Design and implementation of a distributed file directory for mobile peer-to-peer.

Helsinki University of Technology **Thesis Abstract**

Author:	Victor Hugo Morales Reyes
Title of the Thesis:	Design and implementation of a distributed file directory for mobile peer-to-peer
Date:	9th of July, 2007
Pages: ix + 68	Professorship: S - 38
Faculty:	Electrical and Communications Engineering – Networking Laboratory
Supervisor:	Professor Raimo Kantola
Instructor:	Lic.Sc. Nicklas Beijar

The absence of a Peer-To-Peer (P2P) network designed specifically for mobile phones, which has proven extremely popular on fixed networks, is a very attractive topic from an academic research standpoint. This thesis begins by exploring the work done by the scientific community thus far in the field of mobile Peer-To-Peer. It then describes a novel approach [1] that utilizes the Session Initiation Protocol (SIP) to carry P2P control messages. This approach is called P2P-Over-SIP, and has several advantages over other non standard protocols.

We then describe a software implementation using P2P-Over-SIP. Especially we describe our implementation of a super-peer node, used to build a mobile P2P network with an unstructured semi-centralized architecture. We detail the results obtained from testing our implementation with realistic usage scenarios. An analysis of our results is done and conclusions are drawn on the properties of our network. Lastly we comment on possible future work to be done in this area.

Keywords: Peer-To-Peer (P2P); Session Initiation Protocol (SIP); IP Multimedia Subsystem (IMS); 3rd Generation (3G) cellular network.

Tekniilinen Korkeakoulu **Diplomityön Tiivistelmä**

Tekijä:	Victor Hugo Morales Reyes
Työn nimi:	Mobiilivertaisverkon hajautetun tiedostoluettelon suunnittelu ja toteutus
Päivämäärä:	9. kesäkuuta 2007
Sivuja: ix + 68	Professuuri: S - 38
Osasto:	Sähkö- ja tietoliikennetekniikan osasto
Työn valvoja:	Professori Raimo Kantola
Työn ohjaaja:	TkL Nicklas Beijar

Vertaisverkot ovat osoittautuneet hyvin suosituiksi kiinteässä Internetissä. Akateemisesta näkökulmasta mielenkiintoista on, että matkapuhelinverkkoa varten suunniteltuja vertaisverkkoja ei vielä ole.

Tässä diplomityössä käymme aluksi läpi mobiilivertaisverkosta tehtyjä tutkimuksia. Tämän jälkeen kuvaamme uudenlaista lähestymistapaa, joka käyttää SIP (Session Initiation Protocol) -protokollaa vertaisverkon merkinantosanomien kuljettamiseen. Tästä lähestymistavasta, joka tarjoaa useita etuja, käytämme nimeä P2P-over-SIP.

Seuraavaksi kuvaamme mobiilivertaisverkosta tehdyn toteutuksen. Erityisesti kuvaamme tekemämme supersolmun ohjelmistotototeutuksen, jonka avulla puolikeskitetty järjestämätön arkkitehtuuri muodostuu. Testaamme toteutusta oikeudenmukaisissa skenaarioissa ja tuloksia analysoimalla teemme päätelmiä arkkitehtuurin soveltuvuudesta laajempaan käyttöön. Lopuksi pohdimme aiheeseen liittyviä kehitysmahdollisuuksia.

Keywords: Peer-To-Peer (P2P); Session Initiation Protocol (SIP); IP Multimedia Subsystem (IMS); 3rd Generation (3G) cellular network.

Acknowledgements

This work is the culmination of a lifetime of efforts from my parents Guillermo Morales and Emma Reyes. It is to them that I dedicate it.
I want to thank my thesis instructor, Nicklas Beijar who was extremely patient with me throughout the entire project, providing very insightful advice, I am very grateful for that. I also want to thank Tuomo Hyyrylä who helped me on many occasions with his programming expertise.
I want to thank also the rest of my family for their support, my girlfriend Saara Oksanen, and the many friends I now have in Finland. Your friendship and support have been invaluable to me.

I also want to thank Professor Raimo Kantola.

This project was done for the University of Helsinki in the networking laboratory.

May 17, 2007
Victor Hugo Morales Reyes

Abbreviations

3G	Third Generation
3GPP	Third Generation Partnership Project
AH	Ad Hoc (Latin: for this purpose)
AOR	Address Of Registry
API	APplication Interface
AS	Application Server
CD	Compact Disc
CPS	Calls Per Second
CSCF	Call Session Control Function
DNS	Domain Name Service
DUM	Dialog Usage Manager
GCP	Gateway Control Protocol
GGSN	Gateway GPRS Support Node
GSM	Global System for Mobile communications
GPRS	General Packet Radio Service
HSDPA	High Speed Download Packet Access
HSS	Home Subscriber Server
IMS	IP Multimedia Subsystem
IP	Internet Protocol
ISDN	Integrated Services Digital Network
MP	Mobile-peer
MP2P	Mobile P2P
MP3	MPEG-1 Audio Layer 3
MPEG	Moving Picture Experts Group

MSC	Mobile Switching Center
MT	Mobile Terminal
OS	Operating System
P2P	Peer-To-Peer
PSTN	Public Switched Telephone Network
REPRO	reSIProcate PROxy
RESIP	reSIProcate stack + DUM
RFC	Request For Comments
RI	Routing Index
RTO	Retransmission Time Out
SGSN	Serving GPRS Support Node
SIP	Session Initiation Protocol
SQL	Structured Query Language
SP	Super-peer
TCP	Transmission Control Protocol
TTL	Time To Live
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
UML	Unified Modeling Language
XML	eXtensible Markup Language

Table of contents

List of Figures

List of Tables

1 Introduction

1.1 Background

The Internet and the personal computer changed the way we use and acquire music and many other digital media and services. After the advent of the Personal Computer (PC) and the Compact Disc (CD), it soon became possible to perfectly duplicate a CD at very little expense and with very little technical expertise needed. Not only that but a major breakthrough came in the form of MPEG-1 Audio Layer 3, most commonly referred to as MP3, a new encoding format that allowed excellent reproduction of the sound file but only at a fraction of the computer memory that an original recording needed.

Soon a revolution was inevitable. The small size MP3's were ingeniously made available throughout the Internet via a new file sharing system: Peer-to-Peer (P2P) networks. These networks are overlay networks that work on top of the Internet that enabled people to exchange all sorts of files using mainly the resources of their own computers added together rather than downloading them from a centralized server. By combining the resources of many computers such as disk storage space and bandwidth, P2P networks allow for the distribution of large digital media files such as music and movies.

At the same time the world of telephony was experiencing dramatic changes. Fixed telephony gave way to mobility. Improvements in system characteristics and compatibility added to stiff competition allowed widespread adoption of mobile terminals; we went from the heavy and bulky to light and slim phones.

Standardization played a key role enabling the success of Global System for Mobile communications (GSM) as a truly global technology.

Several new technologies allowed Internet connectivity to reach the mobile phone. However, still today network operators have not been able to successfully market these technologies and relatively few people use them. The 3rd Generation (3G) Mobile Network is meant to change this status quo; this new mobile phone platform has higher bandwidth and more secure radio channels and perhaps not surprisingly an all IP (Internet Protocol) core in the IP Multimedia Subsystem (IMS).

The IMS is a new service platform defined by the 3rd Generation Partnership Project (3GPP) for packet switched services. The IMS has at its core the Session Initiation Protocol (SIP) which has been embraced by the telecommunications industry as the standard for session control.

SIP could potentially be used for a P2P mobile network that would operate "natively" in the 3G core network. The advantages of such a network would be multiple; it would give control to the network operator of such things as access control and content restrictions. In this scenario it would be straight forward to think of a digital media marketplace in which not all media is free of charge but the operator could provide content for download for a given fee, and the same could be true of any mobile-peer. But is this all possible? Are IMS and SIP suitable for P2P?

Technology convergence is bringing the Internet, mobile telephony and Peer-To-Peer networks closer and closer together.

According to [2] by 2010 less than one percent of European mobile users will have a GPRS mobile phone as their primary phone, the great majority will be 3G capable. However, the same study points out that around a tenth of today's 3G handset owners actually use 3G functionality. The hardships for adoption of modern services are many ranging from national regulations to marketing and pricing. Nonetheless there is some consensus that the main driver for widespread 3G service adoption has to come in the form of new enticing applications. One such application could be mobile peer-to-peer file sharing.

1.2 Problem

While the Internet is thriving on new P2P applications, their adoption on the mobile network has not yet come. As [3] points out, the different technical capabilities of mobile phones have to be taken into consideration when designing such applications, many of the current P2P protocols and applications are simply not designed with those constraints in mind.

The motivation for this work is then to create an efficient distributed file directory that is designed specifically for mobile phones. This file directory will forward queries for specific files in a Peer-To-Peer fashion under certain circumstances. The reader is directed to Chapters 2, 3 and 4 for a description of the theory behind this scheme and its possible uses in a modern mobile network.

The relevance of this project can be seen from the following factors:

1. The growing popularity of Peer-to-peer applications.
2. The impact of these applications on established business models for long distance telephony, music/movie distribution and distributed computing.
3. The recent introduction of category 6 mobile terminals that support High Speed Download Packet Access (HSDPA).
4. Growing mobile network support for 3G and HSDPA.
5. Growing digital media capabilities of mobile terminals (MT).
6. Increasing media creation and consumption.
7. Finally the recent surge of interest in P2P by the scientific community.

The objectives for the present work are listed below, note however that some of these objectives are not easily verifiable, for those that are, there will be a set of test cases and the results of those tests will be summarized in chapter 7.

General goals:

- Evaluate the feasibility of a SIP based super-peer.
- Examine the capacity of SIP based super-peer.
- Examine possible modifications and extensions to SIP signaling.
- Evaluate user perceived performance of the mobile P2P network.

In order to achieve our general goals, the choice was made of using a software prototype implementation of the super-peer. The specific goals we set for this application are listed below.

The software application:

- Should have sufficient documentation to facilitate maintenance.
- Could conceivably be easily integrated into a modern mobile phone network.
- Could serve as a future basis for research work.
- Should be written in an object oriented language.
- Should be based on standardized protocol that is supported by the mobile network.
- Could conceivably scale to answer queries in the numbers expected from a live network.

This work is carried out in a framework of previous and ongoing research projects at the Networking Laboratory of Helsinki University of Technology. As such there are already some basis onto which this thesis will build.

Related specifically to this project are two papers, [1] and [4], in which the idea of a semi-centralized P2P-over-SIP network for mobile phones is first proposed. The implementation of the mobile client is described in [3] and [5]; this piece of software was reused for the purposes of the present work.

The software resulting from this work will further clarify the feasibility of a full scale implementation, the performance improvements gained by the architectural planning of the project and finally it will produce some clues as to which further optimizations can be made and possible future research areas to explore.

1.3 Scope

Has the current SIP standard the necessary features to carry P2P control messages? If so, could it be possible to implement a P2P network for mobile phones that works directly on top of the IMS? What would be the performance of such a network? These are the questions that we address in the present work.

This work's focus will NOT be:

- To design software for mobile phones. It is a fortunate situation that a suitable mobile client has already been designed and tested.
- To implement novel P2P forwarding algorithms. Although we do mention such methods and comment on their improved performance over traditional flooding, we will focus on building a platform onto which future research can be done, possibly including improved forwarding algorithms.
- To optimize the database. We will not delve into database optimization details, and other such issues which could potentially improve query performance. Our main task it to prove the feasibility of our proposed P2P network.

1.4 Structure of the thesis

We now turn our attention to the organization of the present work. First in Chapter 2 we go through some background information regarding Peer-To-Peer, the reader can see some history about pioneering P2P systems and their characteristics. We also lay down some architectural details of those systems. Finally we describe P2P forwarding algorithms and learn about their different

characteristics. In chapter 3 we go into SIP and briefly touch on the architecture of the IMS. In chapter 4 we put together SIP and the IMS and evaluate the feasibility of our proposed software development project and the likelihood that a super-peer could be set up as an Application Server(AS) within the IMS.

Chapter 5 is dedicated to describing the basic functionality of the super-peer as a file directory. We show our design choices regarding SIP methods to use, and how they are used. Finally, we show how mobile-peers and super-peers interact and how super-peers interact with other super-peers.

Chapter 6 gives an overview of the software architecture and the most relevant implementation details. Chapter 7 describes the equipment that we used and shows the results we have obtained, by evaluating the performance of our implementation in several test cases which are also described.

Chapter 8 gives final thoughts about the project and suggests future research possibilities in this area.

2 Peer-to-peer background

2.1 What is Peer-to-peer

For a basic understanding of what is a P2P network the reader is refered to [3]. Summarizing, we can state a general definition as; "the sharing of computer resources and services by direct exchange between systems". It is an alternative to the client-server model in which peers volunteer some of their resources and in exchange they get resources from other peers.

In a client-server model the server is passive and waits for queries from a client, if it receives one query it answers to it and waits indefinitely for another query to arrive. In contrast, a P2P model could be visualized as all nodes being clients and servers at the same time, since any one of them can initiate communication with another.

In P2P, nodes cooperate with one another by routing messages and storing or processing information, and in doing so, they form a network, peers can join or leave the network at any time. When a node joins the P2P network, it brings resources into it. When it leaves, it takes them away, thus the more peers there are in such a network, the more "valuable" it is. If at any one time there are no peers in the P2P network, the network itself ceases to exist because a P2P network is entirely built on software.

2.2 Architectures

2.2.1 Peer-To-Peer taxonomies

Understanding the different taxonomies of the P2P networks is helpful in comprehending the characteristics and architecture of our program and its classification in the spectrum of P2P applications.

We can classify P2P networks into structured and unstructured. While the former establishes strict rules for file placement and discovery, the latter uses a more loose approach with arbitrary network topology, file placement and search.

Unstructured P2P networks can be implemented in a centralized, hybrid or decentralized manner. Decentralized, also called "pure", Peer-to-peer lacks any centralized structure and should not be confused with ad-hoc networks. Centralized P2P requires a server where peers learn about other's available resources. A hybrid P2P combines the two aspects.

2.2.2 Centralized P2P

The centralized directory is a model that was introduced by Napster. In this type of system a peer would connect to a central directory where it publishes all the files that it wishes to make available. When a node wants a file, it queries the directory and the directory answers with the address of the node that can best serve the request (based on several possible criteria like bandwidth, availability or closeness to the requesting node). This method is simple and very robust but has the disadvantage of having to maintain the centralized infrastructure, and that possibly lacks scalability for the same reason.

The raise and downfall of Napster has been very well documented. There are several technical downsides to Napster's approach to P2P such as lack of scalability, single point of failure, resiliency against attacks, etc. However, the reason for Napster's demise was legal rather than technical; the centralized directory structure was perceived by a court of law to mean direct responsibility over the network's content even when no content ever resided on that centralized server.

2.2.3 Decentralized P2P

The distributed directory, also known as decentralized or "pure", is a model where all peers are equal and therefore no single point of failure exists. This is obviously advantageous for resilience purposes but requires complex algorithms that decide where to place what information, and also how to search for that information efficiently. Because all nodes are equal, they have to take on some responsibility in forwarding queries. The complexity of routing messages in such a network is large, and that requires similarly large computing resources which makes this kind of network unsuitable for mobile P2P (MP2P). Gnutella is an example of a decentralized P2P network.

2.2.4 Semi-centralized P2P

The semi-centralized approach uses nodes with higher capacity to forward queries, such nodes are called super-peers. The rest of the nodes (i.e. those that are not super-peers) can be seen as "leaf" nodes. Leaf nodes are effectively isolated from the rest of the P2P network; they are just limited to a one-to-one relationship with the super-peer they are attached to, moreover, they are

unaware of the P2P network. There are, however some approaches like M-CAN [8] in which one node can be attached to more than one super-peer.

Semi-centralized networks rely heavily on super-peers, they are responsible for the forwarding of messages and are usually nodes that have greater processing power and resources than normal peers. This arrangement places less strain on "weaker" leaf nodes.

If a peer wants to make files available for others to download, it has to register those files in its associated super-peer. When a peer wants a file, it queries its associated super-peer, in the same manner as a centralized directory. If the super-peer receives a query for which it does not have an answer in the local database it will query other super-peers. All the super-peers together form a pure-P2P network where queries will be forwarded.

2.3 Search algorithm

We now address the possible implementations of the search algorithm which is critical to the performance and scalability of the system. There are several approaches to the solution of this problem.

2.3.1 Blind vs. informed search

We can categorize search algorithms into blind and informed. Blind search is performed when we do not have any information of where the files we are looking for could possibly be located. In such a case the only possible course of action is to forward the query to all nodes in the network or a part of them, this is called flooding. Informed search is when a node can learn about the information contained in other nodes then any such node can make "informed" decisions about where a query should be forwarded.

Since we would expect the super-peers to reside in the core mobile network where bandwidth is not as expensive as in the MT air interface, we can learn from protocols that were not strictly designed to be used for mobile peer-to-peer, and possibly use some of those ideas.

P2P nodes are inherently untrusted because they can be located in different administrative domains. For the same reason, the availability of their resources cannot be guaranteed; nodes can disconnect voluntarily at any time but also there might be some technical impediment like availability of wireless connection or any natural or man-made disaster. These problems are subject to active scientific research and we will not treat them in detail. However, they will be taken into consideration when designing the architecture of the program.

2.3.2 Flooding

When forwarding queries the most common solution is to just flood the network with the request hoping that "some" node that possesses the file we are looking for will answer, this is one example of blind search. This method is also extremely simple and scales very well but has the downside of arguably low efficiency because every request requires considerable network bandwidth. Another problem with this approach is that there might be loops forming where the request that one node sends will possibly be received by the same node. To avoid this problem every request can have a unique identifier so that we can identify this situation. Another solution is to have a TTL specified but this also limits the coverage of the search. The TTL parameter is a loop-recovery technique and does not avoid multiple receptions of the same query; it is used as a last resort for protection against loops.

2.3.3 Random walks

If we want to still use a blind search but improve on flooding one alternative is to use random walks. This is a method in which one specific query is only forwarded to k neighbors chosen randomly rather than sending the query to all of them. The most important advantage of using random walks is the reduction in bandwidth which is k x TTL messages in the worst case. This algorithm works by uniformly sampling the super-peer nodes, and achieves a final random state quickly which translates intro practical network efficiency. The assumption here is that there are no bad links or "bottle neck" hops, this assumption is necessary for our previous assertion that convergence is fast.

Each forwarded message follows its own path, and is called "walker". A walker can terminate in failure either limited by a TTL or by communicating periodically

with the querying node and asking if the termination condition has been satisfied. Since choosing nodes is a random process there is a positive effect of load balancing at each node.

The disadvantage of this method is a highly variable performance since success depends on the random path chosen and the network topology.

2.3.4 Routing indices

One methodology utilizing informed search is the Routing Index (RI), RIs are described in [6]. Using routing indexes will save bandwidth and reduce delay by forwarding queries only to those super-peers that are most likely to have results for a given query. The answer to a given query will be a list of neighbor nodes ranked by their "goodness" for that particular query. The goodness of a node will generally relate to the number of relevant files, owned by the node, that match the query's criteria. We now make a very short description of how this method could be used in our mobile peer-to-peer application.

In this scheme, once a node receives a query it does not forward it further into the network, but rather answers with the RI for that query. The querying node will then decide if the results reach a predefined stop condition which would be the maximum number of files we seek. If the results are not enough (i.e. we do not reach the stop condition) the node will then forward the query to the second best neighbor.

In essence the RI will work as a routing table at every super-peer; telling the querying super-peer which nodes to forward its query to according to their

goodness for that query. There is one special consideration to be made here; as it is presented, the usefulness of RI will only apply to systems where files are subdivided into categories.

2.3.5 P2P taxonomies and forwarding schemes summarized

It is clear that categorizing files in our network would be of benefit, if we query for say, music files, we will decrease the data processing latency at each node. By using category information to guide queries we could decrease the bandwidth used by our network. The additional expense we incur is the memory required to keep track of previous queries seen by any one node. We believe that implementing categorization and exploiting it by using RIs would improve the performance of our network in a realistic commercial grade application.

There exist other forwarding algorithms; we will summarize those that are most relevant to us in *Table 1*.

Forwarding Algorithm	Algorithm complexity	Breadth of search	Worst case delay	Worst case # of messages for one query
Flooding	Very Low	Limited by TTL	Very High	Very High
Expanding ring	Low	Limited by ring horizon	High	High
Compound Routing Index [6]	Low	Limited by TTL	Unknown	Moderate
Hop Count Routing Index [6]	Moderate	Limited by horizon of RI	Unknown	Moderate
Exponentially Aggregated Routing Index [6]	Moderate	Limited by TTL	Unknown	Moderate
ORION [7]	Moderate	Unknown	Unknown	Moderate
M-CAN [8]	High	Unknown	Moderate	Unknown
Random walks	Low	Limited by TTL	High	Low
Adaptive Probabilistic Search [9]	Moderate	Limited by TTL	High	Low

Table 1: Forwarding algorithms for Peer-To-Peer networks

Figure 1 shows graphically the P2P taxonomies we have discussed and places our software application as an example of semi-centralized unstructured P2P network.

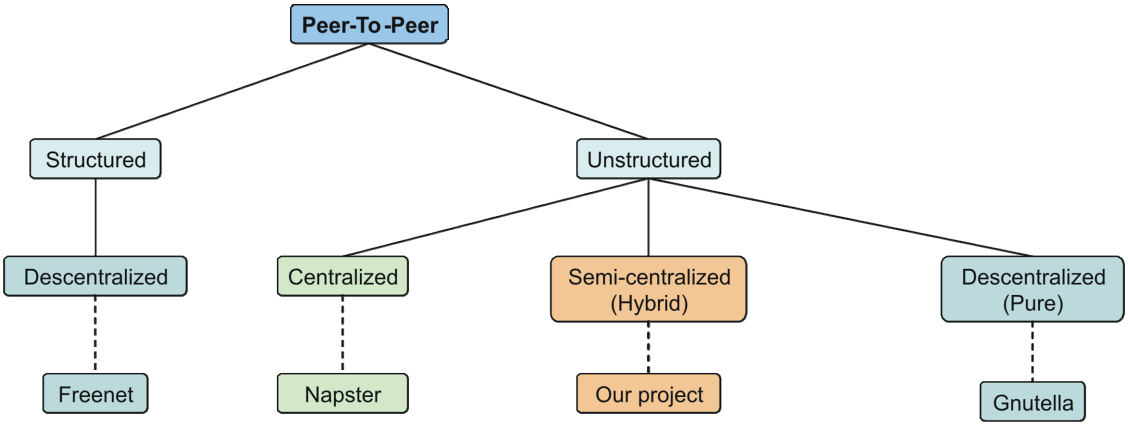


Figure 1: Peer-To-Peer taxonomies and example applications

We leave structured P2P aside because this approach requires high complexity algorithms, and also because we perceive the loose requirements of unstructured P2P as beneficial to practical network implementations. Furthermore, structured P2P places strict rules on where data should be placed and does not allow complex queries, so it is not suitable for generic searches.

The ability to make searches with incomplete file names, or even just file extensions is one of the strengths of our unstructured P2P network.

3 Session Initiation Protocol and the IP Multimedia Subsystem

In this chapter we overview the technologies on which we plan to build our test application and indeed our P2P network. We give a brief summary of their most important aspects, and comment mainly on those that we intend to use in our project.

3.1 Session Initiation Protocol

Session Initiation Protocol (SIP) is an extensively used protocol, defined in RFC 3261 [10] by IETF's network working group. It serves as means to establish, modify and terminate media sessions in a data network. By media sessions we can think of voice, video, data or any combination of them. SIP was later chosen by 3GPP as its standard protocol for session control in the 3rd generation mobile phone network.

SIP entities are called User Agents, of which we have two varieties; User Agent Client (UAC) and User Agent Server (UAS). Normally an application that uses SIP will contain both of them. The UAS will serve queries, that is, it will receive incoming SIP messages, while the UAC will send outgoing SIP messages.

In order to be able to receive messages, the UAS has to be registered to a SIP entity called SIP registrar. The SIP registrar can be collocated with the SIP proxy whose task is to relay messages between SIP user agents. The proxy will direct messages towards their destination since it can find out the IP address and port combination of all registered entities by querying the SIP registrar.

SIP itself is similar to P2P in the sense that very little intelligence remains on the proxy and registrar servers; SIP is then just the means by which peers are able to communicate and exchange session parameters.

Let's now review how a SIP call is made and which SIP entities are involved. We point out at this time that this example of SIP usage abides to the SIP standard. When developing our application we use SIP for transporting P2P control messages and, as it stands, this is not a standardized usage of SIP. However, since SIP is an evolving standard, it is conceivable that in the future we could see SIP extensions for P2P usage.

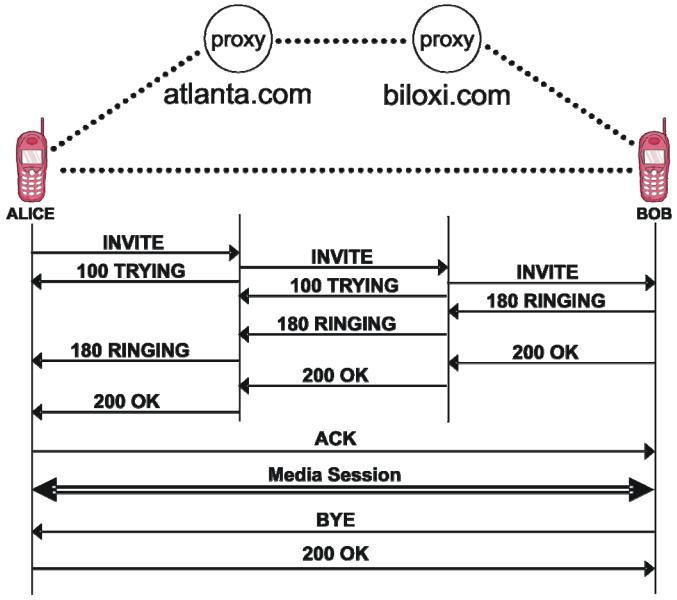


Figure 2: SIP session setup example with SIP trapezoid [10]

Figure 2 shows how a session is established using SIP, Alice is registered on the atlanta.com proxy and initially forwards messages to her proxy because she does not know Bob's IP address. SIP is the means by which she learns Bob's IP address, and other session parameters, for instance if we assume that this is a

voice call, Alice's phone has to learn which voice codecs Bob's phone can handle, and agree on one to be used for the duration of the session.

There is much to be told about SIP, but we only need to know the basic functionality to understand how we can use SIP for P2P. The reader should consult the appropriate RFC if interested on a detailed description of SIP. SIP has been supported by many companies who are themselves involved in the standardization and development effort.

3.2 IP Multimedia Subsystem

3GPP is the standards organization responsible for the further development of the highly successful 2nd generation GSM and the new 3rd generation UMTS. But the 3rd generation mobile phone network has been implemented differently to GSM. A redefinition of the architecture within the network itself was deemed necessary, and so it was split in horizontal layers as *Figure 3* shows.

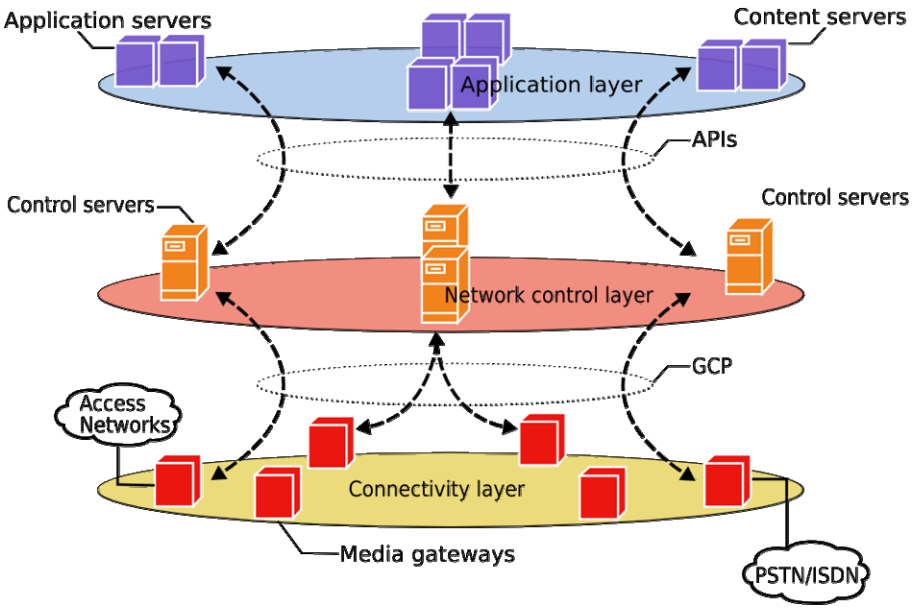


Figure 3: Horizontally layered network architecture [11]

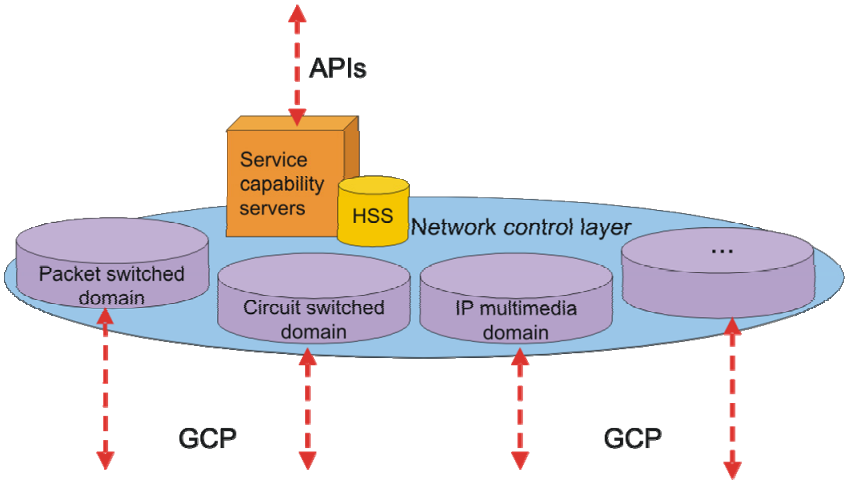


Figure 4: Domains in the network control layer [11]

This change in architecture required changes in the network nodes. Within the network control layer there are several "domains". These are shown in *Figure 4*.

Each domain has a server or servers within the network control layer that implement its functions. The circuit switched domain has the MSC server (Mobile Switching Center), the Packet switched domain has the SGSN server (Serving GPRS Support Node), and the IP multimedia domain has the CSCF server (Call Session Control Function). As we can observe from the figure the network control layer has APIs that define interfaces towards the application layer, and it uses Gateway Control Protocol (GCP) to connect with the connectivity layer. The Home Subscriber Server (HSS) will interact with all domains by storing and providing user information on request.

The IP multimedia domain is in fact the IP Multimedia Subsystem (IMS). IMS inner workings are beyond the scope of this work; suffice it to say that inside it SIP is used as the main protocol for establishing multimedia sessions between peers.

4 Mobile Peer-to-Peer

Mobile P2P is not new to scientific research, several projects have dealt with the subject, here we name the most relevant to our project and describe their work to some extent. We then lay out the theoretical background of our proposed mobile P2P network, afterwards we take a look at the state of the MP2P project when this thesis was started. Finally, we describe the MP2P architecture we intend to use.

4.1 Existing work on mobile Peer-To-Peer

4.1.1 Earthlink SIPshare

Earthlink is one of the largest Internet service providers in North America. Earthlink Research and Development (R&D) developed a proof of concept application known as SIPshare [12] which implements P2P over SIP using a Java application and an implementation of the JAIN SIP stack specification [13].

This is an interesting project and some of their ideas are valuable, for instance the peer discovery process through SUBSCRIBE/NOTIFY SIP methods. However, this application is only a proof of concept, because of this, SIPshare does not implement any functionality for MP2P nor does it consider a mobile client application. One aspect worth considering is that many new smart phones provide support for Java, however we did not test their program so we are not able to verify that their client program indeed works on a mobile platform. We must also point out that the developers do not make any claim regarding the suitability of SIPshare for mobile environments.

4.1.2 Symella and SymTorrent

Symella [14] and SymTorrent [15] are applications developed for the Symbian S60 platform and are designed to enable mobile phones to download files from Gnutella and BitTorrent, respectively. Although these programs partially inter operate with the above mentioned P2P file sharing systems, they do not allow the user to share content available from the MT. As such, these programs are used as file fetching utilities. A user wishing to share some file would first have to make it available through a Personal Computer (PC). Because of these reasons we do not consider Symella nor SymTorrent to be MP2P applications.

4.2 SIP based Peer-To-Peer in the IMS

From the background information presented for SIP and the IMS we can infer the possibility of a peer-to-peer network that operates "natively" in the IMS core.

Although SIP has not been specifically designed for peer-to-peer file sharing, this project shows the feasibility of such a network.

When the present work was started, there was already a working mobile client and a super-peer python script prototype was developed to mimic the behavior of the super-peer, which used an in-memory database.

There was, however, one obstacle that prevented direct TCP connections between mobile-peers; the mobile operator has in place a firewall that prevents the direct connection needed to achieve the download. Only outgoing TCP connections to the Internet are allowed from a mobile terminal. In order to bypass this restriction, a TCP relay was developed. The relay will thus match two

incoming TCP connections and relay the file as originally intended. One would expect that this relay will no longer be needed when a terminal is operating in an IMS network.

As such, the mobile-peer to super-peer connection was working as expected; relaying information from the registered users in the super-peer to those mobile-peers that sent queries. The objective was then to design a new super-peer which would increase the functionality of the previous implementation by relaying queries between super-peers in a peer-to-peer fashion when no matching files were found in the local database. What was also important was to evaluate the feasibility of using a standards compliant SIP stack without any modifications made to suit our project.

Also worth noting is that there is ongoing research that analyzes the possibility of implementing security features in this network.

4.3 Architecture of SIP-based Peer-To-Peer

Here we present our intended architecture for use in MP2P. This architecture was first proposed in [4] and was further elaborated in [1]. *Figure 5* details this architecture.

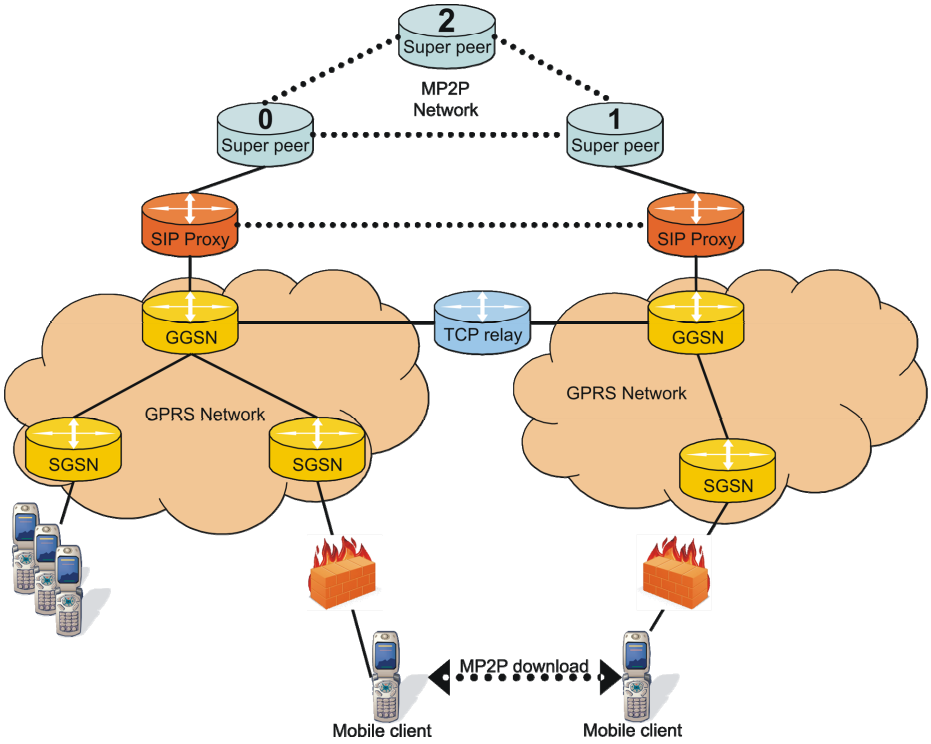


Figure 5: MP2P architecture

From the MP2P network point of view the mobile GPRS network is transparent, indeed the MP2P network does not make any special provision for accepting connections from different networks because SIP itself deals with Address Of Registry (AOR) as means of identification. The underlying protocols use IP addresses and UDP/TCP ports.

We depict the MP2P network as a separate network only for clarity reasons. MP2P network nodes can be deployed alongside GPRS nodes, and therefore reside within the mobile network itself, network operators can take advantage of this configuration to place control restrictions on which content is available for download. Another positive aspect of our proposed configuration is that network operators can place content available for download for a given price, they are in an ideal position to market such services and use MP2P flexibility for minimal configuration requirements.

Another aspect to consider is that 3G mobile network standards establish a "cleaner" interface to mobile network services; we described this interface in *Section 3.2*. It is possible then to implement the MP2P network as a mobile network service and take advantage of the available interfaces and control mechanisms in 3G. In such a scenario MP2P network nodes would become Application Servers (AS).

Finally, we note that our figure shows firewalls between the MT and the SGSN. This firewall is placed by the network operator and restricts the MT to only *outgoing* TCP connections. To bypass this restriction a TCP relay was developed to "bind" two outgoing TCP streams together, and in this way make the direct MT to MT file download possible.

5 Super-peer Implementation

In this Chapter we describe the high level implementation details of the super-peer which is an essential node to the MP2P network. We show the interaction between network entities in our network. Finally, we describe some aspects of our database.

5.1 Overview

The primary function of the super-peer is to interact with mobile-peers by:

- Updating and maintaining a file share database according to messages received from the mobile-peers.
- Answering queries from the mobile-peers.
- Forwarding queries to other super-peers in the case that the local database has no files matching the specified criteria.

The current implementation of the client software on the Symbian platform served as a reference model, as such, specifics about the interface were known beforehand. There were however several decisions to be made regarding the architecture and choice of software.

Some important considerations are worth mentioning. The use of SIP as the signaling protocol for this project is justified on the grounds of compatibility with existing and future 3G networks which use IMS as their packet service delivery platform. When analyzing the viability of this project from the perspective of network operators we can further support our choice by noting

that in using SIP we can directly allow a more thorough supervision of the content that is allowed into the network.

This peer-to-peer mobile system uses only a subset of all SIP methods. *Table 2* summarizes not only those methods used but also some error messages that can be seen during our network's operation.

SIP method or response code	Direction	Type	Usage	Content type
REGISTER	MP→proxy SP→proxy	request	Registration to proxy	
MESSAGE	MP→SP	request	Content update to super-peer	Text/xml
INVITE	MP→SP SP→SP	request	Query to super-peer	Text/xml
OK	proxy→MP proxy→SP	response	Proxy response to register	
ACK	SP→SP	response	Stack response to invite	
606	SP→SP	response	Query response	Text/xml
408	SP→SP	response	Proxy response "query timed out"	
480	SP→SP	response	Proxy response "temporarily unavailable"	
482	SP→SP	response	Super-peer found a loop	Text/xml

Table 2: SIP methods used in the MP2P network

We point out several things from this table. The direction of the SIP method is just an indicator of its intended destination. Even when there is a label indicating direct communication between super-peers all messages go through our proxy server. Another important point is that the super-peer should handle all error messages and reply to the mobile-peer either by including the files found or by stating that no files were found. Our mobile client software does not distinguish between SIP error codes.

5.2 Interaction between super-peer and mobile-peer

Upon start up every SIP peer has to perform a registration procedure. The super-peer does so automatically while the client software needs to be setup

appropriately with the AOR of the proxy. Registration consists of sending a REGISTER message to our SIP proxy and if the proxy accepts the registration, we will get an OK message back. After that, communication can begin between the registered entities. Registration expires after 70 seconds have elapsed; then it has to be refreshed in the same manner as the original process.

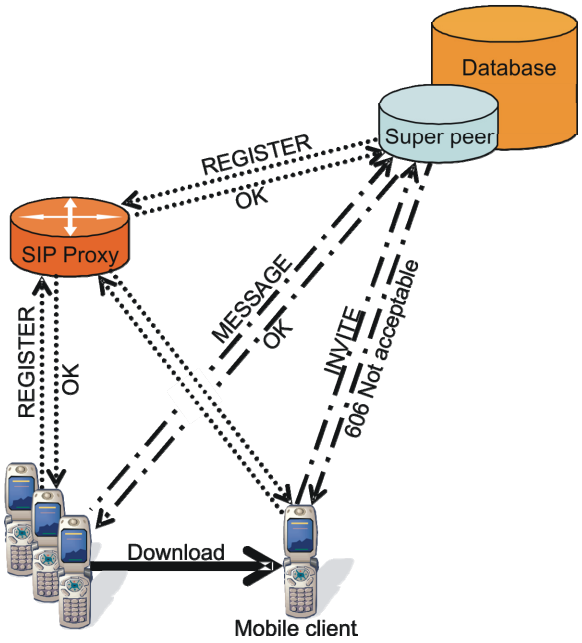


Figure 6: Basic message exchange between mobile-peers and super-peers

In *Figure 6* we indicate this registration procedure with a continuous dashed line. Once registered, the mobile client will send MESSAGE messages to the super-peer that has been previously specified in the configuration options. These messages will contain XML documents in the body. Within the message body, actions are specified in order to update database entries. In effect, this MESSAGE constitutes a second registration in which the mobile-peer registers to the super-peer so that it knows about it.

In *Figure 7* below we can observe a basic XML document which would be included in a MESSAGE method as part of a file update from a mobile-peer.

```
<?xml version="1.0" standalone = "yes" ?>
<contentupdate>
        <clearall/>
        <add>
                <type>File</type>
                <hash>3e8ec29cf981e55e240eb9dac11a4c98</hash>
                <cluster></cluster>
                <name>music.mp3</name>
                <extended type="size">6220486</extended>
        </add>
</contentupdate>
```

Figure 7: XML body of a MESSAGE method (used for file updates)

Our message shows some header information regarding XML version being used which will be common to all our XML messages.

XML messages have a hierarchical structure; *contentupdate* is the "root" of this particular XML "document". File information is a "child" of the *add* tag which itself is a "sibling" element of the *clearall* tag.

The file information matches the structure of the MySQL database which we discuss in *Section 5.4*. For every *contentupdate* "element" (such as *add* and *clearall*) indicated in the file, a corresponding SQL query will be sent to our database. Finally, the MESSAGE method will be answered with an OK message.

Every peer will update its own file information at will since files can be added or removed from the shared folder on the phone client at any time. Once the MESSAGE method is processed, each client will have information about their

shared files uploaded to the database where they will be available for other peers.

When looking for a file, a client sends an INVITE method to the super-peer. Attached to the body will be an XML file that has a different structure to that used in MESSAGE method. Searching for files once the INVITE query is received, is done by making an SQL query looking for the given string in a file name. An example is presented in *Figure 8*.

```
<?xml version="1.0" standalone = "yes" ?>
<request>
        <name>Tkk</name>
        <extended type="size">0</extended>
        <extended type="date">01.01.1900-01.01.2100</extended>
</request>
```

Figure 8: XML body of an INVITE method (used to query the super-peer)

We specify that this is a query message using a request tag. Using XML hierarchy we place three tags as "children" of our request. These three children tags contain information about the file we are looking for.

The answer is straight forward, in the case there is no file found in the database the answer will be that in *Figure 9*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<reply status="Not found">
</reply>
```

Figure 9: XML body of a 606 response (used when no files have been found)

We observe that "Not found" is not a tag but an attribute of our "reply" tag.

And when there is one or more matches to the query there will be an answer with a structure similar to the one shown in *Figure 10*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<reply status="OK">
        <content type="File">
            <hash>a796b38702ffe56e22ae20674ee05061</hash>
            <name>TkkSept.jpg</name>
            <extended type="size">26770</extended>
            <cluster></cluster>
            <location>
                <username>sip:kenny@mp2p.netlab.hut.fi</username>
                <ipaddress>130.233.154.182</ipaddress>
            </location>
        </content>
</reply>
```

Figure 10: XML body of a 606 response (used when files have been found)

We note that the status attribute is OK which means the super-peer found files matching our search criteria, also note how there are several levels of XML hierarchy in this response.

Once the user decides to download one file, it is done by just selecting the name of the file from the different files shown that are available for download. The download process is indicated in *Figure 6* by a thick solid line. In practice however, there is one glitch that has to be overcome; the mobile network provider that was used for the trial has a firewall that prevents direct TCP connections between the mobile phones. For this reason a relay script was previously developed to link together outgoing TCP links to the relay matching connections by checking the hash of the file being exchanged.

5.3 Interaction between super-peers

The basic functionality section above describes how peers interact between them when they are registered to the same super-peer. However, when there is a query that does not find a match in the local super-peer database it is then forwarded to other super-peers. The super-peers form together a peer-to-peer network for relaying these queries.

What was shown in *Figure 6* as a super-peer / database pair will, from here onwards, be shown only as one entity; namely the super-peer. It is possible to have several super-peers share the same database or even have the database in a separate node, however from a Peer-To-Peer stand point there is no distinction between these alternatives. The proxy server can also be collocated with the super-peer, this is the configuration we used to avoid using additional resources, however this will not necessarily be a good choice for a "live" super-peer because we have to consider that the proxy may not only handle MP2P messages but also regular SIP messages. For practical reasons we no longer show neither the proxy nor the registrations to it in our diagrams.

Figure 11 shows how a super-peer would forward queries to its neighbor super-peers when it finds no matching file in its local database to a specific query. In such a situation the final 606 answer to the mobile client would have to wait until answers from all neighbor super-peers have been received. Note that there are mobile-peers that are shown nearby super-peers 1 and 2. These mobile-peers do not take an active role in answering queries because their shared file's information is already stored in their respective super-peer's database.

If there are files found for a given query, the querying mobile-peer can initiate download of the file it desires by establishing a TCP connection to the mobile-peer that owns it.

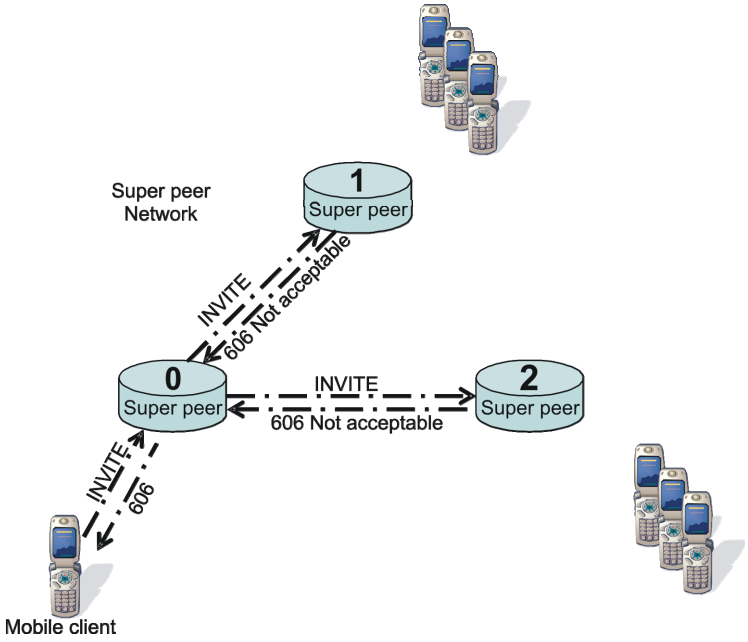


Figure 11: Relying queries in the super-peer network

5.4 Database

One of the most important aspects that is key to the performance and scalability of the super-peer is the database. *Table 3* shows the basic structure of our database table.

Table field	MySQL type	Default value
Name	Varchar (50)	None
size	Mediumint unsigned	None
aor	Varchar (50)	None
time	Timestamp	Current_timestamp
hash	Char(32)	None

Table 3: Database fields, types and default values

This table has the minimal information required for our network's operation. The name is a variable length field with up to 50 characters. Size is an unsigned

medium size integer, its value can be up to 16777215, and since this value is intended to represent the file size in Kilo Bytes (KB) we considered it to be large enough for our purposes. AOR is our SIP identifier; it is of the same type as the name field. Time is a timestamp field that will tell the super-peer the date and time any file information was last updated. The time and date will automatically be assigned by the database at the time a transaction is executed; this is made possible using the *current_timestamp* default value for this field. The last field is the file hash, this is a 32 character long field that uniquely identifies a file in the MP2P network. However, because we wish to allow the same file to be available from two or more different MTs we will use two fields as the "primary key" for our database. Those two fields are the file hash and the AOR of the mobile-peer that owns that file.

The database itself can be used on the same physical node or operated remotely as the MySQL API used allows it.

6 Super-peer software

In this chapter we discuss our software library selection, we then proceed to elaborate on our software architecture, and also comment on each of our software modules. Finally, we detail the major difficulties we faced during code development.

6.1 Choosing software libraries

The initial programming language of choice was C++. Also the test platform that was already operating used a Linux host; we continued working on that platform for this project. The need then was to find a suitable C++ SIP stack that was open source and that provided a stable platform onto which we could build the application. Several stacks were considered, an important consideration was that the code should be stable enough for our purposes and that there was enough usage of the stack so that we could find appropriate support when needed. One short document which considers several stacks and compares them is [16].

Features	Sofia-SIP	oSIP	reSIProcate	sipX	Opal	Vocal
Win32	*	√	√	√	√	**X**
Linux	√	√	√	√	√	√
TCP	√	*	√	√	√	√
UDP	√	*	√	√	√	√
TLS	√	*	√	*	√	**?**
Footprint	**<500KB**	**~400KB**	**<2.5MB**	**<4MB**	**?**	**?**
License	LGPL	LGPL	Vovida	LGPL	MPL	Vovida

Table 4: C++ SIP stack comparison [16]

Table 4 shows the SIP stacks we considered for our project. Below we detail the meaning of the features (rows):

- **Win32 and Linux:** indicate which stacks are available on each platform.
- **TCP/UDP/TLS:** indicate the available transport options each stack provides.
- **Footprint:** shows the RAM memory consumption of each stack during normal operation.
- **License:** shows the license under which each stack is distributed.

After considering our options we decided on reSIProcate(resip) [17]. Resip complies with most relevant RFCs and can be compiled under Windows or Linux among other platforms. Resip has the advantage of a relatively active mailing list where we could find support about specifics of its usage.

The client software uses XML to specify the contents of information to be exchanged between peers. For that purpose Tiny XML parser [18] was chosen mainly because of its small memory footprint and ease of use. Tiny XML is also open source software and is distributed under zlib/libpng license.

The consideration of implementing an in memory database as it was done in the preliminary version of the super-peer was further discussed and a decision was taken to use a more robust implementation that could scale to host a large number of client terminals. It was very straight forward to select MySQL [19] since there is already one C++ API for interfacing with our super-peer, MySQL++ [20].

6.2 Architecture

The software is divided into three modules. The main module owns all the SIP stack processes and communication states. The main module interacts with the data base handler (second module) which provides the interfaces needed to store and access data file information into a previously set up database. The database is initially designed for simplicity but further optimizations that would require changes in both the MySQL database and the super-peer code are both possible and desirable if we are to look for optimum performance.

The third module will handle communications between super-peers in order to relay queries for which local peers (the ones attached to a common super-peer) do not have relevant files. It is in this module that actual peer-to-peer functionality is implemented.

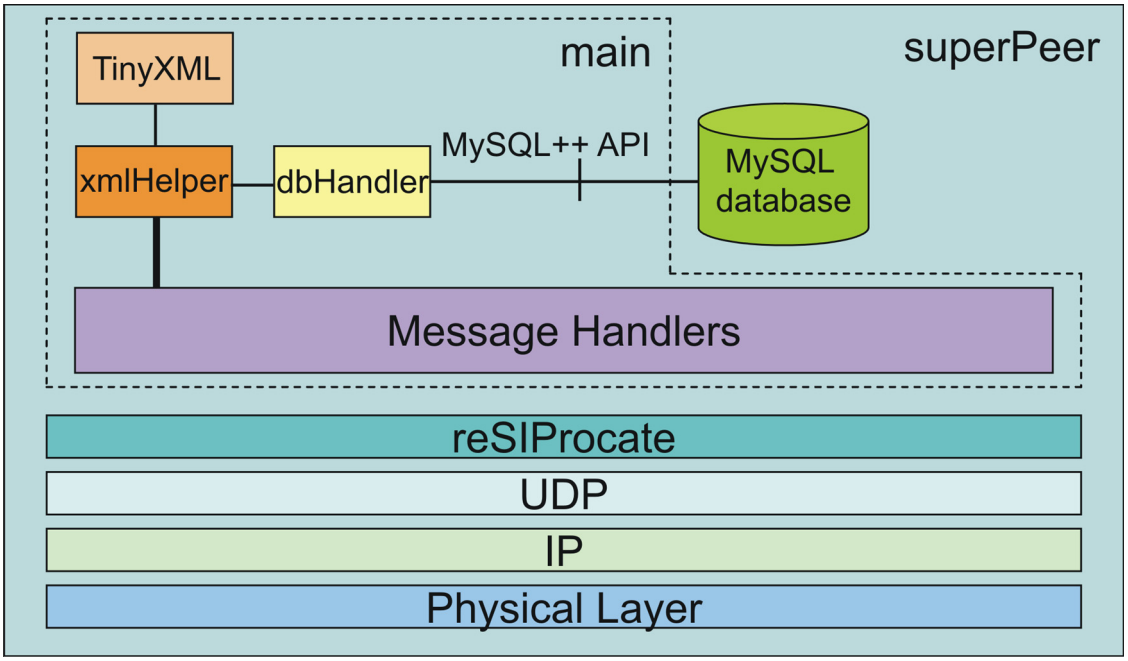


Figure 12: Architecture of the super-peer

Figure 12 shows the relationships between modules and how they are related to each other and to the SIP stack.

The main module interacts with most other modules in some way and is shown as a dashed line. The second module is the database handler shown in yellow, the xmlHelper and TinyXML are related to, and assist the database handler. The third module is embedded into the message handling code, shown in purple.

6.3 reSIProcate

This high performance SIP stack was chosen for our implementation. *Table 5* shows some features of the reSIProcate project.

Stack features
> 1000 TPS
< 500 KB memory footprint
Good security
Multi platform
Test framework
Support for most relevant RFCs
Active newsgroup
Active development
Dialog Usage Manager (DUM)
Repro (SIP proxy/registrar)
Rutil (various utilities)

Table 5: reSIProcate feature list

At the time of this writing resip is very much alive, during the course of the present work there were two major releases and two minor ones. Additionally there is some ongoing discussion about the possibility to include reSIProcate as a standard Linux Debian package, which makes it even more compelling.

6.4 Module walkthrough

The super-peer program consists of several modules, what follows is an overview of the functionality implemented in each of them.

6.4.1 superPeer module

It is the main module which includes all other modules. It implements the **superPeer** class. In this module the needed stack and DUM components will be instantiated and initialized as required.

6.4.2 dbhandler module

It is the main interface to the MySQL database, and contains all the classes needed to use, execute and store queries. It also contains the query templates that

will be completed and used at execution time. This module uses the MySQL++ library.

This module implements the **dbHandler** class which will be instantiated by the super-peer. Adding functionality to this class is straight forward due to the flexibility provided by SQL. Storing and retrieving file information is done by using this module's functions, for instance, looking for a file can be done using the **searchByName** function.

There are other functions that can serve for debugging purposes; if we wanted to print all the database contents to the screen we could do so by using **getAll** and **printResult** functions.

6.4.3 dbutil module

This module contains some SQL result printing functions and also the vital connection function that will communicate with the database. This module uses the MySQL++ library.

6.4.4 xmlhandler module

This handler contains the functions necessary to parse the XML body contained in the SIP messages. This module will include the **dbhandler** and **dbutil** modules in order to execute required transactions on the database.

Specific functions within the **xmlHandler** class are aimed to parse incoming content updates and store the given file information in the MySQL database. Correspondingly, there will also be functions to parse queries and replies.

6.5 Challenges during software development

6.5.1 Learning curve

Every software development effort faces difficulties, in our case there was a specific need to use open source software. It is a well known fact that many open source projects lack in the quantity and quality of the documentation provided.

The main challenge we were faced with was to understand the inner workings of the resip stack. Many hours were spent browsing and reading the stack's source code. The project's newsgroup is a good source of information. The main tasks a developer faces when learning resip is the correct usage of message handlers and dialogs, which are the way the stack handles SIP messages. This was the steepest learning curve we faced.

6.5.2 P2P message identifier

When forwarding a query message throughout the P2P network we need to have a unique message identifier that will help with message processing and loop detection; it would be desirable that such identifier is carried as a message header. The standard SIP header Call-ID is very well suited for this purpose since the SIP standard specifies that the value of this header *must be globally unique*.

We learned the hard way that the Call-ID cannot be copied directly from one message to another if they do not belong to the same dialog, the result of such transgression is that the stack will discard the message response as a stray response. It was thus necessary to add a user defined header as this is allowed by the SIP standards. In this user defined header we just copied the original Call-ID, so that any query traversing the P2P network has two identifiers; the Call-ID which has only local significance between two super-peers involved in

forwarding a message, and the user defined Message-ID which carries the Call-ID that identifies the original transaction between the MT and its associated super-peer.

6.5.3 PartySIP proxy vs Repro

Another lesson learned was that not all open source SIP proxies are standards compliant. For historical reasons we used PartySIP as our proxy since this was the one that was used for the old super-peer prototype. However as stated in *Section 6.5.2* we require a user defined header to be carried in all P2P messages. PartySIP did not work as expected; it was dropping our user defined header, contrary to the SIP specification [21]. We then tested repro, and it proved to be a good alternative which was quite easy to set up and manage, and does not alter our packets in any way.

6.5.4 Loop detection

During development of the software we had a problem when loop detection was not finalized; the problem manifested itself when forwarding a query for which none of the nodes had a matching file, in such a situation the query went on from one super-peer to the next going round in an infinite loop precisely because loop detection was not correctly implemented at that point.

6.6 Known bugs

There are a number of known "weaknesses" in the super-peer software at this stage of development.

6.6.1 Input checking

The input checking of the software is quite scarce. It can be made to crash by forwarding queries with non alphanumeric characters and line termination characters at the beginning of the query. This kind of crash is possibly related to

the mysql++ library but this has not been confirmed. Additionally, the input is not checked for SQL sequences which pose a security threat.

6.6.2 Memory management

The program contains several tables that are stored in dynamic memory because it is thought that the performance of the super-peer would be greater this way. However, the performance of these tables should be compared to having these tables stored in a MySQL database. If a super-peer would need to manage mobile-peers in large numbers, the memory consumption and performance of the tables could change. Further testing is needed to verify these scenarios.

6.7 Installing, configuring and running the super-peer

Several preparations need to be done before running the super-peer software. In our case we used a Linux Debian server. First we need to install the required libraries for running our SIP stack. Then we should download the SIP stack from [17] , and proceed to unzip the package, run the configuration script and compile it. We then need to make sure the resip libraries are up and running. Finally we need to compile and configure the super-peer program.

Additionally, we need to install and set up the MySQL database. Many Linux distributions come with MySQL installed or can be easily configured to install it during OS installation.

We provide instructions for many of these tasks in the Appendices.

6.8 Configuring and using the mobile-peer client

We provide a sequence of screenshots that detail the client's configuration and use. *Figure 13* shows the initial screen and, following the arrows takes us through the configuration procedure and registration to the SIP proxy. We can observe from the second screen in the sequence that the TCP relay must also be configured here. Additionally, the download folder and the shared folder must be specified. The screens from the second row show a search for all files containing the sequence "jpg". When we press OK in this screen the mobile-peer client sends the super-peer our query and the next screen shows that we got some results back. Now we can choose which one we want to download. The download process is shown in the last screen with a progress report. Lastly, the screen also shows the time it took from the point we sent the query to the time we received the answer.

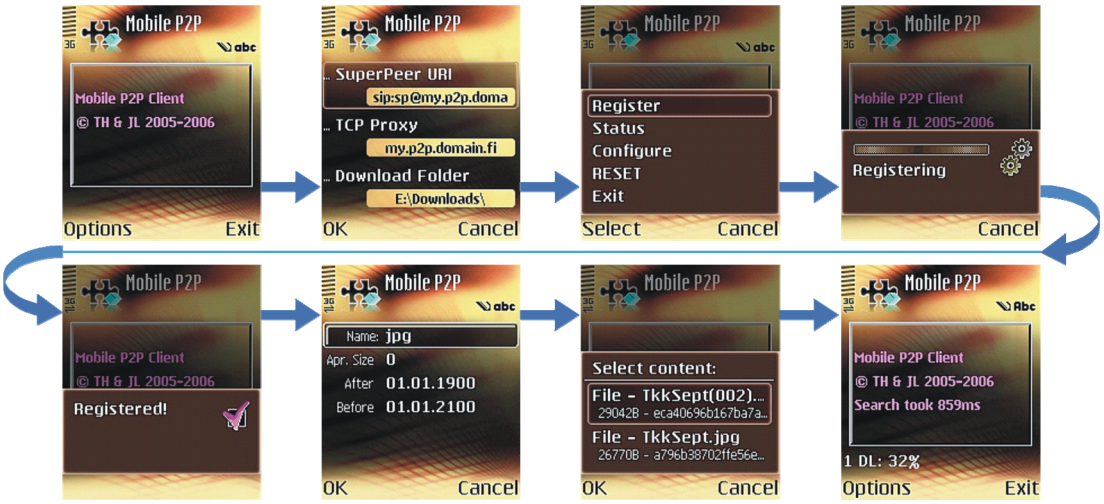


Figure 13: Configuration and usage of mobile-client

7 Testing and measurements

7.1 Testing environment

Table 6 shows a breakdown of how the super-peers were set up for the testing phase of this work and also some information about the mobile-peers and the equipment they were running on.

Entity	AOR	hardware	software
Proxy Server	my.domain.fi	Computer 2	Linux Debian 3.1
Super-peer Zero	sip:spzero@my.domain.fi	Computer 1	VMware server with Linux Fedora Core 5
Super-peer One	sip:spone@my.domain.fi	Computer 2	Linux Debian 3.1
Super-peer Two	sip:sptwo@my.domain.fi	Computer 1	VMware server with Linux Fedora Core 5
Mobile-peer Karl	sip:karl@my.domain.fi	Phone 1	Symbian 8.0
Mobile-peer Kenny	sip:kenny@my.domain.fi	Phone 2	Symbian 8.0

Table 6: Test environment

The following is a list of additional tools that were used during software development:

- Eclipse SDK environment [22] with plugins for C++ was used as development environment.
- WinCVS [23] was used for version control.
- Wireshark [24] was used for packet capture and analysis.

7.2 Test cases

For this test we decided on four basic test cases which show the functionality of the super-peer. These test cases are shown in *Figure 14*. Test case one and two show the very basic operation of the super-peer; answer queries on its own and relay to other super-peer when it cannot find a match in the local database for a query. Test case three and four show how the complexity increases when there are more super-peers present; super-peer zero must wait for the messages to be flooded to the network and for all answers to arrive in order to answer the

original query. Note that in test case four there is a loop. Loop detection is an important part of the flooding algorithm.

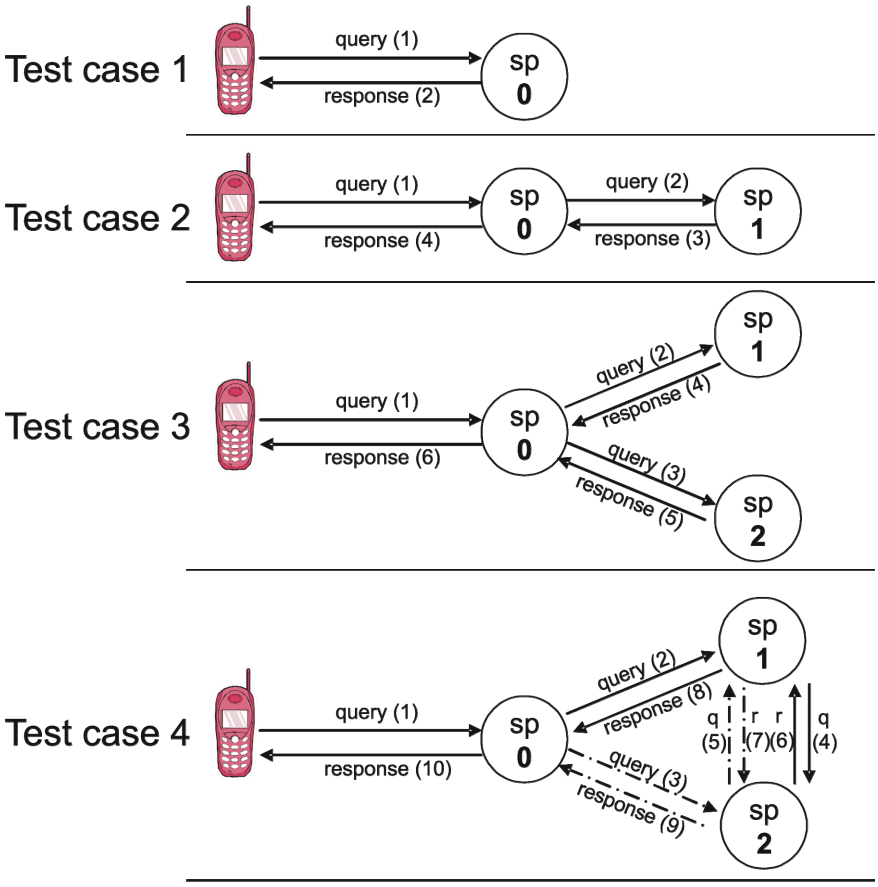


Figure 14: Test cases

By measuring performance of these test cases we can get some idea of the end user experience when using our application. One modification was required of the client software; we needed to set up some timer in the user interface module in order to measure end user delay.

7.3 Packet capture

To better exemplify the behavior of our super-peer let us analyze a real-time packet capture sample that we took when testing the application. This packet capture is shown in *Table 7* and was taken using test case 4 configuration.

Time	Source	Destination	Info	Extended Info
14.74368	199.299.154.182	199.299.154.52	Request: INVITE	sip:sptwo@199.299.154.52:12005; rinstance=8ebf455f4f8c925f
14.88827	199.299.154.182	199.299.154.36	Request: MESSAGE	sip:spzero@199.299.154.36:12005; rinstance=ff419af90789d179
14.92697	199.299.154.52	199.299.154.182	Request: INVITE	sip:spzero@my.mp2p.domain.fi
14.93014	199.299.154.182	199.299.154.36	Request: INVITE	sip:spzero@199.299.154.36:12005; rinstance=ff419af90789d179
14.98232	199.299.154.52	199.299.154.182	Request: INVITE	sip:spone@my.mp2p.domain.fi
15.01716	199.299.154.36	199.299.154.182	Status: 200 OK	
15.02745	199.299.154.36	199.299.154.182	Request: INVITE	sip:spone@my.mp2p.domain.fi
15.06277	199.299.154.182	199.299.154.36	Request: INVITE	sip:spzero@199.299.154.36:12005; rinstance=ff419af90789d179
15.06553	199.299.154.182	199.299.154.36	Status: 482 Loop Detected	
15.06756	199.299.154.36	199.299.154.182	Request: ACK	sip:spone@my.mp2p.domain.fi
15.06763	199.299.154.36	199.299.154.182	Status: 606 Not Acceptable	
15.07049	199.299.154.182	199.299.154.36	Request: ACK	sip:spzero@199.299.154.36:12005; rinstance=ff419af90789d179
15.07050	199.299.154.182	199.299.154.52	Status: 606 Not Acceptable	
15.12129	199.299.154.52	199.299.154.182	Request: ACK	sip:spzero@my.mp2p.domain.fi
15.12523	199.299.154.36	199.299.154.182	Status: 482 Loop Detected	
15.12792	199.299.154.182	199.299.154.36	Request: ACK	sip:spzero@199.299.154.36:12005; rinstance=ff419af90789d179
15.13424	199.299.154.182	199.299.154.52	Status: 606 Not Acceptable	
15.17385	199.299.154.52	199.299.154.182	Request: ACK	sip:spone@my.mp2p.domain.fi
15.17392	199.299.154.52	199.299.154.182	Status: 606 Not Acceptable	
15.17655	199.299.154.182	199.299.154.52	Request: ACK	sip:sptwo@199.299.154.52:12005; rinstance=8ebf455f4f8c925f

Table 7: Packet capture for test case 4

This table illustrates how the message sequence takes place when a node queries for a file that is not found in any super-peer of the MP2P network. From this sequence there is only one message (the second one) that does not correspond to the query process. The MESSAGE method is sent periodically by the MT to keep alive its connection state at the mobile network operator's firewall. The MESSAGE is answered with an OK method which is sent by our proxy.

We illustrate the same packet capture in *Figure 15* below. What is important from this figure is that we can observe how the flooding and the loop detection take

place. The vertical axis represents time, so we can also draw some conclusions about how the actual message sequence takes place.

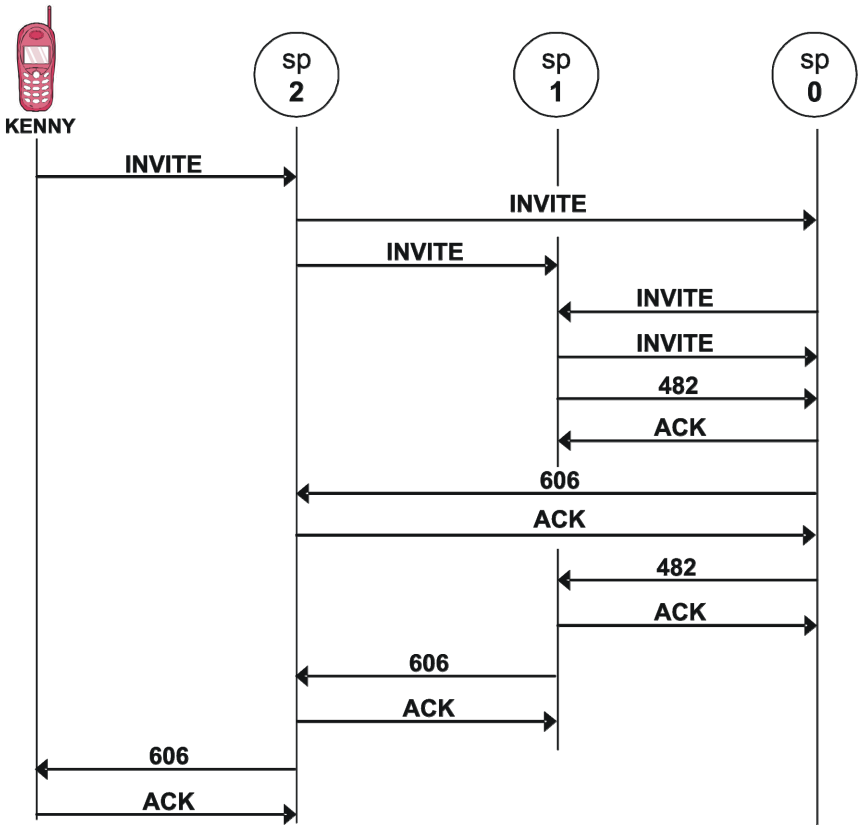


Figure 15: Message sequence diagram for test case 4

Kenny (MT) has already registered to super-peer 2 (SP2) by sending a MESSAGE (not shown in the diagram), he then decides he wants to look for some file, this is done by sending an INVITE to its associated super-peer (super-peer 2 in our case). SP2 parses the query and searches its local database looking for the file Kenny wants. In this instance SP2 does not find any matching file so it queries SP0 first and then SP1 by sending a similar INVITE to the original it received from Kenny. The reason SP2 forwarded the query to both SP0 and SP1 is because both are defined statically in a super-peer table at SP2, that is, SP2 is a neighbor super-peer of SP0 and SP1.

Independently SP0 and SP1 look for the file and realize there are no matching files in their local databases. SP0 forwards the query to all its neighbors except the one it received the query from, in this case SP1. Note that SP0 has no idea that SP1 already received that same query. Using the same reasoning SP1 forwards the query also to SP0. SP1 informs SP0 that a loop was found by sending a 482 (loop detected) message. SP0 answers with an ACK message.

At this point SP0 has run out of neighboring super-peers and replies SP2 with a 606 message stating that no files were found, the message body corresponds to that shown in *Figure 9*, SP2 acknowledges by sending an ACK message back to SP0. Then the same sequence takes place at SP1 first a 482 message for loop detection followed by another 606 message sent to SP2.

Now SP2 "knows" that the file Kenny is looking for is not available in any of the super-peers of the network, and replies with a 606 message informing Kenny of that fact.

7.4 Measurements

The measurements we made are shown in *Figure 16* and *Figure 17* below. While in the first graph we differentiate between queries that resulted in a file found and queries whose result was empty, in the second graph we take them as a whole set of trials for every test case (i.e. no differentiation between queries that returned results and those that didn't).

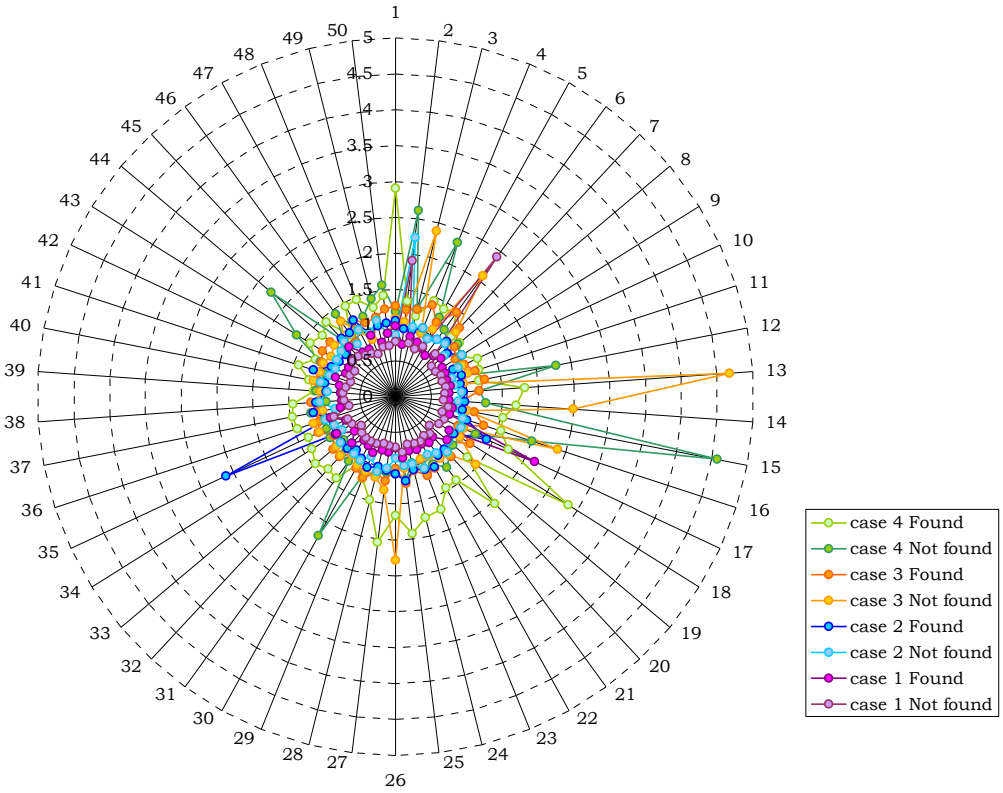


Figure 16: Radial graph with 50 trials per test case

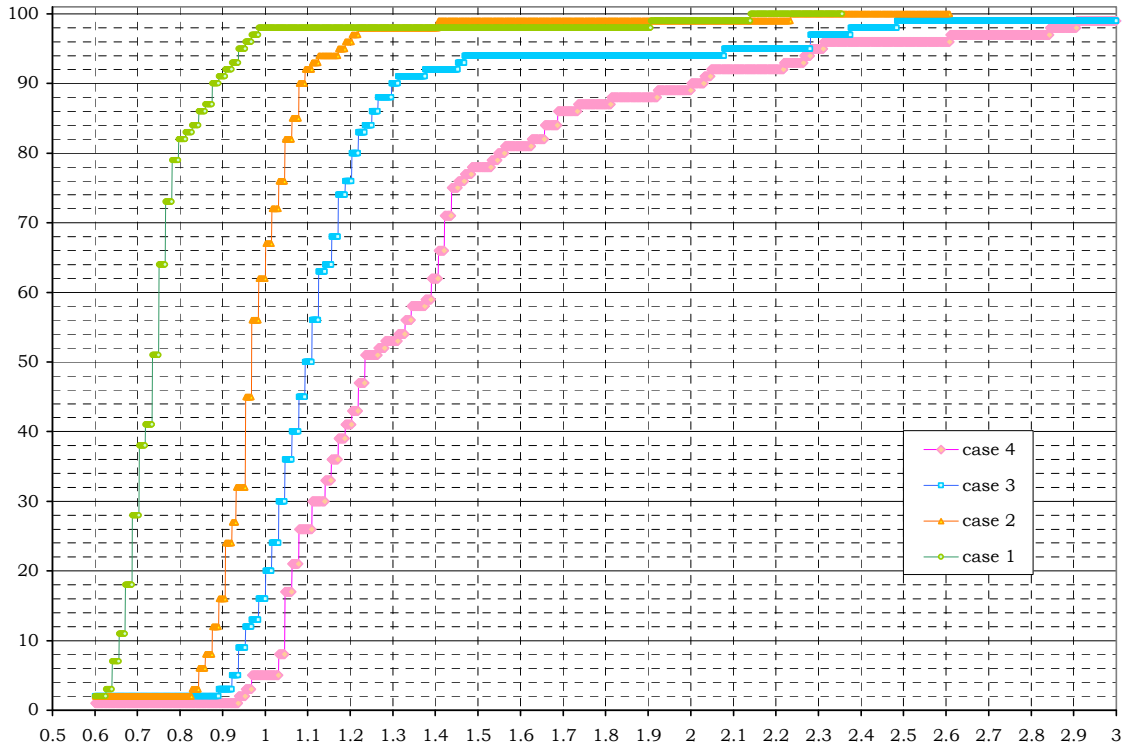


Figure 17: Cumulative delay chart with 100 data points per case

From *Figure 16* we can observe that the great majority of trials made return an answer within 1.5 seconds from the time the query was sent. We would like to emphasize here that this measured delay is in fact end user delay, as measured from the MT and includes processing delay in the client, transfer delay within the mobile network and of course processing delay at the super-peer and transfer delay within the P2P network.

One more thing we can notice from the radial graph is that because the trials are shown in the order they were made there is a clear correlation between queries that took more than 1.5 seconds to complete, that is, if one query takes more than 1.5 seconds to return an answer then it is likely that the next one will also take 1.5 seconds or more to complete if it is done immediately after the first one. This is probably due to mobile network congestion at those specific moments in time.

Figure 17 shows interesting figures for the delay in every test case. We observe that around 90% of the queries get an answer before 0.9, 1.1, 1.3 and 2 seconds for the 1st, 2nd, 3rd, and 4th test cases respectively. Also worth noting is that when our P2P network grows in complexity our curve gets more distorted because transfer delays between super-peers are more prominent in the overall delay.

These are quite reasonable performance figures; we can comment that, from a user's perspective, the delay at the MT is acceptable.

7.5 Super-peer stress test

In addition to the testing of the network as described in previous sections of this chapter we tested the super-peer in an attempt to find out whether it scales to answer many consecutive queries. For this purpose we used SIPp [25]. SIPp is an open source SIP call generator software that is perfectly suited for stress testing SIP implementations.

For the purposes of this test we set up two new virtual machines running Linux Debian 4.0. We disabled the peer-to-peer functionality of the super-peer and also the registration process. We then generated a comma separated value (CSV) file which includes more than 750 unique names and query words. The names and words were chosen randomly from a database and placed in the CSV file. We then configured SIPp to send queries at different call rates. *Figure 18* shows our setup. In the Appendices we include the input files needed by SIPp to run this scenario.

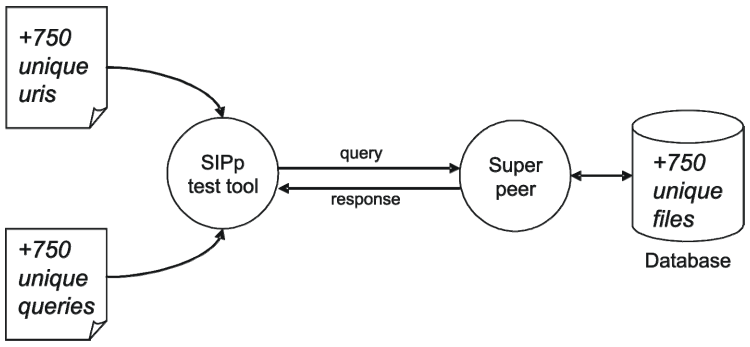


Figure 18: Stress test setup

We found the performance of the super-peer to be low for our current setup. Our results indicate a maximum of 10 Calls Per Second (CPS).

A lot of effort went into finding the root cause of the performance problem. Since the stack is capable of greater than 1000 CPS, and several benchmarks show MySQL to

achieve very high performance, we inferred that a problem was present either in our source code or in the stack.

The problem manifests when going higher than 10CPS which triggers retransmissions of 606 messages. These retransmissions get more and more frequent the higher call rate is, finally "drowning" the super-peer in them. In this state it is no longer possible for the super-peer to return answers to the queries because all the time is spent retransmitting previous 606 messages.

At the SIPp test tool these 606 retransmissions are discarded as "out of band" messages, and rightfully so, since the SIP dialogue was terminated once the ACK message is sent in response to a 606. The problem solution then lies in determining why the ACK messages seen on the wireshark trace arriving at the super-peer machine are not processed correctly and are being silently dropped (no error messages appear in debug level logs).

It was initially assumed that the reason for not processing the acknowledgments in time was that the higher call rates would generate a heavier load on the super-peer since they have to be processed sequentially because of our software design. The ACKs would then have to wait in the stack queue for longer than 500 ms which is the default T1 timer that determines, among other things, the retransmission timeout (RTO). This was not the case however, since we recompiled the stack with a T1 timer set to 1000 ms with the exact same results.

One possible reason for our problem that was suggested by one of the stack's developers was that the 606 could somehow be processed wrongly by the stack since 6xx messages are not commonly used in normal circumstances and he recommended against their use. In any case this issue deserves more time and effort to find a solution.

8 Conclusions

Nowadays mobile phone users who want Internet connectivity have several options. They can use GPRS or EDGE connectivity paying premium rates if they want fast access or be content with low bitrates and high delays. 3G, IMS and more recently HSDPA are promising to change all this with high bitrates and always-on Internet connectivity. However, there are steep costs for network operators to upgrade their networks and the users themselves have to be willing to pay substantial amounts of money for a 3G handset. Mobile Peer-To-Peer (MP2P) is the mobile counterpart of P2P networks that have proven highly successful for PCs. MP2P could potentially be a driving force for 3G service adoption by providing new media for end user consumption.

The present work was devised as a means to prove the feasibility of a new type of MP2P network based on the Session Initiation Protocol (SIP) standard that is supported in modern 3G mobile networks. This MP2P network works by relaying P2P control messages using SIP; this approach is called P2P-Over-SIP.

By combining P2P-Over-SIP and a semi-centralized P2P architecture we achieve compatibility with 3G networks and relieve MTs of having to route MP2P messages with the associated costs in battery, memory and bandwidth consumption.
In a semi-centralized architecture MTs are not involved in routing P2P traffic but instead only specialized peers (super-peers) with more resources are given this task. In this way we transfer the MP2P traffic from the expensive Mobile Terminal-Radio Access Network (MT-RAN) air interface to the mobile core network where bandwidth is cheaper.

The present work centers around the software development of the super-peer MP2P network node which is tasked firstly with maintaining state of a subset of the mobile-peers and handling their queries, and secondly with forwarding any query for which an answer in the local database is not found. The fundamental concepts and system architecture for this project were defined during previous research projects at Helsinki University of Technology's Networking laboratory. A Symbian mobile client had already been developed and the present work utilized this code mostly unchanged.

When forwarding queries between super-peers we do so using P2P forwarding algorithms. In our case we used flooding for its simplicity but other algorithms could easily be implemented. We discussed several of those algorithms and identified Routing Indexes as a viable alternative to flooding.

We detailed the software architecture design of the super-peer and described its main components. Finally, we tested our application on four different test cases and performed delay measurements for all of them. The super-peer software developed for these tests functioned as expected. We tested the application with two mobile clients querying each other's contents repeatedly and changing the point of attachment to the MP2P network by changing the super-peer settings on the client software. The results were satisfactory since the end user's perceived delay was quite low for most requests (below 2 seconds for the selected network configurations).

The stress test we performed revealed some performance issues that need deeper investigation. The current performance limit of our super-peer was shown to be 10 CPS. However we feel that given the proven performance of the reSIProcate SIP stack of greater than 1000 CPS and the high performance of MySQL (especially for the relatively small table sizes we use), are indications that much higher performance is possible.

Regardless of these issues, we feel that this work has proven the feasibility of a P2P-over-SIP network for mobile terminals.

From our accumulated delay results we can see an inflexion point at around 80 percent of all messages for every case. We observe a difference of approximately 0.8 seconds between first and last test cases at this point. We can extrapolate this result and predict that a network with ten to twenty nodes using our current configuration would still yield reasonable delays below 10 seconds for most queries.

Further performance improvements gained from using improved forwarding algorithms could be of at least an order of magnitude in mean convergence delay. We expect our server to perform 1000 CPS once the performance issues are resolved. This means that we consider the SIP stack to be the performance bottleneck of the whole MP2P system. We could again extrapolate this to assume one super-peer would be able to serve at least 5000 clients, considering that the transactions will not only be direct queries from mobile-peers but also P2P transactions between super-peers.

With these numbers we can claim that a city the size of Helsinki with around 1 million inhabitants and a mobile phone penetration of around 100% could be served at least initially by our MP2P network if we assume that less than 10% of its mobile subscribers are interested in the service. This means that around 100,000 MP2P clients could be served by around 20 super-peers located strategically in the mobile network. A “live” trial would be ideally suited to verify these claims.

8.1 Future work

Since our goal was to implement a prototype, some optimizations and features were excluded. Several improvements to the super-peer program are feasible and reasonably straight forward.

- Memory management of mobile-peer, super-peer, ongoing-queries and ready-answers tables could be improved.
- Some basic database optimization could be done to improve the query performance of the super-peer. For example the fields could be sorted so that lookup is faster.
- Improved forwarding should be considered, Routing Indexes (RI) seem to be a good and simple enough option but there are many other options. The tradeoff is simplicity and speed against complexity and improved resiliency. If RIs are used the database must be categorized and split into several database tables. The simplest categorization could be; music, video, pictures, pdf files, etc.
- Computing resources of the super-peer should be taken into account, and effects of disk and memory access performance against heavy load, should be studied.
- Security features could be implemented. There are ongoing research projects at Helsinki University of Technology – Networking Laboratory that investigate this topic.
- The performance issues we encountered should require relatively low effort to be resolved. In resolving these issues more testing is needed as well as revising the super-peer's queries and answers table's performance and possibly redesigning those could be necessary. Also an important source of assistance is the reSIProcate developer's mailing list.

9 Appendices

9.1 Compiling hints for the super-peer

Prerequisites for compiling the super-peer binary

The following libraries were compiled and configured in the testing platform.

a. ARES

b. Resiprocate 1.1

c. MySQL 5.0.22 (server and client)

d. MySQL++ 2.2.2

e. Standard development libraries (gcc, g++, etc.)

Note:

Ares comes with resiprocate distribution.

MySQL++ needs the client and development libraries installed.

Both gcc and g++ are needed to compile resiprocate. In Debian there is a dummy library that depends on the versions installed of each of them. These dummy packages need to be installed as well.

9.2 Installing reSIProcate

Installation considerations

- We need to have ARES DNS library installed.
- We need **gperf** library.
- We need to have **openssl** library installed ***and*** its development package also installed. (Note: In Fedora Core 5 development package is called **openssl-devel** while in Debian it is called **libssl-dev**)
- We need to have **popt** library installed ***and*** its development package.
- We need to have **db4** installed ***and*** its development package . (Note: in file repro/BerkleyDb.hxx, we need to change #include db4/db_cxx.h to #include db_cxx.h)
- After installing all required libraries we can type:
    - ./configure
    - make
    - make install

Installation notes

The test subdirectories are not compiled when typing make. After doing make install we need to create etc/ld.so.conf including the line /usr/local/lib, and then do ldconfig.

9.3 Running the super-peer

Prerequisites for running the super-peer binary

For the super-peer program to work we need to take care of the following items:

1. A SIP proxy server needs to be set up and running before the super-peer is started so that it is able to register to it. The super-peer contains the setup information in the **superPeer** class constructor; it will register to the host that has the same name as the host part of its own AOR.
2. The relay script should be running, this will allow for the TCP connection needed to download the files.
3. We need to configure the client phone with the following parameters:
    a. SuperPeer URI including the scheme and a colon.
    b. TCP Proxy is the machine that will serve as relay.
4. The database needs to be set up and the appropriate tables need to be created.
5. Appropriate access and modification rights to the database should be granted to the super-peer account.

9.4 Database setup and configuration

Setting up MySQL

Setting up the MySQL database requires the following commands on our Fedora Core 5 test platform, they may differ if other linux distribution is used.

On the command line:

To start MySQL daemon on OS startup (only for Fedora Core distribution)

chkconfig –levels 235 mysqld on (two consecutive dashes)

To use super-peer account on MySQL

mysql –u sp –p (and type the password when prompted)

On the MySQL prompt:

To show current usernames

SELECT user,host FROM mysql.user;

To add a new user

INSERT INTO mysql.user(Host,User,Password) VALUES('localhost','sp',password('superpeer'));

To activate new account we need to restart the MySQL daemon ***or*** type

FLUSH PRIVILEGES;

To secure root account on new install by adding password (also done for super-peer)

SET PASSWORD FOR 'root'@'localhost' = PASSWORD('3wRkzpu6');

To grant privileges to super-peer account

*GRANT ALL PRIVILEGES ON *.* TO 'sp'@'localhost' IDENTIFIED BY 'superpeer' WITH GRANT OPTION;*

Or alternatively

GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP on shares. to 'sp'@'localhost' identified by 'superpeer';*

To create the database

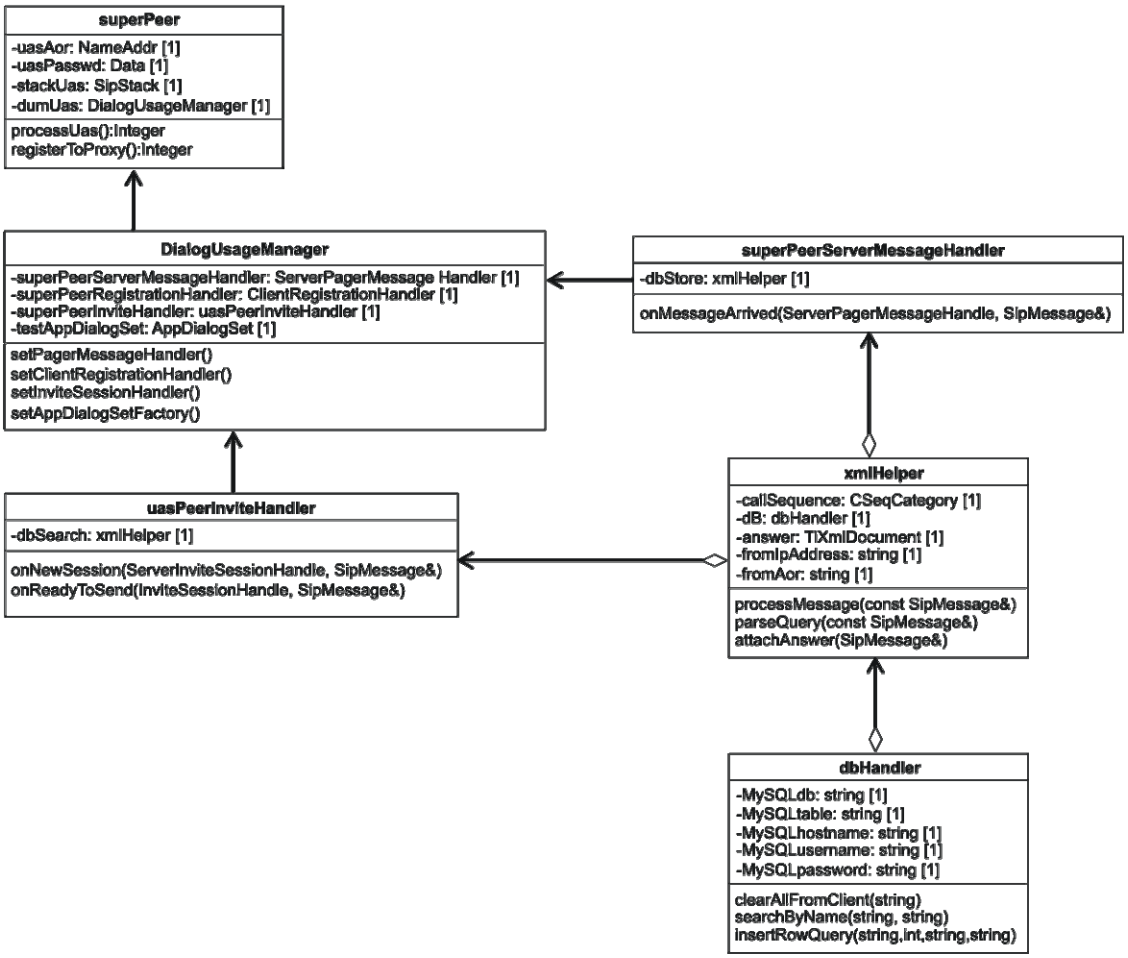
CREATE DATABASE shares;

To create the table

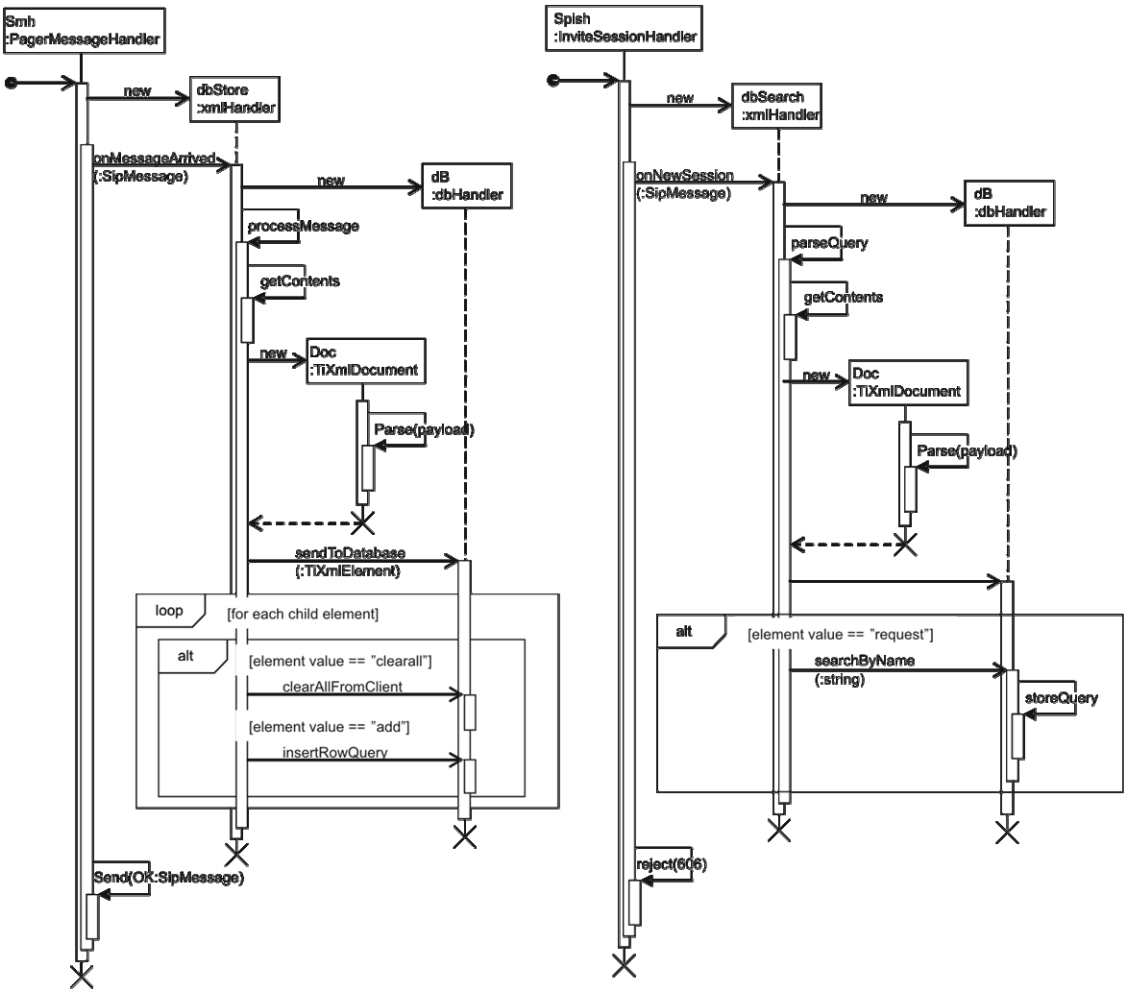
CREATE TABLE files (name varchar(50), size mediumint unsigned, aor varchar(50), time timestamp DEFAULT current_timestamp, hash char(32) ASCII, PRIMARY KEY (hash, aor));

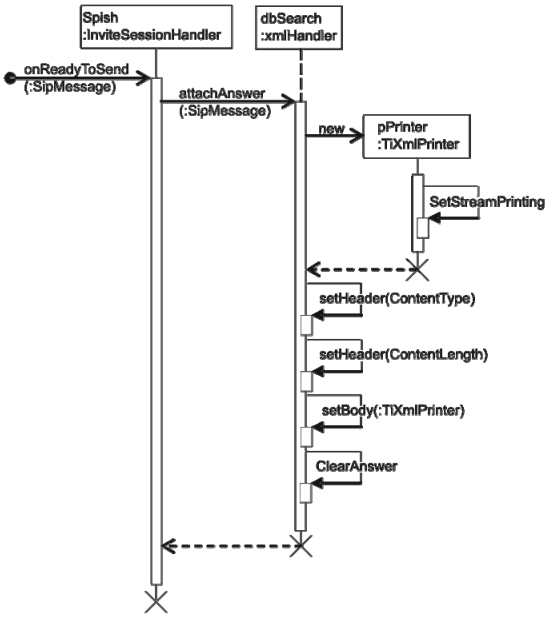
9.5 Super-peer UML diagrams

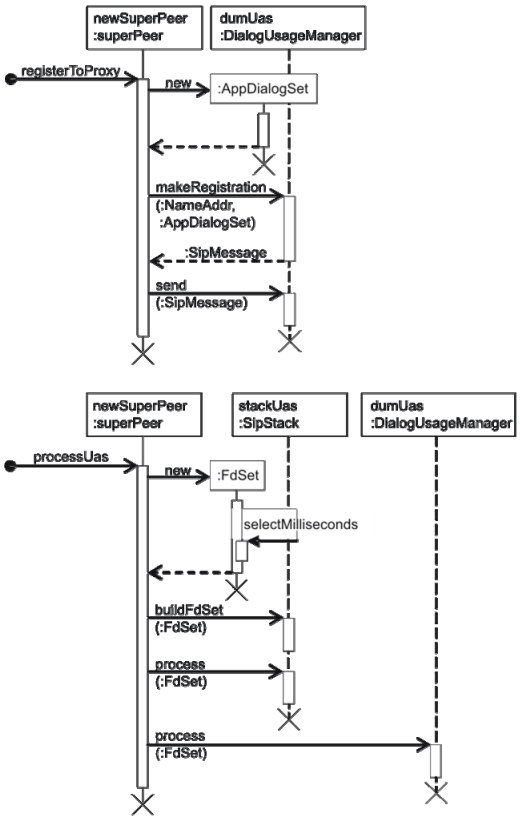
UML class diagrams



UML sequence diagrams




Spish
:InviteSessionHandler
dbSearch
:xmlHandler
onReadyToSend
(:SipMessage)
attachAnswer
(:SipMessage)
new
pPrinter
:TiXmlPrinter
SetStreamPrinting
setHeader(ContentType)
setHeader(ContentLength)
setBody(:TiXmlPrinter)
ClearAnswer


newSuperPeer
:superPeer
dumUas
:DialogUsageManager
registerToProxy
new
:AppDialogSet
makeRegistration
(:NameAddr,
:AppDialogSet)
:SipMessage
send
(:SipMessage)
newSuperPeer
:superPeer
stackUas
:SipStack
dumUas
:DialogUsageManager
processUas
new
:FdSet
selectMilliseconds
buildFdSet
(:FdSet)
process
(:FdSet)
process
(:FdSet)

9.6 SIPp test tool configuration

XML SIPp scenario file

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="super-peer stress test UAC">
<send start_rtd="1">
<![CDATA[

    INVITE sip:spzero@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/UDP [local_ip]:[local_port];branch=[branch]
    Max-Forwards: 70
    Contact: <sip:[field1]@[local_ip]:[local_port]>
    To: spzero <sip:spzero@[remote_ip]:[remote_port]>
    From: [field1] <sip:[field1]@[local_ip]:[local_port]>;tag=[call_number]
    Call-ID: [call_id]
    CSeq: [field0] INVITE
    Allow: INVITE, ACK, MESSAGE
    Content-Type: text/xml
    Content-Length: [len]

    <?xml version="1.0" encoding="ISO-8859-1" ?>
    <request>
    <name>[field3]</name>
    <extended type="size">0</extended>
    <extended type="date">01.01.1900-01.01.2100</extended>
    </request>
]]>
</send>
  <recv response="100" optional="true" />
  <recv response="606" rtd="1" />
<send>
<![CDATA[
      ACK sip:spzero@[remote_ip]:[remote_port] SIP/2.0
      [last_Via:]
      To: spzero <sip:spzero@[remote_ip]:[remote_port]>[peer_tag_param]
      From: [field1] <sip:[field1]@[local_ip]:[local_port]>;tag=[call_number]
      Call-ID: [call_id]
      CSeq: [field0] ACK
      Content-Length: 0

  ]]>
 </send>
</scenario>
```

CSV SIPp input file

```
SEQUENTIAL
1;valiant;Don't Lie;Loo;
2;korene;19 de noviembre;Dru;
3;rubaina;Shut Up and Dance;Kin;
4;barras;Alicia Villareal - Las cuentas claras;Libertad;
5;hades;Hasta Que Amanezca;ahora;
6;mali;Carta;Boy;
7;adeline;A Perfect Teenhood;Tim;
8;buddy;Kill You;Fee;
9;cady;Sideways;soledad;
…
…
…
755;hop;Valon juuri;Sient;
```

10 References

[1] Marcin Matuszewski, Nicklas Beijar, Juuso Lehtinen, Tuomo Hyyrylainen, Content sharing in mobile P2P networks: myth or reality?, Mobile Network Design and Innovation, Vol.1 Nos. 3-4, 2006.

[2] Telemedia, 3G will become a dominant technology, 2006. *http://www.wtmag.co.uk/newsitem.asp?docid=689*, Referenced: 27/03/2007

[3] Juuso A. Lehtinen, Design and Implementation of Mobile Peer-to-Peer Application, Networking Laboratory, Helsinki University of Technology, 2006

[4] Nicklas Beijar, Marcin Matszewski, Juuso Lehtinen, Tuomo Hyyryläinen, Mobile Peer-to-Peer Content Sharing Services in IMS, Helsinki University of Technology, 2006

[5] Tuomo Hyyryläinen, Mobile P2P client implementation on Symbian, Networking Laboratory, Helsinki University of Technology, 2006.

[6] Arturo Crespo, Hector Garcia-Molina, Routing Indices for Peer-to-Peer systems, Standford University.

[7] Alexander Klemm, Christoph Lindemann, and Oliver P. Waldhorst, A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks, Dortmund University, 2003.

[8] Gang Peng, Shanping Li, Hairong Jin, Tianchi Ma, M-CAN: A lookup protocol for mobile peer-to-peer environment, Zhejiang University, 2004.

[9] Dimitrios Tsoumakos, Nick Roussopoulos, Adaptive Probabilistic Search for Peer-to-Peer Networks, Maryland University, 2003.

[10] J. Rosenberg et al, SIP: Session Initiation Protocol, IETF, 2002,RFC 3261.

[11] Andreas Witzel et al, Control servers in the core network, Ericsson review 4/2000.

[12] EarthLink. SIPshare: SIP-based P2P Content Sharing Prototype. 2005. *http://www.research.earthlink.net/p2p/*, Referenced: 26/03/2007

[13] JAIN specification. *http://www.jcp.org/aboutJava/communityprocess/final/jsr032/* Referenced: 26/03/2007

[14] Symella, *http://www.aut.bme.hu/Portal/Symella.aspx*, Referenced: 27/03/2007

[15] SymTorrent, *http://www.aut.bme.hu/Portal/SymTorrent.aspx*, Referenced: 27/03/2007

[16] Martin van den Berg, Open Source SIP stacks compared, *http://www.enseirb.fr/~kadionik/sip/stacks.pdf*, Referenced: 27/03/2007

[17] Open Source Community, reSIProcate, *http://www.resiprocate.org/Main_Page*, Referenced: 27/03/2007

[18] Lee Thomason, TinyXML, *http://www.grinninglizard.com/tinyxml/*, Referenced: 27/03/2007

[19] David Axmark, Allan Larsson, Michael Widenius, MySQL, *http://www.mysql.com*, Referenced: 27/03/2007

[20] Kevin Atkinson, MySQL++, *http://tangentsoft.net/mysql++/*, Referenced: 27/03/2007

[21] D. Willis, B. Hoeneisen, Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration, IETF, 2003,RFC 3608.

[22] Eclipse, *http://www.eclipse.org/*, Referenced: 27/03/2007

[23] WinCVS, *http://www.wincvs.org/*, Referenced: 27/03/2007

[24] Wireshark, *http://www.wireshark.org/*, Referenced: 27/03/2007

[25] SIPp, *http://sipp.sourceforge.net/*, Referenced: 17/05/2007

[26] Dejan S. Milojicic, Peer-to-Peer Computing, HP Laboratories Palo Alto, July 2003

[27] Kalman Marossy, Peer-to-Peer content sharing in wireless networks, Budapest University of Technology and Economics