INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Academic Instant Messaging System

Deploying Instant Messaging Over an Existing Session Initiation Protocol and LDAP Service Infrastructure Using the Message Session Relay Protocol

João André Pereira Antunes

Dissertação para obtenção do Grau de Mestre em

Engenharia de Redes de Comunicação

Júri

Presidente: Prof. Doutor Rui Jorge Morais Tomaz Valadas
Orientador: Prof. Doutor Fernando Mira da Silva
Vogais: Prof. Doutor Paulo Jorge Pires Ferreira

Outubro 2009

Acknowledgements

I would like to thank the following people for helping me in one way or another, directly or not, during the timeframe of this thesis:
Prof. Fernando Mira da Silva – making the theme of this work possible in the first place, for directions, valuable tips, review efforts, and pep talks.
José Santos – for all of the availability to concede me the requested resources, but mostly for the talks that ultimately provided me some very helpful insights and ideas about the current service IT infrastructure and deployment of the IM service.
André Brioso – for the helpful talk that gave me a valuable insight regarding the LDAP service infrastructure;
All of the members of the open source Sip-Communicator community that helped me in one way or another.
Special thanks to Emil Ivov – for all of the support and guidance regarding the feature enhancements that I developed for Sip-Communicator; for contributing to the notion that I have about open source;
To Anthony Schmitt for his work and development of the first file transfer GUI for Sip-Communicator and his availability to give me the needed pointers on how to use it;
To Lubomir Marinov and Yana Stamcheva for the new file transfer GUI and generic SLICK file transfer test;
To all the people that make the Google summer of code possible. It really got me into the open source philosophy. In the end, for the visionary Google company, that makes these kind of programs possible, and for showing to the world scenarios where everybody can win.
To all of the people that made and make the excellent work of the NlNet foundation a reality. Special thanks for Valer Mischenko, for being the nice person that he is but mostly for the decisive contribution that made my participation on TERENA's 2009 network conference (TNC 2009) possible.
For TERENA and Cisco Inc. for listening to Valer Mischenko and for being flexible enough to provide me a free student pass to TNC 2009 in such a short notice;
For all of my friends, that helped me indirectly by cheering me up and being friends, or directly for contributing to the thesis. More specifically I would like to thank:
Ricardo de Gomes Rodrigues – for his inspiring example of dedication to his work. For providing me essential support that allowed me to participate in TNC 2009;
Pedro Manuel Vieira Gomes – for contributing directly to this work for the short but effective pep-talk at the a moment of great need;
Gonçalo Manuel Mendes Duarte Gomes Ivo – for punctual pragmatic pointers and advices on thesis related subjects; for the conversations we had during the course of the semester on which we shared stories about our work in the thesis;
Andreia Mateus Martins – for helping me to confirm to myself what I'm made of, and how I act in the face of despair. For all the friends that should have been listed here but that I forgot, and already forgave me for it :).
And last but not at all least, my family, my parents, Silvia Antunes and Francisco Antunes, not only for the genes but mostly for the values and support. Specially to my mother for caring so much and for never have stopped nagging me :). My sister, Joana Antunes for listening to me and for promptly providing me all the help in times of need, despite of all the misdeeds we might have done to each other in our childhoods :).

Abstract

In the last years, proliferation of the Instant Messaging (IM) systems occurred. This growth is propelled by the widespread usage of Internet and people's innate needs to communicate virtually anywhere and at anytime.

Instituto Superior Técnico (IST) is a leading Engineer faculty in Portugal, with a total population of 10,000 users. Network access at IST is available almost anywhere, both in wireless and wired modes. In this and similar academic environments, IM is a powerful collaborating tool, providing remote and instantaneous communication between users.

This work consists on planning, developing and deploying an IM system solution for IST's community, adapted to its current service infrastructure. IM addresses and accounts are associated with the central Identity Provider (IdP) service. This association enables the fast deployment of the optional IM service and its widespread use. The IM system is supported by the current Session Initiation Protocol (SIP) infrastructures using the SIP Instant Messaging and Presence Leveraging Extensions (SIMPLE) standards.

This IM service is based on the development of a Java generic purpose open source Message Session Relay Protocol (MSRP) peer library, and its integration with a popular Java multiprotocol IM open source client, Sip-Communicator. This work results in a contribution to the open source and IM worlds by means of software development, and deployment of an IM beta-stage system. The carried implementations as well as evaluations and results are presented. Tests were devised and executed to draw conclusions about the correct provisioning of resources for future versions of the IM system.

Keywords: Instant Messaging, SIP, SIMPLE, MSRP, IdP, LDAP

Resumo

Ultimamente, a proliferação de sistemas de mensagens instantâneas (MI) ocorreu. Esta proliferação pode-se explicar pelo enorme crescimento da utilização da Internet. As necessidades de comunicar, por parte das pessoas, têm alimentado o desenvolvimento destes sistemas.

O Instituto Superior Técnico (IST) conta com uma população total de cerca de 10,000 utilizadores. O acesso à Internet no IST é praticamente ubíquo no seu campus. Neste e em semelhantes campus, os serviços de MI possibilitam uma comunicação remota e instantânea entre utilizadores, representando uma poderosa ferramenta de colaboração.

Este trabalho consiste no planeamento e desenvolvimento de uma solução de MI que sirva a comunidade do IST e que esteja interligada com a sua actual infra-estrutura de serviços. Os endereços de MI ficam assim interligados com o serviço de identificação centralizado já existente. Esta interligação é prevista que promova a utilização generalizada do serviço, que é opcional. O sistema de MI montado é suportado pela já existente infra-estrutura de serviços de Session Initiation Protocol (SIP) através dos standards SIP Instant Messaging and Presence Leveraging Extensions (SIMPLE).

O precoce sistema de MI implementado, assenta no desenvolvimento de uma biblioteca Java do Message Session Relay Protocol (MSRP) e integração com o Sip-Communicator, um popular cliente Java de MI. Este trabalho, resulta numa contribuição para o mundo do open source. O trabalho desenvolvido, tal como a sua avaliação e resultados são apresentados. Foram elaboradas e executadas metodologias de teste, de forma a tirarem-se conclusões acerca do provisionamento adequado de recursos para uma futura versão do sistema.

Palavras chave: Sistema de mensagens instantâneas, SIMPLE, SIP, MSRP, LDAP

Table of contents

List of Tables

List of Figures

List of Acronyms

AIM America Online Instant Messenger

API Application Programming Interface

BBS Bulletin Board System

BOSH Bidirectional-streams Over Synchronous HTTP

CAP Common Alerting Protocol

CAPTCHA Completely Automated Public Turing test to tell Computers and Humans Apart

CMU Carnegie Mellon University

CPIM Common Profile for Instant Messaging

DCC Direct client-to-client protocol

DHT Distributed Hash Table

FAQ Frequently Asked Questions

FSF Free Software Foundation

GUI Graphical User Interface

IANA Internet Assigned Numbers Authority

ICQ I Seek You IM service

IdP Identity Provider

IETF Internet Engineering Task Force

IM Instant Messaging

IP Internet Protocol

IRC Internet Relay Chat

IT Information Technologies

JID Jabber Identifier

JSF Jabber Software Foundation

LDAP Lightweight Directory Access Protocol

MIT Massachusetts Institute of Technology university

MSN Microsoft Network

MSRP Message Session Relay Protocol

NAT Network Address Translation

PIDF Presence Information Data Format

PSTN Public Switched Telephone Network

PSYC Protocol for Synchronous Conferencing

RADIUS Remote Authentication Dial In User Service protocol

RFB Remote Framebuffer protocol

RFC Request For Comments

RLS Resource List Servers

SC Sip-Communicator

SDK Software Development Kit

SDP Session Description Protocol

SER Sip Express Router

SIM-IM Simple Instant Messenger

SIMPLE SIP Instant Messaging and Presence Leveraging Extensions

SIP Session Initiation Protocol

SMTP Simple Mail Transfer Protocol

SOAP Simple Object Access Protocol

SVN Subversion

TCP Transmission Control Protocol

TLS Transport Layer Security

UNI Uniform Network Identifications

UNL Uniform Network Locations

UPnP Universal Plug and Play protocols

URI Uniform Resource Identifier

VNC Virtual Network Computing system

VoIP Voice over IP

WG Working Group

XEP XMPP Extension Protocols

XHTML eXtensible Hypertext Markup Language

XML eXtensible Markup Language

XMPP Extensible Messaging and Presence Protocol

XSF XMPP Software Foundation

Chapter 1

Introduction

In the last years, there was a proliferation of the Instant Messaging (IM) systems. This growth is propelled by the widespread usage of Internet and people innate needs to communicate virtually anywhere and at anytime. IM technologies long ago left the ambit of providing text only communication. Currently, it is common to find that most IM technologies offer video and voice calls; folder sharing; whiteboard shared sessions; desktop sharing; picture sharing; voice clips communication; etc. This new kind of IM services offer users new, useful and joyful possibilities to collaborate and communicate over the Internet. IM is one of the technologies that more contributes to enhance social usefulness of the Internet.
Along with the joyful aspect of IM, it can be also an invaluable work tool. It can be used to enable collaborative problem solving to people that are physically distant. It can also provide ways for people to attend and participate in events that are physically distant from them.

This work consists on planning, developing and deploying a prototype IM system solution for IST's academic community, adapted to its current service infrastructure, in which all network users will automatically have an IM address associated with the central Identity Provider (IdP) service. While usage of the IM service will be obviously optional, this scheme enables the fast deployment of the service and its widespread use.
Instituto Superior Técnico (IST) is a leading Engineer faculty in Portugal, with a total population of around 10,000 users, including students, teachers, researchers and support staff. Network access at IST is available almost anywhere, both in wireless and wired modes. In this and similar academic environments, IM is a powerful collaborating tool, providing remote and instantaneous access between all academic users.

A wide-spread use of an IM system on a student community like this can prove to be a very important tool. It can aid students by facilitating collaboration. It can provide new and more efficient ways of problem solving, especially in an IT engineering community of students like the one IST has. Given the communication features that modern IM systems have, like voice and video calls, desktop sharing and whiteboard sharing, students can work together in group projects without the need to be in the same physical space. Professors assistance can be facilitated as they can be more easily contacted and they have ways to provide help remotely.

With thorough testing and architecture details disclosure, a successful implementation and deployment of the IM system can be used and applied to other similar communities. As part of this effort, the contribution given by this work will hopefully add a successful open source

alternative to the use of Skype[1] and other popular commercial IM applications and protocols, therefore contributing to the IM and open source worlds.

1.1 Contributions

Contributions to the open source community were made with this work. An open source Message Session Relay Protocol (MSRP) Peer library in Java [38] was developed entirely from the ground up. Also, the open source application Sip-Communicator [47] was enhanced with file transfer capabilities over the SIP/SIMPLE protocol.

The IM system solution and idea was promoted in the 2009 edition of the TERENA[2] Networking Conference. Promotion was done via a Poster that I presented during the conference.[3]

1.2 Structure of this thesis

Chapter 2 of this document reviews the currently available IM clients and IM protocols that may represent a solution for the IM system. Chapter 3 discusses the motivations for the selected and therein presented solution, and also gives implementation details of the developed work. Following, chapter 4 evaluates the implementation work and the deployed IM system, by defining tests in several scopes, and presenting the gathered results, along with relevant discussions regarding the ambit of the tests and attained results. Picking up with the conclusions drawn from chapter 4, this dissertation ends with chapter 5, where a detailed list of conclusions as well as future work is presented.

[1] The third Open Source community's top priority to the date of writing of this paper, is the replacement of Skype by an Open Source alternative.
[2] TERENA is the a Trans-European Research and Education Networking Association. More information about this organization can in: http://www.terena.org/
[3] The poster, along with the submitted abstract can be obtained in the following website: http://tnc2009.terena.org/schedule/posters/index.php?poster_id=18

Chapter 2

Related work / State-of-the art

There are already a considerable number of Instant Messaging (IM) technologies developed and available to use for this work. In this chapter, the details of the research carried with the intent to assess the current technologies and viable possibilities for the IM system is presented. This chapter represents a structured and logical research of technology specifications and their implementation states. It aims to provide the essential information for defining the choices of the IM protocol for the IM system and client to be promoted, as well as define the related work to be carried.

Thus, the two main sections of this chapter: 2.1 where IM protocols are reviewed, and 2.2 where IM clients are discussed. At the end of the chapter, the discussion of available alternatives, is reduced to two for the IM protocol and three for the IM client. The selected solution is described with detail in chapter 3, that relies heavily on this chapter.

2.1 IM protocols

This section presents an overview of the performed research on the various available IM protocols, which lead to the selection of the IM system. To search the most appropriate IM protocol, I start by reducing the list of all the numerous available IM protocols in subsection 2.1.1, creating a short list of possible solutions. Next, a more or less detailed view of each protocol in this short list, is given in subsection 2.1.2. In this subsection, a ranking of the most popular IM protocols is presented. This ranking, while not unique, contributes to make clearer which protocols are more valuable from an interoperability point of view. Interoperability becomes an even more valuable feature given that most of the potential IM users for this system probably already have other IM accounts.

2.1.1 Pre-requisites

For the IM system, due to the academic nature of the developed work, the only IM protocols that were considered to be part of the solution are the ones that have open standards.

Therefore, the subset found to match this criteria is listed on table 2.1.

Name of IM protocol	Protocol's Official Website (or protocol's working group official website)
CSpace	http://cspace.in/
Gale	http://www.gale.org/
IRC[4]	http://www.irc.org/
PSYC[5]	http://www.psyc.eu/
Retroshare	http://retroshare.sourceforge.net/
SIP/SIMPLE[6]	http://www.ietf.org/html.charters/sip-charter.html

[4] **I**nternet **R**elay **C**hat (IRC) [1]
[5] **P**rotocol for **SY**nchronous **C**onferencing (PSYC) [2]

XMPP[7]	http://www.xmpp.org/
Zephyr Notification Service	Not Available / Not Found

Table 2.1. List of IM protocols considered for the solution.

These protocols listed in table 2.1 are subject of analysis in the next subsection.

2.1.2 Overview of the considered IM protocols

The following sub-sections contain a discussion of the protocols considered for the solution. They have also a brief commentary on a possible contribution of the protocol for the solution.

CSpace

CSpace[8] is more than a simple IM protocol, it's designed with the intent to provide "a platform for secure, decentralized, user-to-user communication over the internet". It aims to be more like an API in which applications can rely on to provide a secure and NAT/friendly communication between users, than just a simple IM protocol.

The users of the CSpace protocol are required to generate a 2048-bit RSA[9] key pair. This key pair is used to uniquely identify a user of the CSpace protocol. When a user wants to communicate with another, he must have the user's RSA key he wishes to communicate with. After having the key, the steps that must be performed are:

- The user connects himself to the CSpace's decentralized network[10];
- The client uses the network to locate and establish a secure Transmission Control Protocol (TCP) connection (using the Transport Layer Security (TLS) protocol) directly with the user. It is also possible for the client to connect with an intermediate proxy if the destination user is behind a Network Address Translation system (NAT) or behind a firewall.

Although this protocol can be proven to be very useful in some environments, it's not optimal for our IM system due to several reasons. A big, structural one, is the fact that the protocol establishes only secure TCP connections. Secure TCP connections, due to performance issues, are not the optimal ones to transmit multimedia data, like for instance, Voice over IP (VoIP) data. Voice calls, and even video calls, are nowadays a common feature amongst IM systems. As the users expectations tend to grow, not to decrease, providing an IM system without voice and/or video support would probably mean that it wouldn't be used by most of its potential users.

Another reason is that there aren't many client applications that currently support this protocol. Also, if used, the system would be dependent on the CSpace's network.

---

[6] **S**ession **I**nitiation **P**rotocol for **I**nstant **M**essaging and **P**resence **L**everaging **E**xtensions (SIP/SIMPLE) [3]

[7] **E**xtensible **M**essaging and **P**resence **P**rotocol (XMPP) [4]

[8] Tachyon Techonologies, "CSpace", http://cspace.in/

[9] RSA – an algorithm for public-key cryptography, [5].

[10] Decentralized in the sense that it uses a distributed hash table (DHT) [6] based on the Kademlia[7] system to locate its users and to enable communication between them.

Gale

Gale[11], is an IM protocol that claims to be secure and scalable. Unfortunately, the details of the architecture of this protocol are not available on the official webpage. However, the major features that the protocol offers are provided in the webpage. Therefore, I made a first analysis to assert if the protocol was a good candidate, based on its announced features. Based on the protocol's webpage, the protocol's features are:

1. Use of "strong" cryptography technology to ensure privacy of communication and user authentication.
2. Possibility of communication between two users and communication from N to N users.
3. Scalability – as stated on the webpage, is attained by two main features:

3.1 It's network relies on a series of "loosely-connected […] servers which locate each other via DNS only when they need to talk to each other."

3.2 It supports multicast by creating self-healing spanning trees of interconnected servers that, as well as the clients, can detect and route around failures on communication. It also claims that doesn't have the same problems of performance that are common in networks such as IRC and Usenet.

Unfortunately, along with the features, I also noticed that the protocol seems to be abandoned. Its latest release is dated from around 2003. It also doesn't support voice or video communication and their clients also suffer from the same state of abandonment that the protocol suffers. These facts justify abandoning this protocol as a possible solution. Therefore, I did no more research about it (e.g. through the inspection of the source code).

Internet Relay Chat (IRC)

The IRC protocol [1] has been developed since 1989, when it was implemented as a mean for users on a bulletin board system (BBS) to chat amongst themselves. The protocol is designed to be essentially for text based communication. Due to its release date and pioneer status, the IRC features and functionalities inspired several features and functionalities of most IM protocols. As a consequence, some concepts like chat rooms, nicknames, bots, amongst others, were adopted by more recent IM protocols.

A very brief overview of how the protocol works is given in the remainder of this section. It's not within the scope of this document to explain all protocol details. It only provides enough information for the reader to have a general picture of how the protocol works, and the kind of communication/user interaction it offers.

The protocol is based on a client-server model. Each client that wishes to communicate connects to a server of an IRC network. The network might be composed by one or more

[11] Gale, http://www.gale.org/

servers. In order for a user to connect to an IRC network, he must choose a nickname to use. This nickname uniquely identifies the user during the established session. Some IRC networks allow the user to register the nickname[12].

The server is responsible for delivering/multiplexing the messages and also other functions (e.g. the aforementioned nickname registration). An IRC network can be composed of just one or several servers interconnected with each other (see fig. 2.1 for an example of such a network)

The clients can chat with several other users simultaneously through the use of a channel[13]. For description purposes, a channel can be compared to a special physical chat room. In this room, there are other users, and the text written by each user is made available to all users.

Each channel has a name that uniquely identifies it within the IRC network, and also a list of its current users that is available to every user in the same room and can also be consulted in some special conditions by users outside of the channel.

Channels have the option to have access to them restricted by:

- The use of a passphrase;
- Requiring a user to be invited by another currently active user on the channel;

Unlike suggested by the analogy with a physical room, users can be in multiple channels simultaneously. Users can also communicate privately between them, provided that they know the nickname of the destination user.

In order to maintain the orderliness, each channel can have a set of channel operators. These channel operators are users that have special privileges in that channel (e.g. among other things, channel operators can kick and ban a user out of the channel). The network itself has a set of operators. These operators, called IRC operators, have special privileges within the network, and they can kick and ban users from the entire IRC network or from specified servers.

There exists also another status of users within channels, called voices. The users that have such a status within a channel can speak even if a channel has a special moderated mode[14] set. By giving the operators ways of giving the right to “speak” to the users of their choice, without giving them operator status, it provides IRC channels a suitable method for conducting debates or giving lectures, amongst other uses. It’s also not uncommon to give voice status to acknowledge the importance of a given user to the channel (i.e., as an intermediate level between being a normal user and an operator).

The architecture of an IRC network is depicted in Fig. 2.1

[12] The registration process allows use of the nickname exclusively by the user who registered it (usually ensured through the use of a password).

[13] There are other ways for a user to make one-to-many communications but due to the fact that those are usually not used by the normal IRC user, they weren’t detailed. Because as stated, the objective is to give only the big picture of the protocol.

[14] There also exist several other channel modes, but explanation of such modes is beyond the scope of this work. Further information can be found on http://www.irc.org or the Wikipedia page about this protocol http://en.wikipedia.org/wiki/Irc

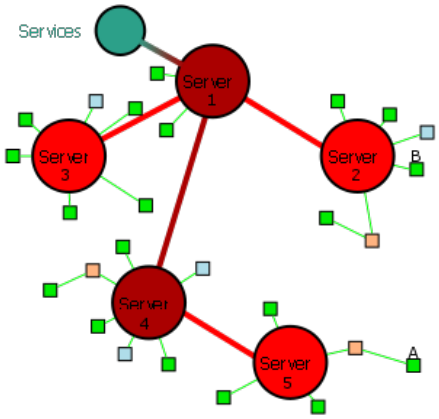

Fig. 2.1. A schema of the architecture of an IRC network – the clients are represented by the little squares, green squares are normal users, light blue squares represent bots and the orange squares bouncers. The servers are represented by the dark and light red circles (dark red if connected to more than one server, light red otherwise). Services, which may or may not exist in an IRC network, are represented by a blue circle and each line that connects each shape models a connection. The dark red server represent servers which are connected to more than one server.

The Network in Fig. 2.1 is a representation of a network composed of more than one server. Fig. 2.1 also illustrates special types of users, such as bots and bouncers, as well as a special server entity entitled Services. A general explanation of these entities is given next:

- Bots are automated clients of the network. They started as being a way for users to maintain and protect the channels the bots served. Nowadays, they are used for several different reasons (e.g. to give all kind of information to the users; to allow 24/7 logs of the channel conversations to be made, etc.)
- Bouncers function as a persistent proxy for one or more users. Its purpose is to maintain a connection to an IRC server. They can provide several kind of features. For instance, they can provide a means of logging all the communications that were made to the user while he wasn't connected to the bouncer. Therefore, with this feature, they serve a similar purpose that an answering machine serves in typical telephony.
- Services are a special kind of bots with IRC Operator status that, like the name suggests, offer services to the users. The services typically offered are nickname and channel registration. Depending on the network, the services offered can be very diverse[15], and even non-existent.

Due to the way that the architecture of IRC was conceived, it suffers from some chronic problems, namely:

[15] A simple example of the services offered by a very popular IRC network, and the evolution of the offered services can be found here: http://web.archive.org/web/*/http://www.dal.net/services/index.php3. There is also a noteworthy ongoing effort to develop a standard for IRC services, called IRC+, more info can be found on their official website: http://www.irc-plus.org/en/.

- Delay between message sending and receiving due to the network topology of IRC – which could ultimately be avoided if the network's architecture were different;
- NetSplits ;
- Nickname/Channel collisions;
- Scalability problems;

These chronic problems are explained below. There are also other issues that the protocol suffers from. A very synthetic explanation of all the problems with the IRC server and client protocols can be found on RFCs 2813 [8] and 2812 [9], respectively.

Messages between users of the same IRC network travel forcefully through at least one server[16]. As an example, taking into consideration the case depicted in Fig. 2.1, if user A, that is connected to server 5, chats privately with user B, or chats in the same channel where user B is, the message must travel through servers: 5, 4, 1 and 2 before arriving to user B. This is the root cause behind the large latency that sometimes an IRC user may experience. Obviously, this delay worsens if any link between the servers deteriorates.

When the link between servers deteriorates to the level of extinction, a NetSplit occurs. A NetSplit is basically a breaking up of the IRC network, that results in two or more IRC networks that should be communicating but aren't. (NetSplit word was formed from the words "network split"). The most immediate result of such a split, in the user's perspective, is the apparently normal leaving/quit of the network of the other users that sit on the other side(s) of the network. To allow the users to understand what happened, the quit reasons are forced to be a string with a format that contains information about the link that was broken. As an example, considering Fig. 2.1, if the link between server 5 and 4 ceases to exist, it would appear to user A as if user B had left the network. From the perspective of user B, user A would have been the one that had left the network.

A NetJoin (word formed from the words "network join") is the name given when servers on a network reconnect themselves after a NetSplit. This is also a source of problems, mostly due to the fact that all the servers must keep all the information about the users and channels that exist on the entire network. When the NetSplit occurs, the information is no longer shared. Therefore, when the NetJoin occurs, there is a high information exchange between servers. Given that the network was split, information in one side might become inconsistent with the other sides. This translates into problems in the form of nickname and channel collisions. Nickname collisions is what happens on a NetJoin, when in both sides of the network, the same nickname exists. As the nickname must be unique, this will lead to one or both of the users with the same nickname to be killed (IRC term for disconnected from the network). Whoever is killed depends on the specific IRC server implementation, which can use more or less sophisticated methods to solve the problem of deciding what to do (e.g. through the use of timestamps, etc.).

[16] Excluding the case where an IRC user establishes a direct connection to the other user via the direct client-to-client (DCC) protocol which can be used to chat or to transfer files.

The scalability problem is mostly related to the fact that all the servers on an IRC network are required to keep all the information about all the users and channels that exist in the whole network. Therefore, this is a problem due to the size of the data that needs to be known and abruptly exchanged by every server when a NetJoin occurs. Also, the servers that belong to a network must be extremely reliable due to the high bandwidth and user experience costs of the NetSplits and NetJoins. These problems account for the fact that the IRC protocol doesn't scale as well as it would be desirable.

Currently, the protocol has no support for any other type of communication besides text chat. Due to its limitations, and also because of the protocol problems that were exposed in the last paragraphs, this protocol's contribution to the IM system was dismissed. The presented details above served only as an introduction to most of the concepts that were also adopted by many IM protocols that historically followed. Also, most of the IM protocols learned the lesson from the IRC and built more decentralized network topologies.

Protocol for Synchronous Conferencing (PSYC)

This protocol first appeared in 1994 in the form of a binary implementation of a server and client (without disclosing the protocols specifications). In 1995, its draft, containing the specifications, was published. PSYC was born from the initiative and work of Carlo von Loesch [2]. It was strongly influenced by IRC and it was built with the intention to be an alternative to IRC. As an alternative, developed from scratch, it doesn't have the scalability and other chronical problems of IRC. A more concrete goal of this protocol design can be found on the following transcript from the original 1995 draft's abstract: "[this protocol is intended to] allow pairs or groups of people to communicate in real time over a computer network. It also provides for interactive information services and answering machine/notification type services when a person [psyc user] isn't currently online". More notably this protocol is being used since 2003 by MTV Europe Music Awards for the backstage video chat show[17].

Being a protocol designed with scalability in mind, it is interesting to notice that the concepts used are very similar with other concepts used in other protocols (e.g. URIs in SIP/SIMPLE, URIs in XMPP). Therefore, I chose to give a more detailed view of these concepts in the following paragraphs.

The protocol works by the use of uniforms, either Uniform Network Locations (UNLs) that are very similar to Uniform Resource Locators (URLs) [10] in structure, and Uniform Network Identifications (UNIs) that share a similar structure with (URIs) [10]. The UNLs (e.g. *psyc://bill.pcnet.whitehouse.gov:34209*) are locators that give an address to a client to connect directly. An UNL, could have been retrieved for instance, by means of a PSYC client that was trying to reach Gerhard at his publicly known UNI (e.g. *psyc://psyc.kanzleramt.de/~gerhard*),

[17] As stated for instance in: http://dbpedia.org/page/MTV_Europe_Music_Awards_2006

and whose server was instructed to allow the client to let it establish a peer to peer connection at the provided UNL. The given UNL should point to Gerhard's latest location, and it enables the PSYC user to be accessible even if he is not connected at the usual home or office computer network system.[18]

This use of UNIs and UNLs, allows the PSYC to have full control on what happens when he is not connected to the PSYC protocol. As examples:

- Gerhard's home-server can accept messages for him when users try to reach him at his UNI and he's not on-line;
- Gerhard's home-server can also notify him later to call-back when he comes back on-line;
- Also, the protocol could translate the PSYC messages to other protocols and back;

There are a vast number of possible actions that can be configured to be done. Regarding the translation, currently, PSYC supports e-mail gateways that translate between the PSYC and SMTP protocol.

Both UNLs and UNIs address objects on the PSYC protocol. These objects, (all of the objects that can be addressed by the means of uniforms[19]) are called entities. Inherent to any entity on the PSYC protocol there is:

- A state, that is a set of persistent PSYC variables;
- A trust network; This trust network works by assigning a value of trust for each entity present in an entity's network;

The Gerhard's UNI presented above is an example of a person entity. In PSYC, there also exist the so called *place entities*. A place entity is a special type of UNI that usually involves communication between various PSYC entities, that can be another place and/or person entities. Typically, a place UNI is used for chatrooms. A place UNI can also work well as a newsfeed, as described in more detail below, or any other kind of use that seems fit. One of the protocol's strengths relies on the fact that it allows a very flexible and configurable message/content distribution system.

PSYC's entities can use the PSYC's multicast technology, as well as relaying, to distribute the messages/content. Relaying can be done by any kind of entity. To relay content, the only requirement is that there must exist a mutual high level of trust between the relay entity and the sender. An example of a relaying scenario can be found on Fig. 2.2.

[18] In a similar manner that SIP URIs (e.g. Gerhard@sipdomain.com) can be used to retrieve the actual location of the SIP device registered by the SIP user at sipdomain.com

[19] *Uniforms* is the term employed to describe all UNIs and UNLs.

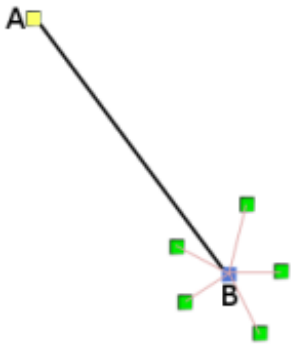

Fig. 2.2. PSYC relaying scenario – The user A, in yellow, relays the presence information to user B that propagates the information to all the users (depicted in green) that subscribed user's A presence information. User A and B must have mutual high levels of trust. The distance between the users represent the communication cost, either in time or bandwidth, therefore this setup offers more advantages than if user A sent its presence information directly to all the other users.

It is also noteworthy to expose the protocol's channels and channel inheritance features. Each entity can have a multitude of channels associated, but there exist a list of official predefined channels. Fig. 2.2 depicts a predefined channel for person entities called *#_presence*. For instance, if a user wishes to know when Gerhard is online, he would have to subscribe[20] to Gerhard's presence channel (*psyc://psyc.kanzleramt.de/~gerhard#_presence)*. The presence channel is one of the official predefined channels that exist for every person entity. Channels, along with message types (or methods) and variables, support inheritance. This document will not explain in detail the inheritance of methods and variables, only channels, but they work in a similar manner[21].

To better explain the channel inheritance, and the newsfeed usage of place UNIs, a fictitious but possible newsfeed UNI is introduced, *psyc://news.newspaper.example/@news*.
The newsfeed owner's could then provide headlines in inherited channels like *psyc://news.newspaper.example/@news#_sports*, users that subscribe to this channel would receive the sports news headlines. Channel inheritance also allows the newsfeed to have a channel like *psyc://news.newspaper.example/@news#_sports_talk* which can be defined as a chat room for users to talk amongst themselves about the sports headlines. A user that would subscribe the *#_sports_talk* channel, by the inheritance rules, would not only receive the same headlines as users that subscribe the *#_sports* channel but also the comments and talks about the headlines. In opposition, users that only subscribed the *#sports* channel wouldn't be nagged with the conversations between users that would take place in the *#_sports_talk* channel. These

[20] The act of subscribing a channel in PSYC is twofold, first the subscriber needs to make the request and the subscribed user to accept/reject it. Acceptance/Rejection can be done automatically via any kind of rules or manually through user interaction.
[21] Similar in the sense that both, method and variable, inheritances allow an extension of the methods and variables in a way similar to the one used in the channels. More information on this subject can be found on PSYC's webpage [2].

are example consequences of the fact that the *#_sports_talk* channel, as an inheritance of the *#_sports* channel, also inherits its users.

Also, this protocol has the feature of allowing integration with other known and widely used IM protocols. At the moment, support for IRC and XMPP protocols is provided by working software. The rest of the efforts to integrate PSYC with other protocols can be found on its official webpage, see table 2.1. To note that most of these integration efforts are perceived by me as abandoned.
Regarding the integration with IRC, there are several methods for it to be deployed. The one that is considered by the authors of the PSYC protocol as being the most efficient, is via a special IRC Server that acts as a gateway and must be part of the IRC network. Before closing, BrasNet[22] was an example of PSYC integration.

PSYC's efficiency in distributing messages, can be very useful for multicasting files or streams. Several methods and concrete examples of PSYC uses in various scenarios can be found in its site. The PSYC protocol supports file-transfers[23], but implementations are scarce and usually require the use of other pieces of software to work[24].
PSYC, as stated by the authors in the PSYC webpage, can also be used to implement a decentralized social network. Currently, there are ongoing efforts to use PSYC in this way. Implementations of such efforts are already available[25]. As a side note, an interesting and key-feature of the use of PSYC to implement social networks is through the use of the trust networks that the protocol already supports.

The PSYC protocol offers and absorbs many innovative and useful concepts, and has a large potential for future expansion. However, use of this protocol for this IM solution has been dismissed because:

- The clients for the protocol are still very immature;
- Future developments of the protocol are doubtful that will occur, since it doesn't seem to have a big developer or user base;
- Lack of a multiprotocol client;
- Lack of full server side interoperability with other IM protocols;

One must note that, due to the flexibility and potential of this protocol, its development should and will be watched closely. PSYC's user base growth can kick off with some good concretizations of the protocol's possible features.

[22] A defunct IRC Network, old WebPages can be found here: http://web.archive.org/web/*/brasnet.org
[23] PSYC supports file transfer in a similar manner than SIP. Both PSYC and SIP are used to negotiate the media session that actually carries the file to be transferred.
[24] For instance, HTTP[11] servers, as exemplified in http://about.psyc.eu/File_Transfer
[25] For more info about this topic, consult http://about.psyc.eu/Social_network

Retroshare

RetroShare is, by the author's general definition, "an Open Source, cross-platform, private and secure decentralised communication platform.". Its first version appeared in 21 March 2007. The protocol's network is decentralized in the sense that uses a distributed hash table (DHT) to store network information. More than a messaging protocol, it was built with the intent to offer file sharing in an encrypted and private way. The protocol has only one implementation of a client application. This client doesn't support interoperability with other IM protocols.
In this section, it is given a brief overview of the protocol, and a comment regarding its possible use in the IM solution. Since scarce amounts of technical information are available regarding this protocol, this overview, will have more emphasis on the protocol's features rather than on its architecture.
In the official documentation of Retroshare, there isn't a clear separation between the protocol and client application. They were designed together and the specifications of the protocol itself aren't fully available. Therefore, I assume that all of the features that the protocol offers are the same that the client application offers. Currently, the RetroShare platform (protocol and client application) offer the following features:

- Chatting, Instant Messaging and file sharing with friends;
- Server-less decentralized network based on DHT technology;
- Open Source protocol and client – although there is an unclear reference to a proprietary messaging protocol in the FAQ[26];
- Encrypted communication, both for chatting and file sharing;
- Ability to search among the files that friends share;
- Ability to search and retrieve the files that friends of friends share;
- Creation of channels that allow chat room style sending of messages as well as recommending files;
- Ability to add or accept/deny friends of friends to the user's network through the Auto Discovery system;
- Direct connections between clients ensure the lowest delay and the highest connection speed;
- Privacy of the list of file shares, only trusted friends have access to it;
- UPnP support for easier connectivity behind NAT/Firewall;
- Caching of user's file lists to enable accessing them while the users on the network are offline;

RetroShare's decentralized network operates using the OpenDHT[27] that relies on PlanetLab's[28] infrastructure to offer reliability and availability of the RetroShare network.

[26] FAQ available at: http://retroshare.sourceforge.net/wiki/index.php/Frequently_Asked_Questions. The question that has references to the proprietary protocol is question 5.1.
[27] OpenDHT - http://www.opendht.org/

This platform, the protocol, and the client application, could be used to deploy the IM system using the LDAP directory services to authenticate and distribute the required signatures. Unfortunately, there are some severe limitations:

- The fact that there is no multiprotocol IM client;
- There is no interoperability allowed amongst IM protocols at the server side;
- Current lack of video and audio communication;
- Unknown and proprietary protocol for transferring messages[29];

These limitations lead me to dismiss the protocol as a possible solution.

Extensible Messaging and Presence Protocol (XMPP)

The Extensible Messaging and Presence Protocol (XMPP) protocols [12] [13], and extensions[30], specify a set of open IM and IM-related technologies. The XMPP standards base their communications mostly on XML. XMPP is a somewhat mature and relatively widely deployed technology.

Jabber, the former name of XMPP, was announced in January 1999, by Jeremie Miller. Jabber was stated to be an open technology for instant messaging and presence. During 1999, there was a fast development of open-source servers, clients, and libraries. Soon after, efforts to standardise the Jabber protocol were initiated amongst IETF. These efforts eventually led to the protocol changing names to XMPP, to emphasize the difference between the protocol and Jabber Inc, and an IETF working group was created. In 2004, the standardisation efforts resulted in RFCs 3920 [12] and 3921 [13] that represent the core standards of the protocol. Although the core RFC standards are maintained by IETF, an alternative foundation was created, first called Jabber Software Foundation (JSF). The now called XMPP Software Foundation (XSF) which is responsible for discrimination of XMPP standards. XSF's standardisation process shares some similarities with IETF's process. However, XSF's standardisation process, can be characterized by the phrase: "rough consensus and running code" were sometimes the features appear implemented before the standard is written[31]. The XSF maintains the RFC standards and validates extensions that can be submitted to XSF by anyone. This kind of flexibility allowed a rapid growth of the XMPP technology. Amongst the existing XMPP extensions, one of the most prominent is Jingle [14] an extension designed by Google and XSF. In 2008, Cisco acquired Jabber Inc. with the main intent to enhance Cisco's products with IM capabilities.

XMPP continues to be developed and implemented, and is a good choice for the IM solution. Due to the protocol's potential, further details about it were gathered. Following, I present a list of XMPP's features specified in the XSF's extensions and core protocols:

- Instant messaging (mainly RFC 3921 [13]);

---

[28] PlanetLab - http://www.planet-lab.org/

[29] As stated in http://retroshare.sourceforge.net/wiki/index.php/Frequently_Asked_Questions#5-1_Is_Jabber_XMPP_used_for_the_Message_transfer_protocol.3F

[30] XMPP extensions available at http://xmpp.org/extensions/

[31] This mainly means that XSF has quicker processes for standardisation than IETF

- Presence (mainly [13]);
- Multi-party chat (XEP-0045);
- Voice and video calls compatible with SIP;
- Collaboration;
- Lightweight middleware;
- Generalized routing of extensible markup language (XML) data;
- Support of RPCs over XMPP (XEP-0009);
- Information about the user's last online date and status (XEP-0012);
- Offline message support (XEP-0013);
- Buddy lists, support for blocking users and miscellaneous buddy list functionalities (XEP-0016, XEP-0144, XEP-0145);
- Service and entity discovery (XEP-0030, XEP-0128);
- Support for streams over XMPP (in-band) and in a separate protocol (out-of-band) as well as mechanisms for negotiating out-of-band URIs (XEP-0047, XEP-0065, XEP-0066, XEP-0095);
- Bookmarks (e.g. user chatrooms, web-pages) and the ability to act on them. (e.g. auto-joining rooms) (XEP-0048);
- Search mechanism (that can be used for searching users, chatrooms, interface to conventional engines, etc.);
- Support for custom text style (fonts, sizes, color, etc.) using XHTML (XEP-0071);
- SOAP over XMPP support (useful for instance for: asynchronous, frequent, and close to real time SOAP messages) (XEP-0072);
- Advanced message processing – time sensitive delivery of messages; expiration of messages; reliable data transport of messages. (XEP-0079);
- Standardization of geographical location sharing (XEP-0080);
- Group hierarchy in buddy lists (XEP-0083);
- User avatar exchanges (XEP-0084);
- Chat state notifications, e.g. typing notifications, inactive status, etc. (XEP-0085);
- File transfer support (XEP-0096);
- Gateways, server-side IM protocol interoperability (RFC 3920, XEP-0100);
- Formalization of common IM information about the IM user. i.e. mood (XEP-0107), what he is doing (XEP-0108), what music he is listening to (XEP-0118);
- Broadcast and dynamic discovery of a XMPP entity capabilities (XEP-0015);
- XMPP through HTTP protocol with bidirectional streams support (BOSH) (XEP-0124, XEP-0206);
- Invisibility mode support – seeming as disconnected to others while remaining logged in (XEP-0126);

- Common Alerting Protocol (CAP)[32] support – Taking advantage of XMPP's ability to deliver IM in near-real-time and its native information structuring capabilities (XEP-0127);
- Waiting IM signup lists – Ability to allow an entity to be notified when a user signs up to a XMPP service (XEP-0130);
- Remote control of other XMPP clients (e.g. changing status, forward messages, etc.) useful when a user is using several clients (XEP-0146);
- Miscellaneous features – CAPTCHA support (XEP-0158);
- Jingle – negotiation of peer to peer media sessions between XMPP entities in an interoperable way with existing Internet standards (SIP, SDP). Allows simple (without some functionalities e.g. call waiting, call forwarding etc) voice and video calls (XEP-0166, XEP-0167, XEP-0176, XEP-0177, XEP-0181, XEP-0251);
- Specification of a format to support language translation (XEP-0171);
- Direct messaging between XMPP clients (XEP-0174);
- Message receipts (XEP-0184);
- Public key publishing through XMPP (XEP-0189);
- Application pings (XEP-0199);
- File sharing through a repository – experimental (XEP-0214);
- "Nudge"[33] support (XEP-0224);
- ZRTP[34] support in Jingle RTP Sessions – experimental (XEP-0262);
- Early media[35] support in Jingle (XEP-0269);
- Server filtering of XML stanzas (messages, requests, notifications, etc. XEP-0273);

The general architecture scheme, as presented in RFC 3920 [12], can be seen on Fig. 2.3.

The server's main responsibilities consist of:

- Managing connections and sessions for other entities in the form of XML streams;
- Route XML stanzas;

Also most of the XMPP-compliant servers also assume responsibility for the storage of the data needed by the clients such as contact-lists.

[32] CAP is an open format for alerts and notifications, defined by OASIS (http://www.oasis-open.org). CAP was developed to address the call, published in a (U.S.) National Science and Technology Council report, for "a standard method ... to collect and relay instantaneously and automatically all types of hazard warnings and reports".

[33] A nudge, is a call for attention from one IM user to the other. Unless disabled by the client, it makes a special action to call the attention of the user. (e.g. a different sound, IM client window popping up, etc)

[34] ZRTP is a cryptographic key-agreement protocol to negotiate the keys to encrypt VoIP phone calls. ZRTP method, uses Diffie-Hellman to provide key agreement, and SRTP for content encryption.

[35] Early media are all kinds of media that is sent before the actual media session is established. (e.g. the tone a caller listens to when a call is ringing, announcements, etc.)

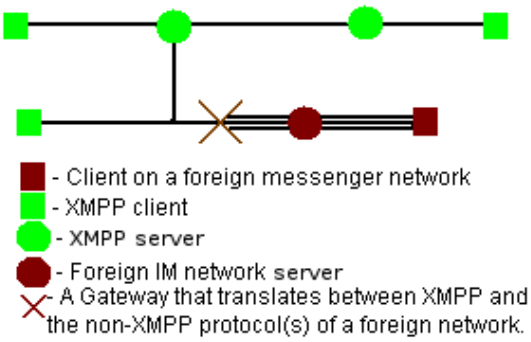

Fig. 2.3. XMPP architecture - The general architecture supported by the XMPP protocols. The *simple single black line* represents XMPP connections between the entities it connects. The *three black lines* represent a connection done in a foreign (non-XMPP) protocol.

Clients usually connect a TCP connection to the XMPP server on port 5222[36]. Multiple devices or locations can connect on behalf of one user. These devices and locations (e.g., PDA, mobile phones, work desktop P.C., home desktop P.C.) are called resources and are distinguished and named after the slash in a typical XMPP address (e.g., <user@domain/resource>).
Gateways are a special server-side service whose primary functions are to translate between the XML streams and other foreign protocols. Examples of such gateways are gateways to email (SMTP), Internet Relay Chat (IRC), SIP/SIMPLE, Short Message Service (SMS), and other proprietary instant messaging services such as AIM, ICQ, MSN Messenger, and others.
XMPP networks work essentially by relaying, through XMPP servers, the messages from users that belong to a domain to other users of other domains, as depicted in Fig. 2.3. This kind of network topology is criticized by PSYC's authors, see subsection *'PSYC'* above for more information about PSYC, because of the inherent scalability issues.
The XML's general data stream formats, and similar details, are omitted in this paper.

Recognising the importance of the SIP/SIMPLE protocol, XSF has standardisation efforts to ease the interoperability between the XMPP and SIP/SIMPLE protocols. Besides the possibility of inclusion of gateway entities (that allows translation between XMPP and other protocols, including SIP/SIMPLE) several extensions take into account the interoperability of these two protocols. A full list of efforts present either in extensions or in core specifications is provided next:

- The XMPP protocol extensions for communicating URIs between Jabber entities support SIP URIs (XEP-0066);
- The extension that specifies how geographical information data of entities should be exchanged has a section that maps the defined message format to the Presence

[36] Which is the recommended port for connection between a client and a server as registered with the Internet Assigned Numbers Authority (IANA).

Information Data Format (PIDF, [15]). PIDF is a specification developed by the SIMPLE working group (XEP-0080);

- Unsupported characters in Jabber Identifiers (JIDs) that can be part of identifiers of other protocols are converted to allow interoperability between XMPP and those protocols. The extension that specifies the conversion (XEP- 0106) has a sub-section entirely dedicated to give examples of the conversion to SIP URIs;
- The XMPP protocols were extended to allow negotiation of out of band multimedia using the SDP protocol. SDP is the standard protocol used by SIP to negotiate multimedia sessions. The TINS extension (XEP-0111) allows XMPP to be used to exchange SDP content, replacing SIP, but at the same time facilitating the interoperability between TINS and SIP through the use of gateways;
- Encapsulation of data that otherwise would be difficult to transmit using only the original formats specified by XMPP – These allow the XMPP protocol to transmit any kind of metadata such as the one defined in the SIP standard (XEP-0131);
- XSF efforts to change the user's profile format to take into account the SIP addresses (XEP-0154);
- An extension that provides ways of negotiating the exchange of XML stanzas, takes into account the SIP protocol. It provides resources on how to map the negotiations to the SIP protocol, enabling developers to implement such a negotiation through a XMPP/SIP gateway (XEP-0155);
- Jingle's extensions (XEP-0166, XEP-0167, XEP-0176) try to fully maximize the interoperability between the Jingle and SIP protocols, allowing Jingle/XMPP clients to be connected to current VoIP networks that use SIP;
- Specifications to translate between the Common Profile for Instant Messaging (CPIM) format, which is the format used on the SIMPLE session mode, to the XMPP format, are specified in RFC 3922 [16];

The XMPP protocols and software compliant applications are currently used by some companies to provide for its IM needs. Two of the testimonies given in the 54th IETF meeting in Yokohama, Japan, in 2002, that gave birth to the XMPP IETF working group (WG), were given by employees of companies.[37] Dale Malik from the Bellsouth company talked about the use of the XMPP protocols as part of a commercial offering provisioned for 1.5M users. Alexandre Noell, from the France Telecom gave the example of the protocols being used by a commercial IM system that at that date had around seven million users. The France Telecom adopted XMPP standards (at that time Jabber standards) after pursuing the development of its own solution.

The famous Portuguese IM system, SAPO Messenger[38], also uses XMPP.

[37] Extracted from the proceedings of the 54th IETF conference. Details extracted from http://www.ietf.org/proceedings/54/130.htm

[38] SAPO Messenger webpage is available at: http://messenger.sapo.pt/

Google Talk's technologies[39] based their protocol in the XMPP and they were behind the design of the Jingle extension to allow them to do SIP-like signalling of voice and video calls.

The XMPP set of protocols and its extensions make it a very attractive protocol. Interoperability possibilities, extensibility, and available software, make XMPP protocol and extensions, a possible choice for the IM system. Even if XMMP is not chosen, due to the popularity and set of features of the service, an inclusion of a XMPP SIP Gateway is something that has to be considered at least for future developments of the IM System.

SIP/SIMPLE

The SIP protocol [17] is a generic signalling protocol that serves the purpose of creating, modifying, and terminating generic purpose sessions with one or more participants. SIP is an open standard currently maintained by the IETF[40].

SIP first appeared in 1996 and was originally designed by Henning Schulzrinne and Mark Handley. Since 1996, SIP standards evolved. Currently, SIP is deployed in numerous VoIP solutions and is also used for miscellaneous multimedia distribution, including multimedia conferences.

SIMPLE[41], is an IETF working group created with the main purpose of endowing SIP with instant messaging and presence (IMP) capabilities. The group, created at the end of 2000[42], successfully accomplished the standardisation of protocols and SIP extensions that currently permit SIP to be used as an IM protocol. Many of the standards that allow SIP to have presence, and (page mode) instant messaging, have open implementations of both servers and clients. This section gives an overview of how the protocol works and how it is structured.

Before explaining the method extensions that SIMPLE provided to the SIP protocol, some basic definitions of SIP terms must be presented for understanding the SIMPLE contribution. The specific terms of the SIP protocol, that are used on this section and their official definition, as encountered in [17], are:

- *Message*: Data sent between SIP elements as part of the protocol. SIP messages are either requests or responses.
- *Method:* The method is the primary function that a request is meant to invoke on a server. The method is carried in the request message itself. e.g. INVITE and BYE methods.
- *Proxy, Proxy Server*: An intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays the role of routing, which means its job is to ensure that a request is sent to another entity "closer" to the targeted user. Proxies are also useful for enforcing policies (e.g. making sure

[39] http://www.google.com/talk/
[40] IETF – **I**nternet **E**ngineering **T**ask **F**orce – http://www.ietf.org
[41] SIMPLE Working Group charter – http://www.ietf.org/html.charters/simple-charter.html
[42] As can be inferred from the December 2000's proceedings http://www.ietf.org/proceedings/00dec/index.html

a user is allowed to make a call). A proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it.

- *Redirect Server*: A redirect server is a user agent server that generates 3xx responses to requests it receives, directing the client to contact an alternate set of URIs.
- *Session*: A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session – as found on [18].
- *Event Package*: An event package is an additional specification which defines a set of state information to be reported by a notifier to a subscriber – as found on [19].
- *Notification*: Notification is the act of a notifier sending a NOTIFY message to a subscriber to inform the subscriber of the state of a resource – as found on [19].

SIP is a text-based protocol, like most of the IETF protocols. SIP messages can either be a request from a client to a server or a response from a server to a client. Requests can easily be differentiated from responses if it's taken into account that responses have a three digit status code. The first digit in a response, classifies the response per type: (i.e., 1xx responses are provisional ones; 2xx responses are indicators of success; 3xx responses are redirection responses; 4xx are request failure responses; 5xx are server failure responses and 6xx are global failure responses);
Whatever the mode implemented by a SIP based IM system, it relies heavily on the SIP architecture that is explained below.

SIMPLE, as stated above, has the responsibility to define standards that enable presence in addition to the IM capabilities. Presence information can be used not only for IM purposes but also for the other typical SIP signalled multimedia sessions in general (e.g., video and voice). Currently, SIMPLE already defined the extensions for the NOTIFY and SUBSCRIBE methods [19] that allow generic event[43] notification.

These standards are sufficient to allow a user agent to be notified of presence information regarding his buddies. However, the user agent will have to issue a SUBSCRIBE request to each of them. To solve this problem in a more efficient way, standards [20] [21] were created that define document formats that specify resource[44] list servers (RLSs), and ways of managing the resource lists contained in them. Also, user agent's are allowed to change its state regarding any event package through the fully specified PUBLISH method [22].
Buddy lists, privacy policies, and other data that is represented in XML, can be managed by the user through the use of *'XML Configuration Access Protocol'* (XCAP). All of the XML documents

[43] An example of a possible event, is the message waiting indicators that can notify users of waiting to be read voice-mail messages. Other example of an event is notifying users when their buddies are online.
[44] User resources can be of very distinct kinds. As an example, a resource could be a user's buddy list.

regarding presence, privacy and policy have a well defined syntax in several RFCs (for further RFC reference, please consult IETF's draft 'simple-made-simple' [24] sections 2.2 and 2.3).

Still regarding presence capabilities, various standards that offer:

- Presence document formats;
- Privacy and policy considerations;
- Protocols and norms on the management of data, by the user, required for the presence system, e.g. buddy lists management – XCAP mostly;
- Interconnection of different presence and instant messaging systems for the purposes of federation;
- Optimizations of all the relevant standards for wireless links;

Have already been written and validated.

A full list can be found on [24], but it's only the purpose of this document to give a more in-depth overview of the standards that are considered as the presence core protocol machinery.

There are two main modes of instant messaging defined by the SIMPLE standards:

- Page mode (see example in Fig. 2.4) – were the messages are sent in the body of a SIP request;
- Session mode (see example in Fig. 2.5) – the instant messages are exchanged in a media session that is negotiated via SIP. This media session negotiation is similar to the other common ones used in voice and video calls established through SIP[45];

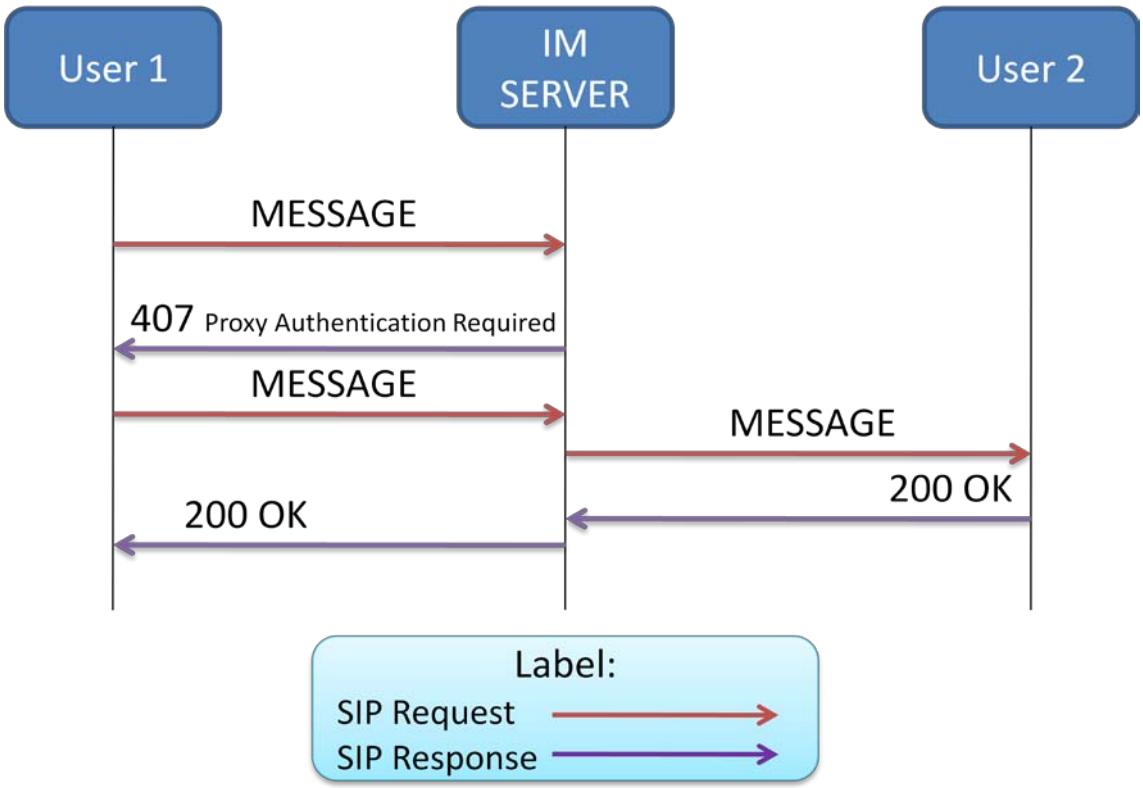

Fig. 2.4. SIMPLE's peer mode message exchange example diagram.

[45] Session establishment follows rules specified in RFC 3264 [25], where more details can be found.

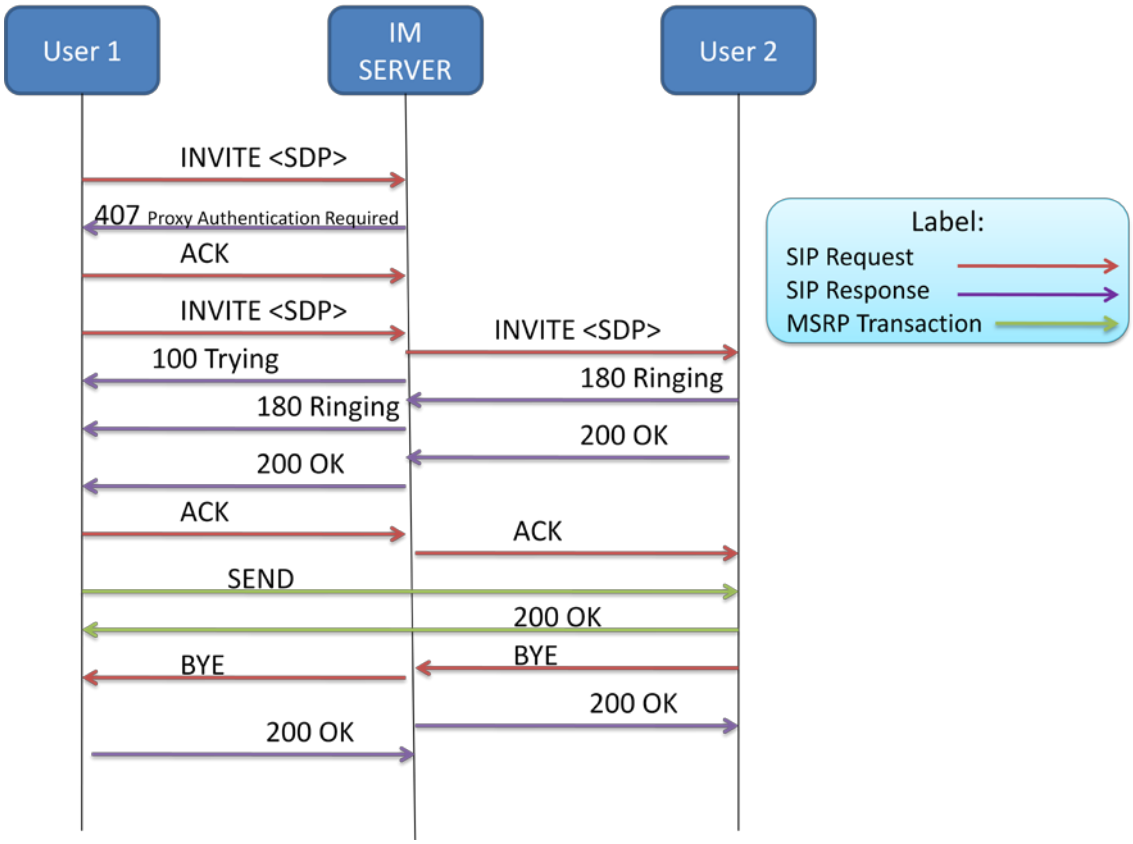

Fig. 2.5. SIMPLE's session mode example diagram.

Page mode uses SIP requests to carry the instant messages. Therefore, on a network that already has a SIP services infrastructure, there is no need to add extra servers. The only requirements are that the SIP servers support the extensions that enable the presence and IM services (which most open source implementations of SIP server software do), and adjust the configuration so that IM and presence requests and responses are routed and treated accordingly.

Session mode uses SIP to control sessions that itself carry the instant messages. Therefore, if session mode is used, the instant messaging is seen as another media type controlled by SIP. Message Session Relay Protocol (MSRP) is the chosen protocol to carry the media in a session mode set-up. More features such as file-transfer could be implemented with the use of the MSRP protocol. Details on the contributions of MSRP to SIMPLE's world can be found below in the subsection '*MSRP's role in the SIMPLE architecture'*.

While page mode is more efficient for one or two message conversations, session mode is more efficient for longer conversations. Page mode has the great disadvantage that all of the messages are exchanged through the SIP server, making it a potential bottleneck. Therefore, session mode allows the IM system to serve more users with the same resources (because the SIP server doesn't have to handle as many requests as it needs with page mode).

MSRP's role in the SIMPLE architecture

MSRP is a resource efficient way to deliver general purpose messages. MSRP works over TCP connections. MSRP also has support of secure TCP communications, TLS, therefore making possible for MSRP to inherit TLS's security features[46]. These messages can be in any format, therefore making possible an additional end-to-end format that also provides encryption and other security features.

The MSRP-related specifications, [26] [27] and [28], also provide other important MSRP entities. MSRP's entities and a synthetic explanation of their role are summarised below:

- MSRP Peer [26] – Peers are the main responsibles for the communication. They generate and receive the messages as well as negotiate (through the use of SIP for instance) and establish the MSRP sessions.
- MSRP Relay [27] – Relays are entities whose main purpose is to help MSRP peers to communicate. They offer a simple way to provide accessible network paths. A relay, better explained through observing Fig. 2.6, is of vital importance to allow a MSRP session to go through NAT and firewalls for instance.
- MSRP Switch [28] – These are hubs that allow more than two MSRP peers to connect at the same time. They basically route the content to the peers to whom the content is addressed to.

The use of these entities, makes possible scenarios as the ones presented from Fig. 2.6 to 2.9. MSRP, being a general purpose protocol to carry messages, can be used as an enabler for other activities and services, such as:

- Desktop sharing [35];
- Whiteboard sharing, easily implemented in a similar manner to desktop sharing;
- File transfers;
- File repositories / P2P file systems – as an example, consider Fig. 2.9;
- Others..

[46] TLS provides: authentication, privacy, and integrity validations.

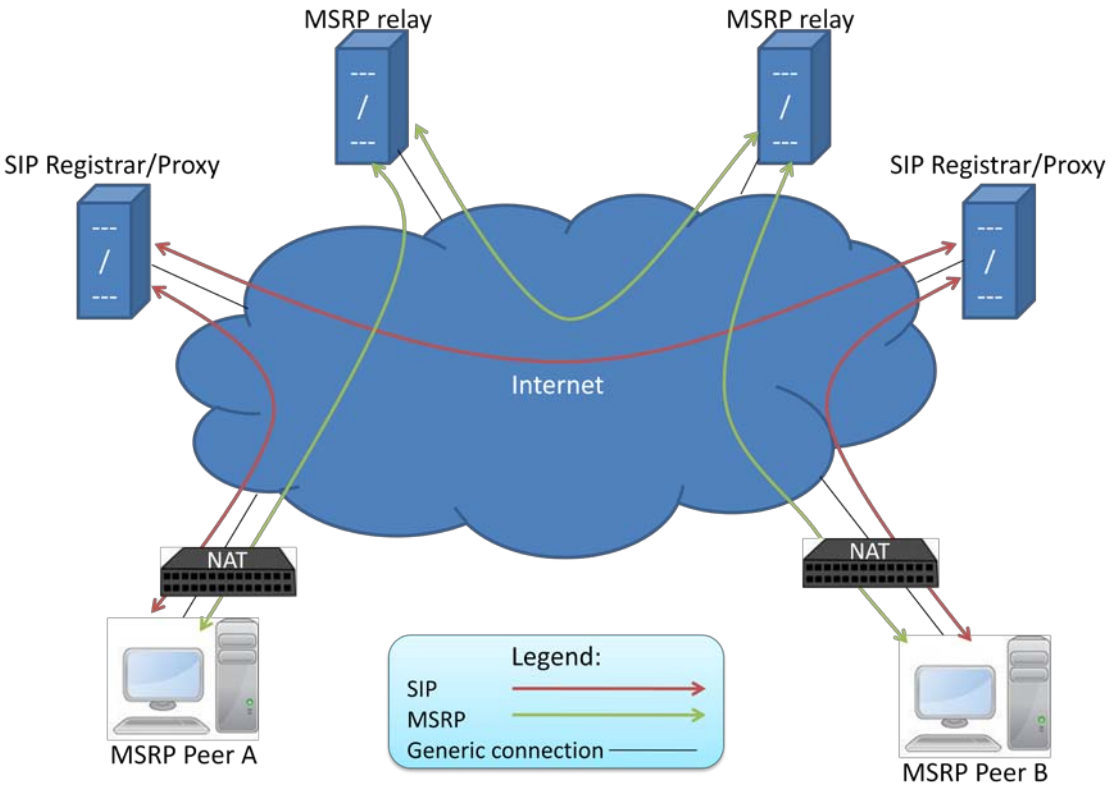

Fig. 2.6. Using MSRP relays and SIP negotiation to establish a MSRP session by peers A and B that sit behind NAT.

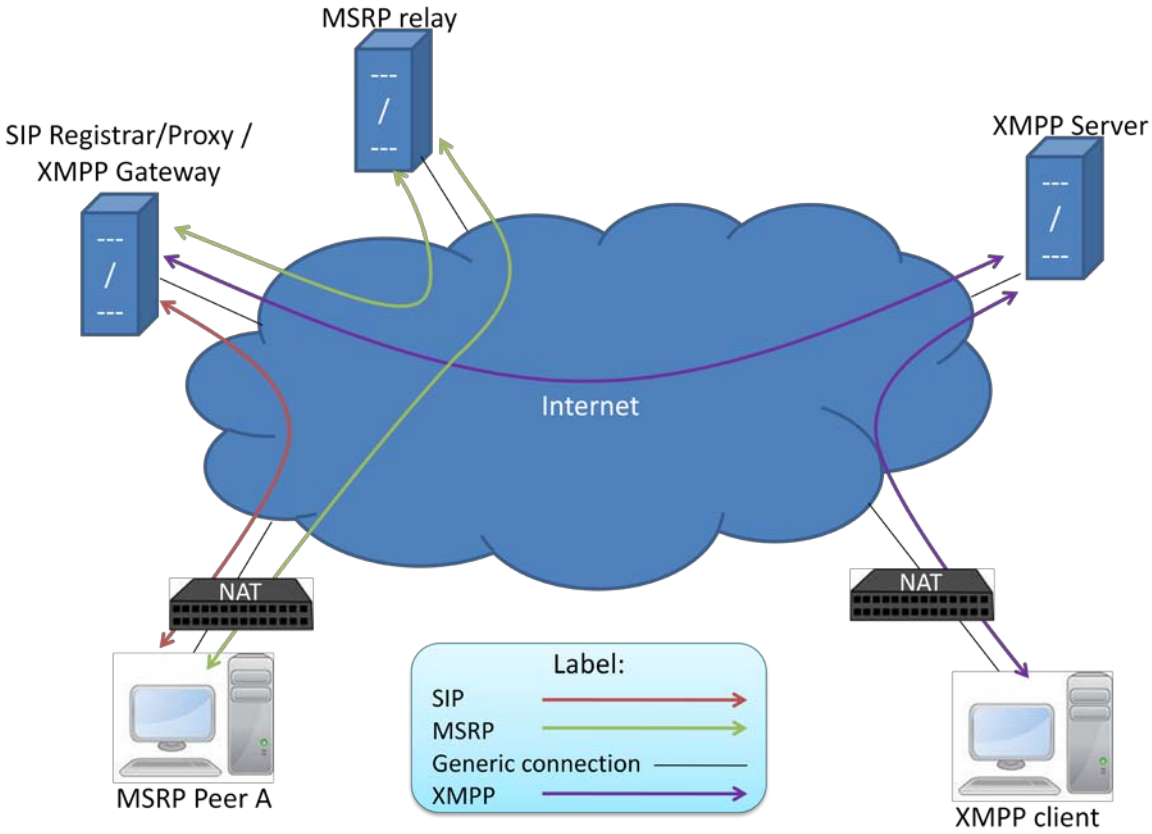

Fig. 2.7. Using MSRP to XMPP translation to interconnect a SIP/SIMPLE and a XMPP user.

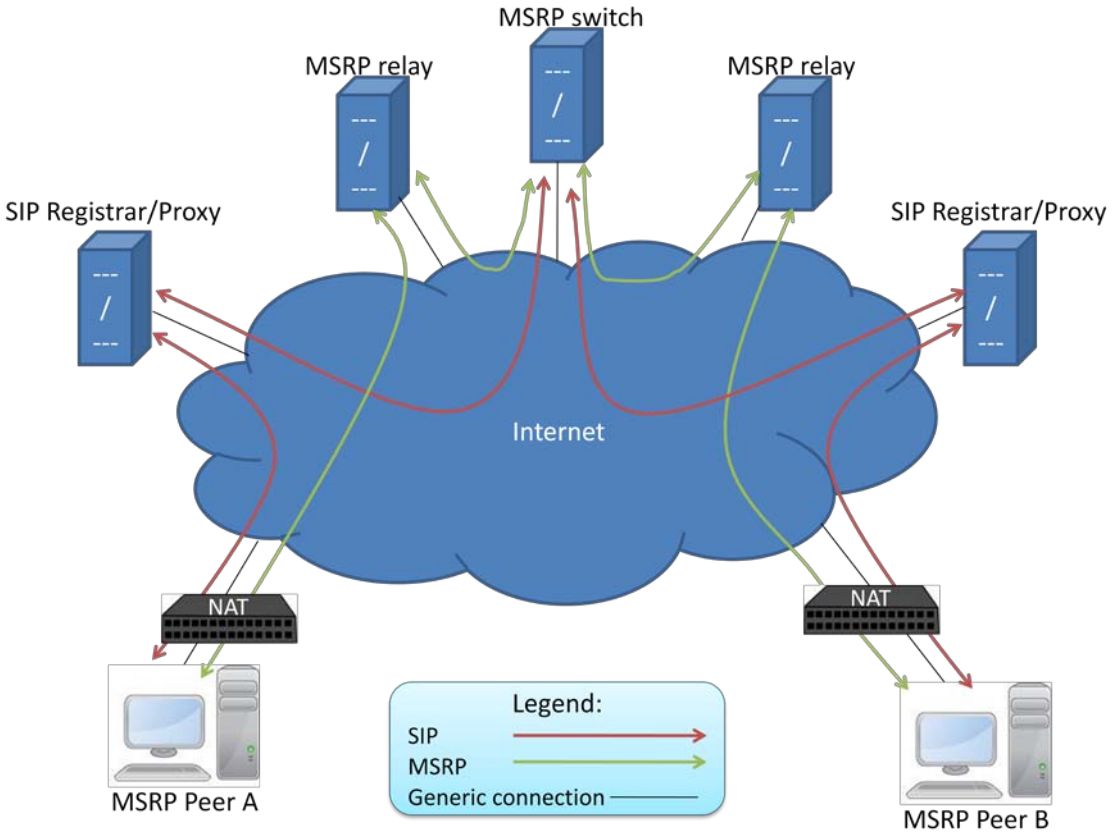

Fig. 2.8. Use of MSRP switches [28] to enable MSRP conferencing.

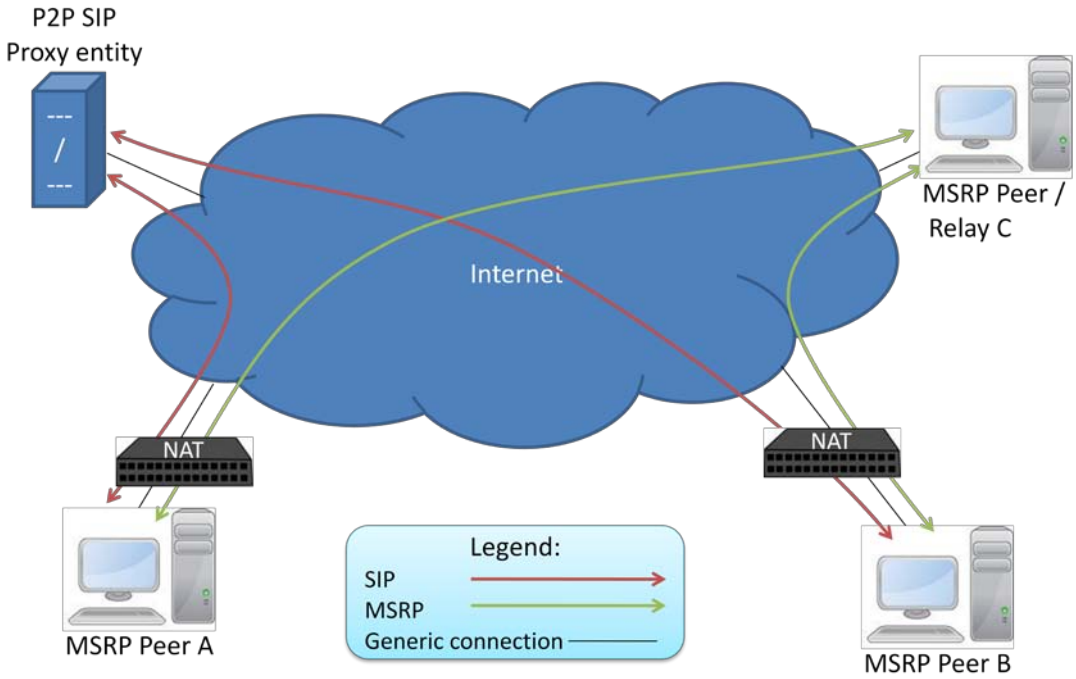

Fig. 2.9. Using relay/peer software to enable P2P networks through the use of MSRP;

Current MSRP open implementations and libraries;

A more or less thorough search, revealed some finished and ongoing open source efforts for implementing MSRP's libraries, relays and clients:

1. MSRP Relay[47] – relay software in python;
2. SIP SIMPLE client library[48] ;
3. LibMSRP[49];
4. Python SDK [50] – The SDK has a MSRP lib, that both the relays and the SIP SIMPLE client library use;
5. Blink[51] SIP/SIMPLE full featured client – pre-release;
6. C++ OPAL SDK[52] – MSRP chat support;

The MSRP Relay (1) project provides a Python implementation of the MSRP Relay entity. This software is available and working. It features integration with OpenSIPS SIP Server/Proxy/Registrar database authentication.

The SIP SIMPLE client python library (2), which is a work in progress, pretends to be a comprehensive library for many of the SIMPLE standards. Based on its MSRP API documentation[53], it seems less of a generic purpose MSRP library as it is more oriented for IM MSRP specific uses.

LibMSRP (3) supports: chat, file transfer and desktop sharing. Judging by the MSRP's API documentation, currently, only support for chat is implemented. LibMSRP was driven by an IMS implementation project called CONFIANCE[54]. LibMSRP Sourceforge's site, lists the last activity as being from two years ago. In a similar state, according to CONFIANCE's Sourceforge webpage, CONFIANCE's last activity tracked is dated from two years ago. Therefore, LibMSRP and CONFIANCE, seem abandoned. According to the available LibMSRP documentation and source code (version 0.2), LibMSRP is not a generic purpose MSRP library as LibMSRP v0.2 only supports MSRP chat sessions.

The Python-SDK (4) has a library that provides MSRP functionalities to the Python developers. This library is very valuable to any developer that wishes to use MSRP. Currently, for instance, the MSRP Relay software (1), the SIP SIMPLE client library (2), and Blink (3) use this library.

The Blink (5) client is announced to be a fully featured SIP SIMPLE client. Due to its potential and features, more details about this client can be found in section 3.1.1.

[47] Available at: http://www.msrprelay.org/
[48] Available at: http://sipsimpleclient.com/
[49] http://sourceforge.net/projects/libmsrp/
[50] Available at: http://www.python.org/
[51] http://icanblink.com/
[52] http://www.opalvoip.org/
[53] Documentation available at http://sipsimpleclient.com/wiki/SipDeveloperGuide
[54] http://confiance.sourceforge.net/

C++ OPAL SDK (6) is a full-featured library that supports a variety of protocols and multi-media formats for open source voice, video and fax implementation. Currently, MSRP chat is supported in its Lalande (currently stable) version.

MSRP's implementations (1), (2), (4), and (5) are recent, and in my opinion, they will allow a fast change in scenario concerning the widespread use of SIMPLE for IM systems.

A word on SIP/SIMPLE adoption

There is an existing lack of software support for the session mode of SIP/SIMPLE, most probably because the most prominent MSRP implementations are recent (as detailed in *'Current MSRP open implementations and libraries'* subsection). However, I believe that now there are conditions for this scenario to change rapidly.
Page mode is supported by many software applications. Page mode's worst disadvantage is the possible bottleneck on the SIP server. Avoiding page mode's bottleneck is as simple as making sure that the IM server(s) support the number of potential users. Future adoption of session mode depends only on client support for it. In most cases, it should not even require no changes in the SIP server configuration (that usually is blind to the type of media sessions that are being negotiated through them).
Audio and video capabilities inherited from the SIP protocol are also in most cases supported by SIMPLE clients.
SIMPLE is completely specified and maintained by an IETF working group. SIMPLE standards are supported/specified and maintained by many people that actively contribute to the set of SIMPLE related specifications.
Therefore, SIP/SIMPLE represents a good choice for the IM system. It would represent a great choice if more open-source implementations existed. However, most of the features are standardised, so interoperable implementations are possible.

Zephyr

The Zephyr Notification Service appeared in the late 1980's as an effort initiated by Ciarán Anthony DellaFera, as part of MIT's Project Athena [29]. It was one of the first IP-based instant messaging systems to appear, and although it has been replaced with more popular IM systems, it still can be found on a few university environments.[55] Despite the existence of replacements to this protocol, which allow other kinds of real time communication, this was an important system that still has unique features such as workstation login notifications and others.
In this section a feature overview of the protocol's features is given. The protocol's architecture wasn't explained due to the fact that this protocol is being abandoned and only figures here due to its pioneer status. Also, it is given a brief comment on the utilization of some of the protocol's concepts as part of the IM system solution.

[55] Namely: MIT, CMU, Iowa State University amongst other north American universities.

Zephyr is a notification system that also allows a user to exchange instant messages and locate other users. It was designed to be integrated in a workstation environment that supported MIT's Kerberos [33] authentication. Its original client and server applications were designed and implemented to work on Unix systems. Its original clients were a command-line *talk* like program and a limited user interface for the X-Window system.
Zephyr allows:

- A user to locate other users that configured the service to disclose such information;
- Receive miscellaneous notifications – including logins and logouts of users;
- Subscribe to only certain type of notifications;
- Broadcast messages/notifications;
- Send and receive instant messages;

It's also worthy of mentioning that, from the IM clients considered, there are two modern multiprotocol IM clients that support the Zephyr protocol, Adium and Pidgin .

I find that the Zephyr protocol doesn't represent a viable solution to the IM system being developed. However, some of the ideas behind this protocol are to be considered of great value in implementing IM in an academic community. More specifically:

- The notification system, that could for instance allow notification for mail arrival/printing job status/network, workstation logins, etc.

Popularity of the IM protocols

This section provides an overview of the most popular IM protocols available. Both open standard and closed standard protocols are considered in this section.
An investigation of the protocol's popularity helps to assign different values to different possible server and client side interoperability features.
The popularity indicator used is the protocol's worldwide user base. A more ideal indicator would be the user base of the protocols restricted to the university's community, that for instance could be known by means of an enquire to the community. Unfortunately, such statistical work wasn't available on the time of writing of this section. Table 2.2 provides the popularity ranking of the protocol based on the user base and the sources were the information was collected.

Protocol/Service	User count	Date/source
Tencent QQ	856,2 million total accounts (majority in China) 355.1 million active	12 November 2008[56]
Windows Live	Over 300 million active	Unknown, somewhere

[56] Third Quarter 2008 Tencent QQ results press-release: http://www.tencent.com/en-us/content/ir/news/2008/attachments/20081112.pdf

Messenger		in late 2008, 2009[57]
Yahoo! Messenger	248 million active	17 January 2008[58]
Skype	About 13 million active; Total of 338,2 million, that includes inactive and duplicate accounts	16 July 2008[59]
AIM	53 million active >100 million total	September 2006[60]
Jabber	90 million total	January 2007[61]
eBuddy	18 million active	4 December 2008[62]
ICQ	Around 420 million total, including inactive and possible duplicate users; 30 million active	March 2008[63]
Xfire	Around 12 million total	January 2009[64]
MXit	More than 4.8 million total	10 August 2007[65]
Gadu-Gadu	6 million total	21 August 2008[66]

Table 2.2. Protocol's popularity by user count indices.

Most of the information presented in table 2.2 are from different dates. Therefore, the presented figures may differ significantly from the current reality but in my opinion, they still serve as a good indicator.

[57] As divulged in the video: http://www.microsoft.com/windows/windows-7/beta-videos.aspx?vindex=7
[58] Journal article – http://www.searchenginejournal.com/yahoo-to-support-openid-for-its-248-million-users-openid-to-support-yahoo-ids/6258/
[59] The article: http://skypejournal.com/blog/2008/07/skypes_fy08q2_results_generate.html, the Ebay Inc. Second quarter 2008 results: http://files.shareholder.com/downloads/ebay/226550478x0x213648/375348ca-3cd2-4c58-98a5-e24a06092b30/213648.pdf
[60] As can be found on the estimates in: http://arstechnica.com/news.ars/post/20060927-7846.html
[61] Based on calculations of Process-One: Process-One uses ejabberd as Jabber server software. If it is assumed that ejabberd has a 40% market share amongst public and private open source server deployments, there are 50 million users using open source servers. With Jabber Inc's numbers, this adds up to the 90 million number stated here. http://www.process-one.net
[62] As found on ebuddy press release: http://www.ebuddy.com/press/eBuddy_10M_mobileIM_4Dec2008.pdf
[63] As found on: http://www.pr-inside.com/icq-celebrates-30-miilion-active-users-r596754.htm
[64] The number of total users of XFire is displayed on their main page http://www.xfire.com
[65] As found on the article: http://www.bizcommunity.com/Article/196/78/17027.html
[66] As found on the article: http://tinyurl.com/8wud8l

As a user of the target community, it's easy to get the perception of the most popular IM network. My perception is that the MSN messenger IM platform is the most used in IST's community. It is also perceptible that although the ICQ, AIM and Tencent QQ have a very big worldwide user base, they are more localized in regions outside of Portugal. Also, Gadu-Gadu and MXit have spread mostly on Poland and South Africa respectively and have few users outside these countries. It was also perceived by me, that some of the users of the IST community use Skype and Google Talk IM regularly.

2.2 Instant Messaging Clients

The number of IM clients and protocols proliferated over the last years along with the number of Internet users [32]. Success and usefulness of the IM system deployment, can be measured by the number of users that adhere to it and use it on a daily basis. Choosing and promoting an IM client as part of the system, is a needed and critical task, because this will be the primary interface for the user with the rest of the IM system.

Developing an IM client for the system from scratch would be too much resource consuming and ultimately a waste of resources given the already available IM clients.
There is a large spectrum of possibilities to choose from for the IM client. In order to make this list smaller, I devised the following list of requirements:

- *Multiprotocol IM client* – I assume that the large majority of the most probable potential users of IST's IM system are users that already use IM and have their own IM accounts in the most popular protocols. Therefore, having an IM client application that provides compatibility with the IST's IM system and other IM systems, can function as an unifying application. Liberating the user from the hassles associated with using different applications for the same purpose;
- *Free to use application* – There are lots of good and free IM clients. It makes no sense to promote/choose an IM client for which users or IST needs to pay for.
- *Support for the major features (file-transfer, instant messaging) of the major IM protocols*:
    - MSN Messenger;
    - XMPP/Google Talk;
    - SIP/SIMPLE;
    - ICQ;
- *Support for the following operating systems* (a must, even if the client is open source, due to the issues involved in porting a software to another operating system (OS).)*:*
    - Microsoft Windows NT family – A must, as this is the most common OS used;
    - Mac OS X – Not required, but very valued;
    - Linux – Not required, but very valued;
    - Other OSs – Not required, valued;

Table 2.3, provides a list of all the IM clients that were founded suitable given the requirements. The criteria to compile the list on table 2.3 was that the IM client already fulfilled or has good perspectives of fulfilling the requirements in 2009. In the case that they don't fulfil the requirements, they can still figure on the list if they are Open Source (so that they could be altered in order to fulfil the requirements). The clients that figure on Table 2.3 are compared in the next subsection, entitled *'Choosing between the suitable IM clients found'*.

IM client name	Available for download at:
AYTTM	http://ayttm.sourceforge.net/
BitlBee (IM Gateway)	http://www.bitlbee.org/main.php/news.r.html
CenterICQ	http://thekonst.net/centericq/
Climm	http://www.climm.org/
InstantBird	http://instantbird.com/
Miranda	http://www.miranda-im.org/
Naim	http://naim.n.ml.org/about
QuteCom	http://www.qutecom.org/
Pidgin	http://www.pidgin.im/
Sim-IM	http://sim-im.org/wiki/Main_Page
Sip-Communicator	http://www.sip-communicator.org/

Table 2.3. – List of considered IM clients/gateways – with the websites where they can be found.

Choosing between the suitable IM clients found

In this subsection, I offer points against and/or for choosing each of the IM clients that are listed in table 2.3. With the given comments in this subsection, I further limit the possible choices of IM clients. The final choice and reasons are given for the solution are detailed in section 3.1.2.

From all the clients in table 2.3, I choose to dismiss the following:

- Climm;
- Naim;
- CenterICQ;

Due to the lack of a graphical user interface (GUI). I believe that most users are used to work with IM clients that have a graphical interface. Therefore, providing and promoting an IM client that doesn't have such a feature would go against user's expectations.

AYTTM doesn't support file transfers in the main protocols. It also doesn't have, in my opinion, an appealing GUI (with a modern feeling to it). In the list, there are other IM clients that have more appealing GUIs and support file transfer for the main protocols. These three reasons made me exclude AYTTM as the choice for the IM client to promote.

BitlBee, is the only IM gateway that appears in table 2.3. BitlBee users connect to the gateway through the IRC protocol. The gateway, is then responsible to connect to the other IM protocol's servers and translate messages back and forth between those protocols and the IRC protocol. It is a very interesting concept, unfortunately, very few features of the most popular IM protocols are supported. Therefore, it doesn't represent a viable choice to be promoted and used by the system.

Instantbird is a potential candidate. It has a more appealing GUI than AYTTM. It supports the basic features for the more popular IM protocols. Unfortunately, the latest non-alpha version (0.1.3.1) doesn't support the major features and some other basic things like contact deletion, etc. By inspecting the project's Roadmap, it hasn't yet attained the 0.3. version, that is considered to be attained when the client has: *"everything that is an absolute necessity on a modern IM client"*. Also, in the project's roadmap, it can be verified that the current available version hasn't feature parity with Pidgin. To sum it up, due to the project's underdeveloped state, it doesn't yet represent a viable choice taking into account the other possibilities.

Miranda is a very decent choice of IM client. The following disadvantages were identified, that exclude it from the solution:

- Doesn't support SIMPLE IM protocol[67];
- No audio/video support;
- No support for other operating systems rather Microsoft Windows;

QuteCom has all of the requested features. The only disadvantage is that it has no support for file transfers in the main IM protocols;

Pidgin is also a very decent choice. The only disadvantage is the lack of video support for most of the protocols, with the exception of XMPP in Linux only.

SIM-IM's development is abandoned. It also only supports the basic features of the main protocols. It has no support for SIP/SIMPLE. These reasons make it a bad choice for the IM client to be promoted for the IM system.

Sip-Communicator also represents a good choice. It supports all of the requirements. Due to the fact that is written in Java, it is cross platform and supported on most operating systems. It has a working prototype version for the Google's mobile operating system Android. Currently the main disadvantages are:

- Doesn't support file transfer for all of the protocols – this is an on-going effort, file transfer for most of the protocols is attained in the latest version in the SVN;

[67] Although a plugin was found in the plugins IM client webpage, it is dated from 2007 and it doesn't seem to work with the latest version of Miranda IM.

From the analysis made above, the IM clients that are considered a viable choice at this point are:

- QuteCom;
- Pidging;
- Sip-Communicator;

Chapter 3

Implementation

This chapter details the carried implementation work as part of this thesis. It starts by presenting the motivation and choices that were made for the IM solution and goes on to detail each of the three sub-projects that were developed to make the solution a reality.

3.1 Solution

Gathered the state of the art in the available IM protocols and IM clients, in chapter 2, it's time to make an informed decision about the solution to implement the IM system. This section presents the chosen solution, as well as the motives for the choices made.

3.1.1 Motivation

SIP/SIMPLE vs XMPP – the only two real possible choices

After doing my research on the available IM protocols and clients it became very clear to me that the choice of which protocol to be used would be either SIP/SIMPLE or XMPP. The main reasons behind this are:

- Both SIP/SIMPLE and XMPP have reliable and considerable size organizations (SIP/SIMPLE has IETF[68], and XMPP the XSF[69]) that produce and provide oversight to the specifications. These organizations are open for contributions from everyone. None of the other considered IM protocols has such kind of organizations behind them. In both organizations, several contributions have been made from industry leading companies that in most cases also implement the protocols in their products.
- SIP/SIMPLE and XMPP are in practice more deployed and enrooted than others. They count with numerous server and client implementations that the other considered protocols don't have.
- Both SIP/SIMPLE and XMPP protocols have more features already specified and being specified than any of the other protocols.

Therefore I consider that both these protocols are in a healthy state, because they are actively being developed and maintained. These reasons make me believe in the prediction that they represent the future of openly specified IM protocols.

PSYC [2] is also a very promising protocol. However, in my opinion, its development is actively maintained by a too small number of people. Therefore I believe that it has a dubious future ahead of it. With current development efforts, I find it very improbable that PSYC [2] gains sufficient critical development mass to have an important role in the IM open specifications scenario. Especially given the XMPP and SIP/SIMPLE alternatives.

[68] More info can be obtained in their official website at: http://www.ietf.org/
[69] Information about this organization can be found at: http://xmpp.org/xsf/

To further help with the choice, the pros and cons of adopting each of the two protocols is given:

XMPP – Pros and cons.

Reasons to choose XMPP:

- There are a great number of specifications both wrote and being written;
- The XSF (XMPP standard organization) has a policy of *"rough consensus and running code"*[70] that may be the reason of why they have much more implementations than SIP/SIMPLE;
- As stated above, it has more server/client/gateways implementations than SIP/SIMPLE – Therefore implementation of a system based on this protocol would require less effort;
- Wide server-side interoperability support with several other popular IM protocols through the use of gateways – Various implementations of these gateways, called transports, can be found in the Internet;
- XMPP was chosen to deliver Google's Gtalk[71] – being chosen by Google gave the protocol a big boost. However, rumours exist that in future versions of Gtalk, will support SIP and not only through the Jingle XMPP extensions (see chapter 2, XMPP section for details);
- The company that started the XMPP specifications, Jabber Inc.[72], was bought by one of the major network systems company, CISCO[73] – reaffirms that XMPP is here to stay;

Reasons not to choose XMPP:

- Added value to the IM world by helping developing applications and libraries related with the SIP/SIMPLE protocols;

SIP/SIMPLE – Pros and cons.

Reasons to choose SIP/SIMPLE:

- IST currently has a VoIP, SIP-based, infrastructure – Most SIP softphones have support for the basic SIMPLE IM functionalities. Therefore, deploying IM in the current infrastructure will probably bring automatically to the IM system most of the existing VoIP user base;
- It's relatively easy to deploy SIMPLE in existing SIP infrastructures – contributing to SIMPLE by enhancing and developing open source SIMPLE-related key software, helps opening the IM horizons to the already deployed SIP systems worldwide – contributing to the widespread use of the IM service;

[70] As stated in XMPP's website: http://blog.xmpp.org/index.php/2009/04/xmpporg-and-jabberorg-rough-consensus-and-running-code/;
[71] Product website at: http://www.google.com/talk/
[72] Jabber Inc. old website can still be found at: http://www.jabber.com
[73] More information about the acquisition can be found in: http://www.cisco.com/web/about/ac49/ac0/ac1/ac258/JabberInc.html

- IP Multimedia Subsystem (IMS) systems also adopted one of the core SIMPLE protocols, MSRP, [30]. For that reason, IMS has also something to gain with the development of open-source implementations of MSRP [32];
- Desktop sharing possibilities using SIP combined with MSRP/RTP together with a desktop sharing protocol like VNC's RFB[74].[75] –This feature is of great value to an IM application. Sip-Communicator IM client has support for this feature in its roadmap[76].

Reasons not to choose SIP/SIMPLE:

- There aren't as many implementations of clients that use this specification and all of its features as there are for XMPP – This makes it harder and more troublesome to implement a system based on this protocol. On the other hand, developing open-source implementations of protocols/features of SIMPLE contributes more to the IM world than it would choosing XMPP – Because choosing XMPP it's not expected to require as much development of software as it should be done in case SIMPLE is chosen.

Tendencies of IM

This subsection gives a view of the general tendencies of the IM world with a focus on the XMPP and SIP/SIMPLE protocols. It also exposes an upcoming technology and product that promises to change the way the communication in the internet is made by mixing the IM and e-mail concepts.

Interoperability tendency between XMPP and SIP/SIMPLE

An unified communication environment can be gained by seamless interoperation[77] of these two technologies. Both organizations behind the standardization of the protocols have written standards for interoperation. Jingle [14] and the *'Basic Messaging and Presence Interworking between the Extensible Messaging and Presence Protocol (XMPP) and Session Initiation Protocol (SIP) for Instant Messaging and Presence Leveraging Extensions (SIMPLE)'* draft [34] are examples of such efforts at a specification level. On a more practical level, there also exist some evidences, like the XMPP modules in OpenSIPS and Kamailio.

Communication revolution – Google Wave – merging IM and e-mail concepts

Google's Wave product was presented[78] as a product that will change the way communication is made via Internet. This might be not be an overstatement, given that:

- The presented preview of Google Wave is indeed promising – Google Wave is a mixture between IM and e-mail. It is a new concept and provides a way for several users to communicate both in real-time and offline;

[74] RFB – Remote framebuffer protocol – used by the VNC system to transfer the data needed to provide desktop sharing.
[75] An open-source proof of concept and technical report of this feature already exists [35].
[76] Sip-Communicator's roadmap is available at: http://www.sip-communicator.org/index.php/Development/Roadmap
[77] As stated in http://www.asipto.com/index.php/simple-xmpp-developer-workshop-2008/
[78] More information about Google Wave including the referenced presentation, are available at: http://wave.google.com

- The proven capability of Google to deliver innovation to the IT networks and services that changes the user's experience (e.g. Gmail);
- Federation possibility and efforts (i.e. Wave isn't only a centralized service, federation gives the possibility to have several wave per organization interoperate. As it occurs in XMPP and SIP/SIMPLE protocols) – Google is providing efforts to make federation possible with Google Wave[79]. These efforts are not only at a specification level but currently also translate to concrete software. Google open sourced the core algorithm of Wave (Operation Transformation's implementation) and provided also an open source basic implementation of a Google Wave client and server;

For all of the reasons given above, I believe that Google Wave, as advertised, might be a serious competitor to both IM & E-mail. Currently, Google Wave is a reality that is available only to a small number of users. In relation to the IM services, if the technology is as disruptive as announced, it might be a threat to the usefulness the IM systems. However, in my opinion, SIP infrastructures and general IM infrastructures will last at least for some time. There are simply too many already deployed IM infrastructures with a big number of users associated that lead me to believe that the scenario won't change completely overnight.

Blink! A promise of a full-featured SIP/SIMPLE client

Recently there has been an announcement of a SIP/SIMPLE client, called Blink, which has the following features:

- Wide-band voice over IP;
- High definition Video calls;
- Session based (with MSRP) Instant Messaging (IM);
- File transfer and multi-party chat sessions negotiated via SIP and that use the MSRP protocol and its relay extension;
- Desktop sharing using VNC's RFB protocol over MSRP;
- Publication and subscription for rich presence information such as: availability, moods, activities and geo-location;
- Management for the presence rules, resource lists, and RLS services documents using XCAP protocol;
- Multi-Party conferencing;
- Strong identity and security mechanisms;

Blink[80] isn't currently available, but its release is planned to be done soon[81].
This client suffers only the big disadvantage of not supporting client side interoperability with other protocols. The level of development and feature exploitation that this client has of a

[79] Webpage of the effort to promote Wave's federation: http://www.waveprotocol.org/
[80] Blink project's webpage: http://icanblink.com/
[81] As advertised by the one of the main developers in IETG's SIMPLE-WG mailing list: http://www.ietf.org/mail-archive/web/simple/current/msg08511.html

SIP/SIMPLE set of specifications, could justify adding interoperability with other IM protocols in the server side with the addition of XMPP gateways to the IM system.

SIP-Communicator (SC) features and potential

SC, as a multi-protocol, cross-platform IM client. Therefore, I believe that it has the potential to be widely used and adopted. Many people have more than one IM account. Managing and using each one with a different client application can become unpractical and deteriorate user experience. I perceive SC's potential as coming mostly from its unifying and user-friendly already deployed capabilities and goals.

Currently, SC supports most of the IM protocols available. However, at the moment, it doesn't support some of the more basic and important features of some important protocols, such as file transfer support. I believe this is one of the main reasons why a user would still prefer to have multiple client applications than to use SC. However, SC's current roadmap[82] features are very promising, and I believe that when accomplished, the number of users of this application will increase dramatically. From the full list of road mapped features, the ones which I found most notable and important to this work are:

1. File transfer through:
    - XMPP;
    - Yahoo! Messenger;
    - ICQ;
    - AIM;
    - MSN;

    To note that these capabilities are almost fully implemented and available at the project's SVN repository[83].
2. Integration of LDAP directories – This allows the user to make queries to the LDAP directory through the SC's interface. It would permit a user to search for details and IM addresses for any person in the LDAP directory. Note, a proof-of-concept and working implementation of LDAP directory search support is already done and available at the SC's SVN LDAP branch[84]. However, integrating that feature into the main branch is yet to be done.
3. Conference chatting for:
    - IRC;
    - ICQ;
    - AIM;
    - MSN;
    - Yahoo! Messenger;
    - XMPP

[82] http://www.sip-communicator.org/index.php/Development/Roadmap
[83] Information about accessing the SVN repository is available at: http://www.sip-communicator.org/index.php/Development/VersionControl
[84] SVN repository link: https://sip-communicator.dev.java.net/svn/sip-communicator/branches/ldap

Currently, the conference chatting is partially implemented, but unstable.

4. Desktop Sharing with MSRP and the MSRP Relay Extensions through SIP and VNC over MSRP – This feature is expected to be a very valuable one, especially in the IST's community as it could allow:
    - A very useful tool for students to collaborate to projects etc.;
    - A useful tool for teaching in IT related practical classes;
    - An alternative way to provide IT support by IST's helpdesk support personnel;

   This feature will take more time to implement and is strongly dependent on the availability of a MSRP peer library. More details about the library's importance can be found below.

When the previous features are achieved, especially if they are achieved in the estimated timeframes, SC will, in my opinion, represent a very feasible and wanted alternative as an IM client. Specially for people that have more than one IM account.

The MSRP protocol and relay extensions general importance.

As stated in the subsection '*SIP/SIMPLE'* of section 2.1.2, the MSRP protocol plays a key role in the SIP/SIMPLE protocol. Besides of the important role through the features that enables in the SIP/SIMPLE protocol, other architectures and network applications can benefit from it. MSRP plays an important role in the IP Multimedia Subsystem (IMS) architecture [30] [31]. It provides for:

- Conferencing;
- Instant Messaging;

It can also be useful to any network application that:

- Needs a protocol to reliably transmit content to other peers;
- Needs a protocol that is be able to transmit content behind NAT – MSRP currently does this through the use of relays, but specifications exist for other alternatives which don't require relays[85];
- Has content to be transferred in an efficient manner regarding the use of TCP connections – The specifications promote connection reusing;
- At different points in time, has several contents to be transferred through one connection to the other peer – MSRP can easily do this by associating each content with the concept of a message in the MSRP protocol. The message transmission can be paused and resumed at will, and several messages can be transmitted through the same connection – This allows the application to control when each content is sent, and therefore allows the application to give periodic or custom connection timeshares to each type of content;

[85] E.g.: The 'An Alternative Connection Model for the Message Session Relay Protocol' available at: http://tools.ietf.org/html/draft-ietf-simple-msrp-acm-01.

- Needs to transmit content and receive feedback about its reception – MSRP has such mechanisms at the application level, in the form of transaction responses and REPORT requests;
- Need to transmit content with the use of some custom transmission policy to allow quality of service or bandwidth throttling – MSRP's mechanisms, through the message pause system, allow these kinds of policies to be implemented.

3.1.2 Solution, choices and architecture

The previous subsections and chapter, converge to the solution which is presented in detail in this subsection.

Regarding the solution, there were choices to be made regarding both the protocol to use and the client. The choices and reasons follow:

1. IM protocol – SIP/SIMPLE was chosen. **Reasons:**
    - Facilitates adoption of the IM system by the IST's users that already are using the SIP infrastructure. – As stated in Chapter 1, most SIP softphones already support IMing. The current domain (voip.ist.utl.pt) can both be used for the VoIP and IM services;
    - Choosing a XMPP based system would probably prove to be less troublesome to implement, as more software is already implemented for this protocol. On the other hand, choosing SIP/SIMPLE and the required software developments, allowed me to give a more valuable contribution to the IM world and to the open source community. Seen that choosing SIP/SIMPLE, without developing other pieces of software could have lead to:
        - Worse interoperability with other IM protocols – As SIP/SIMPLE doesn't has the gateways (transports) software that XMPP has, it becomes a requirement to use a multiprotocol IM client;
        - An IM system that couldn't even provide file transfer capabilities – There were no clients at the time of start of the work, that had implemented file transfers over SIP/SIMPLE;
        - Less functionalities, as generally, there are more clients with more features supported for XMPP than for SIP/SIMPLE;

      In sum, an IM system with less features. Seen that his work aims to serve IST's community, this solution wasn't tolerable. To mitigate this problem the following decisions were made:
        - Choose a multiprotocol IM client – mitigates the interoperability problem of choosing SIP/SIMPLE;
        - Create the MSRP peer library and use it to enhance the chosen IM client – solves the file transfer problem as well as provides feature enhancement possibilities for the chosen IM client;

2. IM client – Sip-Communicator was chosen. **Reasons:**
    - Fairly numerous and valuable set of features that are currently implemented, namely:
        - Secure call support through the use of zRTP (one of the first clients that support this protocol).
        - Support for some of the most popular protocols:
        - Supports the Portuguese language for the interface – although not listed in the Roadmap, this feature is implemented;
        - Video and Voice calls through SIP signalling;
        - Support for the most basic functionalities of the protocols listed in the requirements available in section 2.2.;
        - Appealing, modern-looking minimalist and very functional GUI;
    - Very good features expected to be implemented in the not so far future, see *'SIP-Communicator (SC) features and potential'* above;
    - Out of the box interoperability support between IM protocols at the client side;
        - Interoperability at the client side is preferred over server side. Client side interoperability by definition represents a more distributed effort. Server-side will impose extra resource requirements from the server;
    - Has an active and dedicated group of developers[86]. Developer efforts have a tendency to grow as SC is listed in the Free Software Foundation (FSF) high priority list[87];

Fig. 3.1 depicts the structure of the adopted solution.

[86] Evidences of the communities activity can be found here: https://sip-communicator.dev.java.net/servlets/Search?scope=project&resultsPerPage=40&query=top+project&Button=Go

[87] Sip-communicator is listed under the '*Free software replacement for Skype'*. List available at: http://www.fsf.org/campaigns/priority.html

Architecture

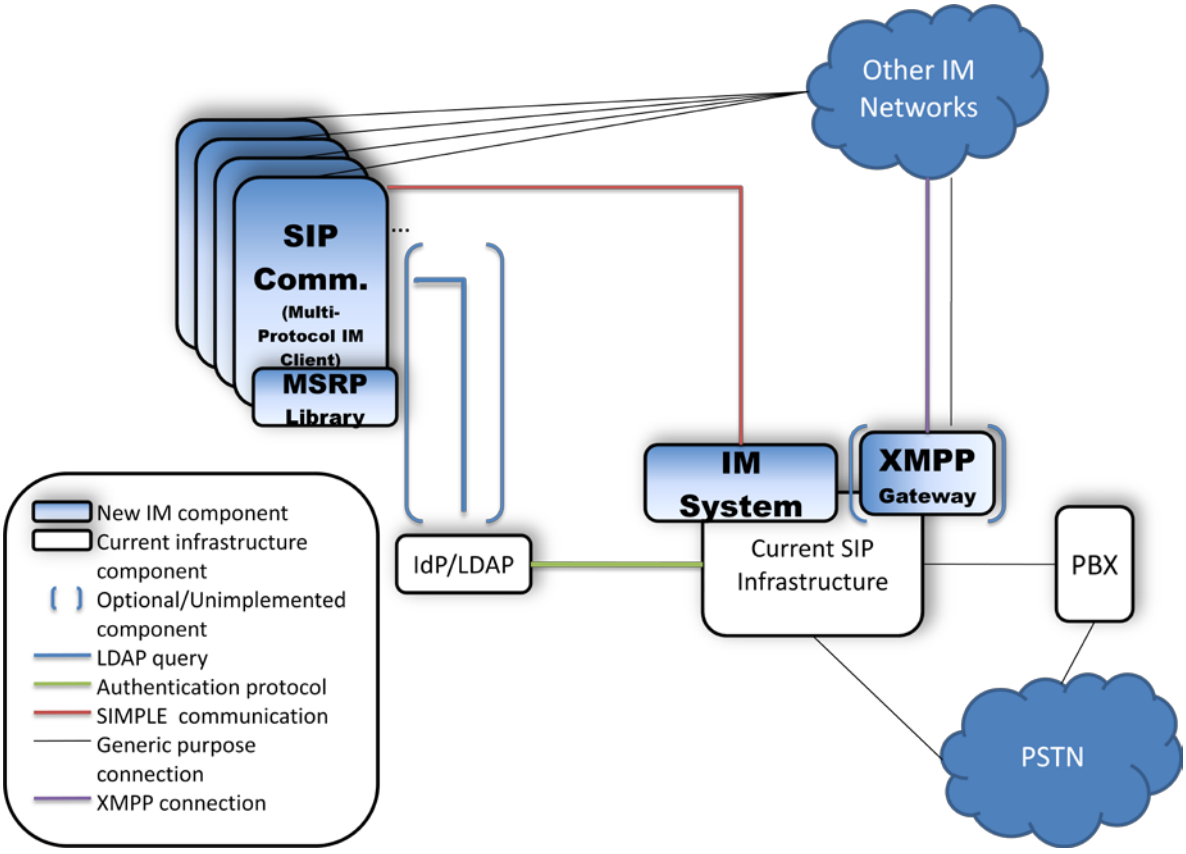

Fig. 3.1. IM system solution architecture.

Figure 3.1 also gives general details on how the IM system and current SIP infrastructure will be integrated. To note that there are two optional components, the XMPP gateway, and the direct LDAP connection between SC and IST's LDAP. This is because:

- The direct LDAP connection is probable to be a reality in the near future (see *'SIP-Communicator (SC) features and potential'* subsection on section 3.1.1 above).
- The XMPP gateway is always an option and can be deployed at anytime if seen as required (for the reasons stated in subsection '*Blink! A promise of a full-featured SIP/SIMPLE client',* and in section 3.1.2, above).

The solution's architecture is expected to be fully integrated with the current IST's VoIP services. Seen that the VoIP services were already at use, to avoid VoIP service disruption, a testing system was devised and implemented. This testing system is a transitory one that will converge to the final production system that is depicted in Fig. 3.1.

Testing system vs. Production system phase

In order to provide an independent platform for the tests without disrupting the VoIP services, a testing system was planned and implemented.
When planning the testing system, I took into consideration the following:

1. The production system must offer all of the previous VoIP features, plus the IM features;
2. The software application used at the VoIP infrastructure (Sip Express Router, SER) is outdated, and current software versions don't offer support for some IM functionalities like RLS;
3. Putting both services, IM and VoIP, under the same domain is desirable;
4. The testing system must not disrupt the VoIP services;
5. If the SIP server software in the current VoIP infrastructure is upgraded, the RADIUS service is no longer necessary. Therefore, the authentication process should be less resource consuming if an upgrade is made;
6. It's better to have a testing system that can easily be migrated to a production system with minimal downtime on the VoIP services;
7. IST's VoIP infrastructure is composed of two servers [36]:
    - VoIP-PRI – associated with the domain voip-pri.ist.utl.pt and responsible only for the connection between the public telephone switched network (PSTN) and the IP network – as stated in [36] this server was kept isolated due to security reasons;
    - VoIP – Responsible for all of the other VoIP infrastructure services;

With those facts in mind, the solution that I defined for the testing system was for it to clone the features and services that the VoIP machine has. Fig. 3.2, based in a figure from the *'Voice Over IP System in an Academic Environment'* [36], gives a visual description of how the testing system was deployed in the SIP infrastructure.

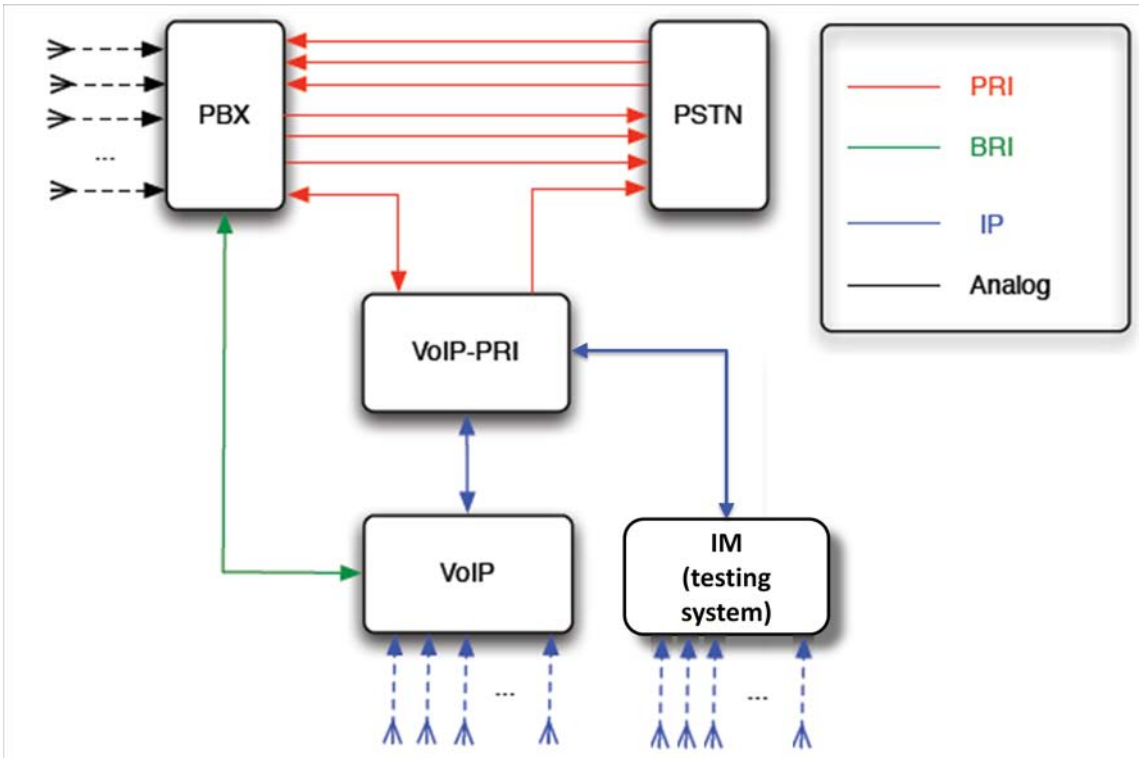

Fig. 3.2. Structure of the deployed testing IM system and current VoIP system.

In the production system, the IM server should replace the VoIP one. The BRI line, as stated in [36], isn't necessary. After exhaustive tests on the testing system are complete, placing it on a production phase can be achieved with minor configuration tweaks (both in the IM system and/or in the VoIP-PRI one). The chosen production system can join together all services available under the current VoIP and IM under one server, or it can separate them for performance's sake. Also, clustering, load balancing and redundancy can easily be implemented with the chosen software packages.

The following sections of this chapter provide the implementation details of each of the new non-optional implemented IM components (depicted and colored blue in Fig. 3.1).

3.2 Instant Messaging (IM) System

The IM system represents the core of this work. Although the MSRP peer library and SC with file transfer implementations were more time consuming, this is the most important part of the work. This IM system represents a state-of-the-art in instant messaging over SIP. Its architecture is worthy to be divulged on the SIMPLE IETF working group mailing lists, since such systems are rare (only sip2sip.info[88] currently provides a similar architecture).
In this section, the IM system's implementation has full focus. Details on the chosen solution software components as well as the reasons of the choices are given in this section. Also, the reasons and changes on the previously existing SIP infrastructure are exposed.

3.2.1 Discussion of components

Chosen solution

Due to the lack of support of the session mode IM by most SIMPLE IM clients (as stated in *'A word on SIP/SIMPLE adoption'* section 2.1.2), the IM system was configured to support page mode by routing and handling appropriately the SIP MESSAGE requests. The recommendation[89] that all of the outbound MESSAGE requests that pass through the SIP proxy should require authentication, was followed and implemented..

The current software application being used as a SIP Registrar and proxy server in the VoIP infrastructure is the Sip Express Router (SER). SER development seems abandoned, as the most recent version released of SER dates from January of 1996. Therefore, it has been losing some terrain in the number of implemented features, to other open source implementations as Kamailio[90] and OpenSips[91]:

- The SER currently doesn't support direct connections to a LDAP server – The current VoIP infrastructure [36] use an intermediary RADIUS server to surpass this limitation. Besides using LDAP to provide authentication, LDAP can also be used to gather some

[88] AG Projects Sip2Sip.info is available at: http://sip2sip.info
[89] Recommendation encountered in section 11 of the MESSAGE's RFC 3428 [37].
[90] Kamailio – Project website: http://www.kamailio.org/
[91] OpenSIPS – Project website: http://opensips.org/

other service enhancing (e.g. the multiple identification of the user can be retrieved and registered as SIP aliases. For instance, the user ist154457@ist.utl.pt can also have automatically its alias joao.a.p.antunes@ist.utl.pt registered in the SIP server);

- The SER doesn't offer XMPP server side gateway features;

Both functionalities stated above are offered by Kamailio and OpenSips.
After inspecting Kamailio's history pages, it became clear that some core developers abandoned the SER project to join the OpenSer (that lately forked into OpenSIPS and Kamailio). Due to these reasons, I consider the SER application outdated. Having gathered these facts, it became clear to me that the solution was a choice between OpenSIPS or Kamailio.

OpenSips vs. Kamailio

There aren't many differences between these two applications. Both originated from OpenSER. Apparently, they became two distinct projects due to fundamental divergences between developers.
The chosen brand ended up to be OpenSips 1.5. The only reason to choose OpenSIPS over Kamailio is that OpenSIPS provides out of the box integration with MSRP Relay software (details about implementation of this service can be found below in *'Optional, unimplemented/partially implemented components'*). To note that due to the similarities of the configuration syntax and modules, a migration to Kamailio can be easily attained.

Other components

The IM system also has other software components and applications that enrich the IM and presence features. The components and associated features are:

- MediaProxy[92] – This component acts as a media relay for RTP/RTCP and generic UDP streams. It allows communication between SIP user agents that are behind a NAT mechanism. The RTPProxy software could also have been chosen. No special reason to use MediaProxy instead of RTPProxy exists. To note that replacing MediaProxy with RTPProxy can be attained very easily. As feature-wise, the applications are similar, the main criteria for the choice should be the scalability of the software.

Optional, unimplemented/partially implemented components

As stated in section 3.1.1, there is a tendency to provide interoperability between XMPP and SIP/SIMPLE. The practical result of this tendency is a module that is available in OpenSIPS (and in Kamailio), that allows an easy deployment of a XMPP to SIP/SIMPLE gateway in the server side. There are apparently no differences whatsoever between the module available in OpenSIPS and in Kamailio. Therefore, what is stated here applies to both of them at the time of writing of this thesis.
Currently, there are two modes of operation for this gateway:

[92] Project webpage available at: http://mediaproxy.ag-projects.com/

- Component mode – Requires deployment of a XMPP server;
- Server mode – Has a XMPP server embedded in the module. Currently this embedded server has some limitations and is cited as being in a beta stage;

The gateway operates by translating SIP addresses and messages into XMPP ones and vice-versa (see Fig. 2.3 in chapter 2). This component wasn't implemented, as the chosen client is SIP-Communicator, which provides a XMPP interoperability at a client side. However, if in the near future, XMPP server side interoperability is found required, it can be easily implemented with OpenSIPS. An example scenario where server side interoperability with XMPP might be considered valuable is:

- A more comprehensive full featured SIP/SIMPLE client emerges, as for instance with the upcoming of Blink, see section 3.1.1. This client doesn't support other protocols other than SIP/SIMPLE. Then, a SIP/SIMPLE to XMPP gateway may prove to be valuable as it provides to the IM system:
    - Server side interoperability with XMPP;
    - Server side interoperability with other protocols through the use of XMPP gateways (XMPP transports, more details about the transports can be found in 2.1.2);

In the testing IM system, the following components:

- OpenXCAP – Integrates with OpenSIPS to provide XCAP capabilities to the IM system (see *'SIP/SIMPLE'* in section 2.1.2) specifically: buddy lists and miscellaneous policies.
- MSRP Relay – Provides MSRP relay capabilities to the IM server, therefore providing users a way to surpass the MSRP connection difficulties created by the NAT mechanism (see '*MSRP's role in the SIMPLE architecture'* in section 2.1.2).

These services require authentication from the user to be used. They are partially implemented. i.e. they are configured and running but the services they offer aren't available to the IST community because they have no connection to the LDAP to authenticate the users. As stated in section 5.2, LDAP authentication should be implemented in these services. Also, currently, no free to use SIP/SIMPLE IM clients that I know implement the functionalities offered by this two components.

Changes in the existing SIP infrastructure

The IM production system (see section 3.1.2) can also be used to provide an update to the currently used software in the VoIP infrastructure. This update, if done, won't change any of the features of the current VoIP services. The changes would be:

- Removing the need to have a RADIUS service in the middle to provide authentication (as it is required and implemented in the current SIP infrastructure [36]). Therefore, support for direct LDAP authentication – Seen that the OpenSIPS has a module to communicate directly with a LDAP service, it makes no sense to keep the RADIUS intermediary service. Also, the current VoIP infrastructure applies a patch to the SER's

source code in order to work with the RADIUS intermediary service. Such trick wouldn't be necessary anymore with the upgrade – more details on the tricks can be obtained in below in the subsection Authentication via LDAP;

- General software version upgrade – All of the software has a chance to be upgraded, with the typical inherent advantages associated (e.g. security improvements, optimization improvements, feature improvements, bug removal, etc.);
- Simpler infrastructure maintenance – The software upgrades weren't as simple to make in the existing SIP infrastructure. This happened due to the patch applied in the source code of some of SER's modules;

Authentication via LDAP

The SIP authentication currently used by the VoIP infrastructure, requires an MD5 based hash. The current SIP infrastructure authenticates its clients via the already existing LDAP MD5 hash attribute in the LDAP directory. However, it does not authenticate the clients directly, a RADIUS service between the SIP and LDAP servers is required [36]. This requirement imposed a change in the source code of the SIP server used, SER (for more details and reasons, consult [36]).

By using OpenSIPS, which has a module that provides direct LDAP connection features, the RADIUS service isn't required anymore. Also, as a way of speeding the authentication process, a memory cache is used. The new authentication's pseudo-algorithm is:

1. *Is the password available in memory? Yes, continue to 2. No, step to 3;*
2. *Does the authentication from the request match the found password hash? Yes, authenticated. No, if because the password is invalid, make a LDAP query and try again (maybe the user changed the password recently) if not working resend authentication challenge and exit with unauthenticated result. If working, jump to 6;*
3. *Perform LDAP query for the password hash based on the username; Was the query successful? Yes, continue to 4. No, reply with appropriate response code based on the type of error that occurred and exit as unauthenticated.*
4. *Strip the 'utilizadores@proxy.ist.utl.pt:' from the beginning of the password hash attribute. Was this a successful operation? Yes, continue to 5. No, reply with appropriate response code based on the type of error that occurred and exit as unauthenticated.*
5. *Does the authentication from the request match the found password hash? Yes, authenticated, continue to 6. No, send authentication challenge and exit with unauthenticated result.*
6. *Store the username and password hash pair in memory with a lifetime of 20 minutes*[93]*.*

[93] Meaning that if that memory values aren't retrieved for more than 20 minutes, the credentials are purged from the memory.

The use of the memory cache isn't required. However, it improves scalability by saving in the number of necessary queries. More details about measuring these improvements can be found in section 4.3.
Credential cache lifetime can be augmented without any noteworthy costs.
The LDAP direct authentication raises an issue that has however a straightforward solution. The issue consists in the fact that currently, only the IST users exist in the LDAP directory. The VoIP phone extensions present in the IST's telephone infrastructure do not have a representation in the LDAP directory. In the current VoIP infrastructure, this issue was solved through inserting such credentials for such devices in the RADIUS intermediary service [36]. The testing version of the IM system doesn't yet address this issue. However, solving this issue requires only:

- The addition of one entry for each VoIP phone device with a password MD5 hash attribute –the OpenSIPS authentication algorithm can easily be altered to support querying for these devices as well. These entries don't even need to be in the same tree of directory in which the IST users are.

3.3 Message Session Relay Protocol (MSRP) Peer Library

This section presents an overview of the open source peer library [38] implemented in Java, of the Message Session Relay Protocol (MSRP) [26] that was developed from the ground up.

I aimed for the library to be flexible enough to allow be used by all kind of applications independently of their final usage and programming language. As previously stated in sub-section 3.1.1, complete development of such a library is expected by to foster the SIP/SIMPLE IM system as well as other applications that can take advantage of the MSRP protocol features. One of the interesting features, is that the MSRP protocol allows fair usage policy implementation. Fair usage amongst various MSRP sessions in one peer depends greatly on the design of the library. This library aims to provide it amongst other features.

The open source nature of the library promotes interaction with the end-user application developers. This kind of interaction, that already took place and lead to some improvements, helps to identify new use cases, bugs, provide new design requirements, and new priorities.

This section is divided in the following manner:

- Subsection 3.3.1 provides a brief enumeration and explanation of use cases that originated the requirements and features that the library currently implements and is expected to implement in the future;
- Subsection 3.3.2 specifies the features of the library. Both the ones that are already implemented at the time of writing of this document, as well as the features that will be implemented in the near future under the effort sponsored by NINet Foundation[94];

[94] Most of the work related with the development of the MSRP library is funded by NINet [62]

- Subsection 3.3.3 gives a brief word on the development methodology used to develop this library. This overview will also serve as an introduction for the results and discussions regarding the library present in section 4.1. Information on contributions and on how the contribution system currently works, is also detailed in this subsection;

3.3.1 Use cases and requirements

The library's architecture was designed from a list of requirements that I extracted mostly from a set of use cases. The requirements along with the use cases are presented:

A. General Requirements

The list of the most notable general requirements and their main reasons are:

1. Support not only for the features listed in the main MSRP standard [26] but also for the relay extensions [27] – Support for relays improves the library's usefulness of network endpoints operating behind NAT;
2. Respect the norms of the requests for comments (RFCs) involved [26] [27];
3. Ability to make an informed decision about accepting or denying a message when it arrives to a peer. The API should provide access to all of the related message and session header values;
4. Method to notify the application upon receiving a complete MSRP message – This feature is essential to provide a message level of abstraction to the application that uses the library;
5. Providing a method in the API to warn the implementing application when a previously accepted message got aborted – Essential to provide the message level of abstraction;
6. Ability for an application to abort messages being sent and received – Essential to provide the message level of abstraction;
7. Notifying the application upon receipt of a REPORT request – Though sometimes the REPORT requests can be considered to be useful internally, providing a way for the library users to have access to them enhances flexibility of the library;

B. Use Cases

The rest of the essential requirements were devised from a set of foreseen use cases which are briefly described here. The list of brief use cases and related requirements follows:

1) Using MSRP for file transfers

Being a generic data transfer protocol, MSRP allows file transfers to be made between peers. Specifications written by the Internet Engineering Task Force (IETF) are available. They specify how the Session Description Protocol (SDP) [18] and MSRP can be used together to transfer files [39].

The extracted requirements and their main reasons are listed below:

8. Allow the application developer to define exactly how often and how many success acknowledgements of correctly received parts of a message (success REPORTs)

should be sent to the sending peer (peer that originated the SEND request). While having a default behavior for applications that do not need to define this – **Requirement reasons:** Giving the application's end user the visual perception of what is happening is part of the good practices of building good (usability-wise) interfaces. This is one of the features that allows accurate progress bars for the file transfers. The MSRP protocol standards specify that peers must be able to acknowledge partial reception of the message being received (through REPORT requests) but it lets to the developer of the library the decision with which granularity they should occur. This requirement provides both a default behaviour and makes it easy for the programmer to fully customise and change the libraries behaviour regarding this feature;

9. Support for the library to notify the application on the already "sent" i.e., dispatched to the operating system, data bytes – **Requirement reasons:** The acknowledgements described in the requirement above (#8) allow the library to inform the receiving peer conveniently about the progress of the file transfer. However, a file transfer peer that sends the file, may not encounter receiving peers that are defined to send REPORT acknowledgements as often as needed. It is also possible that those acknowledgments are not even requested by the sending peer (Success-Report header of SEND request set to no). Because of these possible scenarios, this mechanism was created. More succinctly, this feature allows the use of progress bars even when REPORTs aren't sent with enough granularity or aren't sent at all.
10. Support for the developer to define how often the application should be notified of the "sent" data bytes. For flexibility's sake, it is also important to have a default behaviour for applications to whom these details are unimportant – **Requirement reasons:** Without this customization, the library would lose its general purpose goal. And without the default behaviour, it would be unnecessarily harder to be used by applications that should have no need to worry with this detail.
11. Giving the library the ability to transfer files makes it essential that it can:
    a. Read messages directly from files;
    b. Write messages directly to files;

2) Instant Messaging

MSRP can be used to provide instant messaging between peers. MSRP characteristics, i.e.: the bidirectionality of the protocol; the fact that it can carry data that is encoded in various different wrappers, among them the Common Presence and Instant Messaging format (CPIM) [40]; along with other features, make it a good choice to be used to implement chat sessions. In fact, as stated in section 2.1.2, SIMPLE makes use of the MSRP and CPIM specifications to enable session mode instant messaging. The extracted requirements from the instant messaging use case are:

12. Validation mechanism for the data content wrapping at the library level – **Requirement reasons:** MSRP accepts any kind of content wrappers to transport the data of the message. CPIM, as defined in MSRP's specifications [26], must be accepted by every

MSRP peer. CPIM specifications, as most wrappers, can have illegal combinations (wrapping errors occur, either by implementation algorithm errors of the wrapper, or by content tampering of some sort). MSRP defines response codes that should be given when a wrapping error occurs. Seen that the CPIM format must be fully supported, developing a modular and well documented validator at the library level, is expected to allow an easier use of the library with other wrappers. By validating the data before sending it, it is also expected to diminish or even fully avoid errors related with the wrapping of data. Also, there might be performance benefits in doing the validation at a library level;

13. Support for allowing input/output from/to Java data streams – Mostly due to the bidirectionality of the chat sessions, data streams are more suitable for the internals of a chat session than a message in a file or in a memory variable;

3) Multiple MSRP Sessions in one Peer with fair usage

The scenario envisaged for this use case are simultaneous MSRP sessions and messages with diverse purposes on one single peer.

The MSRP protocol allows one peer to have simultaneous sessions and messages with all kinds of sizes. MSRP's connection model and session identifier concepts help preventing that MSRP relays and peers run out of available TCP ports. In order to provide a fair usage of the protocol, the messages can be split and paused. Although MSRP allows for various sessions to coexist in one peer and even in one TCP connection through these mechanisms, fair usage is not inherent to the protocol. Whether MSRP actually provides fair usage or not depends on the implementation.

The requirements for multiple MSRP sessions with fair usage, are the following:

14. Prioritization and scheduling of the sessions that share a connection – **Requirement reasons:** The only way to provide fair usage for connections;
15. Prioritization and scheduling of the connections that share a network link – **Requirement reasons:** This is the way found to allow bandwidth throttling for the MSRP over the network link;
16. Prioritization of the messages – **Requirement reasons:** This prioritization mechanism intends to provide fair/custom usage, based on the concept that not all the messages have the same priority;
17. Possibility to customize the scheduler and priority weight mechanisms of requirements #14-#16 – **Requirement reasons:** Different uses might have the need to have different schedule algorithms or more simply different weights for each priority value.
18. Creation of a default deficit round robin scheduler with three levels of priority – **Requirement reasons:** For simplicity and flexibility's sake, it is important that applications that do not have the need for custom scheduling mechanisms don't have to forcefully implement them in order to use the fair usage feature.

4) Application diversity – diverse goals and diverse programming languages

This use case is driven by the goal of wanting the library to be as useful as possible. Therefore, it should be prepared to be used by different types of applications. The extracted requirements for this use case are:

19. Support for multiple logging systems – **Requirement reasons:** There are already numerous Java logging mechanisms. Being flexibility a goal of the library, having a library seamlessly integrate with the application's logging system, whatever the system is, is definitely a wanted feature;
20. Ability to be used by non-Java applications – **Requirement reasons:** This is an ambitious requirement, and intends to extend the library to other programming languages. The main reason is that Java is everywhere, and is supported on almost every platform. Therefore, one can use the Java's compatibility with most platforms and preserve the core protocol machinery and implement only an interface of communication with applications that are not written in Java. Therefore allowing applications written in other languages to use the library. This requirement will only make sense in a scenario where the library's functionalities are fully implemented and other programming languages still have no open source libraries that provide MSRP support;

3.3.2 Features

The MSRP protocol is a fairly recent one. Currently, as stated in section 2.1.2, there are few open source implementations of it. Also, none of the found implementations of applications/libraries of the MSRP protocol were found to be intended to be as generic as this one. Some recent manifestations of interest by employees of different companies make me think that the scenario of few implementations of the MSRP will tend to change rapidly (most of the users that requested access to the SVN repository[95], which contains the library's source code, had the either the intent to assert if it can be used on their applications, or hoped that it can serve as a reference implementation to test their implementations).

The MSRP peer library is an ongoing effort, and is currently under development. Therefore, this section is divided in two sub-sections: one that lists the current available features; and the other that lists the future features. Currently, development efforts are sponsored by the NlNet foundation.

Currently available features:

- Basic, non-TLS, MSRP peer functionality support (RFC 4975 [26]);
- MSRP message level of abstraction (i.e. the application doesn't have to care with transactions but only with messages) which allows:

[95] The MSRP's source code is available in a SVN repository (available at https://msrp.dev.java.net/). Access through the SVN protocol is dependent on approval by the project owner, me. Therefore I can have a better knowledge of how many people use the code and I asked to each of the users that requested approval why they wanted it. To note that the code can still be obtained through a web interface available for surfing the repository content, but this method is much slower and unpractical for other purposes other than satisfying punctual curiosity about a content of a file.

    - Accepting and denying MSRP messages;
    - Notification of aborted message events;
    - Notification of sent bytes;
    - Notification when REPORTs are received – provides a way for the API to notify the application when the receiving peer successfully acknowledges the receipt of message parts (Success REPORTs). Also notifies the application when a Failure REPORT is received;
- Provides full customisation of the granularity of the sent bytes notifications, as well as of the successfully received parts of message notifications (success reports);
- Data storage abstraction layer – Provides a common set of methods for reading and writing data independently of the used medium. Also provides an easy way to extend the support to other kind of mediums. Mediums currently supported:
    - Files;
    - Memory;
- Support for all the major Java logging systems[96]:
    - JDK 1.4 logging;
    - Log4j version 1.2;
    - Simple logging – to the standard error output;
    - Jakarta Commons Logging;
    - Logback;
- Well documented library (Javadoc and in-code comments) as well as some example classes[97]– some of them requested by fellow developers interested in using the library;

Features to be implemented:

- Support of TLS;
- Relay extensions [27] support;
- Enhancement of the library's robustness and specification compliance;
- Transaction level interface – which allows the application that uses the library a control over each of the MSRP transactions and their responses (e.g. like the one that the JAIN-SIP provides);
- Prioritization and fair usage of connections/sessions support;
- MSRP message objects being extended beyond the session and outliving them – already semi implemented but tests are lacking because it isn't a critical feature for the current uses of the library;
- An even better performance – A lot of performance improvements have been made already. But still, like stated in the results section 4.1, some gains can still be achieved;

[96] This was possible through the use of the simple logging facade for java (SLF4J) more information available at: http://www.slf4j.org/

[97] Example classes are available under the msrp.examples Java package. Source code is available at the project's webpage (https://msrp.dev.java.net/).

- Extending and testing the library to be able to be used by the Android operating system (to which Sip-Communicator has already developed and concluded a working prototype);
- Extending the library to be used by other programming languages – solutions may come from the use of sockets to communicate internally between the API written in another language, and the core written in Java;
- Make clearer to the library users which classes they should use and which are off-limits and should only be used internally – Refactoring the source code and implementing some packages with interfaces should be the way to attain this feature;
- Data content wrapper validator;
- Library level support for the Common Presence and Instant Messaging (CPIM) [40] message format wrapper;

Part of the features are planned to be implemented until February 2010[98], work that is sponsored by the NlNet foundation [62].

3.3.3 Development methodology

The methodology that is applied in the development of the library is a test driven one (TDD – test driven development). Typically, JUnit tests are made before implementing the feature or correcting the bug.

This library was started and is currently maintained by me. It is an open source library, which had some minor contributions so far, and that is developed in a way that facilitates contributions. To facilitate contributions, the following development routines have been adopted:

- Before coding each method and class, the Javadoc comment is written;
- The TDD methodology helps to check if any contribution in the form of a source code alteration by others breaks any of the functionalities;
- Comments are treated with similar importance as the source code – they are updated when required, and extensively used;
- Adoption of code style formatting rules and coding conventions has been made;
- Making the code easy to understand – without having to sacrifice on performance as much as possible. When not obvious algorithms have to be used, an effort on documenting them well with extra documents and extra comments on the code is made[99];
- Applying object oriented development best practices to the best of my knowledge – there is an evolution that can easily be noted on the generated code. In the beginning, the code was more function-like than it is now. Some refactoring of the old code has been made, but still, examples of a not so object oriented code exist.

[98] Work sponsored by the NlNet foundation [62].
[99] As an example, we have the Counter class which algorithm has schemas available in the planning documents in the SVN repository

Contributions & contribution philosophy

No major contributions have been made so far. The most significant contributions so far have been: a report by e-mail of a wrong *if-statement*, which was found by Christophe Lucas; mail exchanges about the library some of which resulted in new ideas for features/enhancements/examples as well as finding some bugs.

Currently, the source code of the library is browsable by web. For someone to be able to make a more practical SVN checkout, they have to request the Observer role through the MSRP's peer library webpage [38]. This is an useful feature, as it fosters communication by allowing me to know how many and which users are interested in the library. Also, this allows me to ask to each user that requested the role what is their interest in the library.
Contributions by users are accepted through SVN patches at a first stage. If eventually this user gives enough contributions with good code, he will be given the possibility to commit its code directly to the library's official SVN repository without supervision. To foster communication and bug tracking, mailing lists and an Issue control system are available for the developers and interested users (links are available on the project's webpage [38]).

3.4 File transfer support via SIP in Sip-Communicator

The file transfer feature is available in every instant messaging protocol considered. The Sip-Communicator application, with the virtues already stated in section 3.1.1, is lacking support for file transfer over SIP. In my opinion, lack of such a feature makes it an unappealing choice to be used on the IM system. Efforts into building an open standard to allow SIP/SIMPLE systems to have file-transfer, had already been initiated by the IETF when I did my initial research to make the choices regarding the system. Therefore, it was possible for me to choose to implement file-transfer in SC guaranteeing interoperability. The standardisation efforts matured to its final RFC form [39] in May 2009.
Implementation of this feature was made easier by some contributions, in the next subsection, an overview of the relevant structure of Sip-Communicator and contributions related to this work are given.
Following the overview, the development methodology subsection explains in general terms the approach I took to implement the file transfer.

3.4.1 Overview of relevant structure and contributions

Sip-Communicator is a very extensible, modularized[100] and in my opinion well-designed Java application. It is based on the OSGi[101] (OSGi – The dynamic module system for Java) specifications.

[100] As stated in the project's about page (http://www.sip-communicator.org/index.php/Main/About), this application is very extensible and developer friendly.
[101] More information about the OSGi project, can be found on its webpage at: http://osgi.org/

The application is GUI based and therefore it makes extensive use of the Listener pattern.
The development of the file transfer feature, was done throughout most of the timeframe of the thesis work. The main reason for this was that it was being done simultaneously with the development and fine tuning of the MSRP library to be used to provide such a feature. During the time that it took to be developed, two GUIs were used: The first one, the proof-of-concept work of Anthonny Schmitt[102]; and the second and the last GUI is a more refined and better looking one, which was part of an effort taken by some of the usual contributors of SC to add file transfer for all the IM protocols (leaving the SIP/SIMPLE one up to me). From their effort, I benefited not only from the already made UI but also from the SLICK[103] generic file transfer tests. SLICK tests were made to be applied to all the IM protocols that had file transfer functionalities being developed. Still, some adaptation was required to make the generic tests work for my implementation, nevertheless, the job became easier and it also represented a good contribution.

3.4.2 Development methodology

The initial efforts of adding file transfer over SIP to SC, resulted in two working prototypes:

- The MSRP peer library prototype – with the bare minimum functionalities to be able to support file transfer;
- An SC prototype that used the aforementioned MSRP library prototype together with the first version of the GUI, to provide file transfer with partial specification compliance.

After these prototypes were developed, the priority became to further develop the MSRP library making it more reliable and efficient. As time went by, the specifications on how to provide file transfer in SIP, matured into an RFC [39].
As soon as the MSRP library was mature enough, efforts to integrate it with SC in order to achieve a fully compliant file transfer over SIP started.
First, I examined the specification's new version for changes. While implementing the file-transfer feature, it became clear that the MSRP library needed some more features. Therefore, the development of the file-transfer was an iterative process that required me to switch focus back and forth between the SC and the MSRP libraries code. Also, implementing this feature, required a better comprehension and study of several specifications: The SDP Offer/Answer mechanism [25], and some SIP related RFCs as well.
The results of these efforts are detailed in chapter 4.

[102] More details about his work are available at: http://gsoc08-e4.blogspot.com/
[103] SLICK stands for Service Leveraging Implementation Compatibility Kit and they are used in SC to test the implementation of an OSGi service. More information can be found here: http://www.sip-communicator.org/index.php/Documentation/HowToImplementProtocols9

Chapter 4

Results and discussion

This chapter presents the results of the Implementation efforts detailed in the previous chapter. Conclusions about the results are drawn, and provide basis for future work and more general conclusions.

This chapter, is composed of three main sections, one for each of the implementations carried. For each of the carried tests, the evaluation methodology is detailed, followed by presentation and discussions of the results.

Next, in the MSRP Peer library section, focus is given to the performance, memory efficiency, and functionality issues. Afterwards, in the the Sip-Communicator file transfer feature implementation section, feature tests and interoperability are the subjects of this section. To end the chapter, I present tests and discussions about performance, features, components, security, and future tests regarding the IM system deployed, the testing version.

4.1 MSRP Peer library

Evaluation methodology

I used a test-driven development (TDD) methodology to develop the library. Therefore, many of the corrected bugs and functionalities have an associated test.

TDD, besides being a methodology that *"encourages simple designs and inspires confidence"* (as stated in [41]) it was also chosen because of the open source philosophy associated with the MSRP library. The idea is to have a vast and solid base of tests, that allows a fast and automated way to check if a contribution breaks any of the previously existing functionalities. To note that as stated in section 3.3.3, no external contributions in the form of source code have been done so far. Also, this methodology doesn't fully guarantee that contributions can't damage the existing functionalities or add bugs. Therefore, as stated in section 3.3.3, the first contributions from fellow developers will be forced to be done through a patch, that will be inspected and validated prior to be committed to the version control system (the Subversion, SVN, was used for this project[104]).

MSRP peer library was evaluated in the following domains:

- Performance;
- Memory efficiency;
- Functionality;

The results in each domain are further detailed in this section. The tools used to make such evaluations were essentially JUnit tests together with support classes and a Java profiler, more details about the classes are available in table 4.1. The profiler used was the Java profiler Test

[104] The used system is hosted by dev.java.net (http://dev.java.net) Details on how to access the repository are available on the project's webpage [38]

and Performance Tools Platform (TPTP)[105]. Before each commit to the SVN is done, an automated library build and batch execution of tests cycle is executed. Table 4.1 gives a full list of the tests available in the library, their description and if they are included in the batch execution of tests.

Test class name	Functionalities tested/General test description	In batch?
MiscTests	Not a JUnit test. A simple Java class with tests made before adopting a TDD methodology. List of tests made by this class: • Java native URL/URI class tests; • Java native Socket class test; • Java native SocketChannel class test; • Test of the random ASCII character generator algorithm (used for generating a transaction ID); • Java Regex tests; • Simple MSRP message exchange test – outdated and non useful at the moment;	No
SimpleProfileTests	• Small 300KB message exchange test – created to use with profiler when the library had a big performance problem and testing with bigger messages was too time consuming; • TestSendingExistingFile – similar to the one above, but doesn't actually make any kind of validations such as making sure that the sent content is the same as the received. This test was designed to: o Be executed by the profiler to give execution and memory statistics; o Provide a test that could be run by the JUnit framework to allow to measure the time it took to send to use a file by directly using the value that JUnit reports when a test is finished;	No
TestAbortMechanism	• Tests the abortion mechanisms of an incoming and outgoing MSRP message;	Yes
TestCorrectlyBreaksSentData	• Tests that the behaviour specified in MSRP's specification [26] is executed. The specifications mandate that if the end of transaction characters	Yes

[105] More details on the TPTP software can be consulted on the project's website: http://www.eclipse.org/tptp/

	are detected in the body of a message, it isn't mistakenly perceived as the end of transaction. To do this, the message is paused and resumed before the special characters of the body are transmitted. These actions assure that a new transaction is made and therefore a new transaction id is generated, with this, the special characters can be safely transferred in the body of the new transaction, without being confused as the end of transaction;	
TestCounter	• Tests the Counter class that is basically used to account for received bytes of a MSRP message. The Counter class had two previous versions that weren't satisfactory in terms of memory and execution time efficiency. The third Counter version has a more complex algorithm[106], but is much more efficient. Due to the complexity of the algorithm, this test class was required;	Yes
TestRegexMSRPFactory	• Tests for all of the regular expression (regex) patterns found in the RegexMSRPFactory (currently incomplete). The RegexMSRPFactory should be, in the near future, responsible for providing all of the regular expressions used throughout the stack;	Yes
TestReportMechanism	• Tests that the default report mechanism reports on sent bytes as expected; • Tests that the default report mechanism acknowledges successfully received bytes as expected (by sending back MSRP success reports); • Replaces the default report mechanism with a custom one and asserts if it behaves as expected – therefore testing the customization of the report mechanism feature;	Yes
TestSendingASCIIMessages	• Tests sending several message sizes with ASCII content ranging from 300Kb to 20Mb, and to and	Yes

[106] The algorithm is a cluster-like one. Basically consists of an array. Each position of the array, contains a two number vector, were the first position has the starting position of the counted byte and the other the number of bytes accounted for starting of that position. More details can be found in the source code [38] and in the planning directory of the SVN repository;

	from different combinations of types of storage mediums (file and memory);	
TestSendingBinaryMessages	• Tests sending several message sizes with binary content ranging from 300Kb to 20Mb, and to and from different combinations of types of storage mediums (file and memory); • One of the tests is rigged with a sequence of bytes that used to mislead the stack's pre-parser algorithm;	Yes
TestSendingExistingFile	• Tests sending an existing file between two peers. This test allows testing with different file sizes by manually changing the file to be sent. • Done to test the performance of the library; • Validates that a correct transfer was made, by comparing the content of the sent and received files; • Also validates that some other basic callbacks to the API are made;	Yes
TestSendingSmallMessages	• Test used to correct a bug the stack had when sending small messages;	Yes
TestTransaction	• Tests some of the parser mechanisms in the Transaction class. More specifically: o A transaction with a complete/normal body; o Parsing a transaction without content;	Yes

Table 4.1. List of implemented test classes – With general description and information about batch test inclusion.

Different mixes between the tests and tools were used to test each of the domains. Specific methodology used to test each domain, along with the attained results and discussion are the subject of the next subsections.

Performance results

To perform these tests, the classes:

- TestSendingASCIIMessages;
- TestSendingBinaryMessages;
- TestSendingExistingFile;

were used. The TestSendingExistingFile ran with three different files, with different sizes. Note that TestSendingASCIIMessages and TestSendingBinaryMessages have similar compositions, just the content type changes. This allows me to assert the change of content in the

performance of the tests and prove that the stack is data agnostic. Table 4.2 provides a summary of the executed tests and their results.

Test name (method name)	TestSendingASCIIMessages class				TestSendingBinaryMessages class			
	Runs: (all values in ms)			Max-Min (ms)	(all values in ms)			Max-Min (ms)
	1st	2nd	3rd		1st	2nd	3rd	
300KB Memory to Memory	63	47	47	16	47	62	94	47
1MB Memory to Memory	63	63	63	0	78	63	78	15
5MB Memory to Memory	234	218	219	16	234	204	250	46
300KB File to Memory	32	31	31	1	16	31	2016	2000
5MB File to Memory	172	187	171	16	187	171	204	33
300KB File to File	172	156	156	16	15	31	31	16
20MB File to File	1156	3094	1016	2078	1125	1031	1125	94
300KB Memory to Memory with Success Report	15	16	16	1	2015	16	15	2000
1MB Memory to Memory with Success Report	31	2047	46	2016	31	63	62	32
5MB Memory to Memory with Success Report	172	156	2156	2000	172	188	203	31
300KB File to Memory with Success Report	31	16	15	16	16	31	31	15
5MB File to Memory with Success Report	187	171	171	16	187	203	203	16
300KB File to File with Success Report	31	31	32	1	47	15	31	32

File size (bytes)	TestSendingExistingFile (with success report)			
	Runs: (all values in ms)			Max-Min (ms)
	1st	2nd	3rd	
1219974	266	156	172	110
16742799	922	953	1047	125
243809310	46391	19188	13907	32484

Table 4.2. Performance test results. All tests ran on an Intel Core 2 Duo T7300 @ 2GHz with 2GB of RAM (LG e500 laptop).

In order to mitigate eventual cache mechanisms that could give misleading results, the following running sequence of tests was used:

1. 1st run of TestSendingBinaryMessages;
2. 1st run of TestSendingExistingFile with 1219974 bytes file;
3. 1st run of TestSendingASCIIMessages;
4. 1st run of of TestSendingExistingFile with 16742799 bytes file;
5. 2nd run of TestSendingBinaryMessages;
6. 1st run of TestSendingExistingFile with 243809310 bytes file;
7. 2nd run of TestSendingASCIIMessages;
8. 2nd run of TestSendingExistingFile with 1219974 bytes file;
9. 3rd run of TestSendingBinaryMessages;
10. 2nd run of of TestSendingExistingFile with 243809310 bytes file;
11. 3rd run of TestSendingASCIIMessages;
12. 3rd run of TestSendingExistingFile with 1219974 bytes file;
13. 2nd run of of TestSendingExistingFile with 16742799 bytes file;
14. 3rd run of TestSendingExistingFile with 243809310 bytes file;
15. 3rd run of TestSendingExistingFile with 243809310 bytes file;

To note that between each run of TestSendingExistingFile, the received file was eliminated.

Both the TestSendingASCIIMessages and TestSendingBinaryMessages tests generate random data to be sent. To note that all of the execution times exclude overhead times. I.e. the time used for generation of content to be sent (that isn't required in the TestSendingExistingFile class), and the miscellaneous validations that are done after the message is sent.

Execution statistics were gathered by running the TPTP profiler for the SimpleProfilerTests.testSendingExistingFiletoFile method, which gave the following noteworthy results:

1. The writing thread spends ~23,9% of the time in 8176 calls to countSentBodyBytes from which, ~13,8% are spent in 8176 calls to the shouldTriggerSentHook;
2. The writing thread spends ~13,04% of the time in 8178 calls to hasDataToSend;
3. The reading thread spends ~43,89% of the time in 16378 calls to countReceivedBodyBlock from which, ~22,18% are spent in 16378 calls to shouldGenerateReport;
4. As expected, the reading thread occupies a bigger percentage of time (~95,85%) than the writing thread (~57,30%).
5. The sending peer takes ~9,6% of the time to read the content from the file.
6. The receiving peer takes ~2,90% of the total time, to write the contents to a file.

Like stated in table 4.1, the used method differs from the TestSendingExistingFile class because it doesn't validate any functionality, it simply transfers the file. 100% of the time corresponds to the following activities:

- Sending peer – reading the data from the source;
- Sending peer – adding the extra headers and information required by the protocol and dispatching it to the operating system (OS);
- OS – transferring the data –seen that the same network endpoint is used, this should take virtually zero time;
- Receiving peer – reading the data and parsing it;
- Receiving peer – writing the data to the destination medium;

Plus some minor JUnit overheads[107].

Performance discussion

The achieved values are good enough for most use cases. However, better performance can be achieved if needed. More specifically, the following optimizations can be considered:

- Both methods countSentBodyBytes and countReceivedBodyBlock, listed in 1 and 3, are defined in the ReportMechanism abstract class. Methods shouldTriggerSentHook and shouldGenerateReport, are defined in the DefaultReportMechanism class, that is a concretization of the abstract ReportMechanism class. Therefore, it suggests that creation of a NullReportMechanism class (with dummy methods), in replacement of the DefaultReportMechanism, can reduce ~13,8% in the time taken by the sending peer, and ~22,18% for the receiving peer;
- Adding the possibility for an application to disable completely the ReportMechanism's features (not neglecting the specification mandatory Success-REPORT, if requested by the other peer, to be sent when a message is completely received) can reduce ~23,9% of the time spent by the sending peer, and ~43,89% for the receiving peer;
- Also regarding points 1 and 3, the abstract method getTriggerGranularity in the ReportMechanism class, defines how many body bytes are transferred between two consecutive calls to the countReceivedBodyBlock method. Making the countSentBodyBytes method being called in the same manner, i.e. based on a customisable granularity, and increasing the value for this granularity, reduces time spent (as less calls are made to the countReceivedBodyBlock and countSentBodyBytes methods);

The hasDataToSend method can also be optimized. Nevertheless, implementing an event driven restructuration on the way the data is sent removes the need for this method altogether.

[107] By comparing the cumulative time that took to run the testSendingExistingFileToFile method, with the cumulative time spent in the four threads (two per peer), we get a difference of less than 1s in the universe of a total cumulative time ~110s. Hence, the overhead is less than 1%, and therefore considered by me as irrelevant.

Variations of ~2000ms observed between runs of the same test (as noticed in runs of the TestSendingBinaryMessages and TestSendingASCIIMessages presented in table 4.2) are odd. The probable causes (listed from the most probable to the less probable) for these differences and solution directions for future work, are stated next:

- The 2000 ms number is a familiar one, as this is the number of ms the writing thread sleeps if not waked up by the need to transfer data. This variation may be related with a failure of waking up the writing thread whenever necessary[108]. – **Solution directions:**
  - Taking the 2000ms waiting period (i.e. making the writing thread wait indefinitely until it gets notified to wake up) and running the tests to see if they fail. This will allow me to determine if a notification to the writing thread is missing;
- With less probability, the computer was running other processes, the big variations can be because of other time consuming processes that were executed. e.g. demanding prolonged disk activities;
- The content of the message can be the cause. – **Solution directions:**
  - Look at the pre-parser algorithm and assert which kind(s) of pattern(s) would consume more resources. Count the time it takes to transmit X bytes of content with that pattern and X bytes of content without that pattern;
  - Run the tests and capture the network traffic. When a spike is detected, analyze the content and try to figure out if the content could be the cause;

To note that the observed variations can have more than one explanation. The big discrepancy of time (32484ms) found between runs of the TestSendingExistingFile with the file of 243809310 bytes (big file), leads me to believe that the cause might not be content related.

Memory efficiency results

The memory efficiency results were gathered using the profiler memory analysis feature of TPTP. The TPTP ran the SimpleProfileTests.testSendingExistingFileToFile method with the big file. The gathered memory statistics can be found in table 4.3.

PACKAGE NAME	Total Instances	Live Instances	Collected by G.C.	Total Size (bytes)	Active Size (bytes)
msrp.testutils	2	2	0	160	160
msrp.messages	2	2	0	136	136
msrp	34	31	3	1792	1688
org.slf4j.* (loggers)	16	16	0	256	256

Table 4.3. TPTP MSRP library memory statistics – collected in a run of SimpleProfileTests.testSendingExistingFileToFile with the 243809310 bytes file.

[108] This behaviour is part of the remains of the first MSRP prototype used, in which the writing thread wasn't notified to wake up, it simply woke up every 2000ms and only went to sleep when no more data was sent. This was restructured, but the 2000ms value was maintained.

Memory efficiency discussion

Results of memory consumption are quite satisfactory. Table 4.3 demonstrates that the MSRP peer library consumes a very limited amount of memory in a file to file transfer.

Functionality results and discussion

The library functionalities were tested mainly through the already described automated batch of tests, which ran successfully. The test batch composition is detailed in table 4.1. Library's main features description can be found in section 3.3.2.
Implemented tests allow me to have a good level of confidence on the working state of the library's functionalities. However, I do take into consideration the fact that some tests may not cover all aspects of the feature, and therefore I'm fully aware of the fact that some implemented features may have defects. The tests are not static immutable classes, as described above, library development is done using the TDD methodology. This methodology requires that also the tests are developed and enhanced along with the library's features.

A word on interoperability

As detailed in *'Current MSRP open implementations and libraries'*, that can be found on section 2.1.2, there aren't any mature open source and free to use MSRP implementations available. Based on my knowledge of the library's implementation, full interoperability probably hasn't yet been achieved. The main reason for this prediction are possible failures on the parser's logic and regular expressions used, that may not fully comply with the formal syntax of the specification. This fact made me enlist the *'Enhancement of the library's robustness and specification compliance',* in the list of features to implement that is detailed in section 3.3.2. This enhancement will be attained by a verification and validation (with tests) of all of the regular expressions and logic of the parsers[109].
These are the main reasons why interoperability wasn't tested yet.
For future interoperability tests, I'm currently considering making a simple program in Python, using the python-msrplib[110].

4.2 Sip-Communicator, adding the file transfer over SIP feature

Evaluation methodology

SC is a GUI based multiprotocol IM client. As stated in section 3.4.1, SC is based on the OSGi[111] system. SC architectural design (because of the OSGi and GUI systems) makes automated tests more difficult to implement. However, to help address this problem, SC has a package with the so called Service Leveraging Implementation Compatibility Kit (SLICK) tests.

[109] This feature is related with issues #9, #11, #16, and #17 whose details are available at https://msrp.dev.java.net/servlets/ProjectIssues
[110] As this seems the more mature library available. Library is available for download at: http://ag-projects.com/debian/pool/main/p/python-msrplib/
[111] More information about the OSGi project, can be found on its webpage at: http://osgi.org/

These SLICK tests are developed using JUnit. They are used to test the service classes found in the service Java packages (net.java.sip.communicator.service.*) of SC.
To provide automatic testing of the implemented file transfer functionality, a SLICK test was implemented. This SLICK test is based on the generic file transfer operation set SLICK test available in the package net.java.sip.communicator.slick.protocol.generic. The implemented test, validates correct behaviour in the following scenarios:

- A simple send and receive of a file;
- The sender of the file cancels the file transfer before the request is accepted by the receiver;
- The receiver declines the file transfer;
- The receiver cancels the file transfer while it was in progress;
- The sender cancels the file transfer while it was in progress;

As automated tests might not provide full guarantees on all of the functionalities behaviour, these test scenarios were also performed manually, by running two instances of SC.

Functionality results and discussions

Currently, execution of both the SLICK and the manual tests, ran without problems. Therefore the features is considered implemented.
There is however one limitation regarding the possible use case scenarios:

- **Limitation:** The MSRP peer library doesn't yet support the MSRP Relays extension. Also, SC doesn't have the necessary GUI(s) for configuring the relays.
  Problem because: Without the relays, the MSRP protocol can only be used if the sending endpoint can contact directly the receiving endpoint's IP, which can be a problem for endpoints behind NAT systems.
  Solution: Add the Relays extensions functionality in the future (that is planned to be done, see subsection *'Features to* implement' in section 3.3.2). Implement the GUI to configure the relay options – recent talks with main developers of SC, reveal that probably I won't need to do the GUI myself, as it may be done by, Yana Stamcheva, the main GUI designer of SC).

This limitation is planned to be solved in the future after the implementation of the Relay extensions to the MSRP library.

Regarding the specifications used to implement this feature [39], SC's implementation isn't fully compliant. These are mostly due to structural limitations of SC, and probably doesn't break compatibility with most clients that may have adopted the same specifications. More details on those limitations can be found in the next subsection about interoperability.

A word on interoperability

As stated in section 3.4, the file-transfer feature was implemented following the specifications available in the *'A Session Description Protocol (SDP) Offer/Answer Mechanism to Enable File Transfer'* [39]. The specifications make it possible for SC to transfer files with other clients.

Interoperability tests weren't carried as I found no other specification-compliant clients available[112] to test it.

The implementation of the file transfer functionality has some limitations and doesn't fully implement the specifications, however, I expect that limitations should provide a minor to null impact on interoperability due to their nature. Current limitations are:

- **Limitation: (1)** Due to the way the SIP Requests are handled, only one of the operation set classes can process a SIP request with its associated SDP body. Also, the good practices and the way SC is designed, suggests/mandates two different OperationSet classes for each different feature (e.g. video, voice, message, file-transfer etc.).

  Problem because:

  - Specifications ([18] [25]) allow the negotiation of several types of media sessions (e.g. video, voice, etc.) in one request. I call these requests mixed sessions (i.e. a request that has in its SDP body at least two distinct types of media sessions). Therefore, handling of these mixed sessions is impossible without major SC architecture redesign.

    Solution: File-transfer SDPs that have also other types of sessions are identified, and the SC acting as a SIP UAC (SIP user agent client, short name for a SIP client that makes the initial request in a SIP dialog, as defined in the SIP specifications [17]) replies with a response code of *'488 – Not Acceptable Here'* and a warning message of *'399 – Miscellaneous'* with a comment *'Mixed, file-transfer and other type, sessions not allowed'*.

    This is the most specification compliant behavior that could be implemented without the major architecture redesign. To note that the fact that mixed type sessions isn't allowed, is not a serious limitation, as it was told me by Emil Ivov, the creator of the SC project, that his empirically knowledge about SIP clients, make him conclude that they usually don't use the same SIP dialog/request to negotiate different unrelated media sessions. Mixed sessions in one SIP request are usually even more difficult to manage, so, as stated by Emil Ivov, most implementations don't use them.

  - Seen that mixed sessions in one SDP aren't supported, there must be a way to identify which Operation Set class should handle which request and reply.

    Solution: For the initial INVITE request, the file transfer Operation Set class searches for the file transfer mandatory attributes in the SDP body so that it can be considered a file transfer. If the attributes aren't found, and if the associated SIP dialog isn't registered to belong to a file transfer, the INVITE is left to be processed by the other operation sets (currently only the telephony one, as the video uses this

[112] Which is comprehensible seen that the specifications have only matured in May 2009. However, the Blink client (see section 3.1.1 in chapter 3), will implement this functionality in the near future. And most probably, other clients will follow the example.

one for interaction with the SIP stack). Also, the Telephony Operation Set checks if the SIP dialog is known to belong to a file transfer, and if not, it also performs the same search in the SDP body of the Request. Seen that to be valid, a response must have a SIP dialog associated, the operation set that processes the SIP responses is determined by the dialog search in the known file transfer dialogs database that is maintained per session.

- **Limitation:** The structure and the generic way that SC deals with file transfer requests, makes it difficult to implement the possibility to negotiate several file transfer per SIP Request.
  Problem because: The file transfer specifications (RFC 5547 [39]) allows negotiation of several file transfers in one SDP body. Therefore other fully compliant clients, might offer such a request, that currently isn't supported.
  Solution: For now, when more than one file transfer is identified in an SDP offer, SC replies to it with a *'488 – Not Acceptable Here'* and a warning message of *'399 – Miscellaneous'* with a comment *'Currently only one file transfer request per SDP body is allowed*'.
  To support several file-transfers per request, is easily achieved in Sip-Communicator. Although this isn't under my perspective a practical behavior, as upon such a request, a user would have to decide upon every file being offered before any transfer could start.
  It's more user-friendly to start transferring immediately when a user accepts each individual offer. That kind of behavior is only possible by sending a single file-transfer offer per request. If in the future, other SIMPLE IM make a habit out of making several file transfer offers per request, changes in SC can be easily made. However, I don't believe that these types of requests will be commonly used.
- **Limitation:** SC doesn't implement the '*The offerer is a File* Receiver' behavior specified in RFC 5547.
  Problem because: this behavior is listed in the RFC.
  Note: This behavior makes possible for an application to request a file via this protocol in an automated way. The only use case I can think of for IM purposes in which this feature would be useful would be to allow implementation of shared folders between IM users. Sip-Communicator currently doesn't support this, and no specifications exist to implement this feature, that would require a list of shared files to be transferred. For typical file transfer, IM users request a file by chatting with the other users. This behavior was probably specified to accommodate other use cases other than file-transfer through IM. (e.g. requesting a configuration file via SIP in an automated way, for which the name or other attribute is previously known, to a file server. More concretely, this behavior can be used by a SIP device to obtain its configuration file).

4.3 Instant Messaging system

Evaluation methodology

IST's community has around 10 000 potential IM users. The tests that I carried on the IM system are divided into:

- Performance tests – To evaluate the testing system's ability to cope with all the potential users;
- Feature tests – To ensure that all of the system's functionalities are working correctly.

The bibliographic research that I did before making the tests, revealed not only the ways to test the IM system but also several other tests for the related VoIP infrastructure. Although related, the VoIP infrastructure is outside of the scope of this work. Therefore, I executed tests only to evaluate the IM features and VoIP functionalities that changed due to the new software components of the new architecture (see section 3.1.2).

In this section, details of the evaluation methodology for each kind of tests are presented. Afterwards, I present the collected results along with a discussion about them. Although no security tests were carried, a word about the applied security recommendations is given. In the same subsection, details about my concerns on some security aspects of the VoIP infrastructure are given. These concerns were motivated by the considerable amount of security related tests for a SIP based VoIP infrastructure, that I found in my bibliographical research, as well as by inspecting the current VoIP configuration file with these concerns in mind. To end this section, possible future tests are presented for both IM and VoIP architectures.

Evaluation methodology – Performance tests

I performed two sets of tests to assess performance:

1. Register test (depicted in Fig. 4.1) – The new IM system architecture, due to the functionalities directly supported by the used software, makes it possible for the server to connect directly with the LDAP servers for authentication purposes. The performance test main objective is to give a result basis to assert the new authentication method scalability;
2. Message test (depicted in Fig. 4.2) – Page mode (see subsection *'SIP/SIMPLE'*, in section 2.1.2) as already stated in section 3.2.1, was the IM mode adopted in the testing system due to current implementation limitations. On a page mode IM setup, as all the messages must pass through the server, the IM server is a probable bottleneck for the distribution of messages. The main objective of this test is to give a result basis to infer how many users can eventually use the system simultaneously.

In order to simulate different users in the tests, I ran two slightly different versions of each test. One with the credential cache turned on[113], and another in which it was turned off[114]. Making a total of four tests.

[113] Credential caching is by on by default, more details on how it works can be found in subsection *'Authentication via LDAP'* in section 3.2.1 of chapter 3.

SIPp [42], in combination with Sysstat [43] were the chosen tools to carry the tests and gather the results.

Sysstat is a powerful tool that can provide all kind of statistics about status of different operating system components (bandwidth, CPU, memory, socket, process creation, interrupts, etc. a full list is available at [43]). At the same time the SIPp tests were being carried, sysstat ran with the following arguments:

sar -bB -n ALL -q -r -u -o <name of test file> 2 3000

This command gathers several information about different components of the OS. The 2 and 3000 parameters mean that statistics are obtained every 2 seconds 3000 times. The two second interval was chosen, as the Sipp tests increase their rate of requests per second every two seconds. The choice of the 3000 number was partially random, following only the requirement to be big enough to make sure that data was collected throughout the duration of the Sipp tests. After each Sipp test ran, the *sar* utility of Sysstat was interrupted, and the values were stored in a comma separated value (CSV) file, for further analysis. CSV files were generated with the use of the *sadf* command that ran with the following arguments:

sadf -d <name of test file> -- -bB -n ALL -q -r -u | tee <name of test file>.csv

Note that more information was gathered with these parameters than the one presented in the results subsection. This is because the other statistics weren't considered relevant after analysis.

Sipp is a very powerful and flexible tool to make SIP performance tests / traffic generation. Sipp allows a user to create custom SIP traffic (scenarios). The scenarios are specified in XML using Sipp's well documented XML syntax. For each set of tests, I created custom scenario files, which are available at appendix A.

For the message test, I created two custom scenario files for the two IM entities that are depicted in Fig. 4.2 (UAC and UAS/UAC).

Although in Fig. 4.2 two SIP entities (UAC and UAS/UAC) are depicted, they were in the same network endpoint.

For the register test, this command was used:

./sipp -rate_increase 5 -fd 2s -rate_max 10000 -i 193.136.132.51 -p 5050 -nd -default_behaviors none -trace_stat -sf AUTH.xml im.ist.utl.pt

This command makes Sipp generate the packets needed for the register scenario (-sf AUTH.xml) and stores the time spent from the first REGISTER request to the last 200 OK

[114] To accomplish this, taking into consideration the authentication pseudo-algorithm presented in *'Authentication via LDAP'* in section 3.2.1 of chapter 3, steps 1 and 2 where commented in the configuration file. Step 6 was left uncommented on purpose so that the time of caching the credentials is also included in the test, as it would if a different user would authenticate.

response, which he designates a call, along with other statistics (-trace_stat). It starts with 10 calls per second (CPS) and for each two seconds (-fd 2s) it adds 5 to the already existing rate (-rate_increase 5). When failed calls started occurring at a regular interval, I stopped the test by manually exiting the application (by pressing the 'q' exit key), the *'–nd'* and '*–default_behaviors none'* made sure that Sipp wouldn't send a BYE request for each existing call.

For the message test, the issued commands were:

./sipp -aa -i 193.136.132.51 -p 5063 -default_behaviors none -sf MESSAGE_UAS2.xml im.ist.utl.pt

For the UAC/UAS, which was responsible for making the test initialization phase (by registering with the IM server) and for receiving and replying with a 200 OK response to the incoming messages from the UAC. The UAC was run afterwards in a different terminal with the command:

./sipp -rate_increase 5 -fd 2s -rate_max 10000 -i 193.136.132.51 -p 5050 -nd -default_behaviors none -trace_stat -sf MESSAGE_UAC2.xml im.ist.utl.pt

Which has similar arguments from the ones used in the register test. While making the test, I noticed that the memory usage was growing. Instead of quitting the test when errors started to appear, I left it running until memory usage stabilized or swap memory was starting to be used.

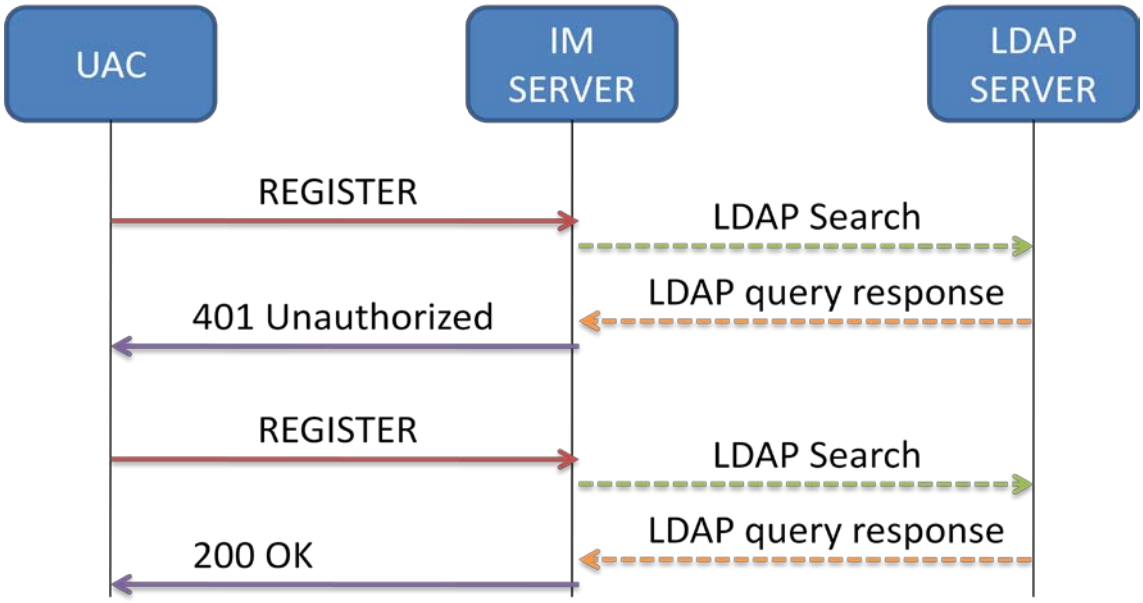

Fig. 4.1. Register test diagram – The dotted lines represent communications that might not happen depending if the credentials are already cached or not.

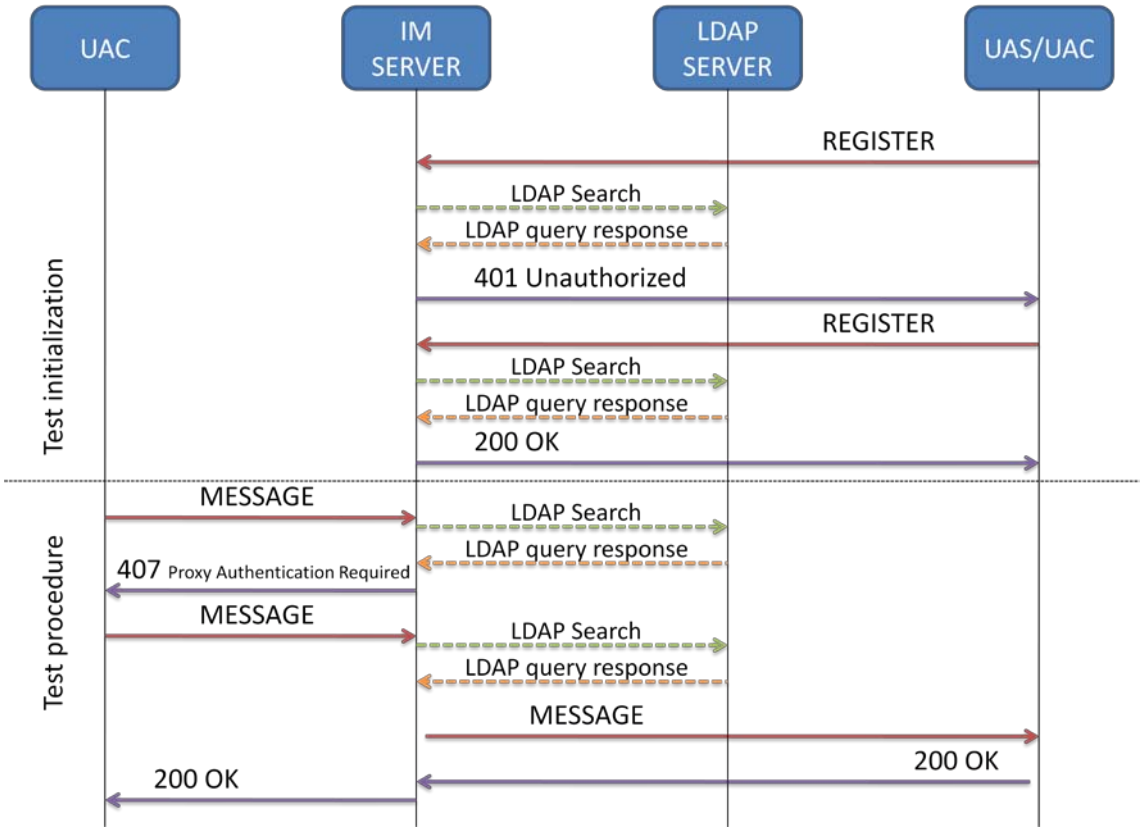

Fig. 4.2. Message test diagram – The dotted lines represent communications that might not happen depending if the credentials are already cached or not.

Evaluation methodology – Feature tests

For the feature tests, very straightforward methodologies were applied. Whenever possible, clients that used the features were used. For the core IM system, Sip-Communicator and

CounterPath's Xlite[115] were used to test the instant messaging and presence capabilities. For OpenXCAP, the software provided a script that acts like a client and that tests multiple commands. To note that although OpenXCAP isn't currently connected with the LDAP server to provide authentication, for these testing purposes, database authentication was used and a fictitious user created.

For the RLS capabilities, the IM server's database was checked to see if the resource's persistence was assured.

Results – Performance tests

Taking into consideration some advices regarding metric and parameters to be reported, that were found in SipStone [44], table 4.4 was created.

Server hardware details:	
Processor:	Intel® Pentium® 4 CPU 3.00GHz 512KB L2 cache Family 15 Model 2 Stepping 9
Memory:	1GB RAM - 2 DIMM DDR working @ 166MHz 512MB each
Network Interface:	3COM 10/100 onboard Ethernet device. – Board brand/model: Asustek/P4R800-VM
Disk:	Size: 80 Gb Speed: 7200 RPM Brand/Model: Barracuda 7200.7 / ST380011A
Server software details:	
Operating system version:	Debian "lenny" 5.0.3
Kernel:	Linux kernel debian package: linux-image-2.6.26-2-686 2.6.26-19
Other miscellaneous info:	
Number of servers: 1	**Number of probes:** 1
Network topology notes:	Connected on the same switch, in the same network over a 100Mbps Ethernet connection
Notes:	The test ran from ~00:30-02:30 in the 17th October 2009

Table 4.4. Details on the performance tests setup.

For the two sets of tests, which ran twice to simulate various users[116], the plots of the most important values are presented in Fig. 4.3 through 4.6. The Successfull Calls parameter represents the number of times the Sipp scenarios, depicted in Fig. 4.1 and 4.2, occurred successfully (without retransmissions, unexpected messages, or other fail reasons). The Failed Calls parameter, that is plotted on the vertical axis on the right, accounts for the number of calls

[115] A popular free to use SIP client that also supports IM functionalities. More information and download links can be obtained at: http://www.counterpath.net/

[116] For the details and reasons, please refer to the performance's evaluation methodology subsection above.

that simply didn't occurred as depicted in Fig. 4.1 and 4.2. All of the failed calls that occured in the tests, represented in the secondary axis of figure 4.3 through 4.6, happened due to the fact that the maximum number of retransmissions has been reached. The maximum number of retransmissions before a call is considered as failed, is 7. This means that all of the failed calls depicted below, have failed, because they had to be retransmitted more than 7 times. A call is retransmitted if a response is not given within 500ms. Therefore, failed calls are requests that took more than 4s (500ms * 8) to obtain a reply.

The Retransmissions parameter accounts the number of SIP messages being retransmitted due to expiration of the SIP transaction timer (as defined in section 17.1.2 of RFC 3261 [17]). The timer was defined for both scenarios with the RFC 3261 [17] recommend value of 500ms.

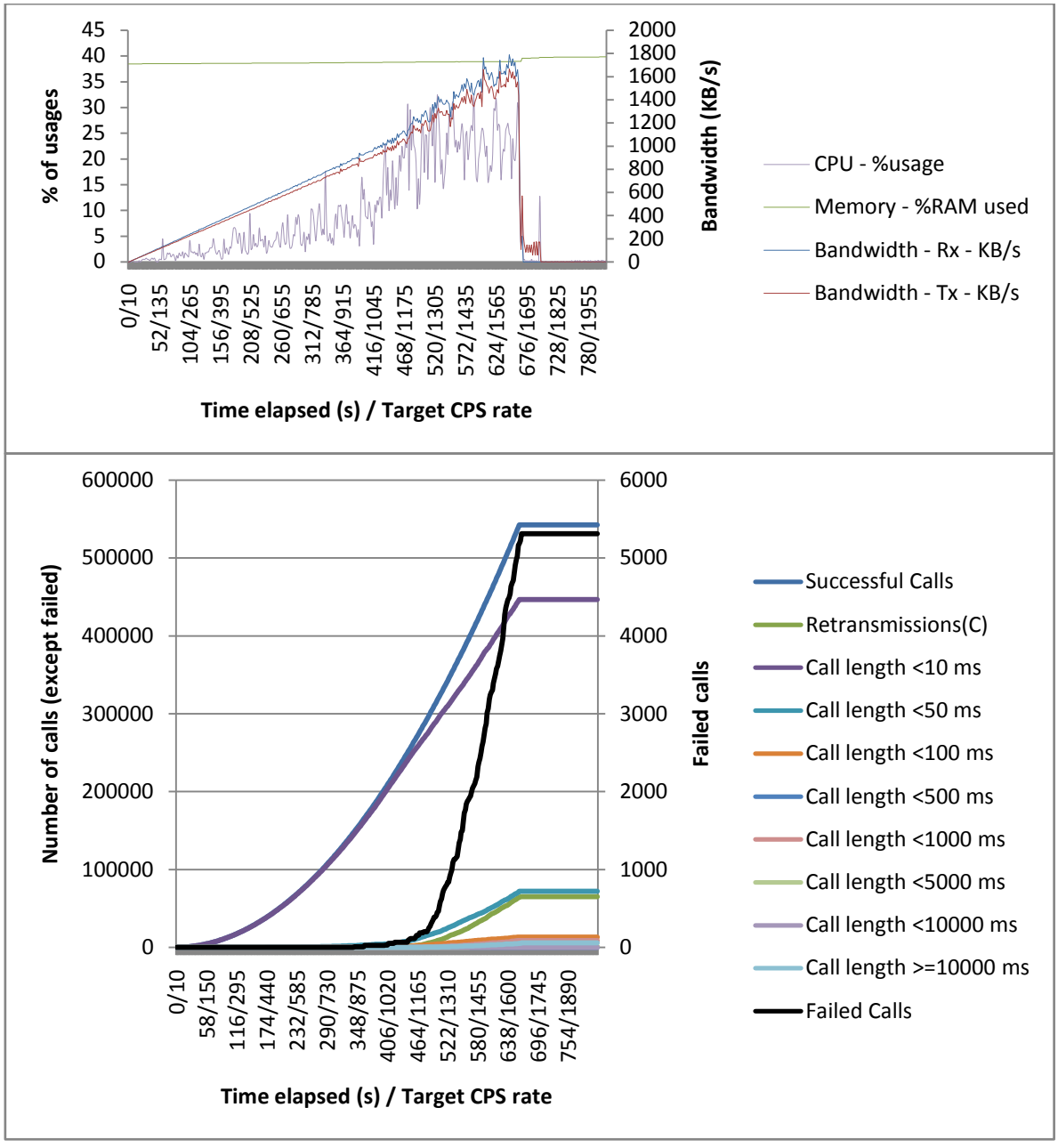

Fig. 4.3. Register test results, with credential cache.

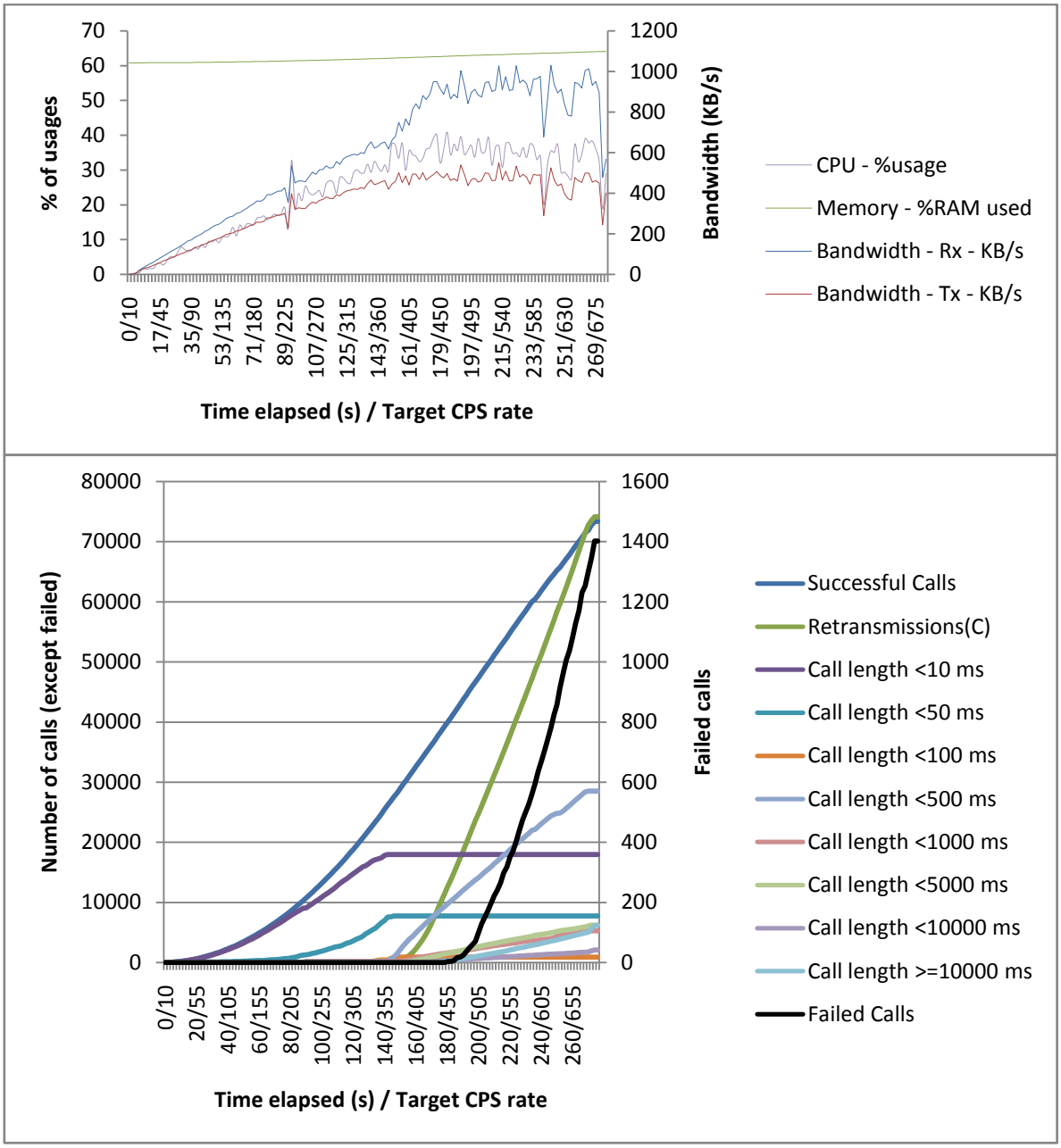

Fig. 4.4. Register test results, without credential cache – to simulate several users (see subsection *'Evaluation methodology – Performance tests'* above).

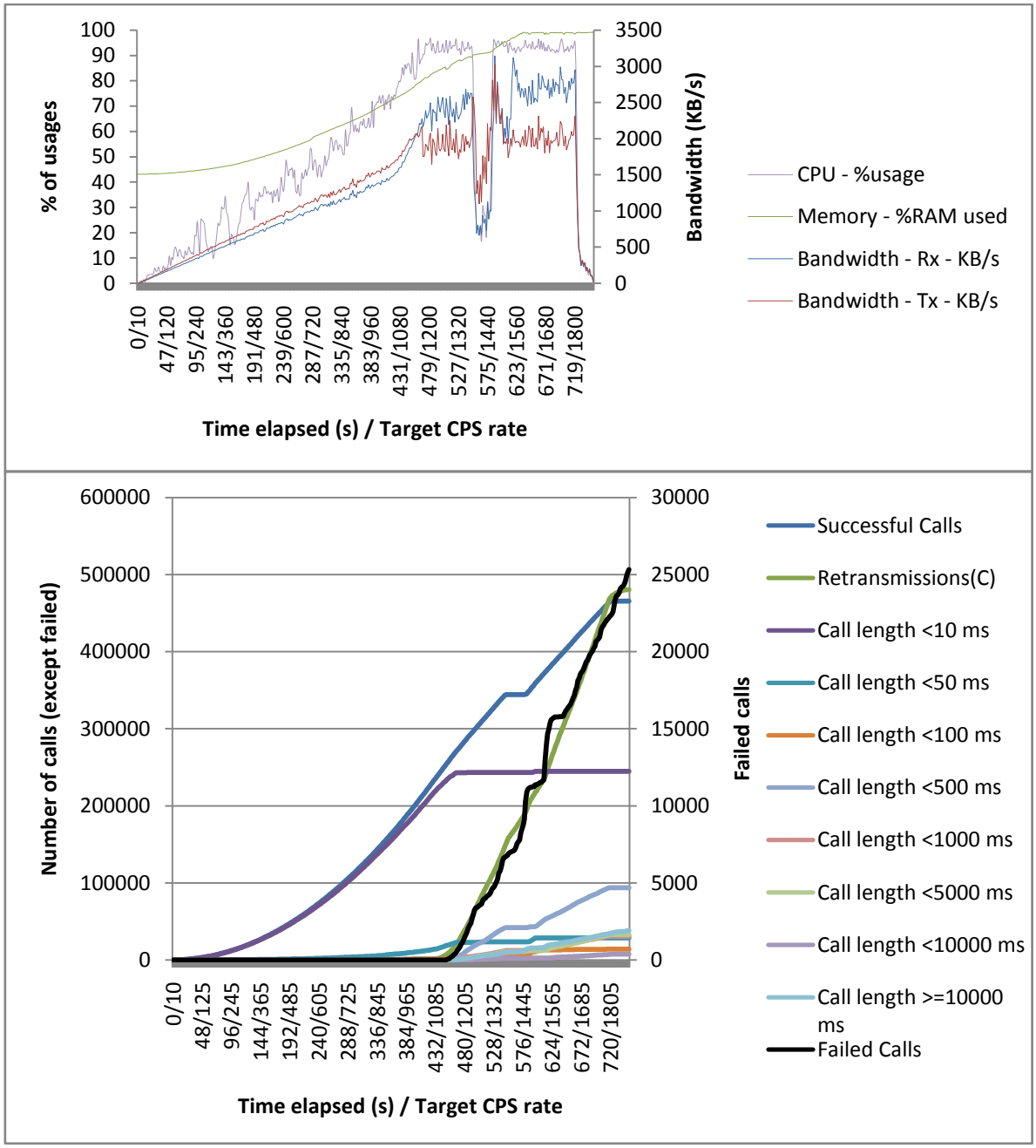

Fig. 4.5. Message test results, with credential cache.

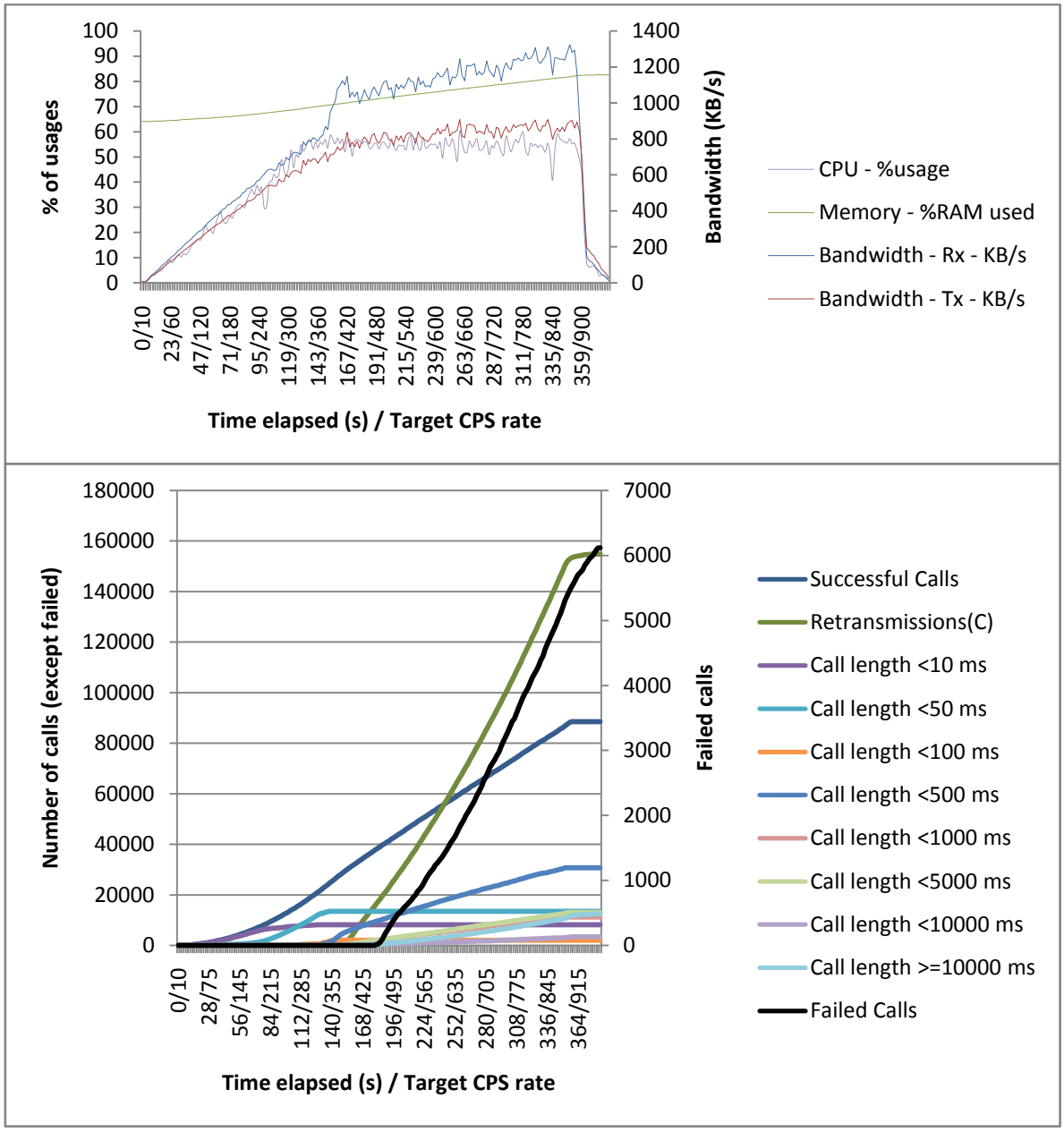

Fig. 4.6. Message test results, without credential cache – to simulate several users (see subsection *'Evaluation methodology – Performance tests'* above).

Figures 4.3 through 4.6 represent tests that ran with OpenSIPS configured to have 4 threads. Also, the same tests ran with the thread parameter set to 16 and 32, but seen that both tests attained a lower CPS rate, therefore, their graphics aren't presented here.

Discussion – Performance test results

The tests that were made with credential cache, intend to evaluate the scalability of the server simulation various simultaneous requests per second, that in a real-life scenario, would come from different users. It may happen that operating system and software optimizations that I'm unaware of (specially with the cache module, memcache, of OpenSIPS) might promote different results if the requests were actually made with different credentials associated with different

users. Still, I think that it's reasonable to assume that all of the results wouldn't deviate much from the actual results if different credentials were used[117]. Therefore, I use the acquired results to make estimates on scenarios where the requests would come from different users.

The other two executed tests were made without credential cache to simulate requests from different users that didn't authenticate themselves previously (via the register request or by authenticating a message request) in the last 20 minutes (because if they had authenticated, their credentials would have been in cache). The problem with the achieved results and with the intended objective, is that LDAP server optimizations used[118], might contribute to very different values if actually different credentials were used. Therefore, the drawn conclusions must take this fact into account. By talking with the IST's LDAP service admin[119], I found that most of the overhead of using LDAP, comes from the ciphered connection that the LDAP service requires. If the LDAP server is found to be a too limiting bottleneck for the IM service, three solutions exist that can certainly solve the bottleneck problems:

1. The number of register requests done without credential cache can be greatly reduced by augmenting the lifetime, or even removing the lifetime parameter associated with the cached credentials. This solution is expected to therefore mitigate problems associated with low maximum CPS rates of tests done without credential cache – This is probably the most practical solution that can be implemented. Problems with this solution might arise if other LDAP attributes are to be stored that are big enough to pose memory consumption problems if they would have to be stored for all of the 10000 potential users. However, due to the usually small number of bytes occupied by the attributes, it is unlikely that memory consumption becomes an issue.
2. Replicate the LDAP service locally (this is currently being done on other IST services that use LDAP and had problems with a centralized server) – this allows an increase of the number of requests per second permitted, as ciphered connections wouldn't provide an overhead anymore, and the LDAP service would be exclusively used by the IM service;
3. Have several IM servers, each with a local replicated LDAP service, and managed with a load balance mechanism – Regarding the LDAP bottleneck only, this solution would only be required in the improbable[120] case where the number of concurrent database accesses would be a limiting factor. It's noteworthy at this point to disclose that OpenSIPS has a module to implement load balancing. Of course, load balancing can also be achieved using other mechanisms than the ones provided by OpenSIPS.

[117] Different credentials weren't used as it would add a great deal of complexity to the test setup. And, like stated in the paragraph, probably without great differences in the outcome of the tests.
[118] Currently, LDAP services use indexes in the databases to optimize the response time of queries.
[119] André Brioso, see acknowledgements, again, thanks for the insights regarding the LDAP service.
[120] As the number of simultaneous requests on a maximum population of 10000 users isn't expected to be high enough to max out a local LDAP service, seen that credential caching is used.

To note that in most tests, CPU, memory, and bandwidth usage graphics, show that the server hasn't fully exhausted its resources. This suggests that there might be another limiting factor, or that fine tuning of the server software configurations (e.g. the number of OpenSIPs threads, the priority of the OpenSIPs processes within the OS, etc.) might provide an even bigger maximum CPS rate. These parameters weren't tuned as it makes more sense to do so in the production system, as a different hardware configuration might render the fine-tuning efforts useless. To note that due to the fact that worse results were attained with a number of threads of 16 and 32 proves that increasing the number of threads can have an adverse effect on performance.

For each of the four executed tests, and whose data is summarised in the graphics depicted in figure 4.3 through 4.6, comments on the implications of the attained results are given:

- Register test results, with credential cache:

Figure 4.3, shows that the number of failed calls starts to grow considerably around 1165 CPS. Register requests don't happen often in a normal SIP session. Usually, a register request has expiry times of minutes or even an hour. Register requests are usually emitted when a SIP agent is started, and when making SIP calls. Therefore, is very unlikely for the server to need to handle so much requests per second for registers with cached authentications. I think it's fair to conclude that the testing system characteristics and configurations offer enough provisioning for all of the potential IM users, considering only register requests whose credentials have been cached.

- Register test results, without credential cache:

Figure 4.4, shows that the number of retransmitted requests starts to grow considerably ~400 CPS. Retransmissions occur when no response is given for more than 500 ms. The fact that retransmissions start occurring before failed calls do, added to the fact figure 4.4 shows that the system can handle ~1165 registers per second, makes it valid to assume that the bottleneck and reason for the retransmissions, is, as expected, the need to obtain a response from the LDAP service.
In the extreme case where all of the 10000 potential IM users would become actual users, and had at the exact same moment the need to make a register request whose credentials would not be cached, they all would be served in ~25 seconds[121]. This is an extreme case. It is also a case that doesn't consider parallel requests.

[121] Assuming that the SIP/SIMPLE agent retries the Register request enough times without asking the user to confirm the credentials. Also assuming that the server doesn't have many other parallel kinds of requests being done at the moment. Note that parallel requests might occur, as the IM server isn't completely maxed out at 400CPS, just the link between the IM and LDAP servers is.

A real life scenario is difficult to implement in the laboratory (with different kinds of requests being made in parallel). Also, I don't find a maximum rate of 400 CPS a very comfortable one. For these reasons, I believe that the production system, if it has a similar maximum CPS rate, should have a mechanism that reports statistics and monitors for failed requests just in case.

This would give the admin of the system a way to deploy in a timely manner one of the solutions presented above to solve the LDAP bottleneck problem.
I believe that total user adherence is not a probable scenario, and if occurs, it probably won't be overnight. Therefore I conclude that this configuration and maximum rates can be used for the production system, as long as it is monitored conveniently to make sure that it continues to serve enough users without service disruptions.

- Message test results, with credential cache:

Message retransmission starts to grow at 1050 CPS target rate, and transmission failures (failed calls) at 1130 target CPS[122]. As page mode is the current mode employed by the IM system, at least until session mode isn't widely implemented in most existing SIP/SIMPLE IM clients, the capacity of the server to handle the MESSAGE requests is critical.
Extending the recommendations found in RFC 3428 [37], as stated in section 3.2.1, message authentication is required by the implemented system. This adds an extra overhead to the transmission of the messages, but with important gains on security.
Message requests must be handled by the IM system servers not only for messages originated internally, but also for messages from other SIP/SIMPLE domains. To note that messages from other domains don't trigger authentication mechanisms in this IM server.
The MESSAGE method specifications (RFC 3428 [37]) recognised that this method is prone to congestion, and defined a behaviour for SIP user agents that provides *"an adaptive mechanism to slow the introduction of new MESSAGE requests to the same destination"* by demanding that no new MESSAGE transactions are sent to a destination before a reply is obtained. This section also states that the UDP transport is the worse for the MESSAGE requests (all of the tests were made using UDP) due to the lack of congestion control mechanisms in UDP and need to implement retransmission due to possible packet loss.
Seen that MESSAGE requests should be exchanged very often by active IM users, the 1050 CPS rate isn't a very comfortable one. These results, assuming that no other parallel requests are made, support a scenario of about 3000 users that are actively engaged in IM, and therefore send a message every 3 seconds. To note that the fact that the rhythm of message generation usually depends on the rhythm of the conversation, when delay is introduced with the retransmissions, the message generation rate per user should also decrease (if the

[122] Although figure 4.5 doesn't provide enough detail, consulting the data that originated the graphic allows me to be more specific about these CPS values. Due to the fact that the data sources for each graphic represent tables of more than 40 pages for each test, I decided not to include them here. They would only add more detail that in most cases would be useless

behaviour stated in RFC 3428 [37] is followed). However, waiting for more than 4s for a message to be delivered, doesn't promote a very instantaneous instant messaging system, and users may abandon such a system if it becomes that sluggish. Therefore, the failed call indicator, although the message might still be transmitted, represents a good indicator of a capacity for a IM system that indeed aims to be instantaneous.

These results support the setup of an IM server were the MESSAGE requests should be routed to a server or a cluster of servers to allow higher CPS rates. These setup recommendations make the implementation of a production system more expensive. However, like stated in *'A word on SIP/SIMPLE adoption'* section 2.1.2, I believe that adoption of session mode IM by SIP/SIMPLE clients is not far away. If all the IM clients support session mode instant messaging, the dedicated server or cluster of servers to handle the MESSAGE requests is no longer needed[123].
At an initial stage, with the appropriate disclaimer about this congestion issue to promote users comprehension and manage their expectations, a more simple and less costly production system comprised of only one server, can be deployed. With the risk that users start to think that it's useless if its processing capacity doesn't grow as necessary along with the number of users. If all of the requests are handled by one server, I would strongly recommend that statistics are made about the users. When more than 900 users start to use the system, a dedicated server to handle MESSAGE requests should be deployed. As an alternative, at an initial stage, access to the system could be allowed only by invitation to a determined number of users that are interested in actively using the system (this could be easily achieved, e.g. with a LDAP attribute setting that would be fetched along with the credentials that would discriminate if the user could have access or not).

- Message test results, without credential cache:

These tests aren't as meaningful as the ones with credential cache. The reason is that usually a SIP/SIMPLE IM client registers itself with the IM server before starting to transmit messages, therefore allowing the server to cache their credentials. The scenario where several users start to transmit messages without their credentials being cached, is very unlikely, even if the credentials have a limited lifetime. Failures start at ~495 CPS. Conclusions about the server scalability are difficult to be drawn from these results due to the facts stated above.
However, they clearly show the advantages of having credentials cached for each user. If no credential cache was introduced, the server would be able to handle a rate ~56,2% lower than

[123] Because that, like stated in section 2.1.2, the session mode is more resource efficient for long run IM sessions. Only the initial IM media session setup costs any resources to the IM server just like a regular voice call. After the initial setup, the connection is typically made directly between the participants (unless some sort of other mechanism is needed to enable connectivity, e.g. MSRP Relay for users behind NAT. But even these mechanisms should consume much less CPU and memory resources).

with credential cache (taking into account that calls start to fail at a 1130 CPS target rate, as stated above in the message test with credential cache discussion).

Results and discussion – Feature tests

All of the tests described in the evaluation methodology subsection were concluded successfully. Due to the nature of the tests, no useful output from them are available, as they are simply pass or fail tests. To note that further tests could be made given more fully featured SIP/SIMPLE IM clients, that simply weren't available at the time of writing of this section.

A word on security

SIP MESSAGE requests are used to exchange messages in the adopted page mode system. As stated in section 11 of RFC 3428 [37], most security issues are a responsibility of the IM clients (referenced to in the RFC 3428 [37] as user agents).
The recommendation regarding IM proxy behaviour for outbound messages was implemented for all of the MESSAGE requests that go through the IM server.
Regarding the SUBSCRIBE and NOTIFY SIP requests, access control, notifier privacy mechanism, and confidentiality security issues are to be solved by the IM clients. The denial-of-service, replay, and man-in-the middle attacks are to be mitigated by proper IM server configuration, that should ensure that certain headers are provided, and that the requests are handled appropriately by the server.
These questions are expected to be addressed and solved in the configuration file of the IM server. However, some tests can be made to ensure that they are indeed working at a practical level.
Regarding the PUBLISH method, specified in RFC 3903 [22], recommendations about the security issues were taken into account in the configuration script. However, further research of this matter is still needed as well as practical evidence gathering that the desired recommendations are indeed enforced by the proxy.

For the non-IM SIP functionalities, it would be useful to test the current infrastructure and the new one with a set of common SIP vulnerabilities. These tests aren't executed in this work. This belief comes from my research about this issue and inspection of the configurations of the actual VoIP infrastructure. These made me believe that the current VoIP infrastructure, and therefore the new one that is integrated with IM (as it uses the configuration of the old one) might be vulnerable to some attacks (e.g. spoofing INVITE requests between SIP users; Spoofing of BYE requests, as they aren't authenticated. Therefore, a malicious user that knows the Call-Id of a SIP dialog between other users, by either listening to the traffic or guessing it, can spoof the request and terminate the session).

A word on future tests

Like stated above, some security tests should be done to assert that some security attack mitigation behaviours are correctly implemented. This applies to the VoIP services as well as some of the IM functionalities.
Also, regarding the performance of the IM system, fine tuning of the thread parameter and priority values of processes on the production system might provide interesting gains in the maximum CPS rates. This fine tuning would be pointless to be made in the testing system as the server characteristics will probably change for the production system. However, the details regarding the methodology of the tests (tools used and respective command parameters, scripts, etc.) that need to be executed to fine tune the system are available here. And once the production system is defined, tests can easily be repeated for the purpose of fine tuning.

It may also be interesting to make a smoother increase of the CPS in the tests, to assert a more precise maximum value for the server in each scenario. This information could be especially useful allow a better decision on when a new server must be added to the system. Assuming that the production system does indeed generate the herein recommended statistics.

Chapter 5

Conclusion

5.1 Discussion

The SIMPLE extensions to the SIP protocol are still in an immature state. This is due to the underdevelopment of actual implementations of the already defined SIMPLE standards, which are relatively recent. Nevertheless, mostly due to the close connection with SIP and SIP related technologies, SIMPLE has a great potential to promote IM services.
This work helped to mature the SIMPLE open standards by contributing with open source implementations of relatively recent standards. It also contributed to the open source world, as all the implementations are open sourced and published [38] [47].
To the IST's community, a new IM service possibility has been opened and deployed in a beta stage. The IM service is in a beta stage mostly due to the immature state of the chosen protocol, that forced me to spend a great deal of effort in software implementations.
Mostly due to the tests made and consequently results and conclusions drawn from them, directions and obstacles are now more clear, providing a good basis to develop a concise and good deployment plan for a mature IM service for IST and, possibly, other communities.

5.2 Future work

Due to the multiple work directions pursued, this work opens many doors to further developments and improvements that ultimately will continue to provide contributions to the IM, SIMPLE, and open source worlds.

MSRP Peer Library

As stated in section 3.3.2, MSRP library development will continue to be performed. In the same section, the features to be implemented are listed. Development of these features might be added by contributions, as I continue to receive manifestations of interest regarding the library via e-mail.
The achieved results of the library state regarding performance, point that the there is room for improvement in the runtime performance of it.

Sip-Communicator

Together with some MSRP library improvements, the MSRP library opened possibilities for implementation of the following features in Sip-Communicator:

- MSRP relays support – improving the connectivity of all of the features that depend on MSRP, including the already implemented file transfer functionality;
- SIMPLE's session mode IM support;
- Desktop sharing;
- Whiteboard sharing;

IM System

Further tests identified in section 4.3, are yet to be done before migrating the system to a production phase. As these tests might impose new requirements to the IM system. Also, the deployment of the MSRP Relays and OpenXCAP components can prove to be valuable in the future, as currently there are no free IM clients that are able to use the features offered by these components. To attain a fully integrated deployment of these two components, LDAP authentication support should be implemented in both of them. As these two software components are developed in Python, with a modular design, and also seen that Python has a mature LDAP library, implementing LDAP authentication should be a very straightforward task.

Deployment of the production system should require an implementation plan as well as promotion amongst the IST and possibly other communities.

NlNet foundation [62] is aimed to stimulate open sourced implementations and documents that promote network research and development in the domain of Internet technology. Therefore, development and deployment of a publicly disclosed architecture of an IM system that uses SIP/SIMPLE open standards and open source solutions as this one is, is well within the scope of NlNet support activities. NlNet already agreed on funding part of the deployment of this IM system, therefore, conditions exist that make probable that the above stated future work will be carried.

Bibliography

[1] C. Kalt, “Internet Relay Chat: Architecture”, RFC 2810, April 2000

[2] Carlo von Loesch, “PSYC – Protocol for Synchronous Conferencing”, http://about.psyc.eu/

[3] J. Rosenberg, “SIMPLE made Simple: An Overview of the IETF Specifications for Instant Messaging and Presence using the Session Initiation Protocol (SIP)”, Internet-Draft draft-ietf-simple-simple-04, 31 October 2008.

[4] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Core”, RFC 3920, October 2004

[5] R.L. Rivest, A. Shamir and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, Communications of the ACM 21, February 1978, 120—126

[6] Hari Balakrishnan, M. Frans Kaashoek, David Karger, Robert Morris and Ion Stoica, “Looking up data in P2P Systems”, Communications of the ACM, Vol. 46, No. 2, February 2003, pp. 43-48

[7] Petar Maymounkov and David Mazières, “Kademlia: A Peer-to-peer Information System Based on the XOR Metric”, 1st International Workshop on Peer-to-peer Systems, 200

[8] C. Kalt, “Internet Relay Chat: Server Protocol”, RFC 2813, April 2000

[9] C. Kalt, “Internet Relay Chat: Client Protocol”, RFC 2812, April 2000

[10] T. Berners-Lee, R. Fielding and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax”, RFC 3986 January 2005

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616, June 1999

[12] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Core”, RFC 3920, October 2004

[13] P. Saint-Andre, “Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence”, RFC 3920, October 2004

[14] Scott Ludwig, Joe Beda, Peter Saint-Andre, Robert McQueen, Sean Egan and Joe Hildebrand, “Jingle”, XEP-0166 http://xmpp.org/extensions/xep-0166.html, 18 December 2008

[15] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr and J. Peterson, “Presence Information Data Format (PIDF)”, RFC 3863, August 2004

[16] P. Saint-Andre, “Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)”, RFC 3922, October 2004

[17] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handely and E. Schooler, “SIP: Session Initiation Protocol”, RFC 3261, June 2002

[18] M.Handley, V. Jacobson and C. Perkins, “SDP: Session Description Protocol”, RFC 4566, July 2006

[19] A. B. Roach, “Session Initiation Protocol (SIP) – Specific Event Notification”, RFC 3265, June 2002

[20] A. B. Roach, B. Campbell and J. Rosenberg, “A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists”, RFC 4662, August 2006

[21] G. Camarillo, A. B. Roach and O. Levin, “Subscriptions to Request-Contained Resource Lists in the Session Initiation Protocol (SIP)”, RFC 5367, October 2008

[22] A. Niemi, “Session Initiation Protocol (SIP) Extension for Event State Publication”, RFC 3903, October 2004

[23] J. Rosenberg, “A Presence Event Package for the Session Initiation Protocol (SIP)”, RFC 3856, August 2004

[24] J. Rosenberg, “SIMPLE made Simple: An Overview of the IETF Specifications for Instant Messaging and Presence using the Session Initiation Protocol (SIP)”, Internet-Draft: draft-ietf-simple-simple-04, 31 October 2008

[25] J. Rosenberg, H. Schulzrinne “An Offer/Answer Model with the Session Description Protocol (SDP)”, RFC 3264, June 2002

[26] B. Campbell, R. Mahy and C. Jennings, “The Message Session Relay Protocol (MSRP)”, RFC 4975, September 2007

[27] C. Jennings, R. Mahy, “Relay Extensions for the Message Session Relay Protocol (MSRP)”, RFC 4976, September 2007

[28] Niemi, A. and M. Garcia-Martin, "Multi-party Instant Message (IM) Sessions Using the Message Session Relay Protocol (MSRP)", draft-ietf-simple-chat-05 (work in progress), 5 October 2009.

[29] J. M. Arfman and P. Roden, “Project Athena: Supporting distributed computing at MIT”, IBM Systems Journal, vol. 31 n. 3 1992

[30] G. Camarillo, M. Garcia-Martin, “The 3G IP Multimedia Subsystem (IMS)” John Wiley & Sons, Ltd, 2005

[31] L. Broman, “IMS platform prototype”, Master thesis of Lulea university of technology, available at: http://epubl.ltu.se/1402-1617/2008/210/index-en.html

[32] Raymond B. Jennings III, Erich M. Nahum, David P. Olshefski, Debanjan Saha, Zon-Yin Shae and Chris Waters, “A Study of Internet Instant Messaging and Chat Protocols”, IEEE Network, July/August 2006

[33] C. Neuman, S. Hartman and K. Raeburn, “The Kerberos Network Authentication Service (V5)”, RFC 4120, July 2005

[34] P. Saint-Andre, A.Houri, and J.Hilderbrand, “ Basic Messaging and Presence Interworking between the Extensible Messaging and Presence Protocol (XMPP) and Session Initiation Protocol”, Internet-Draft: draft-saintandre-xmpp-simple-10, 7 August 2007

[35] W. Toorop, “Desktop sharing with SIP”, available at: http://www.nlnet.nl/news/2009/20090204-sip.html

[36] A. Regateiro, “Voice over IP System in an Academic Environment”, Instituto Superior Técnico MsC thesis, September 2007

[37] B. Campbell, J. Rosenberg, H. Schulzrinne, “Session Initiation Protocol (SIP) Extension for Instant Messaging”, RFC 3428, December 2002

[38] MSRP Java library project, http://msrp.dev.java.net

[39] M. Garcia-Martin, M. Isomaki, G. Camarillo, S. Loreto and P. Kyzivat, “A Session Description Protocol (SDP) Offer/Answer Mechanism to Enable File Transfer”, RFC 5547, May 2009

[40] G. Klyne, D. Atkins, “Common Presence and Instant Messaging (CPIM): Message Format”, RFC 3862, August 2004

[41] K. Beck, “Test-Driven Development by Example”, Addison Wesley, 2003

[42] SIPp project webpage: http://sipp.sourceforge.net/

[43] SYSSTAT utilities home page: http://pagesperso-orange.fr/sebastien.godard/

[44] H. Schulzrinne, S. Narayanan, J. Lennox, and M. Doyle “SIPstone – Benchmarking SIP Server Performance”, 12 April 2002, available at http://www.sipstone.org/files/sipstone_0402.pdf

[45] M. Wahl, T. Howes and S. Kille, “Lightweight Directory Access Protocol (v3)”, RFC 2251, December 1997

[46] MSRP Java library project, http://msrp.dev.java.net

[47] SIP Communicator – the Java VoIP and Instant Messaging Client, http://sip-communicator.org

[48] Google Summer of code 2008, “João Antunes Application Information”, http://code.google.com/soc/2008/sipcomm/appinfo.html?csaid=1CEECECA189098B4

[49] Google Summer of code 2008, “Sip Communicator code samples”, http://code.google.com/p/google-summer-of-code-2008-sipcomm/downloads/list

[50] Carl Shapiro and Hal R. Varian (1999). Information Rules. Harvard Business Press. ISBN 087584863X

[51] M. Handley, V. Jacobson and C. Perkins, “SDP: Session Description Protocol”, RFC 4566, July 2006

[52] Google Inc., “Android”, offcial webpage: http://code.google.com/android/

[53] Sip Communicator community and Cristina Tabacaru, “SIP Communicator on Android”, https://sip-communicator.dev.java.net/servlets/ReadMsg?listName=dev&msgNo=3989, 19 July 2008

[54] Raymond B. Jennings III, Erich M. Nahum, David P. Olshefski, Debanjan Saha, Zon-Yin Shae and Chris Waters, “A Study of Internet Instant Messaging and Chat Protocols”, IEEE Network, July/August 2006

[55] Google Inc., http://www.google.com

[56] J. Klensin, “Simple Mail Transfer Protocol”, RFC 2821, October 2008

[57] 3rd Generation Partnership Project – 3GPP Technical Specification Group Core Network and Terminals, “Technical realization of the Short Message Service (SMS)”, 3GPP TS 23.040 v8.3.0, September 2008

[58] Google Inc., “Google Talk for Developers”, http://code.google.com/intl/en/apis/talk/open_communications.html#developer

[59] Adium, http://www.adiumx.com/

[60] Pidgin, http://www.pidgin.im/

[61] ICQ, http://www.icq.com/

[62] NlNet foundation http://www.nlnet.nl

Appendix A – SIPp custom scenario files used

-- Start of AUTH.xml file --

```
<?xml version="1.0" encoding="us-ascii"?>
<scenario name="AUTHENTICATION test">
	<send retrans="500">
		<![CDATA[
REGISTER sip:[remote_ip] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
To: "IST"<sip:ist154457@im.ist.utl.pt:[remote_port]>
From: "IST"<sip:ist154457@im.ist.utl.pt:[remote_port]>
Contact: <sip:ist154457@[local_ip]:[local_port]>;transport=[transport]
Expires: 3600
Call-ID: [call_id]
CSeq: 1 REGISTER
Content-Length: 0]]>
	</send>
	<recv response="401" auth="true" optional="true" crlf="true" next="2" />
	<recv response="407" auth="true" optional="false" crlf="true" />
	<label id="2" />
	<send retrans="500">
		<![CDATA[
REGISTER sip:[remote_ip] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
To: "IST"<sip:ist154457@im.ist.utl.pt:[remote_port]>
From: "IST"<sip:ist154457@im.ist.utl.pt:[remote_port]>
Contact: <sip:ist154457@[local_ip]:[local_port]>;transport=[transport]
Expires: 3600
Call-ID: [call_id]
CSeq: 2 REGISTER
Content-Length: 0
[authentication username=ist154457 password=<CONCEALED>]]]>
	</send>
	<recv response="200" crlf="true" />
	<label id="1" />
	<ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200" />
	<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000" />
</scenario>
```

-- End of AUTH.xml file --

-- Start of MESSAGE_UAC2.xml file --

```
<?xml version="1.0" encoding="us-ascii"?>
<scenario name="Message test">
	<label id="2" />
	<send retrans="500">
		<![CDATA[
MESSAGE sip:ist1337@im.ist.utl.pt SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
From: ist154457 <sip:ist154457@im.ist.utl.pt>;tag=[call_number]
To: sut <sip:ist1337@im.ist.utl.pt>
Call-ID: [call_id]
CSeq: 1 MESSAGE
Max-Forwards: 70
Content-Type: text/html
```

```
Content-Length: [len]

<SPAN STYLE="FONT-FAMILY:Arial; FONT-SIZE:10pt ">quem es</SPAN>]]>
        </send>
        <recv response="407" auth="true" crlf="true" />
        <send retrans="500">
                <![CDATA[
MESSAGE sip:ist1337@im.ist.utl.pt SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port];branch=[branch]
From: ist154457 <sip:ist154457@im.ist.utl.pt>;tag=[call_number]
To: sut <sip:ist1337@im.ist.utl.pt>
Call-ID: [call_id]
CSeq: 2 MESSAGE
Max-Forwards: 70
[authentication username=ist154457 password=<CONCEALED>]
Content-Type: text/html
Content-Length: [len]

<SPAN STYLE="FONT-FAMILY:Arial; FONT-SIZE:10pt ">quem es</SPAN>]]>
        </send>
        <recv response="200" crlf="true" />
        <label id="1" />
        <ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200" />
        <CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000" />
</scenario>
```

-- End of MESSAGE_UAC2.xml file --

-- Start of MESSAGE_UAS2.xml file --

```
<?xml version="1.0" encoding="us-ascii"?>
<scenario name="Message test">
        <Global variables="registered,10" />
        <nop>
                <action>
                        <test assign_to="10" variable="registered" compare="equal" value="1" />
                </action>
        </nop>
                <nop condexec="10">
                        <action>
                                <jump value="2" />
                        </action>
                </nop>
<!-- let's register ist1337 first if it isn't already-->
<send retrans="500" condexec_inverse="10">
                <![CDATA[
REGISTER sip:[remote_ip] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
To: "IST"<sip:ist1337@im.ist.utl.pt:[remote_port]>
From: "IST"<sip:ist1337@im.ist.utl.pt:[remote_port]>
Contact: <sip:ist1337@[local_ip]:[local_port]>;transport=[transport]
Expires: 3600
Call-ID: ABCDEFGHIJ///[call_id]
CSeq: 1 REGISTER
Content-Length: 0]]>
        </send>
```

```
	<recv condexec_inverse="10" response="401" auth="true" optional="true" crlf="true" next="3"
/>
	<recv condexec_inverse="10" response="407" auth="true" optional="false" crlf="true" />
	<label id="3" />
	<send retrans="500" condexec_inverse="10">
		<![CDATA[
REGISTER sip:[remote_ip] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
To: "IST"<sip:ist1337@im.ist.utl.pt:[remote_port]>
From: "IST"<sip:ist1337@im.ist.utl.pt:[remote_port]>
Contact: <sip:ist1337@[local_ip]:[local_port]>;transport=[transport]
Expires: 3600
Call-ID: ABCDEFGHIJ///[call_id]
CSeq: 2 REGISTER
Content-Length: 0
[authentication username=ist1337 password=<CONCEALED>]]]>
	</send>
	<recv response="200" crlf="true" condexec_inverse="10"/>
	<nop next="2">
	<action>
	<assign assign_to="registered" value="1" />
	</action>
	</nop>
		<!-- END OF let's register ist1337 first -->
	<label id="2" />
	<recv request="MESSAGE" crlf="true" />
	<send>
		<![CDATA[
SIP/2.0 200 OK
[last_Via:]
[last_From:]
[last_To:]
[last_Call-ID:]
[last_CSeq:]
Contact: <sip:[local_ip]:[local_port];transport=[transport]>
Content-Length: 0

]]>
	</send>
	<label id="1" />
	<ResponseTimeRepartition value="10, 20, 30, 40, 50, 100, 150, 200" />
	<CallLengthRepartition value="10, 50, 100, 500, 1000, 5000, 10000" />
</scenario>
```

-- Start of MESSAGE_UAS2.xml file --