Using Simple Mail Transfer Protocol on the Last Hop

IPLU SAHA
and
MOHAMMAD SAQIBUDDIN

KTH Information and
Communication Technology

Master of Science Thesis
Stockholm, Sweden 2007

COS/CCS 2007-11

Using the Simple Mail Transfer Protocol on the Last Hop

Iplu Saha – iplu@kth.se
Mohammad Saqibuddin – smu@kth.se

2007.03.03

Advisor and examiner: Prof. G. Q. Maguire Jr.

Industrial advisors: Anders C. Eriksson
Ila Sabet

Sponsor: Ericsson AB, Sweden

Abstract

Some of the enhancements that hit the mobile phone industry in recent years include checking email, demanded changes in the traditional ways of service delivery. People find it convenient to be able to check for incoming emails without being required to be at a fixed location. High-end mobile phones with high resolution color screens and mail clients (or plug-ins) much like the classic clients that run on desktop and laptop computers, have made mail manipulation on a mobile phone both easy and interesting. However, one key difference is the phone's battery power source. Where power was not an issue for desktop and even many laptop computers, since they are almost all the time connected to an AC supply or have high capacity battery storage, it is a major issue for mobile phones. While email applications have greatly advanced, there has not been much improvement in mobile battery capacity. In addition the battery lifetime has decreased due to the high power demands of multimedia applications, which may be running almost constantly.

Traditionally, a mail client checks for new email messages by *polling* the mail server. This works well with computers attached to the power mains or with a large capacity battery, but for mobile phones, polling causes significant *battery drain*. A solution would be to poll the server less frequently by increasing the polling interval, however this would delay email reception hence increasing *latency*.

In this thesis we implement and evaluate a new mail delivery system without changing the underlying mail or communications infrastructure. The new system eliminates the need for *polling* by using network initiated mail delivery. This means that a mail server will forward mail directly to a particular user. Tests conducted with a prototype are compared to the use of the existing system in terms of *power consumption* and *latency*. These tests show that the new mail delivery system reduces both power consumption and delivery delay.

Sammanfattning

På senaste tiden har mobilindustrin utvecklas och tagit stora steg framåt. Högprestanda mobiler med hög upplösning, färgskärmar och epost klienter (eller så kallade plug-ins) har gjort användingen av epost på mobiltelefoner både lätt och intressant. Användarna har funnit att det är bekvämt att ha förmågan att kunna ta emot och skicka epost vart som helst. En viktig skillnad mellan mobiltelefoner och t.ex. en dator är att mobiltelefonen har en begränsad elförsörjning. Olika Funktioner till mobiltelefoner har utvecklas mycket på sistonde men undertiden har batteritiden förblivit nästintill konstant. Programmen som har utvecklas kräver mer och mer kraft av mobiltelefonen och speciellt den senaste trenden med multimedia applikationer, så som mp3 spelare och kamera, som används mycket frekvent av användarna.

Tratitionellt så skickar en epost klient förfrågningarn med jämna mellanrum till epost servern för att ta reda på om det har kommit några nya e-mails. Detta fungerar väl för datorer som inte har några elförsörjnings problem men för en mobil så kostar den periodiska förfrågningen mycket batteritid. En lösning till problemet vore att höja periodstiden mellan varje förfrågning. Detta skulle dock leda till en högre fördröjning på leveransen av eposten.

I det här examensarbetet har vi undersökt och implementerat ett nytt system för epost leverans utan att förändra den underliggande infrastrukturen. Det nya systemet tar bort behovet av periodisk förfrågan, för att ta emot nya e-mails, genom att låta nätverket initiera leveransen. Detta betyder att e-mail servern kommer att vidarebefordra ett inkommande e-mail direkt till mobiltelefonen. Prestandan, i form av elförbrukning och leverans fördröjning, på det nya systemet har mätts genom att testa en prototyp och jämföra prototypen med de systemen som finns tillgängliga idag. Testen har visat att det nya systemet reducerar både ström förbrukningen och leverans fördröjningen.

Acknowledgements

We would like to start by thanking God Almighty for enabling us to complete our thesis. We would then like to thank our advisor at KTH, Professor Gerald Q. Maguire for his endless support and help through out our thesis work. We would not have made it without his help and guidance.

We are also highly thankful to Ila Sabet and Anders C. Eriksson, our advisors at Ericsson who gave plenty of their precious time and effort to help us in our project. Ila specially was always pleasant and encouraging, she did her best to help us out whenever we needed it. We are also thankful to Andreas Ljunggren, Sven Sköld, Jim Andersson and others responsible for the Student Lab, who provided perfect conditions for us to work on and test our system. Thanks to Berit Erkstrom for handling all the management related issues quickly for us. We are highly thankful to Per Ljungberg for giving us the opportunity to work in a challenging environment.

We would also like to thank our colleague Erik Ehrlund for sharing his knowledge and also for helping us with the Swedish translation of the abstract for this report.

Finally, we are thankful to our families and friends for having faith in us. I (Saqib) would specially like to thank Ramina for always being there.

Table of content

1. Introduction

The Last Hop SMTP project defines a new way of delivering messages to a mobile terminal which makes use of the operator's intrinsic knowledge of the network's configuration close to the terminal (mainly the terminal's current IP address). This delivery model eliminates the traditional polling for new messages, thus reducing battery power consumption and reducing delivery latency.

The classic way to deliver asynchronous messages to the users' terminals in traditional e-mail solutions is polling; has two rather severe side effects. Firstly, polling causes significant battery drain as each poll for new messages causes the terminal to wake up and enter a high-power state (which it may stay in for quite some time). Secondly, use of longer polling intervals will cause additional delay in the message delivery, which is annoying when the user is engaged in "e-mail dialogues", i.e., almost instant messaging use of email. These contradictory requirements on the polling interval are usually "solved" by shifting the problem to the user by making the polling frequency a tunable parameter (which, needless to say, doesn't actually solve the real problem).

In the Last Hop SMTP prototype, we initiate all message delivery connections from the network. Thus we are lowering delivery latency and allowing the terminal to remain in low power mode for longer periods of time (at least when there are few messages to the terminal). Note that this thesis is primarily concerned with deliver to a client where the last hop link is General Packet Radio Service (GPRS).

2. Delivering mail to mobile terminals

E-mail, is one service from the larger area of messaging. It is one of the messaging methods most visionaries expect to remain important in future messaging. E-mail service is a particularly successful service in the enterprise messaging arena, where we see new players providing push-mail services, and several new activities such as p-imap (ietf) and OMA-DS (oma) are in standardization.

2.1 Existing methods

The usual cellular packet data deployment pattern today is that many terminals are deployed behind a Network Address Translator (NAT), hence they are inaccessible from the internet. This presents a problem for Internet-based providers of e-mail services, as they cannot reach the terminal at will. This has fostered today's implementations of e-mail services that make use of polling. Polling provides the (usually Internet-based) e-mail service provider with a degree of robustness and independence from the network operators' configuration of their IP networks (such as NATs), as the solution piggy-backs on the mechanisms used for generic Internet access *from the terminals*.

Ericsson Research showed in early 2006 that the use of chatty protocols (such as polling to retrieve messages) causes high power consumption in mobile terminals. The study observes that whenever a packet is transmitted, the terminal is kept in a high power state for a significant amount of time, adding significantly to the battery consumption.

2.2 Limitations of these methods

Today's commonly used e-mail delivery mechanisms all need to send packets periodically. These mechanisms are summarized in table 1.

Table 1: Commonly used e-mail delivery mechanisms and their effect

Mechanism	**Used by**	**Negative effects**
Active polling	Classic POP/IMAP	Repeated TCP establishments
Long lasting connections	P-IMAP [1] with in-band notifications IMAP IDLE	Requires use of and reliance upon TCP-level keep-alive traffic
Out of band notifications (e.g., an SMS notification)	P-IMAP with out-of-band notifications	Requires use of non-IP based service delivery mechanisms, which simply signals the IP based application to do the poll

Given this table, it can be observed that for an all-IP solution, the client either has to repeatedly poll for messages, maintain a long lasting connection, or face long message delivery delays. Even though the long lasting connection might seem attractive, one must consider its associated cost as well. The connection needs to send keep-alive traffic for in-band IMAP IDLE (causing battery drain), or gamble that any state kept in the network for the connection (e.g. in a Firewall/NAT to the Internet) will not be lost. There is *no* indication to the terminal or network when/if such state is lost, effectively silently causing ISP related problems.

The added power consumption introduced by polling has traditionally not been a problem, as the e-mail service was for systems which did not have to worry about their power consumption (due to mains connection for desktop computers or high capacity batteries in the case of a laptop). The advent of mobile e-mail changes this assumption.

2.3 Blackberry devices

Blackberry is a series of wireless handheld devices developed by Research in Motion (RIM) [2], a Canadian company. Blackberry like normal mobile phones can provide email, text messaging, web browsing, and telephony services, using the services of existing mobile networks. The difference is that Blackberry devices have proprietary hardware and software.

Blackberry is primarily known for its ability to send and receive email using the **push technology**. Push technology [3] gives the "Always on, always connected" capabilities to the email systems, where new email messages are instantly **pushed** to the MUA. Pushing email to handhelds without the need for the end device to poll for it is accomplished by the installation of a Blackberry Enterprise Server (BES) [4]. When a new mail message is received by the MTA, it forwards the message to the BES, the BES then pushes the mail message to the handheld. To do this the BES uses the MDS services which is the next generation of the Mobile Data Service, used for managing interactions between the Blackberry devices and enterprise applications [5]. The BES communicates with the handheld using the HTTP and TCP protocols [6]. The BES is also responsible for providing security by encrypting the data with triple data encryption standard (DES) [7] or the advanced encryption standard (AES) [8].

Although a popular solution when it comes to mobile email, since this approach minimizes both the latency associated with the mail delivery and the handheld's battery power consumption, Blackberry has the following disadvantages.

- Blackberry devices are based on a design and software that has to be licensed from RIM.
- High installation costs due to the deployment of BES. The price for a BlackBerry Enterprise Server Software v4.1 for Microsoft Exchange ranges from US $2,999 to 3,999 depending on 1 or 20 user licenses. Additionally, client access licenses are also required which range from US $99 for 1 user to US $27,499 for 500 users.
- Versions of BES available only for Microsoft Exchange [9], Lotus Domino [10], and Novell GroupWise [11].

3. Problem Definition

In this fast paced world staying connected and up-to-date has become a requirement, particularly for business. Mobile terminals have enabled on-the-run access to information and taken real-time communication to the next step. Using these terminals users are able to check for email, read and send email messages (be they short or long), make appointments, get easy access to entertainment, and perform all the necessary communications to carry on their businesses while on the road.

Of course everything comes at cost. The limiting factor here is that with all these advancements and new technology, the battery life of a mobile terminal has not significantly increased, if anything it has decreased due to all the new demands for power. With regard to email, the current system requires the terminal to poll for new mail messages thus adding to the delay in between the mail's arrival at the mail server and its reception by the mobile terminal. However, if the terminal always (or nearly always) has connectivity, then we can think of changing this pattern of communication to deliver the mail to the device when it arrives at the mail server if the user's device is reachable.

This thesis considers the design and implementation of a new mail delivery system which would reduce the terminal power consumption as well as minimize the latency in message reception. The underlying idea is that instead of the client retrieving messages from the server through the Post Office Protocol (POP) (see chapter 5) or the Internet Message Access Protocol (IMAP) (see chapter 5), the mail server itself notifies the client as soon as a new email message arrives using the Simple Mail Transfer Protocol (SMTP) (see chapter 4); specifically this changes the mail server into a mail relay when the user's client has connectivity. This avoids the need for the client to frequently poll and could minimize power consumption (by avoiding empty polls, i.e., polls which don't end up in mail being retrieved) and should also reduce latency, since the request-response pattern of the current system is not required.

4. Mail delivery via SMTP

4.1 SMTP model

In this section, we will discuss how the Simple Mail Transfer Protocol (SMTP) [12] operates. Using SMTP the user's mail is sent to sender's SMTP server. This SMTP server establishes a two-way communication channel to the receiver's SMTP server. Basically, the sender's SMTP server generates commands and the receiver's SMTP server replies according to the command. Note that the receiver's SMTP server can be the final destination or an intermediate destination.

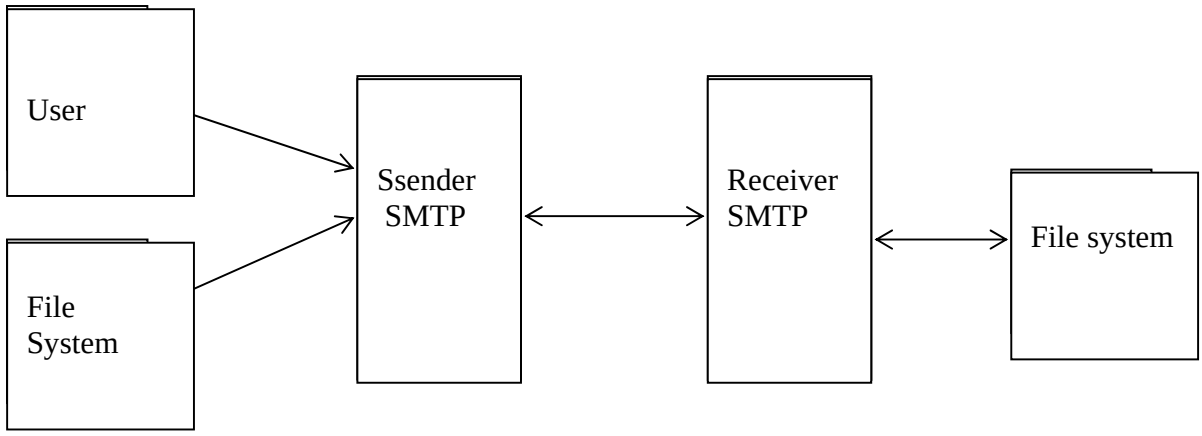

Figure 1: SMTP Model[13]

Once the transmission channel is established between sender and receiver, then the sender's SMTP sends a HELO or EHLO greeting to start the conversation and then the command MAIL is sent, indicating that a mail process is trying to connect along with indicating the sender's SMTP servers identity/address. If the receiver's SMTP server sends an OK reply to the sender, then mail delivery process proceeds. To do so, the sender's SMTP server sends a RCPT command and the receiver's SMTP server checks the address to see if it is a valid recipient, if so, then this receiver SMTP server can receive mail for this particular user. If the receiver rejects the RCPT command, it does not mean that the mail process terminates and the connection is torn down, but rather that the sender and receiver SMTP servers can negotiate regarding other recipients. After negotiation between sender and receiver, then the actual mail data is transmitted. After this sequence, if the receiver SMTP successfully processes the mail data it replies with an OK.

Using the above procedure, SMTP provides a mechanism for transmission of mail. The mail can be transmitted in both directions, the sending user's host can directly connect to the receiving user's host if both hosts are connected to same transport service or by using an SMTP relay. In the later case, when mail is transmitted through an SMTP relay, the sending SMTP server must supply both the mailbox name along with the name of destination host.

4.2 SMTP in detail

In this section we describe SMTP in detail. We will examine mail transaction, forwarding mail, verifying mailboxes name, sending to terminals instead of or in combination with mailboxes, and the opening and closing of the connection between the sender and the receiver.

After initializing the conversation by sending the HELO greeting, the following steps take place.

4.2.1 Mail

For the SMTP mail transaction three steps are needed. The actual mail transaction begins with the MAIL command. The Mail command provides the sender identification.

MAIL <SP> FROM:<iplu@ericsson.com> <CRLF>

The MAIL command sends a request to SMTP receiver, that a new mail transaction is starting. The receiver SMTP prepares itself and if this sender is acceptable by this receiver SMTP it replies with 250 OK.

The second step is the transmission by the sender of the RCPT command, which indicates the recipient's mail address:

RCPT <SP> TO:<saqib@ericsson.com> <CRLF>

The RCPT command specifies the forward path of intended mail to the receiver SMTP server. The receiver SMTP server replies 250 OK if it is willing to accept mail for this destination, otherwise it returns 550 Failure reply.

The last and final step of the SMTP mail procedure is accomplished using the DATA command.

DATA <CRLF>

If the receiver SMTP server accepts the command it returns "354 Intermediate reply" and the actual data of the message is transferred. After receiving the message the receiver SMTP server replies with 250 OK. The mail data includes the message headers such as: To, Date, Subject, From, etc (according to RFC 822 [14]).

4.2.2 Forwarding

In forwarding, the sender SMTP server supplies a forwarding address to receiver SMTP server, then the receiver SMTP delivers the message according to the specified forward path. However, in some cases the destination in the <forward-path> is incorrect. In this case, the receiver SMTP server knows the correct destination it tells the sender SMTP server to correct the destination address. These replies are:

251 user not local, will forward to <forward-path>

This means that receiver SMTP knows that the destination address of the mailbox is on another host and indicates the correct path to use to the sender SMTP. However, the receiver takes the responsibility to deliver the message.

551 User not local, please try <forward-path>

This indicates that the receiver SMTP refuses to accept mail for this particular mailbox and it replies to the sender indicating, either to redirect the mail according to the information provided by receiver SMTP server or to return an error response.

4.2.3 Verifying and Expanding

SMTP provides two special additional features: verifying and expanding. Using these commands a user name can be verified or expanded into a mailing list. These two commands ‘VRFY’ or ‘EXPN’ both have string arguments. For the command ‘VRFY’, the string argument is user name and the reply includes the full name of the user along with the mailbox of this user. For the command ‘EXPN’, the string argument is the name of a mailing list, which can have multiple user names along with the mailboxes of those on this mailing list. These two commands are illustrated in the following examples:

Server: VRFY Iplu
Response: 250 Iplu Saha< iplu@ericsson.com>

Or

Server: VRFY Abc
Response: 550 string does not match anything.

Server: EXPN Example-employee
Response: 250 Iplu Saha< iplu@ericsson.com>
Response: 250 Mohammad Saquibuddin< saqib@ericsson.com>

Or

Server: EXPN Example-Student
Response: 550 Access denied to you

Note that these two commands are not included in our simple implementation of an SMTP server and they are not required to work across relays (since of course the relay doe not know if the address is valid nor if it is actually a mailing list rather than a single recipient's address).

4.2.4 Sending and Mailing

The main purpose of SMTP is to deliver message to user mailbox. The delivery to the user's mailbox calls 'mailing' and delivery the message to user terminal call 'sending'. There are three particular commands for this, which are:

SEND — The Send command indicates that the message will be delivered to user terminal. The mail delivery process will be successful if the message is delivered to the user's terminal.

SOML — The Send or Mail command tells that this message will be delivered to user terminal, if the user's terminal is not active, then it will be delivered to user's mailbox. The mail transaction will be successful if the message is delivered to user's terminal or mailbox.

SAML — The Send and Mail command tells that the mail will be delivered to user's terminal if the terminal is not active, then it will be deliver to mailbox. The mail transaction will be successful if the message is delivered to the mailbox.

4.2.5 Opening and Closing

To initiate a session the SMTP greeting is used, this is written as:

HELO<sender>

At the end of the session the command - QUIT tears down the SMTP connection between server and receiver.

4.3 Address resolution and Mail Handling

In this section, we will describe how address resolution for a particular mail message is done. When the sending SMTP server parses the address, if there is a domain name present, then it does a DNS lookup to see where mail to this domain should be delivered. The DNS lookup first attempts to locate a mail exchange record (MX) associated with this domain name. If a MX record is not found, then it looks for a CNAME record and delivers the mail to resulting name. On the other hand, if only a hostname address (i.e., an A or AAAA Resource Record (RR)) is found, then this A or AAAA RR is treated as an implicit MX RR with preference of 0. It is also possible that an MX record can be found for a particular address, but may not be usable, if so, then SMTP reports an error.

A successful lookup can result in multiple destination addresses, instead of a single address. This may happen because of there are multiple MX records due to multihoming. Therefore, in order to provide reliable mail transmission SMTP can try each of the different addresses from the list until a successful delivery is made.

However, rather than simply trying each of these addresses in order, the sender can rank these host addresses two types of information are necessary. As each MX record specifies a preference, where the lowest number means highest preference. It also possible that the preferences have the same value, in this case the sender-SMTP randomise its selection of address to spread the load across multiple mail receivers

In addition, the destination host might be multihomed, which means the DNS lookup returns a list of alternate IP addresses for the particular destination. The existence of multiple addresses means that there may be multiple paths to the mail server, hence increasing the probability of successful mail delivery despite failures along some routers, however there is no guarantee that the paths are truly independent.

Once a successful address resolution is done, then the SMTP server relays the message by rewriting the MAIL FROM, TO, and CC/BCC fields (mail data, conforming to RFC 822) and delivers the message to destination or hand off responsibility for delivery to some mechanism outside the SMTP-provided transport environment.

4.4 SMTP Security

SMTP is vulnerable because SMTP passes mail from one SMTP server to another SMTP server, and these SMTP servers do not check or verify whether the sender’s address is valid or not. This lack of checking for a valid sender causes a huge amount of spam which generates unnecessary network traffic, especially since most of this traffic will be filtered after it is delivered. Various protocol extensions [15] provides authentication at the transport level. However, end-to-end delivery from SMTP client to SMTP server requires some additional authentication process in order to be more secure.

There are several commands such as VRFY, EXPN that disclose the names of user mailboxes and mailing lists. When two SMTP servers establish a connection between them, the sender also discloses what it claims to be its domain name by using HELO command. Information about user names or mailbox names and mailing list can be disclosed when the forwarding causes reply 251 or 551 codes to be sent by receiver SMTP server. Since SMTP servers do not check or verify the sender's address, it is easy to send e-mail as if they were set by anyone - hence offering an opportunity for deception, fraud, and spam.

In the past few years, SMTP security has been getting better day-by-day. SMTP-AUTH [16] is one of the causes for this. This protocol adds an access control mechanism that allows the receiving SMTP server to verify whether this sender’s address is valid or not. In general, SMTP-AUTH allows authentication between two servers. While relaying mail, the receiver SMTP knows that the sender address is authentic, but does not prevent the sender's e-mail address from being forged. However, if each of the relays has authenticated the sender, you can at least track the mail back to the source SMTP server.

5. Mail reading via POP/IMAP

Post Office Protocol (POP) [17] and Internet Message Access Protocol (IMAP) [18] are application-layer protocols used to retrieve email from a remote server using a TCP connection. POP3 is the third and current widely used version of POP. It allows the email client to download mail to the local machine when it is connected to the Internet. Subsequently these messages can be accessed – even when the device is offline. Email clients using mail agents such as Outlook, Eudora, etc., retrieve the mail from an IMAP or POP3 server.

5.1 Fetching mail

A typical email server listens on port 110 for POP and port 25 for SMTP. The mail client establishes a connection with a POP3 server by opening a TCP connection to port 110, after the connection is established the POP3 server sends a greeting to the client [17]. Mail is transferred based on a series of commands and responses. The server sends commands that consist of case-sensitive keyword followed by one or more arguments. In response to these commands, replies are sent by the client which consist of a status indicator and a keyword.

After the connection is established and the POP3 server sends its greeting to client, the session enters an authorization phase, where the client has to identify and authenticate itself to the server. Several mechanisms can be used to authenticate the client. One is a USER name and password command combination and another is APOP [19]. Once the server authenticates this client is given access to a particular maildrop/mailbox. The server acquires an exclusive access lock on the mailbox so that the message can be removed or modified during a subsequent UPDATE state.

Following the authentication phase, the session moves to the transaction state, here the client can send several different commands and the server responds according to the command. The commands which may be used for transactions are:

STAT
LIST
RETR
DELE
NOOP
RESET and
QUIT

Once the server and client sends the QUIT command, the POP3 session enters into the update state. In this state the server releases any resources required in TRANSACTION state and says bye, then tears down the connection between the client and server.

If the client sends a QUIT command before entering into the TRANSACTION state then the POP3 session does not enters the UPDATE state, in this case it simply tears down the TCP connection and the client has to re-establish the connection.

5.2 Managing Mail Folders

POP3 is a simple and standard way of accessing and downloading mail from a POP3 server. The basic folder at the server that stores the mail for client is known as MAILBOX. When the client establishes a connection with the POP3 server it will download the mail messages regardless of their size. The POP3 server does not allow for any server side modification of the messages, nor does it allow the users to create any other folders at the server.

On the other hand, IMAP is a flexible method of accessing and downloading email. IMAP is a more robust and feature rich protocol, which supports multiple folders at the server, additionally mail need not be deleted from the server whenever the client downloads it from the server. By default, IMAP downloads only the message header, thus the user can preview the subject, message header before deciding if they wish to download the entire message. An IMAP client can also delete the entire message without downloading it, which is a very useful option for junk mail or large messages with attachments.

The primary mailbox for a particular user at the IMAP server is called INBOX. If a hierarchy is needed for mailboxes then the mailbox names must be left-to-right hierarchical using a single character to separate the levels of hierarchy. To easily distinguish between different types of mailboxes, a mailbox name beginning with '#' identifies the 'namespace' of the reminder of the name. IMAP is designed to allow synchronization of mail with clients. If a message is moved from one folder of the client to another, then this will be reflected in server folders too. IMAP also allows the user to access, create, and store mail in a server side folder.

5.3 Multiple Mail Folders

Multiple mail folders are only available from an IMAP server not a POP3 server. IMAP server side folders are very convenient, as user can have different kind of mail categories such as news, personal, business, etc. IMAP server allows the user to differentiate their mail and place it into different server folders. IMAP can send the notification of new mail to the client (without the need for the client to poll).

6. Proposed Architecture

This project examines email delivery via GPRS. GPRS usage is increasing day-by-day, and it enables users to stay connected to the Internet regardless of the time or device location (as long as it is within a GPRS coverage area). Although a GPRS connection is relatively slow, compared to other means of connecting to the Internet, it allows mobile access at volume based pricing rather than duration pricing. If the user is using a GPRS PC Card or their mobile phone, then there are some issues which should be addressed, such as battery power consumption and latency. In this section, we will examine these aspects in detail.

6.1 Pre-study

During the pre-study phase, we gathered information regarding the system requirements, its potential components, different options, and studied different use cases to determine the feasibility of using various technologies.

As we were to use GPRS; we first needed to know its advantages and disadvantages, how a GPRS connection work, how the user would connect to the GPRS network. To learn this, we read lots of documentation concerning GPRS, including Ericsson (public and Internal) documentation [20] and several white papers [21,22,23]. After this we focused on mail systems. While using email was quite familiar to us, we needed to study the details of each of the different mail components, such as the protocols: SMTP, POP, and IMAP; mail handling and delivery systems; and mail clients.

Additionally we considered other message delivery alternatives, such as WAP push [24], SMS messaging [25], SIP [26], etc. This helped us to design our proposed system and to later compare it with the existing means (see section 10.2). We emulated the wide area communications network in our prototype. This had the advantage that we could perform experiments with different network behaviors, representing a GPRS or 3G network under different conditions.

The pre-study phase focused on the technical aspects of the project such as, the use of an IP network on top of a wide area cellular network, the end-to-end details of a mail system, GPRS connectivity, etc.

6.1.1 E-Mail System

In general, the email system consists of one or more Mail User Agents (MUA) used to view and write mails, and one or more Mail Transfer Agents (MTAs). The sender's MUA uses SMTP to send the email message to it's local MTA. Now the sender's local MTA learns the IP address of the receiver's MTA by performing a DNS lookup (as described previously in section 4.3). Once the IP address of the recipient's MTA is known, the sender's MTA forwards the email to the receiver's MTA using the SMTP protocol, to deliver the email into the receiver's mailbox. The recipient can get and/or view the email message using the POP3 or IMAP protocols (as described in chapter 5).

The structure of e-mail flow can be illustrated as shown in a figure 2.

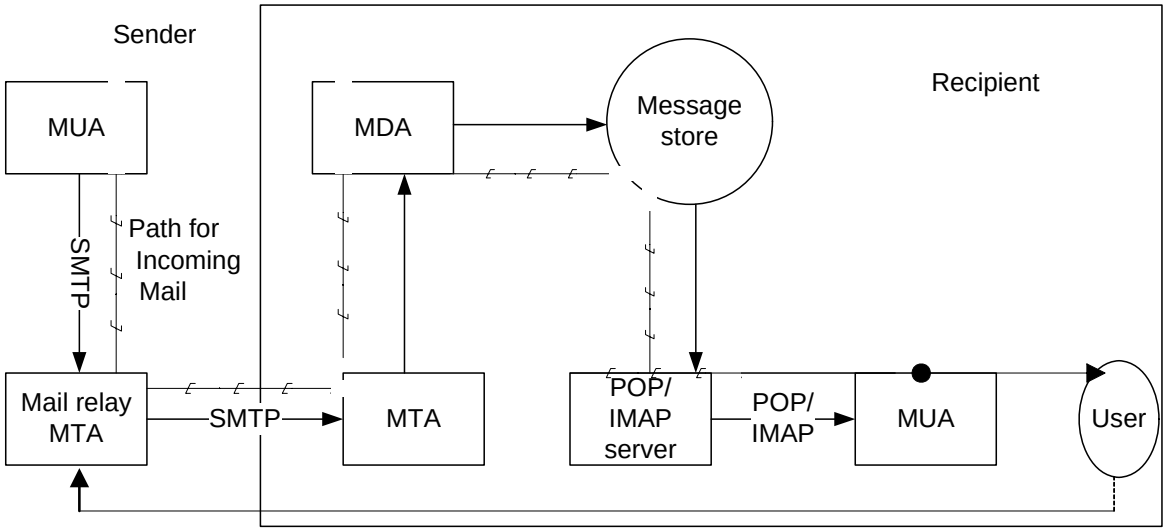

Figure 2: Mail flow [27]

To accomplish sending email from one user to another, there are two ways. The first one is to use webmail as interface to connect and send email directly to the Mail Transfer Agent (MTA). Another method is to use Mail User Agent (MUA) such as Microsoft's Outlook Express, Netscape's communicator, etc. as an user interface which subsequently connects to the MTA to send its outgoing mail MTA acts as a mail relay agent-hence it forwards email to another MTA until this email reaches a destination MTA. (The number of hops depends on the mail routing.) this final MTA will pass the message to a Mail Deliver Agent (MDA) to store as a plain file or in a database for the recipient.

Therefore in our solution we need an SMTP server (a MTA) responsible for receiving e-mail from other MTAs and delivering these emails to the users' mail directories. After analyzing all of advantages and disadvantages of several solutions, we choose Postfix as our mail server. We choose Postfix as it is easy to configure, secure, and moreover is easy to administer. Although sendmail and smail are also widely used today, and they are easier to configuration, the administration of both is more demanding than Postfix. Additionaly, because security is becoming an important issue for mail services, this too makes Postfix a better choice.

6.1.2 Internet Message Access Protocol (IMAP)

The Internet Message Access Protocol (IMAP) performs the same basic function as POP3 protocol but in addition IMAP enables the users to access their email messages residing on a remote server. This means that the client can view these email messages without having to download the full message to its own storage, as would have been the case with POP. Additionally, since the mail remains on the remote server, users are free to move between different devices (such as phones, PDAs, or PCs).

6.1.3 GPRS

General Packet Radio Service (GPRS) runs on the top of existing GSM system. It is a technology often referred to as "2.5G", that is, a technology between the 2nd and 3rd generation. GPRS only reserves radio resources only when there is data to send, thus reducing dependency on circuit-switched network elements. GPRS supports packet-switched data connections, thus multiple users can share the same transmission channel. The main difference between packet-switched and circuit-switched data connection is that packet-switched does not reserve resources in the network. Additionally, packet-switched data are billed per kilobyte of data transmit, whereas circuit-switched calls are billed per second whether there is any useful data transferred or not. Packet switched services also make much more efficient use of the radio resources as these resources are only used when there is traffic to send.

GSM and GPRS uses a time division multiplex channel divided into eight time slots. Voice and data sessions typically use 1 and 2 time slots respectively. Each time slot supports a data rate of 14.4 kbps. So, the total amount of data rate in one cell is 115 kbps (however, the Enhanced Data Rates for GPRS (EDGE) [28] supports higher data rates by using different channel coding during the time slot). Client traffic is routed via the radio access network to a GPRS service node, and then tunneled through the operator's GPRS backbone network to the operator's Gateway GPRS Service Node (GGSN) which acts as the network's egress point. The destination of the tunnel end-point is identified by its Access Point Name (APN). Hence the Gateway GPRS Service Node (GGSN) serves as the gateway to an intranet or Internet, depending on the APN used.

6.1.3.1 GPRS capability

GPRS capability is defined into three classes: A, B, and C. In class A, a user can be connected to both GPRS and GSM systems at the same time. In class B the user can connect to GPRS or GSM, but not both at same time. Most GPRS mobile devices are class B. In class C a user can connect to GSM or GPRS, but not both and they have to switch manually from one service to another.

6.1.3.2 GPRS Architecture

Figure 3 shows the basic GPRS architecture. It consists of a number of types of nodes. These are described below.

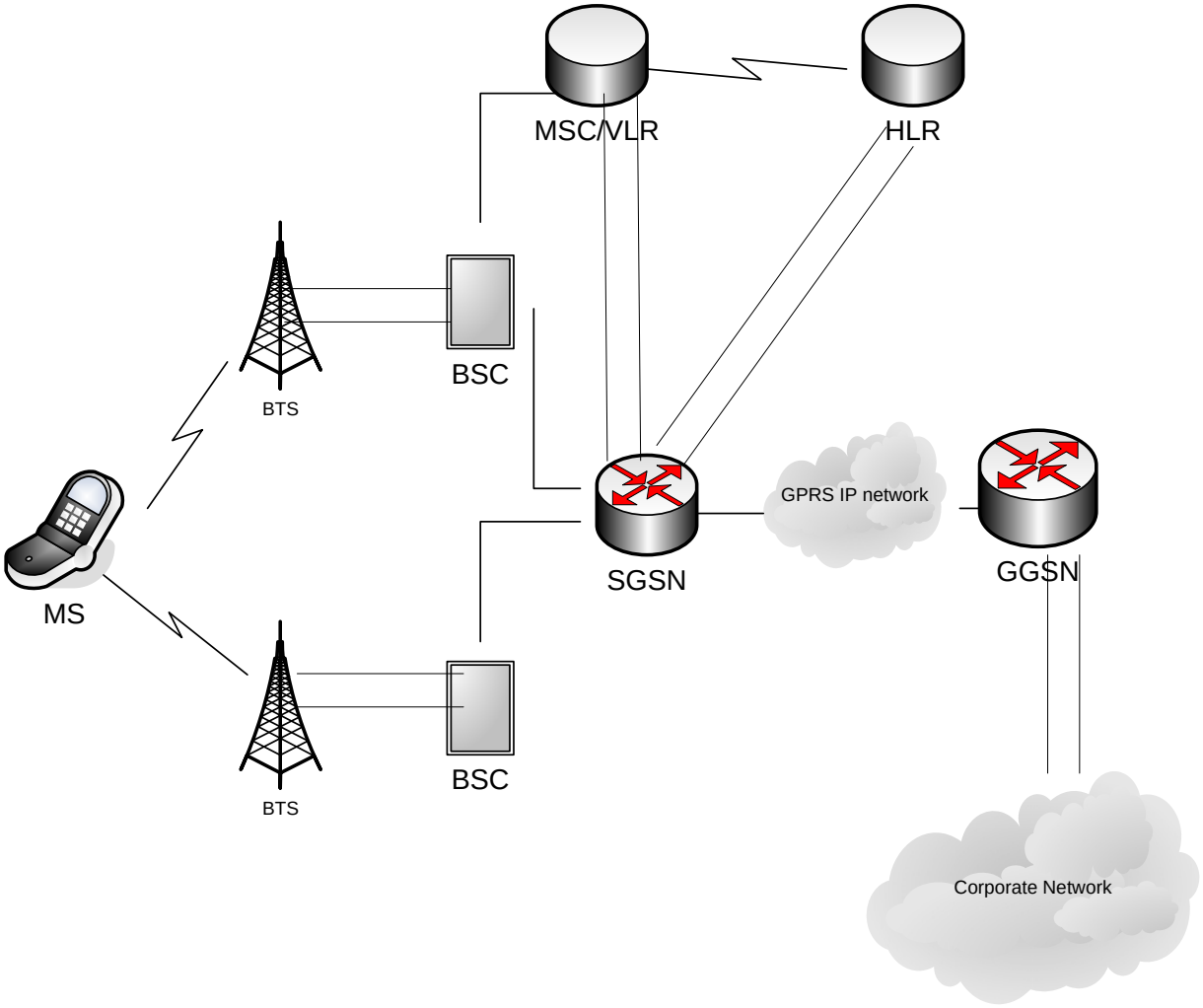

Figure 3: GPRS Architecture

MS: Mobile Station
BTS: Base Transceiver Station
BSC: Base Station Controller
SGSN: Serving GPRS Service Node
GGSN: Gateway GPRS Service Node
HLR: Home Location Register
VLR: Visitor Location Register

MS A mobile transceiver or mobile enabled radio device operating within a mobile network.

BTS	A transceiver station that transmitting or receiving radios in fixed locations.
BSC	A BSC is used to control groups of BTS, provide mobility management to mobile stations, anchor air link protocol and provides connection to Mobile Service Center.
HLR	A database resides within a cellular network to hold and keep track of current customer details, equipments in use, service required, user's identification encryption code.
SGSN	It connects one or more BSC to the GPRS backbone network and provides IP connectivity to GGSN.
GGSN	Last gateway of the network, it provides interconnection between GPRS and external packet data networks.
MSC/VLR	It provides basic telephone switching services to link a cellular network with the public telephone system network.

6.1.3.3 Security

Security is one of the key points in GPRS. Security in the context of GPRS includes:

- SIM based subscriber authentication
- ciphering of the path between mobile terminal and SGSN protects the link against eavesdropping
- GTP encapsulates the packets in the GPRS backbone
- IPsec technology is used to provide a secure connection between SGSN to GGSN to external packet network.
- Packet filtering is use in GGSN and SGSN.

6.2 Modules of the prototype

In this section we will describe the modules that we used in our proposed Last Hop SMTP system. We divided the system into two parts. One is the server side and another is the client side. The specific computers we used were a Dell Server (playing the role of the server(s) in the infrastructure) and a PC/laptop (playing the role of the client).

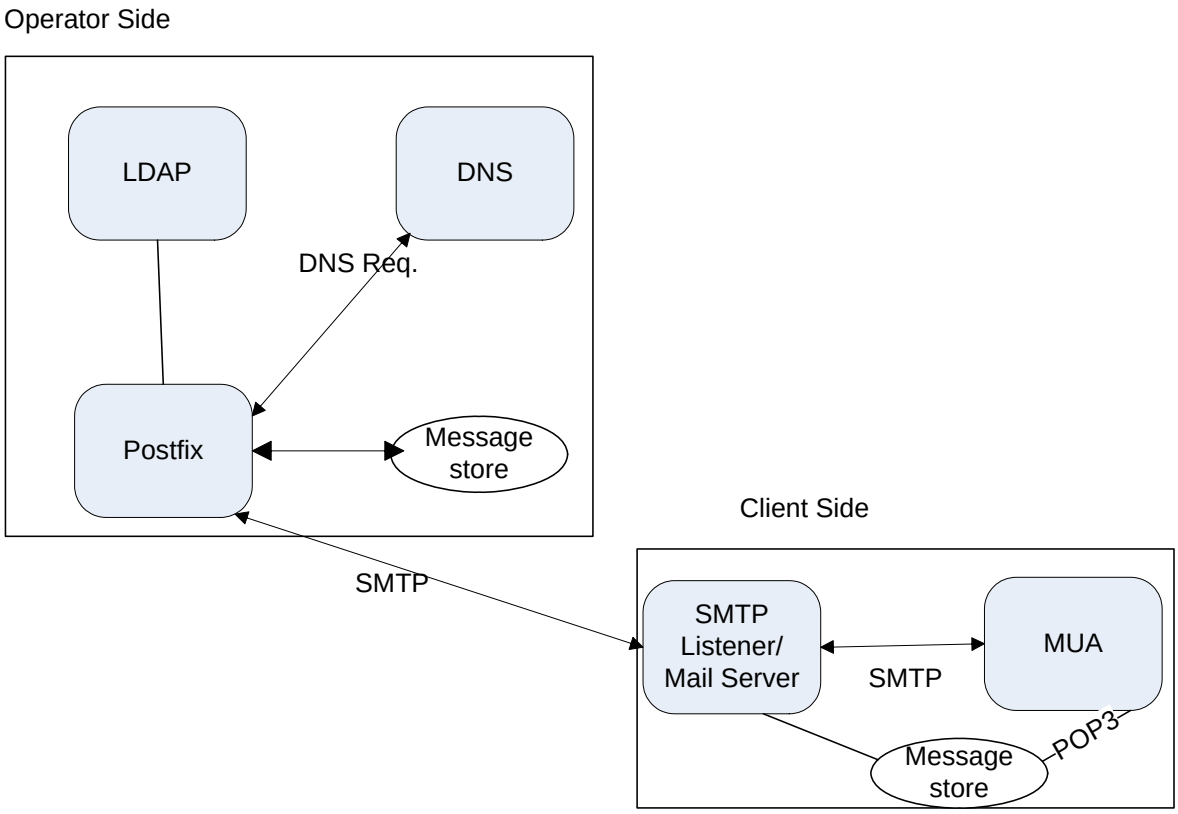

Figure 4: Prototype Model

In figure 4, the left side labeled as ‘Node (Mail relay)’ is our server side implementation and the right side labeled as ‘Node (Client MTA)’ is our client side implementation. For our server side implementation the basic software we used were: Postfix and BIND

6.2.1 Postfix

Postfix [29] is a mail transfer agent that handles the internal and external mail delivery. The reason we chose postfix is that it is easy to configure and offers better security than other MTA such as sendmail [30], exim[31] etc. Our proposed system is based on SMTP delivery, so in the postfix configuration we configured SMTP with TLS, so that we could use TLS to protect the transport connections.

6.2.2 BIND

BIND (Berkeley Internet Name Domain) [32] is an implementation of the Domain Name System (DNS) protocols and provides an open source implementation of the major components of DNS. DNS is also used for reverse lookup of addresses. We chose BIND, because it is used in the vast majority of name serving machines on the Internet, and because it provides a robust and stable architecture on top of which an organization's naming architecture can be built. Bind is a good choice for large-scale networks (more than a couple of hundred workstations). As our mobile

terminal will get a dynamically assigned IP address each time it connects, thus BIND9 can also be used to act as a Dynamic Domain Name Server (DDNS).

Dynamic DNS server allows the domain name data stored in a name server to be updated in real time. By using DDNS one client get a dynamic IP address each time it is connected. To implement DDNS it is necessary to set the maximum caching time into a short period. The terminal uses DDNS client software to let the DDNS server know the current IP address. So the DDNS server can update the particular client IP address and store in the database. BIND is the software we used for DDNS and comparing to DNS server we just changed some parameters.

6.2.3 DHCP

If the client is in the provider's network then it is not necessary to implement DDNS server. In this case the DNS server along with DHCP server is sufficient to give the client's IP address. This server was necessary for our testbed for testing the initial prototype, but it would normally be implemented independently of the Last Hop SMTP service.

Dynamic host configuration protocol is a kind of protocol that assigns IP address to a device in network dynamically. It eases the function of assigning IP address manually. The administrator is saved from manually configuring the IP address when a new device is added to a network. DHCP gives a device a unique IP address every time it is connected to network. And it also leases time for a particular IP address and the lease time can be updated.

Moreover, it also supports static IP address. We used dhcp-3.0.1rc14-1.i386.rpm software for DHCP; in comparison with other versions, this version of DHCP solves the problem of DDNS update, classing, separate address pools with different access permission and conditional behavior.

To implement the client side the required software includes a Java Mail Server and a Mail User Agent.

Java Mail server: It's a small mail server that opens and listens for SMTP traffic from the server to the client. When the mail server wishes to deliver mail to this client machine it simply opens an SMTP connection. After this the server and client side negotiate and the server delivers mails to the client machine. The client machine stores this mail in a particular file for subsequent retrieval by a mail user agent.

Mail User Agent: Once the mail is delivered and stored in the client machine then a Mail User Agent (MUA) such as Microsoft's Outlook Express, Eudora, fetchmail, etc, fetches the mail from the client machine's mail queue and places it in the INBOX of this MUA. The MUA can use POP or IMAP to fetch this mail. The integration with MUA is of course preferable since it makes this incoming mail readable for the user.

6.3 Details of the Proposed Architecture

In this section we describe our proposed system. Presenting the proposed architecture and a stepwise clarification of exactly how it works.

1: The mail transfer agent outside of our network does a DNS lookup based upon our domain name to learn the IP address of the mail transfer agent (relay machine) in our network. To learn this address the mail transfer agent makes a DNS request.

2: The DNS server inside our internal network receives the DNS request and sends a DNS response, which is the IP address of our Mail Relay MTA (the machine acting as a mail relay).

3: Once the external MTA resolves the IP address of our domain it connects to the mail relay machine using SMTP.

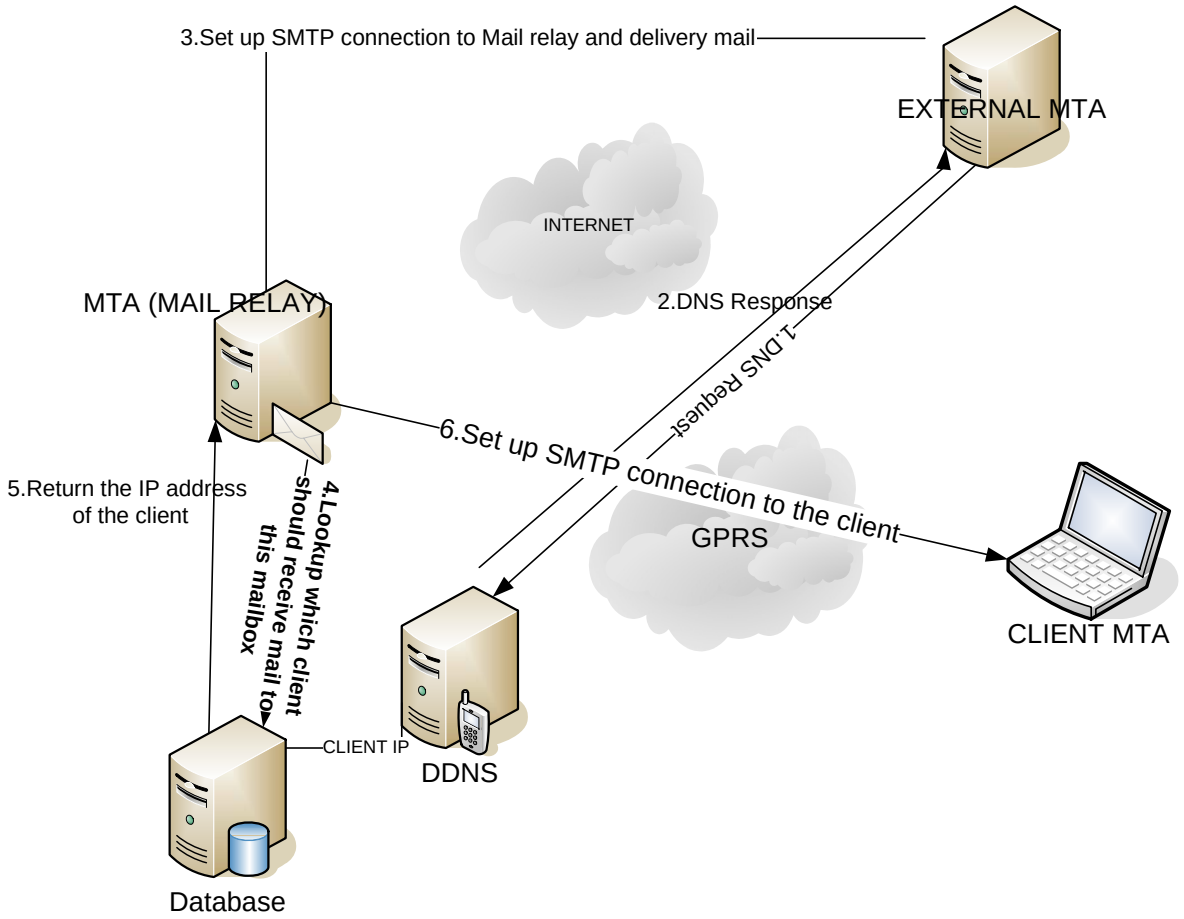

Figure 5: Proposed Architecture

4: Mail relay machine receives the mail and now it checks for the particular user that this mail is intended for. To do this, the Mail relay checks in the database to learn the particular user's details.

5: The database query's the DDNS server to map the username to the current IP address of the terminal that the mail is to be delivered to.

6: Now the mail relay machine knows the IP address of this particular user's client MTA. Now the mail relay machine establishes a connection to the client MTA using SMTP and delivers the mail to client machine.

In figure 5, we show the three different machines: for mail transfer agent (mail relay), database, and DDNS server. Logically there are four separate servers, but physically we implemented three of them in one machine. For DDNS we registered a free domain with a DDNS service provider. Using the above-mentioned steps a successful mail delivery is made to the client machine.

6.4 Comparisons with existing means of mail delivery

Polling is the classic way to deliver messages to the users' terminals in traditional e-mail solutions and the newer Push Mail solutions (such as those described in [33]) have two rather severe side effects on the service experience for the mobile user. Firstly, polling when there is no new message causes unnecessary battery drain. Each poll for new messages causes the terminal to wake up and enter a high-power state, which will be held for quite some time (Ericsson internal research [34]). Thus the solution is to poll infrequently. Secondly, the polling interval adds to the delay in delivering the message, which is annoying when the user is engaged in "e-mail dialogues". These contradictory requirements on the polling interval are usually "solved" by pushing the problem to the user by making the polling frequency a tunable parameter (which, needless to say, doesn't actually solve the real problem).

In the Last Hop SMTP prototype, we initiate all message delivery connections from the network. Thus we lower delivery latency, and we allows the terminal to spend longer low power mode, as ideally it only needs to come out of low power mode when there is actually a message to transfer. This solution assumes that there is some low power way for the terminal to know that there is incoming GPRS traffic for it. We will consider this problem in our testing phase.

6.4.1 Benefits

Integrating Last Hop SMTP into the existing messaging solutions is relatively straightforward. Adding Last Hop SMTP to an email server is trivial. The terminal side implementation is from a technical point of view not complicated, but the terminal needs to run an additional application to act as the local SMTP MTA in order to receive the incoming mail.

Given the attention Push E-Mail has received in the market, Last Hop SMTP should be of significant interest to operators with an interest in providing low delay messaging support. For example, if the terminal has an IMAP client it could simply use the IDLE command to indicate that it is connected and ready for messages. Then the mail server would immediately deliver notifications of new messages when a new message arrives. That prevents an ISP from offering this same solution. All they need to do is have their subscriber install a suitable software client, which acts as the local MTA! This client simply connects via GPRS to their ISP's mail relay-

implicitly letting it know where it can be reached (because it establishes an SMTP connection to the mail relay).

6.4.2. Disadvantages

The disadvantage of Last Hop SMTP is that the Mail is actually sent to the user's device, even if they do not want this sort of mail at this device. The users might not like to receive mails with attachments on the mobile phone or if they never want mail from a particular source etc. Additionally, the user looses device independence since the mails are downloaded on the mobile phone and are removed from the relay server.

6.5 Scaling through LDAP

To scale the system for large number of users, we decided to use the Lightweight Directory Access Protocol (LDAP). LDAP is a networking protocol for querying and modifying directory services running over TCP/IP.

The Postfix mail system can use optional tables for address rewriting and mail routing [35]. By using LDAP together with postfix, postfix can use LDAP as a source for any of its lookup tables. This allows postfix to keep information about mail services in a replicated network database with access control. The advantage of using LDAP is that the network administrator can maintain it from anywhere.

In postfix, we utilize a file called ‘main.cf’, where most of the information about the network, relay domains and local recipient maps is configured. For the lookup table and mail routing, ‘aliases’ and ‘transport’ files needs to be set up.

In order to use LDAP lookup, we need to define an LDAP source as lookup table in ‘main.cf’. It is important to map the LDAP source file by giving the path to where the file is located.

By using LDAP to store source lists such as domain name, relay domains, IP address, local recipient maps, etc. the LDAP table stores each list member as a separate key. The table lookup verifies the existence of the key. To configure LDAP we need to know some basics about general LDAP parameters such as server host, server port, search base, query filter, result format, domain, result attribute, bind, etc. A short description of these LDAP parameters is given below.

Server host	the machine name of the host running the LDAP server.
Server port	port number that LDAP server listens to.
Search base	a configurable parameter according to the domain name at which to conduct the search.
Query filter	used to search the directory which is known as a mail accepting general ID.
Result format	allows one to use mailHost attribute as the basis of the transport table.

Domain	list of domain names or paths to files.
Result attribute	postfix will read the attributes from any directory entries that are returned by lookup queries to resolved and email address.
Bind	to bind to the LDAP server or not. If yes, then we need to supply a distinguished name.

When a mail message is received by the mail server for a local address “ldapuser” which is not found in the aliases database, postfix will search the server host. It will bind anonymously, search for matching directory entry, which has a query filter attribute “ldapuser”, then it will return the resulting attributes that were found and build a list of these results. This list will be treated as mail address to which the message will be delivered.

6.6 Setting TTL for DNS

With a large number of users it is more convenient to use a database to store information rather then DNS zone files. The administrative overhead can be greatly reduced by using databases since the same database can be used for name resolution, mail forwarding, and account management. In addition, due to the DNS design the information needs to be stored redundantly on two or more machines, which becomes easier through the usage of replicated network databases [36].

For dynamic IP addresses where a user might connect from various places with different IP addresses, we need to set the Time To Live (TTL) parameter to a suitable value. The TTL property is associated with all DNS records and specifies the time duration for which other DNS servers can cache the record. Once a record is cached by a DNS server, the TTL value decreases with time and the record is removed from the cache when the TTL reaches zero. In general, the TTL value is set to a few minutes for the records that correspond to hosts with dynamic IP addresses. Caching of records is necessary to reduce network traffic and so a reasonable TTL value should be configured since a very high value would delay the propagation of changes to a record through the network and a very low value would increase the network traffic.

It should be noted that for a small number of users it is even feasible to give the TTL as zero since the number of name resolution queries will not exceed one lookup for every incoming message per user.

7. Setting up the client machines

To demonstrate the flexibility of the proposed architecture we will show how it can be set up and used on two different types of client machines: a linux machine and a Microsoft Windows XP machine.

7.1 On Linux

To test our prototype we configured a machine to act as a client using the Linux operating system. As we discussed earlier, we have to set up a Mail Transfer Agent (MTA) at our client machine.

First we installed UBUNTU Dapper Drake version 6.0.1 and then installed Postfix. This software is open source which means customer don't need to pay for using this software and they can make changes to it if necessary. We adjusted some parameters according to our setup, specifically in the configuration file main.cf and in the transport file. Because we need to specify a hostname for our client machine, we named it 'saha'. A sample configuration is given in figure 6.

```
Myhostname = saha
Alias_maps = hash:/etc/aliases
Alias_database = hash:/etc/aliases
Transport_maps = hash:/etc/postfix/transport
Myorigin = no-ip.org
Mydestination = ericsson.no-ip.org, no-ip.org, localhost.no-ip.org, localhost
Relayhost =
Relay_domains = no-ip.org
```

Figure 6: sample configuration-contained in main.cf

```
no-ip.org       smtp:[ericsson.no-ip.org]
```

Figure 7: sample configuration- contained in transport file

Next we configured resolv.conf to configure the name server and the default domain name. A sample configuration is shown in figure 8.

```
nameserver 192.16.149.74
search no-ip.org
```

Figure 8: Sample configuration-contained in resolv.conf

Now our client machine is ready to receive mail from our relay server through SMTP. However, in order to make the new mail available to the end user we need to use Mail User Agent (MUA) such as Mozilla thunderbird [37], evolution mail [38], etc. For our purpose, we chose Evolution Mail.

7.1.1 Setting up the MUA

In this section, we will show some screenshot of how we configured the Evolution Mail MUA and the basic parameters that need to be configured.

To set up a MUA we must configure four different aspects: our account information, where we should receive email, our choice of receiving options, and where we should send outgoing email. The screenshots below clear shows which parameters need to be changed.

Figure 9: Entering your Account information, here the user's name is ericsson and their e-mail address is ericsson@no-ip.org (free domain registered at no-ip.com, a DDNS provider).

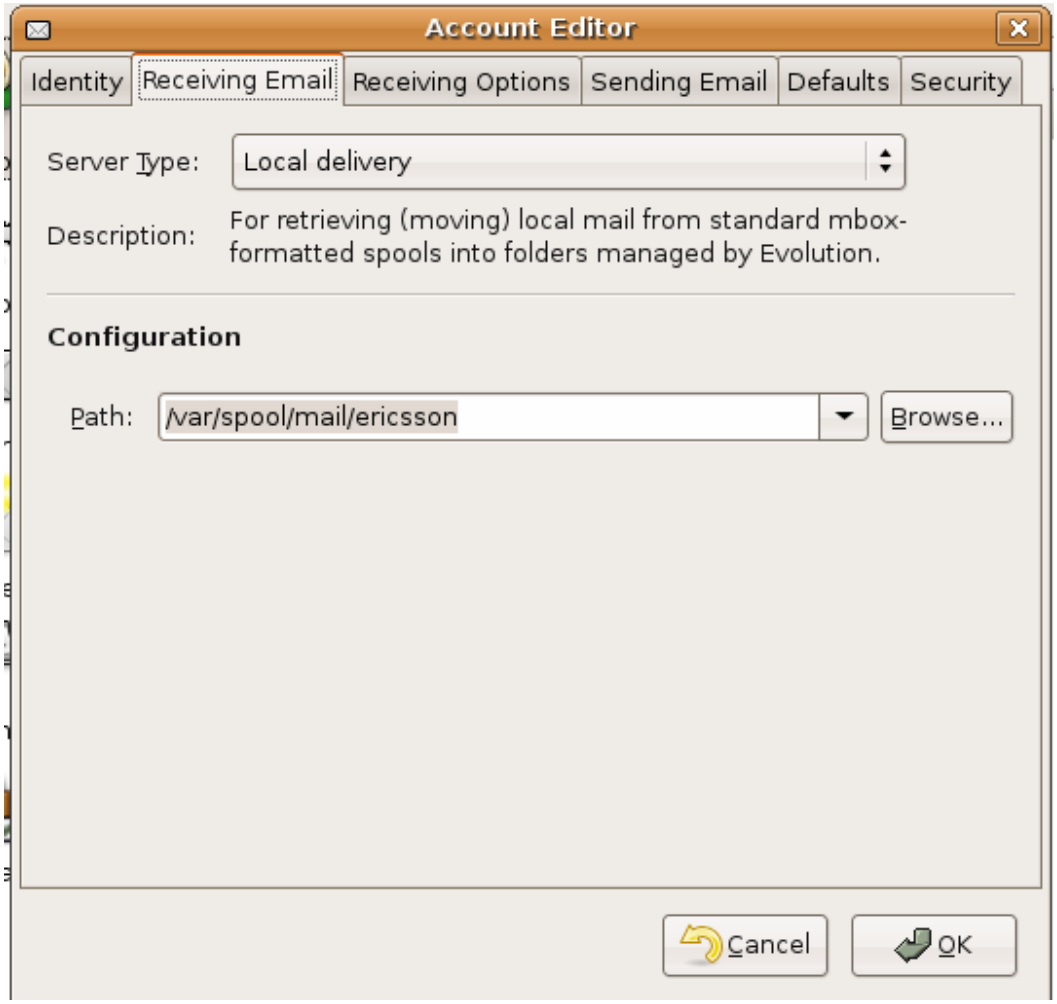

Figure 10: Indicating where you will receive your Email (i.e., where the MTA deposits incoming mail).

The mail is received by the MTA running on the client machine and simply placed into a mail spool area in /var/spool/mail under the user's local account name. So, MUA's task is simply to fetch mail from this particular folder of the client machine. Note that this delivery occurs locally hence there is no network traffic generated when the user reads this message.

.

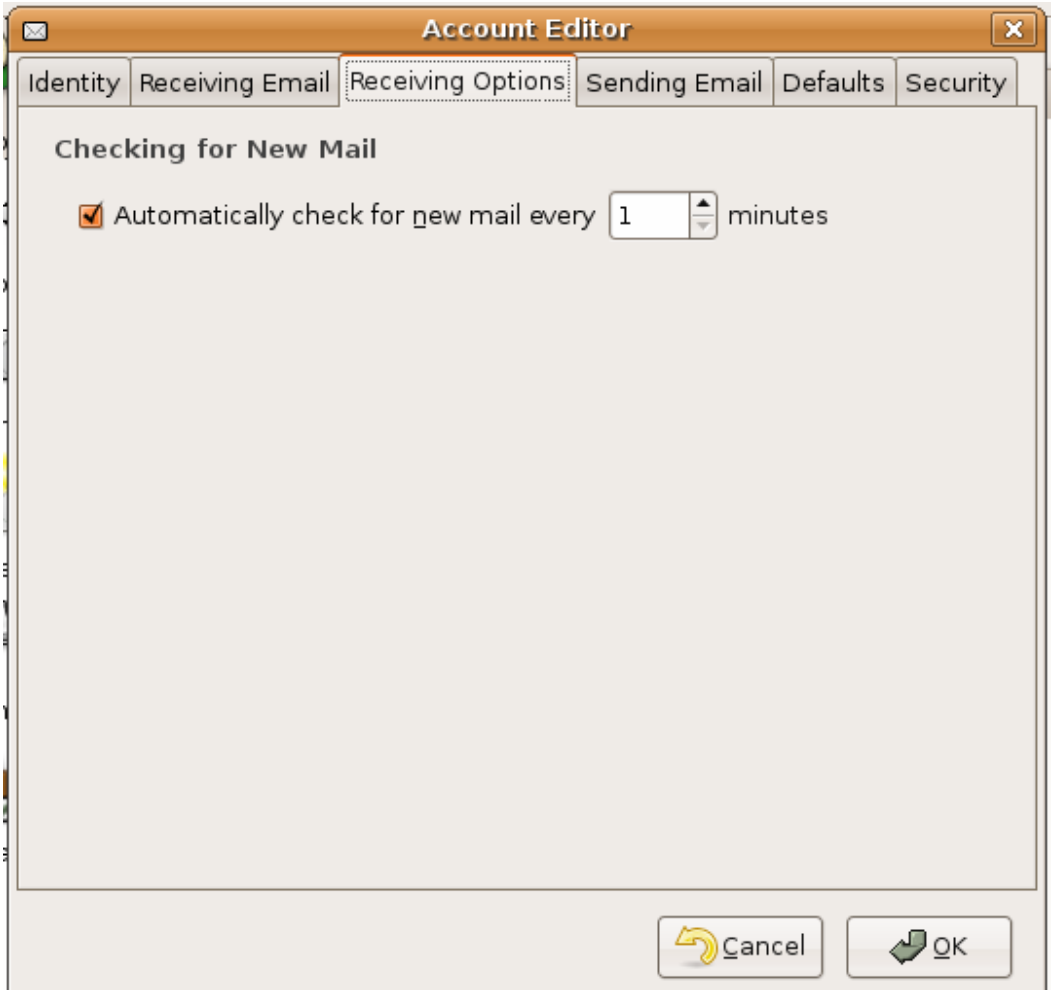

Figure 11: Receiving options.

We selected a polling interval of the client MUA (which locally polls the local MTA) to its lowest value: one minute. Note that there is no need for the network interface to be connected or even on for this to happen, since it is simply a local poll of the /var/spool/mail/ericsson directory to check for new mail on this computer.

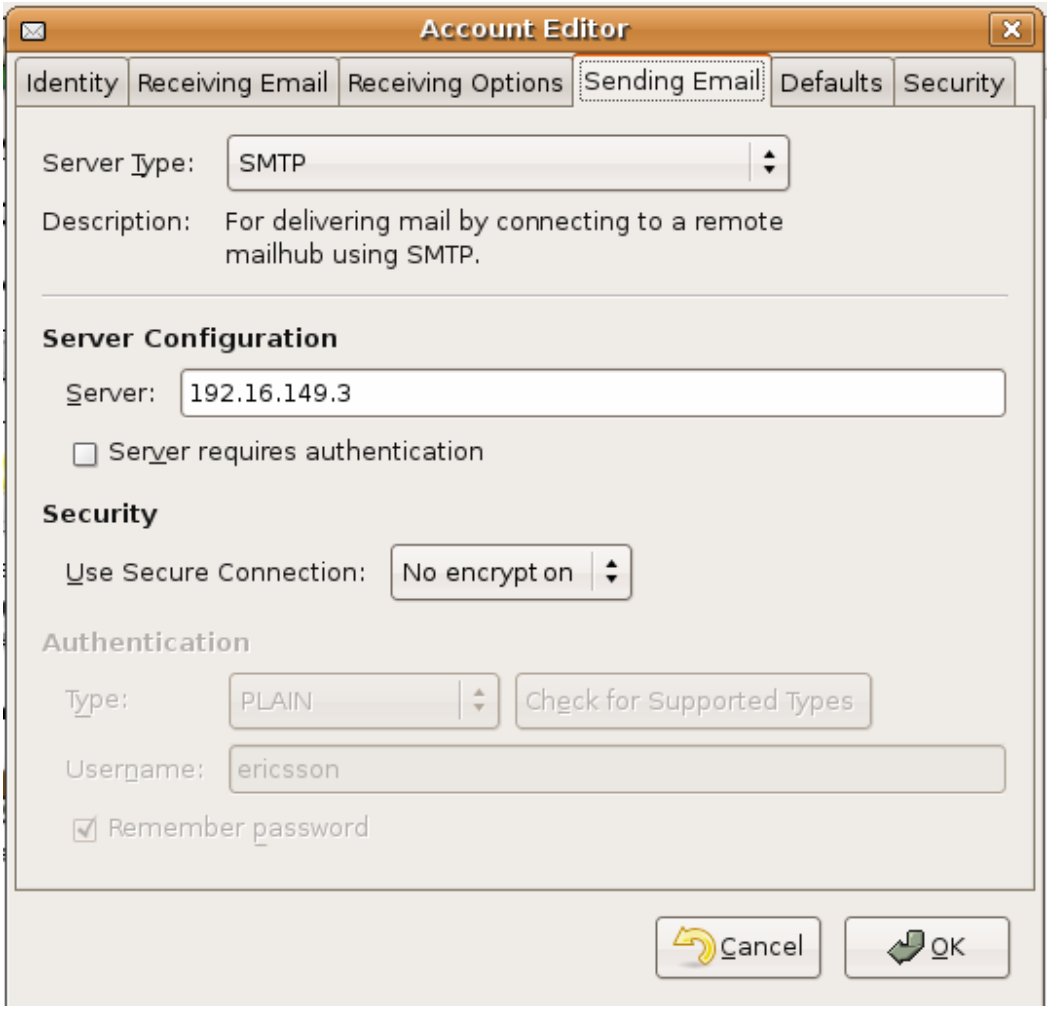

Figure 12: Sending email to mail relay machine

The MUA will use SMTP to send outgoing mail directly to the specified mail relay machine's IP address, using the specified protocol for sending mail.

Now the client machine is ready to send and receive mail.

7.2 On a Microsoft Windows machine

For Microsoft Windows client machines we needed to set up a SMTP listener that will listen for incoming mail and a POP server, which the mail client uses to fetch the mail locally.

For windows, we used a Java based mail server (See Appendix A), which is easy to set up with very few parameters to be configured. It comes as a zipped package, which needs to be unzipped on the primary drive. Certain parameters such as server name, server IP address, and user lists can be configured through its mail.conf file. For further details regarding the mail server configuration, please refer to appendix A.

We used Microsoft's Outlook as the mail client for the user to send and receive mail. The mail client setup is similar to the usual configuration of such a mail client, with a slight difference being that for the Incoming POP server we gave the localhost address, since the client will only need to check for new mails from this local mail server. Again, it should be noted that this traffic is only local - as it is actually the MTA which receives the mail

8. Event notification and Filtering

8.1 When will the user know that they have gotten mail

Once the email has been sent to the mobile terminal, the issue of notifying the user arises. One method is to use an event notifier, such as an external program built directly into the mail server, which pops up a message when a new message arrives, or xbiff [39], which would continuously listen to the incoming mail spool file and notifies the user of new emails. It should be noted that the user generally is not be interested in reading each and every mail as it is received. Although these notifiers would minimize the notification delay, it is also important to note that using a notifier is not recommended due to two reasons. Firstly, getting frequent notification would complicate things for the user of a hand-held device and secondly, the value of a solution is said to decrease per extra click. The users would like to avoid needing to click several times; especially if they are not interested in the mail that the mail notifier has announced.

Another method for email notification is to use the mail client's default notification mechanism. In the case, the mail client checks for new mail messages according to a given (adjustable) interval of time. In our case, since the client would be checking for the new mail locally, hence no network traffic would be generated, which implies that the mobile terminal's radio interface could remain in a low power state for longer. Thus this interval can be set to the minimum value, which for most mail clients, including Microsoft Outlook is 1 (one) minute. So the maximum delay for mail reception that a user might encounter should be less than 60 seconds.

Since MS Outlook includes a smart mail notification mechanism we used this on the Microsoft Windows client. Certain parameters can be set to customize this new mail notification.

Figure 13: new mail receives notification with subject and sender address

Figure 13 shows the notification that is displayed for a new mail message. It contains the sender's address, subject of the mail, and a part of the message body to let the user know the content at a glance. This enables the user to minimize the interruption due to mail message that are not high priority and for which the user would not like to bother currently. This notification can be set to appear and automatically disappear after a specified period of time. Thus the user need not click at all, unless they want to see the rest of this message right now!

Figures 14 and 15 shows the screen shots for scheduling mail check and to set the duration for which the pop-up for incoming mail would remain visible.

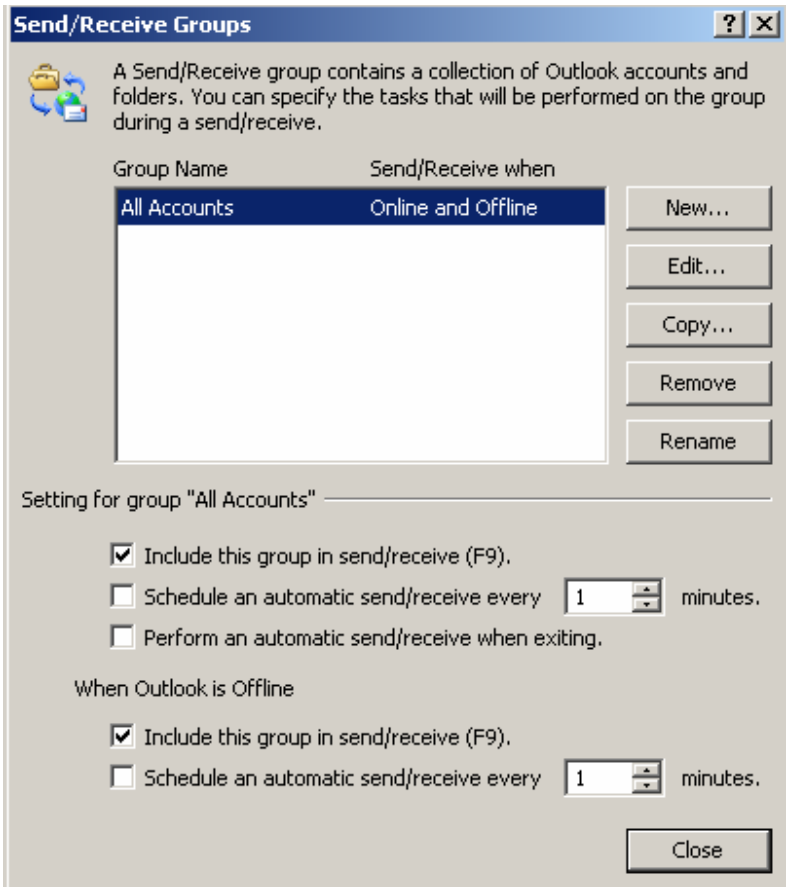

Figure 14: How to create a mail account

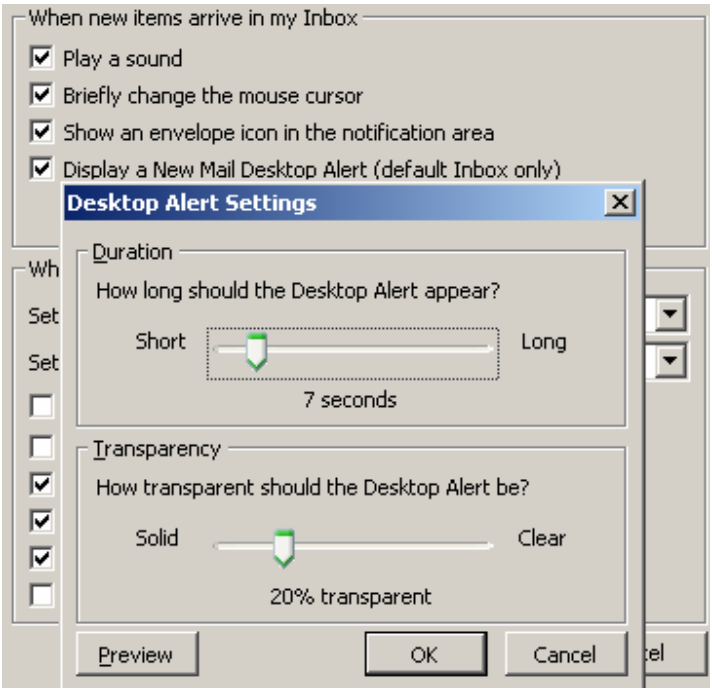

Figure 15: Mail notification display settings

8.2 Filtering out unwanted mail

In our case we are primarily concerned with hand-held devices with limited processing power and limited memory capacity, but with GPRS connectivity. It is therefore desired that the mail filtering be done at the server side of the network. As few users will want to waste their mobile device's resources due to spam, nor would they be willing to pay for the GPRS usage required to receive this spam.

Nowadays server side filtering has matured quite a bit and in a intra-company environment, it is relatively easy to apply basic mail filters. It might also be possible to filter mails for individual users according to their specific preferences. Applying server side filtering reduces unwanted network traffic and saves the terminal's resources.

9. Evaluation

9.1 Functional testing

In our tests we have attempted to show that the system functions correctly and to measure the network bottlenecks, measure its performance, and to determine its degree of reliability and robustness.

We performed several individual and combined tests using our prototype. We began by individually test each of our different servers. Without a registered domain, we could only perform local testing. Thus we formed a private local network by connecting all three computers to a router. We used one computer as a mail relay machine where the DNS server was configured, and two additional machines acting as clients: running Linux and Windows respectively.

For our DNS testing, we assigned each machine a name and IP address. We also implemented reverse lookup. We manually checked the DNS configuration using the command “nslookup”.

To check that the mail server was running properly, we performed several different kinds of tests. From our mail relay machine, we sent mail to each of the client machines. Then we examined the “/var/log/mail.log” file to check the mail delivery process. This mail.log indicated that the mail delivery to the client machines was successful.

Once the mail relay machine was working properly, we configured our client machines to send and receive mail through this mail relay machine. From the windows client we sent mail to the Linux client and from the Linux client to the windows client. At the windows client we ran our java mail client, which listens to SMTP port. When the mail relay received mail for this client it established a SMTP connection to the client and delivers the mail.

For our demonstration in a real life scenario, we registered our clients with a free domain name “no-ip.org” at “no-ip.com”, a DDNS provider. Thus each of our clients had a public domain name. We gave our mail relay machine a public IP address and placed it in another room. To provide Internet connectivity to our clients, we used a GPRS PCMCIA card. This time we send mails from yahoo, hotmail, and gmail accounts to our client machines. In each case, our client machine received this mail through our mail relay machine.

9.2 Performance analysis and comparison with existing mail delivery

9.2.1 Latency

The existing system which we compare to, polls for mail from a server. In our prototype system we will receive mail directly through SMTP instead of polling for it from a server. We did the test with two client machines. From one client machine we poll for mail from server and from another we set our mail client to receive mail through SMTP. During a thirty minute interval we sent three mail messages to each of these client machines. Each message was the same size and each of the MUAs was set to check for new mail every one-minute. We captured the packets as they traversed the network using ethereal [40].

Table 2: Ethereal results for 30 minutes interval

Mail Systems	Network Action (no. of bursts of traffic)	Mails received	Bytes transferred (KB)
Polling	30	3	122
Last Hop SMTP	3	3	34

As shown in table 2, polling results in accessing the GPRS network 30 times during the half hour interval. In terms of mobile phones which would mean going into high power state 30 times (each time bytes are exchanged over the network) as compared to 3 times for Last Hop SMTP that performs network action only when a mail is received. The difference in data transferred indicates the amount of extra POP traffic overhead. With a cost of 15kr/MB the cost of this extra traffic will be 1.320 kronor. To put this into context, even at this rate for 12 hours per day and 30 days per month + a fixed 89 kr per month for all traffic between (19:00 and 07:00) {based on Telia's current offering} - the extra cost would be 1039 kronor. If you were willing to wait to a maximum of two minutes for mail during the day, this would drop to 564kr. For comparison if these messages were typical of the size of messages which the user receives, then they each message would cost 0.510 kronor. So the difference in polling vs. not polling for the 30 minutes is less than the cost of 3 such messages.

Latency is inversely proportional to this extra network traffic. If we increase the time between checking for mail in our existing system then latency will increase. Whereas in our proposed system, we are receiving mail immediately through the mail relay machine without any added latency and the polling interval of the mail client is decoupled from the traffic generated. The following graphs clearly show the difference between the two systems in terms of latency.

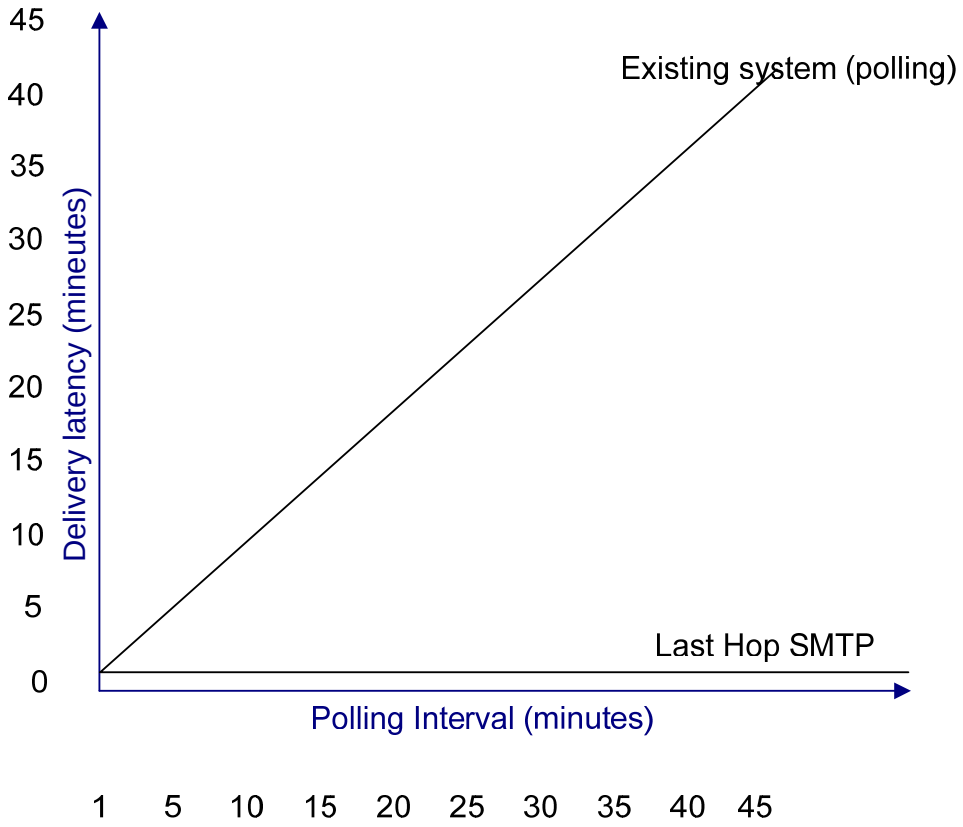

Figure 16: Comparison of delivery latency and polling interval between Polling and Last Hop SMTP.

Figure 16 shows that the delivery latency is directly proportional to the polling interval for the polling solution. If we increase the polling interval then delivery latency will increase as well. On the other hand, Last Hop SMTP has essentially a fixed delivery latency value (this delay will be roughly 1/2 second on average with a variance of roughly 1 second). By default, we set the MUA to check for new mail after each minute. Thus the maximum delay to receive a mail in the user's inbox, once it has been relayed through the SMTP server is approximately one minute (since Last Hop SMTP does not utilize polling to get the mail to the client machine and thus is independent of the polling interval).

The constant delay for Last Hop SMTP shown in the above figure is calculated from the time difference of when the mail is delivered to the terminal through the SMTP listener and the time when the mail is fetched by the MUA to be displayed to the user. Figure 17 illustrates the mail reception process through SMTP.

In figure 17 to start the mail transfer, the mail relay sends a HELO command to begin the SMTP session. The Java mail server is programmed to check for EHLO (extension to HELO command) first and if the string from the mail relay does not match, then try by checking HELO. Once the session is acknowledged by the mail server running on the mobile terminal, user verification is

handled by the mail relay machine configurable through the ‘Mail.conf’ file, which is the configuration file for the Java mail server.

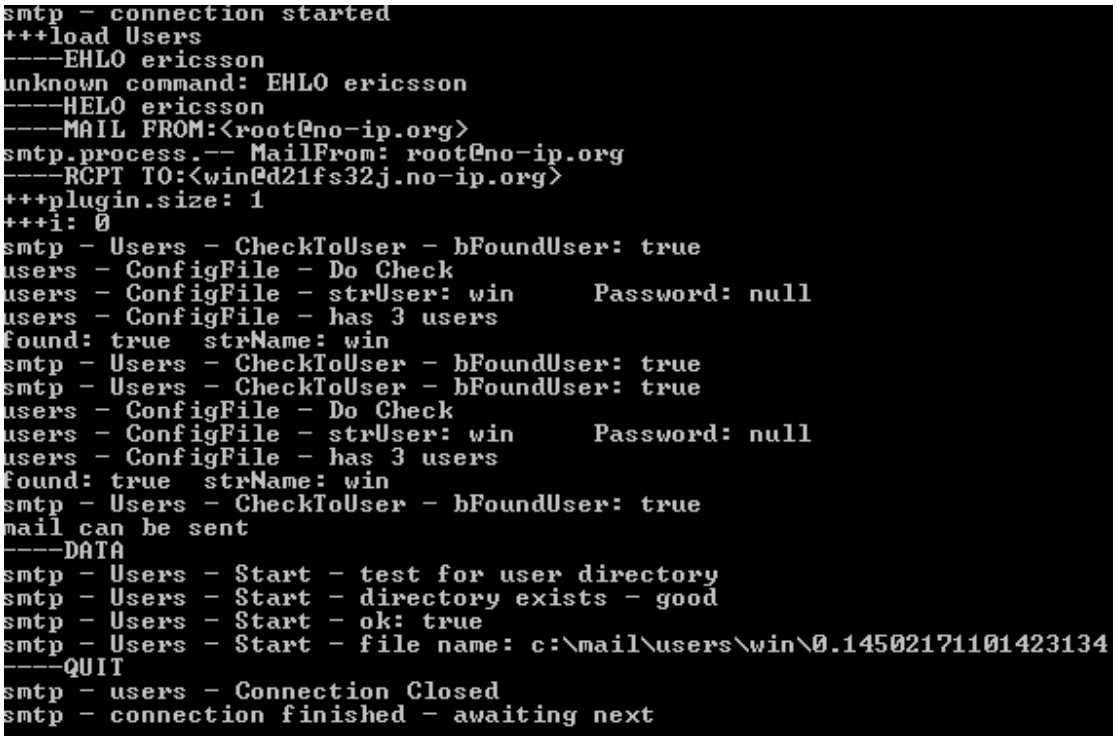

Figure 17: Mail reception through SMTP. Snapshot from the execution of java mail server.

After the ‘MAIL FROM’ and ‘RCPT TO’ users verification, the DATA command is issued which deals with locating the user mail directory and storing the mail body. Finally, the SMTP session is closed by issuing the ‘QUIT’ command and the mail server goes back to listening phase.

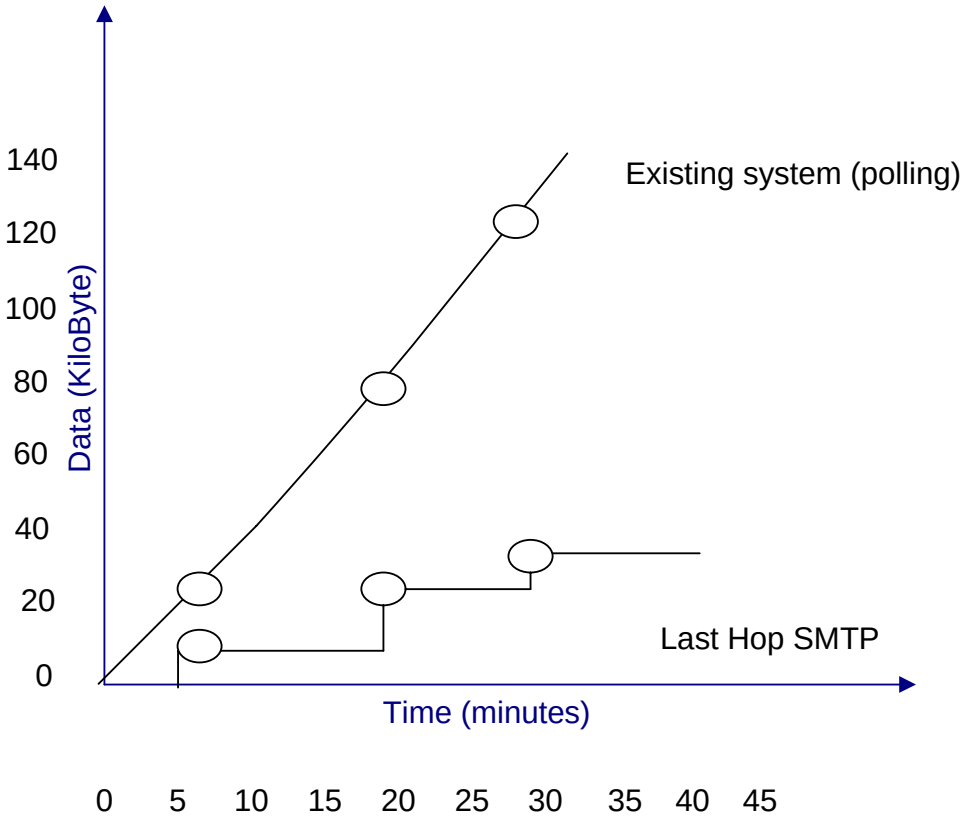

Figure 18: Comparison for amount of data transfer between polling and Last Hop SMTP

Figure 18 shows that, if we check for the mail after every minute, then in polling system, approximately 2.93 KB of extra POP traffic will be generated per minute. If three mails are received in a span of 30 minutes, then 88 KB of extra data traverses the network in case of polling. On the other hand, with Last Hop SMTP only the actual mail data and a few bytes of SMTP traffic associated with each mail which totals 34 KB, will be transferred.

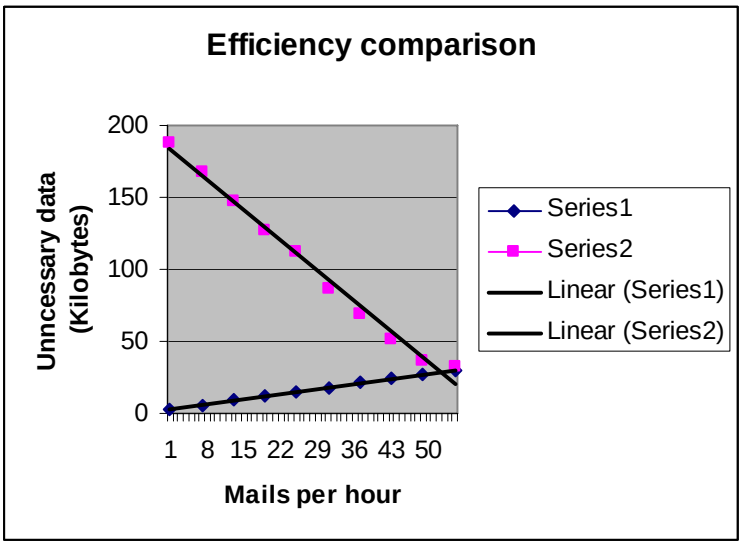

Figure 19: Efficiency comparison between Last Hop SMTP and polling

Figure 19 shows the efficiency comparison between the Last Hop SMTP and polling methods based on our testing criteria of polling every minute. It is evident that our solution is ideal in case of a low mail density. The figure assumes uniform mail arrival so as to calculate the number of empty polls in terms of polling.

Table 3: Efficiency comparison, mail density against additional data sent

Mails per hour	Last Hop SMTP Unncessary Data (Kbs)	Polling Unncessary Data (Kbs)
6	3	188
12	6	168
18	9	147
24	12	127
30	15	112
36	18	87
42	21	69
48	24	51
54	27	36
60	30	32

As we move near to 1 mail each minute the efficiency of both systems becomes almost leveled. The above table illustrates the values against which the graph has been plotted.

9.2.2 Power Consumption

As stated earlier, the battery drain is directly proportional to the number of times the mobile phone sends or receives data over the network. The mobile remains in a high energy state for some period of time after sending or receiving the last byte of a data transfer. If the time in the high power state is one minute, then in the above scenario, the terminal would remain in high

power state for almost 24 hours even if it does not receive mail messages. For a standard phone, this means having to recharge the phone 3 or 4 times a day since the high power state normally corresponds to the talk time a mobile is capable of delivering, which is practically not possible.

Thus, Last hop SMTP is a solution which not only minimizes the latency associated with the mail delivery, but it also preserves the battery power since the only network action that the terminal will perform would occur when a mail arrives. Otherwise, the terminal will remain in the standby mode as usual.

To check the actual power status of user terminal (laptop in this case) we wrote a small program in C++ based on the GetSystemPowerStatus API [41]. Please see Appendix B for coding details.

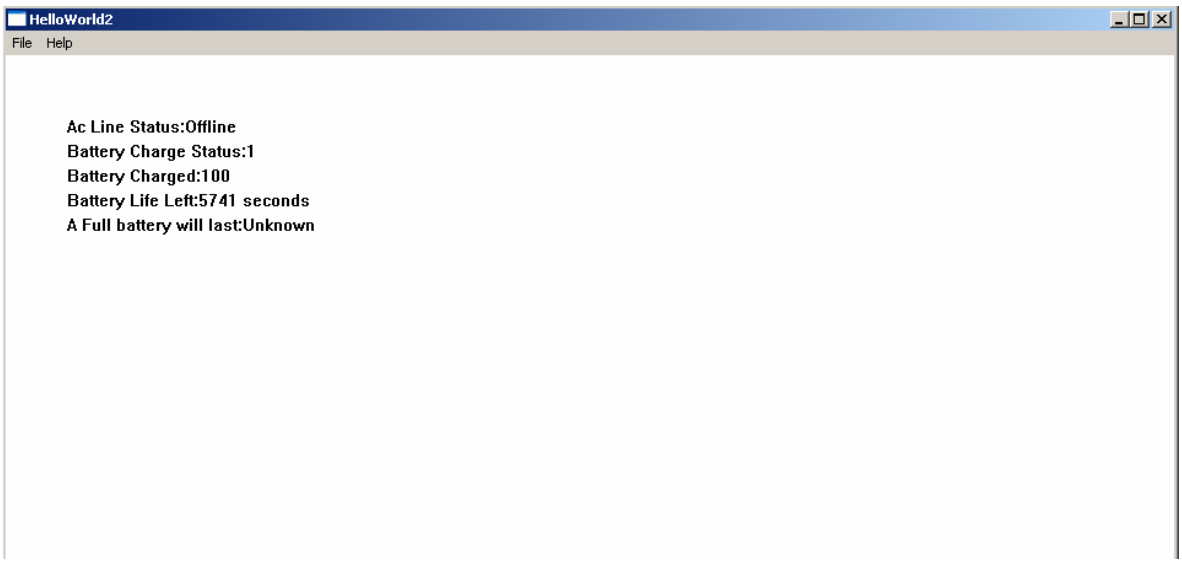

Figure 20: Power status at the beginning of test

Figure 20 shows that the battery is fully charged. AC line status offline means the external power source is not connected and laptop's own battery is in use. Total battery life is 5,741 seconds (95 minutes).

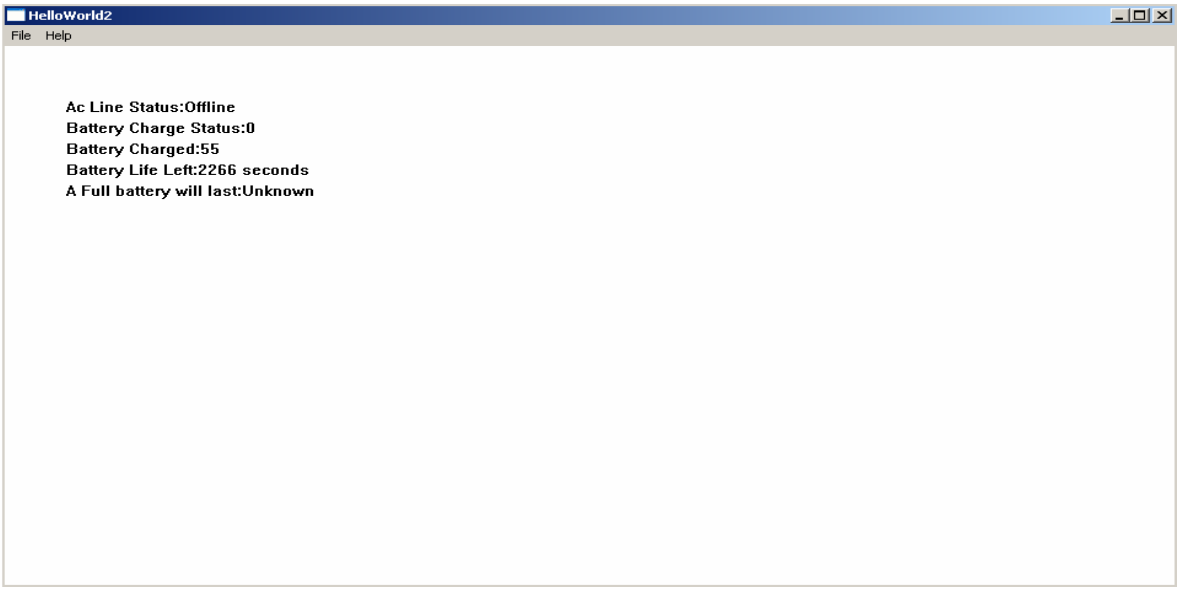

Figure 21: Power status after 30 minutes of polling

Figure 21 shows the battery status after polling for 30 minutes, checking for mail each minute. So, after 30 minutes it drains 45 percent of battery's life.

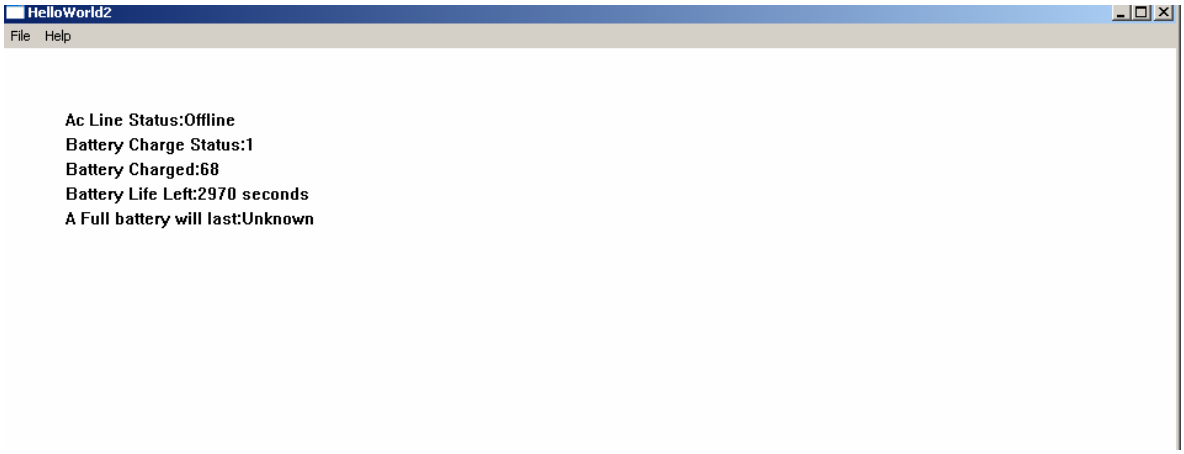

Figure 22: Power status of Last Hop SMTP after 30 minutes.

Figure 22 shows the battery status for Last Hop SMTP which stands at 32 percent drainage in 30 min.

Table 4: Battery drain as a function of time

Time (in minute)	Polling system	Last Hop SMTP	Idle laptop operation
1	1.5%	1.067%	1.053%
30	45%	32%	31.59%
60	90%	64%	63.18%
66.67	100%	71.14%	70.2%
93.7		100%	98.67%
95			100%

From the above table it is clearly shown that using Last Hop SMTP battery drainage is slower than polling. The battery lifetime for the laptop when no operation is being done is 95 minutes. Battery drainage time in case of Last Hop SMTP is 93.7 minutes, a minute and a few seconds earlier then idle state battery drainage time. Whereas in case of polling, the battery drains completely in less the 67 minutes, reducing the battery life time by approximately half hour. It should be noted that these values are based on a laptop connected via the GPRS connection using a PCMCIA card.

9.3 Scaling through LDAP

As described in section 6.5, LDAP can be used to scale the Last Hop SMTP system for large number of users. A detailed study to analyze the performance of LDAP directories has been done at Columbia University. The performance of LDAP was studied in a dynamic environment, with frequent directory access.

This study concluded that under normal operating conditions, a directory with 10,000 488 byte entries, and a cache size of 10,000 entries - the LDAP server has a response latency of 8 ms at loads up to 105 search requests per second, and a maximum throughput of 140 search requests per second. Out of the total response latency of 8 ms, 5 ms comes from the processing latency, 36% of which is contributed by back-end processing (entry retrieval from the database), and 64% by front-end processing. In general, at high loads, the connect latency increases sharply to dominate the overall response, and eventually limits the server throughput. Consequently, a change in the processing time due to changes in system parameters has a relatively smaller effect on the maximum throughput [42].

9.4 Multiple mail folders

As described in sections 5.2 and 5.3, while it is possible to have multiple mail folders they are all on the mobile terminal. Which means that unless your mail relay is configured such that it forwards a copy of the mail messages to some other mail server, in the event of loosing or destroying the mobile terminal would destroy all the mail as well.

Additionally, the proposed system does not provide device independence to the users. The users are not free to move between devices and access their mail as they can do when using IMAP(as described in section 6.1.2).

10. Alternate Solution

An alternate solution for delivering emails to the user terminals with low latency and power consumption can be realized by replacing the SMTP listener with a simple mail notification listener on the client device. Once a mail notification is received, then the client can fetch the message. This can simply be a one byte notification from the mail relay of the arrival of a new mail message. Figure 23 shows the prototype design for this alternate solution.

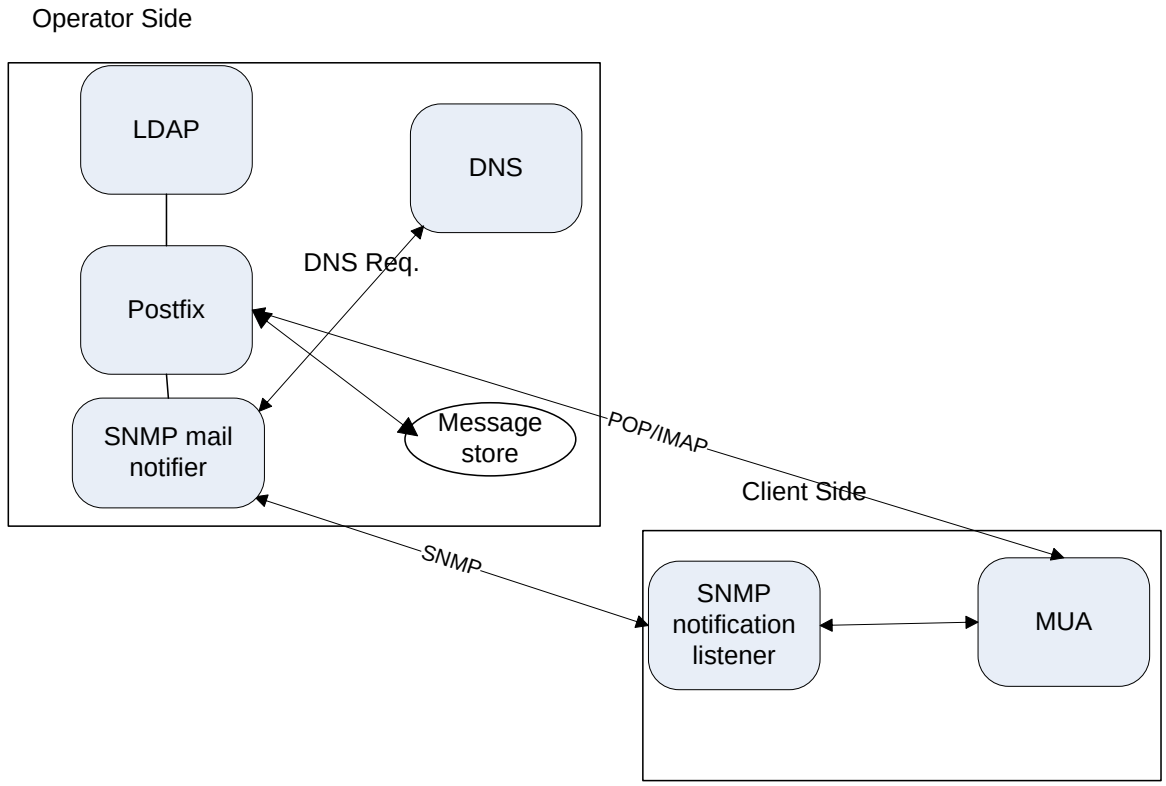

Figure 23: Mail notification through SNMP

A SNMP mail notification can be sent to the client as soon as the mail is received by the mail server. Once the notification is received, the client can then poll for the mail using the traditional polling approach. This model solves the latency problem by notifying the client as soon as a mail arrives at the mail server and reduces power consumption by eliminating the need for empty polls. Like Last Hop SMTP, there will only be network traffic when there is a new mail. One advantage in this approach is that, the client side software is much simpler. Another advantage is that the mail does not necessarily have to be downloaded on the device, the users can view their mails using IMAP which gives the users device independence.

In comparison with Last Hop SMTP the amount of network traffic in this case can be as little as one byte of payload, thus this represent the lower limit in the minimal amount of traffic which is sent to provide low latency notification of mail arrival. If the application were to encode some additional information in this byte, such as what type of message and what priority it was, then the client might even purposely delay further processing of this mail until there was other mail

which was important enough to actually get. This is another way of lowering the total traffic while still providing low latency for the e-mail messages which the user is most interested in receiving. Of course in the limit, this additional coded information approaches that of IMAP and IMAP IDLE, hence such an IMAP implementation would be very interesting to evaluate; however, it was outside the scope of the project for our employer.

11. Future work

After the successful implementation of the Last Hop SMTP prototype, it is worth thinking about what future work could be done. One of the areas can be to remove the small amount latency associated with Last Hop SMTP. Another is server side filter according to priority, subject, sender, etc, in order to decide which mails should be forwarded to the mobile phone. Yet another area that can be considered would be to allow mail notifications to be sent for multiple mailboxes a user might own.

IMAP IDLE [43] can be used to remove the above mentioned delay by removing the need for the MUA to check for new mail. IMAP IDLE will allow the server to notify the MUA when a new mail is received and the mail will directly be injected into the client's mailbox.

Procmail [44] can be used to make the system more user friendly. The users will have control of which messages (based upon user controlled mail filtering) they want receive on to their mobile terminal directly.

Another feature of using procmail is that, it can remove the email attachments and store them at the mail relay server. The user receives a notification about the attachment and can choose to download the attachment to their mobile terminal if they wish.

A possible future enhancement to the Last Hop SMTP prototype will be the addition of a Push Proxy Gateway [45] in the architecture. This enables email delivery even if the terminal does not have an active PDP context and is not registered with a DNS server and the DNS server receives a query for that particular user. The DNS server in this case will send a Push Access Protocol (PAP) request to the Push Proxy Gateway. The Gateway will then instruct the mobile terminal Over The Air (OTA) using the mobile's IMSI number to establish a PDP context and register itself with the DNS server. Push Proxy Gateway does this with the help of IMSI to FQDN mappings.

Session Initiation Protocol (SIP) with LDAP can handle the scalability problem and forward the email message to the user's terminal avoiding the addressing problem caused by NAT.

SIP can be an alternate solution for this prototype. In this case, a location server, a SIP proxy server, and a SIP redirect server [24] will handle the problem and satisfy the goal of receiving email message without polling from the user terminal.

12. Conclusion

In this thesis we have studied and implemented a new email delivery system based on SMTP. A functional prototype of the system was tested and compared with the existing system. We concluded that Last Hop SMTP is a better solution then polling since it reduces power consumption and latency. On the other hand, with polling one needs to make a trade off between on timely email delivery and battery lifetime. The current solution can easily be implemented on most laptop computers running linux or Microsoft Windows operating systems.

References

[1] S.H. Maes, C. Kuang, “et al.” Push Extensions to the IMAP Protocol http://tools.ietf.org/wg/lemonade/draft-maes-lemonade-p-imap-12.txt - 2007.01.13
[2] Research in Motion (RIM), Home page http://www.rim.com/ - 2007.02.24
[3] Push e-mail, Wikipedia http://en.wikipedia.org/wiki/Push_e-mail - 2007.02.24
[3] Blackberry Enterprise Server (BES), Wikipedia http://en.wikipedia.org/wiki/BlackBerry_Enterprise_Server - 2007.02.25
[5] Blackberry MDS Services, Home page http://www.blackberry.com/developers/promos/mds.shtml - 2007.02.28
[6] How a Blackberry works, Howstuffworks http://electronics.howstuffworks.com/blackberry.htm - 2007.02.24
[7] Triple Data Encryption Standard (DES), Wikipedia http://en.wikipedia.org/wiki/Triple_DES - 2007.02.25
[8] Advanced Encryption Standard (AES), Wikipedia http://en.wikipedia.org/wiki/Advanced_Encryption_Standard - 2007.02.25
[9] Microsoft Exchange, Wikipedia http://en.wikipedia.org/wiki/Microsoft_Exchange - 2007.02.25
[10] Lotus Domino, Wikipedia http://en.wikipedia.org/wiki/Lotus_Domino - 2007.02.25
[11] Novell GroupWise, Wikipedia http://en.wikipedia.org/wiki/Novell_GroupWise - 2007.02.25
[12] Jonathan B. Postel, Simple Mail Transfer Protocol, RFC 821, IETF, Internet Society, August 1982 http://www.ietf.org/rfc/rfc0821.txt
[13] Jonathan B. Postel, Simple Mail Transfer Protocol, RFC 821, SMTP Model www.freesoft.org/CIE/RFC/821/2.htm
[14] David H. Crocker, Standard for the format of ARPA Internet Text Messages, RFC 822, IETF, Internet Society, 1982.08.13 http://www.ietf.org/rfc/rfc0822.txt
[15] P. Hoffman, SMTP service extension for secure SMTP over TLS, RFC 2487, IETF, Internet Society, January 1999 http://www.ietf.org/rfc/rfc2487.txt
[16] J. Myers, SMTP service extension for Authentication, RFC 2554, IETF, Internet Society, March 1999 http://www.ietf.org/rfc/rfc2554.txt
[17] J. Myers, Carnegie Mellon, M. Rose, Post Office Protocol - version 3, RFC 1939, IETF, Internet Society, May 1996 www.ietf.org/**rfc**/**rfc**1939.txt
[18] M. Crispin, Internet Message Access Protocol – version 4rev1, RFC 3501, IETF, Internet Society, March 2003 www.faqs.org/**rfc**s/**rfc**3501.html
[19] Authenticated POP http://isp.webopedia.com/TERM/A/APOP.html - 2006.12.23
[20] Hakan Granbohm, Joakim Wiklund, General packet radio service, Ericsson review www.ericsson.com/ericsson/corpinfo/publications/review/1999_02/files/1999024.pdf - 2006.0.11
[21] Introduction to GPRS, Wikipedia, http://en.wikipedia.org/wiki/Gprs - 2006.10.12
[22] GPRS white paper, Cisco www.cisco.com/warp/public/cc/so/neso/gprs/gprs_wp.pdf - 2006.10.11
[23] GPRS white paper, VOCAL technologies, www.vocal.com/data_sheets/**gprs**5.html - 2006.10.15
[24] WAP Push, Wikipedia, 2006.12.24 http://en.wikipedia.org/wiki/Wap_push
[25] Short Message Service, Wikipedia, 2007.02.22 http://en.wikipedia.org/wiki/Short_message_service

[26] Session Initiation Protocol, Wikipedia, 2007.02.21
http://en.wikipedia.org/wiki/Session_Initiation_Protocol
[27] O'Reilly Postfix: The Definitive Guide, Kyle D. Dent, December 2003.
[28] Enhanced Data Rates for GSM Evolution, Wikipedia
http://en.wikipedia.org/wiki/Enhanced_Data_Rates_for_GSM_Evolution - 2006.12.02
[29] The Postfix Home Page http://www.postfix.org – 2006.10.14
[30] Sendmail, Home page http://www.sendmail.org – 2006.10.14
[31] Exim Internet Mailer http://www.exim.org
[32] Berkeley Internet Name Domain, Wikipedia http://en.wikipedia.org/wiki/**BIND** - 2006.10.14
[33] Push Mail Technology [www] http://www.webindexing.biz/articles/Pushtechnology.htm
[34] Ericsson internal research (EAB 05:043836 Uen RevB)
[35] Postfix optional tables www.informatik.uni-bonn.de/pub/software/**postfix**/regexp_table.5.html
[36] DNS zones in LDAP
http://www.section6.net/wiki/index.php/Configuring_DNS_zones_in_LDAP - 2007.02.02
[37] Mozilla thunderbird www.**mozilla**.com/en-US/**thunderbird** - 2007.01.11
[38] Evolution mail http://www.gnome.org/projects/evolution/ - 2006.10.27
[39] xbiff, manual page http://www.xfree86.org/current/xbiff.1.html - 2006.11.09
[40] Wireshark Network Analyzer http://www.wireshark.org – 2006.11.13
[41] GetSystemPowerStatus API, Microsoft's API for power management
http://msdn2.microsoft.com/en-us/library/aa372693.aspx - 2007.01.31
[42] Xing Wang, Henning Schulzrine, Analysis of LDAP performance [www]
http://www.cs.columbia.edu/~hgs/papers/Wang0006_Measurement.pdf - 2007.02.23
[43] IMAP4 IDLE command http://tools.ietf.org/html/**rfc**2177
[44] Procmail, Home page http://www.procmail.org – 2006.10.17
[45] Push Proxy Gateway http://en.wikipedia.org/wiki/Push_Proxy_Gateway - 2007.02.18

Appendices

Appendix A – Java Mail Server

Java mail server can be downloaded from http://sourceforge.net/projects/crsemail/.

Once the zipped file is downloaded, unzip it in (c:) drive. This will create a folder named “mail”. Under the mail folder the configuration file for the server called “Mail.conf” is placed. A sample file is given below. It should be noted that only the parameters in **bold** fonts were changed for the server to fit our prototype.

Mail.conf

```
--------------------------------------------------------------------------
# this is the configuration file for the mail server
# if a key is given twice then the system merges the two together
# all keys are case insensitive

# name(s) of the server
# this is how the system knows if incomming mail should be stored
# or forwarded to someone else, if forwarded then the first server listed
# is who the mail forwarder says the mail is from +++ change this later +++
Server=ericsson.no-ip.org,
Server=d21fs32j.no-ip.org,
Server=192.16.149.74

# maildir is the directory where the mail server is installed
# the program files must be in: maildir\bin
maildir=c:\mail

# setup the ports to listen to
# standard setup is smtp = port 25, pop3 = 110
# however some ISP's (netzero) block one or more of these ports.
# to use more than the standard ports separate port numbers with a comma
# I STRONGLY recommend having at least one smtp port be 25 or you will
# find that mail from the outside will never arrive
smtp.port=25,
smtp.port=8078

pop3.port=110,
pop3.port=8079

# limit the number of threads that are used in the system.
# This is usefull to keep performance up.  Without this the server could
# get too busy handling many users to be usefull to any of them.
# If the max number of threads has been reached then future visitors
```

```
# will have to wait till a thread is available
# If a client get's timed out while waiting then he can try again later.
# currently un-implemented
Threads=50

# the error log is used to store information generated as the system runs
# mostly just debug information

ErrorLog.file=c:\mail\error.log

# location of the nslookup program.  This is used to get the MX records from
# a dns server.
# Need an all java version of this instead since right now the output is parsed
# and if microsoft changes the output of their program then the parsing will break down
# no longer used since dnsjava is used instead
# smtp.dns.nslookup=c:\winnt\system32\nslookup.exe

#
# setup users names
#

# the plugins to load user names
# if users names should be gotten from a config file then use plugin ConfigFile
Users.plugin=ConfigFile,
# if users names should be gotten from a database then use plugin DB
#Users.plugin=DB

#
# options to be passed to the plugins are passed as: Users.<name of plugin>.<option>
#

# options for what database to get users from

# the driver to use - must be a valid jdbc driver -
#Users.plugin.DB.Driver=sun.jdbc.odbc.JdbcOdbcDriver
# the connection string to pass to the driver
#Users.plugin.DB.Connection=jdbc:odbc:myusers
# user name for the database
#Users.plugin.DB.User=sa
# password for the Database
#Users.plugin.DB.Password=
# table used to store name and password
#Users.plugin.DB.Table=users
# field used for the user name
#Users.plugin.DB.UserNameField=username
# field used for the password
#Users.plugin.DB.UserPasswordField=password
```

```
# user's that are not in the database, or all users if not using a database
# list user name followed by password
Users.user=win,123456,
Users.user=ericsson,123456,
#Users.user=mohsoh,123456,
#Users.user=webmaster,test,
# this is the reply address if a message bounces
Users.user=root,123456

# All plugins.  All incomming mail is sent to each plugin that wants it.
# ie. if the same user exists for both the users and the forward mail plugins
# then the message is both stored, and forwarded.

# ForwardMail is used to have all mail sent to it automatically forwarded.
# to another email address
#smtp.plugin=ForwardMail,
# users on this server
smtp.plugin=Users,
# forward incoming mail destined for another server
#smtp.plugin=NonLocal


#
# options to be passed to the plugins are passed as: smtp.<name of plugin>.<option>
#

# user names for the mail forwarder are setup here
#smtp.plugin.ForwardMail.user=win,win@kth.test,
#smtp.plugin.ForwardMail.user=george,george@netscape.net

# servers that are not forwarded to
# All mail to these places will be rejected
#smtp.plugin.NonLocal.Forbid=localhost,
#smtp.plugin.NonLocal.Forbid=127.0.0.1,
#smtp.plugin.NonLocal.Forbid=me.com
```

After the changes have been made, place the file under ‘C:\mail\bin’ and type the following commands in the Command Prompt to start the mail server.

```
# cd c:\mail\bin
# java mail Mail.conf
```

Mail server on windows machine is up and ready to receive mail through SMTP. Please ensure that no error messages have been displayed, in case of which recheck the steps taken to sort out the problem.

Appendix B – API for checking system's power status

The API to check the power status of the system was developed on C++. This API was built using the Microsoft's GetSystemPowerStatus API (http://msdn2.microsoft.com/en-us/library/aa372693.aspx)

Code

```
--------------------------------------------------------------------------
// HelloWorld2.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "HelloWorld2.h"
#include <stdio.h>
#define MAX_LOADSTRING 100

void ReportPowerStatus(HDC hdc);
// Global Variables:
HINSTANCE hInst;                                    // current instance
TCHAR szTitle[MAX_LOADSTRING];                      // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];                // the main window class
name


// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

enum ACLineStatus {Battery= 0,AC=1,Unknown=255};



int APIENTRY _tWinMain(HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       LPTSTR    lpCmdLine,
                       int       nCmdShow)
{
        // TODO: Place code here.
        MSG msg;
        HACCEL hAccelTable;

        // Initialize global strings
        LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
        LoadString(hInstance, IDC_HELLOWORLD2, szWindowClass, MAX_LOADSTRING);
```

```
	MyRegisterClass(hInstance);


	// Perform application initialization:
	if (!InitInstance (hInstance, nCmdShow))
	{
		return FALSE;
	}

	hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_HELLOWORLD2);

	// Main message loop:
	while (GetMessage(&msg, NULL, 0, 0))
	{
		if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
		{
			TranslateMessage(&msg);
			DispatchMessage(&msg);
		}
	}

	return (int) msg.wParam;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage are only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
	WNDCLASSEX wcex;

	wcex.cbSize = sizeof(WNDCLASSEX);

	wcex.style			= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= (WNDPROC)WndProc;
```

```
        wcex.cbClsExtra         = 0;
        wcex.cbWndExtra         = 0;
        wcex.hInstance          = hInstance;
        wcex.hIcon              = LoadIcon(hInstance, (LPCTSTR)IDI_HELLOWORLD2);
        wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
        wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
        wcex.lpszMenuName = (LPCTSTR)IDC_HELLOWORLD2;
        wcex.lpszClassName  = szWindowClass;
        wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);

        return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  HWND hWnd;

  hInst = hInstance; // Store instance handle in our global variable

  hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

  if (!hWnd)
  {
    return FALSE;
  }

  ShowWindow(hWnd, nCmdShow);
  UpdateWindow(hWnd);

  return TRUE;
}

//
//  FUNCTION: WndProc(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Processes messages for the main window.
```

```
//
//  WM_COMMAND - process the application menu
//  WM_PAINT         - Paint the main window
//  WM_DESTROY    - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
        int wmId, wmEvent;
        PAINTSTRUCT ps;
        HDC hdc;

        switch (message)
        {
        case WM_COMMAND:
                wmId    = LOWORD(wParam);
                wmEvent = HIWORD(wParam);
                // Parse the menu selections:
                switch (wmId)
                {
                case IDM_ABOUT:
                        DialogBox(hInst, (LPCTSTR)IDD_ABOUTBOX, hWnd,
(DLGPROC)About);
                        break;
                case IDM_EXIT:
                        DestroyWindow(hWnd);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
                }
                break;
        case WM_PAINT:
                hdc = BeginPaint(hWnd, &ps);
                // TODO: Add any drawing code here...
                ReportPowerStatus(hdc);
                EndPaint(hWnd, &ps);
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
        }
        return 0;
}
```

```
// Message handler for about box.
LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
{
	switch (message)
	{
	case WM_INITDIALOG:
		return TRUE;

	case WM_COMMAND:
		if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
		{
			EndDialog(hDlg, LOWORD(wParam));
			return TRUE;
		}
		break;
	}
	return FALSE;
}
void ReportPowerStatus( HDC hdc ){
	LPSYSTEM_POWER_STATUS sps =
(LPSYSTEM_POWER_STATUS)malloc(sizeof(SYSTEM_POWER_STATUS));
   BOOL result = GetSystemPowerStatus(sps);
	char res[10000];
	if( result){
		strcpy(res,"Ac Line Status:");
		if( sps->ACLineStatus == 0 ){
			strcat(res,"Offline");
		}
		else if( sps->ACLineStatus == 1 ){
			strcat(res,"Online");
		}
		else{
			strcat(res,"Unknown");
		}
		TextOut(hdc,50,50,(LPCTSTR)res,strlen(res));

		strcpy(res,"Battery Charge Status:");
		sprintf(&res[strlen("Battery Charge Status:")],"%d",sps->BatteryFlag);
		TextOut(hdc,50,70,(LPCTSTR)res,strlen(res));
		strcpy(res,"Battery Charged:");
		if( 255==sps->BatteryLifePercent){
			strcat(&res[strlen("Battery Charged:")],"Unknown");
		}else{
			sprintf(&res[strlen("Battery Charged:")],"%d %",sps->BatteryLifePercent);
		}
      TextOut(hdc,50,90,(LPCTSTR)res,strlen(res));
```

```
                strcpy(res,"Battery Life Left:");
                if( -1==sps->BatteryLifeTime){
                        strcat(&res[strlen("Battery Life Left:")],"Unknown");
                }else{
                        sprintf(&res[strlen("Battery Life Left:")],"%d seconds",sps-
>BatteryLifeTime);
                }
                TextOut(hdc,50,110,(LPCTSTR)res,strlen(res));

                strcpy(res,"A Full battery will last:");
                if( -1==sps->BatteryFullLifeTime){
                        strcat(&res[strlen("A Full battery will last:")],"Unknown");
                }else{
                        sprintf(&res[strlen("A Full battery will last:")],"%d seconds",sps-
>BatteryFullLifeTime);
                }
                TextOut(hdc,50,130,(LPCTSTR)res,strlen(res));
        }else{
                strcpy(res,"Failed to get Power Status");
                TextOut(hdc,50,50,(LPCTSTR)res,strlen(res));
        }
}
--------------------------------------------------------------------------------
```

Appendix C – Mail Relay configuration

Operating system: Ubuntu Linux
Mail server package: Postfix
Files to be manipulated: aliases, transport and main.cf

aliases (/etc/aliases)

The aliases file stores the FQDN to username mappings for the mail relay to forward incoming mail to each user's preferred address.

```
------------------------------------------------------------
# Added by installer for initial user
root:   ericsson

mohsoh: mohsoh@your-4e16zad0z2.no-ip.org
win:        win@d21fs32j.no-ip.org
test:        test@e000802d91004.no-ip.org
------------------------------------------------------------
```

transport (/etc/postfix/transport)

The transport file is used to define the mail relay server for a particular domain.

```
------------------------------------------------------------
no-ip.org     smtp:[ericsson.no-ip.org]
------------------------------------------------------------
```

main.cf (/etc/postfix/main.cf)

The main configuration file for postfix.

```
------------------------------------------------------------
myhostname = ericsson
alias_maps = hash:/etc/aliases
alias_database = hash:/etc/aliases
transport_maps = hash:/etc/postfix/transport
myorigin = no-ip.org
mydestination =ericsson.no-ip.org, no-ip.org, localhost.no-ip.org, localhost
relayhost =
mynetworks = 127.0.0.0/8 192.16.149.0/26 217.174.88.0/24 192.36.164.0/24
relay_domains =ericsson.no-ip.org
mailbox_size_limit = 0
recipient_delimiter = +
inet_interfaces = all
```

```
smtpd_sasl_local_domain =
smtpd_sasl_auth_enable = yes
smtpd_sasl_security_options = noanonymous
broken_sasl_auth_clients = yes
smtpd_recipient_restrictions = permit_sasl_authenticated,permit_mynetworks,reje$
home_mailbox = Maildir/
mailbox_command =
-------------------------------------------------------------
```

After all the files have been configured, postfix should be restarted.

```
# /etc/init.d/postfix restart
```

The postfix log can be viewed by,

```
# /var/log/mail.log
```

Appendix D – Abbreviations and Acronyms

APN- Access Point Name

BIND- Berkeley Internet Name Domain

BSC: Base Station Controller

BTS: Base Transceiver Station

CNAME- Canonical Name

DHCP- Dynamic Host Configuration Protocol

DDNS- Dynamic Domain Name Server

DNS- Domain Name System

GGSN- Gateway GPRS Service Node

GPRS- General Packet Radio Service

GSM- Global System for Mobile communication

HLR: Home Location Register

IMAP- Internet Message Access Protocol

IP- Internet Protocol

ISP- Internet Service Provider

LDAP- Lightweight Directory Access Protocol

MDA- Mail Deliver Agent

MS: Mobile Station

MTA- Mail Transfer Agent

MUA- Mail User Agent

MX- Mail Exchanger

NAT- Network Address Translator

PC- Personal Computer

PCMCIA- Personal Computer Memory Card International Association

PDA- Personal Digital Assistant

POP- Post Office Protocol

RR- Resource Record

SGSN: Serving GPRS Service Node

SIP- Session Initiation Protocol

SMS- Short Message Service

SMTP- Simple Mail Transfer Protocol

TCP- Transmission Control Protocol

TTL- Time To Live

VLR: Visitor Location Register

WAP- Wireless Application Protocol

COS/CCS 2007-11

www.kth.se