

RTP redirection using a handheld device with Minisip

OSCAR SANTILLANA



**KTH Information and
Communication Technology**

Master of Science Thesis
Stockholm, Sweden 2007

COS/CCS 2007-10

KUNGLIGA TEKNISKA HÖGSKOLAN
The Royal Institute of Technology

RTP redirection using a handheld device with Minisip

Master thesis Final Version

Oscar Santillana
osantillana@gmail.com

01/03/07

Supervisor and Examiner: Gerald Q. Maguire Jr.

Abstract

This report presents several different techniques for diverting RTP streams when using a handheld mobile device. This device is running a version of Minisip as the SIP user agent.

An introduction to the SIP protocol is given to provide some background to the reader prior to focusing upon the main goal: redirecting RTP streams. A set of requirements are defined and an RTP media transfer mode is chosen based upon these requirements. The requirements are derived from a study of a Linux cellular phone's mobile device features and capabilities. Minisip was ported to this platform and a series of tests conducted to evaluate the design decisions made. These tests show that the best method of redirecting RTP media streams is third party call control (3PCC).

Sammanfattning

Den här rapporten presenterar flera olika teknikerna för att dra RTP strömar när man använder en mobil anordning. Den här anordningen löper en version av Minisip som den SIP användare agent.

En introduktion till SIP protokoll är gjord för att ge läsaren någon bakgrund på focusen ovanför det huvudsakliga målet : omdirigerande de RTP strömarna. En set av bestämd behov är definierad och en RTP media transfer sätt är vald på grund av de här behoven. Behoven är härrörda från en studie över en Linux mobiltelefon. Minisip var installerad till den här plattformen och en serie av test dirigerad för att utvärdera de gjorda designsbesluten. De här testen visar den bästa metoden för att omdirigera RTP media strömar är den tredje part kalla kontrollen (3PCC).

Acknowledgements

I would like to sincerely thank Professor Gerald Q. Maguire Jr. for all the help and advice given to develop this thesis project. Without him, the realization of this project could not been possible.

I would like to thank also Motorola for the resources provided such as the Linux mobile phone and the SDK libraries. This could not be possible without the help of Fred Kitson and Mat Hans. The respective titles are:

- Fred Kitson, PhD, Vice President, Applications Research Center, Motorola Labs
- Mat Hans, PhD, Distinguished Member of the Technical Staff, Applications Research Center, Motorola Labs

Table of Contents

Abstract	i
Sammanfattning.....	ii
Acknowledgements	iii
Acronyms	vii
1. Introduction	1
1.1. Linux Smart Phones.....	1
1.2. Overview of the Thesis Project.....	2
2. Session Initiation Protocol.....	4
2.1. SIP Network Elements	5
2.1.1. User Agents.....	6
2.1.2. SIP Servers.....	7
2.2. SIP Messages.....	10
2.2.1. SIP URIs	11
2.2.2. SIP Requests.....	11
2.2.3. SIP Responses.....	12
2.2.4. SIP Headers	13
2.3. SIP Transactions.....	14
2.4. SIP Dialogs	15
2.5. Complementary Protocols.....	16
2.5.1. SDP.....	16
2.5.2. RTP and RTCP.....	16
3. Mobility.....	18
3.1. Types of Mobility Supported by SIP	19
3.2. Component Overview	19
3.3. Session Location.....	20
3.4. Session Mobility.....	21
3.4.1. Mobile Node Control Mode.....	21
3.4.2. Session Handoff Mode.....	23
3.4.3. Bicasting Method.....	24
4. Framework.....	27
4.1. The Mobile Device.....	27
4.1.1. Hardware Configuration	27
4.1.2. Architecture.....	28
4.1.3. Operating System.....	28
4.1.4. Application Framework.....	29
4.2. The Development Environment.....	29
4.3. Cross-Platform Development	30
5. Method for Redirecting RTP Streams	31
5.1. Choosing the Best Approach to Session Mobility	31
5.1.1. Transfer Mode Comparison	31
5.1.2. Chosen Transfer Approach	33
5.1.3. Test Scenario.....	33
5.2. Implementation	35
5.2.1. Minisip Port.....	35
5.2.2. Adaptation of Minisip	39

5.2.3.	GUI for the Motorola Phone	39
6.	Analysis.....	41
6.1.	Evaluation	41
6.2.	Analysis.....	41
6.2.1.	Port and adaptation of Minisip.....	41
6.2.2.	RTP transfer approach	46
6.3.	Study	48
6.3.1.	Transfer Response Time.....	49
6.3.2.	Media Independency.....	49
7.	Conclusion.....	52
8.	Future Work	53
	References.....	54
	Appendix A	57
	Appendix B	59

Table of Figures

Figure 1.1: Smartphone Marketshare – Q3/2006	2
Figure 1.2: Task Overview	3
Figure 2.1: User Agent Behaviour	6
Figure 2.2: Proxy Server Scenario	8
Figure 2.3: Registrar Server Scenario	9
Figure 2.4: Redirect Server Scenario	10
Figure 2.5: SIP Transactions	14
Figure 2.6: SIP Dialog	15
Figure 3.1: Mobility Scenario	20
Figure 3.2: Control Node Mode Call Flow	22
Figure 3.3: Session Handoff Mode Call Flow	23
Figure 3.4: NAT traversal using RTP Proxy	25
Figure 4.1: Front and Back Motorola E680i	27
Figure 5.1: Test Scenario	35
Figure 5.2: Obtaining the toolchain	36
Figure 5.3: Toolchain environment variables	36
Figure 5.4: Library Dependence	37
Figure 5.5: Library installation steps	37
Figure 5.6: Minisip binary installation steps	37
Figure 5.7: Exports content	38
Figure 5.8: Mount command	38
Figure 6.1: Environment variables in the phone	42
Figure 6.2: Phone's wireless configuration	43
Figure 6.3: Ser execution	43
Figure 6.4: Minisip textUI execution	44
Figure 6.5: SER's contact database	44
Figure 6.6: Minisip textUI call usage	45
Figure 6.7: Test 1 Simple Call	45
Figure 6.8: Test 1 Statistics	46
Figure 6.9: Minisip textUI mobileTransfer usage	46
Figure 6.10: Test 2 RTP transfer	47
Figure 6.11: Test 2 Statistics	48
Figure 6.12: Media independency Test	50

Acronyms

GUI	Graphical User Interface
HTTP	Hyper-Text Transfer Protocol
IMS	IP Multimedia Subsystem
IP	Internet Protocol
LAN	Local Area Network
MLI	Mobile Linux Initiative
NAT	Network Address Translation
OSDL	Open Source Development Labs
PSTN	Public Switched Telephone Network
RTCP	Real Time Control Protocol
RTP	Real Time Protocol
SDP	Session Description Protocol
SER	SIP Express Router
SIP	Session Initiation Protocol
SLP	Service Location Protocol
STUN	Simple Traversal of UDP through NAT
TURN	Traversal using Relay NAT
UA	User Agent
UPnP	Universal Plug and Play
VoIP	Voice Over IP
WiFi	Wireless-Fidelity
3G	Third Generation
3GPP	3rd Generation Partnership Project
3PCC	Third Party Call Control

1. Introduction

Transferring media streams from one endpoint to another is a widely used technique in both conventional and IP telephony. This thesis project proposes another approach of using this technique in a new environment. A user desires to transfer a video stream from his handheld device to a large display in the room which he has just walked into. In addition, it is possible to transfer the audio streams to high quality speakers in the room where the user has just entered.

IP Multimedia Subsystem [1] (IMS) is a standardised Next Generation Networking architecture for telecom operators and is rapidly becoming the de facto standard for real-time multimedia communications services. It uses Voice-over-IP (VoIP), which is based on the Session Initiation Protocol (SIP), and runs over the standard Internet Protocol (IP). Although IMS was originally specified for third generation (3G) mobile networks, it also provides a service deployment architecture for fixed or wireless networks, such as Wireless Local Area Networks (WLANs), and the public Internet [2]. IMS defines open interfaces for session management, access control, mobility management, service control, and billing. This allows the network operator to offer a managed SIP network, with all the carrier-grade attributes of the switched circuit network, but at a lower cost and with increased flexibility. In addition, the use of SIP as a common signalling protocol allows independent software developers to leverage a broad range of third party application servers, media servers, and SIP-enabled end user devices to create next generation services.

1.1. Linux Smart Phones

Manufacturers are increasingly turning to Linux as a strategic platform to deliver more capable mobile devices, increase flexibility, speed time-to-market, and lower costs.

Open Source Development Labs [3] (OSDL), a global consortium dedicated to accelerating the adoption of Linux and open source software, has announced that a Chinese handset manufacturer, Datang Mobile [4] has joined OSDL as an active member of the Mobile Linux Initiative (MLI). Figure 1, using data extracted from the Symbian [5] web page, shows that Linux's share of the smartphone market is around 22 percent in Q3 of 2006.

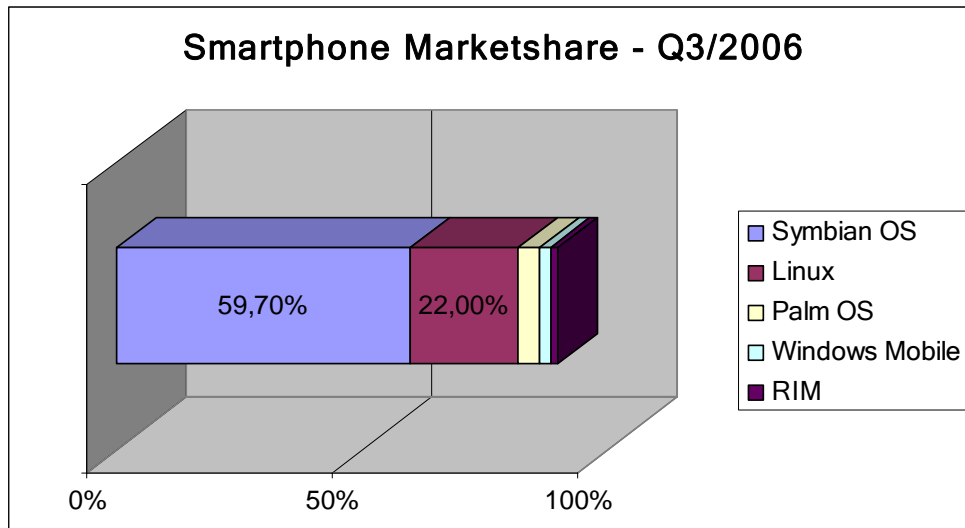


Figure 1.1: Smartphone Marketshare – Q3/2006

It is predicted that the mobile Linux handset market share will continue to grow, eclipsing SymbianOS. Using Linux as the operating system along with the GNU tool and other tool chains provides developers with a platform which they can easily develop applications for.

In this thesis project, a SIP application will be developed for an ARM-Linux environment. Motorola [6] has provided some tools to develop applications on their smartphone. The specific Linux phone used in this thesis project is the Motorola E680i. However, this is not the only Linux phone in the market. A complete list can be found at Linux devices web page [7].

1.2. Overview of the Thesis Project

This thesis project is focused on the development of new services in a mobile SIP environment. Specifically, the service is a variant of Session Mobility using RTP diversion (i.e. redirecting one or more RTP streams to new end points). This service will be described in detail in chapter three, among different techniques that could be used to realize this service. An important part of this thesis is an examination of each technique in order to decide which approach is the most suitable. An introduction to SIP, SDP, and RTP are given in sections 2.1, 2.5.1, and 2.5.2 respectively. Figure 1.2 shows the task which this thesis project addresses. It is not simply session mobility – since it is only the RTP stream which is being redirected to another device and not the control of the session. The discovery of new devices which could be the target of the migration is not considered in this thesis project – as this is work of other thesis projects.

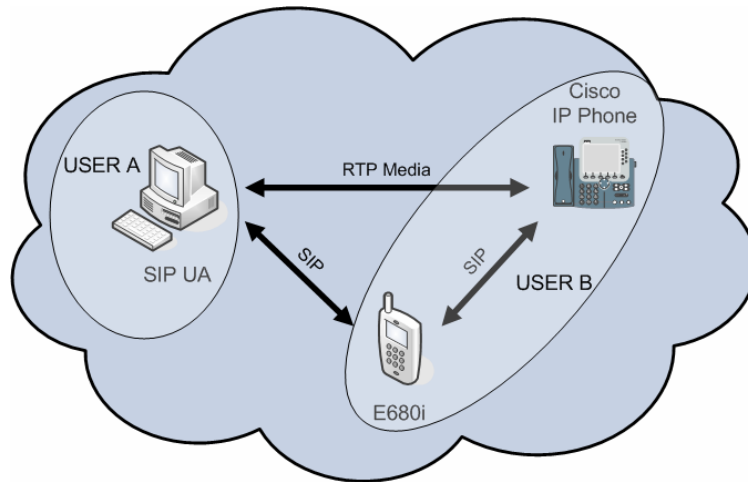


Figure 1.2: Task Overview

Once the means for transferring the RTP streams have been chosen, it is necessary to build a framework for this service in a mobile SIP environment. To achieve this, a mobile device from Motorola is used. This Motorola phone is Linux-based and will be examined in great detail in chapter four. The main part of this task was to port Minisip [8], a SIP User Agent (UA), to the ARM-Linux architecture of this mobile device in order to be able to carry out experiments.

Minisip is a SIP user agent like many others, but what differentiates it from others UAs is its focus on security. Moreover, Minisip is available for a number of different operating systems, such as: Linux for PCs, Linux Familiar for the IPAQ PDA, Windows XP, and soon Windows Mobile 2003 SE. These features and the possibility of video support, makes Minisip a great platform to extend to support RTP mobility.

Once the framework has been designed, it was necessary to adapt Minisip to the capabilities of the mobile device. This primarily required adapting the Graphical User Interface (GUI) to the specific of this mobile devices and implementing the RTP transfer service.

Finally, based upon some test scenarios a number of conclusions are drawn. These examine if the technique chosen did in fact meet the stated requirements.

2. Session Initiation Protocol

The Session Initiation Protocol (SIP) is an application-layer control protocol designed for creating, modifying, and terminating sessions with one or more participants. These sessions include Internet telephone calls with multimedia contents and multimedia conferences. SIP was specified in several RFCs, but the most important is the RFC 3261 [9], which contains the core protocol specification. In November 2000, SIP was accepted as a 3rd Generation Partnership Project [10] (3GPP) signalling protocol and a permanent element of the IMS architecture (see chapter 1). The SIP protocol is widely used as signalling protocol for VoIP [11].

SIP was designed (as many other successful Internet protocols) with the following goals:

Extensibility	the design takes into consideration future growth. SIP has to be able to support new scenarios, new multimedia services, and new uses
Flexibility	if new circumstances, environments, or purposes occur during a session, the protocol has the ability to adapt to them
Scalability	scaling from small or home office deployments to large-scale telecommunications networks

Personal mobility can be achieved because SIP transparently supports name mapping and redirection services. Thus users can be accessible with a single identifier despite their network location and despite their using more than one device. Note that in this thesis we will be concerned with moving the media *streams* and not the *session* (for details see section 3.4); however, a user can take advantage of SIP's support for personal mobility to change which device they are using for controlling their session.

SIP is used in multimedia communications for:

User location	specification of the end system to be used for a session
User availability	specification of the desire of the called party to start a session
User capabilities	specification of the media and media parameters to be used in a session
Session setup	establishment of session parameters at both called and calling party
Session management	transfer or termination of a session, modifying session parameters, and invoking services

SIP's main purpose is simply to enable a communication session; it is not a general purpose protocol. Communicating devices utilize other protocols such as Session Description Protocol (SDP) and Real-Time Transport Protocol (RTP) for their actual communication.

SDP specifies a format for describing streaming media parameters. It is used by each of the communicating endpoints in order to express its preferences and capabilities for a session. Note that new SDP exchanges can occur during a session to change which streams are sent, where they are sent, and to indicate which CODECs and other parameters are to be used.

RTP provides real time transport of the multimedia stream as created by one or more CODECs. RTP defines a standardized packet format for delivering media, such as audio or video, over the Internet. RTP supports the task of splitting, encapsulating, and transmitting multimedia data, as well as via RTCP providing means to monitor the transmission quality (delay, jitter, bandwidth...). Details of RTP are presented in section 2.5.2

SIP is based upon an end-to-end-oriented architecture; hence it is very scalable because it requires only a simple core. In fact SIP messaging only occurs at the setup of a session, during modification of a session, or at the termination of a session. Hence the signalling is proportional to the number of sessions and not to the duration of a session. Additionally, SIP is independent of the type of session, be it voice, video, timed text, etc. Moreover, SIP's naming scheme allows for a highly distributed architecture. These features are the basis of SIP scalability which enables a given SIP infrastructure to support a very large number of simultaneous sessions.

The Public Switched Telephone Network (PSTN) differs from SIP and its end-to-end approach because all the state is stored in the network rather than in the end-devices. Hence in the PSTN end-points have very limited functionality and any additional functionality is limited to that provided by the network infrastructure. It is for this reason that in traditional PSTNs new services are very hard to implement. While the aim of SIP is to provide similar functionality as traditional PSTNs, but to enable third parties to implement new services easily.

Finally, SIP builds upon lessons learned from the HyperText Transfer Protocol (HTTP) protocol [12]. HTTP is probably the most successful and widely used protocol in the Internet. HTTP in turn was based on the encoding of message headers from RFC 822 [13], which has been shown to be robust and flexible over many years.

2.1. SIP Network Elements

A typical network will contain more than one type of SIP element (as shown in figure 2.2). The simplest configuration uses only two user agents that send SIP messages directly to each other. The basic SIP network elements, which will be described below, are user agents, proxies, registrars, and redirect servers.

SIP entities are identified within a domain using a SIP URI (Uniform Resource Identifier). A SIP URI consists of a user name part and a domain part, delimited by the “@” (at) character. SIP URIs are very similar to e-mail addresses and because these names are resolved in different ways it is possible to use the same URI for both e-mail and SIP communication.

2.1.1. User Agents

RFC 3261 defines the SIP endpoints as User Agents, which are combinations of user agent clients (UACs) and user agent servers (UASs). The UAC is the only entity in a SIP network that is able to create an original request. On the other hand, the UAS receives requests and sends back responses. SIP UAs can be implemented in hardware such as IP phone handsets and gateways or in software as softphones running on a computer.

User agents can behave as a UAC or a UAS because they usually contain both entities. A user agent from a calling endpoint behaves as a UAC when it sends an INVITE request and receives responses to this request. A user agent from a called endpoint behaves as a UAS when it receives the INVITE and sends responses. However, if the called endpoint decides to send a BYE message to terminate the session, then both user agents simply change roles. Thus, the user agent that has sent the BYE message behaves as a UAC and the other user agent behaves as a UAS receiving it and sending back a response.

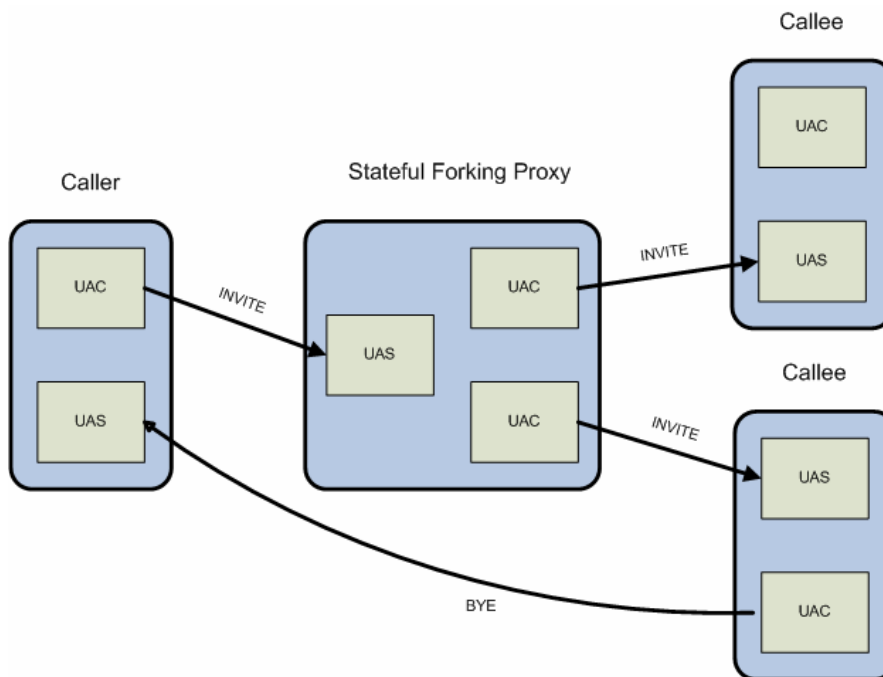


Figure 2.1: User Agent Behaviour

Figure 2.1 illustrates the behaviour explained above; here every user agent endpoint acts

as both a UAC and UAS. However, it is possible to have a UA which only implements the UAS behaviour - hence it can never initiate a SIP session. For example a network attached loudspeaker might only implement a SIP UAS.

2.1.2. SIP Servers

Even though the UA contains a server component, when most developers talk about SIP servers, they are referring to server roles usually played by centralized hosts on a distributed network.

2.1.2.1. Proxy Server

In a SIP network, the infrastructure may include a number of network hosts known as proxy servers. Given such an infrastructure, each UA can send messages to a proxy server and depend upon this proxy to forward the messages appropriately. Proxy servers play a very important role in such a SIP infrastructure because they can route session invitations depending on the location, authentication, accounting, or other attributes of the endpoints. Additionally, they simplify the configuration of UAs, much as the use of a default router simplifies the configuration of individual network attached computers.

The main task of a proxy server is to route session invitations to an endpoint while observing the preferences of both the caller and callee. In many cases, the session invitation may be routed by a set of proxies until the actual location of the called party is found. Finally, when the session invitation is delivered directly to the called party by the last proxy, the endpoint will accept or decline this invitation.

Proxies can be classified as outbound or inbound. Outbound proxies route messages generated within a local domain to an external domain. While, inbound proxies deliver incoming messages to the user's proxy – to which the user can delegate some processing (for example to enable context-aware call dispatch as described by Alisa Devlic [14] and Sergi Laencina [15]).

Each of these proxy servers can be stateless or stateful.

Stateless proxies	are very simple message forwarding entities. They forward messages according to some basic rules without being aware of any session state (i.e., they only use the information that is in the SIP message headers). As a consequence, they are very fast and can be used as load-balancers, message translators, or basic routers. On the other hand, disadvantages appear in message retransmissions and in lack of functionality to perform more advanced routing techniques as forking or recursive proxying.
-------------------	--

Stateful proxies are more complex than stateless proxies. When a request is received, stateful proxies create and maintain state until the transaction finishes. Some transactions, especially those created by INVITE, can last quite a long time. As a result, the performance of a stateful proxy is more limited because these proxies must maintain the state for the duration of the transactions and it takes a finite amount of space to store this state. Additionally, it takes time to retrieve this state when a message is to be handled.

The ability to associate SIP messages with a transaction gives stateful proxies some interesting features such as:

- Forking abilities when a message is received it is possible to send out two or more instances of this message
- Absorption of retransmissions the proxy knows from the transaction state if it has already received the same message or if a decision has already been made concerning how to handle this transaction

Most SIP proxies today are stateful and their configuration is usually very complex. They often perform accounting, forking, and offer some sort of NAT traversal aid. All of these features require a stateful proxy.

A typical scenario where a proxy server is deployed is illustrated in the following diagram.

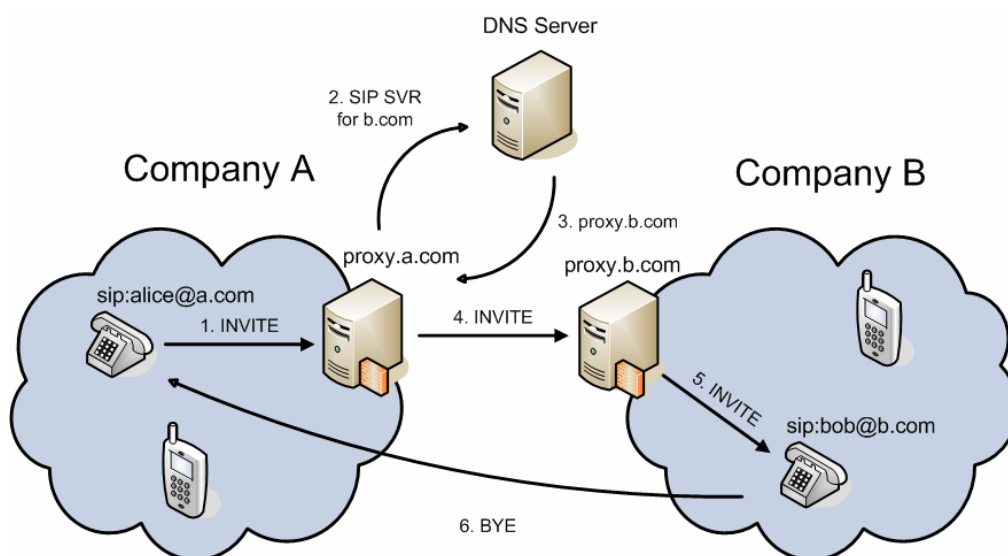


Figure 2.2: Proxy Server Scenario

In this scenario, Alice uses the SIP URI sip:bob@b.com to call Bob. Alice’s UA does not know how to route the invitation itself, but it is configured to send all outbound traffic to company A’s SIP server. This proxy server discovers that the user Bob’s URI is in the domain of another company (company B). As a result, the invite has to be forwarded to the other company’s proxy server. To do this, proxy A sends a request to a DNS server to find the SIP server associated with the domain “b.com”. The DNS server returns the location of proxy B thus proxy A can forward the invitation to proxy B if proxy B is aware of Bob’s current location, can forward this invitation to Bob’s user agent. If proxy B does not know Bob’s current location, it returns an error message to Alice, via each of these proxies – hence these proxies know that the transaction can not complete, thus it no longer needs to keep state information about this transaction.

2.1.2.2. Registrar Server

In order for proxy B in the above scenario to know about Bob’s location it has to be told this location by Bob’s UA. The registration process allows a SIP user to announce the address of a UAS. At least once such UAS must be registered in order to be reachable. When a UA starts, it sends a REGISTER message containing a contact header with this UA’s network location (i.e., an IP address and port of at least one interface) to a Registrar server. A Registrar server now knows where to find this SIP user within the specified SIP domain.

Figure 2.3 shows a typical SIP Registration. A register message is sent to the Registrar. The Registrar extracts the user’s account name and authentication, along with the UA’s location information and if the request is properly authenticated it stores the location information for this account into the location database. If the UA’s authentication was successful and the database update was successful then the Registrar sends a confirmation to the user agent; otherwise it sends an error message.

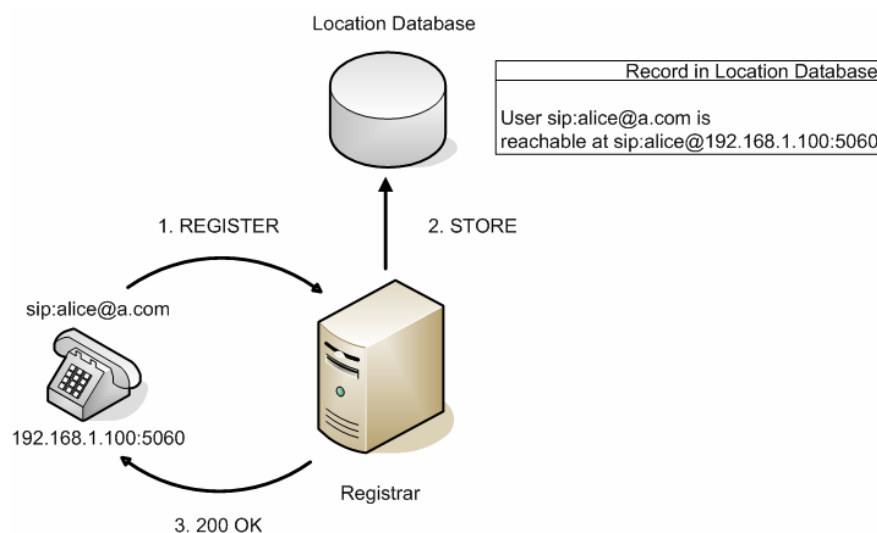


Figure 2.3: Registrar Server Scenario

Every registration has a limited life span. The REGISTER request includes an Expires header that establishes the user-to-location binding duration. The UA should renew its registration before it expires if it wishes to continue to be available.

2.1.2.3. Redirect Server

A Redirect Server is an entity that accepts a SIP request, maps the address into zero or more new addresses, and returns these addresses to the requestor. Unlike a proxy server, it does not accept calls but only generates SIP responses that instruct the UA to contact another SIP entity. The basic actions of a redirect server are shown in figure 2.4.

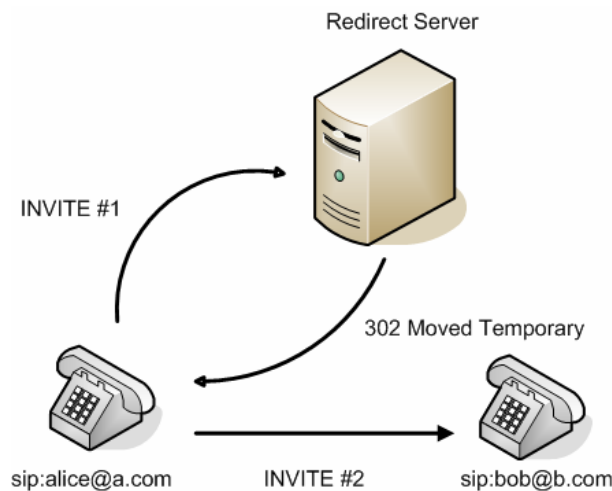


Figure 2.4: Redirect Server Scenario

2.2. SIP Messages

SIP messages are text using UTF-8 coded strings, compliant with the Unicode standard [16]. Each message in SIP is usually transported in a separate UDP datagram. However, SIP messages can be transmitted over several transport protocols, such as UDP, TCP, SCTP, or TLS (secure TCP). SIP messages are composed of a first line, which indicates the type of the message. Following this is one or more headers, which carry important protocol information and optionally a body section, which can carry any type of payload, but often is used to carry a session description using SDP. SIP messages can be divided into two types: requests and responses. Requests are usually used to initiate some action or inform the recipient of the request about an event. On the other hand, replies are used to confirm the reception and processing of requests and contain the status of the requested processing.

2.2.1. SIP URIs

A SIP URI identifies a communications resource. It also contains enough information to initiate and maintain a communication session with a resource due to SIP's routing scheme. The general format of a SIP URI is:

```
sip:user:password@host:port;uri-parameters?headers
```

As noted earlier, a SIP URI identifies a user at a host or within a SIP domain and might also carry special parameters required for the communication session. SIP URIs can be found in many sections and headers of SIP messages because they are a key element of SIP messages. It is the translation of these URIs to specific address, ports, and parameters to UAs which gives SIP its power.

2.2.2. SIP Requests

The first line of each request starts with the method name. The most commonly used methods in SIP are:

INVITE	requests another SIP UA to establish a new media session or to modify an existing session
BYE	requests a UA to terminate an established session
ACK	acknowledges the reception of a response
CANCEL	cancels a previously sent request
REGISTER	provides information about the location of a SIP UA to the SIP network

In addition, in order to implement new services such as Presence or Instant Messaging, new methods have been defined as SIP extensions. Some of these less commonly used messages are: INFO, OPTIONS, SUBSCRIBE, NOTIFY, UPDATE, MESSAGE, REFER, PRACK, and COMMET.

Finally, a typical SIP request is shown. The first part indicates the method, the second the headers, and in the third the body – a simply SDP session description is given.

INVITE sip:bob@b.com SIP/2.0	Method
Via: SIP/2.0/UDP 10.20.30.40:5060 From: Alice <sip:alice@a.com>;tag=589304 To: Bob <sip:bob@b.com> Call-ID: 8204589102@example.com CSeq: 1 INVITE Contact: <sip:alice@a.com> Content-Type: application/sdp Content-Length: 141	Headers

```
v=0
o=alice 2890844526 2890844526 IN IP4 10.20.30.40
s=Session SDP
c=IN IP4 10.20.30.40
t=3034423619 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Body containing SDP

2.2.3. SIP Responses

A SIP response is a reply from an UA or a proxy server due to a request message. Every request must be replied to except ACK requests, which do not need replies. Responses differ from requests in their first line, which contains the SIP protocol version (usually SIP/2.0) of the sender, a reply code, and reason phrase. The reply code is a number between 100 and 699 and indicates the purpose of the response. Responses can be divided into six groups:

- 1xx provisional responses - carry provisional information about the processing of a request. The sender must stop re-transmitting the request upon reception of a provisional response.
- 2xx positive final responses. A final response indicates the result of the processing of the associated request. Final responses also terminate transactions.
- 3xx These responses indicate redirections. For example, a new user location or alternative service to complete the call. Redirection responses are usually sent by proxy servers. When a proxy receives a request and cannot process it, it will send a redirection response to the calling party indicating a new location which the calling party might want to try. It is up to, the calling party to send a new invitation request to the new location given. Redirection responses are final.
- 4xx negative final responses. This type of response means that the problem was caused by the calling party. The request could not be processed because it contains bad syntax or cannot be fulfilled at that server.
- 5xx negative final responses to notify the calling party about a server failure. The request is apparently valid, but the server failed to fulfil it. Clients should usually retry the request later.
- 6xx when a request cannot be fulfilled at any server. This response is usually sent by a server that has definitive information about a particular user.

In addition to the response code, the first line also contains the reason phrase, which expresses the response in a human readable way.

The request to which a particular response belongs is identified using the CSeq header field. This header field also contains the method of corresponding request. A typical response received when a user agent tries to INVITE another party is the following:

SIP/2.0 200 OK	Method
From: Alice <sip:alice@a.com>;tag=589304 To: Bob <sip:bob@b.com>;tag=314159 Call-ID: 8204589102@example.com CSeq: 1 INVITE Contact: <sip:bob@b.com> Content-Type: application/sdp Content-Length: 140	Headers
v=0 o=Bob 2890844527 2890844527 IN IP4 10.20.30.41 s=Session SDP c=IN IP4 10.20.30.41 t=3034423619 0 m=audio 3456 RTP/AVP 0 a=rtpmap:0 PCMU/8000	Body containing SDP

2.2.4. SIP Headers

SIP headers are very similar to HTTP headers in both syntax and semantics. Messages use headers to specify a caller, callee, the path of the message, type and length of message body, and so on. The order of appearance within the headers sections is generally of no importance, except for the Via field, which always has to be at an early position. The most common headers are:

Allow	Lists the set of methods supported by the resource identified by the Request-URI
Call-ID	Uniquely identifies a dialog
Call-Info	Provides additional information about a caller or callee.
Contact	Provides URL(s), where the user can be found for further communications.
Content-Length	Indicates the size of the message body sent to the recipient.
Content-Type	Indicates the media type of the message body sent to the recipient
CSeq	Uniquely identifies a request within a Call-ID.
Encryption	Specifies that the content has been encrypted.
From	Indicates the initiator of the request.

Route	Determines the route taken by a request.
Subject	Indicates the nature of a call.
To	Specifies the recipient of the request.
Via	Indicates the path taken by the request so far.
WWW-Authenticate	Announces the client to send authorization information.

2.3. SIP Transactions

SIP messages are sent independently over the network, but are arranged into transactions. A transaction is a sequence of SIP messages exchanged between SIP network elements. A transaction is formed by a single request and all responses to that request, including zero or more provisional responses and one or more final responses. The purpose of the transactions in SIP is to achieve some degree of reliability for inherently unreliable protocols, such as UDP.

INVITE transactions are special because they might not include an ACK message. If the final response was not a 2xx response, then the ACK response is included in the transaction. Meanwhile, if the final response was a 2xx response, then the ACK is not considered part of the transaction. The reason for this difference is the importance of delivery of all 200 OK messages. These messages usually carry a description of a session in SDP, and it is vitally important that this message is received by the other party. Therefore, user agents retransmit 200 OK responses until they receive an ACK. Also note that only responses to INVITE are retransmitted.

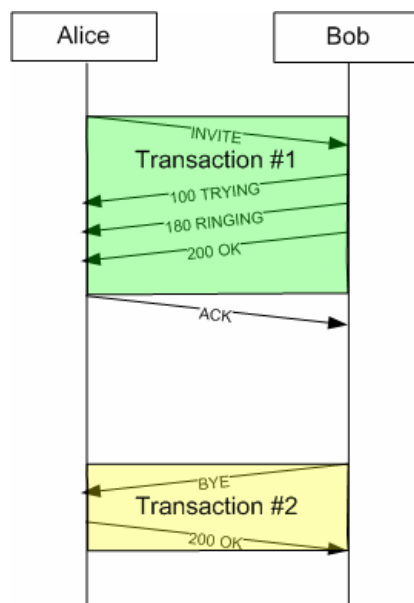


Figure 2.5: SIP Transactions

Every SIP message received at a stateful entity is matched against existing transactions, in order to determine whether it is a new request or a retransmission from a UAS, a response to a pending transaction, or even a misrouted response to a UAC.

For transaction matching, a transaction identifier is needed each message. This identifier is called the branch parameter, and it resides in the Via header.

2.4. SIP Dialogs

The purpose of the SIP protocol is to establish sessions between endpoints. The most important message used to establish such sessions is the INVITE. When a session is created via an INVITE, SIP internally creates a structure called a dialog. Dialogs are only created through a limited set of messages (currently: INVITE, SUBSCRIBE, and REFER), other messages such as REGISTER are strictly transactional.

Dialogs are identified by their *call-ID*, *from* tag, and *to* tag. All the messages with these three pieces of information belong to the same dialog. Dialogs facilitate the proper sequencing and routing of SIP messages between user agents. CSeq is used to order messages within a dialog. In fact, the CSeq number identifies a transaction within a dialogue.

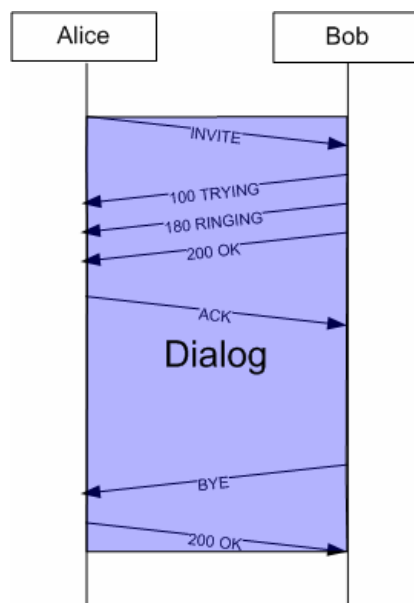


Figure 2.6: SIP Dialog

A dialog is composed of a sequence of transactions, in any direction, thus dialogs have a longer life span than transactions. When a Dialog is created at each endpoint, it is necessary to set up some state information. In case of INVITE dialogs, the BYE message is used to terminate the dialog, thus finishing the established multimedia session. In a SUBSCRIBE or a REFER dialog, in order to finish the dialog the NOTIFY message is used.

2.5. Complementary Protocols

As noted before, SIP's purpose is to establish a communication session possible. End-points then use other protocols such as SDP, RTP, and RTCP for their actual communication.

2.5.1. SDP

SDP specifies a format for describing media parameters to be used in a SIP session. It is described in RFC 4566 [17].

SDP is used within SIP to specify what kind of media, CODEC(s), addresses, and ports are available to be used in a session. Note that not all of these media, CODEC(s), etc. will necessarily be used in a session – but the SDP specifies which ones can potentially be used. SDP is included in the body of a SIP message. SDP messages can be divided into three categories of information:

- Session data and information to receive media (addresses and ports)
- Time description
- Media description comprising the session

2.5.2. RTP and RTCP

The Real-time Transport Protocol defines a standardized packet format for delivering media such as audio and video. In addition, the Real-time Transport Control Protocol provides out-of-band control information for an associated RTP flow.

RTP and RTCP were developed by the Audio-Video Transport Working Group of the IETF and initially was described in RFC 1889, now RFC 3550. RTP carries data that has real-time properties; while RTCP is used to monitor the quality of service and to obtain information about the participants in an ongoing session. The services provided by RTP are:

- Payload identification (which CODEC(s) were used)
- Sequence numbering
- Time stamping
- Delivery monitoring

The RTP protocol usually uses UDP to provide multiple connections between two entities, although RTP could use another transport protocol. It is important to note that RTP neither provides a means to provide a guaranteed QoS nor assumes the underlying network delivers packets in order.

RTCP periodically sends control packets to all session participants. Every RTP channel using port number N has its own RTCP protocol channel with port number equal to N+1. The services provided by the RTCP are:

- Provides feedback on the RTP delivery
- Transports a constant identifier for the RTP source (CNAME)
- Advertises the number of session participants which is used to adjust the RTP data transmission rate
- Carries session control information used to identify the session participants

3. Mobility

Mobile devices have been improving over the years; present devices include many features for IP-based multimedia communications. On the other hand these devices are still limited in terms of bandwidth, display size, and computational power. They still do not conveniently support user mobility. There is not yet a seamless transition between devices, such as stationary IP multimedia endpoints, hardware IP phones, videoconferencing units, and softphones. As explained in the last chapter, SIP has been chosen by the 3GPP as its standard for session establishment in the IMS and SIP is being deployed in both hardware and software IP multimedia clients. Therefore it is desirable to specify an architecture for seamless mobility for SIP [18].

In order to obtain a SIP-based seamless transition, two different methods have been proposed: third-party call control (3PCC) and the REFER method. They will each be explained in detail in the following sections. A new architecture has been proposed to achieve session mobility using these methods.

The main objective of this thesis project is to allow a mobile node to discover available devices and to include these devices into an active session (while not changing the locus of control of the session). To accomplish this objective, two main components are defined:

Service Location	Learning what devices are available area and their capabilities
Session Mobility	during a session with a remote device, to transfer an active media service to one or more devices

We will first introduce these components and then indicate why they are not sufficient to solve the problem which we pose - due to the constraint that we do not wish to change the locus of control - hence rather than session mobility we actually want to simply redirect the RTP streams and not move the session. This is because the user may want to redirect the streams to other devices and because some of the devices to which the user will redirect RTP streams may not even have a user interface - so in this later case there would not be any ability to control the sessions. To address the later case we will describe how session retrieval can be performed.

The discovery protocol proposed for this architecture is the Service Location Protocol (SLP) [19]. SLP is a service discovery protocol that allows devices to find services in a local area network without a prior configuration.

Session mobility requires the following:

Interoperability	every SIP-compliant device should work together with any other compliant device and should be capable of handling session transfers
Backward Compatibility	both mobility-enhanced and basic devices should be available as targets for a transfer

Flexibility	differences in devices capabilities, e.g. different CODECs used in a session should be addressed
Seamlessness	session transfer should be as transparent as possible for users

3.1. Types of Mobility Supported by SIP

SIP supports personal mobility and can be extended to support terminal, service, and session mobility. Each of these will be described below.

Terminal mobility allows mobile hosts to move between different subnets and still be reachable by other devices and to continue any ongoing session(s). Terminal mobility requires that SIP can establish a connection either at the start of a new session (pre-call) or in the middle of a session (mid-call). In the first case, the mobile device has to register its new IP address, to continue being reachable. The technique used in the second situation is to inform the communication peers about the new IP address. To do this the mobile device sends a new INVITE with updated information in the SDP body indicating the new IP address.

Session mobility allows a user to preserve a session while moving from one device to another. SIP provides two solutions, using third-party call control and the REFER method.

Personal mobility allows a user to be identified by the same logical address, even when the user is using different devices. The solution used by SIP involves forking proxies, which make the user's selection of their device transparent to a third party.

Service mobility allows a user to use a set of services independently of the device or the network attachment points. SIP utilizes a home server that stores the personal information profile for a user. If a user wants a service from a given device, the device contacts the home server. This provides access to all the relevant details about this user, along with the authorized set of services.

3.2. Component Overview

Session mobility involves five basic entities: The Correspondent Node (CN), the Mobile Node (MN), the local devices, an SLP Directory Agent (DA), and, optionally, a Transcoder.

The Correspondent Node is a basic multimedia endpoint being used by a remote participant. It could be for instance, a SIP UA. A Mobile Node is a mobile device incorporating a SIP UA with SIP-handling and device discovery capabilities. Local devices are located in the user's local environment; upon discovery they can be used in the current session. Basic devices include an IP phone without special capabilities, but with a SIP UA. The SLP Directory Agent is aware of devices – and knows their location and capabilities. Finally SIP-based transcoding services might be necessary in order to translate between format media streams.

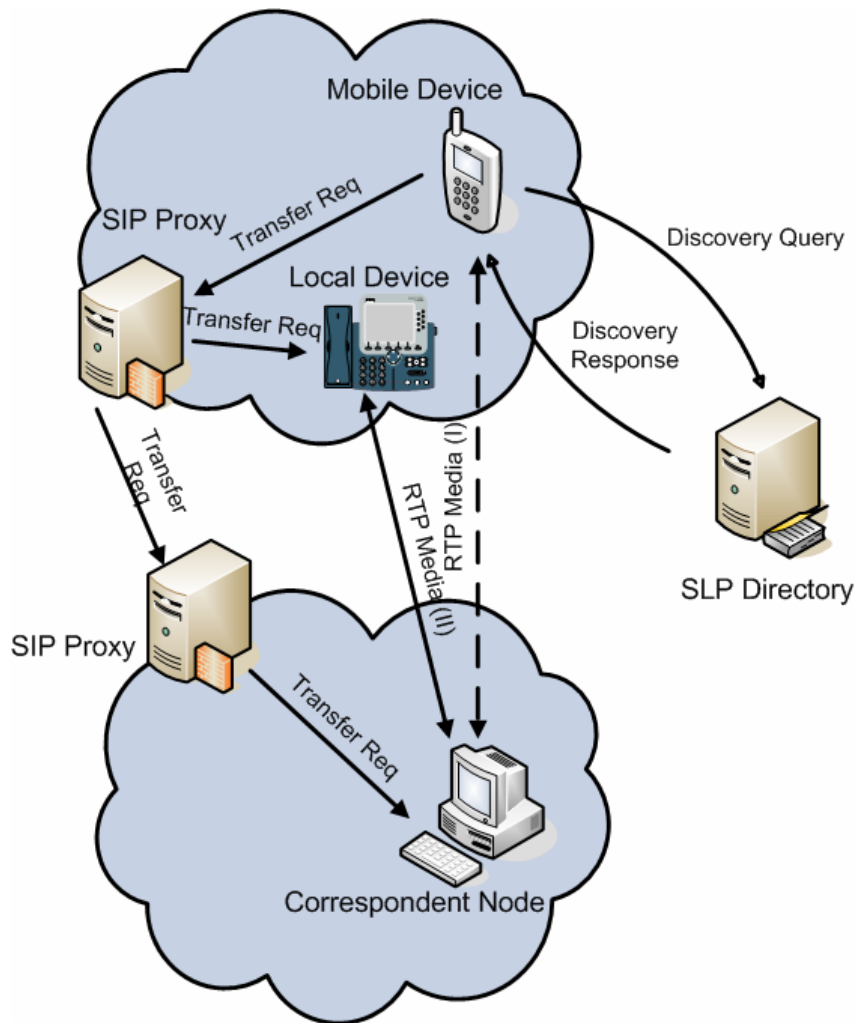


Figure 3.1: Mobility Scenario

Figure 3.1 illustrates all the components involved during session mobility. First of all, a Mobile Device with advanced SIP capabilities is exchanging media (via RTP) with a Correspondent Node in a media session. When the Mobile Device arrives at a new network, it asks the SLP Directory about what services are available and finds a Local Device suitable for the media stream it wishes to send or receive. Then, the Mobile Device sends to its SIP proxy a transfer request that depends on the transfer mode. The SIP proxy sends an INVITE to the Local Device and to a proxy that is able to reach the Correspondent Node. The Transcoder (not shown) might be used if the Local Device does not fulfil the Correspondent Node media requirements.

3.3. Session Location

Peer discovering is a requirement for mobile devices to achieve session mobility. Bluetooth is a direct method used by many devices to discover peers in close proximity (for limitations of this discovery method see the thesis of Cécile Ayrault [20]). Other methods are centralized, such as the Service Location Protocol. The main advantage of these different methods is the discovery of devices at different location granularities. On

the other hand, they have the disadvantage of requiring mobile devices to discover their location in order to perform such queries. However, a number of service discovery protocols are based upon a local broadcast – so the co-location with the other device/service is implicit.

3.4. Session Mobility

In this section several issues concerning session mobility will be explained in detail, specifically transfer and retrieval differences, media transfer possibilities, and the transfer modes.

Transfer and retrieval of a session are an important part of session mobility. A transfer moves the current session from one device to one or more other devices. While, retrieval means to remotely transfer a current session from a remote device to the local device. For instance, if a user discovers a large display using his mobile device, the video media stream could be transferred to this display. However, when the user and their mobile device leave this room, the media session should return to the mobile device. After this retrieval the communication session continues using the device's own display.

Session media streams may either be transferred to a single device or be split across several devices. In the last example, when the user discovered a large display and transferred the video media stream, the video stream was the only media stream transferred – thus the audio stream remained at his or her mobile device. However, this audio (output) stream could be transferred to a local amplifier and speaker system. This was possible because it is possible to independently transfer each direction of a full-duplex communication to one or more devices.

In order to transfer media sessions there are two different modes: Mobile Node Control mode and Session Handoff mode. In addition there is a third mode called RTP bicasting - this involves another entity, a RTPproxy. The following sections will describe each of these modes

3.4.1. Mobile Node Control Mode

Using Mobile Node Control transfer mode, the mobile node uses third-party call control. This establishes sessions between other nodes, hence the use of term third party. A node updates its session with the CN, using a new set of SDP parameters to establish media sessions between the CN and each device to which media streams are being transferred. The main disadvantage of this technique is that it requires the mobile node to remain active in order to maintain the sessions – this may consume resources (particularly power battery).

Figure 3.2 shows the Mobile Control transfer mode following Third Party Call Flow as specified in the RFC 3725 [21]. This is the simplest mode because it requires no manipulation of the SDP by the mobile node and works for any media types supported

by the endpoints. We have assumed that there is not a timeout problem, as the endpoints should answer immediately.

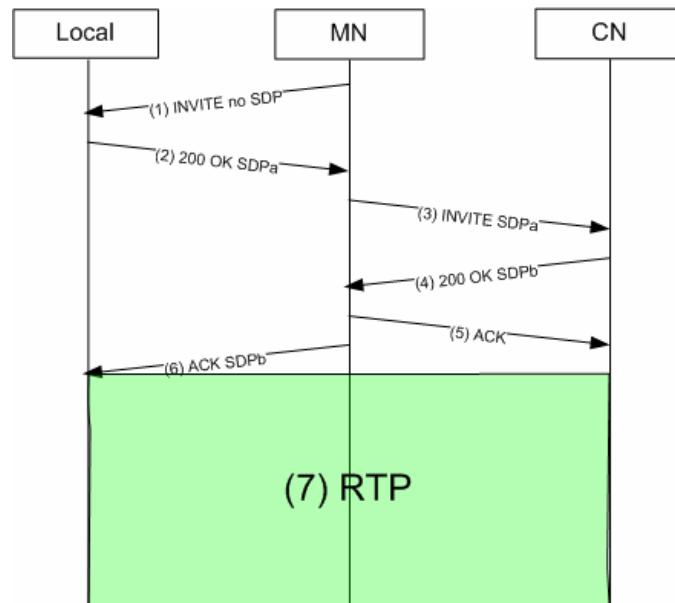


Figure 3.2: Control Node Mode Call Flow

Initially, MN sends a SIP INVITE (1) request to the local device (here labelled as “Local”), without an SDP body, requesting a new session to be established. As a consequence, the local device responds with a 200 OK (2) with an SDP body that includes the address and ports it will use for any media, and also a list of CODEC(s) it supports for each type of media. Next, the MN sends a RE-INVITE (3) to the CN in order to send it the updated session description. This request contains the local device’s media parameters in the SDP body. Note, that the MN might change the local device’s SDP depending on the type of media that it wishes to transfer to the CN. Afterwards, the CN sends a response (4) and includes, in its SDP body, the media parameters that it will use; these might be different from those used in the present session. Finally the MN acknowledges each endpoint, but in the local device’s acknowledge (6) it sends the SDP information concerning the relevant stream to/from the CN. Finally a RTP session (7) is established directly between the local device and CN.

When multiple devices are involved in a transfer it may be necessary to make a small modification to the above call flow. In order to split a session across multiple devices, the MN establishes a new session with each local device using a *separate* INVITE request. As a result, the MN updates the existing session with the CN with a SDP body that combines the media parameters of the multiple devices to be involved in the transfer. Finally the CN responds with its parameters and the MN has to send the relevant information to each of the respective nodes.

Next there is an example of SDP used in a multiple devices scenario with multiple combined media parameters (such as audio and video).

```

v=0
m=audio 48400 RTP/AVP 0
c= IN IP4 audio_dev.example.com
a=rtpmap:0 PCMU/8000
m=video 58400 RTP/AVP 34
c= IN IP4 video_dev.example.com
a=rtpmap:34 H263/90000

```

Finally if the MN needs to retrieve the session, it has to send a new INVITE to the CN with its own address in the parameters, this will cause the media streams to return to the MN. Subsequently, it sends a BYE to each local device in order to tear down these previous sessions.

3.4.2. Session Handoff Mode

Session Handoff Mode is based on the SIP REFER method. This method was described in RFC 3515 [22] and indicates that the recipient, identified by a Request-URI, should contact a third party using information from the request. Refer-To is a request header field that only appears in a REFER request. This header provides an address for the third party.

Figure 3.3 illustrates how a transfer is performed using a REFER request. Once the transfer is completed the “referer” does not belong to the session anymore. However, using the retrieve method that will be explained in next section it is possible to recover the session.

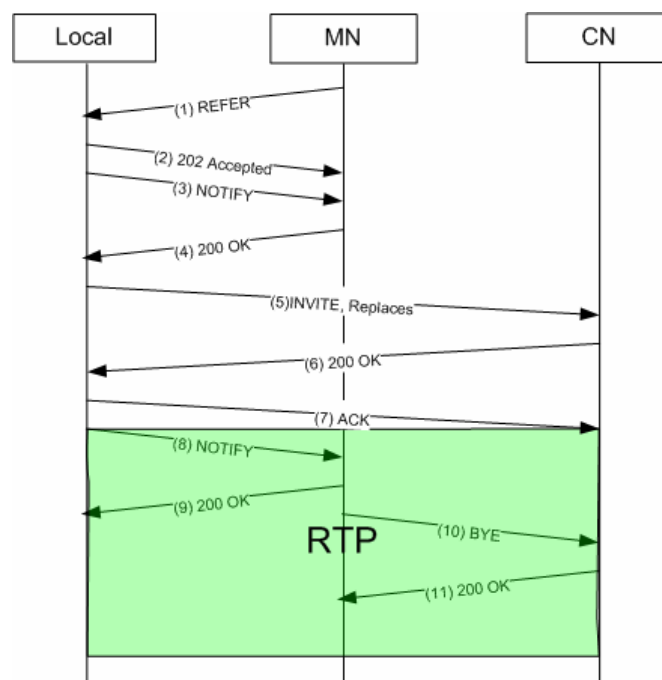


Figure 3.3: Session Handoff Mode Call Flow

First, the MN sends a REFER request (1) to the local device. The header Refer-To contains information about the URI of the CN. When the local device receives the request it should ask for user confirmation (assuming that the request is well-formed). If the refer is confirmed by the MN, then the local device will send a 202 accepted response (2). Next the local device sends a NOTIFY request (3) in order to inform the MN about the status of the reference. Then, the local device sends an INVITE request (5) with the “Replaces” header. This header identifies an existing session that should be replaced by the new session. The following responses correspond to the confirmation of the new invitation (6) and the acknowledgement (7) per part by the local device. After the ACK is sent to CN, another NOTIFY request (8) is sent to the MN. This message informs the MN of status the refer. As a result, the MN sends a BYE message (10) to the CN because the transfer has been successful. At this point the MN is no longer part of the session and need not remain powered on.

Unfortunately, a transfer to multiple devices using this mode is not as easy as in the Control Node approach. Splitting a session requires multiple media sessions to be established between the CN and local devices, without the MN controlling the signalling. This could be achieved using several REFER requests to local devices, referring each one separately to the CN. The problem is that currently there is no standard way to associate multiple sessions with a single call in SIP. As a result, each session between a local device and CN will be treated as a separate call and this does not fulfil the seamlessness requirement (as stated at the beginning of the chapter).

Finally in order for the MN to recover the session it is necessary to initiate another session with the CN to replace the current session. The MN needs to receive a REFER from the local device, in order to recover the old session. This can be achieved if the user can use the local device’s interface to cause it to send a REFER to the MN. Otherwise, it is possible to recover the session using a “Nested REFER” (RFC 3892 [23]). A nested REFER is based on indicating in the header Refer-To an URI indicating the original REFER method. Then, when the local device receives this request, it automatically sends a REFER to the MN and the session retrieval can be performed.

3.4.3. Bicasting Method

This method is not based on any SIP extension; instead it uses a new entity to support the mobile SIP scenario. This new entity is called an RTP Proxy. The RTP Proxy is a symmetric proxy designed to be used in conjunction with a SIP proxy, such as SIP Express Router (SER) [24]. This SIP Proxy has to be able to rewrite SDP bodies in SIP messages that it processes. This approach of rewriting SDP has already been used quite a bit, as SIP does not work well with NATs, thus sometimes communication through a NAT is not possible, however, using a RTP Proxy is one possible solution (along with others, such as Simple Traversal of UDP through NAT (STUN) servers [25], Universal Plug and Play (UPnP) [26], or Traversal using Relay NAT (TURN) [27]).

Figure 3.4 shows the RTP Proxy integration with a SIP Proxy and how these achieve NAT traversal.

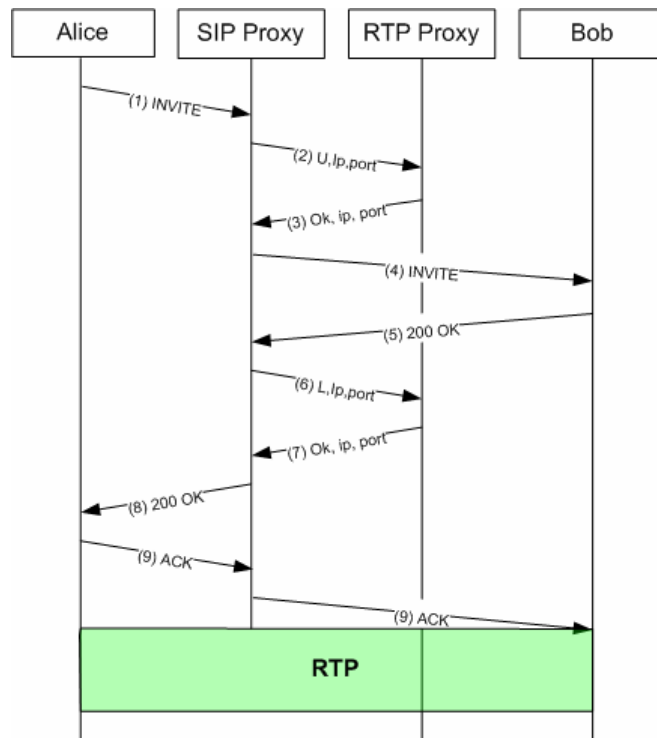


Figure 3.4: NAT traversal using RTP Proxy

Alice and Bob are in two different network domains, where Alice is behind a NAT. The SIP messages are the same as usual. However, when the SIP Proxy receives the first SIP message, it detects that Alice is behind a NAT, thus it initiates communication with an RTP Proxy. It communicates to this RTP proxy Alice's IP address and port. The RTP proxy responds with the IP address and port which should be given to the called party. Given this information the SIP proxy rewrites the SDP body of the INVITE and forwards it to Bob. Bob's behaviour is the same as usual, thus if Bob is available and wishes to accept the call his UA responds with a 200 OK message. When the SIP proxy receives this message, it sends the IP address and port number information from Bob's SDP to the RTP Proxy. In response the RTP proxy returns a new IP address and port which Alice should use. This SIP proxy rewrites the SDP body and forwards this message to Alice. Finally, Alice sends an ACK response that is forwarded to Bob via the SIP Proxy.

The assumption here is that the RTP Proxy is able to reach each end-user and that it can match the IP address and port information, so that when it receives an RTP stream from Alice, is able to relay the stream to the port and IP address that Bob is listening to. For details of a similar approach see the thesis of Gustav Söderström [28].

Bicasting replicates the RTP stream at the RTP Proxy. This can be used to support a soft handover [29] when the location of the mobile node is not clear. For example, the RTP Proxy can send the RTP stream through different networks, such as WLAN and GPRS. Thus it is possible to ensure that the mobile node will receive the RTP stream despite the location and connectivity of the mobile node.

This approach can be adopted to support session mobility – where it offers benefits which are not possible with the other approaches. This Bicasting mode is useful in a

number of different situations, e.g., when the MN wants to keep the RTP stream when doing a transfer. Another possibility occurs when the MN situation is not clear and there are many devices to transfer media to. In such a situation it would be possible to replicate the RTP stream to several devices, thus reaching the selected device.

To achieve these objectives, the MN has to be able to directly communicate with the RTP Proxy or do so through a SIP proxy.

4. Framework

This chapter presents the framework used for the thesis project. This framework is based on a mobile device, a development environment, some software development kits, and some adaptation to the Minisip UA in order to provide suitable session mobility – as required in the mobile SIP scenario given in section 3.2.

4.1. The Mobile Device

In this section, an overview of the Motorola Linux phone will be given. The specific Motorola phone which I have used is model E680i. This phone can be currently found in the Asian market. The specific phone used in this study was enabled as a developer's phone at the company's research laboratory. The phone has a PDA form factor (109 x 53.8 x 25 mm) with a touchpad based screen. Further details of the phone are given in the next subsection.



Figure 4.1: Front and Back Motorola E680i

4.1.1. Hardware Configuration

The E680i phone's hardware description [30] is:

CPU	Intel Xscale Bulverde revision 7 (PXA 270) 312MHz CPU, with support for OMA's digital rights management (DRM) Phase 1
RAM	32 MB
Flash Memory	50 MB of internal end user memory
Weight	133g

An essential part of the hardware configuration is the phone's display. The display's characteristics are:

Screen Resolution	320 x 240 pixels
Screen Dimensions	320 x 240 mm viewing area
Pixel Pitch	0.156 mm pitch, square
Color Depth	16 bits
Maximum Colors	65K colors

4.1.2. Architecture

The architecture of the E680i is ARM-based [31] (as noted above the CPU is a PXA 270). ARM CPUs have become the de facto standard by powering the majority of high end mobile devices, due to the following features:

- Algorithms can be implemented efficiently, thus reducing CPU, memory, and power requirements.
- High performance core – which can provide significant processing power when needed.
- Wide range of software tools.
- Low power consumption (with support for various power saving mechanisms).
- Low cost of silicon.
- Wide support for related hardware, software components, lots of developers, etc..

This architecture has three layers: application, service, and driver layers. All layers are Linux-based; however the application layer includes both a Java Virtual Machine and an Application Manager.

The main layer, upon which this thesis is focused is the application layer where the Minisip UA will run , this in turn depends on the underlying, Service layer, specifically the APIs related with the graphical user interface, connectivity, multimedia, system, and network.

4.1.3. Operating System

The Motorola phone runs an embedded Montavista Linux Consumer Electronics Edition 3.0 [32] (MVLCEE) and has the following features:

Linux OS services	Linux kernel version 2.4.20 with Bulverde support package memory management, interrupts and exceptions, kernel synchronization, process management, file systems, networking, etc.
Standard drivers	USB, UART, SPI, I2C, Flash drivers, GPIOs, power management, audio drivers etc

As it has been detailed before, the Motorola phone has 50MB of free space for useutilities, 32 MB of RAM and has the possibility to expand its storage space using an SD memory expansion slot.

4.1.4. Application Framework

The EzX GUI framework is based on the Trolltech's Qt Embedded GUI toolkit [33]. The current version available on the Motorola phone is 2.3.8. QT is a cross-platform application development framework widely used for the development of GUI programs. Some QT-based applications are the KDE desktop project [34] and web browser Opera [35]. QT uses standard C++ but can also be used by programmers using other languages such as, Python, Ruby, Java, and etc.

The services provided by QT are the following:

- Inter object communication using Signals and Slots.
- Events
- GUI primitives such as buttons, combo-boxes, scrollbars, etc
- Advanced user interface controls such as list views, progress bars, etc
- Window and Dialog Manager

4.2. The Development Environment

EzX is used to provide a smart phone. Such a device combines the features of a PDA, an internet appliance, and a multimedia player. EzX is a software development environment where application developers can use the tools and interfaces provided by the software development kit to develop their own application to run on EzX phones.

The development environment has been built to run on a PC running Linux with Kernel 2.4 or above. In order to set up this environment some basic knowledge of Linux is necessary. Moreover, some knowledge of cross-platform development is useful to build applications for an architecture such as ARM. In the next section, the required cross-platform basic knowledge will be explained.

The EzX software development kit also includes a plug-in for the Eclipse IDE tool [36]. Eclipse is an open source platform-independent software development environment for creating internet applications. Eclipse offers an IDE with a Java compiler and a full

model of the Java source files. Eclipse employs plugins in order to provide all of its functionality, in contrast to some other IDEs where such functionality is typically hard-coded. For example using plug-ins eclipse can be extended to support programming languages such as C, C++, and Python. In this case we have used it to support the development of Minisip in C++.

The Eclipse plug-in integrates an EzX Montavista tool chain into Eclipse and also supports onboard debugging via Eclipse using a remote gdb-server. For further information about software components required, installation and configuration, and how to onboard debugging via the USB or WiFi links, it is necessary to have access to the Motorola internal document provided with this software development environment.

4.3. Cross-Platform Development

To generate code for the phone, I used a cross-compiler. A cross-compiler is a compiler capable of creating executable code for a platform different than the one on which the cross-compiler is running. This technique is particularly useful when it is necessary to compile for a platform that is not accessible or is not convenient or difficult to compile on (as is the case with embedded systems, on microcontrollers with a minimal amount of memory).

For cross-platform development a toolchain is needed to build the cross-compiled executable. A toolchain is a set of utilities that are used to create another executable. These tools are usually used in a chain, so that the output of each tool becomes the input for the next. A simple software development toolchain consist of a text editor for editing source code, a compiler and a linker to transform source code into an executable program, and libraries to provide interfaces to the operating system.

When building cross compilation tools, usually there are two different systems involved: the system on which the tools will run, and the system for which the tools generate code.

- The system on which the tools will run is called the **host** system
- The system for which the tools generate code is called the **target** system.

Here we have used a compiler which runs on a GNU/Linux system and generates ELF programs for an ARM embedded system. In this case the GNU/Linux system is the host and the ARM ELF system is the target.

It is possible to create a cross-compiler with several GNU Tools, such as gcc [37], binutils [38], and uclibc [39], but it can be a difficult to configure the tool chain properly. Another alternative is to use a toolchain already created from another person. Toolchain for arm-processors include the one provided with the EzX SDK, the one recommended by the Minisip authors [40].

5. Method for Redirecting RTP Streams

Chapter 3 has given an overview of several ways to implement session mobility in a mobile SIP environment. In this chapter we examine which technique is most suitable for the purposes of this thesis project.

Once the transfer method had been selected, it is important to decide how to implement it, in order to accomplish the objectives of this thesis project. The methodology will be explained in great detail in section 5.2.

5.1. Choosing the Best Approach to Session Mobility

We begin with an evaluation of all three approaches. The advantages and disadvantages of each approach will be detailed. Next we describe a specific approach that has been selected for this thesis project. Finally we will describe a test scenario that will be used to evaluate our selection.

Bicasting has been excluded as it is not a SIP based solution and because it requires the addition of a new network node. However, we emphasize that it could be used together with the approach which is selected; but this remains for future work.

5.1.1. Transfer Mode Comparison

In this section a comparison between Mobile Node Control Mode (3PPC) and Handoff Mode (REFER) will be performed. The objective is to find the method that best fulfils the requirements of this thesis project. First we consider the requirements for this service:

Media independent	The solution should be independent of the media transferred in the session
Mobile environment	The solution has to be suitable for use in a mobile environment
Simplicity	The solution should be as simple as possible due the resource limitations at the end points
Low response time	The solution should be fast and not depend on the response time of the end points
Scalable	The solution should support as many participants as possible
Easy integration of new services	The addition of new features should be easy

The selected choice should fulfil as many requirements as possible. The advantages and disadvantages of the two transfer modes are shown in the following table.

Table 5.1

3PCC	
Advantages	Disadvantages
Simple	MN remains active as a central point
No changes in SDP	Timeout problem
Works with any kind of media	INVITE without SDP
Multiple device transfers	Used in midcall control
REFER	
Advantages	Disadvantages
Decentralized	SIP entities have to support REFER
No timeout problem	Endpoints more complex
	No Multiple device transfers

The principal 3PCC advantage is that is very simple approach and it does not need any extension of the SIP protocol to work. Another advantage is that no SDP body changes are necessary and it works with any kind of media - as long as this media is supported by both parties. Finally it is possible to perform multiple device transfers using a new session for each.

The main drawback of using 3PCC is that it requires a central point of control, in this case the MN, which might be not desirable. It is important to note that the MN is a mobile device with limited resources (such as battery power), so using this approach, the signalling of the session will still be controlled by the MN - as a consequence MN resources will be used. There is also a problem of timeout as already explained in section 3.4.1. It has been reported that some UAs do not behave as expected when they receive an INVITE without a SDP body. Finally 3PCC can only be used during midcalls.

On the other hand, the main advantage of using REFER method is its decentralized architecture. The MN need not take part in the signalling for the session once the media is transferred. Thus, the MN's resources will be saved. Moreover the timeout problem is solved using the SUBSCRIBE and NOTIFY requests that will inform the MN of the current situation.

However, as it has been explained before, the REFER request is an extension of the SIP protocol, so not all the SIP entities support this feature. Moreover, the endpoints have to be more complex because of this decentralized architecture. Finally it is not possible to make multiple device transfers using this approach.

Finally a resources comparison will be performed. As explained before the 3PCC approach consumes more resources than the REFER approach. When redirecting RTP streams, there are three possibilities:

- Being the central control point of the redirection with RTP and SIP support
- Being the central control point of the redirection with only SIP support
- Not being part of the transfer any more.

The first approach was discarded because it has high resource consumption due to the RTP redirection and the SIP signalling. The second approach, known as 3PCC, has the advantages of maintaining control of the session and it has lower resource consumption because the RTP redirection work has been transferred to another device. Finally the third approach, known as REFER, has minor resource consumption compared to the other two, but the device still has to listen in order to recover the control of the session if and when desired.

5.1.2. Chosen Transfer Approach

The approach that best fulfils the requirements is Mobile Node Control Mode (3PCC). This technique is much simpler than the REFER method. This is particularly important as most current UAs, do not implement the REFER extension. Minisip has implemented this extension, but still is in development. Additionally, the timeout issue is not a problem because in the proposed scenario there will not be any delay due to a UA. The media independence is very useful because not only can an audio stream be transferred, but so can a video stream. Thus makes it possible to start watching a video via the mobile phone and when it discovers a large screen display, it could send the video stream to this large display's UA. Note that since there is likely to be a significant difference in the total resolution of this new display another video stream might be selected by the source. Moreover, most prestored video sequences are likely to be available in multiple formats - due to the wide spectrum of devices and the very large differences in the data rates required for different resolution displays.






The main drawback of this approach is that the MN is the central point of communication. Thus signalling still will be continued to be received as was explained in the previous section. However SIP signalling is used at the beginning of the communication, while the RTP media is being exchanged the signalling is not used so often. Thus, it is possible for the MN to spend most of its time in sleep mode – and only waking (perhaps once every 100ms) to listen if there is signalling traffic or not.

Therefore, 3PCC fulfils the requirements. However, is still necessary to check the behaviour of the UAs to see what happens when they receive an INVITE without SDP and to measure the latency of the transfer. These issues will be evaluated in a test scenario, as described in next section.

5.1.3. Test Scenario

This analysis evaluates the chosen transfer approach (see section 5.1.2). In addition, we examine the behaviour of different UAs when they receive an INVITE without SDP and measure the response time of the transfer requests.

The entities that are involved in this scenario and their configurations are:

 MN	IP address: public address obtained by DHCP Port: 5060 UA: Minisip for ARM SIP: 1006@130.237.15.221
 SIP proxy	IP address: 130.237.15.233 Port: 5060 Version: 0.94 Mode: Proxy + Register
 CN	IP address: 130.237.15.222 Port: 5060 Cisco IP Phone model 7960 series SIP: 1005@130.237.15.222
 LOCAL	IP address: 130.237.15.247 Port: 5060 UA: Xlite 3.0 SIP: 1000@130.237.15.247
 Discovery Server	IP address: 130.237.15.233 Port: 427

All the IP addresses of the entities are public in order to avoid any NAT problems (hence avoiding the need to use a STUN server or an RTP Proxy). Figure 5.1 shows the desired behaviour of the UAs, Proxy, and Discovery Server. The call flow starts with a media session between the Cisco IP Phone and the Motorola phone. Once, the mobile node E680i discovers that there is another device (a softphone indicated as Local) to continue the session, it initiates the transfer mechanism.

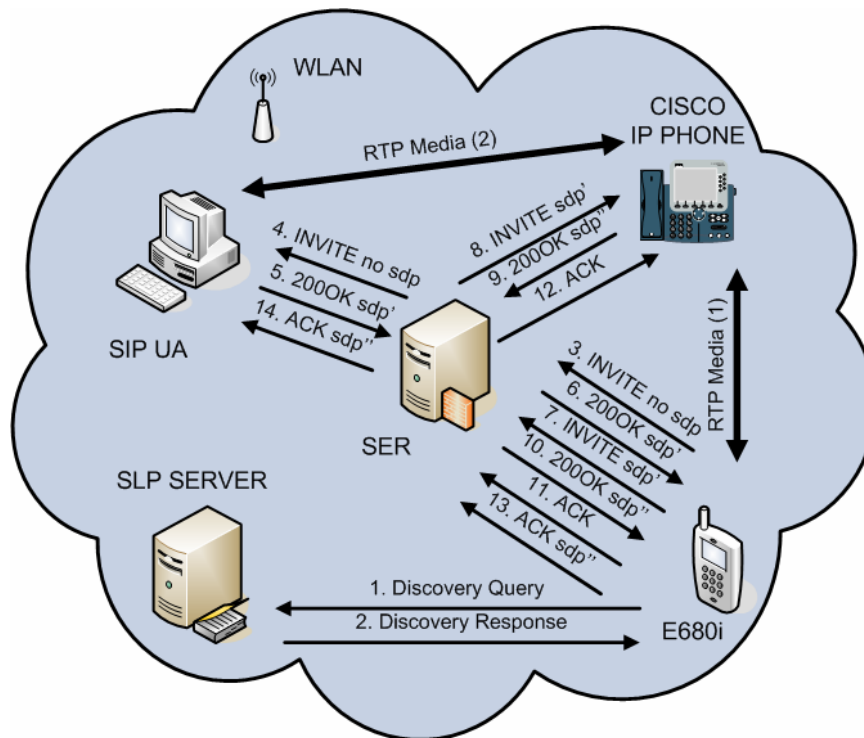


Figure 5.1: Test Scenario

As explained in the transfer method section, the E680i phone acts as a central point of the communication. After the transfer the RTP media is sent by the Cisco IP phone directly to the SIP UA although the SIP signalling control point is still the E680i smartphone.

5.2. Implementation

In this section, a detailed methodology for implementing the selected solution will be presented. The steps involved in the implementation are:

- Port a preliminary version of Minisip to the Motorola Linux phone.
- Adapt the Minisip source code to implement the selected transfer approach.
- Create a GUI for Minisip using the SDK provided by Motorola.

Each of these steps will be divided into subtasks. In the subsequent sections each of these subtasks will be described.

5.2.1. Minisip Port

We began by porting a stable version of Minisip to the Linux phone. Although this preliminary version might have reduced features and only used on a text user interface it established the basic SIP UA functionality for this platform.

To generate the necessary executable code required the utilization of a toolchain. There are several toolchains for the ARM platform. The Minisip project recommended the utilization of a toolchain from HandHelds.org. This toolchain can be retrieved as shown below:

```
oscar@ccsmot:/home/oscar> wget http://www.handhelds.org/download/projects/  
toolchain/arm-linux-gcc-3.3.2.tar.bz2
```

Figure 5.2: Obtaining the toolchain

Another toolchain that can be used to build binaries for this phone is the toolchain provided by the EzX SDK from Motorola.

Both toolchains require setting some system environment variables in order to be able to compile and link code for this Linux phone. In the EzX SDK this process is done by an configuration script.

```
export PATH=/path_where_the_toolchain_is/bin:$PATH  
export PKG_CONFIG_PATH=/path_where_the_toolchain_is/lib/pkgconfig  
export CC=arm-linux-gcc  
export CXX=arm-linux-g++  
export AS=arm-linux-as  
export AR=arm-linux-ar  
export NM=arm-linux-nm  
export LD=arm-linux-ld  
export RANLIB=arm-linux-ranlib  
export LD_SHARED="arm-linux-gcc -shared"
```

Figure 5.3: Toolchain environment variables

Minisip UA is based upon several libraries. In order to compile the final binary it is necessary to first compile and install each library correctly. The libraries which Minisip is based on are:

- libmutil
- libmcrypto
- libmnetutil
- libmikey
- libmstun
- libmsip
- libminisip

The order of compilation and installation is important because some libraries have dependences on previous ones. The proper compile and installation is shown in figure 5.4. This figure does not depict libmcrypto and libmstun because they are not part of the core framework.

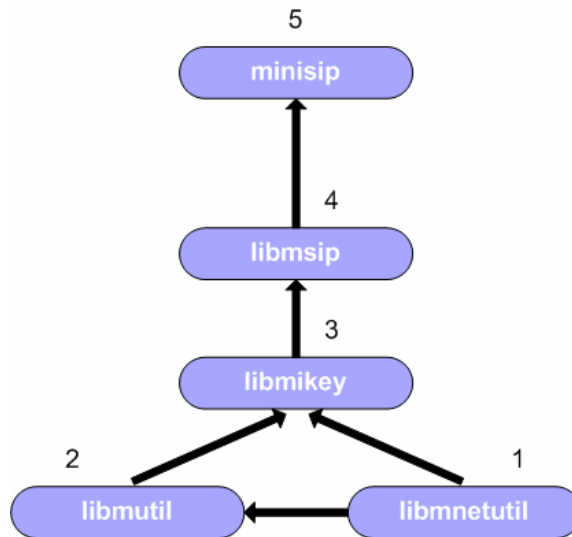


Figure 5.4: Library Dependence

In order to compile and install each library, the steps to follow are to connect to the relevant directory for this library, then:

```

./bootstrap
./configure --host=arm-linux --prefix=/path_where_the_toolchain_is/
make && make install
  
```

Figure 5.5: Library installation steps

There is a special library, libmcrypto that depends upon another library outside the Minisip project. This external library is Openssl [41] and libmcrypto requires a version 0.97b or greater. The Motorola SDK provided version 0.98b.

When all the libraries are correctly installed, then the Minisip binary can be built. The steps to build a simple statically link version with only the command line interface (“text GUI”) are:

```

./bootstrap
./configure --host=arm-linux --prefix=/path_where_the_toolchain_is/
--enable-textui --disable-gtk --enable-static
make && make install
  
```

Figure 5.6: Minisip binary installation steps

The binary has been built in same part of the file system where the toolchain is. Once the binary has been linked it is time to upload it to the phone using, e.g. an NFS server to share directories between the computer and the phone or a file transfer program. As Linux distributions have a NFS server as a default service, it is often only necessary to start it. Before starting it, it is important to modify the file “/etc/exports” to specify the directory where the executable file is to be found mounted file system. Note that is also possible to export more of the file tree, but for our purposes that is not necessary.

```
The contents of /etc/exports is:  
/home *(rw, sync)  
/home/oscar/Motorola/debug 192.168.1.2(rw, async, no_root_squash)
```

Figure 5.7: Exports content

The Motorola debug directory will be exported as a mounting point. The address 192.168.1.2 is the IP address of the phone, this and the parameters allow the phone to mount this directory and to read and write files to this directory. Note that it is useful to make this directory writeable in order to store debugging and test information on the cross compiling host - as it generally has lots of storage space (unlike the phone itself). An advantage of having these files on this host is that they can easily be processed with other tools when debugging.

When using this NFS shared folder we had a problem with IP fragmentation. This problem can be solved using the following parameters when mounting the NFS filesystem:

- wsize: 1024
- rsize: 1024

The phone also requires some configuration. A folder named “ide” should be created in “/tmp” (which is a writable directory), and the following command issued to mount the remote directory on the phone at the specified mount point (/tmp/ide).

```
mount -t nfs -o nolock,hard 192.168.1.1:/export/home/oscar/Motorola  
/debug tmp/ide
```

Figure 5.8: Mount command

The command mount is called specifying the type of the file system to mount as type nfs. The address 192.168.1.1 is the IP address of the host computer followed by the name of the path to the directory which is to be mounted. The second path specifies the mount point on the phone.

Some Linux distributions have a software firewall activated by default. In such cases is necessary to configure the firewall to permit the necessary communication (additionally you will need to have the NFS services mountd, lock, statd, and quotad use fixed ports –

so you know which ports to let through; For directions in configuring these to use fixed port numbers see [42]).

Alternatively because the phone is generally attached to the development host by a USB cable it may be reasonable to simply turn off all the firewall filtering on this interface or at least all the filtering relevant to packets to and from the IP address of the phone.

5.2.2. Adaptation of Minisip

The last step is to adapt Minisip to the phone's architecture and features. There are several issues to consider in this process:

- Providing the basic functions of Minisip
- Develop a transfer feature
- Communication with a SLP server
- Adapt the QT GUI to support both an interface to Minisip and to control the transfer process

Once Minisip has been ported to the phone's ARM platform, it did work as expected. Although the phone and the PC system environments are relatively similar, i.e., both are Linux-based, there will be incompatibility problems, such as properly accessing the media system of the phone.

To develop the transfer feature it is necessary to create a manager which is aware of the phone's session status.

In this thesis we will assume that some other system provides a list of devices and their capabilities (as this is being developed in another set of theses). Thus we will simply offer the user a choice of a set of devices with different capabilities - these will be retrieved from a configuration file.

Finally, the last task was to adapt or implement new QT GUI with both the Minisip and transfer control functions. This subtask will be described in the following section.

5.2.3. GUI for the Motorola Phone

Another step is to develop a graphical user interface for the Motorola phone. The phone's SDK is based on QT 2.3.8, but with some new class implementations specifically designed for this phone.

The easy way to implement this GUI is to use the Eclipse integrated development system and the plug-in to upload and debug the code to the phone, as it was described in chapter four.

The main objective was to develop a simple GUI, even though it might not have all the functionalities of the original GUI. However, there are many features that must be supported in order to test Minisip in a SIP mobile environment. These functions are:

Basic call handling	Basic call functions, such as making a call and hanging-up, in order to interact with other end points
Discovery of devices	Session mobility requires discovery of devices to transfer the RTP stream to.
Configuration dialog	A basic configuration dialog is required to configure SIP features, such as SIP-URI, password, SIPPROxy, and port.
Transfer capabilities	A means of controlling the transfer service is necessary to transfer RTP streams from one device to another.
Debug dialog	Debugging information about call processing is necessary to be aware of the media handling operations.

The GUI should take advantage of the features of the phone's screen, for example using the touch screen, emulated keyboard, or handwriting recognition. Although a QT GUI exists for Minisip, has not been utilized or kept in synchronisation with developments for some time. However, it might be possible to adapt or reuse some functions previously developed. Unfortunately at the time of the writing of this report the implementation of the GUI has not been completed.

6. Analysis

In this chapter we will analyse the RTP method selected. Afterwards, we will study if this method fulfils the requirements proposed in chapter five. However, before doing the analysis we will explain how we are going to evaluate what we have done.

6.1. Evaluation

In this section we will explain how the analysis will be performed and how we are going to evaluate this RTP transfer approach.

In the analysis section, we will discuss the porting and adaptation of Minisip to the Motorola Linux phone and also the RTP redirection using the selected approach. Concerning porting Minisip, an explanation about the tools used and the problems encountered will be given. With regard to the RTP redirection approach, we will check if the behaviour of those endpoints is the expected from the 3PCC RTP approach explained in section 3.4.1. To verify this, a normal transfer will be performed and all the traffic will be captured with a sniffing tool, in this case Ethereal [43]. The scenario will be the same as was detailed in section 5.1.3.

To evaluate the RTP method transfer we will consider the requirements that were proposed in section 5.1.1. Mainly we will study the following requirements

Transfer response time	Several transfers will be performed in order to obtain a mean time between the first INVITE and the last 200 OK.
Media independency	A codec that is not supported by the MN will be used in order to check for media independency.

6.2. Analysis

To analyze the RTP redirection approach, initially it was necessary to port a preliminary version of Minisip to the Motorola Linux phone, as explained in section 5.2.1.

6.2.1. Port and adaptation of Minisip

This port has not been an easy task because when the Minisip project started, it was developed initially on x86 Linux systems. Later, the project was moved to different platforms, such as familiar Linux, ARM, Windows, and recently to Windows Mobile [44].

The process followed to port Minisip to the Motorola Linux phone was explained in section 5.2.1. The tools used were mainly two different toolchains.

We started the process using the tool chain that was proposed by the Minisip project, i.e. the one from Handhelds.org. During the compilation of the libraries we encountered different incompatibilities that were successfully resolved. Finally we obtained the compiled libraries and the binary with a textUI of Minisip. To upload it to the phone we used the NFS shared folder and we modified the following environment variables in the phone.

```
#export LD_LIBRARY_PATH=/tmp/ide:/lib:/usr/lib
#export HOME=/tmp/ide
```

Figure 6.1: Environment variables in the phone

LD_LIBRARY_PATH is an environment variable that tells “ld” where to find libraries if they are not where it expects them. As we have uploaded all the libraries to the NFS shared folder in /tmp/ide we have to update this variable. If the port were included in the flashed file system of the phone, that libraries and executable could be placed where libraries and executables normally reside.

HOME is another environment variable that tells where the user’s home directory is. In our case we have set the home to be the same NFS shared folder, so that the Minisip textUI executable could find the Minisip configuration file.

Once the setup was complete, we tried to execute the binary, but the execution failed, it appeared to be an error. We tried to do remote debugging to detect where the execution error was, but little information was obtained.

Then, we decided to use the toolchain provided by the Motorola EzX SDK. The procedure was the same that with the other toolchain, several other incompatibilities were found and resolved. Finally the libraries and the Minisip textUI binary were built using this toolchain and were uploaded to the phone using the NFS shared folder.

Now we could execute the Minisip textUI binary in the shell of the phone without problems. In order to configure Minisip there is a configuration file, an example is shown in the Appendix [A],

Before executing Minisip it is necessary to configure the wireless interface from the phone. As it is explained in chapter four, there is an SD slot; into this slot we have placed an SDio card with IEEE 802.11b support. The configuration commands are shown in figure 6.2:

```
#iwconfig eth0 essid Default mode Managed
#dhcpcd eth0
dhcpcd: MAC address = 00:e0:00:de:5a:6b
dhcpcd: SetDHCPDefaultRoutes=1
DhcpOptions.len[routersOnSubnet] = 4
dhcpcd: setting default route to \uffff
dhcpcd: your IP address = 130.237.15.221
dhcpcd: orig hostname = (none)
dhcpcd: your hostname = dhcp015221.wireless.kth.se
dhcpcd: orig domainname = (none)
dhcpcd: your domainname = wireless.kth.se
```

Figure 6.2: Phone's wireless configuration

In addition, the Sip Express Router (SER) will be started in a server in order to register this Minisip UA and also act as a proxy for signalling sessions.

```
# ser
Listening on
      udp: 130.237.15.233 [130.237.15.233]:5060
      tcp: 130.237.15.233 [130.237.15.233]:5060
Aliases:
      tcp: ccsmot:5060
      tcp: ccsmot.wireless.kth.se:5060
      udp: ccsmot:5060
      udp: ccsmot.wireless.kth.se:5060
```

Figure 6.3: Ser execution

The configuration file from SER is shown in the Appendix [B]. It is a simple configuration file, every UA can register and there is no need for password.

Finally we will start the Minisip textUI binary execution.

```

# ./minisip_textui

Starting MiniSIP TextUI ... welcome!

Creating TextUI
Minisip: 1
Minisip: 2
Library: file not found
SipIdentity::SipIdentity : created identity id=1
SipIdentity::setSipProxy: autodetect is false; userUri=
1006@130.237.15.233; transport = UDP; proxyAddr=130.237.15.233;
proxyPort=5060
SipIdentity::setProxy: else ...
SipProxy:setProxy(str) : addr = <sip:
130.237.15.233:5060;transport=UDP>
SipIdentity::setProxy: manual sipproxy success ...
Minisip is using IP = 130.237.15.221

To auto-complete, press <tab>. For a list of commands, press <tab>.
...

```

Figure 6.4: Minisip textUI execution

It is possible to check in the SER if the user agent has registered successfully as shown in figure 6.5.

```

# serctl ul show
Dumping all contacts may take long: are you sure you want to
proceed? [Y|N] y
===Domain list===
---Domain---
name : 'location'
size : 512
table: 0xb5ed1ed8
d_ll {
    n      : 2
    first: 0xb5ed3ee0
    last  : 0xb5ed3fe8
}
...Record(0xb5ed3fe8)...
domain: 'location'
aor   : '1006'
~~~Contact(0xb5ed4028)~~~
domain   : 'location'
aor      : '1006'
Contact  : 'sip:1006@130.237.15.221:5060;transport=UDP'
Expires  : 793
q        :
Call-ID  : '744687153@130.237.15.221'
CSeq     : 101
User-Agent: 'Minisip'
received : ''
State    : CS_NEW
Flags    : 0
next     : (nil)
prev     : (nil)
~~~/Contact~~~~
.../Record...

```

Figure 6.5: SER's contact database

Once we have checked that the binary works and that the configuration file, hence the UA has successfully registered with the SER proxy, it is possible to perform a call between to endpoints. Here we will call the user 1005 – who is registered at the Cisco IP Phone.

```
Usage: call <userid>
IDLE$ call 1005
```

Figure 6.6: Minisip textUI call usage

The configuration of the endpoints is the same as shown in the test scenario in section 5.1.3 page number 35.

We have started a SIP communication between the MN and the CN. The following diagram shows the network packets sent over the wireless interface. We have used a tool called SipScenario [45], which uses the capture of a sniffer tool to create a visual scenario.

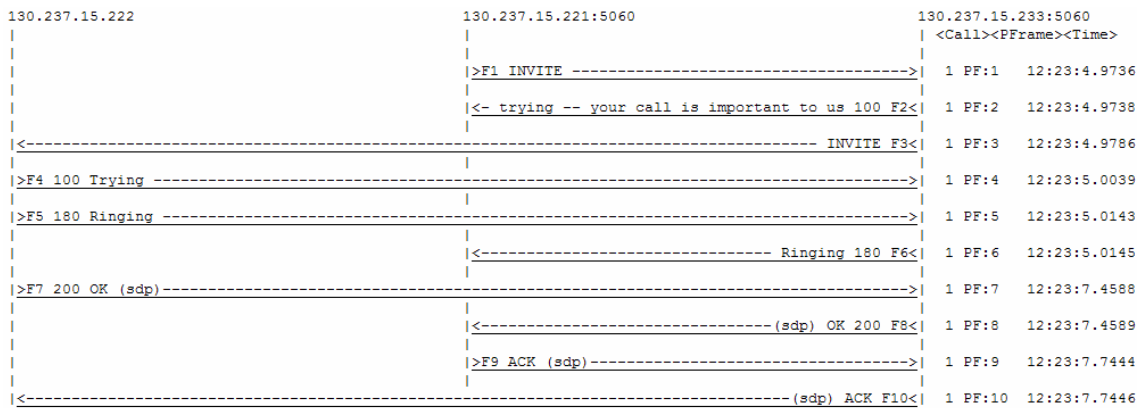


Figure 6.7: Test 1 Simple Call

Figure 6.7 shows the SIP messages exchanged between different endpoints and a SIP proxy. The MN starts the communication sending an INVITE(1) to the CN, but this request is special because it does not have any SDP. The behaviour of the MN is as expected, but when it receives the 200OK(8), it sends an ACK(9) with the SDP information that should have been in the first INVITE. This test corroborates the expected behaviour of the MN with the SIP signalling and thus the behaviour of the CN when receives an INVITE without and SDP. In chapter three, we discussed that some UAs have a strange behaviour when they receive such an INVITE; this test does not such an incompatibility.

Nevertheless, the audio quality of the call was not good. We have checked the statistics of the RTP streams using the Ethereal tool. The results are shown in figure 6.8.

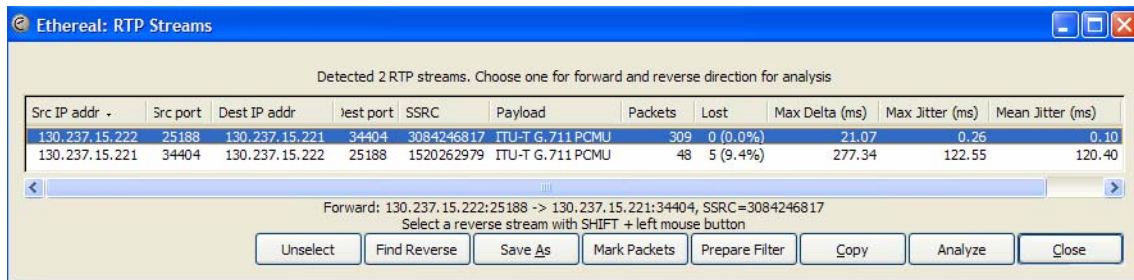


Figure 6.8: Test 1 Statistics

The highlighted stream corresponds to the stream from the CN to the MN. This stream seems correct, there were not any packets lost and the jitter is very small (with a mean of 0.10ms). On the other hand, for the second stream that is sent from the MN to the CN, there were some packets lost and the jitter is very high (around 120 ms). In addition, the number of packets sent are very small (approximately 6 times fewer) in comparison with the packets sent in the other stream.

To understand this behaviour we wrote several small applications that read and write from the audio devices. The results of these applications were satisfactory. Minisip uses a circular buffer to decouple the reading and writing of received packets on their way to the audio device driver and from the audio device driver on their way to becoming packets. It seems that there is not synchronization when accessing to this buffer in this port (which has not shown up in any of the other ports). Although the sound quality is poor the RTP stream is created, and it is possible to perform the transfer using the 3PCC.

To perform this transfer a new command has been added the textUI interface. Some additional classes have been created in order to support two dialogs at the same time. In next section there is another test that shows how the transfer has been achieved.

6.2.2. RTP transfer approach

In this section the 3PCC transfer approach will be analysed. To test this approach we will use the same scenario from section 5.1.3. The entities have exactly the same configuration as in last section.

To invoke this new functionality it is necessary to use the following command, where the first user is the location to where the RTP will be redirected and the second user is the user to transfer.

```
Usage: mobileTransfer <transferTo>-<transferred>
IDLE$ mobileTransfer 1005-1000
```

Figure 6.9: Minisip textUI mobileTransfer usage

The next diagram shows the packets sent to each of the endpoints.

```

130.237.15.222      130.237.15.221:5060      130.237.15.233:5060      130.237.15.247:5061
|<Call><PFrame><Time>
|
|>F1 INVITE ----->
|<-- trying -- your call is important to us 100 F2<
|
|<----- INVITE F3<
|
|>F4 100 Trying ----->
|
|>F5 180 Ringing ----->
|<----- Ringing 180 F6<
|
|>F7 200 OK (sdp)----->
|<----- (sdp) OK 200 F8<
|
|>F9 200 OK (sdp)----->
|<----- (sdp) OK 200 F10<
|
|>F11 INVITE (sdp)----->
|<- trying -- your call is important to us 100 F12<
|
|>F13 INVITE (sdp)----->
|<----- Trying 100 F14<
|
|>F15 180 Ringing ----->
|<----- Ringing 180 F15<
|
|>F17 200 OK (sdp)----->
|<----- Ringing 180 F16<
|
|>F19 200 OK (sdp)----->
|<----- (sdp) OK 200 F18<
|
|>F21 ACK ----->
|<----- (sdp) OK 200 F20<
|
|>F23 ACK (sdp)----->
|<----- (sdp) ACK F24<
|
|>F25 175 21:05:38.2069
|>F27 178 21:05:38.2172
|>F29 204 21:05:38.5170
|>F31 205 21:05:38.5171
|>F33 211 21:05:38.5623
|>F35 212 21:05:38.5624

```

Figure 6.10: Test 2 RTP transfer

The endpoints behave as expected from the 3PCC section. It is noticeable that the Cisco IP Phone sends up to three 200 OKs before it receives the ACK with the SDP. The time to complete the transfer is about 5.59 seconds. This is partially due to the delay between when the phone starts ringing (at this point the phone sends 180 response) and at time the user picks up the phone (at this point the phone sends the 200 OK). Hence this delay is attributed to the end user. In addition there is some additional delay due to the utilization of a SIP proxy that resends every SIP message. This delay is approximately 100µseconds per message.

The screenshot shows a window titled 'Ethereal: RTP Streams' with a table of detected RTP streams. The table has columns for Src IP addr, Src port, Dest IP addr, Dest port, SSRC, Payload, Packets, Lost, Max Delta (ms), Max Jitter (ms), and Mean Jitter (ms). The first row is selected, showing a stream from 130.237.15.247:8000 to 130.237.15.222:31342 with SSRC 3893888624 and ITU-T G.711 PCMU payload. Below the table, there are buttons for 'Unselect', 'Find Reverse', 'Save As', 'Mark Packets', 'Prepare Filter', 'Copy', 'Analyze', and 'Close'. A status bar at the bottom indicates the selected stream: 'Forward: 130.237.15.247:8000 -> 130.237.15.222:31342, SSRC=3893888624'.

Src IP addr	Src port	Dest IP addr	Dest port	SSRC	Payload	Packets	Lost	Max Delta (ms)	Max Jitter (ms)	Mean Jitter (ms)
130.237.15.247	8000	130.237.15.222	31342	3893888624	ITU-T G.711 PCMU	358	0 (0.0%)	484.01	13.51	9.11
130.237.15.221	34548	130.237.15.222	31342	1844008428	ITU-T G.711 PCMU	68	0 (0.0%)	288.79	114.62	107.50
130.237.15.222	31342	130.237.15.247	8000	3084246817	ITU-T G.711 PCMU	387	0 (0.0%)	20.29	0.06	0.02
130.237.15.221	33000	130.237.15.247	8000	1433322778	ITU-T G.711 PCMU	1	0 (0.0%)	0.00	0.00	0.00

Figure 6.11: Test 2 Statistics

Figure 6.11 shows the statistics of the RTP streams in this RTP transfer test. The first thing noticeable is that there are four streams instead of the expected two from the CN and Local endpoints. These additional streams should not appear. The MN still assumes that these packets belong to it. The quality of the call is good despite these extra streams. The first and third streams representing the traffic from the LOCAL to the CN endpoint and from the CN to the LOCAL endpoint have almost the same number of packets, there are no lost packets, and the jitter is quite low.

These additional streams should be eliminated by changing the source code of Minisip. Based upon the low quality of the last call, there is any kind of bad implementation in the Minisip running on the Motorola Linux phone that results in the poor quality just as experienced in test 1.

6.3. Study

The previous section has shown the 3PCC transfer using Minisip in a mobile scenario with a Motorola Linux phone. In this section, we will discuss if this approach fulfils the requirements previously stated. To study these requirements two tests will be performed as explained in section 6.1.

6.3.1. Transfer Response Time

We begin by estimating the mean transfer time. This transfer time begins with the first INVITE sent by the MN and ends with the last ACK received by the LOCAL endpoint. As explained before there is some extra delay due to the end user and the utilization of a SIP proxy in the scenario.

In order to estimate the mean transfer time, ten transfer tests have been completed and recorded. This test is the same as that the test described in section 6.2.2.

Table 6.1

First INVITE (sec)	Last ACK (sec)	Difference (sec)
22.278	27.36	5.082
6.32	11.34	5.02
15.316	20.221	4.905
11.427	16.441	5.014
9.373	14.418	5.045
16.771	21.592	4.821
12.019	16.679	4.66
10.023	15.152	5.129
13.654	18.632	4.978
11.629	16.543	4.914




Finally the mean transfer time is estimated using the following equation.

$$MeanTransferTime = \frac{\sum TimeDifferences}{NumberOfTests} = 4,9568 \text{ seconds}$$

The mean transfer time is small and acceptable although it depends directly upon the endpoints of the transfer. This delay could be reduced if the UA implemented an auto response feature (sometimes called automatic answer). This feature could be used in streaming media servers, for example when you place a call on hold and should be redirected to a Asterisk [46] server which plays some audio to let the other end user know that the call has been placed on hold.

6.3.2. Media Independency

In this section we will check the features of 3PCC and at the same time verify that the requirement of our transfer approach, specifically media independency, works in our scenario. For this test, different CODEC(s) have been used in the endpoints.

 MN	IP address: public obtained by DHCP
	Port: 5060
	UA: Minisip for ARM
	SIP: 1006@130.237.15.221 CODEC G.711
 SIP proxy	IP address: 130.237.15.233
	Port: 5060
	Version: 0.94
	Mode: Proxy + Register
 LOCAL	IP address: 130.237.15.247
	Port: 5060
	UA: Xlite 3.0
	SIP: 1005@130.237.15.210 CODEC GSM

The CODEC GSM is the CODEC used in the GSM communications; it has a bit rate of 13kbps instead of the 64kbps of G.711; although the sampling rate is the same, i.e. 8 KHz.

Next, as in the other tests, we will use the tool SipScenario to create a diagram showing the SIP messages.

```

130.237.15.221:5060                               130.237.15.210:1618                               130.237.15.233:5060
|<Call><PFrame><Time>
|>F1 INVITE ----->| 1 PF:1 20:49:24.3510
|<----- trying -- your call is important to us 100 F2<| 1 PF:2 20:49:24.3512
|<----- INVITE F3<| 1 PF:3 20:49:24.3513
|>F4 180 Ringing ----->| 1 PF:4 20:49:24.5379
|<----- Ringing 180 F5<| 1 PF:5 20:49:24.5380
|>F6 200 OK (sdp)----->| 1 PF:6 20:49:25.4419
|<----- (sdp) OK 200 F7<| 1 PF:7 20:49:25.4420
|>F8 BYE ----->| 1 PF:8 20:49:25.6642
|<----- BYE F9<| 1 PF:9 20:49:25.6644
|>F10 481 Call/Transaction Does Not Exist ----->| 1 PF:10 20:49:25.7784
|<----- Call/Transaction Does Not Exist 481 F11<| 1 PF:11 20:49:25.7785
|>F12 400 Bad Request ----->| 1 PF:12 20:49:25.7794

```

Figure 6.12: Media independency Test

Figure 6.12 shows the behaviour of the MN when using a unsupported CODEC. The SIP communication starts as a normal transfer, but when the MN receives the 200 OK(7) with the SDP information about the CODEC used, it has an strange behaviour. As configured our Minisip did not recognize this CODEC and sends a BYE(8) message. The transfer can not be accomplished because we are not using a compatible CODEC. Finally there are some messages sent by the LOCAL device about a Call or Transaction that does not exist. In the BYE(8) message sent by the MN there is an erroneous header.

Hence the transfer was not accomplished; Minisip does support several other CODEC(s) including GSM, via plug-ins. However, even though Minisip (as configured) does not support the CODEC of the transfer, it should have worked because the MN node is simply acting as a central point rather than as an end point.

The problem is basically that Minisip only supports one dialog per command. If you are on a call you can not make another request to another device, because still is not supported. Minisip was developed to check every response and not to resend it as a central point. Hence the problem about unsupported CODEC, if it is not supported, Minisip aborts the communication

7. Conclusion

In this thesis project we have ported a preliminary version of Minisip to a Motorola Linux phone based on an ARM platform. We have encountered several problems due to the lack of experience of porting such as large project to a handheld device. Moreover there were several possible alternatives to perform the porting and we had to decide what was most suitable given to our porting experience.

Next we decided what kind of RTP transfer approach was best for our own purposes. Several methods have been studied, but only one has been selected and implemented. 3PCC was selected instead of the REFER method or the RTP multicasting method because it better fulfilled the requirements that were stated when the thesis project started. Although 3PCC has not fulfilled all the requirements, we believe that it is a good choice.

Finally a modification of the source code of Minisip was made. In this modification, a transfer approach using 3PCC was developed. We have studied how Minisip works and we have developed a simple application that performs this transfer. We encountered difficulties when understanding the source code because of our limited experience in C++.

Along this thesis project, we have acquired experience about how to start a project and perform it step by step from the initial idea to the final report. The idea seemed very ambiguous at the beginning, but once we move forward in our thesis project it began to take shape and became more concrete. The resulting transfer method has shown that it can function in the real world and meet the basic requirements.

Although all the objectives have been accomplished, if I had to repeat the project I would have made a better project plan.

8. Future Work

Several tasks can be suggested for future work. Due to limitations of time they were not possible to complete as part of this thesis project. Firstly the RTP transfer approach developed using the source code of Minisip should be modified. We have used the CALL service and we have adapted this service to achieve our goals. In the future a new TRANSFER service should be independently developed. It is necessary to use a different state machine than the used in the CALL service because the requirements of this new service are different than the requirements of the CALL service.

As explained in section 6.3.2 Minisip only supports only dialog per command. This limitation does not meet the media independency requirement. The new TRANSFER service should support more than one dialog per command. In addition, it would be interesting to enable a video CODEC to send video to and perhaps even from the phone.

When we ported Minisip to the Motorola Linux phone, we encountered several difficulties and overcame all but one of them: the audio quality of the call is very poor. It would be interesting to improve audio quality.

Finally the creation of a GUI using the SDK provided by Motorola is another important task, as this would make the system much easier to use by the typical user. We started to create such a GUI, but due to time limitations the other tasks concerning redirecting the RTP streams and testing this had greater priority. This GUI could use the features of the Motorola Linux phone to execute Minisip using the touchscreen and handwriting recognition.

References

- [1] Wikipedia. IP Multimedia Subsystem. Last access February 2007.
http://en.wikipedia.org/wiki/IP_Multimedia_Subsystem
- [2] IMS Converged Services Gateway. Last access February 2007.
[http://www.sipcenter.com/sip.nsf/html/WEBB6BWJL4/\\$FILE/IMS_ConvergedSvcGateway.pdf](http://www.sipcenter.com/sip.nsf/html/WEBB6BWJL4/$FILE/IMS_ConvergedSvcGateway.pdf)
- [3] OSDL Mobile Linux Initiative. Press release, October 28, 2006.
http://old.linux-foundation.org/newsroom/press_releases/2006/2006_nov_28_beaverton.html
- [4] Datang Mobile. Last access February 2007.
<http://www.datangmobile.cn/en/AboutUs/Introduction.htm>
- [5] Symbian OS. Last access February 2007.
<http://www.symbian.com/>
- [6] Motorola. Last access February 2007.
<http://www.motorola.com>
- [7] LinuxDevices. Last access February 2007.
<http://linuxdevices.com>
- [8] Minisip UA. Last access February 2007.
<http://www.minisip.org/>
- [9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler. RFC 3261. SIP: Session Initiation Protocol. June 2002.
<http://www.ietf.org/rfc/rfc3261.txt>
- [10] Generation Partnership Project. Last access February 2007.
<http://www.3gpp.org/>
- [11] IP Telephony CookBook. Last access February 2007
<http://www.terena.nl/activities/iptel/contents1.html>
- [12] HTTP Protocol. Last access February 2007.
<http://www.w3.org/Protocols/>
- [13] David H. Crocker. RFC 822. Standard for the format of ARPA Internet text messages. August 13, 1982.
<http://www.ietf.org/rfc/rfc0822.txt>
- [14] Alisa Devlic, Context-Addressed Communication Dispatch, Royal Institute of Technology, Department of Communication Systems, 2006.
<http://web.it.kth.se/~devlic/thesis.pdf>
- [15] Sergi Laencina, Model driven context awareness, Masters Thesis, Royal Institute of Technology, Department of Communication Systems, February 2007.
ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/070130-Sergi_Laencina_Verdaguer-with-cover.pdf
- [16] Unicode Standard. Last access February 2007.
<http://unicode.org/>
- [17] Internet Engineering Tasking Force. Last access February 2007.
<http://www.ietf.org/>

- [18] R. Shacham, H. Schulzrinne, S. Thakolsri, W. Kellerer. November 2006. Draft-shacham-sipping-session-mobility-03.
<http://www.ietf.org/internet-drafts/draft-shacham-sipping-session-mobility-03.txt>
- [19] E. Guttman, C. Perkins, J. Veizades, M. Day. RFC 2608. Service Location Protocol. June 1999.
<http://www.rfc-editor.org/rfc/rfc2608.txt>
- [20] Cecile Ayrault, Service discovery for personal area networks, Masters Thesis, Royal Institute of Technology, Department of Communication Systems, August 2004.
ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/040826-Cecile_Ayrault-with-covers.pdf
- [21] J. Rosenberg, J. Peterson, H. Schulzrinne, G. Camarillo. RFC 3725. Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP). April 2004.
<http://www.ietf.org/rfc/rfc3725.txt>
- [22] R. Sparks. RFC 3515. The Session Initiation Protocol (SIP) Refer Method. April 2003.
<http://www.ietf.org/rfc/rfc3515.txt>
- [23] R. Sparks. RFC 3892. The Session Initiation Protocol (SIP) Referred-By Mechanism. September 2004.
<http://www.ietf.org/rfc/rfc3892.txt>
- [24] RTP Proxy. Portaone RTP Proxy. Last access February 2007.
<http://www.voip-info.org/wiki-Portaone+rtpproxy>
- [25] STUN. Last access February 2007.
<http://www.voip-info.org/wiki-STUN>
- [26] UPNP. Universal Plug and Play. Last access February 2007.
http://en.wikipedia.org/wiki/Universal_Plug_and_Play
- [27] Adrian Georgescu. TURN. Best practices for SIP NAT traversal. Last Access February 2007.
<http://mediaproxy.ag-projects.com/NATtraversal-BestPractices.pdf>
- [28] Gustav Söderström, Virtual networks in the cellular domain , Masters Thesis, Royal Institute of Technology, Department of Communication Systems, February 2003.
ftp://ftp.it.kth.se/Reports/DEGREE-PROJECT-REPORTS/030211-Gustav_Soderstrom.pdf
- [29] Soft Handover. UMTS handover. Last access February 2007.
<http://www.umtsworld.com/technology/handover.htm>
- [30] Motorola E680i specifications. Last access February 2007.
<http://developer.motorola.com/products/handsets/e680i/>
- [31] ARM architecture. Processor core overview. Last access February 2007.
<http://www.arm.com/products/CPUs/index.html>
- [32] Montavista. Motorola phones run Montavista. Last access February 2007.
<http://www.mvista.com/dswp/stories/motorola.html>
- [33] QT. Trolltech. Last access February 2007.
<http://www.trolltech.com/products/qt>
- [34] KDE project. Last access February 2007.
<http://www.kde.org/>
- [35] Opera Web Browser. Last access February 2007.
<http://www.opera.com/>

- [36] Eclipse IDE. Last access February 2007.
<http://www.eclipse.org/>
- [37] GCC. Last access February 2007.
<http://gcc.gnu.org/>
- [38] Binutils. Last access February 2007.
<http://www.gnu.org/software/binutils/>
- [39] Uclibc. Last access February 2007.
<http://uclibc.org/>
- [40] Handhelds Toolchain. Last access February 2007.
<http://www.handhelds.org/geeklog/index.php>
- [41] OpenSSL project. Last access February 2007.
<http://www.openssl.org/>
- [42] Configuring NFS under Linux for Firewall Control. Last access February 2007.
http://www.lowth.com/LinWiz/1.09/notes/nfs_help.php
- [43] Etherreal. Last access February 2007.
<http://www.ethereal.com/>
- [44] Cesc SantaSusana. Port of Minisip to Windows Mobile 2003
- [45] SipScenario. Last access February 2007.
<http://www.iptel.org/~sipsc/>
- [46] Asterisk. Last access February 2007.
<http://www.asterisk.org/>

Appendix A

An example of Minisip configuration file – as used on the Linux phone.

```
<version>
  3
</version>
<network_interface>
  eth0
</network_interface>
<account>
  <account_name>
    My account
  </account_name>
  <sip_uri>
    1006@130.237.15.233
  </sip_uri>
  <proxy_addr>
    130.237.15.233
  </proxy_addr>
  <register>
    yes
  </register>
  <proxy_port>
    5060
  </proxy_port>
  <proxy_username>
    1006
  </proxy_username>
  <proxy_password>
    1006
  </proxy_password>
  <pstn_account>
    no
  </pstn_account>
  <default_account>
    yes
  </default_account>
  <secured>
    no
  </secured>
  <ka_type>
    psk
  </ka_type>
  <psk>
    Unspecified PSK
  </psk>
  <certificate>
  </certificate>
  <private_key>
  </private_key>
  <ca_file>
  </ca_file>
  <dh_enabled>
    no
  </dh_enabled>
  <psk_enabled>
    no
  </psk_enabled>
```

```
<check_cert>
  yes
</check_cert>
</account>
<tcp_server>
  yes
</tcp_server>
<tls_server>
  no
</tls_server>
<local_udp_port>
  5060
</local_udp_port>
<local_tcp_port>
  5060
</local_tcp_port>
<local_tls_port>
  5061
</local_tls_port>
<sound_device>
  /dev/dsp16
</sound_device>
<mixer_type>
  spatial
</mixer_type>
<codec>
  G.711
</codec>
<phonebook>
  file:///tmp/ide/.minisip.addr
</phonebook>
```

Appendix B

SER configuration file used for the tests in this thesis.

```
#
debug=3
fork=yes
log_stderr=no

listen=130.237.15.233          # INSERT YOUR IP ADDRESS HERE
port=5060
children=4

dns=no
rev_dns=no
fifo="/tmp/ser_fifo"

loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"

modparam("usrloc", "db_mode", 0)
modparam("rr", "enable_full_lr", 1)

route {

    # -----
    # Sanity Check Section
    # -----
    if (!mf_process_maxfwd_header("10")) {
        sl_send_reply("483", "Too Many Hops");
        break;
    };

    if (msg:len > max_len) {
        sl_send_reply("513", "Message Overflow");
        break;
    };

    # -----
    # Record Route Section
    # -----
    if (method!="REGISTER") {
        record_route();
    };

    # -----
    # Loose Route Section
    # -----
    if (loose_route()) {
        route(1);
        break;
    };

    # -----
```

```

# Call Type Processing Section
# -----
if (uri!=myself) {
    route(1);
    break;
};

if (method=="ACK") {
    route(1);
    break;
} else if (method=="REGISTER") {
    route(2);
    break;
};

lookup("aliases");
if (uri!=myself) {
    route(1);
    break;
};

if (!lookup("location")) {
    sl_send_reply("404", "User Not Found");
    break;
};

route(1);
}

route[1] {
# -----
# Default Message Handler
# -----
if (!t_relay()) {
    sl_reply_error();
};
}

route[2] {
# -----
# REGISTER Message Handler
# -----
if (!save("location")) {
    sl_reply_error();
};
}

```

