow codecs, that is to say, the sampling frequency is 8000 Hz.

Therefore, we just have the possibility of the iPCM_wb and iSAC codecs. Nevertheless, iPCM_wb has built-in voice activity detection functionality (VAD), something that we have verified Skype does not incorporate. Consequently, we can affirm that codec used is not iPCM_wb.

iSAC is codec "which maintain wideband communication over low and high bit rate connections" [60]. This codec makes an automatic tuning of the transmission rate between 10kBps and 32kBps. This automatic tuning would explain the size variation in the exchanged UDP messages between the users. The problem of this codification is that the rate of transmission is between 10 and 32 KBps whereas Skype assures a rate located between 3 and 16 KBps. It supposes that either it is not a viable option unless they are used different codecs based on the number of bytes to transmit at each moment and then iLBC could be used for low rates and iSAC could be used by high rates.

But, apart from the information presented by Skype, I performed an experiment which consists of recording a speech conversation and processing it using Matlab with the following code:

```
clear all;
    descriptor = fopen('marco.wav');
    signal = fread(descriptor);
    media = mean(signal);
    %% marco.wav has 20 sec.

    for i=1:length(signal)
  signal(i) = signal(i)- media;
```

```
end;
fsample =length(signal)/20;
t = 0:20/length(signal):20-20/length(signal);

figure(1);
plot(t,signal);

transf=fft(signal);
transfMod=fftshift(transf);
figure(2);
f= -0.5:1/length(transfMod):0.5-1/length(transfMod);
plot(f, abs(transfMod)/length(transfMod));
potencia = abs(transfMod).*abs(transfMod)/length(transf)^2;

figure(3);
plot(f,potencia);

figure(4);
potencia2=transfMod.*conj(transfMod)/(length(transfMod)^2);
plot(f,potencia2);
```

What this code does is to take the signal received, eliminates the DC level (not part of the speech communication as voice does not have DC level) with a low band filter. The signal obtained is in Figure 40.

The next step is to extract the frequencies of the signal. To do that, we calculate the Fourier transform of the signal and we display it centered in Figure 41.

In the previous figures we can see that the frequency axis is in digital frequency. In the previous figures, the value 0,5 corresponds to the half of the sampling frequency and -0,5 to minus half that frequency. By checking the number of samples of the signal and knowing that the duration is 20 seconds, it is easy to calculate a sampling frequency of 22542 KHz. Therefore, we prove that the signal contains frequencies upon almost 11,25 KHz so the codec must be a broadband codec. Therefore, for low data rates the codec used is also broadband so that iLBC never is used.

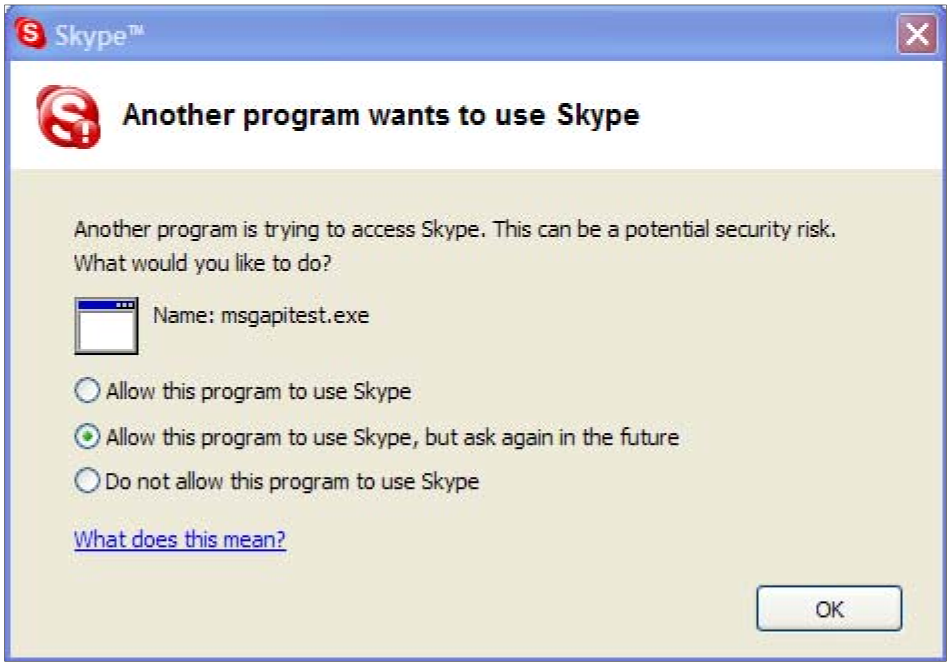
Moreover we can see in Figure 42 that the main power of the signal is located in the low frequencies.

Appendix B Introduction to the Skype API in our interface

At the Skype website [21], it is possible to find an example of the API controlled by a console so that the user might control it introduce with keyboard commands.

Unfortunately this API does not allow the use of the full functionality of Skype, shown previously in this thesis. The aim of this document is neither to explain each of the commands and options the API includes. Instead of that, we explain the options used to connect our graphical user interface with Skype and show the results in the console to illustrate our comments.

Upon executing the API functions, in the console version, directly from our GUI, the following message will pop up:



Another program wants to use Skype

Another program is trying to access Skype. This can be a potential security risk. What would you like to do?

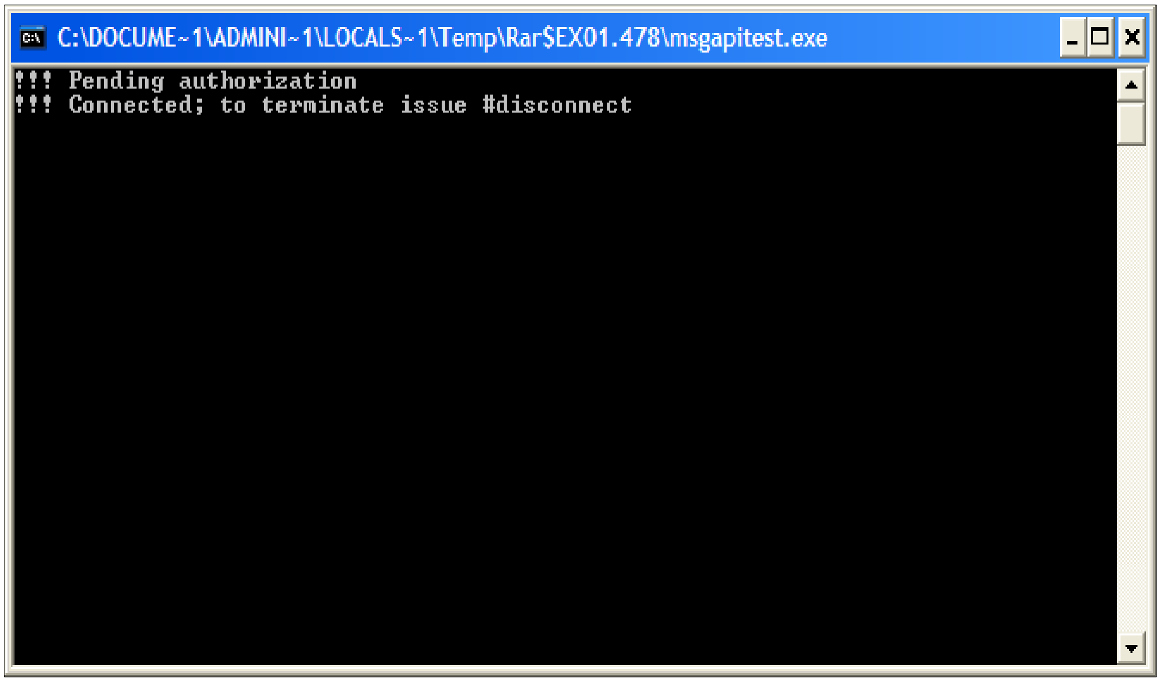
Name: msgapitest.exe

- ( ) Allow this program to use Skype
- [✓ Allow this program to use Skype, but ask again in the future]
- ( ) Do not allow this program to use Skype

Figure 59. Another program tries to connect with Skype.

Such a dialog allows us to control the connection with the Skype program. In our interface we do not want to see this dialog every time we execute the program. Therefore, once the graphical user interface is implemented we can select the first one of these choices in the above dialog (which is called “Allow this program to use Skype”), then this dialog will not appear again.

Furthermore, when we have accepted the connection by selecting the first or the second option, the connection will be established. Using the console to utilize the API, the figure below shows the result we obtained.


C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\Rar$EX01.478\msgapitest.exe

```
!!! Pending authorization
!!! Connected; to terminate issue #disconnect
```

Figure 60. Skype API controlled by a console.

In the case of working with the API from our interface, the same communication from Skype will be received and will be saved in a file called “stateAPI.txt”, but it will not be shown here because it is not useful for the reader.

When we are connected to Skype whatever we enter in the console (or executing via the interface) will return the same result than executing the command from the Skype GUI will cause.

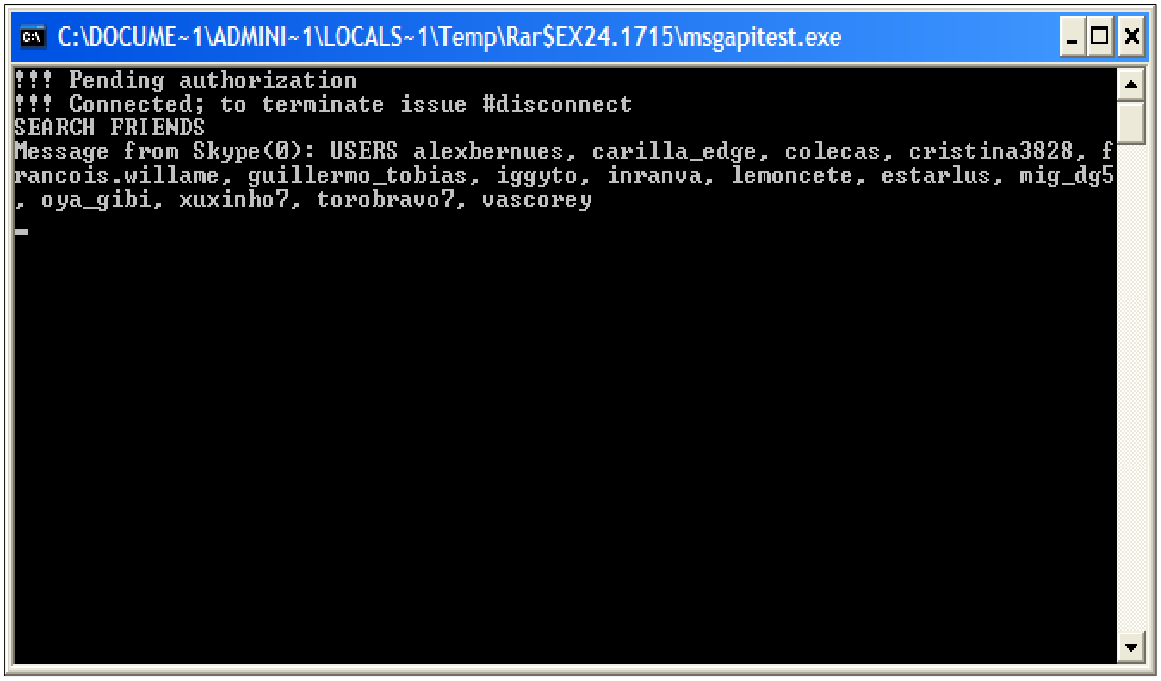
Once I have explained the operation of our interface it is easy to find the set of messages we need from the commands that the Skype API provides. We need four basic messages, the first one to establish the connection with Skype and another three to retrieve the information that we need to operate from this interface.

The first message is the “SkypeControlAPIDiscover” which is sent to all the windows on the top level of the windowing system. Skype replies by sending the message “SkypeControlAPIAttach” indicating the status of the connection, i.e. the status shown in the figure above. The code to send this first message is:

```
    if (SendMessage(HWND_BROADCAST,
uiGlobal_MsgID_SkypeControlAPIDiscover, (WPARAM)hWnd, 0)==0 )
    {
            Communication = fopen("Communication.txt", "w");
            fprintf(Communication,"failed message");
            fclose(Communication);
    }//end Communication
```

Once we have established the connection to Skype, we can send additional messages.

Firstly, we need to retrieve the client´s contacts. To do that, we have to send a string "SEARCH FRIENDS" and send it to Skype. The result of this action is shown in the following figure:


C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\Rar$EX24.1715\msgapitest.exe

```
!!! Pending authorization
!!! Connected; to terminate issue #disconnect
SEARCH FRIENDS
Message from Skype(0): USERS alexbernues, carilla_edge, colecas, cristina3828, f
rancois.willame, guillermo_tobias, iggyto, inranva, lemoncete, estarlus, mig_dg5
, oya_gibi, xuxinho7, torobravo7, vascorey
```

Figure 61. Reply to the SEARCH FRIENDS message

The code necessary to perform this (from the file SecondButton.cpp) is shown below:

```
char *szRetrieve= "SEARCH FRIENDS";

if(hGlobal_SkypeAPIWindowHandle!=NULL)
{
   // COPYDATASTRUCT is the structure that will contain the message
         COPYDATASTRUCT rCopyData;
         rCopyData.dwData=0;
         rCopyData.lpData=szRetrieve;
         rCopyData.cbData= strlen(szRetrieve)+1;
         // we create the SearchFriends.txt file to check that all is
correct
         messSent=fopen("SearchFriends.txt", "a");
         fprintf(messSent,"%s\n", rCopyData.lpData);
         // we check the string is correct
         fclose(messSent);

         if( rCopyData.cbData!=1 )
         {
         // here is the command to send the message to Skype
               if( SendMessage( hGlobal_SkypeAPIWindowHandle,
WM_COPYDATA, (WPARAM)hWnd, (LPARAM)&rCopyData)==FALSE )
               {
                     hGlobal_SkypeAPIWindowHandle=NULL;
                     discInMessage=fopen("discInMessage.txt","w");
                     fprintf(discInMessage,"!!! Disconnected\n");
```

```
                        fclose(discInMessage);
                    }
                }
    }// end hGlobal_SkypeAPIWindowHandle
```

As we can see in the figure above, Skype replies by sending a string listing the friends. In our interface we simply save that string in a file called “MessaggeFromSkype.txt” to process the information later. The code below stores the message received, i.e., the list of contacts).

```
    case WM_COPYDATA:
        if(hGlobal_SkypeAPIWindowHandle ==(HWND)wParam)
        {
              PCOPYDATASTRUCT poCopyData = (PCOPYDATASTRUCT) lParam;
              // it is not necessary to save each time the contacts -> “w”
              MessageFromSkype= fopen("MesaggeFromSkype.txt", "w");
              fprintf(MessageFromSkype,"Message from Skype(%u): %.*s\n",
poCopyData->dwData, poCopyData->cbData, poCopyData->lpData);
              fclose(MessageFromSkype);
              return 0;
        }
```

For each one of these contacts we will show it via our interface. To be able to show contacts via our interface we have to process the information retrieved. First, I used two functions called “eliminateHeader” and “eliminateComma”, which allow us to remove both the commas and the “Message from Skype(0): USERS“. The result will be saved in a file called “NewPruebaFile.txt”. The definitions of these functions are:

```
    void eliminateHeader(FILE *fd, char *data)
    {
        int cont;
        int exit=0;
        FILE *newFile;

        // fd should be opened and closed outside of this function in order
        // to be a more general function

        // 29 digits to delete Message from Skype(0): USERS in the string
        for (cont=0; cont < 29; cont++)
        {
              if (fread(&data[cont],sizeof(char),1, fd)==0)
                    break;
              else if (data[cont]=='\n')
                    break;
        }

        newFile=fopen("NewFile.txt","w");
        while(!exit)
        {
              if (fread(&data[cont],sizeof(char),1,fd)!=0)
                    fprintf(newFile, "%c", data[cont]);
              else
                    exit=1;
              cont++;
        }
        fclose(newFile);
    }

    void eliminateComma(FILE *fd, char *data)
    {
        int cont=0;
        int exit=0;
        FILE *newPruebaFile;

        newPruebaFile=fopen("NewPruebaFile.txt","w");

        while(!exit)
        {
              if (fread(&data[cont],sizeof(char),1,fd)!=0)
              {
                    if(data[cont]==',')
                          cont--;
                    else
                          fprintf(newPruebaFile, "%c", data[cont]);
                    cont++;
              }
              else
                    exit=1;
        }
        fclose(newPruebaFile);
    }
```

The next step reads from “NewPruebaFile.txt” all the contacts and writes them to an intermediate file one per line. This new file has the name “IntermediateFile.txt”. In order to perform these actions we created two functions called readLine and readLineModified. First, readLineModified is called to write the contacts to the IntermediateFile.txt and later readLine will take each contact and store it in a char[] variable. The codes of these two functions are:

```
void readLineModified(FILE* fd, char* data)
{
    int exit=0;
    int numBytes=0;
    int iden=0;

    while (!exit){
          if (fread(&data[numBytes], sizeof(char), 1,fd)==0)
                exit = 1;
          else if (data[numBytes]==' '){
                exit=1;
                numBytes--;
          }
          numBytes++;
          data[numBytes]='\0';
    }
}

void readLine(FILE* fd, char* data){
    int exit=0;
    int numBytes=0;
    while (!exit){
          if (fread(&data[numBytes], sizeof(char), 1, fd)==0)
                exit=1;
          else if (data[numBytes]=='\n'){
                exit=1;
                numBytes--;
          }
          numBytes++;
    }
    data[numBytes]='\0';
}
```

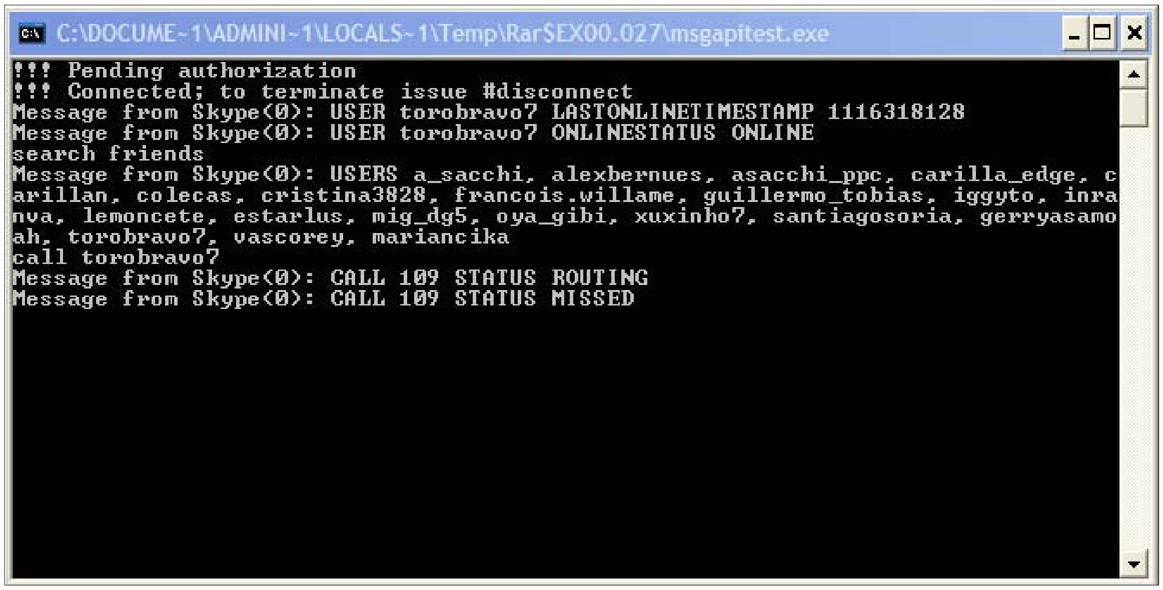
The format of every contact is now a char[], but if we want to display it via the visual interface, we need to have them into TCHAR[] format. To do that it is used the ConvertCToT function which is shown below. Once this is done, we can display all the contacts via the interface.

```
void ConvertCToT(TCHAR* pszDest, const CHAR* pszSrc)
{
    for(int i = 0; i< strlen(pszSrc); i++)
    {
                pszDest[i] = (TCHAR) pszSrc[i];
    }
}
```

Following with the explanation given in section 6.3.2, the user might be interested in calling one of his contacts. When working via the console, we utilize the command CALL <username>, where username is the Skype client contact to be called. When using such a command, the reply from Skype can vary depending on the state of the contact. We first consider the case when the contact is ONLINE and accepts the call. Then, the speech communication starts and will continue until the one of the users decides to end the call and press the "HANG" button.

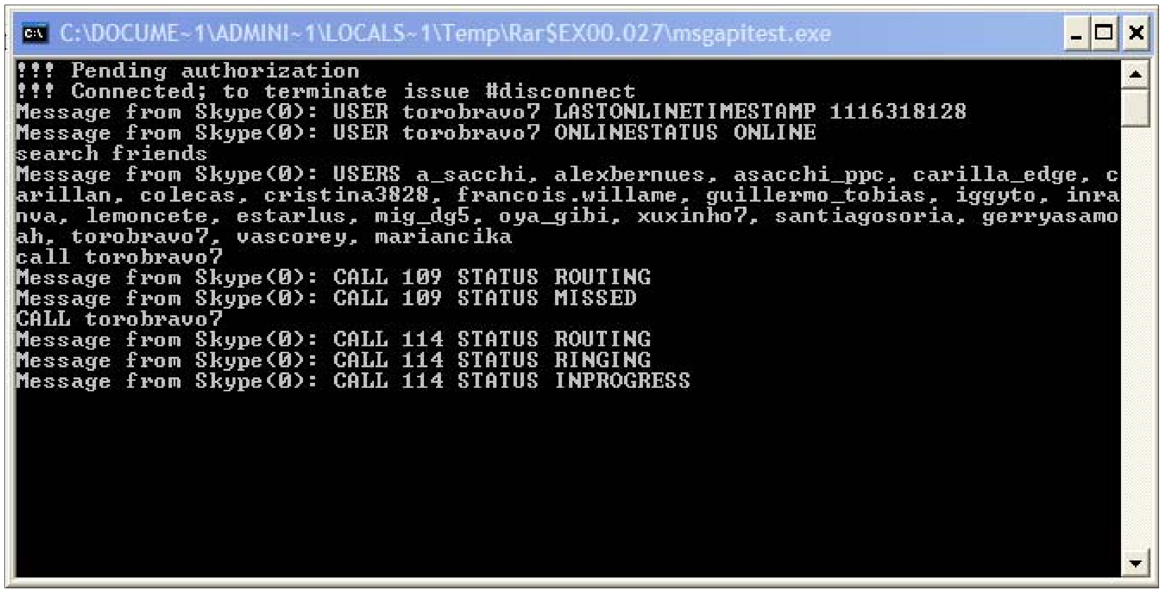
To start the communication when the user presses the "CALL" button, we read the name of user from the editlist and create a CALL <username> string and we send to Skype. The code necessary is:

```
char szCommand[128];
if(hGlobal_SkypeAPIWindowHandle!=NULL)
{
          COPYDATASTRUCT oCopyData;
          oCopyData.dwData=0;
          strcpy(szCommand, "CALL ");
          strcat(szCommand, szContact);

          oCopyData.lpData=szCommand;
          oCopyData.cbData= strlen(szCommand)+1;

          if( oCopyData.cbData!=1 )
          {
                if( SendMessage( hGlobal_SkypeAPIWindowHandle,
                WM_COPYDATA, WPARAM)hWnd, (LPARAM)&oCopyData)==FALSE )
                {
                        hGlobal_SkypeAPIWindowHandle=NULL;
                 }
           }
}//hGlobal_SkypeAPIWindowHandle != NULL
```



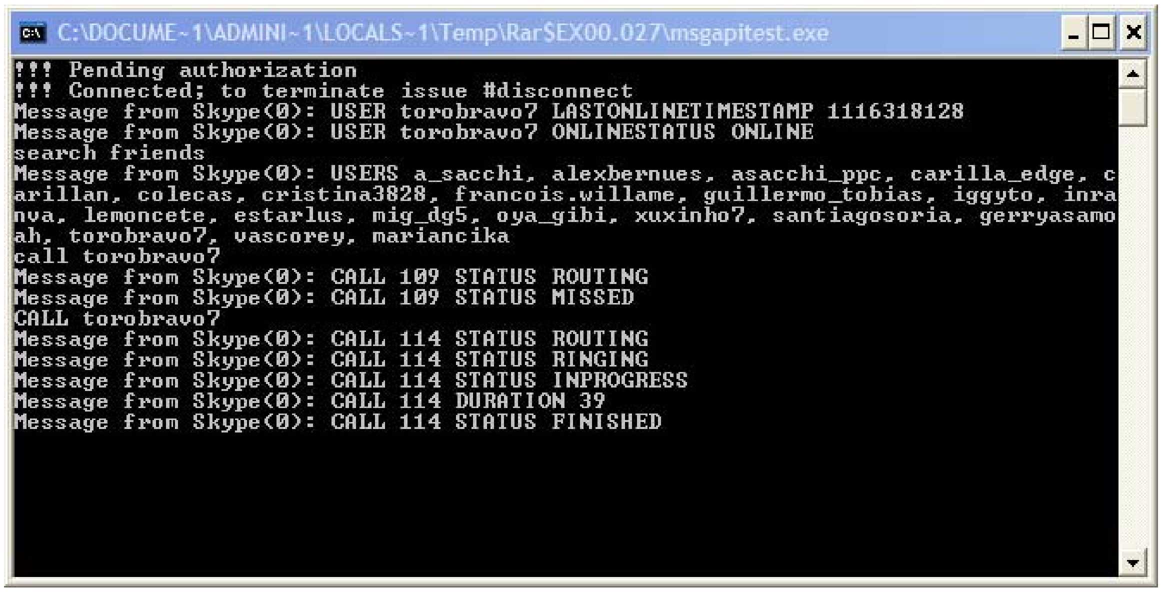
```
!!! Pending authorization
!!! Connected; to terminate issue #disconnect
Message from Skype(0): USER torobravo7 LASTONLINETIMESTAMP 1116318128
Message from Skype(0): USER torobravo7 ONLINESTATUS ONLINE
search friends
Message from Skype(0): USERS a_sacchi, alexbernues, asacchi_ppc, carilla_edge, c
arillan, colecas, cristina3828, francois.willame, guillermo_tobias, iggyto, inra
nva, lemoncete, estarlus, mig_dg5, oya_gibi, xuxinho7, santiagosoria, gerryasamo
ah, torobravo7, vascorey, mariancika
call torobravo7
Message from Skype(0): CALL 109 STATUS ROUTING
Message from Skype(0): CALL 109 STATUS MISSED
```

Figure 62. Call missed. The identifier of the call is 109



```
!!! Pending authorization
!!! Connected; to terminate issue #disconnect
Message from Skype(0): USER torobravo7 LASTONLINETIMESTAMP 1116318128
Message from Skype(0): USER torobravo7 ONLINESTATUS ONLINE
search friends
Message from Skype(0): USERS a_sacchi, alexbernues, asacchi_ppc, carilla_edge, c
arillan, colecas, cristina3828, francois.willame, guillermo_tobias, iggyto, inra
nva, lemoncete, estarlus, mig_dg5, oya_gibi, xuxinho7, santiagosoria, gerryasamo
ah, torobravo7, vascorey, mariancika
call torobravo7
Message from Skype(0): CALL 109 STATUS ROUTING
Message from Skype(0): CALL 109 STATUS MISSED
CALL torobravo7
Message from Skype(0): CALL 114 STATUS ROUTING
Message from Skype(0): CALL 114 STATUS RINGING
Message from Skype(0): CALL 114 STATUS INPROGRESS
```

Figure 63. Call accepted by torobravo7. The identifier of the call is 114.



```
C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\Rar$EX00.027\msgapitest.exe
!!! Pending authorization
!!! Connected; to terminate issue #disconnect
Message from Skype(0): USER torobravo7 LASTONLINETIMESTAMP 1116318128
Message from Skype(0): USER torobravo7 ONLINESTATUS ONLINE
search friends
Message from Skype(0): USERS a_sacchi, alexbernues, asacchi_ppc, carilla_edge, c
arillan, colecas, cristina3828, francois.willame, guillermo_tobias, iggyto, inra
nva, lemoncete, estarlus, mig_dg5, oya_gibi, xuxinho7, santiagosoria, gerryasamo
ah, torobravo7, vascorey, mariancika
call torobravo7
Message from Skype(0): CALL 109 STATUS ROUTING
Message from Skype(0): CALL 109 STATUS MISSED
CALL torobravo7
Message from Skype(0): CALL 114 STATUS ROUTING
Message from Skype(0): CALL 114 STATUS RINGING
Message from Skype(0): CALL 114 STATUS INPROGRESS
Message from Skype(0): CALL 114 DURATION 39
Message from Skype(0): CALL 114 STATUS FINISHED
```

Figure 64. Tear down of the call with torobravo7

When the call is accepted, the reply from Skype is stored in the file “MessageFromSkype.txt” (note that this is different from “MesaggeFromSkype.txt”):

CALL 114 STATUS ROUTING
CALL 114 STATUS RINGING
CALL 114 STATUS INPROGRESS

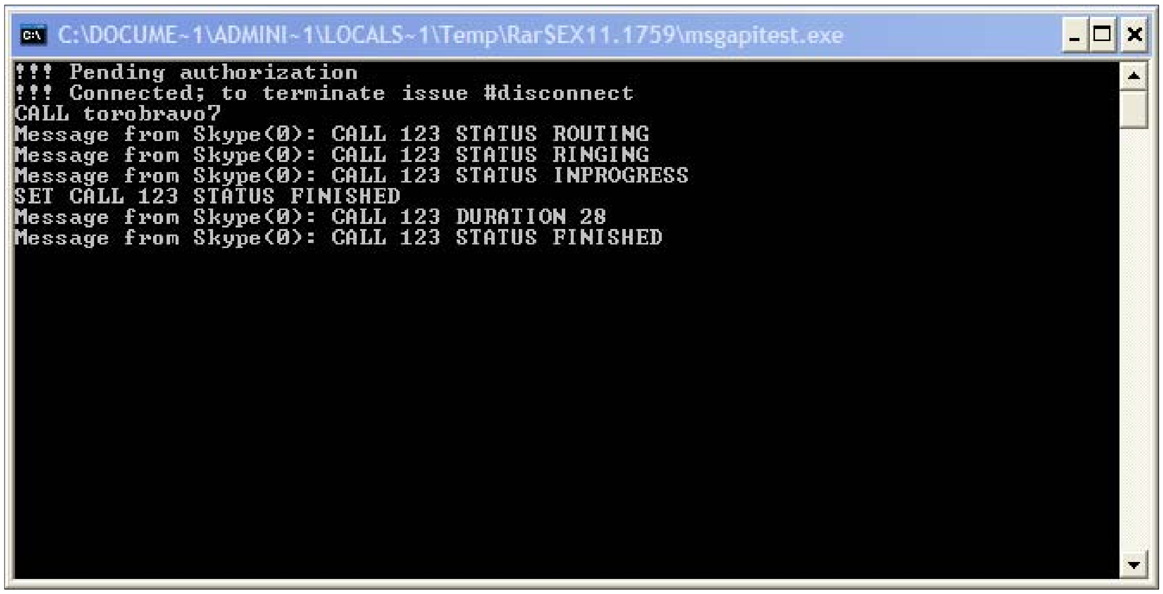
To tear down the call we read the “MessageFromSkype.txt” (different from the MesaggeFromSkype.txt) to obtain the identifier of the call and to create the command to close the communication. This identifier is retrieved with a new function called “readIdentifierCall” which is defined as:

```
void readIdentifierCall(FILE *fd, char *iden, char* data)
{
    int exit=0;
    int numBytes=0;
    int cont=0;
    bool bOtherOption=false;
    FILE *prueba;

    while (!exit)
    {
                if (fread(&data[numBytes],sizeof(char),1,fd)==0)
                {
                      exit=1;
                }
                else if (data[numBytes]=='\n')
                {
                      exit=1;
                }
                else if (bOtherOption && ((data[numBytes]>='0') &&
(data[numBytes]<='9')))
```

```
                    {
                          iden[cont]=data[numBytes];
                          cont++;
                    }
                    else if (bOtherOption && ((data[numBytes]<'0') &&
(data[numBytes]>'9')))
                    {
                          exit=1;
                    }
                    else if ((numBytes>1) && (data[numBytes - 1]==' ') &&
(data[numBytes]>='0') && (data[numBytes]<='9'))
                    {
                          iden[cont]=data[numBytes];
                          bOtherOption= true;
                          cont++;
                    }
                    numBytes++;
        }
        data[numBytes]='\0';
        iden[cont]='\0';
     }
```

The string to send is SET CALL<identifier> STATUS FINISHED and the code which performs this action is:

```
     // less than 5 characters
     char szIdentifier[5];
     szIdentifier[0]= '\0';
     szPrueba[0]='\0';

     readIdentifierCall(MessageFromSkype, szIdentifier, szPrueba);
     char szFinishCall[128];

     sprintf(szFinishCall, "SET CALL %d STATUS FINISHED", szIdentifier);

     COPYDATASTRUCT fCopyData;
     fCopyData.dwData=0;
     fCopyData.lpData=szFinishCall;
     fCopyData.cbData= strlen(szFinishCall)+1;

     if( fCopyData.cbData!=1 )
     {
              if( SendMessage( hGlobal_SkypeAPIWindowHandle, WM_COPYDATA,
(WPARAM)hWnd, (LPARAM)&fCopyData)==FALSE )
                    hGlobal_SkypeAPIWindowHandle=NULL;
     }// there is a command
```



C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\Rar$EX11.1759\msgapitest.exe

```
!!! Pending authorization
!!! Connected; to terminate issue #disconnect
CALL torobravo7
Message from Skype(0): CALL 123 STATUS ROUTING
Message from Skype(0): CALL 123 STATUS RINGING
Message from Skype(0): CALL 123 STATUS INPROGRESS
SET CALL 123 STATUS FINISHED
Message from Skype(0): CALL 123 DURATION 28
Message from Skype(0): CALL 123 STATUS FINISHED
```

Figure 65. Tear down call by sending the SET CALL identifier STATUS MESSAGE

Now I must note that instead of calling a Skype contact it is also possible to call a telephone number and this operation is the same as but simply exchanging the username for the telephone number. However, as it has been indicated earlier, the user should have credit in his or her SkypeOut account.

Actually, in my GUI there is another command sent to Skype in order to retrieve the status of the user and display it after the user pressing the "CHECK" button. When the "CHECK" button is pressed, the interface sends to Skype the message "GET USER <name> ONLINESTATUS as shown in the following code:

```
ConvertCToT(szNameRetrieved, szContact);
sprintf(szState,"GET USER %d ONLINESTATUS", szAuxiliar);
int contador= strlen(szAuxiliar);

if (hGlobal_SkypeAPIWindowHandle!= NULL)
{
          COPYDATASTRUCT sCopyData;
          sCopyData.lpData= szState;
          sCopyData.cbData= strlen(szState)+1;

          if (sCopyData.cbData!=1)
          {
                if (SendMessage(hGlobal_SkypeAPIWindowHandle,
              WM_COPYDATA, (WPARAM)hWnd, (LPARAM) &sCopyData)==FALSE)
                hGlobal_SkypeAPIWindowHandle=NULL;
          }
}
```

Now, Skype will reply by sending the status of the user and which we will save in "MessageFromSkype.txt". To read that value we call the retrieveStatus defined in fifthbutton.cpp as:

```
     void retrieveStatus(FILE* fd, char *data1, char *data2, int i)
     {
        int numBytes= 0;
        int exit= 0;
        int cont=0;
        FILE* prueba3;

        while (!exit)
        {
              if (fread(&data1[numBytes],sizeof(char),1,fd)==0)
              {
                    exit=1;
              }
              else if (data1[numBytes]=='\n')
              {
                    exit=1;
              }
              // remove the first part of the string. We only want the
status
              else if (numBytes > (41 + i))
              {
                    data2[cont]= data1[numBytes];

                    cont++;
              }
              numBytes++;
        }
        data1[numBytes]='\0';
        data2[cont]='\0';
     }
```

Appendix C Attacking Skype´s Security

To decipher the Skype protocol requires in several steps. Each one of these steps will provide us information needed in following steps.

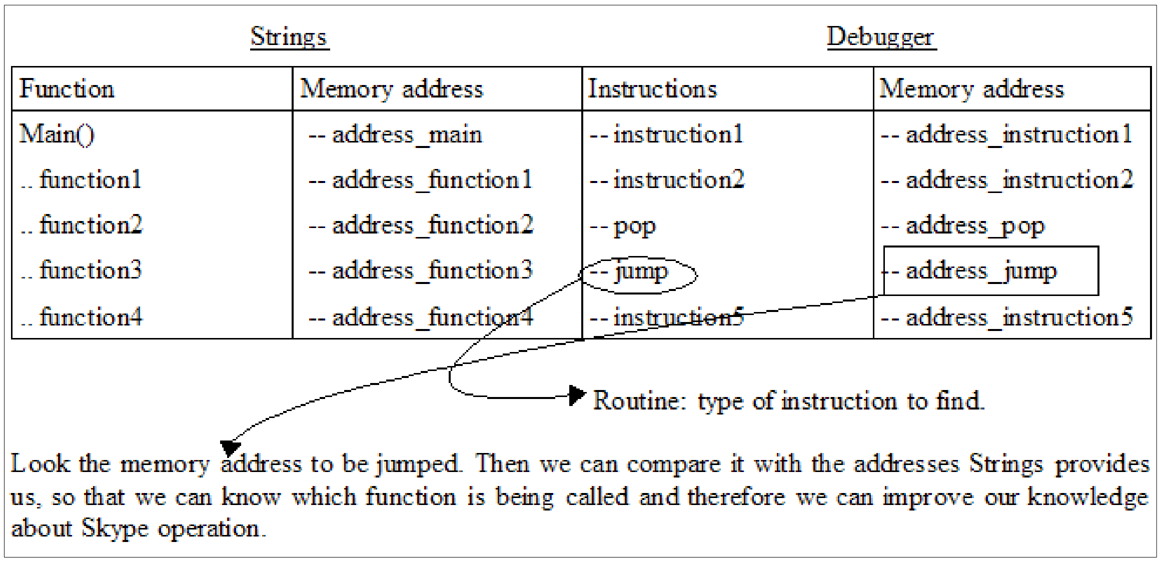
The first information we need is to know the behavior of the Skype protocol. To collect this information I have done a study with Ethereal [61] to expand beyond the earlier study [63]. It is important to note that this study used an earlier version of Skype.

What knowledge does this provide us? Both of these studies focused on the traffic sent by the Skype client; this allows me to discover and determine centralized structures, pattern of communications, as well find out which kind of information is exchanged with which other nodes.

The second step used the program called "strings" [65]. This program permits us to detect or find out possible strings (including the names of external functions used in the operation of the program Skype). We execute this program giving it the Skype executable file, and the program returns every string used as well as it address in memory. Thus, we learn the memory region where the functions, dll libraries, etc. are.

The third step is to use a debugger. We could use the debugger and study the output the debugger. The debugger will show the low level code of the program

In particular we can study the subroutines and other jumps. Running the program in the debugger can enable us to see the call graph of subroutines.



Strings		Debugger	
Function	Memory address	Instructions	Memory address
Main()	-- address_main	-- instruction1	-- address_instruction1
.. function1	-- address_function1	-- instruction2	-- address_instruction2
.. function2	-- address_function2	-- pop	-- address_pop
.. function3	-- address_function3	-- jump	-- address_jump
.. function4	-- address_function4	-- instruction5	-- address_instruction5

Routine: type of instruction to find.

Look the memory address to be jumped. Then we can compare it with the addresses Strings provides us, so that we can know which function is being called and therefore we can improve our knowledge about Skype operation.

Figure 66. Sources of information.

We can correlate this with Skype´s Internet packets. This will allow us to understand better what is happening.

Once we learn how to use these tools, it is now possible to select where to attack the program. As it was explained in the previous study, we want to examine the login process where four transfers between user and a centralized node which is the login server occur.

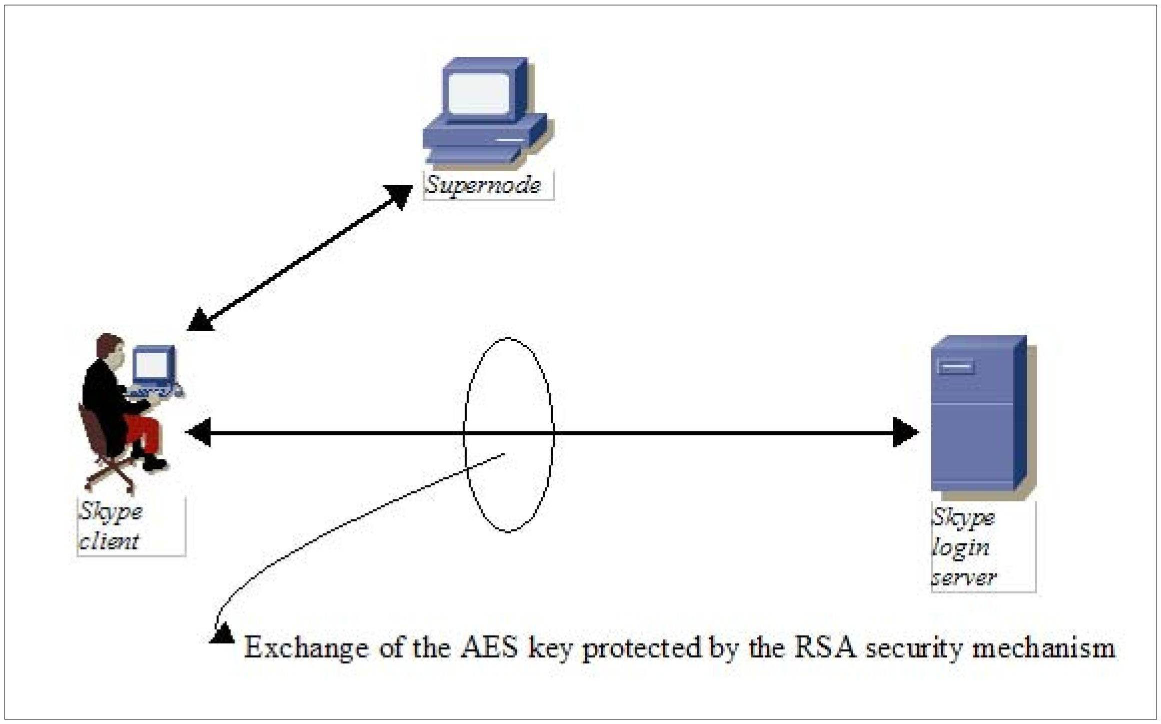

Supernode
Skype client
Skype login server
Exchange of the AES key protected by the RSA security mechanism

Figure 67. Login process in Skype

As a result of this transfer, there must be a copy of the RSA key as our program will need to use it to communicate with the server in later transfers. Thus, this is the key to be attacked or searched because we know that it is inside our computer. Finding this key will enable us to decode the packets which are sent.

Appendix D Common Graphical Interface

As noted earlier, I have created two GUIs which allow us to use either in a (laptop) PC or an HP iPAQ running Pocket PC. In the section 6.3.2, I have already explained the operation of the GUI for the laptop (since the operation of the interface under Pocket PC is very similar). In order to facilitate reading this code, I will expand my earlier explanation and will include references to the code. Not all the code will be explained because writing a program in Visual C++ requires generating code which defines the messages to control, the loop that manages and captures those messages, a relation between the message and the function to call when this message is captured, or the function to call when a identifier is pressed, etc, but at least, the reader will know which part of the code performs each operation. Neither it will be explained again the operation of the API since it was clarified in the appendix B.

Here, only the GUI for a PC running Windows is explained, but the primary difference from the GUI for the Pocket PC are:

- Different types of variables, with different sizes
- A lack of some functions for the Pocket PC
- Special code for this specific operating system

The knowledge necessary to implement the code was obtained mainly from the Microsoft website [66], and from three books:

- Programming Windows With MFC, 2th Edition [67],
- Programming Windows CE .NET, Third Edition[68], and
- Programming Windows, 5th Edition [69].

The common interface appears on our screen as the ”main window” of our interface which contains a main menu composed of four items: File, View, Contact, and Help. The definition of this window is given in the file InterfacePC2.cpp with the following code:

```
// Register application main window class.
 wc.style =0;                                  // Window style
 wc.lpfnWndProc = MainWndProc;           // Callback function
 wc.cbClsExtra = 0;                      // Extra class data
 wc.cbWndExtra = 0;                      // Extra window data
 wc.hInstance = hInstance;               // Owner handle
 wc.hIcon = LoadIcon (NULL, szAppName) ;   // Application icon
                                              // NULL = Standard
                                              // Icon
 wc.hCursor = LoadCursor (NULL, IDC_ARROW);// Default cursor
 wc.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH);
 wc.lpszMenuName = MAKEINTRESOURCE(ID_MENU);  // Menu name
 wc.lpszClassName = szAppName;           // Window class name
```

where it can be seen that this windows load a resource identified by ID_MENU, the name of the window is defined by the string szAppName, and the procedure executed when creating the window is MainWndProc.

Once the definition is stored in the WNDCLASS variable wc, the window will be created with:

```
// Create main window.
 hWnd = CreateWindow (szAppName, TEXT ("InterfacePC2"),
                           | WS_VISIBLE | WS_SYSMENU,
                   CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
                   CW_USEDEFAULT, NULL, NULL, hInstance, NULL);
```

The definition of the resource is written in the file InterfacePC2.rc:

```
//-------------------------------------------------------------------
    // Menu, the RC data resource is needed by the menu bar
    //

    ID_MENU MENU DISCARDABLE
    BEGIN
     POPUP "&File"
     BEGIN
         MENUITEM "Property Sheet",             IDM_SHOWPROPSHEET
         MENUITEM "Modeless Dialog",            IDM_SHOWMODELESS
         MENUITEM SEPARATOR
         MENUITEM "E&xit",                      IDM_EXIT
     END
     POPUP "&View"
     BEGIN
                MENUITEM "Tool Bar"                     IDM_TOOLBAR
                MENUITEM "Address Bar"                  IDM_ADDRESSBAR
                MENUITEM "State Bar"                    IDM_STATEBAR
     END
     POPUP "&Contact"
     BEGIN
                MENUITEM "Call to a contact"            IDM_CALLCONT
                MENUITEM "Message"                      IDM_MESSAGE
                MENUITEM SEPARATOR
                MENUITEM "Answer"                       IDM_ANSWER
                MENUITEM "Ignore"                       IDM_IGNORE
         MENUITEM SEPARATOR
                MENUITEM "Hang"                         IDM_HANG
     END
     POPUP "&Help"
     BEGIN
         MENUITEM "&About...",                  IDM_ABOUT
     END
    END
```

Inside the "File" item, we see a menu item called "Property Sheet" which is linked to an identifier IDM_SHOWPROPSHEET. That means when the user presses this menu item, a new message is generated sending this identifier. Our interface must catch this message and displays the property sheet structure. To do that we established a relation between the identifier and the function as next shown:

```
// Command message dispatch for MainWindowProc
   const struct decodeCMD MainCommandItems[] = {
        IDM_SHOWPROPSHEET, DoMainCommandShowProp,
            IDM_SHOWMODELESS, DoMainCommandModeless,
        IDM_EXIT, DoMainCommandExit,
        IDM_ABOUT, DoMainCommandAbout,
             IDM_HANG,DoMainCommandHang,
             IDM_CALLCONT, DoMainCommandCall,};
```

Thus, the function that will be executed upon receiving the IDM_SHOWPROPSHEET message is DoMainCommandShowProp and its code is:

```
LPARAM DoMainCommandShowProp(HWND hWnd, WORD idItem, HWND hwndCtl,
                             WORD wNotifyCode)
{
          // five tabs
    PROPSHEETPAGE psp[5];
    PROPSHEETHEADER psh;
    INT i;

     memset (&psp, 0, sizeof (psp));
   // Fill in default values in the property page structures (psp =
   // propertysheet page).
    for (i = 0; i < dim(psp); i++) {
             psp[i].dwSize = sizeof (PROPSHEETPAGE);
             psp[i].dwFlags = PSP_DEFAULT;
        psp[i].hInstance = hInst;
        psp[i].lParam = (LPARAM)hWnd;
    }
        // Set the dialog box templates for each page.
        psp[0].pszTemplate = MAKEINTRESOURCE (ID_KEYPAD);
    psp[1].pszTemplate = MAKEINTRESOURCE (ID_EDITPAGE);
        psp[2].pszTemplate = MAKEINTRESOURCE (ID_SCROLLPAGE);
        psp[3].pszTemplate = MAKEINTRESOURCE (ID_STATPAGE);
        psp[4].pszTemplate = MAKEINTRESOURCE (ID_LISTPAGE);

        // Set the dialog box procedures for each page.
        psp[0].pfnDlgProc = FirstButtonProc;
        psp[1].pfnDlgProc = SecondButtonProc;
        psp[2].pfnDlgProc = ThirdButtonProc;
   psp[3].pfnDlgProc = FourthButtonProc;
   psp[4].pfnDlgProc = FifthButtonProc;

   // Initialize property sheet header structure.
   // Propertysheert header contains the different propertysheet pages

   psh.dwSize = sizeof (PROPSHEETHEADER);
   psh.dwFlags = PSH_PROPSHEETPAGE;
   psh.hwndParent = hWnd;
   psh.hInstance = hInst;
   psh.pszCaption = (LPSTR) "Property Sheet";
   psh.nPages = dim(psp);
   psh.nStartPage = 0;
   psh.ppsp = (LPCPROPSHEETPAGE) &psp;

   // On Pocket PC, make property sheets full screen.
#if defined(WIN32_PLATFORM_PSPC) && (_WIN32_WCE >= 300)
    psh.pfnCallback = PropSheetProc;
    psh.dwFlags |= PSH_USECALLBACK | PSH_MAXIMIZE;
#else
    psh.pfnCallback = NULL;
#endif
//defined(WIN32_PLATFORM_PSPC) && (_WIN32_WCE >= 300)

// Create and display property sheet.
  PropertySheet (&psh);
    return 0;
}
```

where PropertySheet() is the function that creates the Propertysheet defined in DoMainCommandShowProp. Then the result when selecting the ”contact” tag in that propertysheet structure is shown in Figure 27.

This second tab presents several editlist and fields in which the user can introduce his or her own name and password. Again that tab corresponds to the loading of a resource when the proper identifier is selected. The next code shows the resource corresponding to this tab:

```
ID_EDITPAGE DIALOG discardable 0, 0, 150, 300
STYLE  WS_POPUP|WS_VISIBLE|WS_CAPTION|WS_SYSMENU|DS_MODALFRAME

CAPTION "Contact"
BEGIN

 LTEXT "Name of the user",   IDC_NAMEUSER       5,10,60,10
 EDITTEXT                     IDC_SINGLELINE,   70,5,60,15
 LTEXT "Password",      IDC_PASSWORD,     5,60,50,10,
 EDITTEXT                     IDC_PASSBOX,  70,55,60,15,ES_PASSWORD
 PUSHBUTTON "&Accept",        IDC_ACCEPT,       30,100,40,15
 PUSHBUTTON "&Cancel",        IDC_CANCEL,       80,100,40,15
END
```

Thus we have two edittext variables that can be used where you can introduce the username and the password of the client who is going to make calls using this interface. The operation of this tab is controlled by the procedure called ”SecondButtonProc” loaded previously in the propertysheet structure.

As I explained, when the user presses the “ACCEPT” button this data will be saved in two files. The code shown below performs that functionality and it is part of the procedure “SecondButtonProc”.

First we retrieve the data written inside the editlist with the function GetDlgItemText and we save them in two strings: szNameUser and szPassword.

```
GetDlgItemText(hWnd, IDC_SINGLELINE, szNameUser, dim(szNameUser));
     GetDlgItemText(hWnd, IDC_PASSBOX, szPassword, dim(szPassword));
```

Once we have these data we store them in two text files named “NameUser.txt” and “Password.txt” shown below:

```
if((result = fopen("NameUser.txt","w"))== NULL)
         {
              return FALSE;
         }
         else
         {
              ConvertTToC(szContact, szNameUser);
              fprintf(result,"%s", szContact);
              fclose(result);
         }// end NameUser
         if((result = fopen("Password.txt","w"))== NULL)
         {
              return FALSE;
         }
         else
         {
              ConvertTToC(szPass, szPassword);
              fprintf(result,"%s", szPass);
              fclose(result);
         }// end Password
```

The ConvertTToC function translates TCHAR[] variables to char[] variables. It is needed because the values represented in the editlist are TCHAR[] but in the text file we have to save a char[] variable. The definition of the function is:

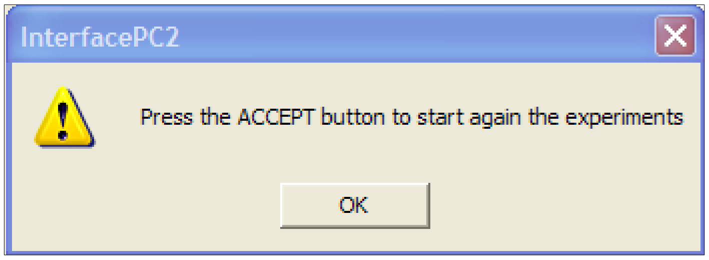
```
void ConvertTToC(CHAR* pszDest, const TCHAR* pszSrc)
   {
         for(int i = 0; i < _tcslen(pszSrc); i++)
              pszDest[i] = (CHAR) pszSrc[i];
}
```

Next we calculate in a randomly way which VoIP program we will call when making a call (that is, Minisip or Skype):

```
   srand((unsigned)time(NULL));
   randomValue = rand();
   randomVal = fopen("RandomValue.txt", "a");
   if (randomValue > threshold)
   {
              fprintf(randomVal,"The program selected was Skype\n");
              IsSkypeSelected=true;
   }
   else
   {
         fprintf(randomVal,"The program selected was Minisip\n");
         IsSkypeSelected=false;
   }
   fprintf(randomVal,"%d\n", randomValue);
   fclose(randomVal);
```

After selecting the program to be used we save this information in the RandomValue.txt file to use later when analyzing the data. We now know which user makes which call and with which underlying program.

If the user decides not to continue, he can close the interface, or start another registration by pressing the “Cancel” button. When pressing the ”Cancel” button a new dialog informing the user about the situation will pop up. The code to perform this action is quite simple:



Press the ACCEPT button to start again the experiments

Figure 68. Dialog displayed when the CANCEL button is pressed.

The editlists are initiallized and the dialog is called:

```
SetDlgItemText (hWnd, IDC_SINGLELINE,TEXT(""));
SetDlgItemText (hWnd, IDC_PASSBOX,TEXT(""));
MessageBox (hWnd, TEXT (" Press the ACCEPT button to start again
the experiments"), "InterfacePC2", MB_ICONWARNING) ;
```

However, if the user selects the ”ACCEPT” button a message to retrieve the contacts is sent since the connection has been established.

Now it is the time to continue with the experiment and we go to the tab called ”Contacts”. The result is shown in Figure 28.

Once we have retrieved our contacts, we have to display them in our interface. First, we have to initialize the two icons to present the state of the user:

```
      //INITIATE THE BITMAPS
hbmpConnected= LoadBitmap(hInst, MAKEINTRESOURCE(IDB_CONNECTED));
hbmpDisconnected=LoadBitmap(hInst, MAKEINTRESOURCE
hListBox= GetDlgItem(hWnd,IDC_MULTILIST);
```

Then, we have to read the contacts stored in the file ”IntermediateFile.txt” as was explained in appendix B. We will read each of the contacts and translate them to TCHAR[] format to display in our interface. When we have read one of our contacts and we have its TCHAR[] representation we establish a connection between the contact, the icon indicating its status (initially offline) and the listbox of our interface with the function ”AddItem”. Then we set the focus to be the handle of that listbox and send the message to incorporate the contact in the correct place.

After that, we increment the number of users read, and finally display the contact via our interface. The initial state is shown in the previous figure with no users connected.

```
if ((fileContacts = fopen( "IntermediateFile.txt", "r" ))!= NULL)
    {
          while( !feof(fileContacts))
```

```
            {
                  szContact[0]= '\0';
                  readLine(fileContacts, szContact);
                  {
                        i=0;
                        while (szContact[i]!='\0')
                        {
                              i++;
                        }
                        for (cont=i; cont < length(szContact); cont ++)
                              szContact[cont]='\0';
                        ConvertCToT(szNameRetrieved, szContact);

            //    INITIALIZE THE LIST BOX TEXT AND ASSOCIATE A BITMAP

                        AddItem(hListBox,szContact,hbmpDisconnected);
                        SetFocus(hListBox);
                        SendMessage(hListBox,LB_SETCURSEL,0,0);
                        SendDlgItemMessage(hWnd, IDC_MULTILIST,
                        disconnected++;
                  }// There is no contact
            }//end fileContacts
            fclose(fileContacts);

            wsprintf (szTitleUsers, TEXT ("FRIENDS (%d/%d)"), connected,
            SendDlgItemMessage(hWnd, IDC_NUMBERUSER, LB_ADDSTRING, 0,

      } else return FALSE;
```

The reason to put all the users in the offline at the beginning is that Minisip does not indicate the current state of the user. When the user wants to make a call, he will enter the name of the user in the editlist and press the “Check” button. At that time the program will check if the selected user is available (on-line) or not. Next, the “CALL” button can be pressed, and two situations are possible:

- It is possible to establish the connection since the user is connected
- It is not possible to establish the call because the user is not available.

In the second case, the dialog in Figure 29 will be shown.

Nevertheless whether the user is available, we will call him and the icon corresponding to that user will change its color to show him online. Moreover the position of such user will be located at the end of the list and the number of users connected will be increased in one as in Figure 30.

The code to perform all this process is shown below:

```
if (strcmp(data2,"ONLINE")!=0)
    // show a dialog
    MessageBox (NULL, TEXT ("This user is not connected"),"ERROR",
    else
    {
            // it is possible to call the user
            hListBox= GetDlgItem(hWnd,IDC_NUMBERUSER);
            wsprintf (szTitleUsers, TEXT ("FRIENDS (%d/%d)"), connected,
            SendMessage(hListBox, LB_DELETESTRING, 0, 0);
            connected++;
            wsprintf (szTitleUsers, TEXT ("FRIENDS (%d/%d)"), connected,
            SendDlgItemMessage(hWnd, IDC_NUMBERUSER, LB_ADDSTRING, 0,
            hListBox= GetDlgItem(hWnd,IDC_MULTILIST);
            int index = SendMessage(hListBox, LB_SELECTSTRING,-1,(LPARAM)
                  szNameContact);
            SendMessage(hListBox, LB_DELETESTRING, (WPARAM)index, 0);
            AddItem(hListBox,szContact,hbmpConnected);
            SetFocus(hListBox);
            SendMessage(hListBox,LB_SETCURSEL,0,0);
            }// USER SKYPE ONLINE
            fclose(MessageFromSkype);
}// USER IN MessageFromSkype
```

The code which defines the tab is included in "interfacePC2.rc" and is shown below here:

```
ID_LISTPAGE DIALOG discardable 0, 0, 150, 300
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | DS_MODALFRAME
CAPTION "Contacts"
BEGIN
   LISTBOX     IDC_NUMBERUSER,   5,5,125,18
   LISTBOX     IDC_MULTILIST,    5,25,125,60,LBS_HASSTRINGS|

   EDITTEXT    IDC_NAMECONTACT,  5,85,125,15

   PUSHBUTTON "&Call",           IDC_ACCEPT, 5,105,40,15
   PUSHBUTTON "&Hang",           IDC_CANCEL, 55,105,40,15
   PUSHBUTTON "&Check",          IDC_CHECK,  105,105,40,15
END
```

My common interface also provides a keypad to make calls similar the one presented in Skype. This keypad allows the user to enter a PSTN number by directly entering the phone number or by entering a contact name or a SIP URI. The dialog and the code to build are shown in Figure 31.

```
ID_KEYPAD DIALOG discardable 0, 0, 150,300
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | DS_MODALFRAME
CAPTION "Key pad"
{
     PUSHBUTTON "7",       55, 30, 55, 20,  20
     PUSHBUTTON "4",       52, 30, 30, 20,  20
     PUSHBUTTON "1",       49, 30, 5,  20,  20
     PUSHBUTTON "0",       48, 30, 80, 20,  20
     EDITTEXT               27, 45, 110, 90, 15

     PUSHBUTTON "8",       56, 60, 55, 20,  20
     PUSHBUTTON "5",       53, 60, 30, 20,  20
     PUSHBUTTON "2",       50, 60, 5,  20,  20
     PUSHBUTTON "9",       57, 90, 55, 20,  20
```

```
        PUSHBUTTON "6",       54, 90, 30, 20,  20
        PUSHBUTTON "3",       51, 90, 5,  20,  20

        LTEXT "Phone Number", IDC_NUMBER, 5, 110, 40, 130,

        PUSHBUTTON "Call", IDD_CALL, 60, 80, 20, 20
        PUSHBUTTON "Hang", IDD_HANG, 90, 80, 20, 20
    }
```

Using this keypad is as easy as dialing a telephone and pressing the "CALL" button on a typical VoIP phone. The code to control the keypad is included in the file firstButtonProc:

```
            // 48 is the value of the identifier of the "0" button
            szBuffer[j] = (TCHAR)('0' + (LOWORD(wParam) - 48));
            j++;
            szBuffer[j] = '\0';
            SetDlgItemText (hWnd, VK_ESCAPE, szBuffer) ;
```

Once the number is entered the user presses the "CALL" button and control is taken by the API function which will send the appropriate command to Skype to start the code.

The call will continue until the user presses the "HANG" button. Then, a dialog will pop up to permit us evaluate the call from the worst quality 1 to the best one 5 (this dialog will appear as well in the contacts tag when finishing the call). That dialog is shown in Figure 32.

The code to display the dialog and the definition of this dialog are:

```
    g_hwndMlDlg=CreateDialog(hInst,TEXT("Valuation"),hWnd,


    Valuation DIALOG discardable  5, 5, 120,  145
    STYLE  WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU |
  DS_MODALFRAME
    CAPTION   "Communication valuation"
    BEGIN
       LTEXT  "Your communication has finished", -1, 4, 6, 115, 20,
       LTEXT  "Please rate the communication (1-5)?:"  -1, 4, 30, 115, 20,

      PUSHBUTTON "-",       27, 25,  74, 70, 15
      PUSHBUTTON "1",       49, 15,  95, 15, 15
      PUSHBUTTON "2",       50, 35,  95, 15, 15
      PUSHBUTTON "3",       51, 55,  95, 15, 15
      PUSHBUTTON "4",       52, 75,  95, 15, 15
      PUSHBUTTON "5",       53, 95,  95, 15, 15

       CTEXT  "Thank you for your help!",    -1,   20,  120, 90,  20,
    END
```

The operation consists of introducing the evaluation form for the call, then the value will be taken from the dialog and saved in a file. This is done with the procedure shown below:

```
BOOL CALLBACK ValuateDlgProc (HWND hWnd, UINT wMsg, WPARAM wParam,
                              LPARAM lParam)
{
//definition of variables
static BOOL  bNewNumber = TRUE ;
static UINT  iNumber, iFirstNum ;
HWND         hButton ;
FILE *result;

switch (wMsg) {
   case WM_CHAR:
         if (hButton = GetDlgItem (hWnd, wParam))
   {
         // Take the handler of the control indicated in wParam
            SendMessage (hButton, BM_SETSTATE, 1, 0) ;
         // This message is sent by an application to change the
              Sleep (100) ;
        SendMessage (hButton, BM_SETSTATE, 0, 0) ;
   }
   else
   {
      MessageBeep (0) ;
      break ;
   }
   case WM_COMMAND:
         SetFocus (hWnd);
         if ((LOWORD (wParam)== IDOK )|(LOWORD (wParam) == IDCANCEL ))
         {
          DestroyWindow (hWnd);
           bNewNumber = TRUE;
          g_hwndMlDlg = 0;  // 0 means dlg destroyed.
          return TRUE;
         }

        if (isdigit (LOWORD (wParam)))    // decimal digit
        {
           if (bNewNumber)
           {
              iFirstNum = iNumber ;
              iNumber = 0 ;
           }
                   // hexadecimal function
          ShowNumber(hWnd, iNumber = 16* iNumber + wParam - (isdigit
          (wParam) ? '0':'A' - 10)) ;

          if((result = fopen("valor.txt","w"))== NULL)
          {
              fprintf(result, "test \n");
              return FALSE;
          }
          else
          {
              fprintf( result, "%d", iNumber);
              fclose(result);
          }
              DestroyWindow (hWnd);
             g_hwndMlDlg = 0;  // 0 means dlg destroyed.
```

```
                    return TRUE;
                }
            }//switch
              return FALSE;
        }
```
As described in section 6.3.2, it is also possible to make the call from the main menu. However, it is not necessary to explain the operation from the main menu since the code used is almost the same. The main difference is that we have to create a new function inside the InterfacePC2.cpp to call dialogs which are controlled by the functions explained previously in this appendix.

Finally, the header for all the functions and procedures I have written, along with the structures generated, and the identifiers defined have been included in a file named "InterfacePC2.h".

IMIT/LCN 2005-13