UNIVERSITY OF OSLO
Department of Informatics

Effects of Different IMAP Clients on Mailservers Performance

Master thesis

Teshome Dagne Mulugeta

Network and System Administration

Oslo University College

August 2, 2010

Effects of Different IMAP Clients on Mailservers Performance

Teshome Dagne Mulugeta

August 2, 2010

Abstract

Over the past decades, the volume of email exchange has increased dramatically and it has become one of the world's most important means of communication. Due to the rapid increase in email message communication, the service infrastructure has also evolved in general to provide optimum service to customers and end users. Among the many technologies invented as components of email service infrastructure, the Internet Message Access Protocol has played a great role by introducing a better and improved means of electronic message manipulation within mailboxes. To fulfill the IMAP protocol implementation, different email clients have been developed since the birth of IMAP, and there are a large number of open source and proprietary clients available for use, all implemented somewhat to drastically differently especially in their default behavior. This thesis will research whether the differences in their implementation have effects on server side resource usage.

Acknowledgements

First, I thank Aeleen Frisch, for her continuous support and inspiration in the process of this thesis. Words cannot express my gratitude for the encouragement and kindness that she showed me while I passed through difficult moments in the process of this thesis.

I am also greatly indebted to my past instructors at Oslo University College for your inspiration and getting me interested in Network and System Administration.

I also would like to thank Oslo University and Oslo University College for providing me this educational opportunity. I hope the support will continue to many young Ethiopians who are looking for this kind of opportunity.

A special thanks goes to my mentor Stein Vrale for his inspiration, support and providing me an opportunity in System Administration working environment.

Thanks to Solomon Ayanaw for the kind understanding and accompanying during the loneliness time at Oslo University College.

I am also grateful to Bengt Olsen who has shown me the real Norwegian friendship and hospitality. I have seen Norwegians through him.

I especially acknowledge and thank my love, my life and my wife Marta Tesfaye who has patiently tolerated my over-enthusiasm on the subject. Without her support I could not be successful. Last, but not least, a special thanks to our son, Dagmawi Teshome who has always inspired me to work hard and study.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Electronic mail is one of the world's most important communication tools [40]. Although it seems other technologies might surpass its popularity, electronic mail is still an easy to use efficient and formal communication service. As email usage increases, the infrastructure used to provide the service must also improve. The old techniques of email usage are changing as new communication devices have been invented. For example, advanced communication technologies like cellular phones, Personal Digital Assistants (PDAs), iPhones and other smart phones and iPods all come with email client software. This contributes to the rapid increase in Internet resource usage related to electronic mail and requires improvement in email service infrastructure to provide optimum service for users.

Desktop email clients are an important component of the email infrastructure as users prefer to manage their messages on personal computers and laptops in addition to the new communication devices mentioned above. A new study from the Radicati Group, Inc. revealed key statistical figures and forecasts in email. According to its "Email Statistics Report, 2009-2013" [46] report, the number of email users will increase from over 1.4 billion in 2009 to 1.9 billion by 2013. The report showed that 74 percent of email accounts will be used by consumers and the rest will be used by corporate users. The same report finds that there were 247 billion messages per day in 2009, and it is estimated that the usage will increase to 507 billion messages per day in 2014.

One can easily see from the preceding statistics how much the email service infrastructure will affect corporate industry in particular and the Internet in general. With an estimated average 75 Kbytes email message size [47] and billions of messages delivered per day, the importance of optimizing the resource usage by the email service infrastructure components like email servers, clients, protocols, and so on should be studied in order to locate potential areas of improvement.

1.1 Motivation

The *IMAP*[1] protocol is an important component of the email service infrastructure and allows users to manipulate emails messages through *email clients*. Some of management functions provided by email clients are creating multiple mailboxes, deleting mail, flagging mail, searching message contents and moving messages between mailboxes [17]. Therefore, the IMAP protocol is designed to manage email message after it has reached at its "final delivery" point.

Thus, it is necessary to study performance related issues of an *IMAP server* as a "final delivery" point because 1) If something goes wrong with the email message before it is read, there is no way for the sender to know the message was not read by the other party; 2) A message can be stored in mailbox for long time, and no one can be sure how long it can remain there; 3) It is difficult to estimate how much disk volume will be needed to store mailboxes; and 4) It is also difficult to estimate the network traffic required for email messages. The IMAP server stores a lot of important information about an organization when it is one of the main media of formal communication between employees and/with the outside world. Thus, scientific evidence about *IMAP clients'* effect on server resource usage would be very useful and important.

1.2 Problem Statement and Objectives

The choice of an email client to access mailboxes on mail servers is often left to end users. In some cases, this choice is influenced by organizational decision makers for several reasons. For example, a lack of required functionality provided by email clients and/or some server-unfriendly behavior of email clients could influence available or acceptable choices. However, server side performance is seldom a consideration for email client choices. This could be due to an assumption that there is no significant difference between email clients' impact on the server even though different client software is implemented differently.

Nevertheless, some organizations have seen critical performance issues in relation to email clients. Problems are typically temporarily solved through allocation of additional resources, so email clients in relation to server performance may not be perceived as a serious issue. However, if optimizing email service is a goal for the email service infrastructure, the individual email client effect on server performance must be evaluated and compared through scientific study.

Accordingly, this thesis will consider the following research questions:

[1] Internet Message Access Protocol. This thesis refers to the current version of IMAP, Version 4 Revision 1 (IMAP4rev1).

1. What are the effects of different IMAP clients on IMAP server performance and resource usage?

2. How do different IMAP clients implement the IMAP protocol?

For the purpose of this thesis, we study the following popular *IMAP clients:* Microsoft Outlook, Mozilla Thunderbird, Opera, Sylpheed and Mulberry. The *Dovecote IMAP server* will be used as the server side application. The main parameters to be measured and compared with respect to server performance will be *disk I/O* and *network bandwidth*. In order to do so, a controlled experimental lab will be set up, appropriate experimental software will be developed and tested, experiments will be designed and carried out, and the results will be analyzed and interpreted.

Chapter 2

Background

2.1 Terms and Protocols

Electronic mail, typically abbreviated as *email* or *e-mail*, is the primary method of sending and receiving digital messages. It is based on a "store-and-forward" technique in which information is sent to and received from an intermediate station. This intermediate node in email infrastructure is called a mail server [29, 16].

A digital message can be communicated using a client/server architecture, as displayed in Figure 2.1. A message is created by a user using an email client program, and the program sends the message to a server. Then the server transports the message to the recipient's mail server where the recipient's mailbox is located. Finally, the recipient reads the message again using an email client. The whole procedure is complicated since it potentially involves several standard protocols, computer machines potentially running different operating systems and a variety of email client programs [22]. This process is discussed below in detail.

We can define a mail server as a particular machine that is responsible for sending and receiving email messages. A mail server functions in the email infrastructure as a Mail Transfer Agent (MTA) (for a general understanding about email infrastructure see [22]). A mail server can receive, deliver, forward and store messages on behalf of end users. What is expected from users is that they will connect to the mail server and submit or retrieve electronic messages through the aid of different client applications and protocols (which will be explained shortly). Besides the various components and programs that the mail server consists of, the presence of other mail servers in the infrastructure is also necessary to fulfill the email service.

A Mail User Agent (MUA), usually called email client application, is a software program that is used to read and compose email messages. These programs

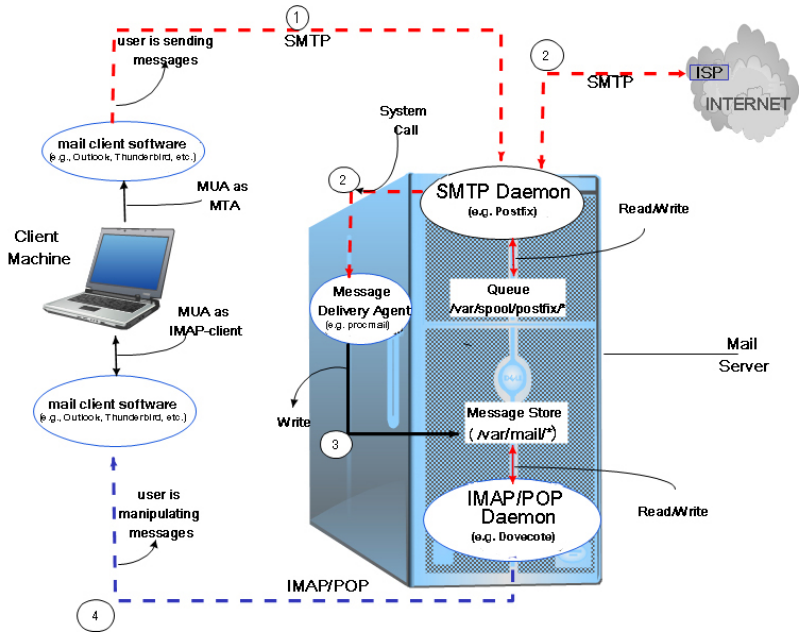

Figure 2.1: Client/Server Mail Architecture

may vary from simple text based email clients like Mutt to modern GUI applications like Thunderbird and Microsoft Outlook. The major purpose of email clients is to access messages in a mailstore located on remote server remotely via access protocols like POP[1] and IMAP. Clients are also able to set up mailboxes to store messages, and manipulate messages in mailboxes, where the precise capabilities depend on the mailstore access protocol that they are configured to use. Email clients can also act like an MTA and send outbound messages directly to a mail server [3].

When we say an email client may act as an MTA to send email, this function must not be confused with the real MTA's role. The main reason that email clients provide this feature is that the machine where a client resides might not have its own MTA in some operating systems. However, it must be clear that these email client programs act as MTA only when sending outbound messages to another MTA. However, they cannot directly deliver messages to mailstores or mailboxes [3].

Software programs like Sendmail and Postfix are used to safely transfer messages between mail servers using the *Simple Mail Transport Protocol (SMTP)*. SMTP is used by mail servers to communicate with other mail servers and to transport messages, but not for receiving them. When a user sends electronic messages via clients such as Microsoft Outlook and Thunderbird, SMTP transports the message until it reaches to its final destination server. Two or more SMTP servers may be required to transport the message, depending on the email service infrastructure.

[1]Post Office Protocol. This thesis refers to the current version of POP, Version 3 (POP3).

Figure 2.1, which represents a simple email service infrastructure, employs only one SMTP server which resides on the same mail server as IMAP/POP daemon. In this case, if both sender and receiver are from same organization and domain, there is no need for sending the messages to another SMTP server. Rather, messages are put in mailboxes of the same mail server. In a more typical case, where the receiver and sender are located in different domains, multiple SMTP servers are involved in routing the message to its final destination [26].

A *Mail Delivery Agent (MDA)* (e.g., *Procmail* and *mail*) is used to filter and move email messages from the MTA's spool file to recipient's mailbox. The MDA plays an important role in the infrastructure by delivering the message into the mailbox to be accessed by the email client program since SMTP can not do this by itself.

On the mail server side, IMAP and POP servers, such as *Cyrus* and *Dovecote*, enable email clients to access their messages. While the IMAP protocol allows general access to mailboxes, the POP protocol typically automatically downloads the full messages to email clients' local systems. These protocols will be discussed in detail below.

2.2 Mailstore Access Protocols

Remote Mailstore Access protocols are important components of an Internet mail infrastructure because they are used to access a mailbox [42]. RFC-1733 defined three types of distributed client/server electronic mail models: offline, online and disconnected [16, 25].

The offline model is implemented by downloading pending messages to the client machine and then deleting them from mail server. The intelligence part of the mail processing task is accomplished locally on client side. This method is called "store-and-forward" since the mail server acts as temporary storage for messages for specific period of time. POP is the main protocol that implements this model(RFC-1225) [16, 25, 17].

In the online model, a client can manipulates messages in a mailbox on mail server without downloading them. This requires a persistent connection to the server. This model also allows one or more clients to manipulate messages remotely at the same time [16, 25, 17].

The disconnected model allows a client to connect to the mail server, cache selected messages, and then disconnect. The client reconnects and resynchronizes with the server whenever it is needed. The major difference from the offline model is that this model leaves the original message on the server for resynchronization purposes whenever the client reconnects to the server. Some people describe this model as a kind of "hybrid" of offline and online

models [16], and others see it as a complement to the online model and as incompatible with offline [25, 17].

2.2.1 Post Office Protocol

POP was defined in 1984 by RFC-918. It is an application-layer Internet standard protocol which is used as a method of delivering email messages to offline clients. The main emphasis was to provide a "simple" method to filter email messages into appropriate user folders so that users could retrieve their mails when they connect to the mail server. Once users are connected to POP server, most email clients by default download the messages permanently to the user's hard disk and delete them from the server. Unlike IMAP, POP was not intended to provide manipulation operations of mailboxes on the server [36]. POP was revised several times, but its developers have remained consistent to the idea of "simplicity" for quick and efficient email retrieval. RFC-1081 was published in 1988 and defines POP3. The POP protocol remains useful for the simple purpose of downloading email from a server, and it is still the preference of many Internet Service Providers [23].

The "download and delete" attribute of POP protocol leads to inconvenience for users when a user tries to access the same mailbox from different client machines, for example from home and office. In this case, a user is obligated to store downloaded messages at different locations. This lack of functionality was resolved through additional settings that provide an option for the user to leave a copy of each message on the server. The *leave mail on server* option was implemented by UIDL[2] command [23].

Currently POP4 is under development, and it is designed to provide some IMAP functionality on server side message manipulation (see below). The additions are new commands like Create, List, Select and Delete Folders, Set and Get Flags on a message, and commands for partial message retrieval and to enable persistent server connections. The More over, Move and Copy commands are also added to move and copy messages from one folder to another [2].

2.2.2 Internet Message Access Protocol

IMAP was first formulated in 1986 by Mark Crispin at Stanford University [19, 34]. Following the invention of the protocol, the first IMAP RFC, RFC 1064, was published by Cripsin in 1988. At that time IMAP was called the Interactive Mail Access Protocol, and the first RFC focused on *C-client* allowing workstations or similar machines to access electronic mail from a mailbox server [15, 25, 17].

[2]Unique IDentification Listing

The first real IMAP client, called *MM-D*,[3] was written for *Xerox Lisp* machines at SUMEX-AIM[4]. It was based on the slightly earlier C-client software written for the Macintosh client foundation. Later on, when Mark Cripsin moved to University of Washington, he continued the improvement of the protocol, and his C-client software were merged with their PINE email client. The blending of the two clients was useful to the PINE email client because it adapted the most important functionalities of the C-client like MIME[5] parsing, decoding and SMTP. In 1990, University of Washington deployed an IMAP server and released PINE version 2.0 with IMAP support [15, 25, 17].

After the release of a series of RFCs (1730-33) for IMAP4 in 1994, the protocol was approved as an Internet standard. RFC 1730 described the major protocols, and it was followed by RFC 1731 for authentication mechanisms. Substantial development of the protocol continued when Carnegie Mellon University released another IMAP4 server in 1995. During the same year, University of Washington released an improved C-client. In 1996, IMAP, the current version name as of the writing of this thesis, was released via RFC 2060 [25, 17].

Thereafter, IMAP development accelerated, and most important RFCs for IMAP were developed since 1996. So far, as of the completion of this thesis, 53 RFCs have been published on IMAP. The latest major IMAP RFC is RFC 3501 written by Mark Cripsin and titled "INTERNET MESSAGE ACCESS PROTOCOL – VERSION 4rev1"; most people call it simply "IMAP4." RFC 3501 has been subsequently updated by several RFCs. A comprehensive list of RFCs in relation to IMAP and their status is included in Appendix A.

The following chart is derived from the Appendix A table to point out which individuals and organizations have contributed most to the IMAP protocol development. As the chart indicates, M. Crispin and A. Melnikov are the most active individual contributers to IMAP RFC publications. Since these individuals are currently with Isode Ltd. and University of Washington (respectively), the organizational contribution chart reflects this [39]. Most importantly, it is worth mentioning that the major IMAP protocol technology was contributed by M. Crispin while at the University of Washington.

2.2.3 Comparing the IMAP and POP Protocols

In general, POP's usefulness greatly relies on its simplicity. Historically, it has left mailbox access capabilities to IMAP [23]. In addition to its online message access service, remote manipulation of mailbox functionality distinguishes IMAP from the POP protocol. Nonetheless, despite some lack of functionality, many mail servers are still using the POP mail protocol because of its simplicity and suitability for users who access their mailbox solely from a

[3] MM-Distributed
[4] Stanford University Medical Experimental Computer for Artificial Intelligence in Medicine
[5] Multipurpose Internet Mail Extensions

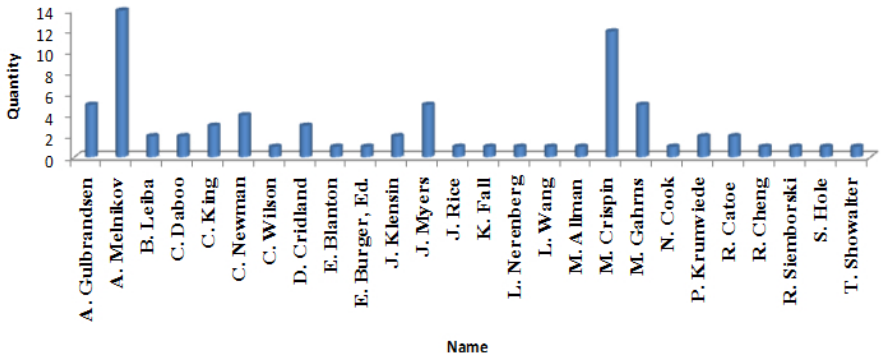

Figure 2.2: Individual RFC Contributions to IMAP

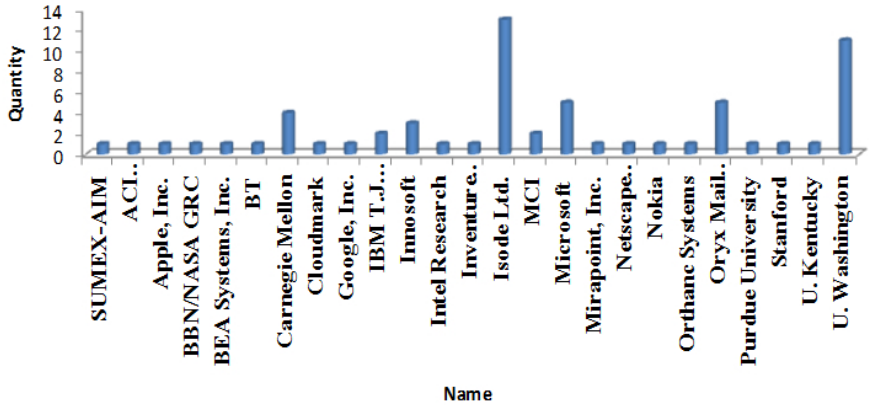

Figure 2.3: Organizational RFC Contributions to IMAP

single machine. Moreover, the offline message access method minimizes disk requirement and server connection time [25, 17].

IMAP has specific advantages over POP in remote folder manipulation, multiple folder support, and online performance optimization. Each of these are implemented by different commands with IMAP protocol (they are discussed in another section of this thesis).

The IMAP protocol allows multiple clients to connect to the same mailbox simultaneously, whereas POP allows only the current connected client to access the mailbox.

IMAP also allows clients to keep track of message state, such as whether a message has been read or replied to, while POP does not implement such functionality. The presence of this functionality allows different clients that access the mailbox to easily detect the state change made by another client and facilitates synchronization.

IMAP allows clients to implement searching for messages meeting specific criteria on the server side, without downloading all messages on client machine.

POP clients do not require MIME formated messages whereas IMAP is designed to work with MIME formated email messages. However, since all Internet email messages come with MIME formatting, POP clients must understand this type of electronic messages. The main difference in this area is that IMAP allows clients to access and fetch only a part of MIME messages. For example, this allows clients to download only the text part of the messages without the attached files [21].

IMAP's complexity in implementation both in the server and client sides is its disadvantage as compared to POP. Although the added complexity has been solved by "server-side workarounds" like backed database and maildir, it still requires more resources, including network bandwidth, disk I/O and disk space usage, as compared to POP, due to the fact that it implements the online mode of client/server communication. These inherently smaller resource requirements could be one reason that POP is popular with many Internet Service Providers. Moreover, if the implementation of searching and other mailstore functionality with an IMAP-enabled client is not accomplished carefully and efficiently, unnecessary server side resource utilization could increase.

Another disadvantage of IMAP is that clients need to stay connected to the server to be notified of the arrival of new messages. Although there is a solution for this called "push IMAP," which sends the whole message instead of a notification, the method has not been accepted by IETF. A Lemonade profile, which is a product of IETF, avoids this problem via the "forward without download" technology. However, this solution relies on IMAP capability and support for the IDLE command. This command will be discussed in detail later. [32]

To minimize the offline access model disadvantages of POP, an online model is integrated into POP, allowing the POP server to be configured to "leave mail on server" rather than deleting messages permanently. However, it is difficult to say whether it provides true online model functionality, and some call it is "pseudo online" because it does not implement a remote file system for online operations. For example, as it has been explained earlier, the state information for each message (e.g.., marking a message as replied or not) is not stored with the message itself in POP. [25, 17].

When a POP client retrieves new messages, it must fetch the entire UIDL map. In contrast, IMAP allows a client to fetch only messages that have a UID higher than all previously retrieved ones. This can lead to a significant difference between the two protocols for large mailboxes because the POP approach requires significant processing time and other resources.

Despite the disadvantages mentioned above, POP is still convenient for users who use only one client system and lack a persistent Internet connection to

their mailbox [3]. While many email clients support both the POP and IMAP protocols, not all Internet Service Providers (ISP) support IMAP for several reasons. First, some ISPs serve as storing messages "in transit" for specific period of time and they do not provide "final delivery" service to manipulate electronic messages. Secondly, since IMAP allows every customer to use potentially increasing storage, ISPs do not support it due to the volume issue. Thus, POP suits the ISP's objectives [42].

Although the two protocols were invented for different purposes, they have developed common characteristics that make them quite similar today. For example, both rely on SMTP for sending messages and a continuously available mail server to access mailboxes. They provide client applications mail access from anywhere in a network. Both support the offline access model and include built-in extension mechanisms to extend the base protocol [17] [23].

2.3 IMAP clients

There are two types of IMAP clients, caching and non-caching. Caching IMAP clients (for example Thunderbird and Outlook) fetch new messages once and depend on the message's flags (meta-data) to synchronize messages. In contrast, non-caching IMAP clients, also known as web mails, fetch the same message again and again. This difference between IMAP clients could easily affect the performance of the IMAP server [3].

There are many web-based email services and desktop email clients developed to implement this protocol. However, few are popular and frequently used. Statistical reports from an email clients usage survey showed that [6, 4] Microsoft Outlook (including Outlook Express) is the most popular desktop email software followed by web-based email services provided from Hotmail and Yahoo! Mail, while Thunderbird and Windows Live have only about 2 percent of the total market share each.

The following section will review some of the most popular email clients.

2.3.1 Web mail or Web-based email clients

As the name itself implies, web mail is a fully web-based email service. That means it is accessible via a web browser and does not need standalone desktop software. Using web mail, users can access their mailbox from anywhere and any machine as long as there is an Internet connection. This is the major benefit of web mail clients. Security, backup and software management issues are taken out of users' hands, and users are thereby free from many irritations. The drawback is that if there is no Internet connection, then a user is not able

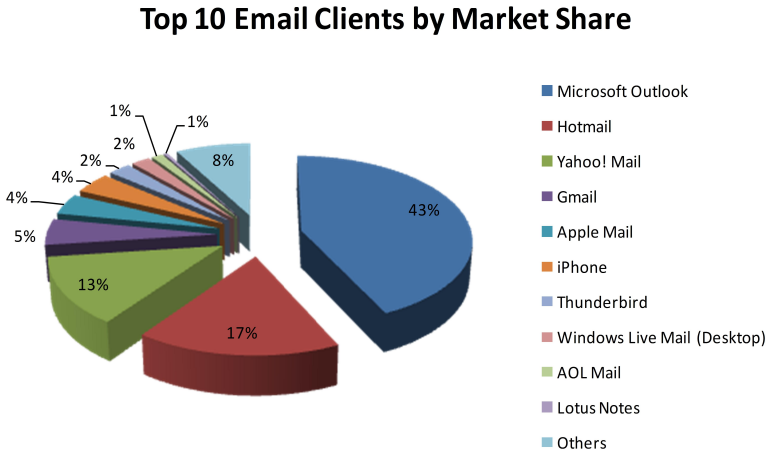

Figure 2.4: Email Clients Usage Distribution [source: litmusapp.com, February 2010]

to do anything offline. Moreover, mail box access usually costs money, and the price increases with mailbox size [10].

The first popular web-based mail clients, Hotmail and Yahoo!, were introduced in 1994 and 1997 (respectively), followed by Gmail in 2004. Web mail clients are widely used [9]. According to sa tatistical survey, Yahoo!, Hotmail (now called Windows Live Hotmail) and Gmail are the top ranked web mail service providers [4] [10]. Currently the AJAX-powered[6] technology helps web mail clients mimic the desktop clients' look and feel [7].

2.3.2 Desktop clients

Microsoft Outlook 2007

Outlook 2007 is part of the Microsoft Office 2007 suite. In addition to email, it provides many personal information management features, including an address book, a calender, reminders, fax, instant messaging, task lists, journals, personal notes and news feeds. These features make Outlook 2007 more than an email client. Outlook 2007 supports most protocols and standards related to e-mail, including both POP and IMAP. It does not support the PLAIN authentication method. Outlook's main window is illustrated in Figure 2.5.

[6] Asynchronous Java Script and XML: a group of interrelated development techniques used in client side applications

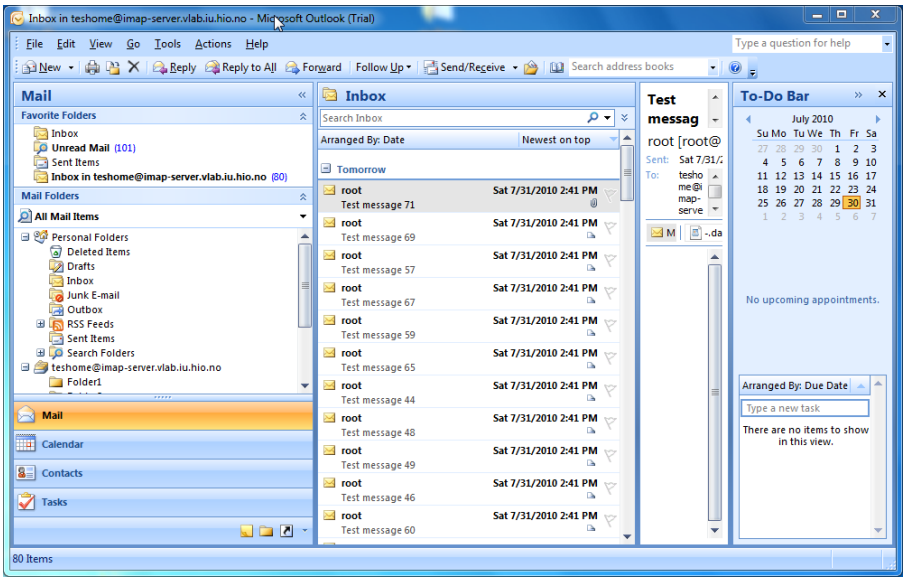

Figure 2.5: Outlook 2007 Front View

Mozilla Thunderbird Version 3.0.5

Mozilla Thunderbird is a free, Open Source email client from the Mozilla Foundation, and it is available for the Unix, Linux and Windows environments. The developers claim that the email client has superlative security and provides sophisticated customization capabilities. It includes SSL/TLS support for communication with IMAP and SMTP servers and S/MIME. Among the many features that the client provides are search functions, multiple account support, message grouping, and extensive filtering and labeling options. Figure 2.6 shows Thunderbird's front view.

Thunderbird allows programmers to create add-on programs (called extensions) which incorporate new features like spam suppression, removing duplicate messages, enhanced address books, and the like. Thunderbird can also import email from other email clients. It incorporates a built in RSS reader, for example, to notify users about updates to monitored web sites. Since it is an Open Source application, new features are integrated frequently into Thunderbird as new extensions are released [43].

Mulberry Version 4.0.8

The Mulberry email client was originally a proprietary software package and started as a pure IMAP client. It was developed in 1995-96 by Cyrus Daboon. Its owning company went bankrupt in 2005, and Mulberry has been Open Source software since 2007. The developers state that the client implementation adheres to the IETF standards[1]. Mulberry runs on both Linux and

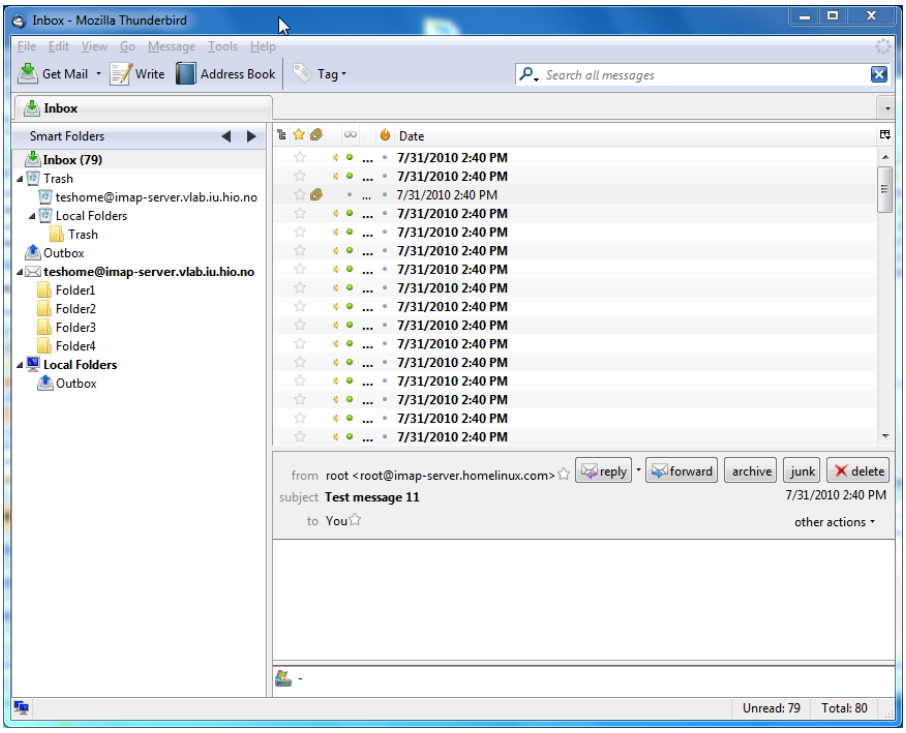

Figure 2.6: Thunderbird Front View as New Message Arrive

Windows systems. Figure 2.7 shows Mulberry's front view.

Sylpheed Version 3.0.0

Sylpheed is a Windows email client. Its developers describe their email client as providing quick response, having a graceful yet sophisticated interface, intuitive to use and easy to configure, and including abundant features. Sylpheed is also news reader based on GTK+. The client's front fiew is shown in Figure 2.8.

Opera Version 10.54

Opera mail an integrated email client with Opera browser. Its developers describe it as designed for low bandwidth mode that helps for users with slow Internet connection and it searchs with speed. Moreover, it is equiped with smart spam filter, auto-sort, and attachments filter. The developers also describes Opera Mail that it can organize, indexe, and sort messages. Opera mail is young and the first in its kind to be integrated with web browser. Figure 2.9.

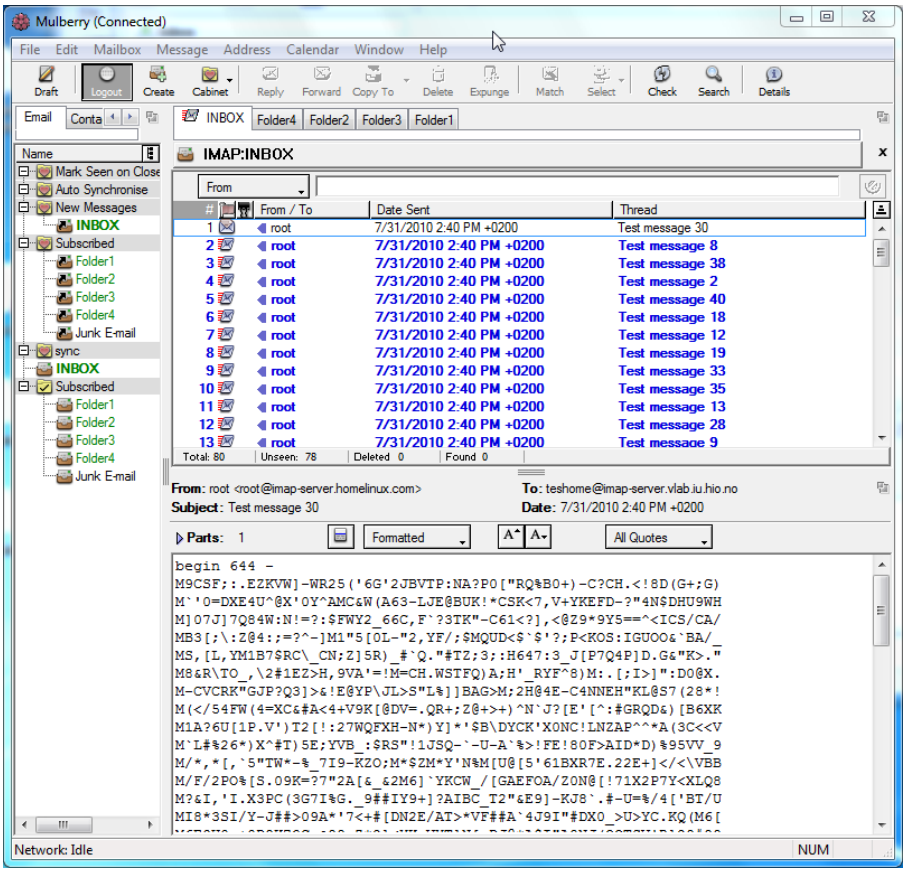

Figure 2.7: Mulberry Front View

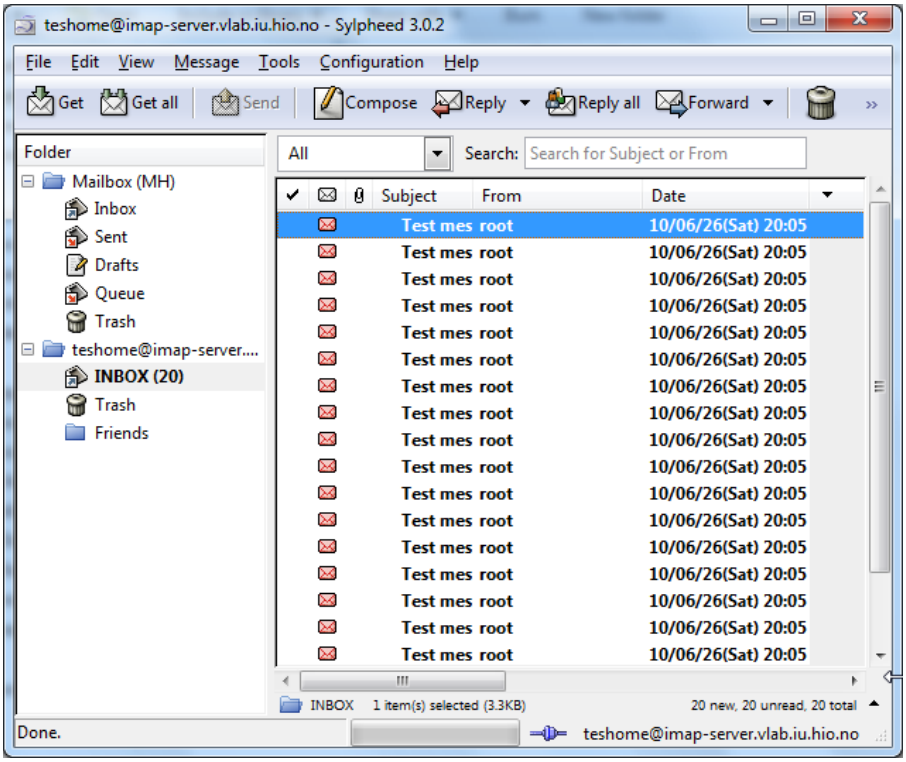

Figure 2.8: Sylpheed Front View

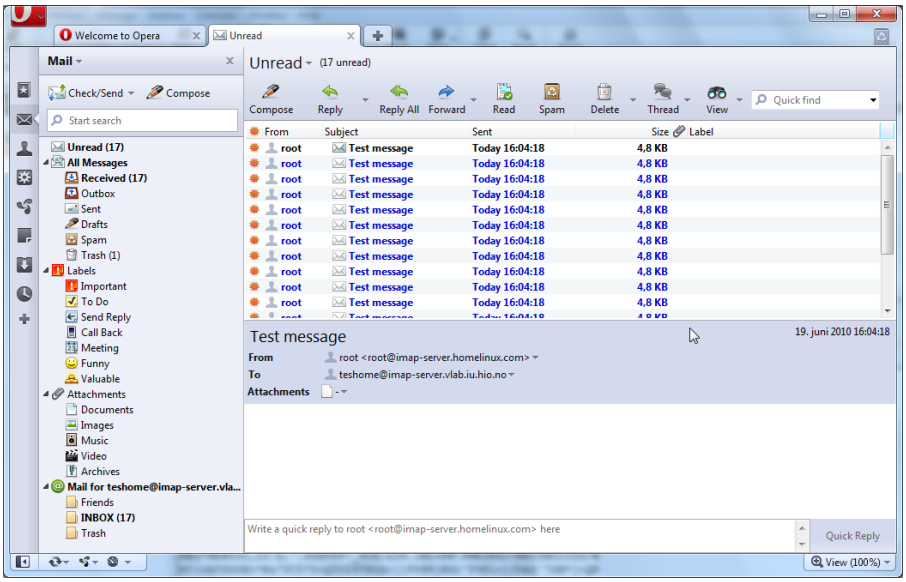

Figure 2.9: Opera Mail Front View

2.4 IMAP Client Implementation

This discussion summarizes the IMAP client capabilities and commands.

2.4.1 Message Attributes

When a message is stored in IMAP mailbox, a *sequence number* and a *UID* are assigned to it. These numbers are used to access and identify the message on a remote IMAP server for various manipulation purposes. The UID value *MUST NOT*[7] refer to any other messages, either in the same mailbox or any subsequent mailbox. Thus, when a message is stored in a mailbox, the next largest UID number will be assigned to it. The protocol strictly forbids changing of UID values during a session and between sessions. Should such a change ever happen, it MUST be detected using the *UIDVALIDITY* attribute. Whenever there is a change in the UIDVALIDITY message attribute, a client is required to remove any cached information about the messages, and the UID assignment starts again. During the lifetime of a message, UIDVALIDITY and UID refer to a single message on that server. This provides the immutability behavior [8] of the message during its existence. [17]

In contrast, the sequence number can be changed dynamically when a message in the mailbox is deleted, which makes a particular message possess multiple sequence numbers. A sequence number starts with one when the first message arrives in a mailbox [17].

A *flag attribute* is a list of zero or more named tokens associated with the message, and each is set and cleared for its addition or removal to the list. There are two kinds of flags in flag attributes: permanent and session-only. While a permanent flag allows client to add or remove from the message flags permanently, session flags changes are effective only in that session [17].

The *Internal Date Message* attribute records the internal date and time of the message on the server, which is different from the date and time which is found on messages when messages are received. The Internal Date Message reflects the timestamps when a message is delivered to the IMAP server [17].

The *Envelope and Body Structure Message* attributes represents RFC-2822 header of the message and *MIME* body structure information of a message, respectively. The Envelope structure is different from *SMTP* envelope [17].

[7]This IMAP protocol specification phrase means that the action described will be almost certain to hurt interoperability. The recommendation should not be ignored.

[8]Meaning that the actual content of message and header cannot ever be changed

2.4.2 IMAP Security

Since IMAP is a client/server protocol, an email client which resides on remote machine runs a process on IMAP server to access a mailbox. Accordingly, IMAP requires the email client to authenticate before it starts to access the mailbox.

IMAP security is conceptually divided in to two categories: authentication and encryption. RFC-1731 and RFC-2595 define IMAP authentication mechanisms and encryption, respectively. Like POP, IMAP allows basic authentication mechanism through userID and clear text passwords over the network. However, this is often undesirable due to security risks.

By using the cryptography-based challenge/response SASL[9] mechanism, the clear text authentication problem was solved [35] [37] (although this method does not encrypt the message content).

The SSL/TLS[10] implementation is the next higher level security solution for IMAP client/server connection. Currently, many IMAP servers can be configured to provide SSL/TLS connections to securely encrypt both authentication and communication between server and client. Most IMAP clients also support this method of connection, and the security risk of IMAP protocol is thereby minimized [35] [37].

STARTTLS, as defined in RFC-2595 for IMAP and POP, solves a number of problems. The major one is that it avoids requiring separate IMAP and POP ports for use with SSL [37].

2.4.3 States of IMAP Connections

An IMAP connection can be in one of the following four well-defined machine states at a given time: Not Authenticated, Authenticated, Selected and Logout. The four states are explained in Figure 2.10 [17].

[9] Simple Authentication and Security Layer
[10] Secure Sockets Layer/Transport Layer Security

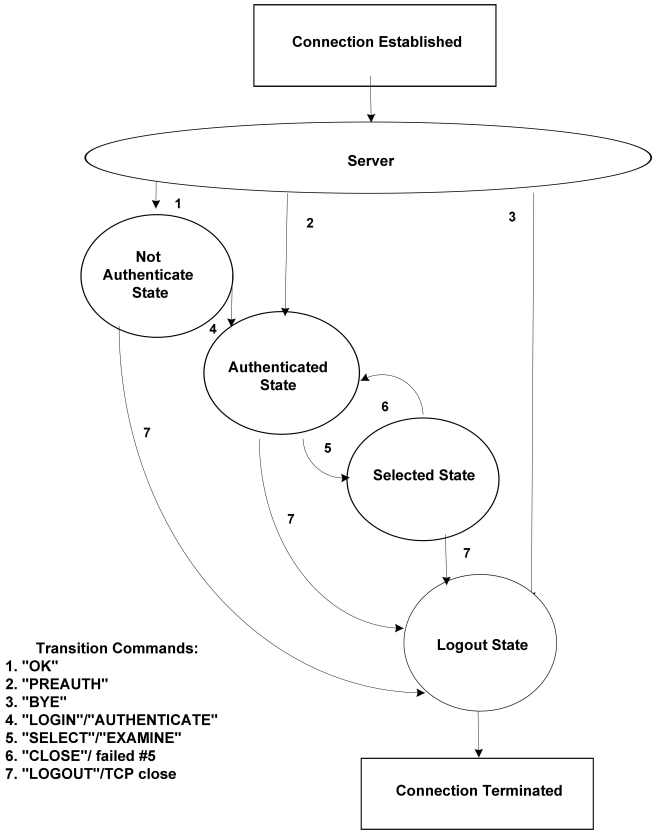

Figure 2.10: IMAP States and State Transitions

The Internet Assigned Numbers Authority (IANA) has assigned TCP port 143 for IMAP use, and the "Connection Established" state in Figure 2.10 indicates the completion of the TCP connection. The Not Authenticated state, the first state after a machine starts, is entered immediately after a connection is successfully established between a client and the server (unless the connection previously has been authenticated). Before the client can send any commands to the server, it must send authentication credentials. This, the Not Authenticated state does not allow clients to issue most IMAP commands. In this state, the server responds with the "OK" command to notify the client that it is ready, and this is the only command option that is available within this state. [17].

The second state, the Authenticated state, is entered when either i) a pre-authenticated connection is started (as indicated in Figure 2.10 by arrow 2); ii) valid authentication credentials are provided by the client (indicated in Figure 2.10 by arrow 4); iii) after an error occurred during mailbox selection attempt or after a successful CLOSE command is issued (indicated in Figure 2.10 by arrow 6) [52].

When a client is authenticated, it MUST select a mailbox to access messages. When a mailbox is successfully selected for message access, then the Selected state is entered [52]. This state can be entered from only one direction, indicated by arrow number 5 in Figure 2.10.

A connection can be terminated in a logout state. A client request to logout or a unilateral action from the client or server can cause an IMAP connection to be in the Logout state [17]. The Logout state can also be entered when a connection is interrupted [52]. Figure 2.10 shows that this state can be entered from any of the four states.

2.4.4 Commands and Responses

A server/client interaction in IMAP connection consist of a client *command*, server data, and a server completion result *response*. A client issues the commands that are appropriate for a specific state (discussed below). This means that most commands for specific state cannot operate in another state.

The server/client interaction is line oriented, using strings that end with CR/LF[11]. Therefore, during server/client interaction, the client and server are either reading in a line or a sequence of octets.

The client starts the interaction by sending a command or commands. Each command generated by client identified by a *tag* followed by the actual command identifier. The client is responsible for generating the tag for each command. When a client sends a command, an IMAP server reads the command

[11]Carriage Return/Linefeed

line, parses the command and its arguments, and replies back by transmitting the server data and finally a response to indicate the command completion result [17]. Figure 2.11 illustrates this process when a client is logging to an IMAP server. "001" is the tag that the client issued, and the server replied back referring to that tag.

```
* OK Dovecot ready.
001 LOGIN mailclient password
001 OK Logged in.
```

Figure 2.11: IMAP Use of Tags

When data is transmitted by an IMAP server to client, one of the following procedures will happen. If a command is not completed, an untagged status response prefixed with the "*" token will be sent to client. For example, as Figure 2.12 indicates, the server uses this token until has issued the final completion response. If a command is completed, a server completion result response is given which indicates a success or failure of the operation by one of the following four possible server completion responses: 1) OK to indicate success (Figure 2.12); 2) NO to indicate failure (Figure 2.13); 3) BAD to indicate a protocol error (Figure 2.14); or 4) BYE to indicate a server is going to close the connection (as shown in Figure 2.15). The last BAD response can happen for several reasons, but an unrecognized command or a command syntax error are the main ones. For example, Figure 2.14 showed the BAD response due to the unrecognized command "READ."

```
002 SELECT Inbox
* FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
* OK [PERMANENTFLAGS (\Answered \Flagged \Deleted \Seen \Draft \*)] Flags permitted.
* 0 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1268643367] UIDs valid
* OK [UIDNEXT 1] Predicted next UID
002 OK [READ-WRITE] Select completed.
```

Figure 2.12: Server Completion Response with "OK" and Use of "*"

```
002 SELECT Outbox
002 NO Mailbox doesn't exist: Outbox
```

Figure 2.13: Server Completion Response with "NO"

The IMAP client reads a server response line and takes action, and it then shows human readable information to the end user. RFC3501 clearly states

```
003 READ Inbox
003 BAD Error in IMAP command READ: Unknown command.
```

Figure 2.14: Server Completion Response with "BAD"

```
004 LOGOUT
* BYE Logging out
004 OK Logout completed.
```

Figure 2.15: Server Completion Response with "BYE"

that the IMAP client MUST be prepared at all times to accept any response from server side, even including any server data that was not requested. The protocol also enforces that a client SHOULD record any received data so that it can be referenced when the data is needed. In this case, the client will not need to send any command for same data.

An IMAP client may issue multiple commands to an IMAP server simultaneously. The command tags are used to determine which command a server response applies to. The following section explains the different states in IMAP protocol, followed by the most common commands and server responses.

IMAP Commands Valid in Multiple States

The CAPABILITY command returns a list of capabilities supported by the IMAP server. This command is available in both the authorization and transaction phases of the IMAP protocol. This command is not dependent on user or connection state in a session. Thus, it is enough for a client to send a CAPABILITY command once in a session. Some IMAP servers advertise CAPABILITY during OK greeting. Currently, IMAP supports IMAP4, IMAP, LOGIN-REFERRALS and ACL. Figure 2.16 is a good example.

```
* OK Dovecot ready.
005 CAPABILITY
* CAPABILITY IMAP4rev1 SASL-IR SORT THREAD=REFERENCES MULTIAPPEND
UNSELECT LITERAL+ IDLE CHILDREN NAMESPACE LOGIN-REFERRALS
AUTH=PLAIN
005 OK Capability completed.
```

Figure 2.16: The CAPABILITY Command

The NOOP command (NOOP means "no operation") does nothing, but it is

useful to maintain a connection by preventing inactivity timeouts on server side.

When a client needs to finish the connection with the server, the LOGOUT command is sent to the server. The server responds to the command with untagged BYE line followed by a tagged result line. This server/client communication is shown in Figure 2.15.

IMAP Commands Valid in the Nonauthenticated State

Since an IMAP session starts with the Nonauthenticated state, the LOGIN command is initiated by client to enter the authenticated state since it can not send most commands until a fully authenticated state is satisfied. The LOGIN command works with username and password. The server then responds with a single tagged result line. This procedure is shown in Figure 2.11.

IMAP Commands Valid in the Authenticated State

As the name by itself implies, the LIST command lists all the mailboxes on the server that match the requirement specified in its arguments: the "reference name" and the "mailbox name." The asterisk (*) and percent sign (%) characters can be used as wldcards to list all mailboxes. Figure 2.17 illustrates this process. As we can see from this output, mailboxes are arranged like a tree, with INBOX serving as its root. This particular IMAP protocol uses a period (.) as a separator between parent and child folders so INBOX.Friends is a child of the INBOX mailbox. The HasChildren attribute simply indicates that this folder has subfolders whereas the other folders do not and are identified by HasNoChildren. This arrangement can vary according to IMAP configuration on the server. It is possible to configure all folders to be created as subfolders of the INBOX even if an email client is configured not to display them the same way.

```
005 LIST "" "*"
* LIST (\HasChildren) "." "INBOX"
* LIST (\HasNoChildren) "." "Junk E-mail"
* LIST (\HasNoChildren) "." "Trash"
* LIST (\HasNoChildren) "." "Sent"
* LIST (\HasNoChildren) "." "INBOX.Friends"
* LIST (\HasNoChildren) "." "INBOX.Family"
* LIST (\HasNoChildren) "." "INBOX.Work"
005 OK List completed.
```

Figure 2.17: The IMAP LIST Command

The STATUS command returns some basic information without selecting the folder. Depending on what information is going to be extracted, the command

takes arguments such as the mailbox name and a status code. More than one status codes can be specified in a single request. Figure 2.18 illustrates a client requesting the total number of messages and the number of recent messages separately.

```
006 STATUS Inbox (messages recent)
* STATUS "Inbox" (MESSAGES 1 RECENT 0)
006 OK Status completed.
```

Figure 2.18: The IMAP STATUS Command

The SELECT and EXAMINE commands basically function similarly by returning the information about the specified mailbox. The basic difference between them is that EXAMINE returns a read-only reference whereas SELECT returns a read-write reference. Both commands take a mailbox name as an argument. These commands MUST be issued before a client can access any messages from their mailbox. The Selected state is named after this command since it is entered after successful selection of a mailbox. The server responds with the status information for the mailbox selected. Among the information returned by server are the FLAGS that are valid for the mailbox, a list of FLAGS that the client is privileged to change, the number of messages in the mailbox, the number of RESENT and UNSEEN messages.

The CREATE, DELETE and RENAME commands are simple and create, delete and rename a mailbox, respectively. They take the name of a mailbox as an argument and request the server to carry out the requested command. The server responds with a flagged result line.

```
009 CREATE Inbox.School
009 OK Create completed.
010 LIST "" "*"
* LIST (\HasChildren) "." "INBOX"
* LIST (\HasNoChildren) "." "Junk E-mail"
* LIST (\HasNoChildren) "." "Trash"
* LIST (\HasNoChildren) "." "Sent"
* LIST (\HasNoChildren) "." "INBOX.School"
* LIST (\HasNoChildren) "." "INBOX.Friends"
* LIST (\HasNoChildren) "." "INBOX.Family"
* LIST (\HasNoChildren) "." "INBOX.Work"
010 OK List completed.
```

Figure 2.19: IMAP CREATE Command Usage

The IMAP protocol allows clients to delete messages. When a user deletes a message, the message is not deleted immediately, but rather marks it with the \Delete flag.

```
011 RENAME Inbox.School Inbox.College
011 OK Rename completed.
012 LIST "" "*"
* LIST (\HasChildren) "." "INBOX"
* LIST (\HasNoChildren) "." "Junk E-mail"
* LIST (\HasNoChildren) "." "Trash"
* LIST (\HasNoChildren) "." "Sent"
* LIST (\HasNoChildren) "." "INBOX.College"
* LIST (\HasNoChildren) "." "INBOX.Friends"
* LIST (\HasNoChildren) "." "INBOX.Family"
* LIST (\HasNoChildren) "." "INBOX.Work"
012 OK List completed.
```

Figure 2.20: IMAP RENAME Command Usage

The EXPUNGE command deletes all messages which are marked with the \Delete flag in the mailbox. This function is implemented by various IMAP clients differently, and many of them provide options for users to decide the fate of a deleted message. For example, the Thunderbird IMAP client provides the following options after a message is deleted: 1) Move the deleted message to a folder like Trash, 2) Just mark it as deleted, and 3) Remove it immediately. It also allows users to issue an EXPUNGE command to clean up the Inbox and/or to Empty Trash on exit. Each option mentioned above is managed by the client software differently to manipulate the mailbox on the server. For example, Figure 2.22 shows the usage of EXPUNGE command assuming four messages were set with the Deleted flag in the INBOX. In this example, the EXPUNGE command deleted all four messages one by one.

```
013 DELETE Inbox.College
013 OK Delete completed.
014 LIST "" "*"
* LIST (\HasChildren) "." "INBOX"
* LIST (\HasNoChildren) "." "Junk E-mail"
* LIST (\HasNoChildren) "." "Trash"
* LIST (\HasNoChildren) "." "Sent"
* LIST (\HasNoChildren) "." "INBOX.Friends"
* LIST (\HasNoChildren) "." "INBOX.Family"
* LIST (\HasNoChildren) "." "INBOX.Work"
014 OK List completed.
```

Figure 2.21: IMAP DELETE Command Usage

The CLOSE command has same effect as EXPUNGE command. The difference is that after CLOSE command deletes the messages, it deselects the currently selected folder. If a CLOSE command is issued, the client can not perform any

```
014 EXPUNGE
* 4 EXPUNGE
* 3 EXPUNGE
* 2 EXPUNGE
* 1 EXPUNGE
014 OK Expunge completed.
```

Figure 2.22: IMAP EXPUNGE Command Usage

action on any messages until the deselected folder or another folder is selected.

The LSUB command functions like the LIST command except that it shows only mailboxes marked as an active with a SUBSCRIBE command, as illustrated in Figure 2.23.

```
015 Create Inbox.School
015 OK Create completed.
016 LSUB "" "*"
* LSUB () "." "Trash"
* LSUB () "." "Sent"
* LSUB () "." "Junk E-mail"
* LSUB () "." "INBOX.Friends"
* LSUB () "." "INBOX.Family"
* LSUB () "." "INBOX.Work"
016 OK Lsub completed.
```

Figure 2.23: IMAP LSUB Command Usage

The SUBSCRIBE command permits a client to add a mailbox to the list of subscribed mailboxes, taking the desired mailbox as an argument. The subscribed mailboxes can be viewed by the LSUB command or LIST command. This process is illustrated in Figure 2.24.

The UNSUBSCRIBE command is used to remove a mailbox from the list of subscribed mailboxes. It also takes the mailbox name as an argument, as shown in Figure 2.25.

The APPEND command is a multi-line command that appends text as a new message within a mailbox. It accepts a list of flags that need to be set on the new message. Date and time strings also included before the text. The bracketed number at the end of the command indicates the length of the message so that the server knows when the client is finished.

```
017 SUBSCRIBE Inbox.School
017 OK Subscribe completed.
016 LSUB "" "*"
* LSUB () "." "Trash"
* LSUB () "." "Sent"
* LSUB () "." "Junk E-mail"
* LSUB () "." "INBOX.Friends"
* LSUB () "." "INBOX.Family"
* LSUB () "." "INBOX.Work"
* LSUB () "." "INBOX.School"
016 OK Lsub completed.
```

Figure 2.24: IMAP SUBSCRIBE Command Usage

```
018 UNSUBSCRIBE Inbox.School
018 OK Unsubscribe completed.
016 LSUB "" "*"
* LSUB () "." "Trash"
* LSUB () "." "Sent"
* LSUB () "." "Junk E-mail"
* LSUB () "." "INBOX.Friends"
* LSUB () "." "INBOX.Family"
* LSUB () "." "INBOX.Work"
016 OK Lsub completed.
```

Figure 2.25: IMAP UNSUBSCRIBE Command Usage

IMAP Valid Commands in the Selected State

The FETCH command is used to access messages in mailbox and it is the most important command to retrieve messages from mailboxes. It has several options depending on what the user wants. Some of them are message flags, email headers, and text of the body.

Using the FETCH command a user can select only one or more messages by using the message sequence number in a range. It is possible to select all messages usine the "*". Figures 2.27, 2.28 and 2.29 illustrate this procedure.

The STORE command adds, replaces or removes IMAP flags on messages, as illustrated in Figure 2.30.

The COPY command copies any number of messages. This command is useful for moving messages from one folder to another since IMAP does not have built-in move command. Figure 2.31 illustrates this procedure.

```
018 APPEND INBOX (\Seen) {262}
Date: Mon, 20 Jul 2010 12:00:01 +2000
From: teshome@imap-client.vlab.iu.hio.no
Subject: This is a test message for APPEND command
To: teshome@imap-server.vlab.iu.hio.no
Message-ID: <20100731124035.34BC643A2DC@imap-server.vlab.iu.hio.no>

Hello, World

018 OK Append completed
```

Figure 2.26: IMAP APPEND command Usage

```
019 FETCH 1 flags
* 1 FETCH (FLAGS ())
019 OK Fetch completed.
```

Figure 2.27: A FETCH Command Selecting One Message

The UID command is issued with COPY, FETCH, STORE, or SEARCH commands, together with unique identifiers in place of message sequence numbers.

The IDLE command allows a client to constantly monitor a mailbox so that a user will be notified when a new message arrives. The server responds and waits until a new message arrives or the client breaks the connector with the DONE command (usually in order to send another command). During this waiting period, the server sends "* OK Still here" message to inform the client that the server is connected. Figure 2.32 illustrates this procedure. The DONE command is the only command without a preceding tag.

2.4.5 Mailbox Synchronization

IMAP clients save some data on client computers in a cache. When a server replies to EXPUNGE command that a message has been removed, the client maps these messages' UIDs to its cache. When a mailbox is selected by client, an IMAP server is required to send status replies. The most important fields for mailbox synchronization are changes to UIDVALIDITY, EXISTS and UIDNEXT. When a client receives these fields, the client performs any action required to bring the mailbox in to a synchronized state by modifying with the Sequence number, the UID number and message flags attributes [17].

Whenever the UIDVALIDITY value is changed on server side, the client cleans its cache by removing any data in relation to its mailbox messages. This could happen when UIDs changed or a message part is modified on the server side.

```
020 FETCH 1:3 flags
* 1 FETCH (FLAGS ())
* 2 FETCH (FLAGS ())
* 3 FETCH (FLAGS ())
020 OK Fetch completed.
```

Figure 2.28: A FETCH Command Selecting Three Messages

```
021 FETCH 1 fast
* 1 FETCH (FLAGS () INTERNALDATE "16-Mar-2010 22:17:11 +0100" RFC822.SIZE 2784)
021 OK Fetch completed.
```

Figure 2.29: Fetch Command with Fast Option

If there is no UIDVALIDITY value or cache information changed, then the client can fetch a complete UID mapping and message flags.

Any change in UIDNEXT value is checked by a change in UID value and EXISTS. If the UID value change is the same as the EXISTS value change when the UIDNEXT is changed, then it means some new messages have arrived but none were deleted. In this situation, the client only has to request the UID values of the new messages and FLAGS for all messages in the mailbox. However, if the changes in UID and EXISTS differ, then the client must retrieve the UIDs of all messages in the mailbox.

Finally, if the UIDNEXT value has not changed, then it means that no new messages have arrived in the mailbox since the last time synchronization. In this situation, if the EXISTS value is changed, then this indicates that some messages are permanently removed, and clients are obliged to fetch their UID to Sequence value again as a whole. However, if EXISTS and UIDNEXT values have not changed, this indicates that there have been no change in the mailbox and only the message flags must be resynchronized.

After accomplishing these checks and communicating with the server without error, then a client is said to be in the fully synchronized state and it is ready to make any mailbox updates.

2.4.6 Differences Among IMAP Clients

Although the purpose of different IMAP clients is to access and manipulate mailboxes on IMAP server in such a way that it provides equivalent functions

```
027 STORE 1:2 flags \Deleted
* 1 FETCH (FLAGS (\Deleted))
* 2 FETCH (FLAGS (\Deleted))
027 OK Store completed.
```

Figure 2.30: IMAP STORE Command Usage

```
028 COPY 3:4 Inbox.Friends
028 OK Copy completed.
```

Figure 2.31: IMAP COPY Command Usage

to local folders, they all implement IMAP differently. This is due in part to the fact that the IMAP protocol recommendations do not require all commands to be implemented. IMAP commands are divided into REQUIRED, RECOMMENDED and OPTIONAL. RFC-2119 describes the exact definitions of these and other terms [11].

For example, in Microsoft's open specification documentation [5], the level of support for Outlook's IMAP4 service did not implement the RFC4315 Required, Recommended and Optional portions. RFC4315 was developed to provide features to reduce the amount of time and resources used by some disconnected-use client operations. Moreover, RFC3501 Recommended portions and some parts of the Optional portion are also not implemented in Outlook. Outlook ignores the next unique identifier value under UID message attribute and some flags under Flags message attribute. Outlook also does not use the CHECK, EXAMINE, SEARCH, UID SEARCH commands and ignores untagged responses UNSEEN and UIDNEXT. Finally, Outlook never does a partial fetch using FETCH command.

2.5 IMAP server Performance

IMAP servers require resources like other server systems/processes. However, an IMAP server must be able to handle large number of connections and processes. It also should stay up 24/7 to provide uninterrupted service to users. Moreover, it should have adequate disk space to store and memory to process even very large numbers of mail messages with little or no significant performance degradation. However, the optimal type of resources and platform is highly dependent on the number of current users and the rate of

```
029 IDLE
+ idling
* OK Still here
* OK Still here
* OK Still here
* OK Still here
* OK Still here
DONE
029 OK Idle completed.
```

Figure 2.32: The IMAP IDLE and DONE Commands

growth. Good capacity planning before putting the server in service usually prevents future problems as well as building users' confidence. Some suggest that the normal workload should not be greater than half the capacity of a mail server [26].

Disk is the most important fixed resource when it comes to an IMAP server. The disk space requirement can grow up to 100 percent per year [26]. In an IMAP server configuration, disk is the most crucial factor because the servers are I/O bound. If the disk configuration is not tuned appropriately, the system might spend unnecessary time in the I/O wait state [26].

An IMAP server's memory requirement is one of the difficult areas in server configuration since it is directly related to the number of users that are actively reading their mail simultaneously rather than to the total number of mail accounts. Moreover, each user connection has different memory requirements since each user spends different amounts of time connected to read and reply to emails and they have different sized mailboxes. The IMAP process must retain the physical memory allocated to a client process until that specific connection is disconnected. Although memory shortages can be solved by swapping out client process(es), minimzing such instances results in the best performance of the server's memory subsystem [26].

2.6 Related Work

As of this writing, there are no previous studies directly comparing the performance of IMAP servers with respect to the impact of different IMAP clients. However, there are several studies which study related areas and issues with respect to IMAP server performance.

One long term traffic traces study, based on application level analysis of SMTP, POP3 and IMAP performance and traffic characteristics, showed that both the duration of command exchange before emails were transmitted and the number of emails transmitted make a significant contribution to the heavy-tailed

distribution. In the same study, serial processing of commands for sending and receiving email was shown to account for a significant share of the latency [12].

Another study on organizational network traffic showed that the email traffic from SMTP and IMAP constitutes 94 percent of the total (in bytes) [41]. However, these researchers found no significant difference between internal and wide area IMAP email traffic in terms of connection sizes except that the former connections are longer-lived. In the same study, the direction of the traffic volume of SMTP and IMAP/S was largely unidirectional to SMTP servers and to IMAP/S clients. This study also discovered that the success rate for IMAP/S connections was 99-100 percent.

A performance comparison between different email storage options provided by IMAP server indicated that the combination of the Cyrus IMAP and mySQL has an advantage over the Courier-IMAP and UW-IMAP server for searching and scanning header fields [20]. The result difference was mainly because of mySQL's full text indexing, which significantly sped up the searching. Moreover, a server-side buffer cache also is another factor that improves performance by providing fast access to recently accessed data during searching. The final conclusion was that IMAP servers with DBMS based file access can search email better than a traditional file-based one.

A security comparison based on an automated entry/exit analysis between open source IMAP servers UW-IMAP, Cyrus, and Courier-IMAP revealed that Courier was the best product of the three with respect to security risk [24].

2.7 Email Infrastructure Future Directions

There is no doubt that the functionality of email services will certainly continue despite the fact that different new technologies are emerging to communicate digital messages. Many agree on the need to change on the current system due to its susceptibility to Spam and security issues. However, the question remains as to what within the system needs to change? There is no clear future design for email infrastructure but the following points have been raised by various researchers in different contexts.

The advancement of human life style with rapid changes in handheld device technology allow users to manipulate digital messages including email. Cellular phones, Personal Digital Assistants (PDAs), iPhones, iPods and email embedded application software are demanding a radical change in the features of MUAs. For example, MUAs are being developed in diversified forms to be as small as possible to fit the capacity of the handheld devices. At the same time, large single user desktop MUAs that can handle multimedia MIME types and graphics are also under development. This change in MUAs calls for upgrading the email infrastructure. For example, BlackBerry Enterprise Server offers

BlackBerry smart phone users the ability to manipulate their email messages and other digital messages. The email is provided by push technology to the users and synchronizes with their desktop mail client [8].

To implement such advancements over a large scale, scalability is the major issue to address in the email infrastructure, necessary due to mobile emailing demands for synchronization. The issue of synchronization is crucial for handheld devices since they have small memories, and they generally do not download messages like desktops do. Moreover, the cost of Internet connections in small devices is rather high compared to other types. In this regard, IMAP plays a central role since the handheld devices prefer to manipulate messages remotely.

With respect to large desktop and single user MUAs, the application software is becoming larger because they are expected to display growing MIME types locally. The demand from the user side to exchange more types of MIME types, HTML formatted text, JPEG and PNG images, and PDF documents makes email client development difficult and complicated. It is worth noting that integration of MUAs with web browsers has become important due to the widespread use of embedded hyperlinks included within email messages [52].

Many applications now must also function in a virtualization environment. However, corporate email infrastructure has not yet exploited the benefits of virtualization. The administrative efficiency observed by other applications through virtualization should motivate sites to consider hosting their mail servers in virtualized environment. This invites researchers to experiment more on possible performance bottlenecks especially in disk I/O and network traffic and high availability of email service if implemented under a virtualized environment.

Chapter 3

Experimental Design and Methodology

This chapter describes the methodology utilized to conduct the experiment. This discussion will include consideration of the following points:

- The design of the experimental environment.
- The design and goals of the experiments.
- The tools and procedures used to carry out the experiments.
- The statistical analysis of the observed performance metrics.

3.1 Experimental Environment

In computer experiments, a large experimental domain is employed to explore complicated non linear functions. This kind of experiment involves observing a large number of variables to obtain the desired results. Since computer experiments are deterministic, which means identical output should be obtained from samples with same input settings [28], one has to able to control the environment where the experiment is going to be conducted. Thus, in the following section, the physical hardware environment settings and the IMAP server and client computer system setup will be discussed.

3.1.1 Hardware Environment

Figure 3.1 shows the experimental lab physical network setup. The IMAP server system was assembled with two network interfaces, one for the external Internet network and the other for communication with the client system.

A cross-over cable was used to connect the IMAP server with client computer. During every active experimental period, the external interface of the IMAP server was blocked. This was the control mechanism used to avoid external network traffic interference.

The whole experiment was controlled through the external interface of the client system. Communication was accomplished using SSH.

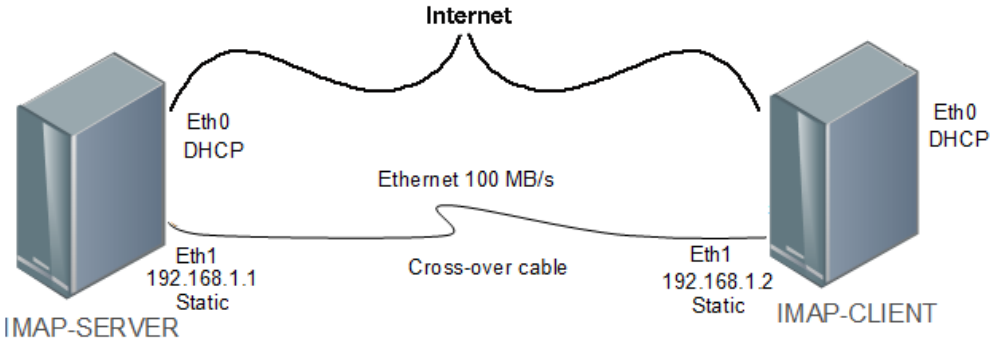

Figure 3.1: Experimental Physical Hardware Setup

IMAP Server System Hardware

This subsection describes the IMAP server system hardware configuration. The report was gathered using linux lshw command, and it includes the memory configuration, firmware version, main board configuration, CPU version and speed, cache configuration, and bus speed for the system.

1. Motherboard

   Product: 0XC320 Vendor: Dell Computer Corporation.

2. BIOS

   Vendor: Dell Computer Corporation; Physical id: 0; Version: A07 (04/25/2008); Size: 64KiB; Capacity: 960KiB.

3. CPU: There are two identical CPUs

4. CPU 0

   Product: Intel(R) Xeon(TM) CPU 3.00GHz; Vendor: Intel Corp.; Physical id: 400; Bus info: cpu@0 and cpu@1; Version: 15.4.10; Serial: 0000-0F4A-0000-0000-0000-0000; Slot: PROC-0 and PROC-1; Size: 3GHz; Capacity: 3600MHz; Width: 64 bits; Clock: 800MHz.

5. Memory Description: System Memory; Physical id: 1000; Slot: System board or motherboard; Size: 2GiB.

6. SCSI Disk

   Product: LD 0 RAID1 69G; Vendor: MegaRAID; Physical id: 2.0.0; Bus info: scsi@0:2.0.0; Logical name: /dev/sda Version: 5B2D; Size: 68GiB (73GB).

   - Volume:0
     Description: EXT3 volume; Vendor: Linux; Physical id: 1; Bus info: scsi@0:2.0.0,1; Logical name: /dev/sda1; Logical name: /; Version: 1.0; Serial: a96f4d92-5dab-48b0-816d-4889d1506136; Size: 67GiB; Capacity: 67GiB.
   - Volume:1
     Description: Extended partition; Physical id: 2; Bus info: scsi@0:2.0.0,2; Logical name: /dev/sda2; Size: 729MiB; Capacity: 729MiB; Capabilities: primary extended partitioned partitioned:extended.
   - Logical Volume
     Description: Linux swap / Solaris partition; Physical id: 5; Logical name: /dev/sda5; Capacity: 729MiB; Capabilities: nofs.

7. Network: 2 identical Ethernet interfaces eth0 and eth1

   Product: 82541GI Gigabit Ethernet Controller; Vendor: Intel Corporation; Physical id: 7 and 8; Bus info: pci@0000:06:07.0 and pci@0000:07:08.0; Logical name: eth0 and eth1; Version: 05; Serial: 00:14:22:20:57:1d and 00:14:22:20:57:1e; Capacity: 1GB/s; Width: 32 bits; Clock: 66MHz; Capabilities: pm pcix bus-master cap-list ethernet physical tp 10bt 10bt-fd 100bt 100bt-fd 1000bt-fd autonegotiation; Configuration: autonegotiation=on broadcast=yes driver=e1000 driverversion=7.3.20-k2-NAPI duplex=full firmware=N/A ip=192.168.1.101 latency=32 link=yes mingnt=255 module=e1000 multicast=yes port=twisted pair speed=1GB/s.

3.1.2 System Environment

Operating System Configuration

Debian GNU/Linux 5.0.5 Lenny was selected as the operating system for the experiment. The default packages installed by the distribution were used to run the system. A few services were disabled, including syslog, cron and the exim4 default SMTP protocol (see Figure 3.2).

The following additional packages were required for the experiments, and they were installed on IMAP server and run during the experiments.

1. SSH

   This package was required for logging into a remote machine and for executing commands on the IMAP server and client system remotely. A

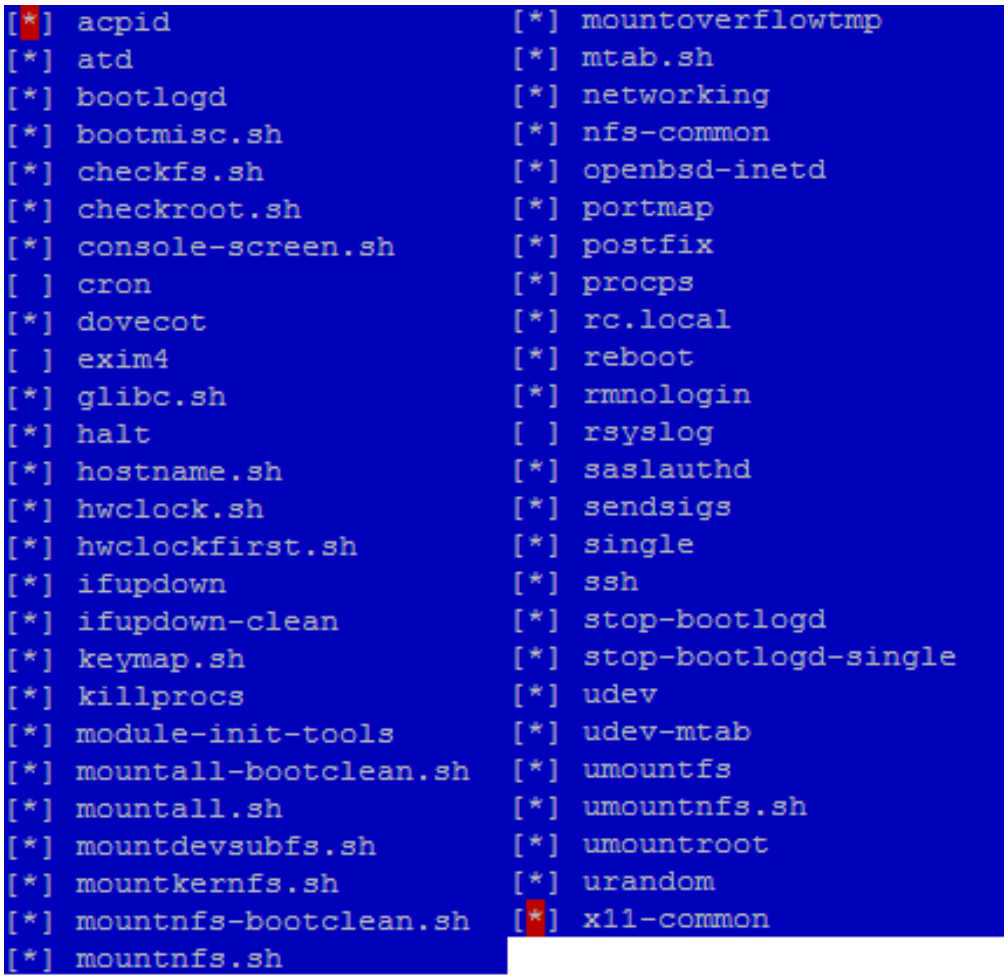

Figure 3.2: Services Running on the IMAP server During Experiments [List obtained from the Debian Runlevel configuration tool rcconf]

passwordless connection between the IMAP server and the client system was achieved via the ssh-keygen mechanism (which is used to generate and manage keys for SSH authentication).

2. Postfix

   This package was required as a Mail Transfer Agent (MTA) to deliver messages to mailboxes. Postfix was installed on IMAP server. The default configuration file, main.cf, was used for this experiment with the addition of the entry "home_*mailbox = Maildir/" at the end of the file.

   To make the Maildir message format workable (this mailbox format is described later in this chapter), the /etc/procmailrc file was created with "DEFAULT=$HOME/Maildir/" as its first entry.

   Since Maildir works with filesystems that support directory indexes, the third extended filesystem (ext3) type was chosen for the volume holding the mailstore. Ext3 is a Linux journaled file system, and it is the default file system for most Linux distributions. Ext3 supports directory indexing, but it needs to be enabled. The following command checks whether directory indexing is enabled or not:

   tune2fs -l /dev/sda1 — grep features

   In this case, directory indexing was enabled because "dir_*index" was listed among the filesystem features. Otherwise, it can be enabled for this filesystem as follows:

   umount /dev/sda1
   tune2fs -O dir_*index /dev/sda1
   e2fsck -fD /dev/sda1
   mount /dev/sda1

   The following library packages also were installed with Postfix: postfix-tls, libsasl2-2, sasl2-bin, and libsasl2-modules.

3. Dovecot

   Dovecot is an Open Source IMAP and POP server for Linux operating systems. The packages dovecote-imapd and dovecote-common were installed to run the Dovecot server. The default configuration in /etc/dovecot/dovecot.conf was used, with the following changes:

   - disable_*plaintext_*auth = no
   - mail_*location = maildir:~/Maildir

4. Dstat

   Dstat is a plugin-based, real-time monitoring tool for generating system resource usage statstics [50]. This tool was used to collect the required data in this experimental setup. It was selected because it allowed network bandwidth measurements to be compared directly with disk throughput during the same time interval [50]. In addition, the tool's ability to export the results in CSV file format was very convenient.

5. Psmisc

   This package was required because it contains the killall command, which was used to kill processes by name.

6. Shar Utils

   This package was installed because it contains the uuencode command. The uuencode command was used to encode binary files created by the dd command in order to create test messages from binary files.The combination of this command with dd proved a useful method of easily increasing or decreasing message size for the experiment.

The following packages were required and installed on client system.

1. Expect

   Since the simulation package is written in using the Expect program (see below), this package was installed and run on the client system. The Expect programming language was used to simulate the clients' behavior because of its capability to automate interactive processes, in this case, an interactive session between the email client and the IMAP server[30] and [31].

3.2 Major Experimental Components Selection

3.2.1 IMAP client selection

The following five email clients were selected for study and comparison:

- Microsoft Office Outlook 2007 SP2 MSO
- Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.10) Gecko/20100512 Thunderbird/3.0.5
- Opera/9.80 (Windows NT 6.1; U; en) Presto/2.5.24 Version/10.54
- Sylpheed version 3.0.0 GTK+ 2.10.14 / GLib 2.12.13 Operating System: Win32

- Mulberry V4.0.8

Five desirable email client attributes were identified and used as the basis of selection. They are: popularity among propritory email clients, popularity among Open Source email clients, simplicity, full integration with a browser, and homogeneity.

Microsoft Outlook 2007 was selected as a candidate because it is one of the top popular, widely used proprietary desktop email clients, although it is platform dependent. Thunderbird was selected because it is a widely used an Open Source desktop email client. Opera was selected because it is fully integrated with Opera web browser. Mulberry was selected because it supports only IMAP, fulfilling the homogeneity criterion. Finally, Sylpheed was selected because it is simple, lightweight yet feature-full and easy to use. The comparison was not intended to compare proprietary vs Open Source software, but rather to investigate differences in IMAP protocol implementations.

3.2.2 IMAP Server Selection

Dovecot is a widely-used Open Source IMAP and POP3 server for Linux/UNIX-like systems. It is fast, simple to set up, and it requires no special administration[45]. Its simplicity is the main reason for selecting it for this experiment because the main purpose of this thesis is studying and comparing IMAP clients behavior under different scenarios.

3.2.3 Mailbox Format Selection

There are several different types of mailbox formats in use today. The most widely used are two: the *flat-file* and *file or message* formats. In the flat-file format, the mailbox and messages in the mailbox are stored in a single file. In the second type, file or message format, the mailbox is a directory and messages under it exist as a separate file citeweb10.

The file or message format was selected for this experiment because the format is highly responsive to disk I/O-intensive IMAP commands like deleting and file status change. This is because of its high amount of inode usage on the disk. In the extreme case, this can result in disk thrashing and other I/O problems when there is an very large amount of message manipulation, especially for creating and deleting messages[18]. While such situations are best avoided in production environments, this tendency could actually prove favorable to an experiment comparing different clients' behavior as problems could manifest themselves more easily and dramatically, making them easier to detect and measure. Thus, among the many file or message format schemes, Maildir was selected for this experiment.

3.3 Client Behavior Simulation

In order to investigate the research questions posed by this thesis, it is necessary to observe email client operations in a realistic manner and environment. However, the inherent interactive nature of these processes makes such tasks challenging. Accordingly, a substantial effort was required in order to perform such operations in a manner that is amendable to repeatable observation and scientific study. Clearly, performing repeated tests of email clients manually would be both inordinately time consuming and difficult to reproduce, and any results would be accordingly unreliable. A way to automate this process to minimize time requirements to perform the experiments and guarantee the reproducibility of the observed metrics was essential.

Currently, there is no benchmarking tool focused on IMAP client behavior. The few related tools that do exist, such as SLAMD [49] and MSTONE [51], focus on stress and performance measurements and thus do not reflect the performance overhead from clients' IMAP implementation choices. The reason that these and other similar benchmarking tools were not selected for this experimental purpose is that:

1. These tools mix SMTP traffic and performance overhead with IMAP during the experimental period, and isolated IMAP traffic was needed for valid network bandwidth metrics.

2. They are designed to conduct different level of stress tests against IMAP servers, not simulate normal operations.

3. Although the tools utilize IMAP commands for benchmarking purposes, they do not reflect actual behavior of existing IMAP client softwares. Thus, they cannot simulate a *specific* client's IMAP behavior.

Therefore, this thesis introduces a new concept of IMAP server benchmarking IMAP that accurately simulates the differing behavior of IMAP client software and their implementation differences.

3.3.1 Preparatory Investigations of Client IMAP Implementations

The benchmark tool created for this thesis simulate the precise IMAP implementation of each client software program. In order to do so, this low-level functioning must be observed and recorded. This preliminary step is common requirement in Open Source programming projects such as the SAMBA facility [48][13].

The methods employed by each email client to communicate with IMAP server for each mail operation were studied by observing the network traffic associated with each supported operation using Wireshark[1]. Each operation was performed manually using the unmodified client software at the GUI level.

A traffic filter was created for the manipulation of a group of 80 email messages. The network traffic was then dumped while the activities were accomplished for all five selected five. The following steps were followed:

Step 1: A script was prepared to create the necessary message folders, and then subscribe to them. The script was used again in the finished benchmarking tool for message folder preparation.

Step 2: The user logged in and waited for messages.

Step 3: Wireshark was started to capture the traffic between the client machine and the server for the filtered IMAP protocol traffic.

Step 4: A prepared script that sends one message every 30 seconds was started. The script sent a total of 80 messages. The 30 second interval is to allow time for the user to manipulate the message. This script was also ultimately incorporated into the completed benchmarking tool.

Step 5: As each message arrived at the client machine, the user manipulated them, based on a previously-generated random number that determined the kind of manipulation for that message. Table 3.1 shows the possible outcomes for messages. The Action column shows what happened to the individual messages: MOVED with Fx indicates the message was moved to Folder x; DELETE means that the message was deleted; SPAM indicates the message was considered as spam and handled according to the email client's defaults. The No. column shows the sequence number corresponding to the message's arrival in the client software.

Step 6: The 80 messages were divided in to four groups of 20 messages. After manipulating all of the messages with a group, the user then SELECTs each of four message folders, one by one, and reads the messages newly moved from the Inbox. The number of messages that were moved to each message folder after the completion of each message group is summarized in the following tables. In each table,l the first column shows the list of created message folders, the second columns shows the amont of messages moved to each folder, and the third column shows the message IDs number upon arrival in the message folder.

Step 7: The client program terminated connection with the IMAP server, and the user was logged out.

Step 8: Wireshark was stopped from capturing the traffic.

[1]Wireshark is a network protocol analyzer for Unix and Windows [44]

Steps 1 through 8 were repeated for all selected client softwares. The captured traffic was then studied to determine the underlying IMAP commands issued, and this information was prepared for simulation.

3.3.2 Programming

Once the network traffic data was analyzed, the Expect facility was used to simulate email client's communication with the IMAP server. The traffic captured using Wireshark was studied, and each client's activity was reproduced using Expect. A flow chart was prepared for each client before programming started. For example, the flow chart for the Sylpheed email client is shown in Figure 3.3.[2] The simulation scripts are not included with this thesis report because of their size. The scripts can be provided by request.

This inherent behavior of Expect was very beneficial to the ultimate goals of the experiment. Fore example, it is the nature of Expect not to not issue the next request before the server responds to the previous request, making scripting using it essentially self-throttling. Therefore, if disk I/O performance is fast, the IMAP server is able to respond to commands sent from clients quickly. If the disk I/O performance is slow, there could be high I/O wait, and the IMAP server will respond to clients more slowly. Expect handles all cases smoothly and reliably. In this way, the whole experiment was controlled, and the relationships and trends among the selected performance metrics were maintained. Moreover, it was also very easy to monitor other undesirable performance overhead that appeared in the experiment.

3.3.3 Benchmarking Tool Architecture

The custom benchmarking tool has three major components:

- The first component is a collection of scripts written in Expect that simulate the five selected IMAP client software.
- The second component is a collection of preparation scripts written in Expect and bash. These include a script that sends message to mailboxes of a specified message size and ID, a script that creates no-login user accounts on server, a script that clears out the existing mail folders directory, and a collection of scripts written for each client that prepares the mailbox for next experiment, depending on the experimental requirements.

[2]Since drawing the flow chart in a publication worthy manner requires a long time, and the programming implementation is similar for the other clients, the flow charts for the other clients are not included in this thesis. At the time of developing the benchmarking tool, hand written flow charts were employed. One can easily understand the simulation scripts for the other clients using Sylpheed's flow chart as a template.

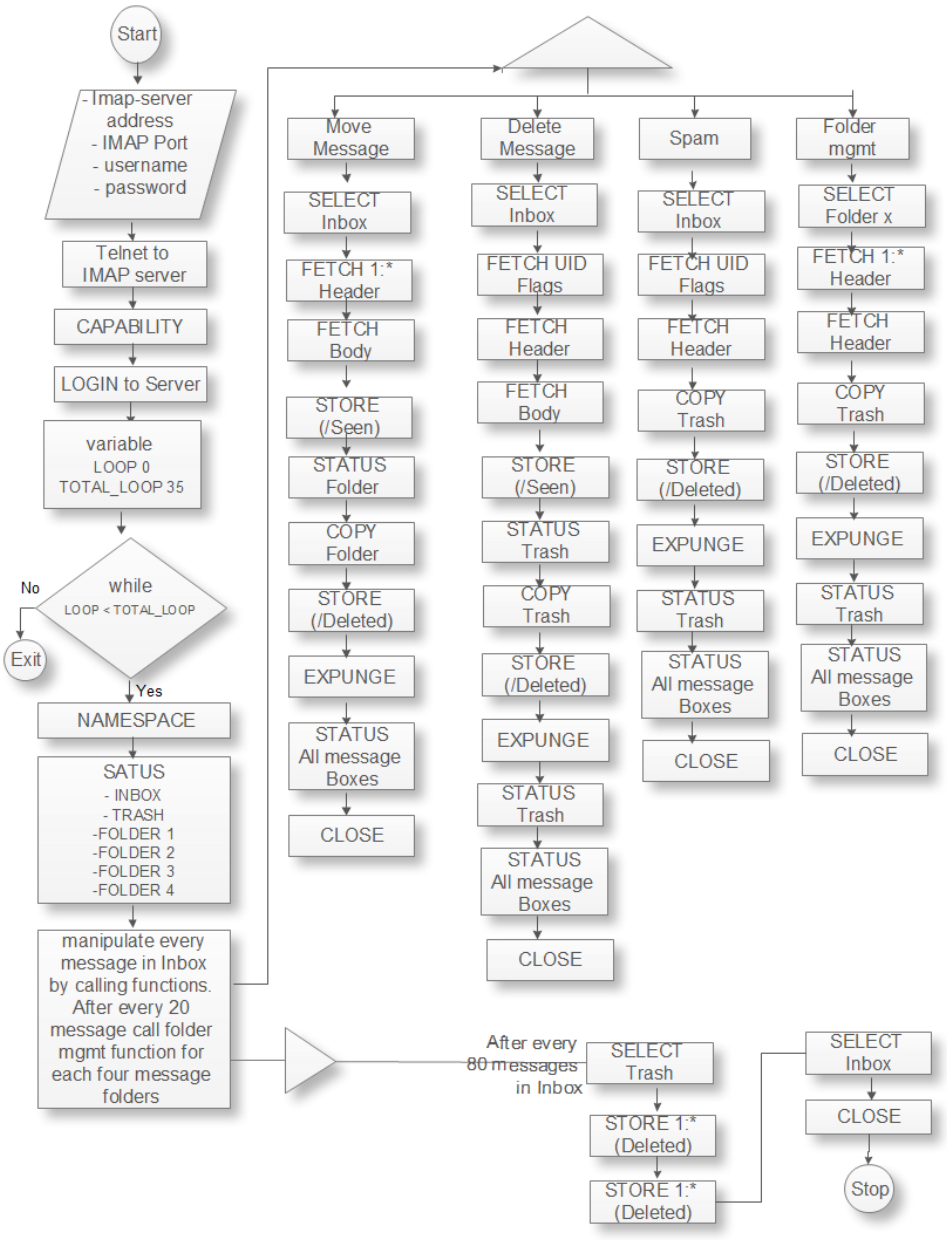

Figure 3.3: Flow chart for Sylpheed Optimized Behavior Simulation Script

- The third component is the main driver script which automates the whole benchmarking process. It is written in bash. It prompts for various experimental parameters and then controls the experiment according to the specified requirements. After collection, the results are put either in CSV format or in text file, so that the experimenter can analyze them as required.

The code for the benchmarking tool is included in an appendix.

3.4 Experimental Design

Identical experiments were run in order to compare the result of different clients' behavior before, up to, and after message manipulation. Each experiment was replicated 35 times because the standard deviation was not known in advance. For stochastic data such as this, each single group of message manipulations was were repeated so that a steady state pattern was achieved to obtain accurate results concerning the trends of client behavior.

The experiment were controlled and automated by the custom benchmarking tool described previously. This helped to eliminate human induced errors from manual operations. Among the many tasks the script accomplishes are: rebooting the server between each replication, ensuring that the desired number of clients are running , starting and stopping the performance monitoring tools, and the like. The functioning of the main automation component of the benchmarking tool is illustrated in Figure 3.4. The scripts to automate the benchmarking process are attached in Appendix B.

3.4.1 Message Group Selection

Message group represents 80 messages in a group. The 80 message group is based on survey results from the Radicati Group [46]. According to the survey, by 2014, the average number of messages received by a typical corporate user per day will be 80. Among these messages, only 65 will be legitimate and the rest will be Spam. Therefore, the number of messages in a group is set at 80.

3.4.2 Message Size Selection

Message sizes during the experiments should reflect realistic average email message sizes. However, it is difficult to find relevant scientifically supported data in this regard. There are different sources that suggest the average size of an email message. About.com reported that average email size is 75 Kbytes [47]. As the source reported, attachments, news letters and marketing email

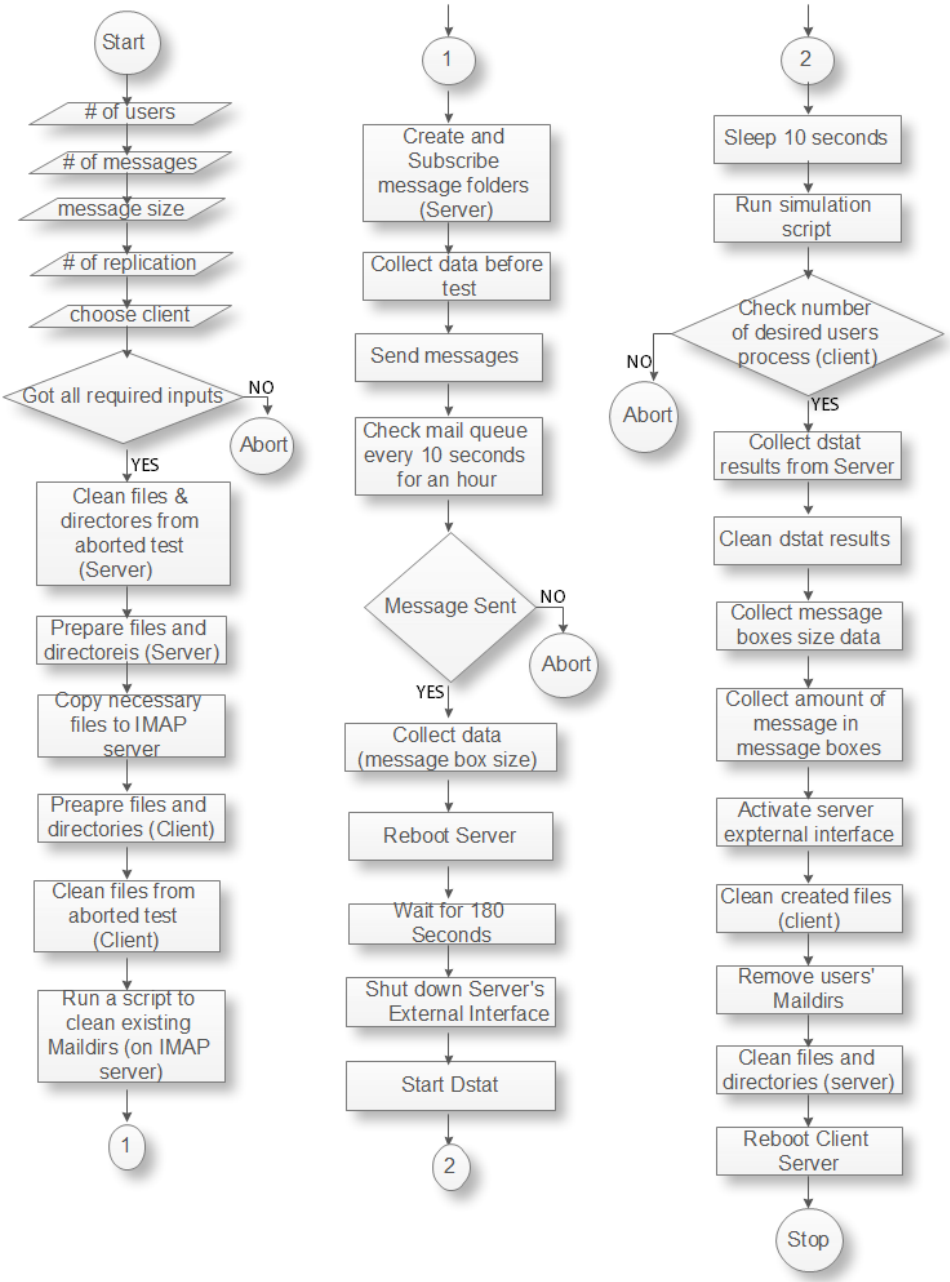

Figure 3.4: Benchmarking Automate Script Flow Chart

skewed the average to this level. Thus, this thesis used this information to decide on the size of an email message for experiment purposes.

Two sizes of message were used to compare the different IMAP clients: 3.4 Kbytes and 76 Kbytes, representing small and average message sizes. Note that the small message size simulates typical text-only messages well, the type which makes up a substantial portion of actual communication-related email messages. Time constraints did not allow messages larger than average to be included in this thesis. Each scenario discussed below was tested with these two message sizes.

3.4.3 Client Scenarios: Default vs Optimized

Client behavior was actually simulated in two modes: the default mode used by the unmodified client software as installed, and an optimized mode corresponding to modifying available client software options and preferences to provide the best possible performance. Both the default and optimized version experiments were repeated with both message sizes (3.4 KB and 76 KB).

Default Client Configurations

The default scenario tried to simulate the IMAP clients' behavior without optimizing it. It focuses on default behaviors that do not have a direct effect on users' message manipulation but might have an impact on server side resource usage.

For example, different clients implement the final destination of message after deletion differently. Outlook and Mulberry leave the deleted messages in Inbox by marking them as deleted, visually in client GUI and flagging them as deleted on IMAP server. In contrast, Sylpheed and Thunderbird move deleted items to Trash folder (which is created and subscribed during initial installation of the client). the Opera mail client just flags messages as deleted and shows the deleted messages in Trash folder (again created on the client machine during initial configuration). It has to be noted that the Trash folder does not exist on IMAP server. Thus, Opera, Outlook and Mulberry do not change UID value that each message received at message arrival in Inbox but identify them as deleted via flags.

Opera provides two options for deletion. The first mode is used when user wants to delete messages permanently. This is useful when deleting unread spam messages. The second mode is performed via the keyboard delete key or by right clicking on the message and selecting "Move to Trash." This feature is shown on Figure 3.5. Note that the Trash folder is visible on client machine. When messages are deleted, they exist physically in Inbox but appear to be in Trash folder. The user can view the messages in Trash as shown in Figure 3.6.

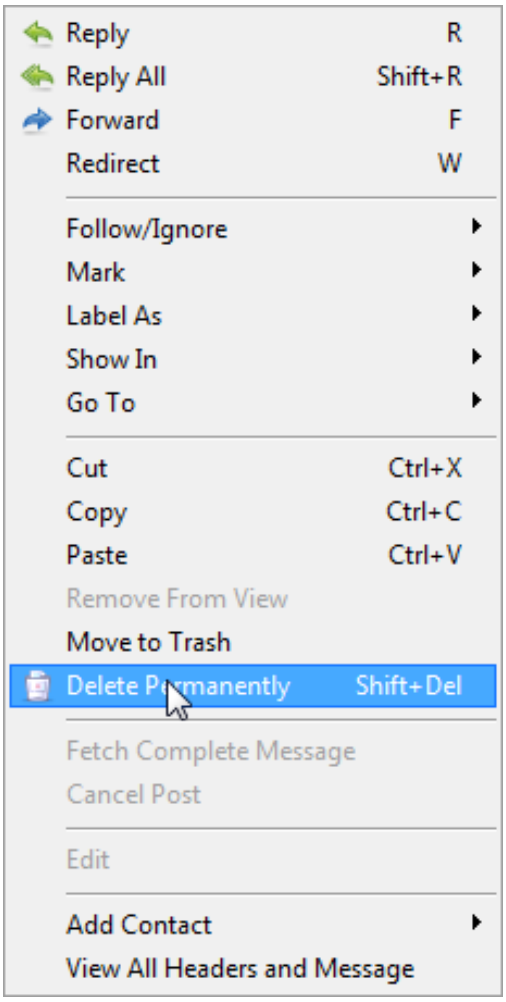

Figure 3.5: Opera Option for Handling Deleted Messages

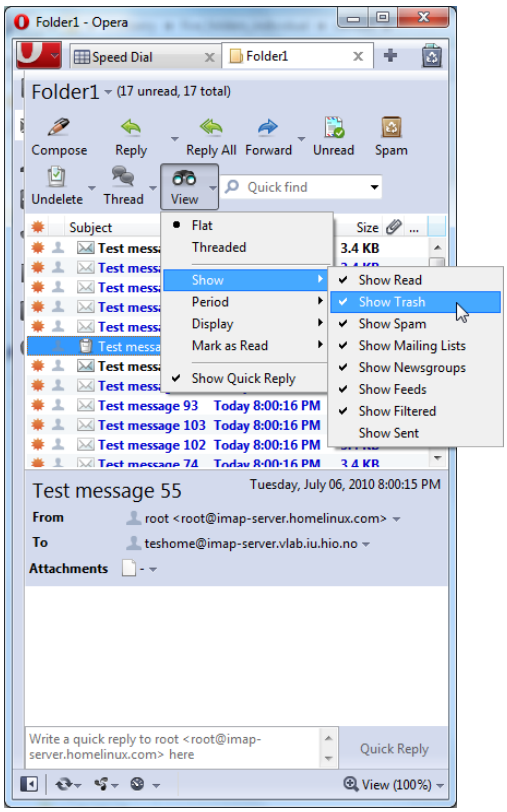

Figure 3.6: Trash Messages in the Inbox

In contrary, Sylpheed and Thunderbird movs the deleted messages permanently from Inbox to Trash folder on server side and provide them with new UID values. This scenario is selected to find out if there were any performance difference between the different clients resulting from the differences in handling deleted messages.

The other important varying default behavior of email clients is in handling the EXPUNGE command. As it has been explained in the background part of this thesis, the EXPUNGE command permanently removes messages with deleted flag set. Opera by default utilizes the EXPUNGE command after deleting messages permanently or moving them to another folder. However, Sylpheed, Outlook and Thunderbird do not use EXPUNGE command to clear messages with deleted flags unless the user explicitly specifies it. Similarly, Mulberry does not use EXPUNGE command by default, but it has a button on front side of its GUI that invokes it.

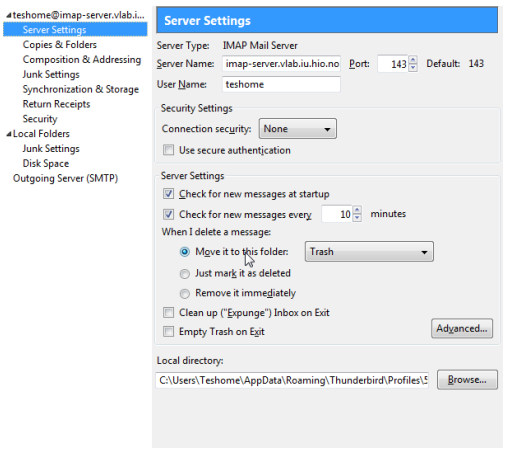

Figure 3.7: Thunderbird's Default settings

Optimized Client Configurations

The optimized scenario was designed to investigate on different clients' capabilities and flexibility in manipulating messages. For example, Thunderbird provides three choices to users for the destination of a deleted message. Users can choose to "Move it to a folder", giving the user flexibility missing from other clients. Users can also choose to "Just mark it as deleted," equivalent to the behavior of Outlook and Mulberry. Finally, users can choose to "Remove it immediately," probably an unpopular option since no one wants to lose a message forever if it is deleted by mistake. These options are offered via a Radio Button control, so a user must choose only one of the three. Thunderbird also provides two options via check boxes further specifying how to deal with deleted messages when exiting from a folder. They are "Clean up Inbox on Exit" and "Empty Trash on Exit." A user can choose one of them, both,

or neither of them. Figure 3.7 shows these features within the client software GUI.

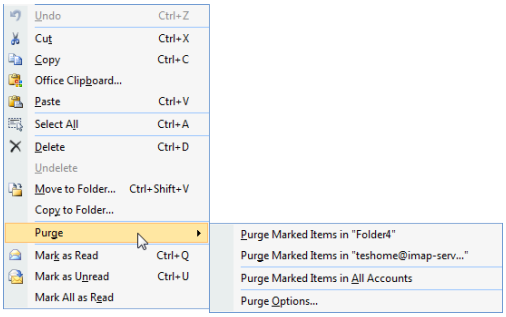

Figure 3.8: Outlook's Purge Option

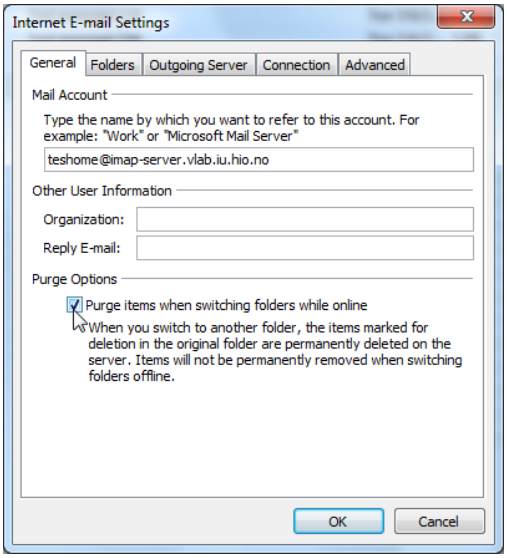

Figure 3.9: Outlook's Purge Option

Outlook provides different options in this regard. Its users can either leave deleted messages marked and visible in GUI or use the purge option shown in Figures 3.8 and 3.9. Figure 3.8 shows temporary clearing of messages with deleted flags. However, Figure 3.9 provides for clearing deleted messages while switching folders if user is online.

Opera does not need to optimize the default behavior of the client because (as explained earlier) it provides a simple purging option via right clicking on Trash folder and selecting "Empty Trash" (see Figure 3.10). Moreover, when an undesirable messages arrives, user has option to delete them permanently using the feature shown in Figure 3.5.

Mulberry by default does not use EXPUNGE command when messages are moved from one folder to another. The users should either click on the Expunge button on toolbar or optimize the default in the preferences as shown

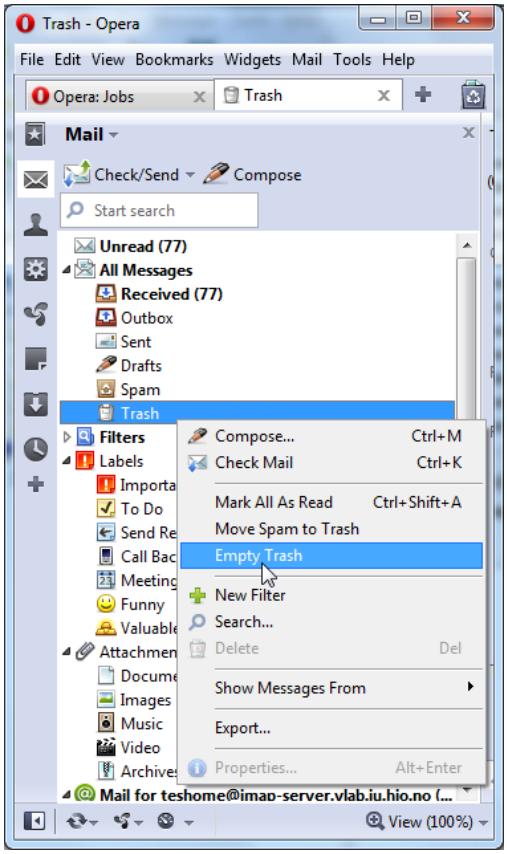

Figure 3.10: Opera Emptying the Trash Option

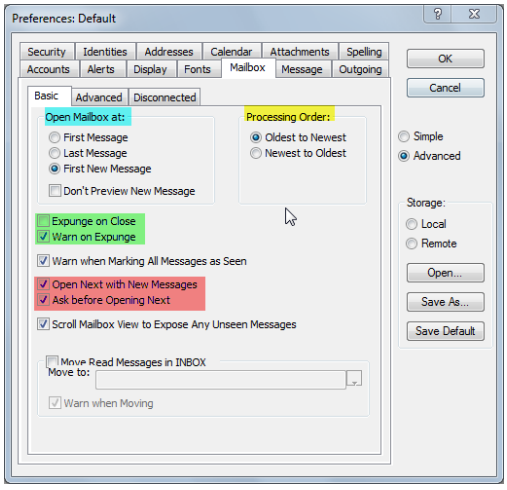

Figure 3.11: Mulberry Optimization options

in Figure 3.11). This client provides an optional expunge warning to users which is useful since Expunge command is dangerous if used inappropriately.

Sylpheed does not Expunge messages with deleted flags and does not provide an option like Thunderbird, Outlook and Mulberry to expunge messages on exit from folders. It behaves like Opera in that user can clear Trash folder from File menu's Empty all trash item.

3.5 Observed Data

This section describes the actual experimental observations in detail.

Each individual experiment was a set of three sub-experiments. Two of them are dependent on the performance measurement portion, hereafter referred to as the *main experiment*, because they were performed before and after it. The sub-experiments obtain the mailbox size before and after the main experiment. The main experiment measures system performance with respect to disk throughput (disk write and read), network bandwidth (packets received and sent), CPU time usage, system interrupts and context switches.

3.5.1 Disk Volume Utilization

The main purpose of this experiment is to compare the different clients and to find out how much server disk space utilized during experimental period.

Each mailbox folder space utilization is measured three times: (1) Right after mailbox folder is created, (2) After desired amount of message are inserted for experimental purposes, and (3) After the main experiment is completed. The du shell command is used to gather mailbox folder disk usage.

The sample size to collect disk space utilization is directly related to the actual experiment replication since the capture is controlled by the benchmarking process.

3.5.2 Performance Metrics

The following computer performance metrics were collected simultaneously using the dstat tool. 1) Disk I/) read and write performance, 2) Network bandwidth usage for packets received from and sent to the IMAP server, 3) System interrupts and context switches issued, and 4) CPU time usage by user, system, idle and I/O wait.

Disk I/O performance metrics are used to evaluate how IMAP client activities affect the read and write performance of the experimental system. For example, message downloads and copy and move operations increase disk read and write activities. Thus, the results from disk I/O performance were the direct reflection of the IMAP command's resource requirements.

Network bandwidth usage for packets received and sent was controlled by the scripts because it is the direct reflection of the amount of commands that each client sends to the IMAP server and the server responses.

The system context switch is the metric associated with the of storing and restoring state of a CPU so that execution can be resumed at a later time. In most cases, in this experiment context switching is caused by system interrupts[38]. A system interrupt is generated by the I/O controller and signals the normal completion or occurrence of an error or failed condition in the I/O activities. Thus, this performance metric is selected since it has direct relationship with disk I/O activity [38] and [27].

Among the CPU time use variables listed above, the main interest was the I/O wait time because it reflected directly the disk I/O activities[38].

These performance metrics was gathered by the dstat tool. The following commands were run on the experimental system remotely from the client side:

dstat –output experimentaloutput.csv –noheaders -dnyc -D total -N total -C total

The –ouput option were used to gather the data and save in CSV format. The –noheaders option was used to avoid headers between data. The -dnyc options tells dstat to relate disk-throughput with total network bandwidth, cpu usage and system counters.

Dstat starts and stops were controlled by the benchmarking tool so that dstat was run for 10 seconds before and after the actual experiment. Each experimental replication run was separated by rebooting the machine to control the cold cache and minimize experimental error.

The amount of time that dstat spends gathering this performance data is dependent on the client. This means, since the nature of the data collected is time series, the length of time required to run each client simulation is different. However, the benchmarking tool controls the simulation scripts existence through their PID so that when the simulation script finishes its job, the benchmarking process stops the data collection by killing the PID of the dstat process on the experimental system.

3.5.3 Number of Messages in Message Folders

This sub-experiment records the number of messages left in each message folder after completion of each replication. A script was written to accomplish this task. The script was run only after completion of the main experiment because the main experiment directly influenced the number of messages in a message folder.

3.6 Data Analysis and Interpretation

After collection, the various metrics were plotted and analyzed. Various forms of standard statistical analysis were used.

- The data collection method is time series-based. After collecting data for 35 replications of each experiment, arithmetic mean values for the 35 replication were calculated. The standard deviation measures how widely each individual values are dispersed from the mean value. The standard deviation results were used to plot error bars.
- Trend analysis was also attempted to project values in the time series graph and to compare the resource usage based on the growth trend on the graphs (via regression analysis).
- The correlation coefficient of the 10 performance metrics were calculated to determine the relationship between them. In the result tables, the correlation matrix lists the performance metrics variable names down the first column and across the first row. The diagonal of a correlation matrix indicates the correlations between each variable and itself, and these values are thus always 1 (a variable is always perfectly correlated with itself).
- The best fit function for each time-series based graph, selected based on R-squared value. This statistic makes it possible to understand the slope increase or decrease of the observed quantity as the amount of messages in the Inbox decreases or increases. The value of R-squared is also reported.
- Trend analysis was used to project values in the time series graph and to compare the resource usage based on the growth trend on the graphs.

Action	No.	Action	No.	Action	No.	Action	No.
MOVE F2	1	DELETE	21	MOVE F3	41	DELETE	61
MOVE F1	2	MOVE F1	22	SPAM	42	MOVE F4	62
MOVE F3	3	SPAM	23	MOVE F4	43	MOVE F4	63
SPAM	4	MOVE F1	24	DELETE	44	DELETE	64
MOVE F1	5	MOVE F3	25	MOVE F2	45	DELETE	65
DELETE	6	DELETE	26	MOVE F4	46	MOVE F3	66
SPAM	7	DELETE	27	MOVE F2	47	SPAM	67
DELETE	8	MOVE F2	28	MOVE F3	48	DELETE	68
MOVE F4	9	MOVE F3	29	MOVE F1	49	MOVE F4	69
SPAM	10	MOVE F2	30	DELETE	50	DELETE	70
SPAM	11	DELETE	31	DELETE	51	MOVE F2	71
MOVE F1	12	MOVE F4	32	DELETE	52	MOVE F3	72
SPAM	13	MOVE F2	33	MOVE F1	53	MOVE F1	73
SPAM	14	DELETE	34	MOVE F2	54	DELETE	74
MOVE F4	15	MOVE F4	35	SPAM	55	SPAM	75
DELETE	16	SPAM	36	DELETE	56	MOVE F2	76
MOVE F3	17	MOVE F1	37	MOVE F1	57	MOVE F4	77
DELETE	18	MOVE F3	38	SPAM	58	DELETE	78
DELETE	19	SPAM	39	MOVE F3	59	MOVE F2	79
DELETE	20	SPAM	40	DELETE	60	DELETE	80

Table 3.1: This Table Shows the fate of each message as they arrived in mailbox. The action column shows what happend to the individual messages. If MOVED with Fx then it indicates the message was moved to Folderx. If DELETE, then the message was deleted. If SPAM then the message was considered as spam and deleted based on the email client implementation. The No. column shows the sequence number according to the messages arrival in the clients software.

Folder	No. of messages	message No.
F1	3	1,2,3
F2	1	1
F3	2	1,2
F4	2	1,2

Table 3.2: This table shows the summery for each message folders created afer the first 20 message group manipulation. The first column shows the list of the created message folders. The second shows the amont of messages moved to each folder. The third one shows the message UID number while arriving in the message folder.

Folder	No. of messages	message No.
F1	3	4,5,6
F2	3	2,3,4
F3	3	3,4,5
F4	2	3,4

Table 3.3: This table shows the summery for each message folders created afer the second 20 message group manipulation. The first column shows the list of the created message folders. The second shows the amont of messages moved to each folder. The third one shows the message UID number while arriving in the message folder.

Folder	No. of messages	message No.
F1	3	7,8,9
F2	3	5,6,7
F3	3	6,7,8
F4	2	5,6

Table 3.4: This table shows the summery for each message folders created afer the third 20 message group manipulation. The first column shows the list of the created message folders. The second shows the amont of messages moved to each folder. The third one shows the message UID number while arriving in the message folder.

Folder	No. of messages	message No.
F1	1	10
F2	3	8,9,10
F3	2	9,10
F4	4	7,8,9,10

Table 3.5: This table shows the summery for each message folders created afer the fourth 20 message group manipulation. The first column shows the list of the created message folders. The second shows the amont of messages moved to each folder. The third one shows the message UID number while arriving in the message folder.

Chapter 4

Results

This chapter presents the results from the actual experiments conducted.

4.1 Overview of Presented Results

The chapter is divided separate sections for each client: Mulberry, Opera, Outlook, Sylpheed and Thunderbird. Each client section contains the following results:

- Graphs plotting the results of the experiments using the small message size of 3.4 KBytes and simulating the client software's default behavior: disk I/O read and write performance, network bandwidth sent and received, system interrupt levels, context switches, and total system CPU usage. These graphs include error bars.
- The correlation matrix between the preceding performance metrics over time. Each correlation matrix lists the performance metrics' variable names down the first column and and across the first row. The diagonal of the correlation matrix always consists of ones (these are the correlations between each variable and itself, always perfectly correlated). The value of any correlation coefficient must be between +1 and -1. In this thesis, correlation coefficient values with absolute values greater than or equal to 0.9 but less than 0.95 are considered to be correlated (positively or negatively, depending on the sign). Correlation coefficient absolute values greater than or equal to 0.95 are considered to be very strongly correlated. Negative correlation indicates an inverse relationship between performance metrics variables: as one increases, the other decreases. Positive correlation indicates a direct relationship between performance metrics: as one increases the other increases. Values approaching zero indicates the absence of any relationship between the performance metrics.

- The same result presentations are repeated for the experiments using the small message size and simulating the client software's optimized behavior.

- Graphs plotting the results for the client's default and optimized behavior for disk I/O and network bandwidth. These allow an easy comparison between the two operational modes for the most important performance metrics.

- Graphs plotting the results of the experiments using the default client behavior and the average message size of 76 KBytes: disk I/O read and write performance, network bandwidth sent and received, and CPU I/O wait time. For convenience, the corresponding small message size data is included in each plot for comparison purposes. Error bars are omitted from these graphs due to time constraints and as they follow the same trends as for the corresponding small message size data.

- Graphs plotting the results of the experiments using the optimized client behavior and the average message size for the same metrics as the preceding. Again, the corresponding small message size data is included for comparison purposes.

4.1.1 Visual Analysis

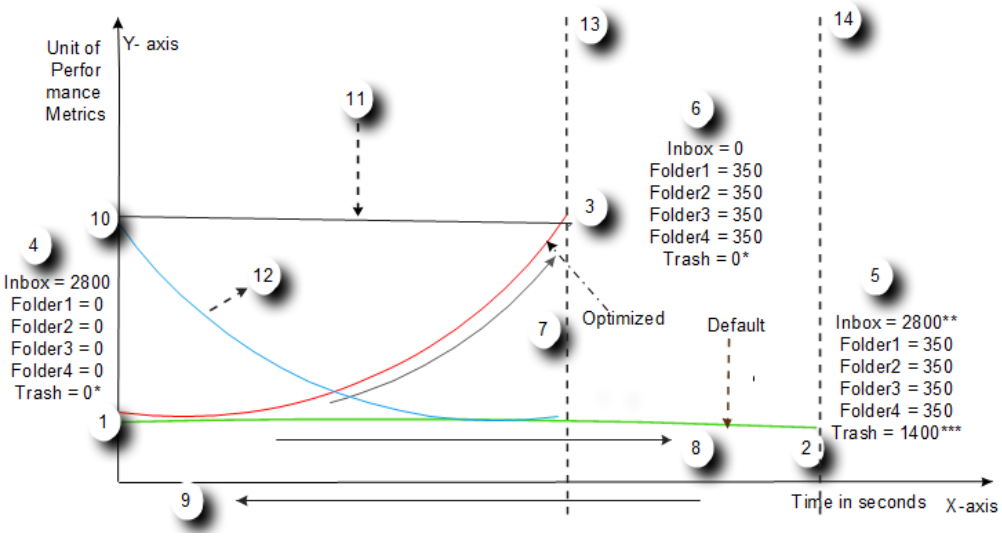

Figure 4.1: This example graph presents data for the client's default and optimized behavior.

This section elaborates how to interpret the graph results obtained from the actual experiments. The discussion is based on the model graph shown in Figure 4.1.

2800 messages were put in mailbox before the start of all experiments. 80 messages at a time were manipulated by each client until all 2800 messages in mailbox were handled, for both the default and optimized behaviors. This process is controlled by the benchmarking tool explained previously. The following explanation summarizes the data presented in each graph and places it into its experimental context.

This example graph is one comparing the client's default and optimized behavior. All labeled items will not be present on every graph.

Circle number 1 is the point in the graph where resource utilization starts. The starting point for each performance metric is similar for the default and optimized behavior of the client because both start with the amount of messages indicated at circle 4. If the amount of messages in the Inbox is increased or decreased at the start, the performance metrics' starting point values on the y-axis also would decrease or increase correspondingly. However, 2800 initial messages in the mailbox was maintained throughout the experiments.

Circle number 2 shows the end of the experiment after manipulating all 2800 messages for default behavior of the client. The length could be different for the various clients.

Circle number 3 shows the end of the experiment after manipulating 2800 messages for optimized behavior of the client. The length could be different for all clients. This point indicates the y-axis value when the Inbox is empty.

Circle number 4 indicates the amount of messages in all mailboxes at starting point of the experiment. This is consistent for all clients under both the default and optimized scenarios.

Circle number 5 shows the amount of messages in all mailboxes for the default behavior of clients at the end of the individual experiment. Note that the Trash box is implemented only by Thunderbird and Sylpheed. The amount of messages in the Inbox is different for each client (explained later). The 2800 value in the model graph is an example.

Circle number 6 shows the amount of messages in all mailboxes for the optimized behavior of clients at the end of the individual experiment. The actual data values will again vary by client.

Circle number 7 shows the slope of the graph for the optimized behavior of clients as the amount of messages in Inbox decreases by 80 and the amount of messages in other message folder increases by 10 for every 80 message group manipulation. In most cases, the best function that fits the graph is a third degree polynomial. The best fit function was selected based on R-squared value. The function equation and R-squared values are included in the graphs. From this data, it is possible to understand the slope increase or decrease as the amount of messages in Inbox decreases or increases.

Circle number 8 shows the slope of the graph for default behavior of clients when the amount of messages did not decrease in the Inbox (for most clients except Sylpheed and Opera). The amount of messages in other message folders increases by 10 as each group of messages is manipulated. In most cases, the best function that fits the graph was closer to linear.

Circle number 9 shows an arrow indicating the the time elapsed in seconds required to complete the experiment.

Circle number 10 is an important point for the visualization discussion. By projecting from circle number 3, it is possible to determine the the performance value of the y-axis when the Inbox is empty.

Circle number 11 visualizes what would happen if the user received 80 messages at a time and manipulated them under the optimized behavior. This best case could not be achieved in the real application since there would not be any messages in Inbox at the end of 80 message group manipulation, merely an increase of 10 messages in each of the other message folders.

Circle number 12 visualizes the same process for the default behavior, indicating what would happen if the user received 80 messages at a time and manipulated them. It is an adaptation of the optimized behavior slope (circle number 7) because they have similar trends except that this one moves from high to low since its initial point is when the value in x-axis is zero and the Inbox is empty. This indicates that the client starts with equal performance resource usage to the visualized optimized behavior (circle number 10) with a slope trend similar to the optimized behavior.

Circle number 13 is the limit of time for the optimized behavior for an increased or decreased amount of messages in the Inbox in the experiment.

Circle number 14 is the limit of time the for default behavior for the constant amount of messages in the Inbox in the experiment.

Thus, the model demonstrates a decreased in performance as the amount of messages accumulate in Inbox and other message folders for the default behavior and a constant resource demand with a very slight slope decline as the experiment runs for the optimized behavior.

4.1.2 Statistical Analysis

The graphs are plotted from the per-value means of 35 measurement replications. Error bars are plotted from standard deviation values. The wider error bars indicate the potential error or degree of uncertainty. The equations displayed on various graphs are the function that best fits the graph. The thin on each the graph is the trend line. The R-squared value shown is the square of

the Pearson product moment correlation coefficient through data points. This value indicates the fitness of the selected function to the actual values in graph.

Most client's default behavior graphs showed a linear function trend line. Graphs from Mulberry, Thunderbird and Outlook are good examples for this. Sylpheed graphs behave the same for its both behavior graphs. For client's optimized behavior, some showed an exponential function but the slopes for this function were not significant because the slope for theses graphs were not steep as normal exponential function graphs. Most graphs were fitted with polynomial function of varying degree from 2 to 5. The equations with R-squared values indicated on the graphs provide evidence as to how well the function fits to actual graph. From these equations, one can easily see the trend by which the performance mertics decreased or increased.

4.2 Mulberry

4.2.1 Mulberry Default Behavior: 3.4 Kbyte Message Size

The following graphs show Mulberry's default behavior for the 3.4 Kbyte message size. Figures 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7 show Mulberry's default behavior server side resource usage for disk I/O read and write performance, network bandwidth usage for packets received and sent, and system interrupts and context switching. Figure 4.8 shows CPU time usage by user and system, and CPU idle and wait time. The graphs are plotted from per-point means of 35 replications.

Table 4.9 shows the calculated correlation coefficient values for this client's default behavior for all measured performance metrics.

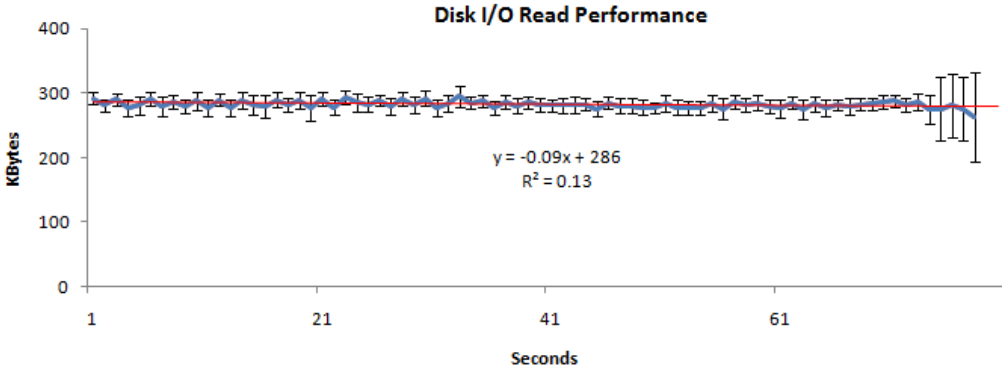

Figure 4.2: Mulberry Default Behavior Disk I/O Read Performance: 3.4 Kbyte Message Size

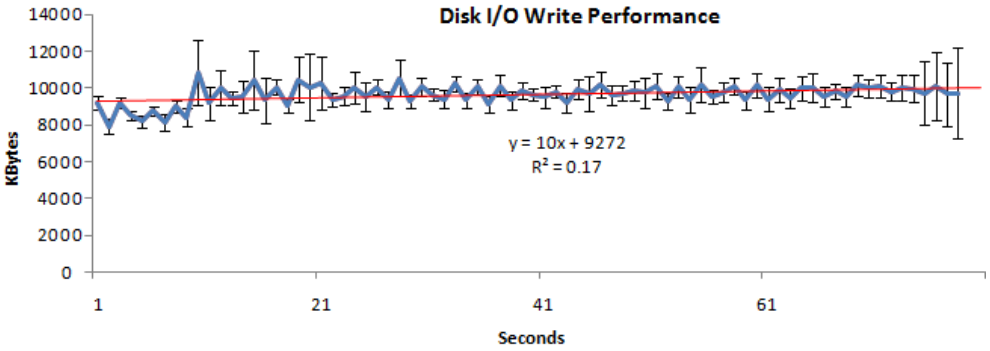

Figure 4.3: Mulberry Default Behavior Disk I/O Write Performance: 3.4 Kbyte Message Size

Figures 4.2, 4.4, 4.5 and 4.7 show a very slight decline in disk I/O read, both received and sent network bandwidth usage and system interrupt performance. The trend analysis equations for these graphs indicate decreased performance

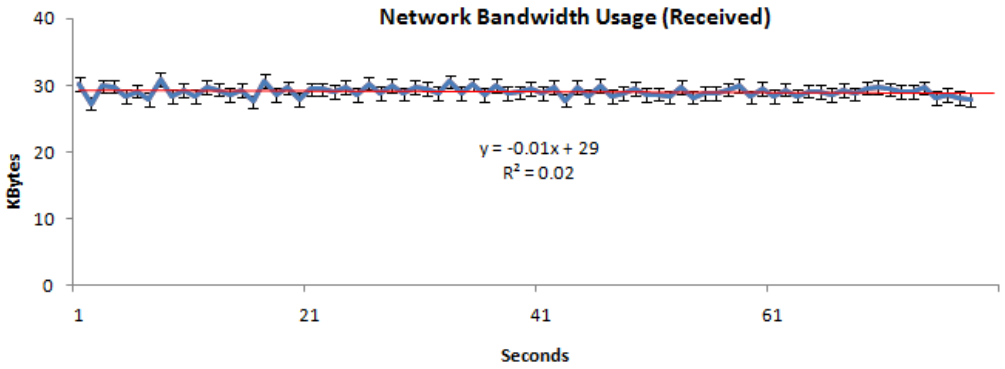

Figure 4.4: Mulberry Default Behavior Network Bandwidth Received Performance: 3.4 Kbyte Message Size

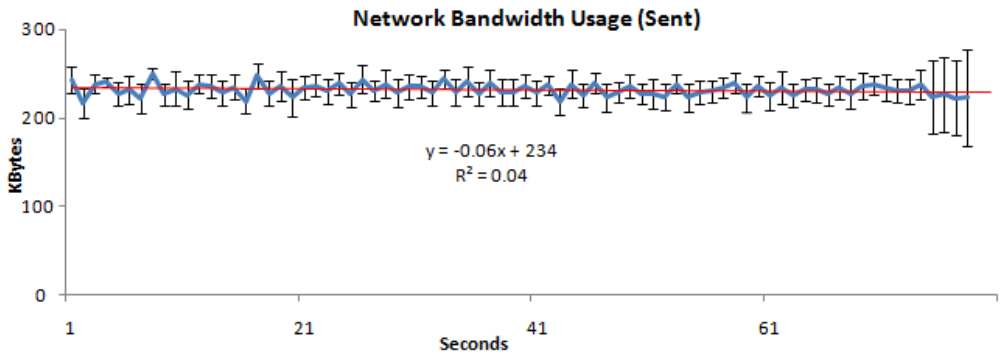

Figure 4.5: Mulberry Default Behavior Network Bandwidth Sent Performance: 3.4 Kbyte Message Size

with time, yielding slopes of -0.09, -0.01, and -2 projected for disk I/O read and network bandwidth usage (both received and sent) and context switch performance.

Since the nature of the data allowed fluctuation in individual mean values, the R-squared value for all trend lines confirms this and it is difficult to conclude that the regression line represents the true relationship of performance metrics across time. The error bars are graphed from calculated standard deviation for each plotted value in each graph. As one can see, the deviation of individual values from mean are reasonable given the nature of the experiment.

This property is demonstrated due to the small number of message increase in message folders other than Inbox. In every message group manipulation, the number of messages increased by 10 in each message folder. In this client's behavior, the amount of messages in Inbox is unchanged because Mulberry by default does not permanently remove deleted messages. Rather it marks them as deleted in the client GUI to make this property visible to users and sets the Deleted flag on each deleted message on server side. Since the number of messages in Inbox, which was 2800 initially, stayed constant through the completion of the experiment, the decrease in performance with regards to the above mentioned parameters is insignificant. However, if the number of messages moved were put in a single folder or the amount of messages moved to each folder were high, the slope in the decreasing trend of the metrics could have been more steep.

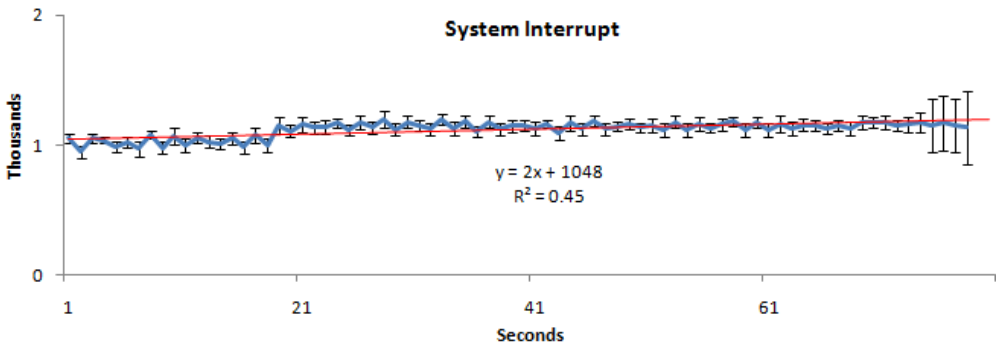

Figure 4.6: Mulberry Default Behavior System Interrupts: 3.4 Kbyte Message Size

In contrast, the slope corresponding to the performance of disk I/O write and system interrupts increased by 10 and 2 respectively (see 4.3 and 4.6). This could be due to a very low start at the beginning of the graph and an immediate increase after 10 seconds. This property could not be associated with the actual client's behavior because the simulation script for one group of 80 messages multiple time during the life span of a single replication experiment. Therefore, it is difficult to rationalize this result from the client behavior point of view.

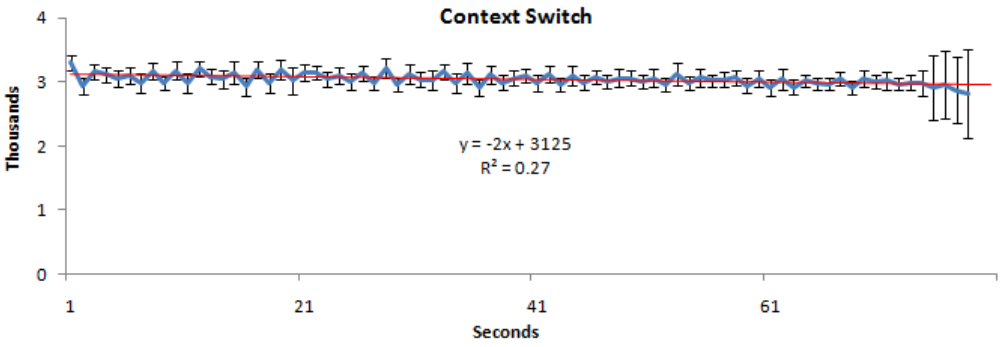

Figure 4.7: Mulberry Default Behavior Context Switches: 3.4 Kbyte Message Size

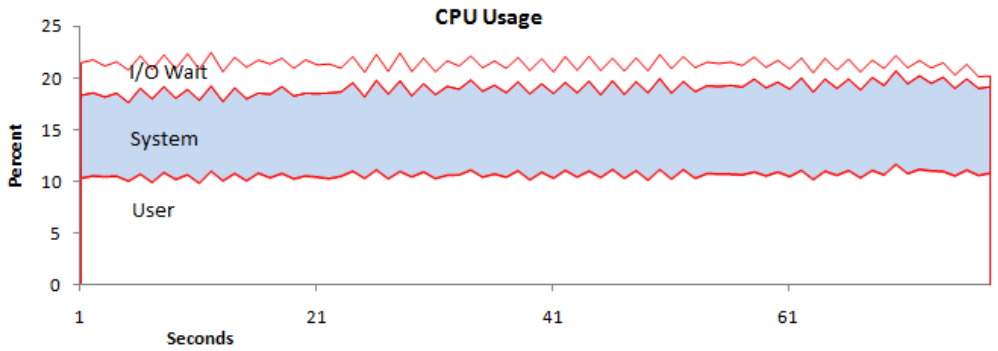

Figure 4.8: Mulberry Default Behavior CPU Usage Performance: 3.4 Kbyte Message Size

As shown in Figure 4.8, user and system CPU time showed slight increases over the experimental run. This is due to the slightly increased requirements for disk I/O. However, the slightly decreased slope seen by CPU time by I/O wait is probably the reflection of the corresponding small decrease in disk I/O read.

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	0.26	1								
recv	0.56	0.46	1							
send	0.50	0.41	***0.99***	1						
int	0.11	0.73	0.41	0.34	1					
csw	0.66	0.33	0.79	0.80	0.07	1				
usr	0.16	0.28	0.19	0.15	0.39	-0.07	1			
sys	-0.01	0.38	0.17	0.12	0.61	-0.19	0.74	1		
idl	-0.37	-0.13	-0.32	-0.31	-0.02	-0.36	-0.73	-0.46	1	
wai	0.32	-0.29	0.12	0.17	-0.65	0.57	-0.31	-0.61	-0.36	1

Figure 4.9: Correlation Between Performance Metrics: Mulberry Default Behavior, 3.4 Kbyte Message Size

In Figure 4.9, the read and writ columns represent disk I/O read and write performance, respectively; recv and send represent network bandwidth usage for packets received and sent, respectively; int and csw represent system interrupts and context switches; usr, sys and wai represent CPU time usage by user time, and system time and I/O wait time. The correlation for any pair performance metrics is located at the row and column intersection for those two variables.

The total amount of messages in the Maildir directory for each message folder increased by 350 at the end of each experiment. The total amount of messages in the four folders has become 1400, and this increases the overall number of messages to 4200. Figure 5.9 indicates that disk usage is increased by half of the total group of messages manipulated: 1400.

The shaded correlation coefficient value (9.9) in Figure 4.9 shows that there was a strong relationship between network bandwidth usage for packets received and sent. The IMAP protocol itself does not enforce any strong relationship between commands received and sent since different commands received from clients request different tasks, and the packet size for the command requested does not correlate with that for the response command in most cases. For example, the response packet for a FETCH command issued from the client to fetch the BODY part of a message can be much smaller than the response packet size for one that encapsulates the body part of the message. However, the simulation methodology as implemented enforces strong relationship between the network bandwidth usage in both directions, and the strong correlation relationship showed in Figure 4.9 could be due to the simu-

lation implementation. This issue will be discussed in detail in the Discussion chapter.

4.2.2 Mulberry Optimized Behavior for 3.4 Kbyte Message Size

The following plotted graphs shows Mulberry's optimized behavior for the 3.4 Kbyte message size. The graphs plot the same data metrics as those in the preceding section. Figure 4.10, 4.11, 4.12, 4.13, 4.14 and 4.15 shows Mulberry's optimized behavior server side resource demand for disk I/O read and write performance, network bandwidth usage for packets received and sent, system interrupts and context switching, and CPU usage (respectively). Figure 4.17 shows the correlation between the performance metrics chosen for this thesis experiment.

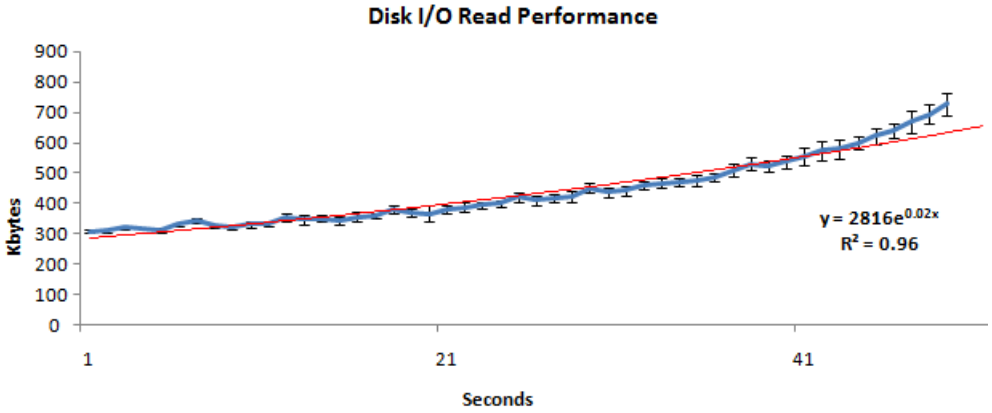

Figure 4.10: Mulberry Optimized Behavior Disk I/O Read Performance: 3.4 Kbyte Message Size

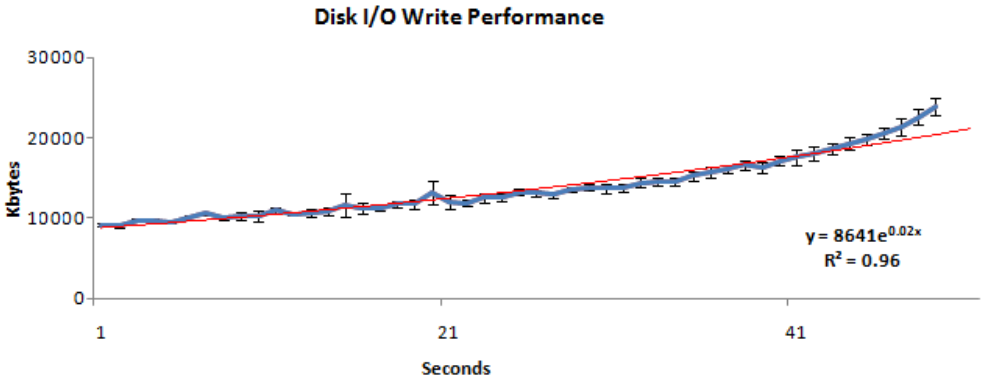

Figure 4.11: Mulberry Optimized Behavior Disk I/O Write Performance: 3.4 Kbyte Message Size

In the optimized behavior mode for Mulberry, all experimental parameters except CPU usage showed an increase in performance exponentially, as shown in Figures 4.10, 4.11, 4.12, 4.13, 4.14 and 4.15. While an exponential function is the best fit, the computed function is not at all steep.

As for the default behavior, the client puts 10 messages in each message folder

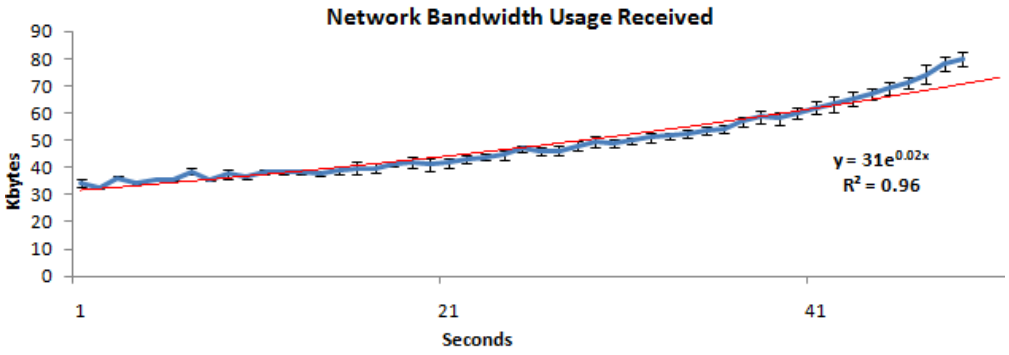

Figure 4.12: Mulberry Optimized Behavior Network Bandwiddth Usage for Received Packets: 3.4 Kbyte Message Sizer

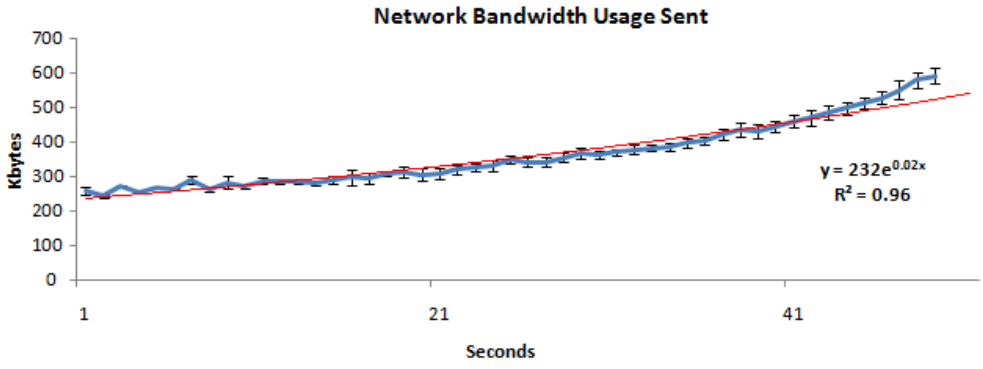

Figure 4.13: Mulberry Optimized Behavior Network Bandwiddth Usage for Sent Packets: 3.4 Kbyte Message Size

under each message group manipulation. For every 80 messages in a group, 40 of them are moved to four message folders, and the remaining 40 are permanently removed from the Inbox. This means that there is a decrease of 80 messages from Inbox and and increase of 10 messages in each four message folders with each message group manipulation. The slope increase in the performance parameters is due to this amount of message change in the Inbox and message folders. Thus, the optimized behavior minimizes the amount of messages in the mail directory, which in turn increases the performance when changing the status of manipulated messages and renaming them after scanning the directory.

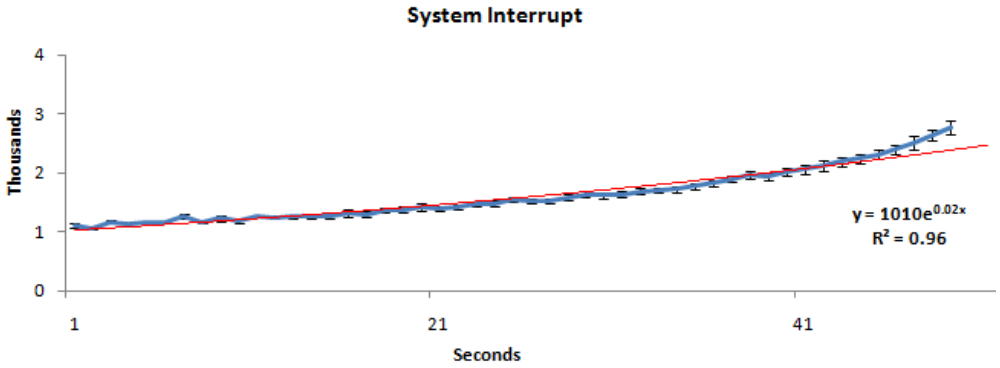

Figure 4.14: Mulberry Optimized Behavior System Interrupts: 3.4 Kbyte Message Size

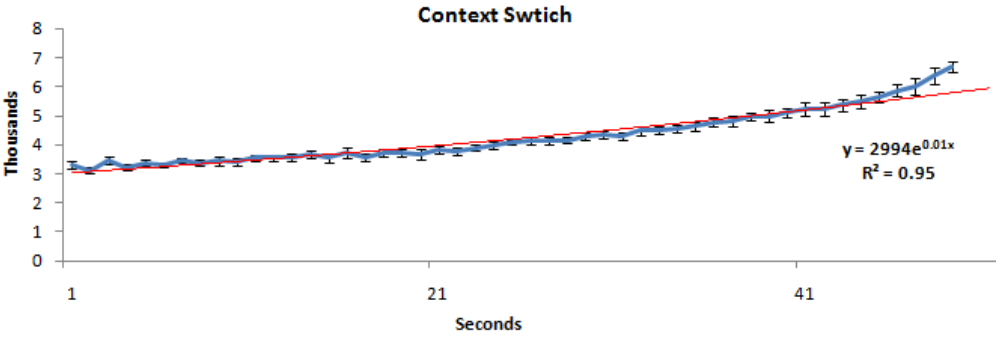

Figure 4.15: Mulberry Optimized Behavior Context Switches: 3.4 Kbyte Message Size

The CPU time usage (system and user) showed a slight decrease (see 4.16). This is due to increased performance for the other performance metrics. In contrast, I/O wait time showed an increase since there was a significant increase in disk I/O operations.

The correlation coefficient values in Table fig:mulberryoptimized2kcorrelation confirms the direct relationship between performance metrics. The very strong correlations between variables with values greater than 0.95 (or less than -0.95) are shaded with the dark color, and correlations with absolute values greater

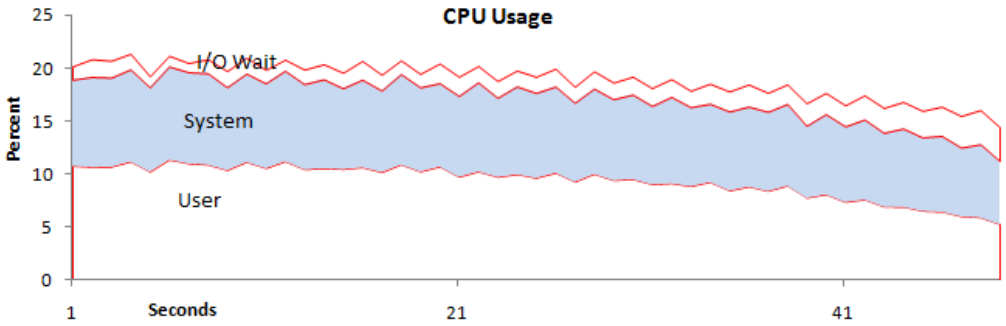

Figure 4.16: Mulberry Optimized Behavior CPU Usage: 3.4 Kbyte Message Size

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	1.00	1								
recv	1.00	1.00	1							
send	1.00	1.00	1.00	1						
int	1.00	1.00	1.00	1.00	1					
csw	1.00	0.99	1.00	1.00	1.00	1				
usr	-0.98	-0.97	-0.98	-0.98	-0.98	-0.98	1			
sys	-0.84	-0.85	-0.85	-0.85	-0.85	-0.86	0.90	1		
idl	0.93	0.92	0.93	0.93	0.93	0.93	-0.97	-0.96	1	
wai	0.91	0.92	0.91	0.91	0.92	0.91	-0.91	-0.82	0.84	1

Figure 4.17: Mulberry Optimized Behavior Correlation Coefficients

than or equal to 0.9 less than 0.95 (or greater than -0.95) shaded with the light color. CPU time by system and user showed inverse relationships with the other performance metrics as we have seen in the graphs.

4.2.3 Mulberry's Default vs Optimized Behavior: 3.4 Kbyte Message Size

The graphs in this subsection compare this client's default and optimized behavior for disk I/O and network bandwidth usage. The first two graphs are combined disk I/O read and write performance measured for Mulberry's default and optimized behaviors, and the last two are for network bandwidth usage when packets were received from and sent to the IMAP server.

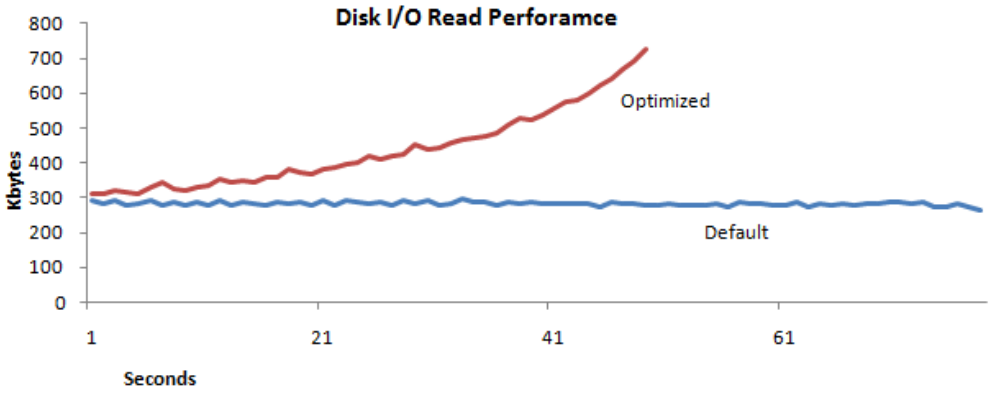

Figure 4.18: Mulberry Default vs Optimized Disk I/O Read Performance: 3.4 Kbyte Message Size

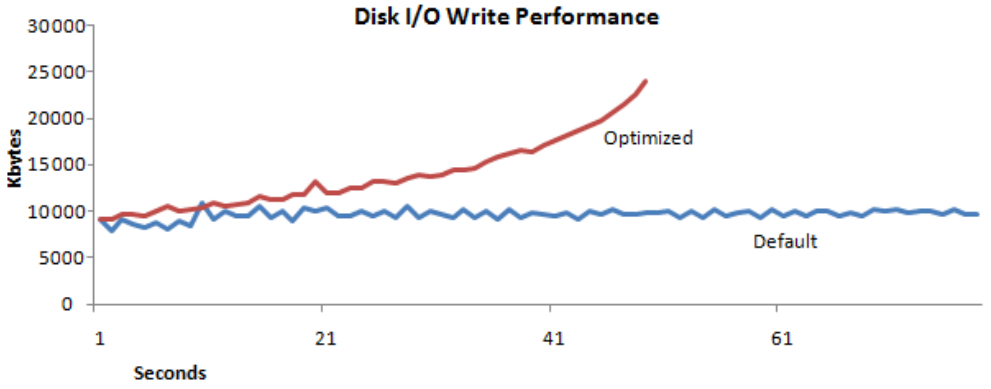

Figure 4.19: Mulberry Default vs Optimized Disk I/O Write Performance: 3.4 Kbyte Message Size

Figures 4.18,4.19,4.20, 4.21 shows the comparison between Mulberry's default and optimized behavior resource consumption while running the experiment.

The major difference between the default and optimized behavior of the Mulberry client is that the default behavior shows a linear relationship between performance metrics because the performance metrics had approximately constant value across the lifespan of the experiment. As it has been explained, the slight decline could be due to a small increase in the number of messages in

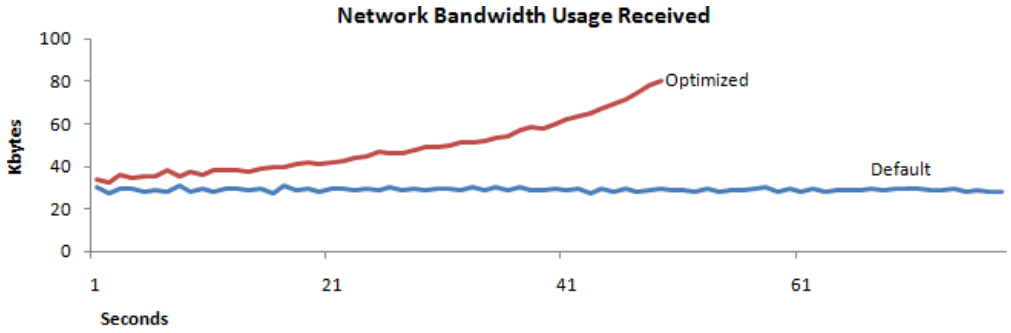

Figure 4.20: Mulberry Default vs Optimized Network Bandwidth Usage, Packets Received: 3.4 Kbyte Message Size

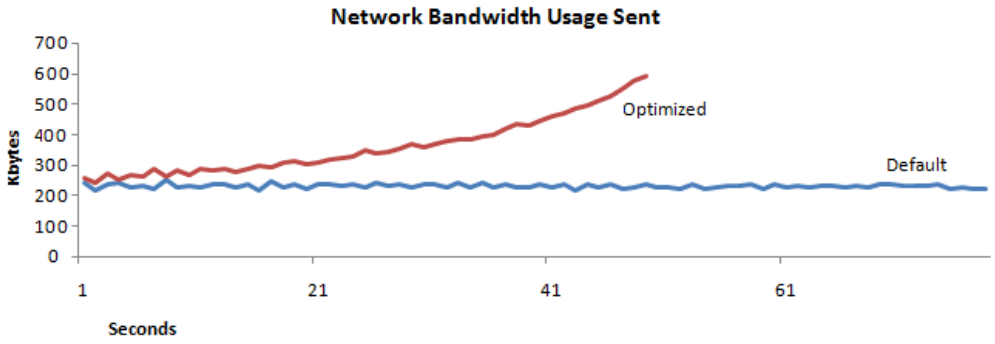

Figure 4.21: Mulberry Default vs Optimized Network Bandwidth Usage, Packets Sent: 3.4 Kbyte Message Size

four message folders other than the Inbox. However, in the optimized case, an exponential property of graph was demonstrated due to the decline in the amount of messages in the Inbox and the corresponding increases in individually folders for each message group manipulation round. Although there was an increase in number of messages in the other message folders, the decrease in the amount of messages in the Inbox seems to have dominated the performance differences, and the exponential performance increase was observed.

4.2.4 3.4 vs 76 Kbyte Message Size Comparison for Mulberry Default Behavior

This section presents the plotted results from the 76 Kbyte message experiments using the Mulberry default behavior, presented in comparison with those for the 3.4 Kbyte messages size. Figure 4.22, 4.23, 4.24, 4.25, and 4.26 present the 75 Kbyte message size results (in comparison with the corresponding results for the Kbyte message size) for disk I/O read and write performance, incoming and outgoing network bandwidth, and I/O wait CPU time (respectively).

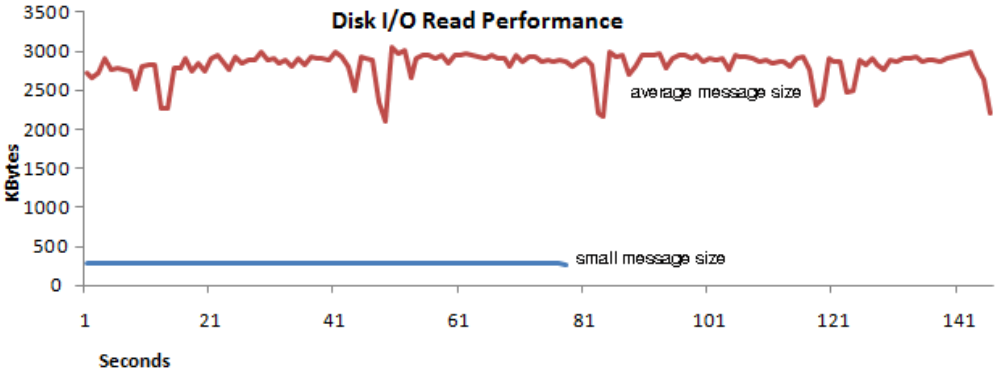

Figure 4.22: Mulberry Default Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

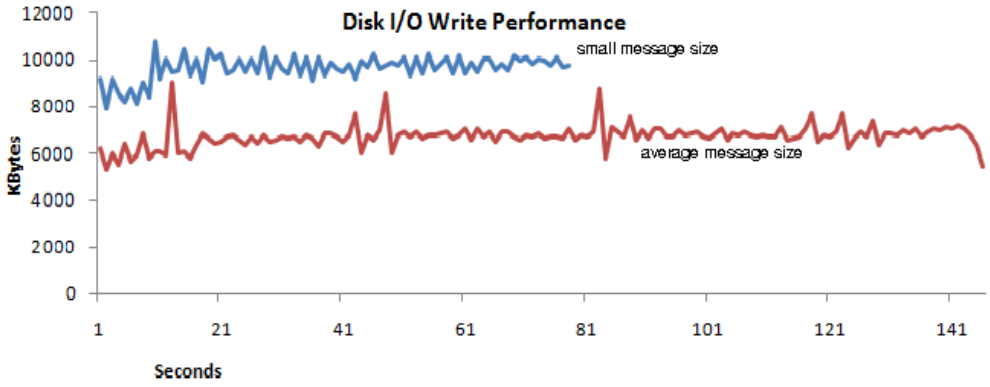

Figure 4.23: Mulberry Default Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

From the preceding graphs, operations involving the 76 Kbyte message size consumed significant resources in disk I/O read and outgoing compared to the smaller message size. In contrast, the 3.4 Kbyte message size consumed higher amounts of disk I/O write and incoming network bandwidth resources. This is a clear indication that as the message size increases, the disk I/O read resource requirement also increases. The reason the disk I/O write rate is lower

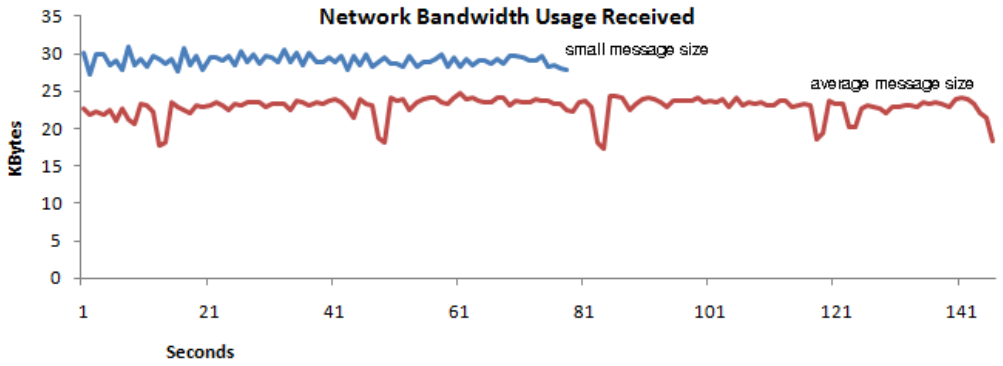

Figure 4.24: Mulberry DEfault Behavior Network Bandwidth Usage (Packets Received) Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

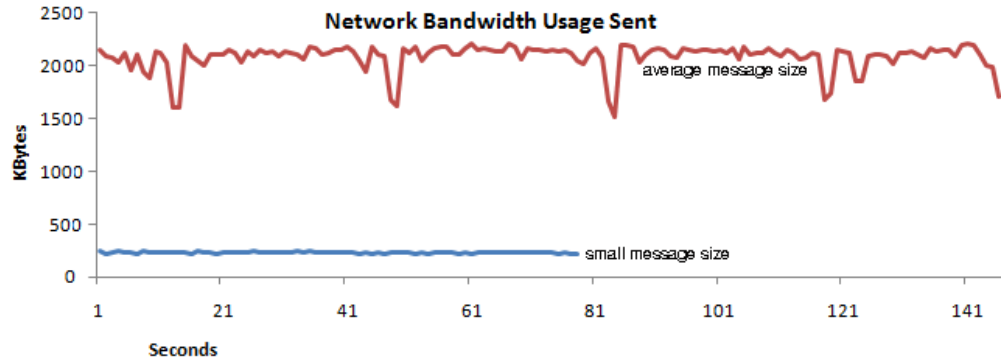

Figure 4.25: Mulberry Default Behavior Network Bandwidth Usage (Packets Sent) Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

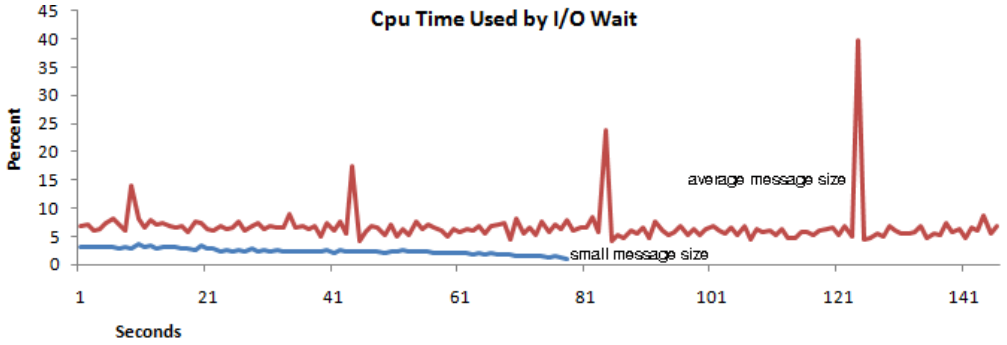

Figure 4.26: Mulberry Default Behavior Disk I/O Wait CPU Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

could be due to the slow network bandwidth usage for packets received because of the high disk I/O read that slows down the overall system performance. The CPU time used by I/O wait is a good indication that this is the case.

4.2.5 3.4 vs 76 Kbyte Message Size Comparison for Mulberry Optimized Behavior

This section presents the plotted graph results from comparisons between Mulberry's optimized behavior for the 3.4 Kbyte and 76 Kbyte messages sizes. Figures 4.27, 4.28, 4.29, 4.30, and 4.31 present the 75 Kbyte message size results (in comparison with the corresponding results for the Kbyte message size) for disk I/O read and write performance, incoming and outgoing network bandwidth, and I/O wait CPU time (respectively).

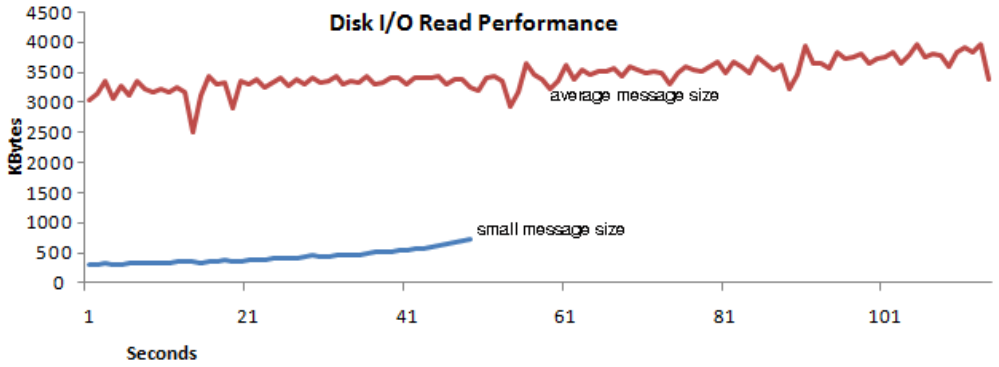

Figure 4.27: Mulberry Optimized Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

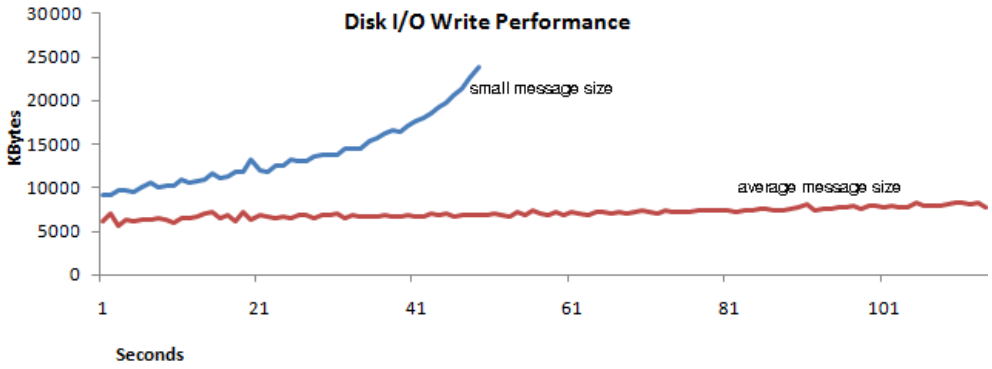

Figure 4.28: Mulberry Optimized Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

From the preceding graphs, the 76 Kbyte message size consumed significant resources in disk I/O read and network bandwidth usage for packets sent from IMAP server, as was the case for the default behavior. The 3.4 Kbytes message size consumed higher amounts of disk I/O write and network bandwidth usage for packets received by the server. The only difference from the default behavior size discussed in the preceding subsection is that the 76 Kbyte message size graph does not follow the approximate exponential function form as for the 3.4 Kbyte message size. This could be due to the high CPU I/O wait time shown in Figure 4.31, which slows the network bandwidth usage

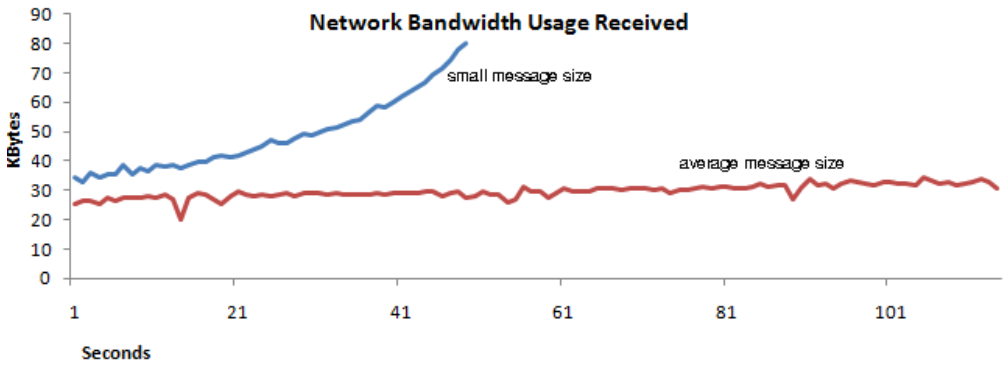

Figure 4.29: Mulberry Optimized Behavior Network Bandwidth Usage (Packets Received) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

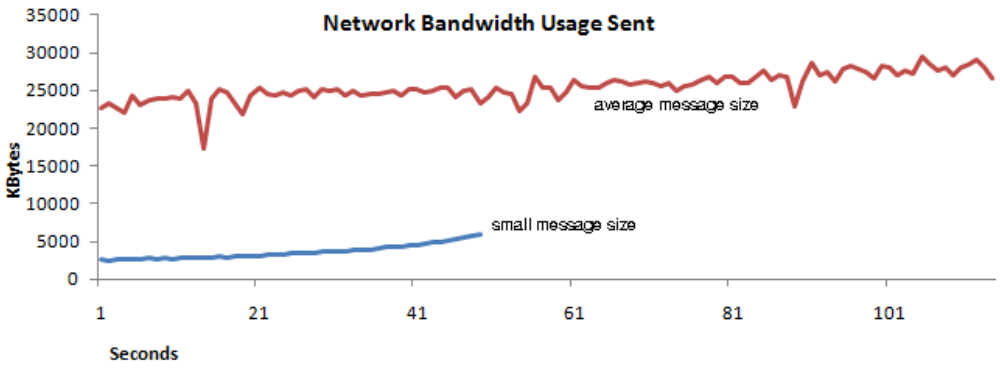

Figure 4.30: Mulberry Optimized Behavior Network Bandwidth Usage (Packets Sent) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

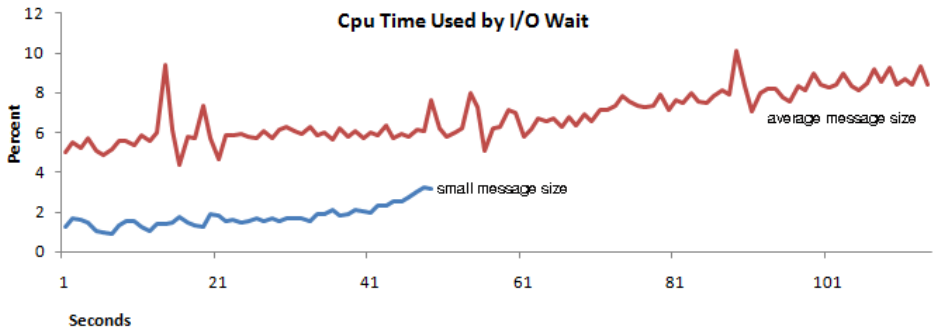

Figure 4.31: Mulberry Optimized Behavior Disk I/O Wait CPU Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

received due to the fact that the simulation script send commands only after the IMAP server has responded to previous requests.

4.3 Outlook

4.3.1 Outlook Default Behavior: 3.4 Kbyte Message Size

The following plotted graphs shows Outlook's default behavior for 3.4 Kbyte message size. The graphs are plotted from per-point means of 35 replications.

Figures 4.32, 4.33, 4.34, 4.35, 4.36 and 4.37 show Outlook's default behavior server side resource usage for disk I/O read and write performance, network bandwidth usage for packets received and sent, and system interrupts and context switching. Figure 4.38 shows CPU time usage by user and system and CPU wait time.

Table 4.39 shows the calculated correlation coefficient values for this client's default behavior all measured performance metrics.

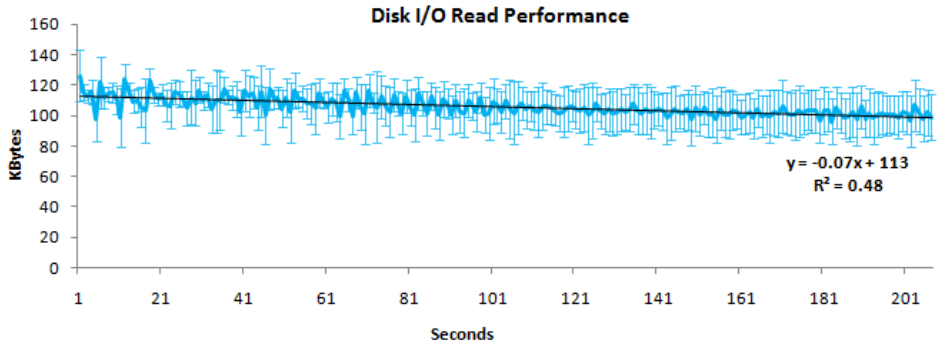

Figure 4.32: Outlook's Disk I/O Read Performance for 3.4 Kbyte Message Size Plotted for the Default Behaviour.

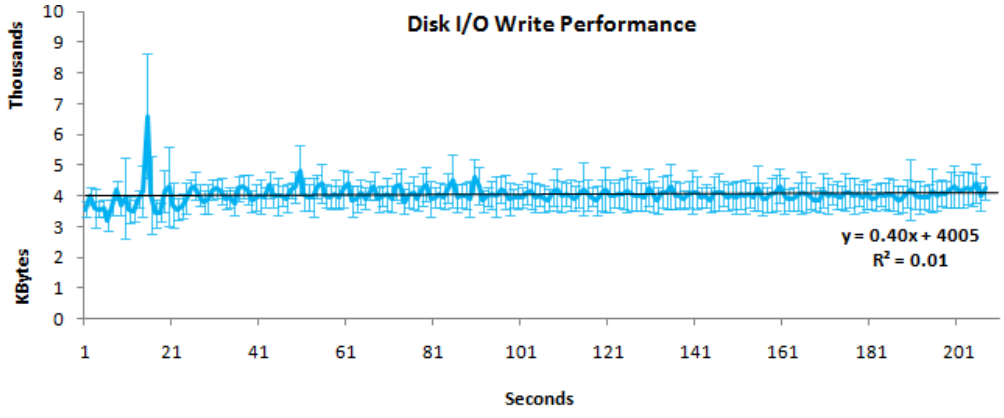

Figure 4.33: Outlook's Disk I/O Write Performance for 3.4 Kbyte Message Size Plotted for the Default Behaviour

Outlook has an IMAP protocol implementation very similar to Mulberry's with regards to managing deleted messages. Like Mulberry, it does not copy

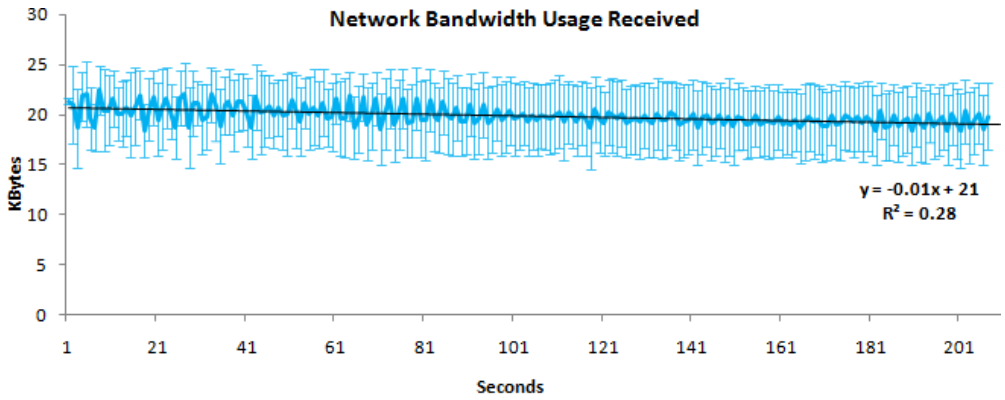

Figure 4.34: Outlook's Network Bandwidth Received Performance for 3.4 Kbyte Message Size Plotted for the Default Behaviour

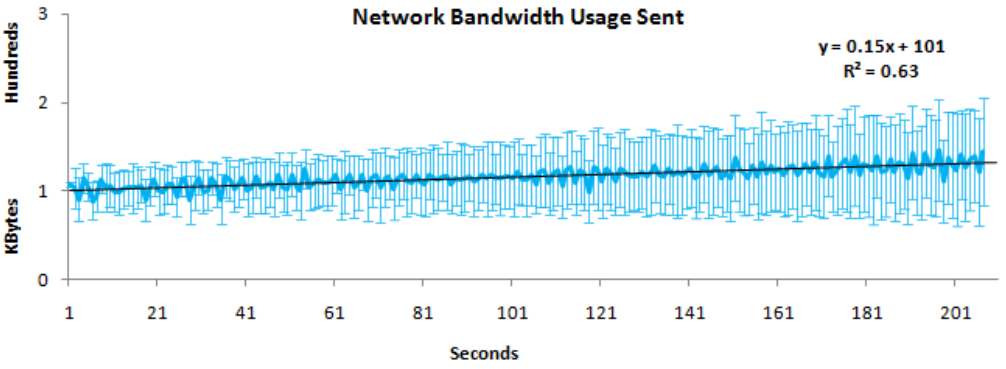

Figure 4.35: Outlook's Network Bandwidth Sent Performance for 3.4 Kbyte Message Size Plotted for the Default Behaviour

deleted messages to the Trash box, and messages are visible on the GUI marked as deleted for users. If the user does not purge the deleted messages, they accumulate in the Inbox.

The slight decline in disk I/O read, network bandwidth usage slope for packets received, and system context switching, -0.07, -0.01, -0.091 respectively, shows the same trends as was previously discussed for Mulberry for the same reasons: due to the increased amount of messages in message folders.

The slope in performance of disk I/O write, network bandwidth usage for packets sent and system interrupts showed a slight increase by 0.4, 0.15 and 0.08 respectively (see 4.33, 4.35 and 4.6). The R-squared values are very low except for the outgoing network bandwidth usage and system context switches, and it is difficult to say that regression line represents the true relationship of performance metrics across time.

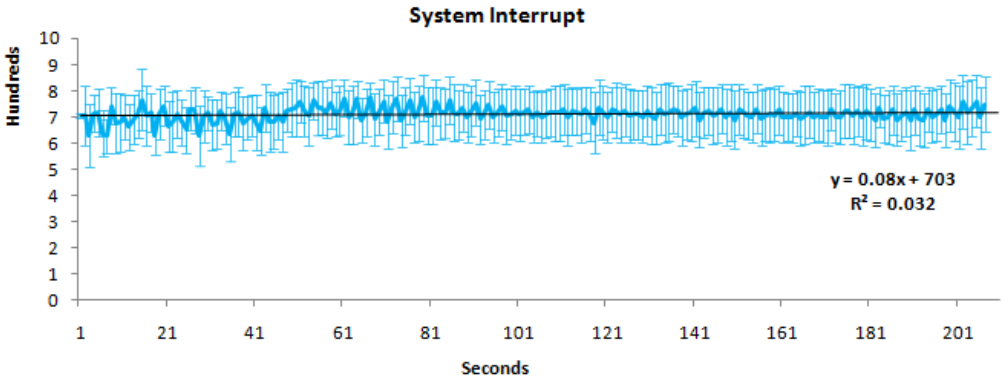

Figure 4.36: Outlook's System Interrupts for 3.4 Kbyte Message Size Plotted for the Default Behaviour

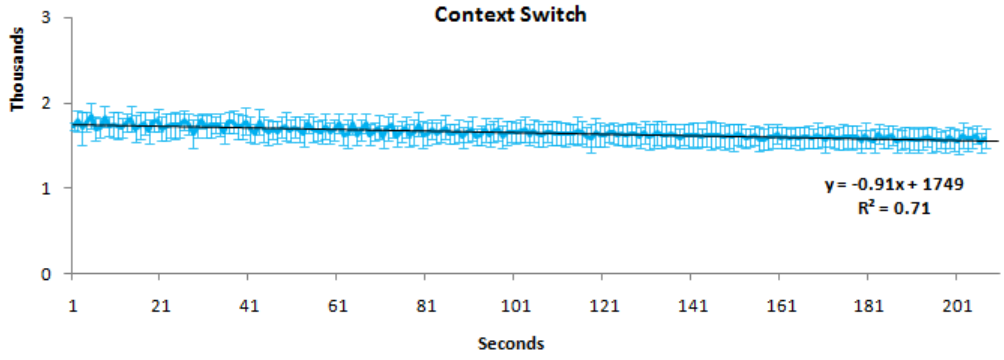

Figure 4.37: Outlook's Context Switch for 3.4 Kbyte Message Size Plotted for the Default Behaviour

CPU time usage by system and user showed very slight increases. This is due to the slight increased performance in disk I/O. The decreased percentage in CPU time for I/O wait is the reflection of the slight increase in disk I/O write.

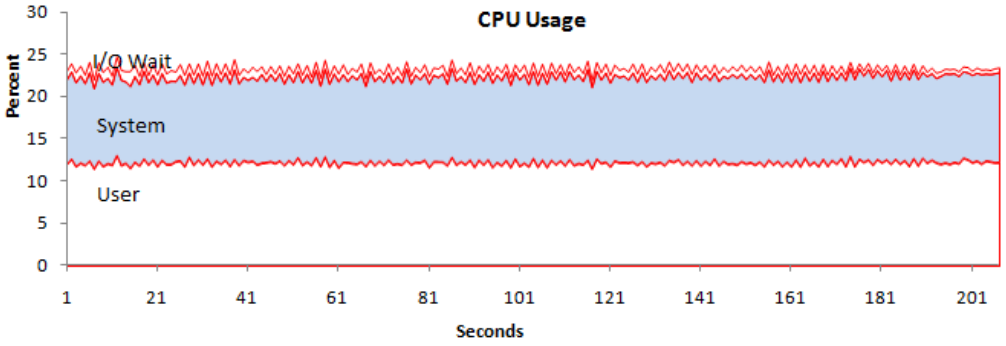

Figure 4.38: Outlook's CPU Usage Performance for 3.4 Kbyte Message Size Plotted for the Default Behaviour

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	-0.14	1								
recv	0.07	0.09	1							
send	-0.64	0.19	-0.03	1						
int	-0.41	0.53	0.58	0.54	1					
csw	0.43	0.08	0.80	-0.48	0.18	1				
usr	0.06	-0.06	-0.06	0.04	-0.06	-0.06	1			
sys	-0.17	-0.02	-0.23	0.35	0.05	-0.35	0.59	1		
idl	-0.11	0.00	0.06	-0.02	0.05	0.02	-0.89	-0.80	1	
wai	0.53	0.10	0.31	-0.61	-0.17	0.62	0.01	-0.22	-0.21	1

Figure 4.39: Outlook Correlation Matrix Between Performance Metrics

Like Mulberry, the total amount of messages in the directory for each message folder increased by 350 over the course of the experiment (see Figure 5.9). Since Outlook handles deleted messages similarly to Mulberry, the consequences of this behavior is the same as for that client (as discussed previously).

Figure 4.39 shows that there was no strong relationship between performance metrics. This is convincing since the increase or decrease trends seen were not significant enough to be observed in a correlation coefficient.

4.3.2 Outlook Optimized Behavior for 3.4 Kbyte Message Size

The following graphs shows Outlook's optimized behavior for 3.4 Kbyte message size. Figures 4.40, 4.41, 4.42, 4.43, 4.44 and 4.45 shows Outlook's optimized behavior server side resource demand for disk I/O read and write performance, network bandwidth usage for packets received and sent, system interrupts and context switching, respectively. Figure 4.46 shows the CPU time usage by user and system as well as I/O wait time.

Figure 4.47 shows the correlation between the performance metrics chosen for this thesis experiment.

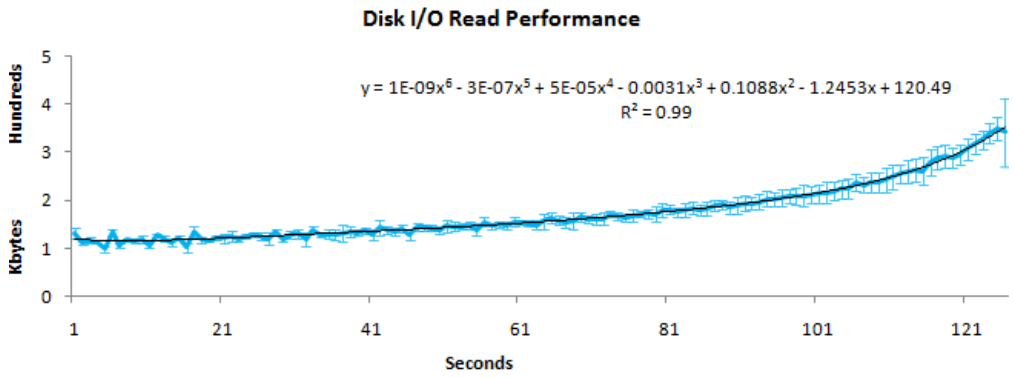

Figure 4.40: Outlook's Disk I/O Read Performance for 3.4 Kbyte Message Size Plotted for the Client's Optimized Behavior

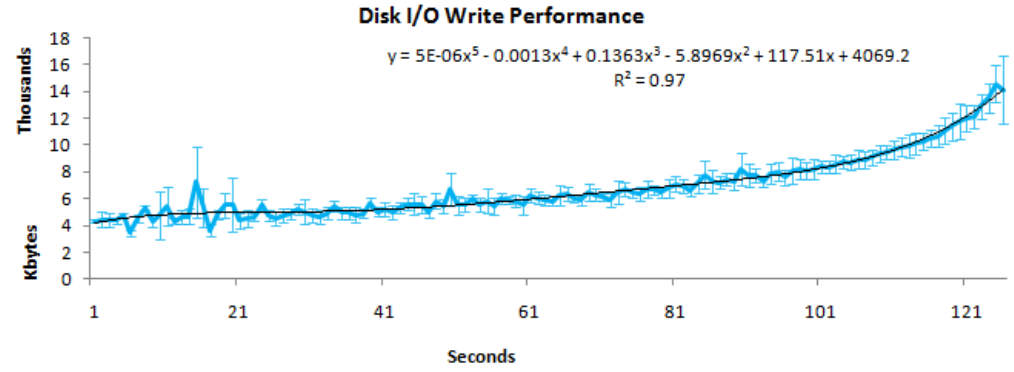

Figure 4.41: Outlook's Disk I/O Write Performance for 3.4 Kbyte Message Size Plotted for the Client's Optimized Behavior

The major optimization achieved for Outlook is enabling purging of items when switching folders while the user is online. This feature is implemented by CLOSE command issued from the client during folder switching. This means the performance issue in the default behavior is still present as long as the user does not switch between folders or use a one time purging command. In fact, allowing a user the option of purging items when switching

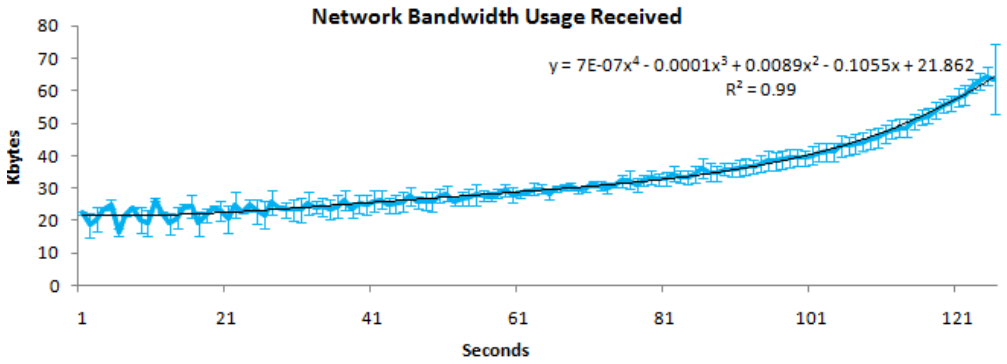

Figure 4.42: Outlook's Network Bandwiddth Usage for Received Packets for 3.4 Kbyte Message Size Plotted for the Client's Optimized Behavior

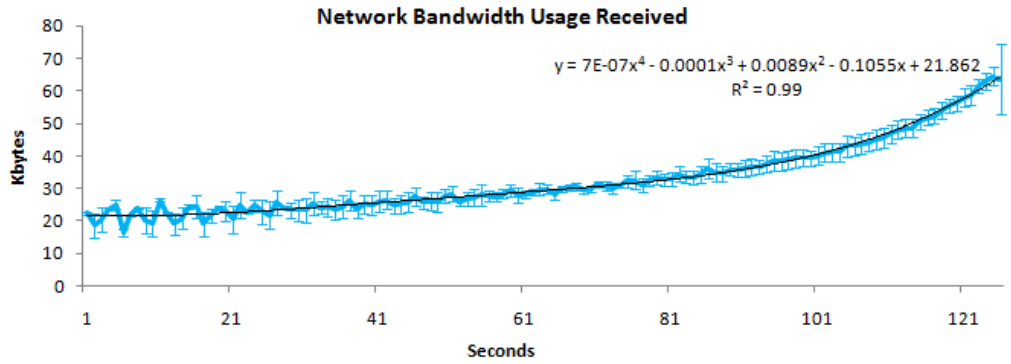

Figure 4.43: Outlook's Network Bandwiddth Usage for Sent Packets for 3.4 Kbyte Message Size Plotted for the Client's Optimized Behavior

between folders is undesirable because a user can switch between several message folders repeatedly, causing any accidentally deleted messages to be lost immediately.

Outlook's benefit from optimization is similar to Mulberry's in that there is a decrease of 80 messages at the end of each message group manipulation cycle. The exponential increase in performance parameters is due to this amount of message change in the Inbox and message folders. Thus, this minimizes the amount of messages in the current directory which in turn increases the performance when changing the status of manipulated messages and renaming them after scanning the directory.

For these reasons, in the optimized behavior of Outlook, all experimental parameters except CPU usage showed a exponential increase in performance although not a steep one.

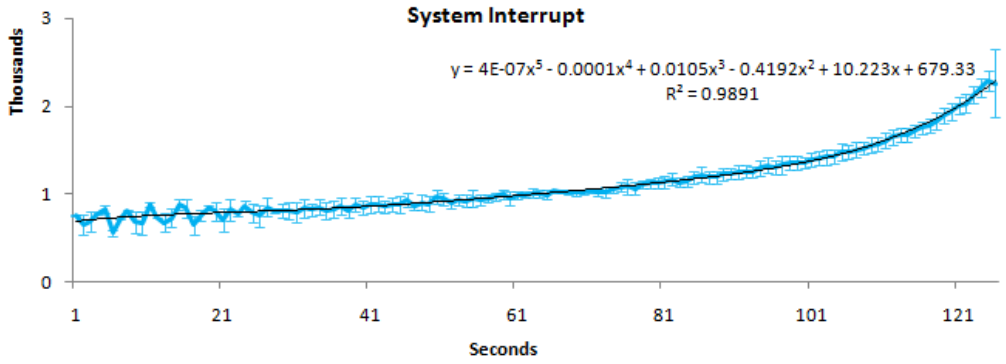

Figure 4.44: Outlook's System Interrupt for 3.4 Kbyte Message Size Plotted for the Client's Optimized Behavior

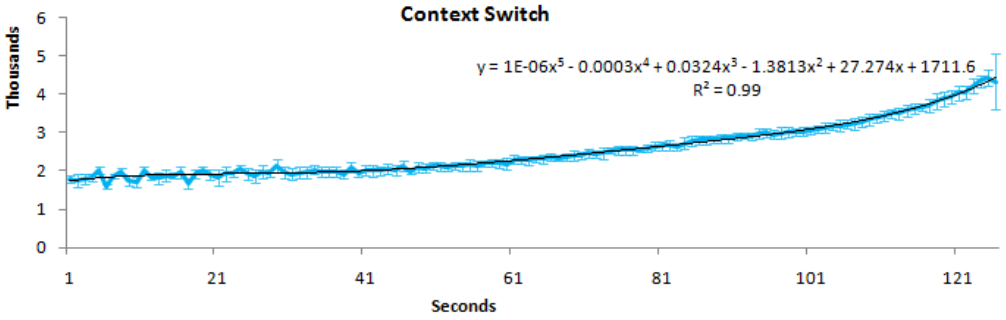

Figure 4.45: Outlook's Context Switch for 3.4 Kbyte Message Size Plotted for the Client's Optimized Behavior

CPU time usage, both system and user, decreased over time. This is due to smaller consumption of CPU time as the amount of messages decreased in the Inbox. This in turn increased the performance for disk I/O because the command request and response between client and server were fast. Thus, the

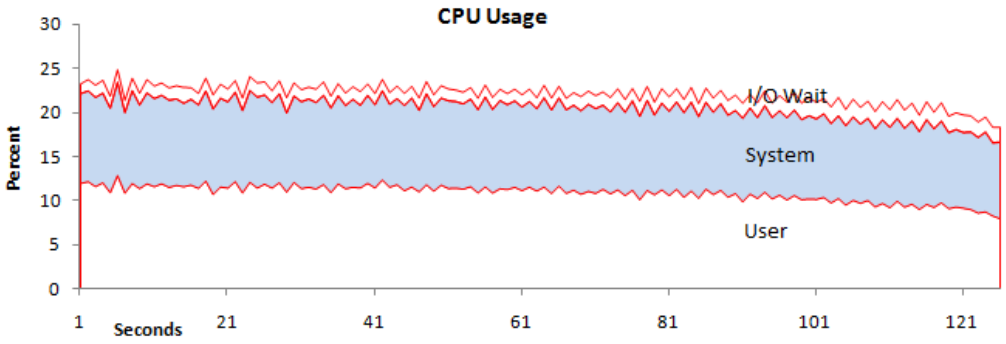

Figure 4.46: Outlook's CPU Usage for 3.4 Kbyte Message Size Plotted for the Client's Optimized Behavior

decreased slope seen by CPU time by I/O wait is the reflection of the increase in disk I/O write.

The strong correlation coefficient between disk I/O and network bandwidth usage is confirmed in Table 4.47. The table shows a negative correlation for CPU time used by user with both disk I/O and network bandwidth usage, and system interrupts and context switches. This reflects the resource requirements for each 80 messages group manipulation during the course of the experiment as the amount of messages decreased across time. The remaining performance metrics showed a significant strong correlation with one another. The increased trend in disk I/O is the reflection of high disk I/O as the amount of messages in the the Inbox decreases.

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	0.98	1								
recv	0.99	0.99	1							
send	0.98	0.99	1.00	1						
int	0.99	0.99	1.00	1.00	1					
csw	0.98	0.98	0.99	0.99	0.99	1				
usr	-0.90	-0.91	-0.92	-0.92	-0.92	-0.92	1			
sys	-0.72	-0.73	-0.74	-0.74	-0.74	-0.73	0.86	1		
idl	0.80	0.81	0.82	0.82	0.82	0.81	-0.95	-0.95	1	
wai	0.75	0.75	0.74	0.75	0.74	0.77	-0.71	-0.53	0.53	1

Figure 4.47: Outlook Optimized Behavior Correlation Coefficients

4.3.3 Outlook's Default vs Optimized Behavior for 3.4 Kbyte Message Size

In this subsection, the comparison between client's default and optimized behavior for disk I/O and network bandwidth usage is presented. The first two graphs are combined disk I/O read and write performance measured for Outlook's default and optimized behaviors. The remaining two are for network bandwidth usage when packets were received from and sent to the IMAP server.

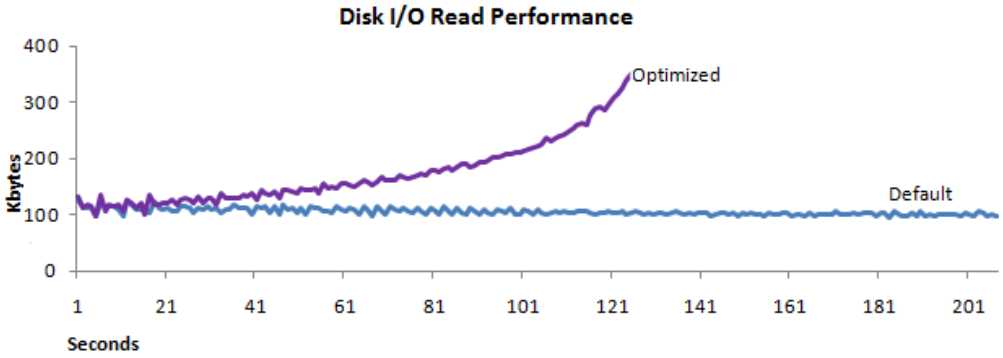

Figure 4.48: Outlook's Default vs Optimized Disk I/O Read Performance: 3.4 Kbyte Message Size

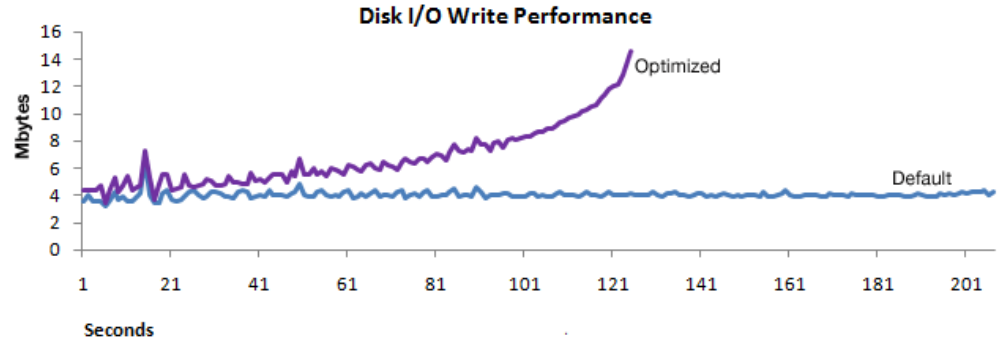

Figure 4.49: Outlook's Default vs Optimized Disk I/O Write Performance: 3.4 Kbyte Message Size

Once again, the difference in default and optimized behavior discussed for Mulberry are applicable to Outlook as well. This is shown in Figures 4.48,4.49,4.50, 4.51. As the figures show, the slope direction, and difference for default and optimized behaviors is significant and this is clearly due to the clearance of deleted messages at the end of each message group manipulation round.

The main difference between the default and optimized behavior of the Outlook client is the slope and direction of the graphs for resource consumption. The default behavior showed linear relationships between performance metrics because the performance metrics had approximately constant value across

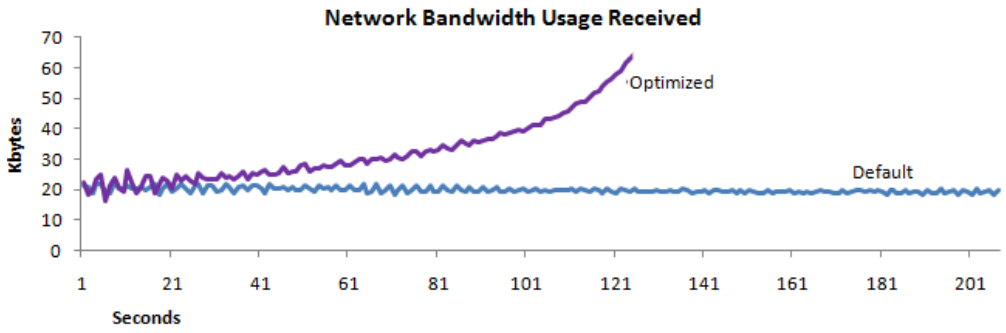

Figure 4.50: Outlook's Default vs Optimized Network Bandwidth Usage, Packets Received: 3.4 Kbyte Message Size

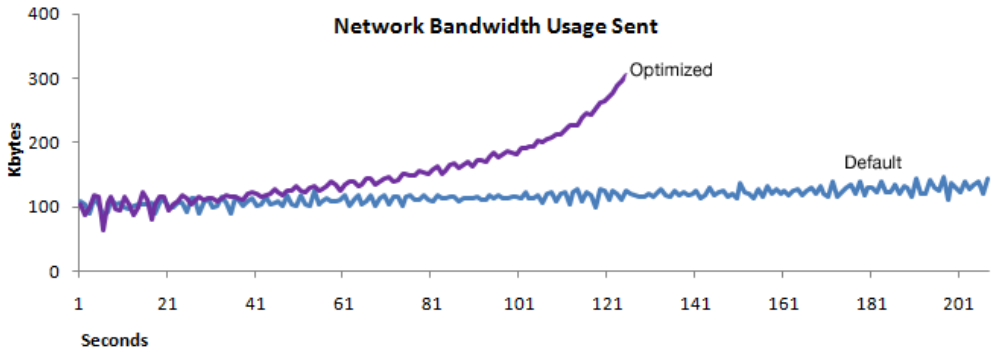

Figure 4.51: Outlook's Default vs Optimized Network Bandwidth Usage, Packets Sent: 3.4 Kbyte Message Size

the lifespan of the experiment. However, in the optimized behavior, an exponential graph was observed as the amount of messages in the Inbox declining over time while increasing in the individual message folders.

4.3.4 3.4 vs 76 Kbyte Message Size Comparison for Outlooks Default Behavior

This section presents the plotted graph results from comparison between Outlook's default behaviors for 3.4 Kbyte vs. 76 Kbyte messages sizes.

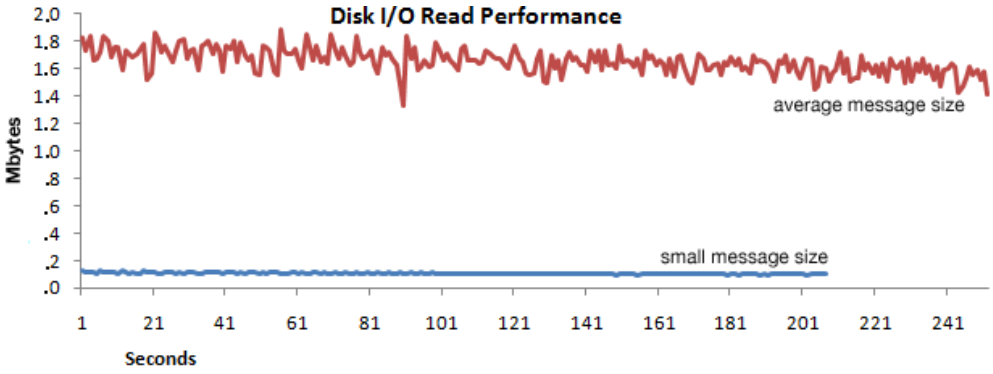

Figure 4.52: Outlook Default Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

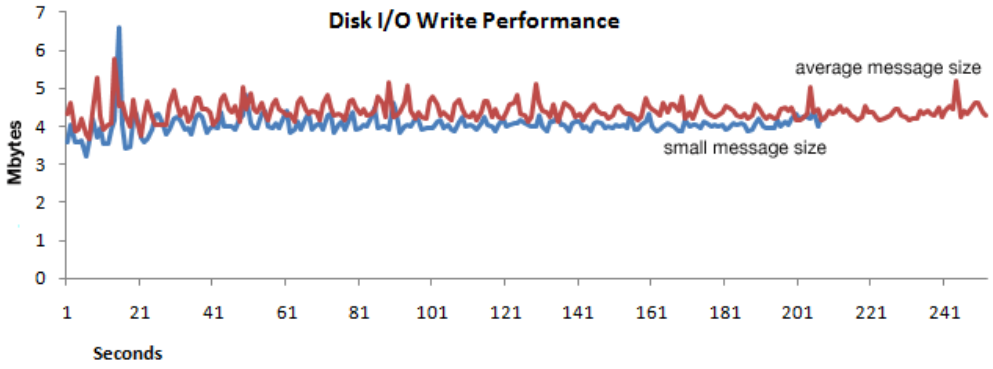

Figure 4.53: Outlook Default Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

These results show that the 76 Kbyte message size consumed significant resources in disk I/O read and network bandwidth usage for packets sent from IMAP server. On the other hand, the 3.4 Kbyte message experiment showed higher amounts of disk I/O write and network bandwidth usage for packets received by the server. This is a clear indication that as the message size increases, disk I/O read resource requirement also showed slight decreases.

Disk I/O write performance for this client showed similar performance for the two message sizes with the exception that the 76 Kbyte message experiment took more time. This can be a good example that, unlike disk I/O read, message size does not show a strong significant difference in IMAP protocol commands during disk I/O write because the status change has a limited relationship with message size. The network bandwidth usage for packets received

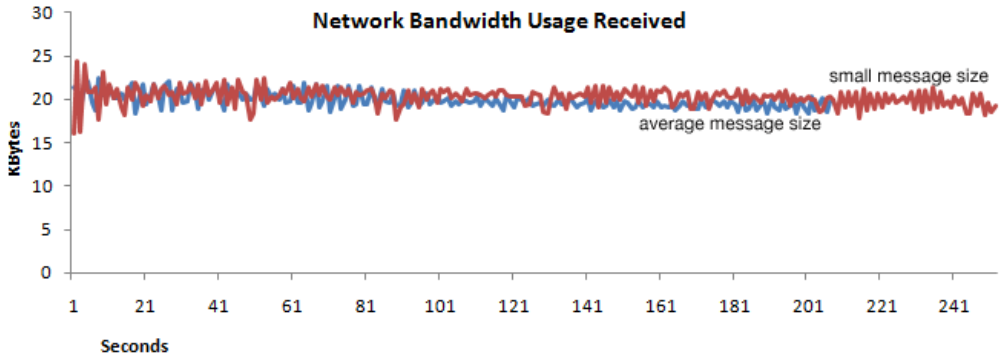

Figure 4.54: Outlook Default Behavior Network Bandwidth Usage (Packets Received) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

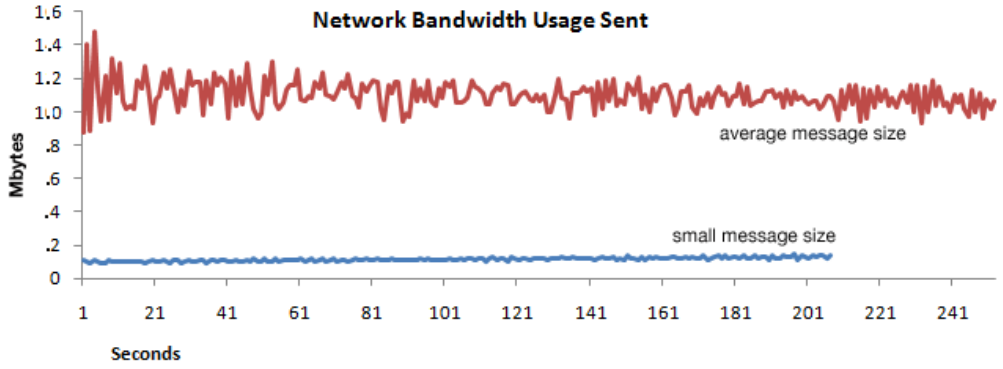

Figure 4.55: Outlook Default Behavior Network Bandwidth Usage (Packets Sent) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

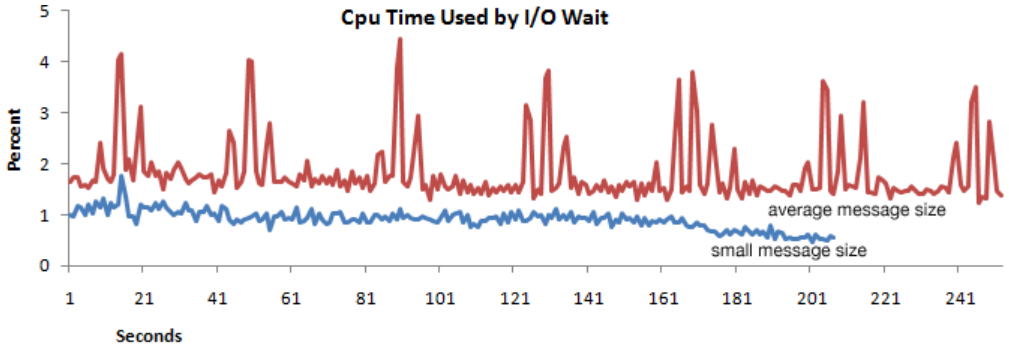

Figure 4.56: Outlook Default Behavior Disk I/O Wait Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

and sent were the reflection of the disk I/O write and read relationship. The CPU time in figure 4.56 is the reflection of the above discussion. High I/O wait CPU time for 76 Kbyte is due to high disk I/O read.

4.3.5 3.4 vs 76 Kbyte Message Size Comparison for Outlook Optimized Behavior

This section presents the plotted results comparing Outlook's optimized behavior for 3.4 Kbyte vs. 76 Kbyte messages sizes.

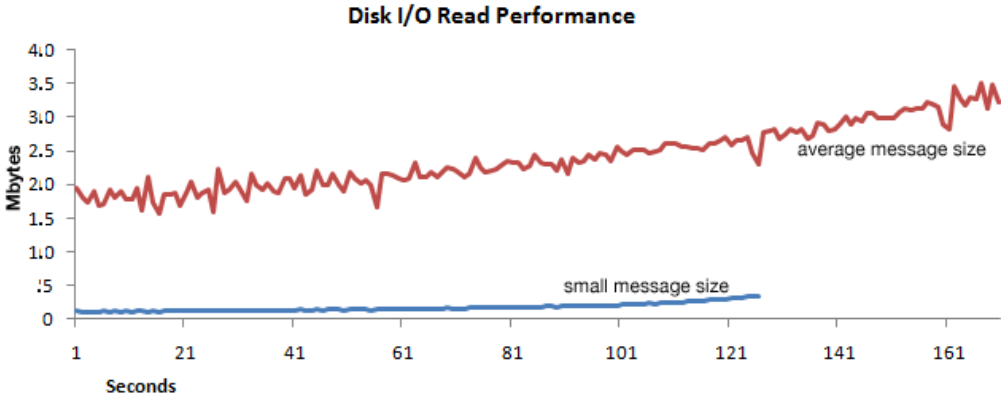

Figure 4.57: Outlook Optimized Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

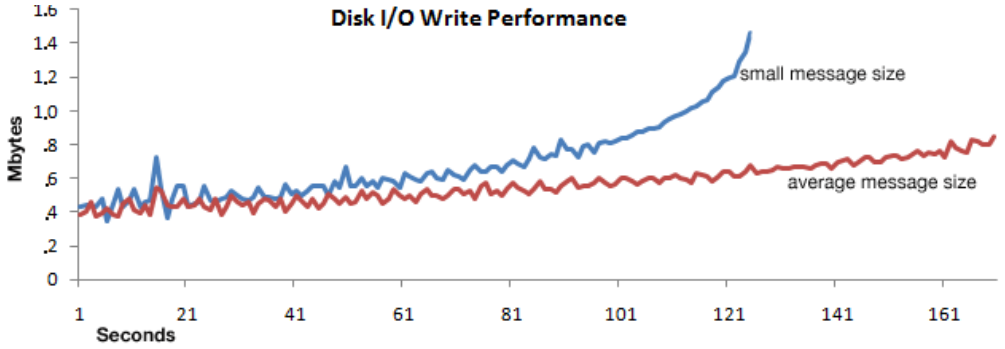

Figure 4.58: Outlook Optimized Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

The explanation for the preceding results is not much different from that for the default behavior comparison. however, the 76 Kbyte message size experiment did not exhibit the increasing slope observed in the 3.4 Kbyte client's default behavior. This is due to an increased I/O wait CPU time because of higher disk I/O read. Otherwise, the results reflect the higher resource demand resulting from higher message sizes.

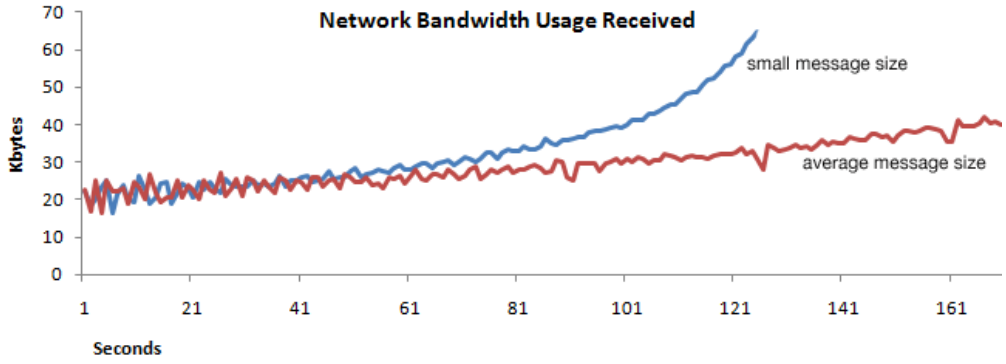

Figure 4.59: Outlook Optimized Behavior Network Bandwidth Usage (Packets Received) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

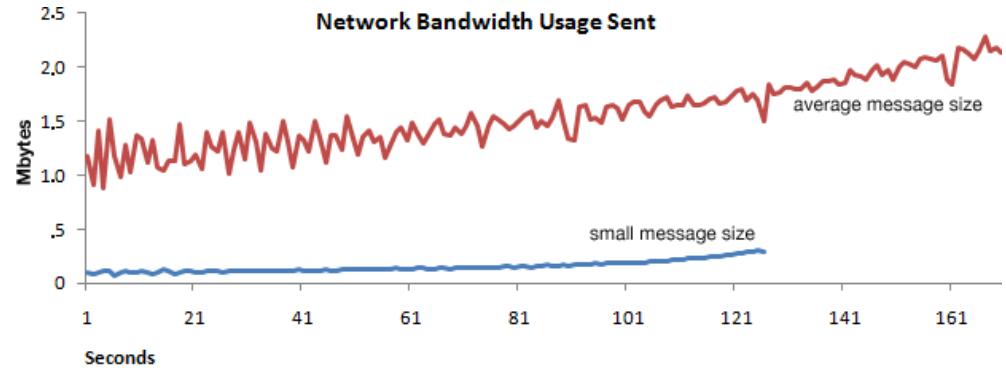

Figure 4.60: Outlook Optimized Behavior Network Bandwidth Usage (Packets Sent) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

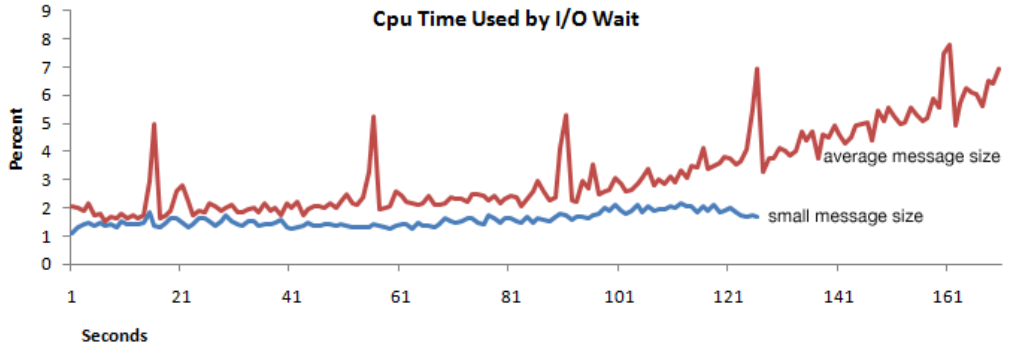

Figure 4.61: Outlook Optimized Behavior Disk I/O Wait Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

4.4 Sylpheed

4.4.1 Sylpheed Default Behavior for 3.4 Kbytes Message Size

The following graphs show Sylpheed's default behavior for 3.4 Kbytes message size. Figures 4.62, 4.63, 4.64, 4.65, 4.66 and 4.67 shows Sylpheed's default behavior server side resource usage for disk I/O read and write performance, network bandwidth usage for packets received and sent, and system interrupts and context switching. Figure 4.68 shows CPU time usage by user and system, and CPU idle and wait time. The graphs are plotted from per-point means of 35 replications.

Table 4.69 shows the calculated correlation coefficient values for this client's default behavior for all measured performance metrics.

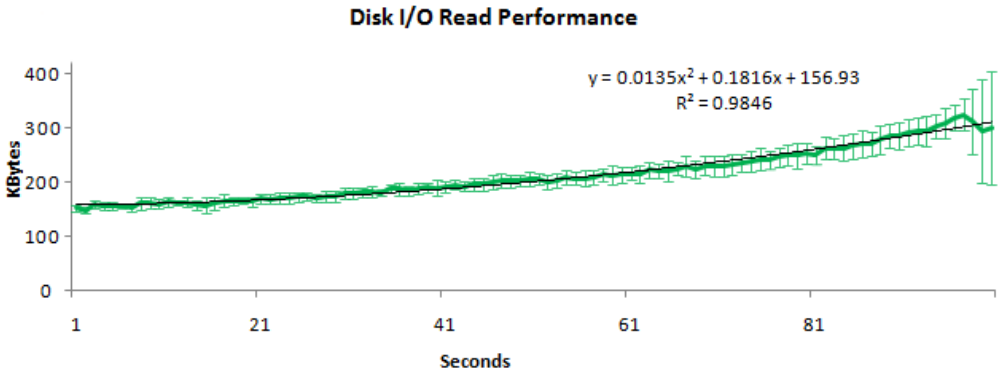

Figure 4.62: Sylpheed Default Behavior Disk I/O Read Performance: 3.4 Kbyte Message Size

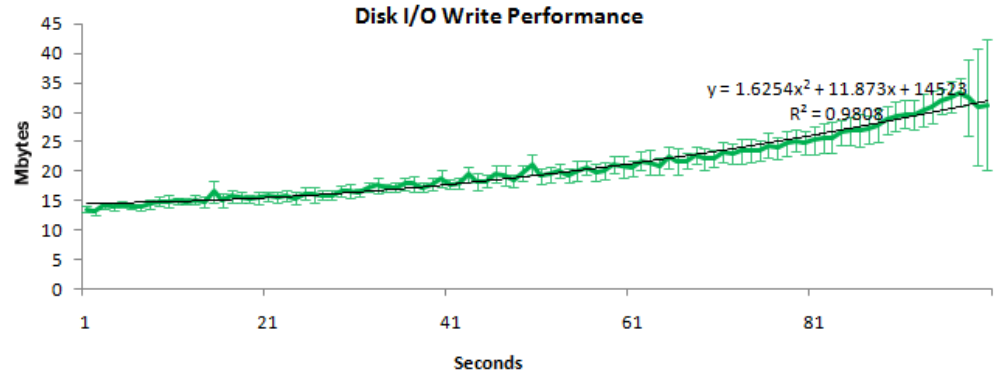

Figure 4.63: Sylpheed Default Behavior Disk I/O Write Performance: 3.4 Kbyte Message Size

Sylpheed has implemented the IMAP protocol differently as compared to other clients. The client is almost consistent in its default and optimized behavior,

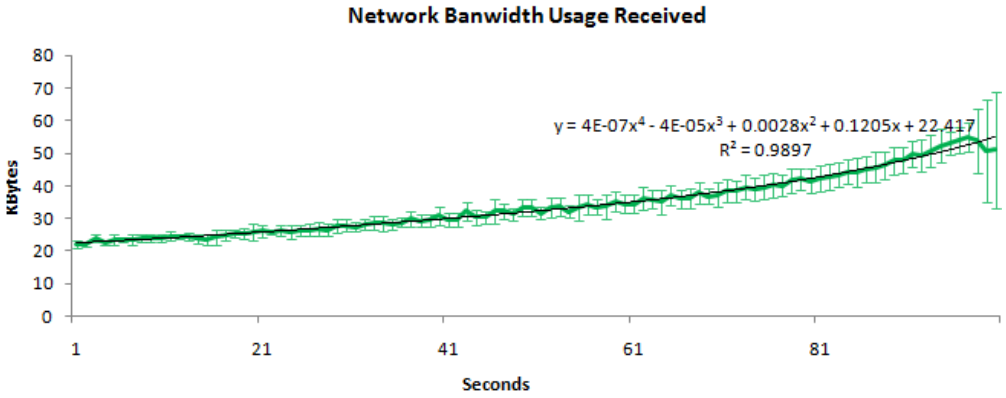

Figure 4.64: Sylpheed Default Behavior Network Bandwidth Received Performance: 3.4 Kbyte Message Size

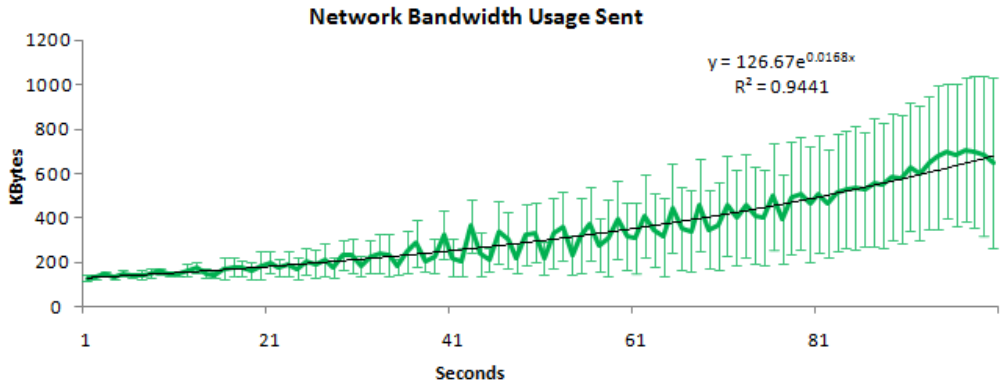

Figure 4.65: Sylpheed Default Behavior Network Bandwidth Sent Performance: 3.4 Kbyte Message Size

especially with respect to handling deleted messages. Sylpheed always runs the EXPUNGE command for all deleted and moved messages in the Inbox. Therefore, the client was able to free the Inbox after every single message manipulation. This useful feature of the client's behavior is shown in Figures 4.62, 4.63, 4.64, 4.65 4.66, 4.67 and 4.68.

Since Sylpheed freed deleted and moved messages immediately from the Inbox in the experiment, the amount of message did not become or remain a bottleneck, and the disk I/O and network bandwidth usage performance improved as the experiment progressed.

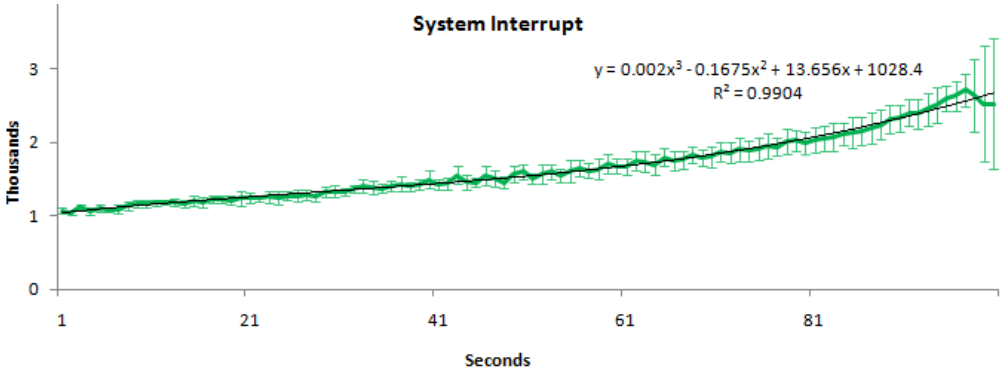

Figure 4.66: Sylpheed Default Behavior System Interrupts: 3.4 Kbyte Message Size

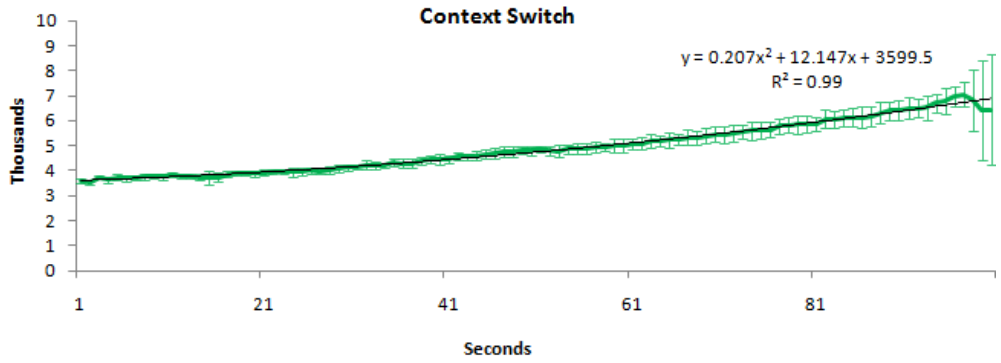

Figure 4.67: Sylpheed Default Behavior Context Switches: 3.4 Kbyte Message Size

Like most clients' default behavior with regards to CPU time, Sylpheed's default behavior showed a decrease in user CPU time consumption as the experiment continued and the amount of messages decreased. From the simulation scripts behavior, it is not difficult to understand this because a single user login for each 80 messages in a group manipulation cycle reflects the user CPU time usage, and each login required different resource levels as the amount of messages in the Inbox decreased.

The very strong negative correlation coefficient values shown in Figure 4.77

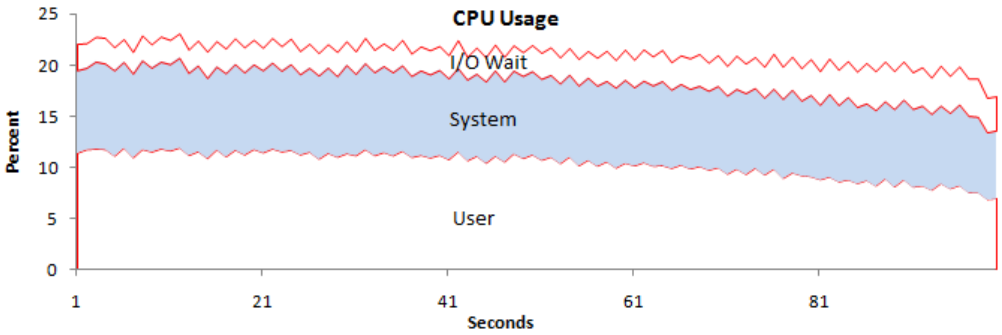

Figure 4.68: Sylpheed Default Behavior CPU Usage Performance: 3.4 Kbyte Message Size

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	1.00	1								
recv	1.00	1.00	1							
send	0.97	0.99	0.99	1						
int	1.00	1.00	1.00	0.99	1					
csw	0.99	0.99	0.99	0.97	0.99	1				
usr	-0.95	-0.96	-0.96	-0.95	-0.96	-0.95	1			
sys	-0.76	-0.77	-0.76	-0.77	-0.77	-0.75	0.85	1		
idl	0.83	0.84	0.84	0.84	0.85	0.82	-0.94	-0.94	1	
wai	0.92	0.92	0.92	0.90	0.91	0.92	-0.88	-0.66	0.69	1

Figure 4.69: Correlation Between Performance Metrics: Sylpheed Default Behavior, 3.4 Kbyte Message Size

are consistent with the observed behavior since they indicate an inverse relationship between the user CPU time and other performance metrics. In this client's behavior, I/O wait showed a positive correlation with disk I/O read and write, outgoing network bandwidth, and system interrupts and context switches.

4.4.2 Sylpheed Optimized Behavior for 3.4 Kbytes Message Size

The following graphs shows Sylpheed's optimized behavior for the 3.4 Kbytes message size. The first two graphs present disk I/O read and write. The next two present network bandwidth usage for packets received from and sent to the IMAP server (respectively). Next, system interrupt and context switch are displayed. Finally, the relationship between the different performance metrics is presented in the calculated correlation coefficient table. All graphs plot the per-point means of 35 replications.

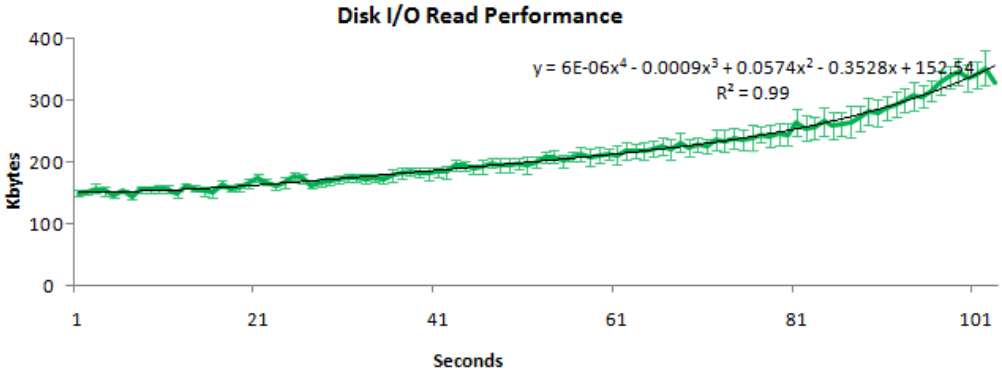

Figure 4.70: Sylpheed Optimized Behavior Disk I/O Read Performance: 3.4 Kbyte Message Size

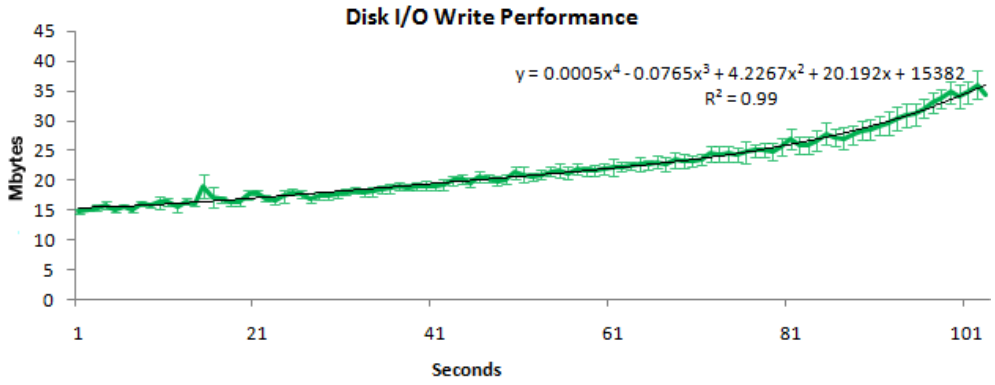

Figure 4.71: Sylpheed Optimized Behavior Disk I/O Write Performance: 3.4 Kbyte Message Size

Like Opera email client, Sylpheed does not require much work to optimized it. The only optimization implemented in this experiment is to empty the Trash box at the end of each message group manipulation. The trend of performance requirements in the optimized behavior of Sylpheed is similar to the default one. This is because of the consistent characteristics of the client in its default and optimized behavior. The similarity in correlation coefficient values and CPU time usage for both default and optimized behavior is futher evidence for this.

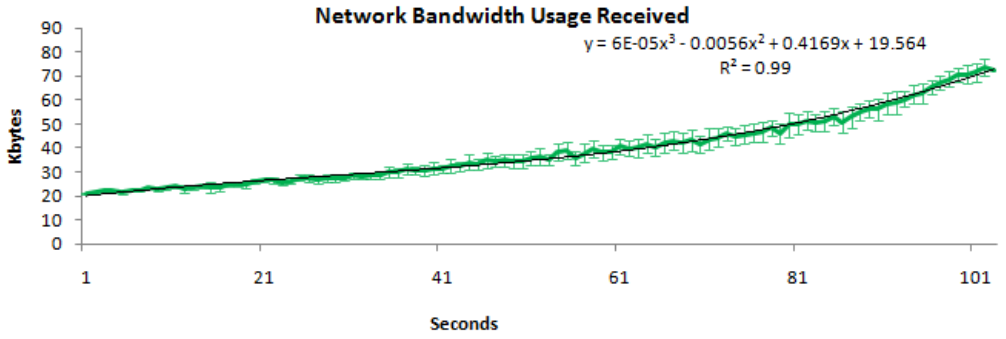

Figure 4.72: Sylpheed Optimized Behavior Network Bandwiddth Usage for Received Packets: 3.4 Kbyte Message Sizer

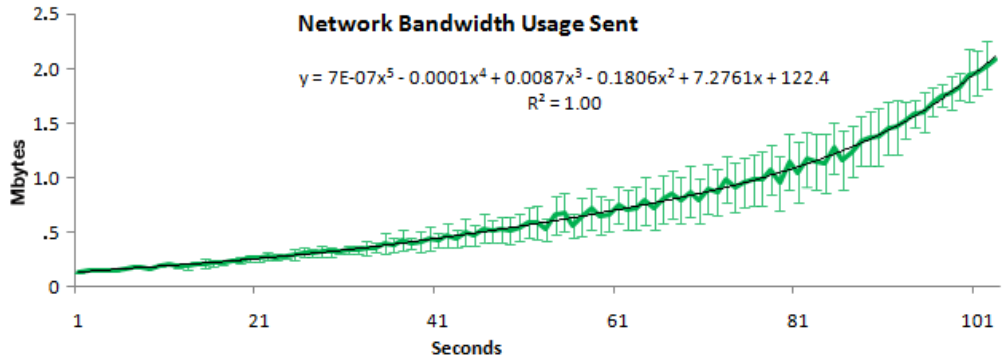

Figure 4.73: Sylpheed Optimized Behavior Network Bandwiddth Usage for Sent Packets: 3.4 Kbyte Message Size

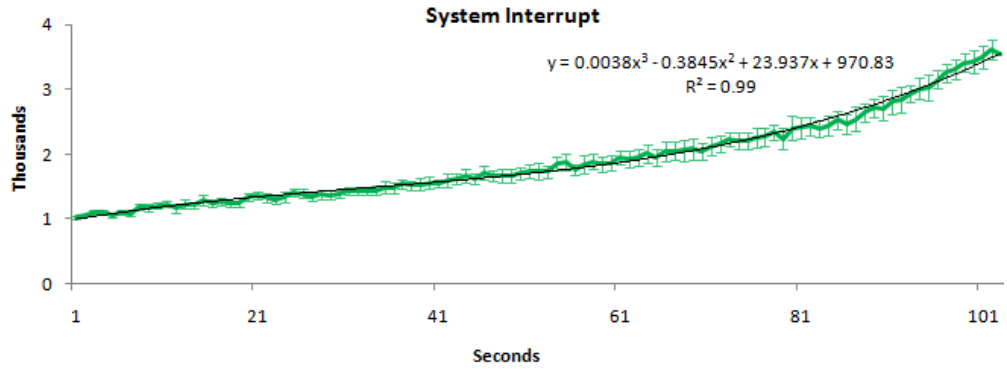

Figure 4.74: Sylpheed Optimized Behavior System Interrupts: 3.4 Kbyte Message Size

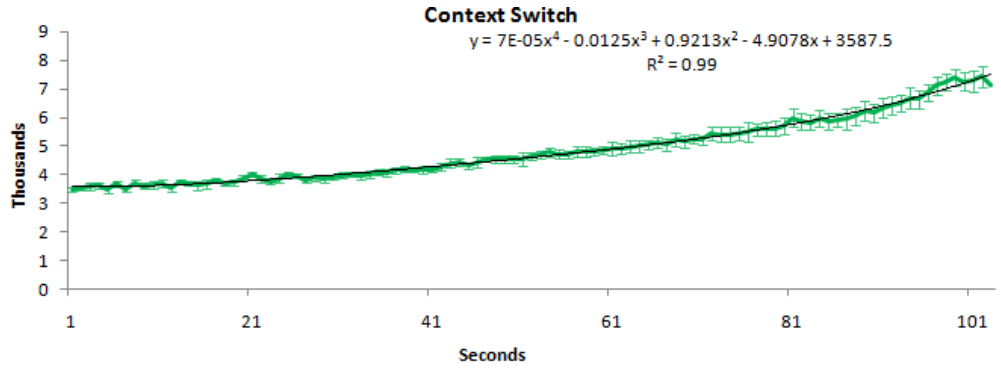

Figure 4.75: Sylpheed Optimized Behavior Context Switches: 3.4 Kbyte Message Size

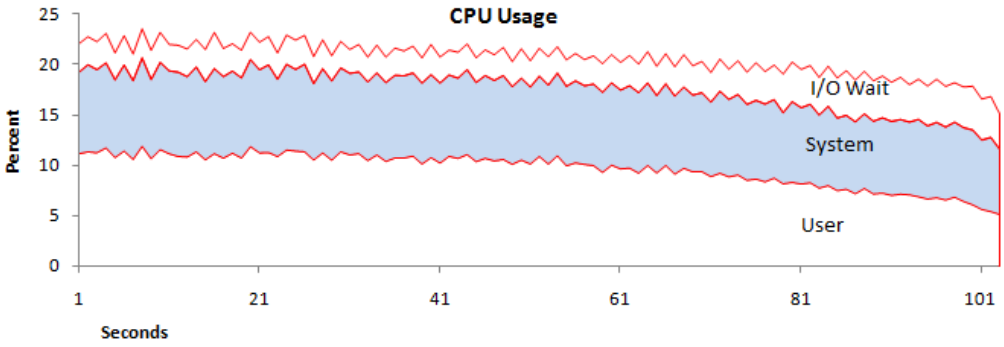

Figure 4.76: Sylpheed Optimized Behavior CPU Usage: 3.4 Kbyte Message Size

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	1.00	1								
recv	1.00	1.00	1							
send	0.99	0.99	1.00	1						
int	1.00	1.00	1.00	1.00	1					
csw	1.00	1.00	1.00	0.99	1.00	1				
usr	-0.96	-0.96	-0.97	-0.97	-0.97	-0.96	1			
sys	-0.76	-0.77	-0.77	-0.78	-0.77	-0.76	0.86	1		
idl	0.89	0.89	0.90	0.90	0.90	0.89	-0.95	-0.94	1	
wai	0.83	0.83	0.82	0.83	0.82	0.83	-0.85	-0.63	0.68	1

Figure 4.77: Sylpheed Optimized Behavior Correlation Coefficients

4.4.3 Sylpheed's Default vs Optimized Behavior for 3.4 Kbytes Message Size

Figures 4.78,4.79,4.80, and 4.81 show the comparisons between Sylpheed's default and optimized behavior resource consumption while running the experiment.

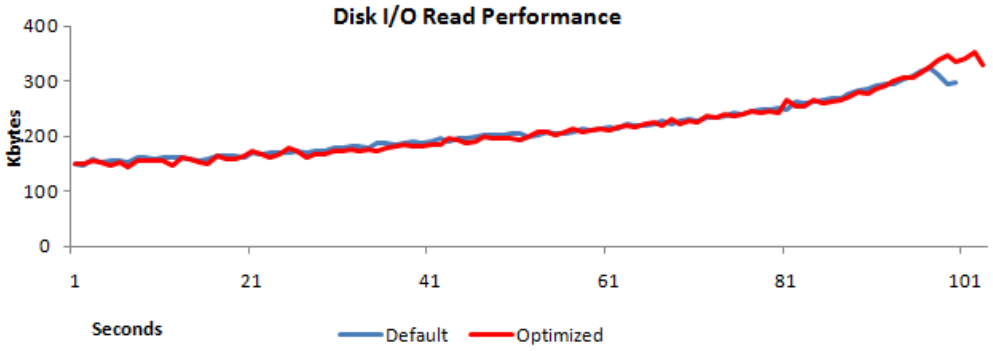

Figure 4.78: Sylpheed Default vs Optimized Disk I/O Read Performance: 3.4 Kbyte Message Size

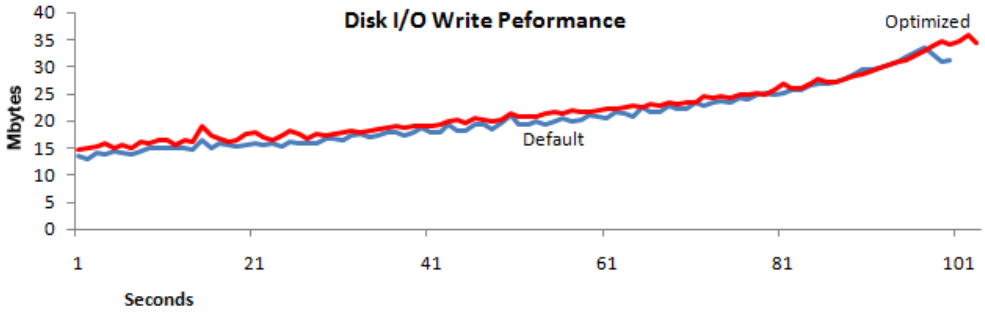

Figure 4.79: Sylpheed Default vs Optimized Disk I/O Write Performance: 3.4 Kbyte Message Size

As the graphs indicate, the disk I/O resource usage for read and write performance was quite similar. The slight differences but similar trends for network bandwidth usage for received packets could be due to the additional commands to optimize the client, such as clearing Trash box at the end of each message group manipulation using Expunge command. The server's response to this additional command with a list of deleted messages could contribute to the significant difference in network bandwidth usage for packets sent.

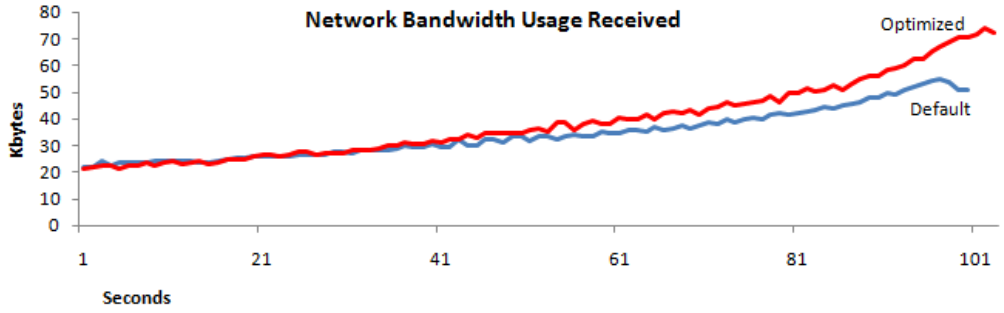

Figure 4.80: Sylpheed Default vs Optimized Network Bandwidth Usage, Packets Received: 3.4 Kbyte Message Size

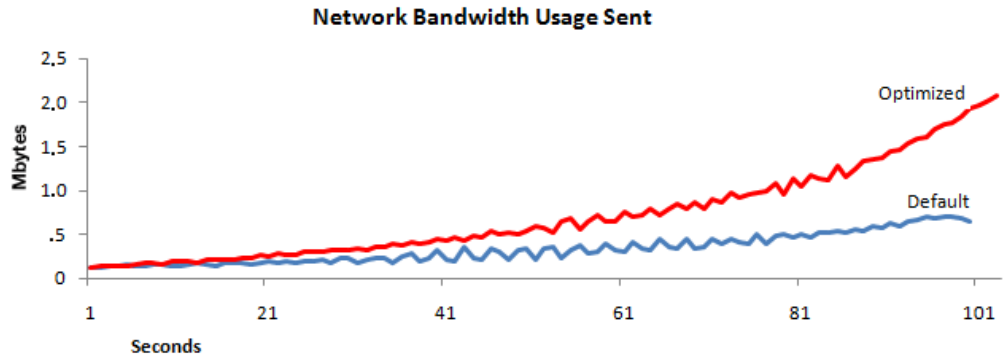

Figure 4.81: Sylpheed Default vs Optimized Network Bandwidth Usage, Packets Sent: 3.4 Kbyte Message Size

4.4.4 3.4 vs 76 Kbytes Message Size Comparison for Sylpheeds's Default Behavior

Figures 4.82, 4.83, 4.84, 4.85, and 4.86 show the comparisons for Sylpheed's 3.4 vs. 76 Kbyte message size results under the client's default behavior.

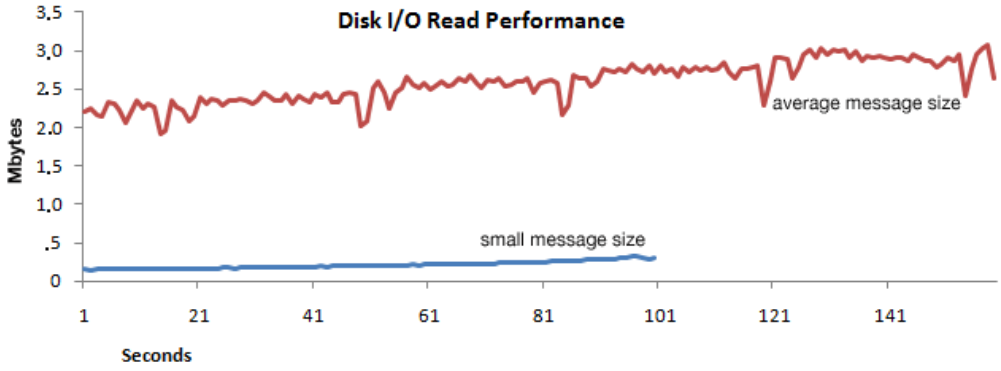

Figure 4.82: Sylpheed Default Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

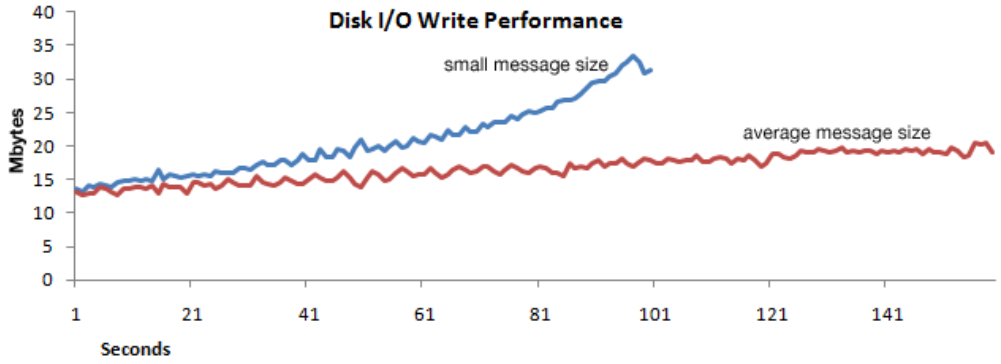

Figure 4.83: Sylpheed Default Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

The comparison between message size results shown in the preceding figures is not much different from other clients behavior. As before, the 76 Kbyte message size required much more disk I/O read resources than the 3.4 Kbyte message size. Disk I/O write performance took longer time to finish for the larger message size because disk I/O activities were slowed down by the high disk I/O read tasks and the performance consumption dispersed across time. This could be the reason that the graph for disk I/O could not follow the 3.4 Kbyte slope for default behavior (see 4.83).

The high CPU I/O wait difference observed in Figure 4.86 is a good evidence for high disk I/O read rates during 76 Kbyte message size experiments.

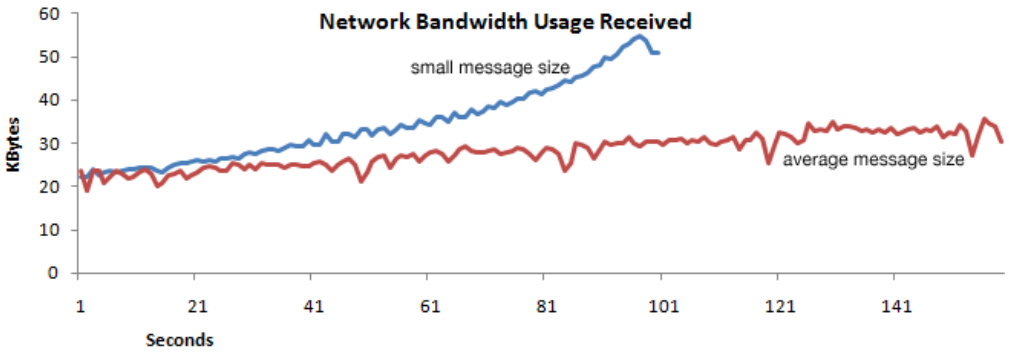

Figure 4.84: Sylpheed Default Behavior Network Bandwidth Usage (Packets Received) Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

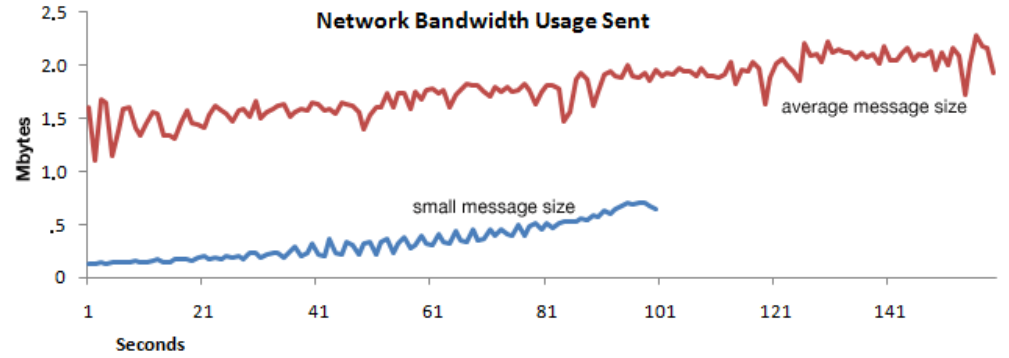

Figure 4.85: Sylpheed Default Behavior Network Bandwidth Usage (Packets Sent) Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

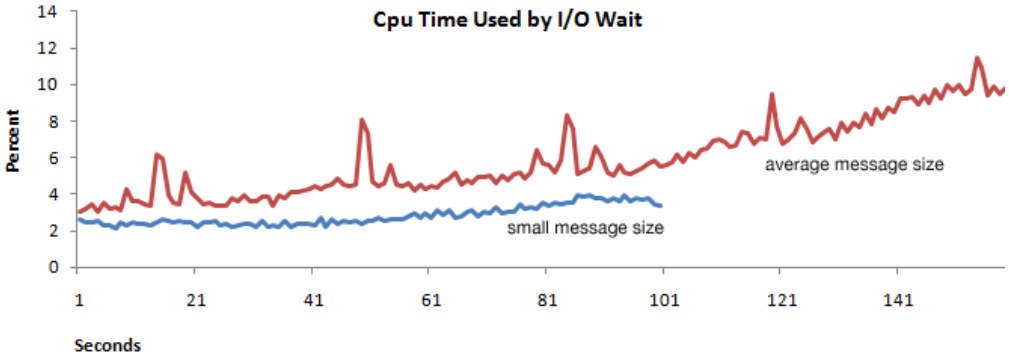

Figure 4.86: Sylpheed Default Behavior Disk I/O Wait Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

4.4.5 3.4 vs 76 Kbytes Message Size Comparison for Sylpheed's optimized Behavior

Figures 4.87, 4.88, 4.89, 4.90, and 4.91 graph Sylpheed's 3.4 and. 76 Kbyte message size optimized behavior results.

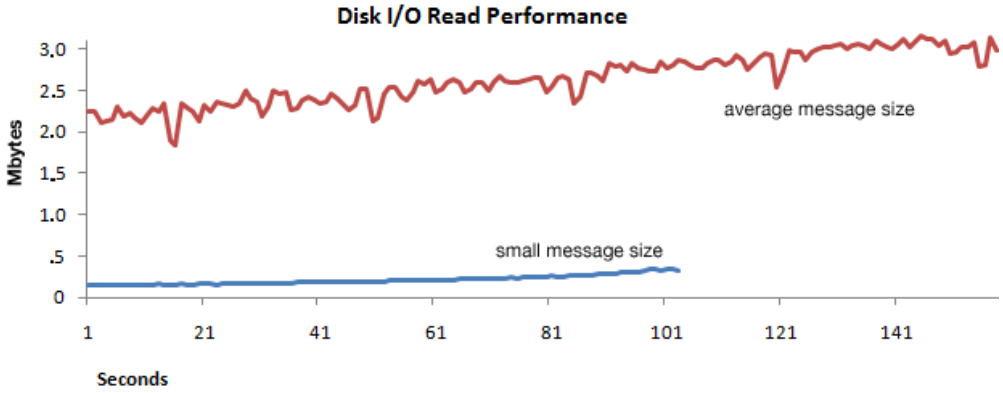

Figure 4.87: Sylpheed Optimized Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

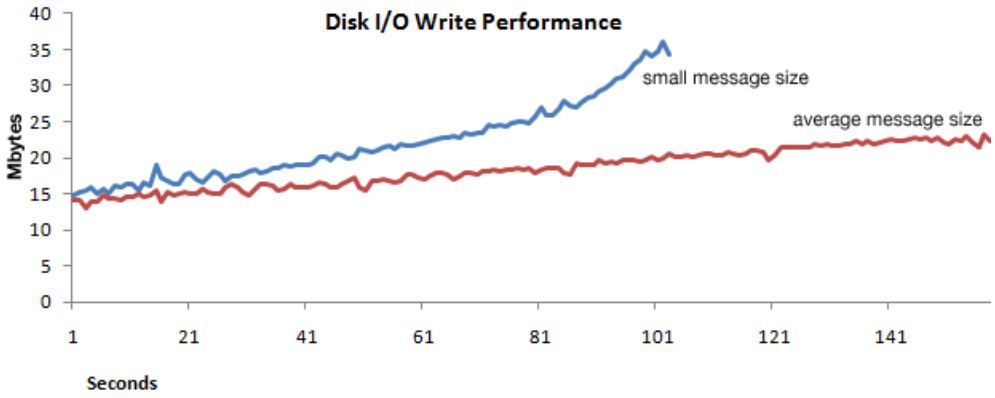

Figure 4.88: Sylpheed Optimized Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

The message size difference comparison for optimized behavior of the client showed in the preceding figures has a similar trend to that seen in the comparison between default vs optimized behavior of the client.

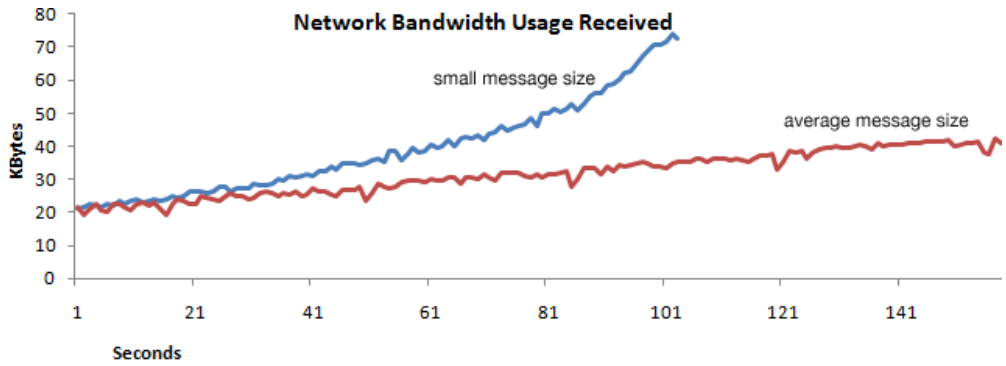

Figure 4.89: Sylpheed Optimized Behavior Network Bandwidth Usage (Packets Received) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

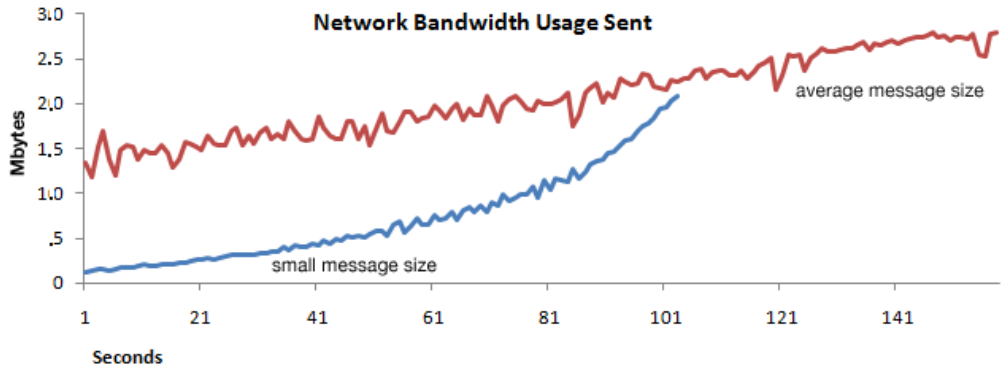

Figure 4.90: Sylpheed Optimized Behavior Network Bandwidth Usage (Packets Sent) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

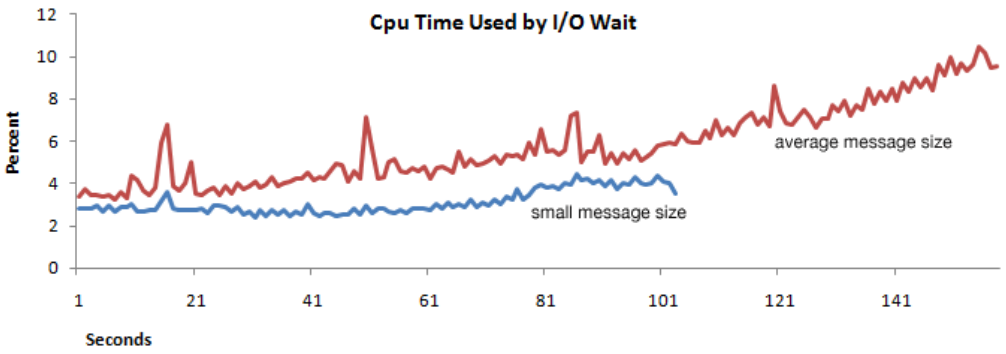

Figure 4.91: Sylpheed Optimized Behavior Disk I/O Wait Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

4.5 Thunderbird

4.5.1 Thunderbird Default Behavior for the 3.4 Kbyte Message Size

The following graphs shows Thunderbird's default behavior for the 3.4 Kbyte message size. Figures 4.92, 4.93, 4.94, 4.95 4.96 and 4.97 show Thunderbird's default behavior server side resource usage for disk I/O read and write performance, network bandwidth usage for packets received and sent, and system interrupts and context switching. Figure 4.98 shows CPU time usage by user and system, and I/O wait time. The graphs are plotted from per-point means of 35 replications.

Table 4.99 shows the calculated correlation coefficient values for all performance metrics.

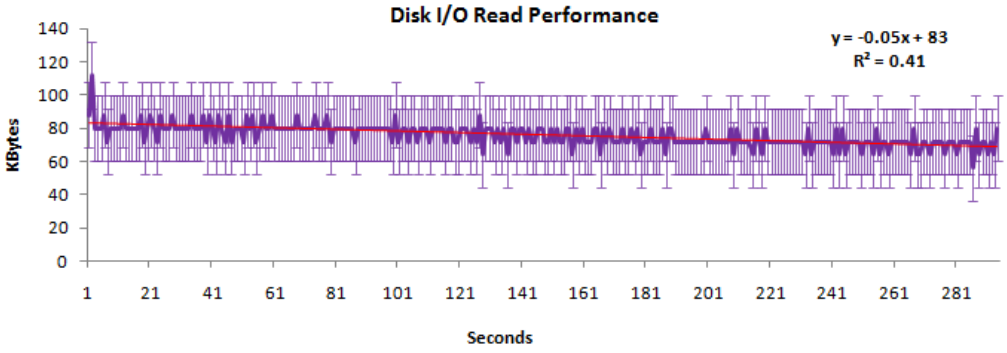

Figure 4.92: Thunderbird Default Behavior Disk I/O Read Performance: 3.4 Kbyte Message Size

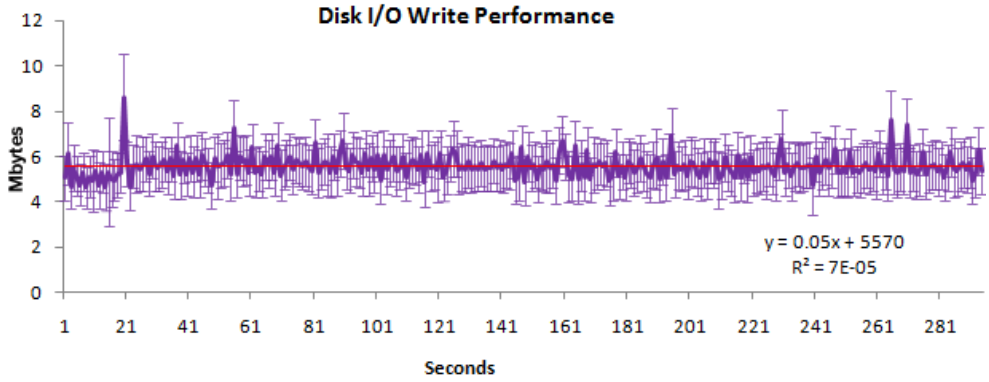

Figure 4.93: Thunderbird Default Behavior Disk I/O Write Performance: 3.4 Kbyte Message Size

The resource requirements for this experiment were almost constant, as in the cases for Thunderbird and Outlook, because the amount of messages in the Inbox were the same throughout the experimental cycle. The slight decrease

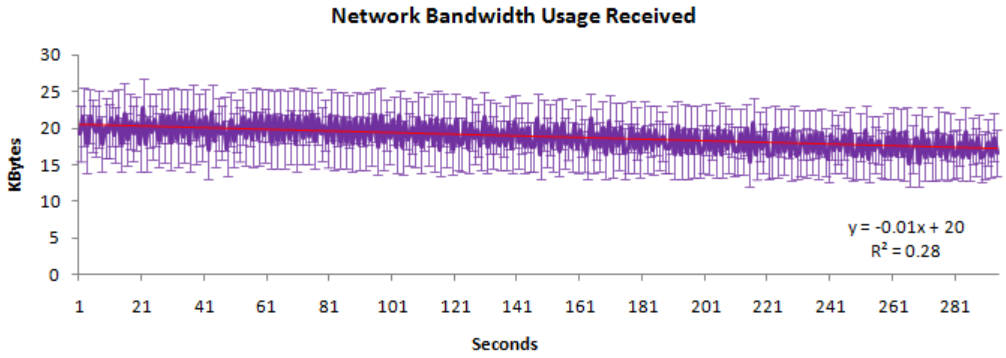

Figure 4.94: Thunderbird Default Behavior Network Bandwidth Received Performance: 3.4 Kbyte Message Size

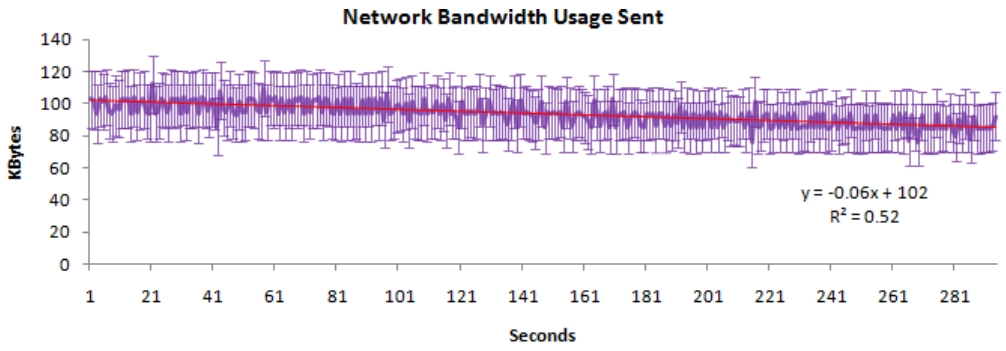

Figure 4.95: Thunderbird Default Behavior Network Bandwidth Sent Performance: 3.4 Kbyte Message Size

in disk I/O read, network bandwidth usage for packets received and sent, and system interrupt and context switch could be due to the slight increase in the amount of messages in the five message folders other than the Inbox. The decrease in performance is indicated by slope -0.05, -0.01, -0.06 and -0.01 for disk I/O read, network bandwidth usage for packets received and sent, and system interrupt and context switching, respectively.

Although the R-coefficient value indicates that the linear function does not represent all points, one can see the decline in the above mentioned parameters was insignificant. This is because Thunderbird by default does not permanently remove deleted messages unless the user utilizes optimizing options explained previously in the methodology chapter of this thesis. Thus, since a large amount of messages remained in the Inbox, the resource usage for the whole experiment was dominated by this message box. The increase in disk I/O write performance, however, could not be explained from client's behavior perspective. This result could be due to the low disk I/O write at the start and sudden increase after 20 seconds.

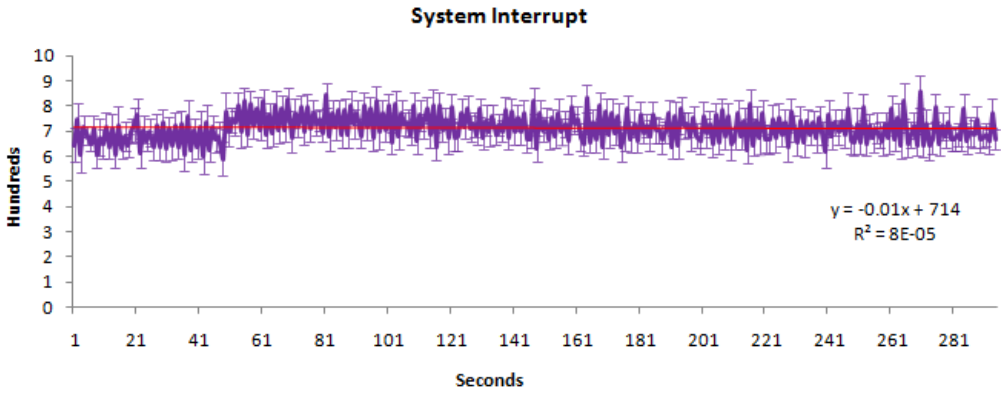

Figure 4.96: Thunderbird Default Behavior System Interrupts: 3.4 Kbyte Message Size

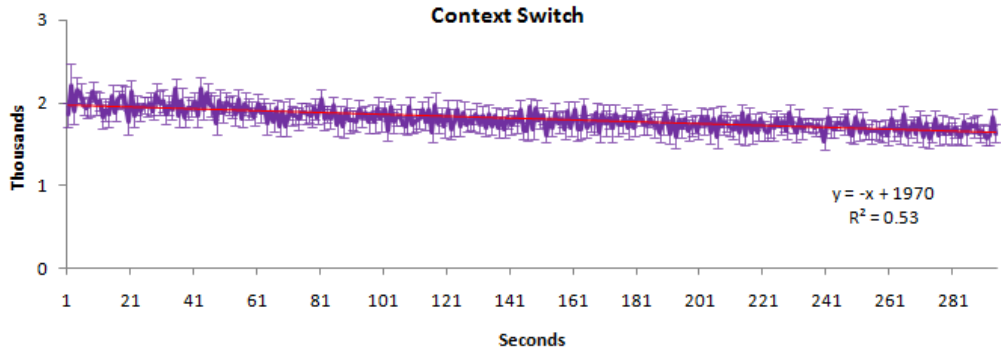

Figure 4.97: Thunderbird Default Behavior Context Switches: 3.4 Kbyte Message Size

Figure 4.98 shows the CPU usage for the entire experimental cycle. As one

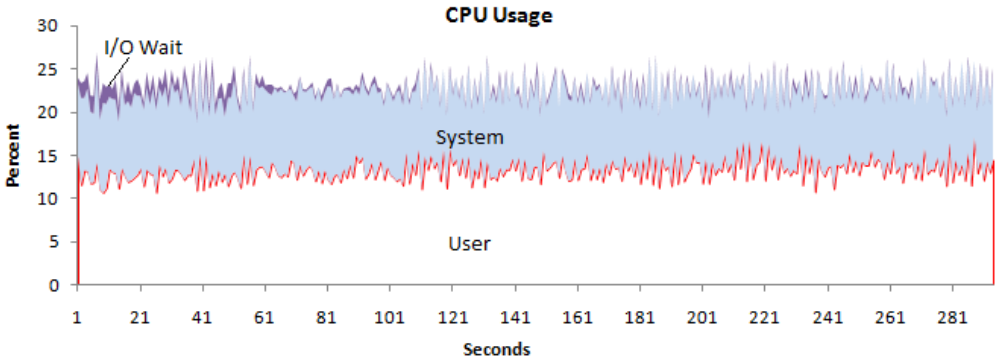

Figure 4.98: Thunderbird Default Behavior CPU Usage Performance: 3.4 Kbyte Message Size

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	-0.03	1								
recv	0.23	0.42	1							
send	0.71	0.23	0.80	1						
int	-0.07	0.73	0.76	0.46	1					
csw	0.41	0.33	0.82	0.75	0.44	1				
usr	-0.13	0.02	-0.15	-0.16	0.03	-0.26	1			
sys	0.03	0.06	0.03	0.01	0.05	0.05	0.10	1		
idl	0.00	-0.01	0.04	0.03	0.02	0.01	-0.74	-0.67	1	
wai	0.28	-0.16	0.19	0.28	-0.27	0.51	-0.23	0.04	-0.14	1

Figure 4.99: Correlation Between Performance Metrics: Thunderbird Default Behavior: 3.4 Kbyte Message Size

can see from the graph, the CPU time usage from user, system, I/O wait was constant. User time comprised the highest amount of time, followed by system time. The I/O wait time was almost zero.

Figure 4.99 shows the correlation between the performance metrics chosen for this thesis experiment. From the table, there was no strong relationship between the select performance metrics, as expected.

4.5.2 Thunderbird Optimized Behavior for 3.4 Kbyte Message Size

The following plotted graphs show Thunderbird's optimized behavior for 3.4 Kbyte message size. Figures 4.100, 4.101, 4.102, 4.103, 4.104 and 4.105 show Thunderbird's optimized behavior server side resource demand for disk I/O read and write performance, network bandwidth usage for packets received and sent, system interrupts and context switching, respectively. Figure 4.106 shows the CPU time usage by user and system time as well as I/O wait time.

Figure 4.107 shows the correlation between the performance metrics chosen for this thesis experiment.

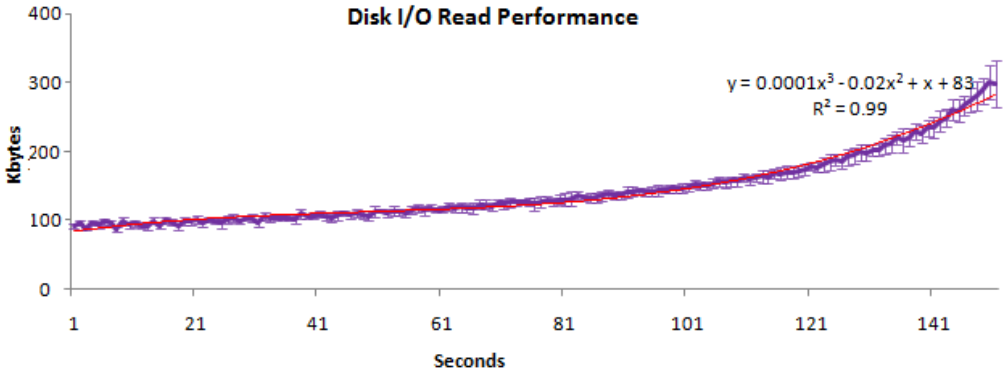

Figure 4.100: Thunderbird Optimized Behavior Disk I/O Read Performance: 3.4 Kbyte Message Size

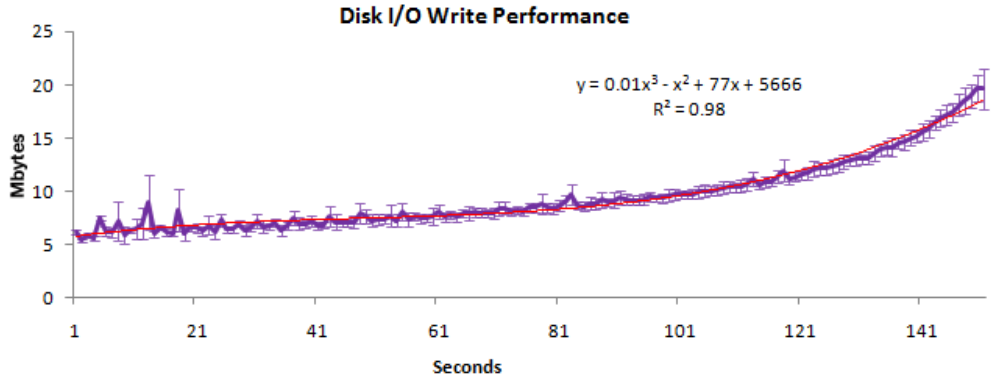

Figure 4.101: Thunderbird Optimized Behavior Disk I/O Write Performance: 3.4 Kbyte Message Size

From the preceding results, it is possible to say that the resource requirements were increasing for all metrics except the CPU usage. The constant start in user CPU time usage is followed by a a very slight decline after almost 80 second (see Figure 4.106). This could be a good reflection of the user CPU time requirement decreasing as the amount of messages decrease in the Inbox.

The strong correlation coefficient seen in Table 4.107 is another good reflection

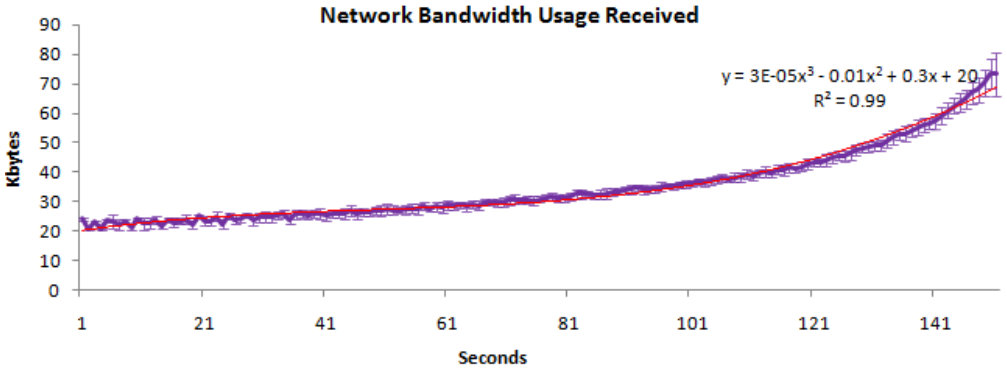

Figure 4.102: Thunderbird Optimized Behavior Network Bandwiddth Usage for Received Packets: 3.4 Kbyte Message Sizer

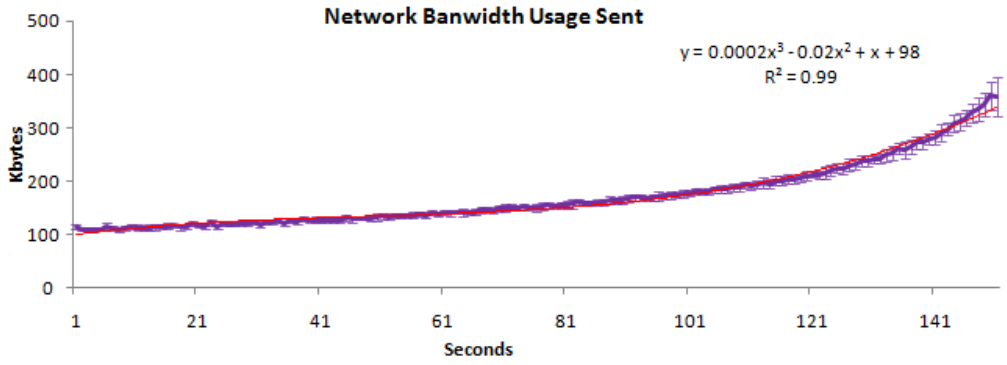

Figure 4.103: Thunderbird Optimized Behavior Network Bandwiddth Usage for Sent Packets: 3.4 Kbyte Message Size

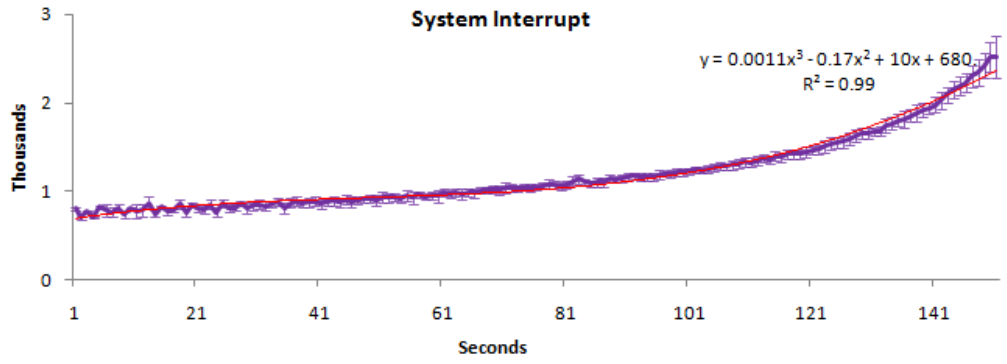

Figure 4.104: Thunderbird Optimized Behavior System Interrupts: 3.4 Kbyte Message Size

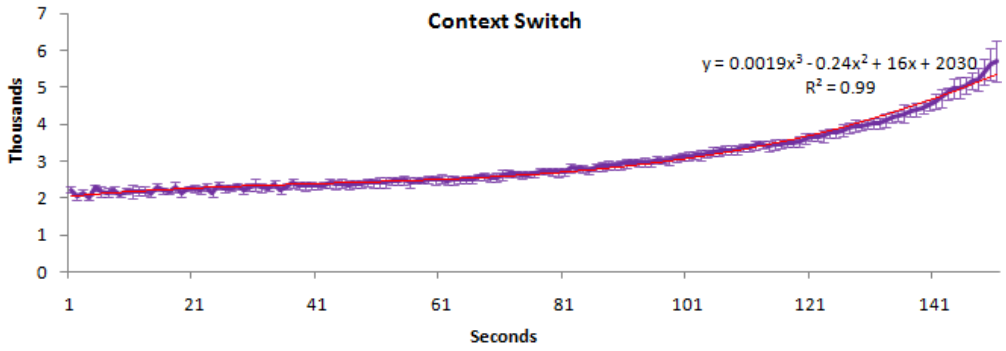

Figure 4.105: Thunderbird Optimized Behavior Context Switches: 3.4 Kbyte Message Size

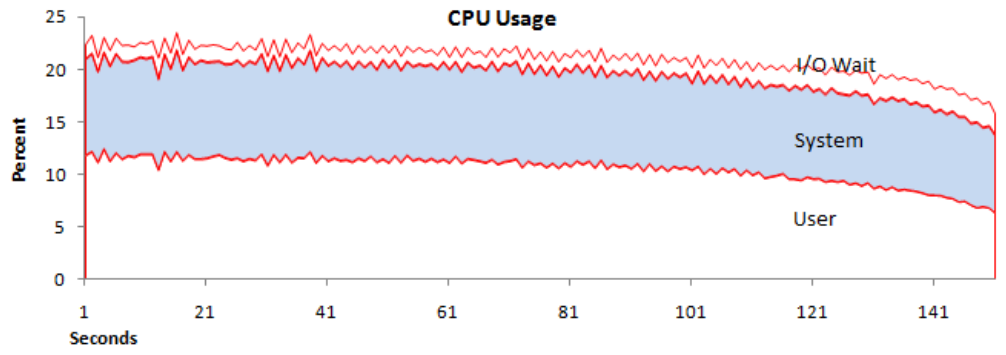

Figure 4.106: Thunderbird Optimized Behavior CPU Usage: 3.4 Kbyte Message Size

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	0.99	1								
recv	1.00	0.99	1							
send	1.00	0.99	1.00	1						
int	1.00	1.00	1.00	1.00	1					
csw	1.00	0.99	1.00	1.00	1.00	1				
usr	-0.97	-0.98	-0.98	-0.97	-0.98	-0.98	1			
sys	-0.82	-0.83	-0.83	-0.83	-0.83	-0.83	0.87	1		
idl	0.93	0.94	0.94	0.94	0.94	0.94	-0.98	-0.94	1	
wai	0.85	0.86	0.85	0.85	0.86	0.86	-0.86	-0.70	0.76	1

Figure 4.107: Thunderbird Optimized Behavior Correlation Coefficients

of strong relationship between disk I/O and network bandwidth usage.

4.5.3 Thunderbird Default vs Optimized Behavior for 3.4 Kbyte Message Size

Figures 4.108,4.109,4.110, 4.111 shows the comparisons between Thunderbird's default and optimized behavior resource consumption for the experiment.

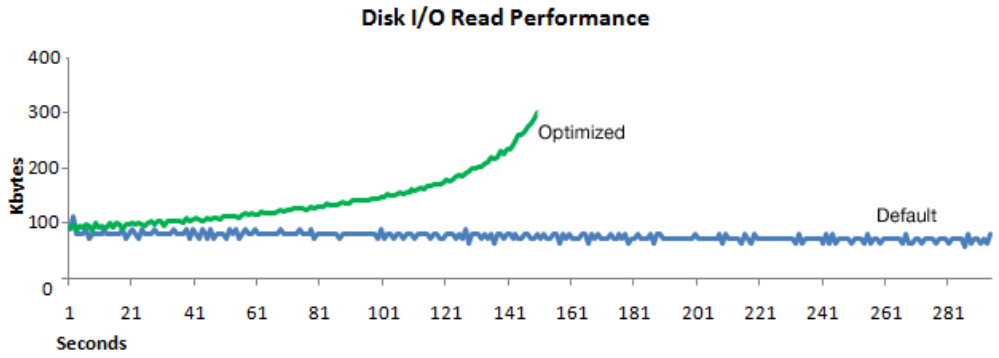

Figure 4.108: Thunderbird Default vs Optimized Disk I/O Read Performance: 3.4 Kbyte Message Size

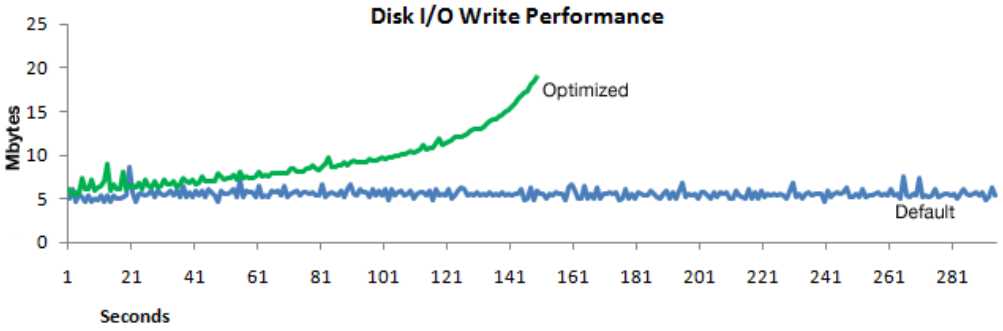

Figure 4.109: Thunderbird Default vs Optimized Disk I/O Write Performance: 3.4 Kbyte Message Size

As shown in the preceding graphs, the default and optimized behavior of the client disk I/O and network bandwidth usage show significant differences. While the default behavior requires very close to constant disk I/O read performance and network bandwidth usage in both directions, the optimized behavior begins with equal performance levels to the default behavior but then shows a significant increase as the amount of messages in the Inbox decreases.

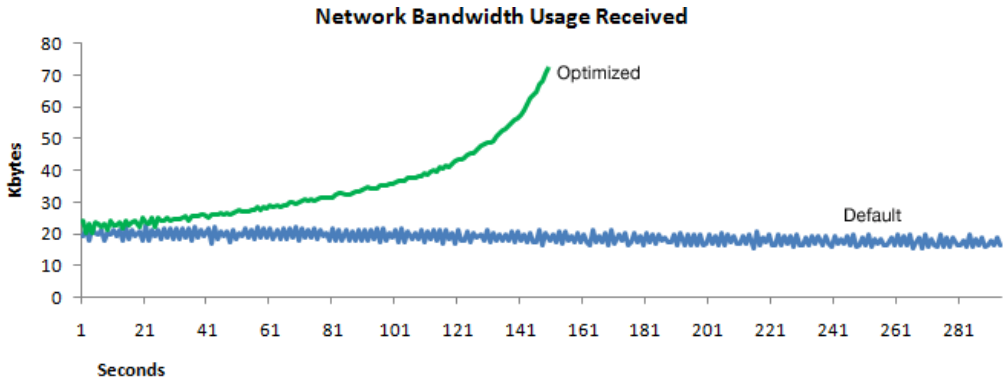

Figure 4.110: Thunderbird Default vs Optimized Network Bandwidth Usage, Packets Received: 3.4 Kbyte Message Size

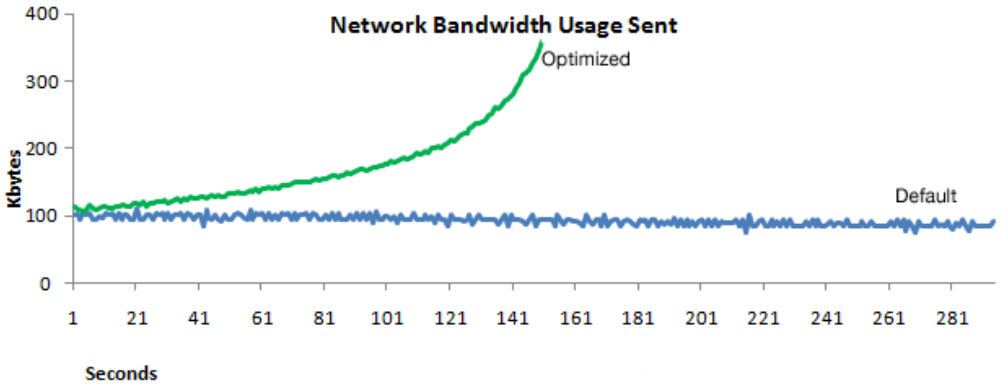

Figure 4.111: Thunderbird Default vs Optimized Network Bandwidth Usage, Packets Sent: 3.4 Kbyte Message Size

4.5.4 3.4 vs 76 Kbyte Message Size Comparison for Thunderbirds's Default Behavior

This section presents the plotted graph results from comparison between Thunderbird's default behaviors for 3.4 Kbyte vs. 76 Kbyte messages sizes.

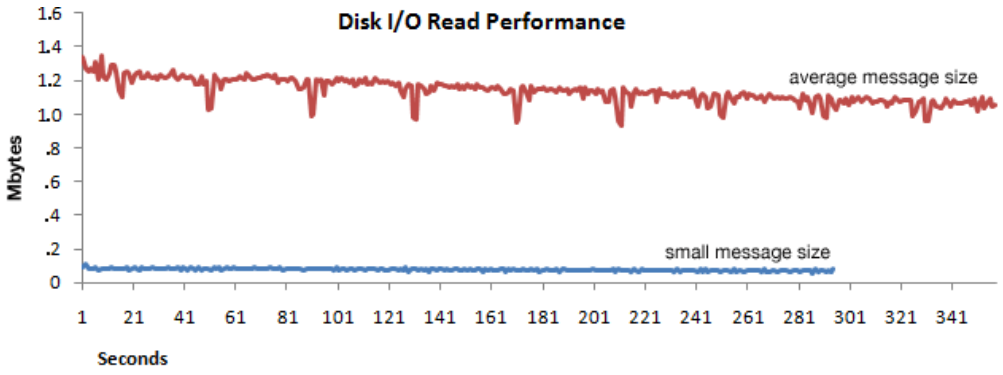

Figure 4.112: Thunderbird Default Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

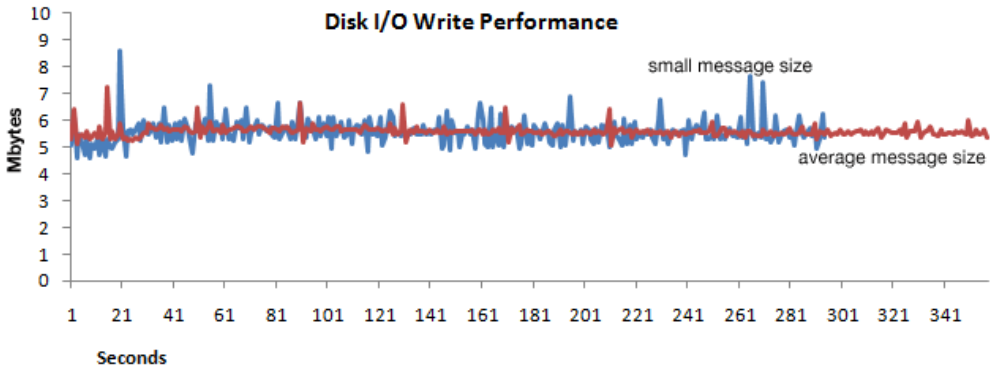

Figure 4.113: Thunderbird Default Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

The results from the preceding figures display a common performance increase for 76 Kbyte as compared to 3.4 Kbyte message size, as discussed for other clients. The only difference here is Thunderbird's high disk I/O write rate. This is supported by the results seen in the I/O wait CPU time (see Figure 4.116).

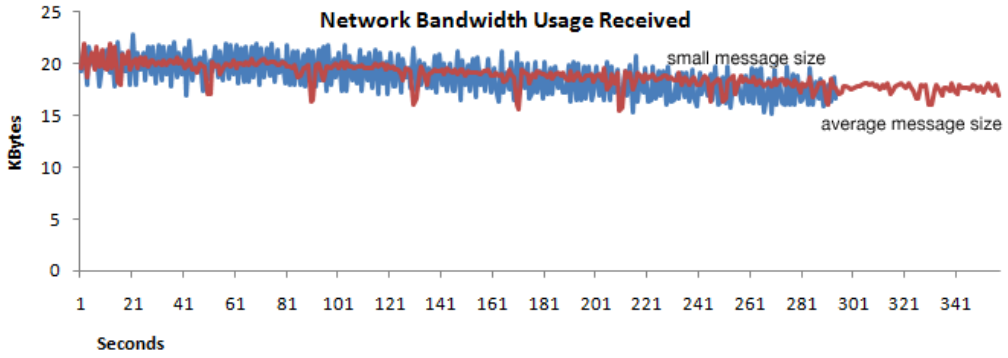

Figure 4.114: Thunderbird DEfault Behavior Network Bandwidth Usage (Packets Received) Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

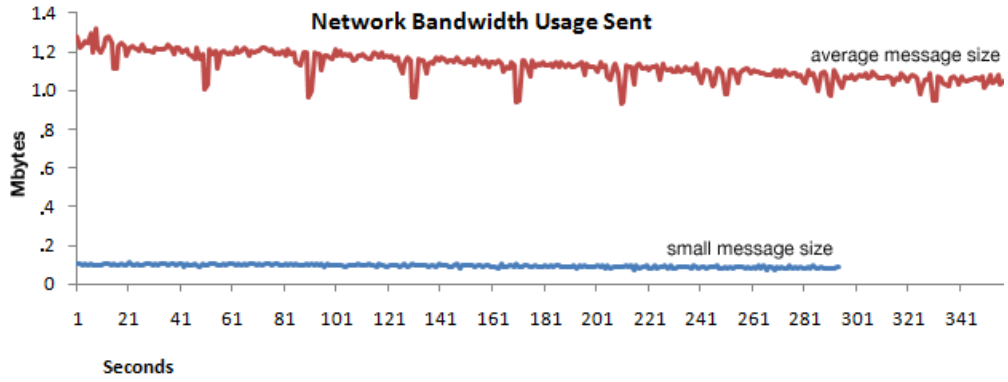

Figure 4.115: Thunderbird Default Behavior Network Bandwidth Usage (Packets Sent) Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

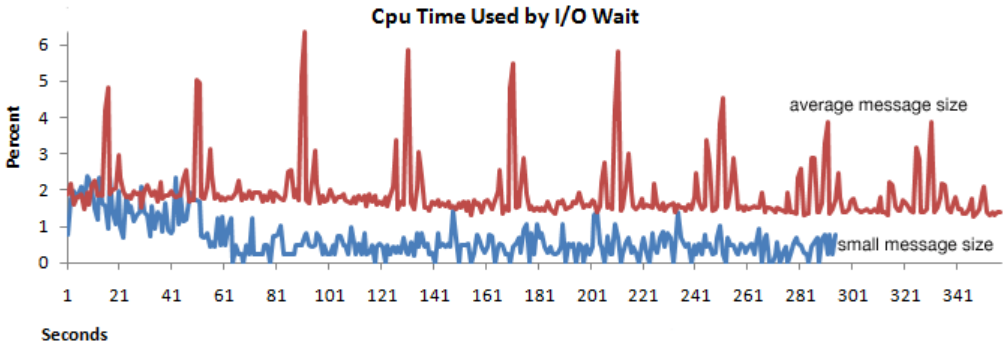

Figure 4.116: Thunderbird Default Behavior Disk I/O Wait CPU Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

4.5.5 3.4 vs 76 Kbyte Message Size Comparison for Thunderbird's optimized Behavior

This section presents the plotted graph results from comparison between Thunderbird's optimized behaviors for 3.4 Kbyte vs. 76 Kbyte messages sizes.

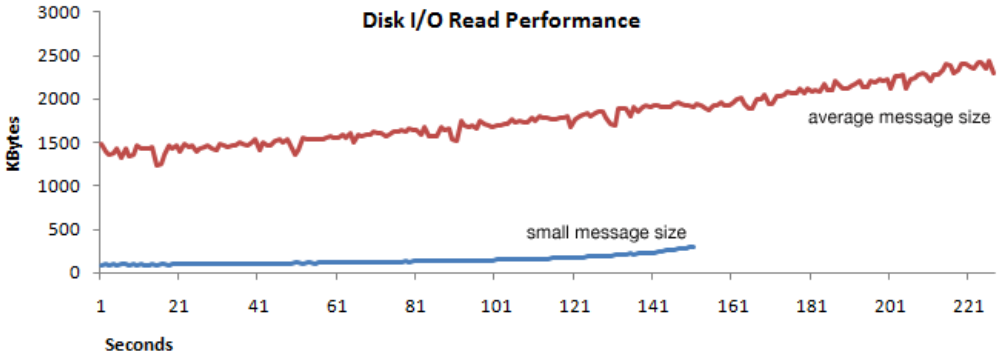

Figure 4.117: Thunderbird Optimized Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

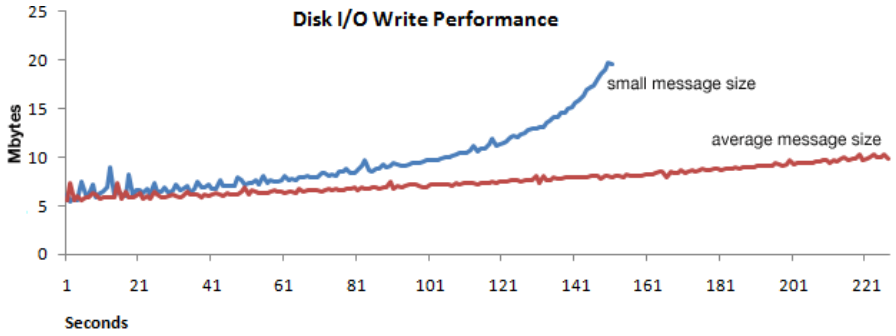

Figure 4.118: Thunderbird Optimized Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

The results in this subsection confirm the previous trends where the 76 Kbyte message size required substantially more resources, especiallforin disk I/O read.

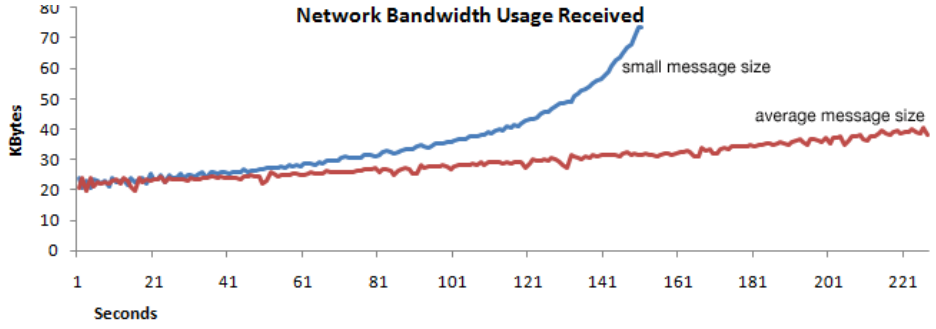

Figure 4.119: Thunderbird Optimized Behavior Network Bandwidth Usage (Packets Received) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

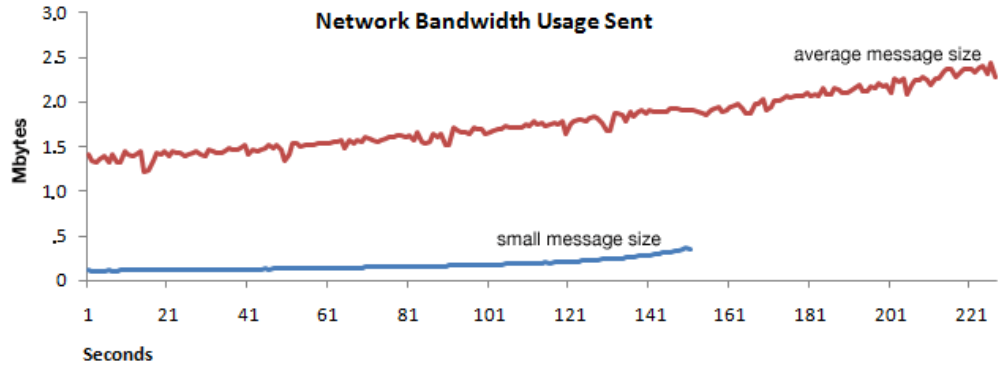

Figure 4.120: Thunderbird Optimized Behavior Network Bandwidth Usage (Packets Sent) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

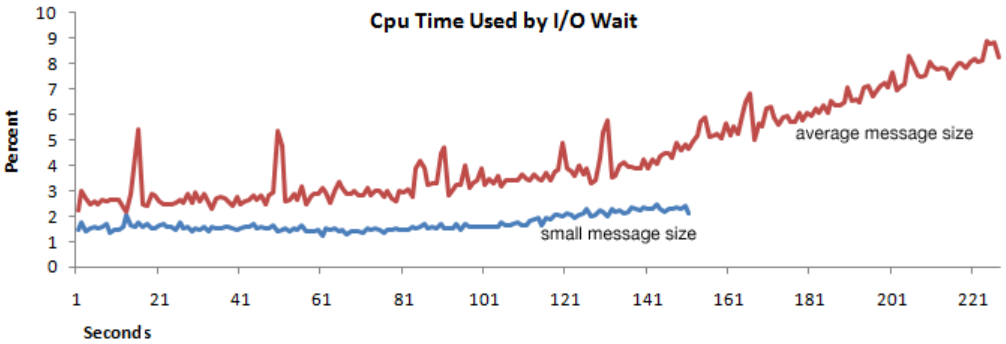

Figure 4.121: Thunderbird Optimized Behavior Disk I/O Wait CPU Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

4.6 Opera

4.6.1 Opera Default Behavior: 3.4 Kbyte Message Size

The following graphs show Opera's default behavior for the 3.4 Kbyte message size. Figures 4.122, 4.123, 4.124, 4.125, 4.126 and 4.127 shows Opera's default behavior server side resource usage for disk I/O read and write performance, network bandwidth usage for packets received and sent, and system interrupts and context switching.

Figure 4.128 shows CPU time usage by user and system, and CPU idle and wait time. The graphs are plotted from per-point means of 35 replications.

Table 4.129 shows the calculated correlation coefficient values for this client's default behavior for all measured performance metrics.

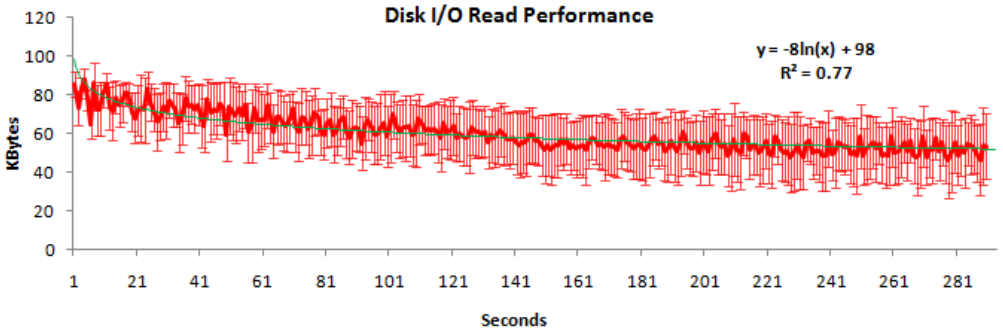

Figure 4.122: Opera Default Behavior Disk I/O Read Performance: 3.4 Kbyte Message Size

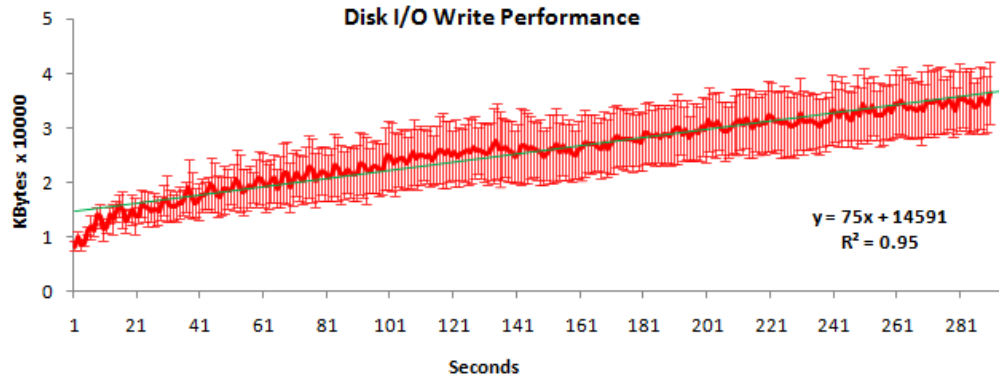

Figure 4.123: Opera Default Behavior Disk I/O Write Performance: 3.4 Kbyte Message Size

Opera has implemented the IMAP protocol differently than other clients. Some commands' observed behavior varies substantially in response to changes in the statuses and amount of messages in message folders. For example, the

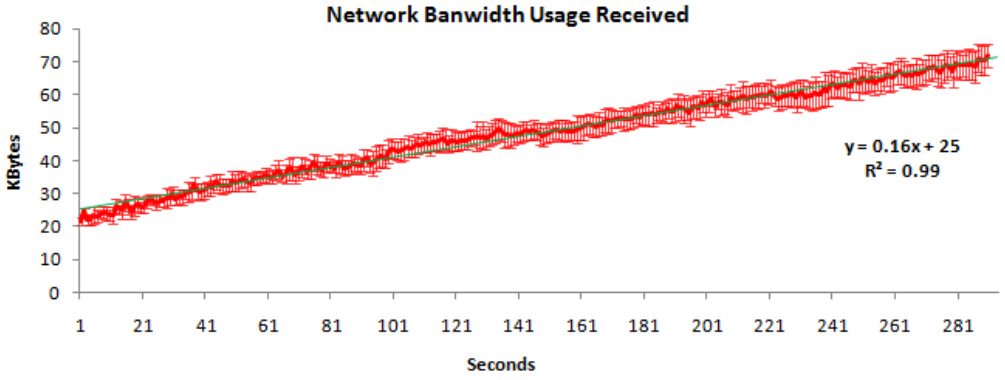

Figure 4.124: Opera Default Behavior Network Bandwidth Received Performance: 3.4 Kbyte Message Size

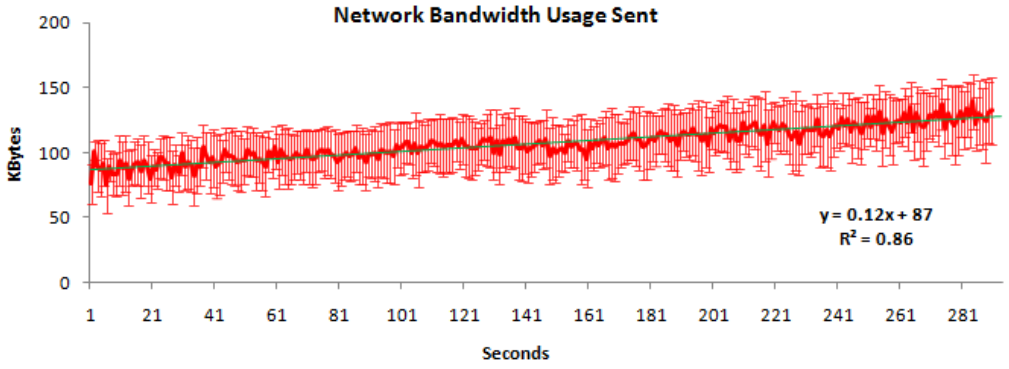

Figure 4.125: Opera Default Behavior Network Bandwidth Sent Performance: 3.4 Kbyte Message Size

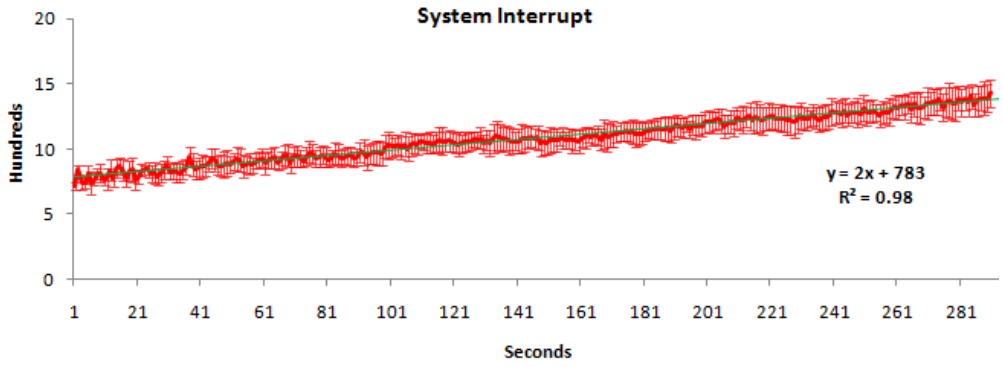

Figure 4.126: Opera Default Behavior System Interrupts: 3.4 Kbyte Message Size

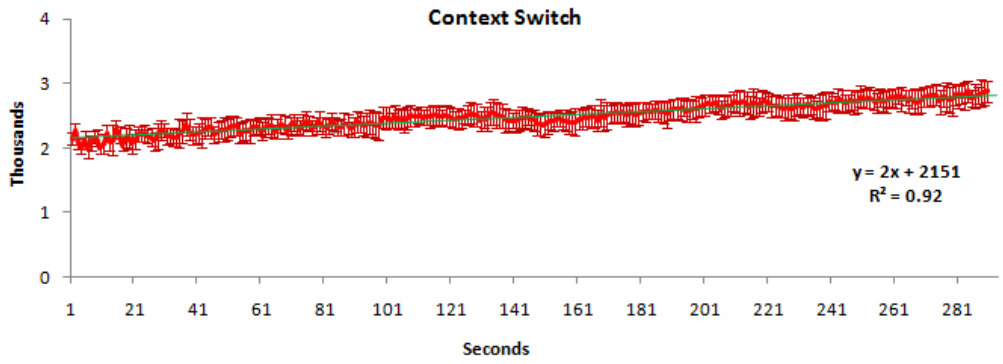

Figure 4.127: Opera Default Behavior Context Switches: 3.4 Kbyte Message Size

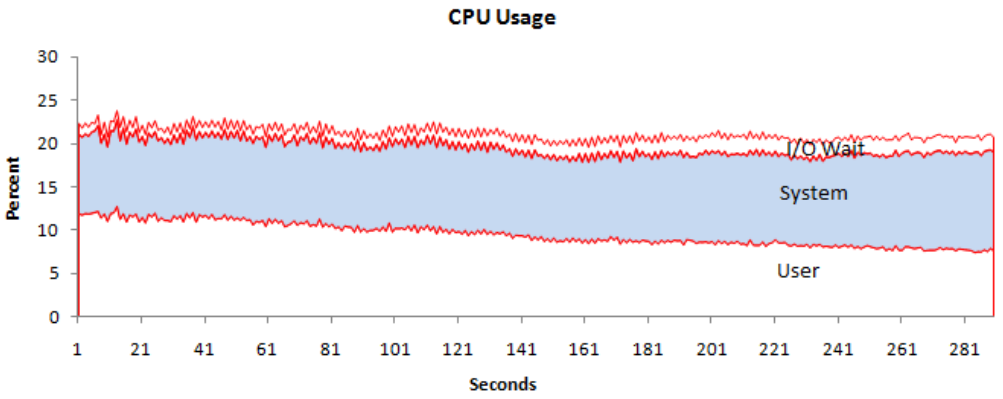

Figure 4.128: Opera Default Behavior CPU Usage Performance: 3.4 Kbyte Message Size

client provides a different approach as to how to handle a deleted message. Since it does not permanently delete all messages with a deleted flag unless the user demands it, Opera distinguishes a deleted message in a pseudo[1] Trash box from a permanently moved message to another folder just by switching between -FLAGS and +FLAGS. This switching method is implemented by flagging messages in the pseudo Trash box with -FLAGS so that the messages existed in users local machine Trash box but are not permanently removed with other messages. This technique becomes progressively more performance intensive as the number of messages in the Trash box increases because the client has to search those messages' UIDs and set -FLAGS for Deleted before issuing any command that searches and/or removes messages with +FLAGS for Deleted.

This switching between +FLAGS and -FLAGS on Deleted messages has advantages and disadvantages. In this thesis experiment, the advantage was dual. First, the client minimized the total amount of messages left at the end of each replication since it does not copy deleted messages from the Inbox to another Trash box unlike Thunderbird and Sylpheed (see Figures 5.9 and 5.9). Thus with Opera the total amount of messages in the Inbox were not duplicated to another folder for the sake of management, saving disk space at the end of the experiment. Secondly, Opera's approach avoids the frequent use of the disk intensive COPY command to move messages from the Inbox to the Trash folder.

The other unique implementation choice made by Opera is that the client provides a feature to permanently delete a message before being read if the user does not want to read it. This feature is available always so that whenever a user receives a spam message in the Inbox, it is possible to permanently remove it without opening it. In this thesis experiment, Opera minimized the amount of messages in the Inbox by approximately 20 percent at the end of the experiment because of this feature.

The combined effect of Opera's behavior from the preceding three factors is demonstrated by a significant linear increase trend (slope 75) in disk I/O write performance (see Figure 4.123). While the amount of messages decreased by 20 percent with every message group manipulation, contributing to fast access of messages in an Inbox, an increased and frequent change of message status (-FLAGS and +FLAGS) contributed to many more system calls and high disk I/O demands.

CPU usage was generally low apart from a slight increase in I/O wait CPU time and a corresponding decrease in user CPU time. This is due to small decrease in the amount of messages in mailboxes as the experiment progresses (see 4.128).

[1]The Trash box exists only on users local machine as folder but it does not exists on server side because the messages exist in Inbox with Deleted flags.

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	-0.89	1								
recv	-0.89	0.99	1							
send	-0.72	0.92	0.93	1						
int	-0.85	0.98	0.99	0.96	1					
csw	-0.82	0.98	0.97	0.96	0.98	1				
usr	0.90	-0.95	-0.96	-0.87	-0.95	-0.91	1			
sys	-0.64	0.80	0.83	0.81	0.84	0.85	-0.70	1		
idl	-0.68	0.60	0.60	0.49	0.57	0.51	-0.77	0.11	1	
wai	-0.77	0.84	0.84	0.77	0.83	0.81	-0.85	0.59	0.55	1

Figure 4.129: Opera Default Behavior Correlation Between Performance Metrics: 3.4 Kbyte Message Size

4.6.2 Simulation Issues for the Opera Client

During the implementation of this client, it was not possible to adapt the dynamic nature of the client's behavior in a straightforward manner. The approximate simulation of this client's activity was implemented through a significant amount of additional, separate commands which Opera is able to implement via a single client command. This necessity resulted in the increased network bandwidth usage trend for both received and sent packets, and it is difficult to discuss the traffic (see Figures 4.124 and 4.125) and the very high slope linear increase trend in disk I/O write performance from the perspective of the client's actual behavior.

The strong correlation shown in Figure 4.129 is also misleading due to the above mentioned simulation problem.

Unfortunately, for these reasons, the simulation implementation for Opera's default behavior is somewhat distorted, and the trends showed in the results graph may not reflect the actual default behavior of the client.

4.6.3 Opera Optimized Behavior for the 3.4 Kbyte Message Size

The following plotted graphs shows Opera's optimized behavior for 3.4 Kbyte message size. Figures 4.130, 4.131, 4.132, 4.133, 4.134 and 4.135 shows Opera's optimized behavior server side resource demand for disk I/O read and write performance, network bandwidth usage for packets received and sent, system interrupts and context switching (respectively). Figure 4.136 shows the CPU time usage by user and system. It also shows the CPU time in percent when it in the I/O wait state. Figure 4.137 shows the correlation between the performance metrics chosen for this thesis experiment.

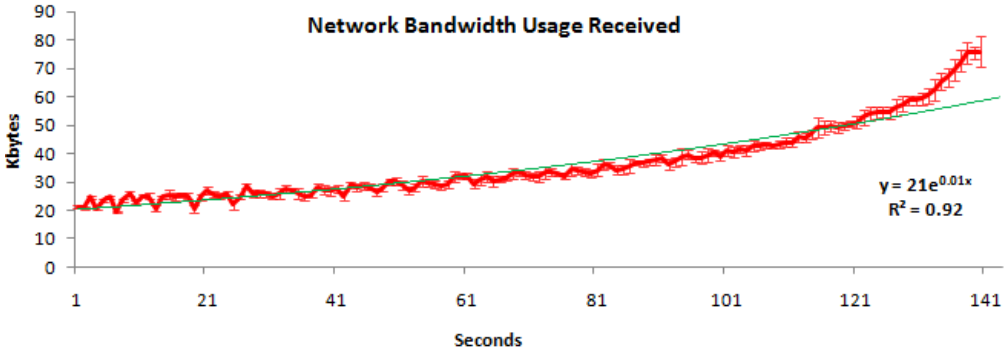

Figure 4.130: Opera's Disk I/O Read Performance for 3.4 Kbyte Message Size plotted for Client's Optimized Behavior

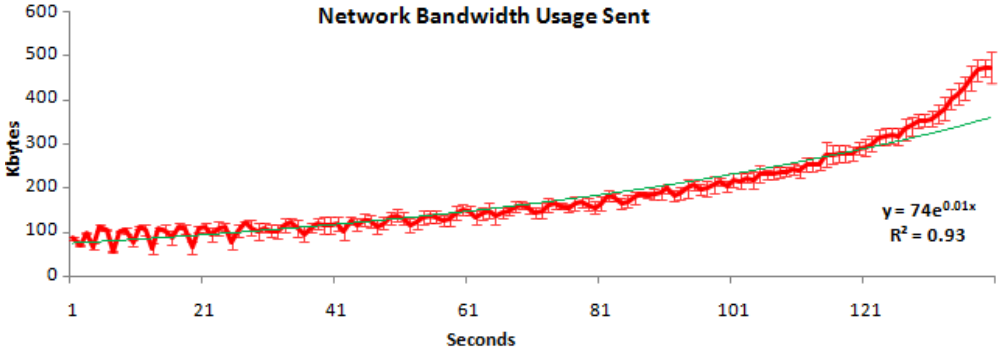

Figure 4.131: Opera's Disk I/O Write Performance for 3.4 Kbyte Message Size plotted for Client's Optimized Behavior

All measure performance metrics except CPU usage showed a similar increase in slope with time. This an increase in the values of the performance metrics is due to the rapid decline in the amount of messages in Inbox and -FLAGS and +FLAGS switching whenever there was a need to run the EXPUNGE command.

The simulation problem was solved in the optimized behavior of this client. This is because emptying the pseudo Trash message box at the end of each

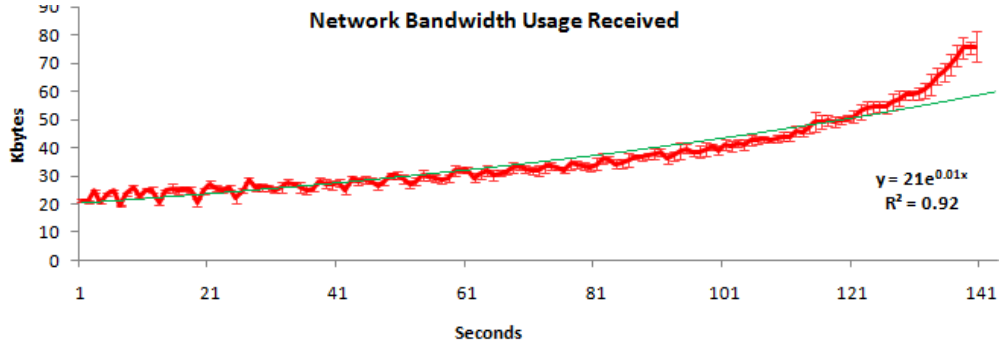

Figure 4.132: Opera's Network Bandwiddth Usage for Received Packets for 3.4 Kbyte Message Size plotted for Client's Optimized Behavior

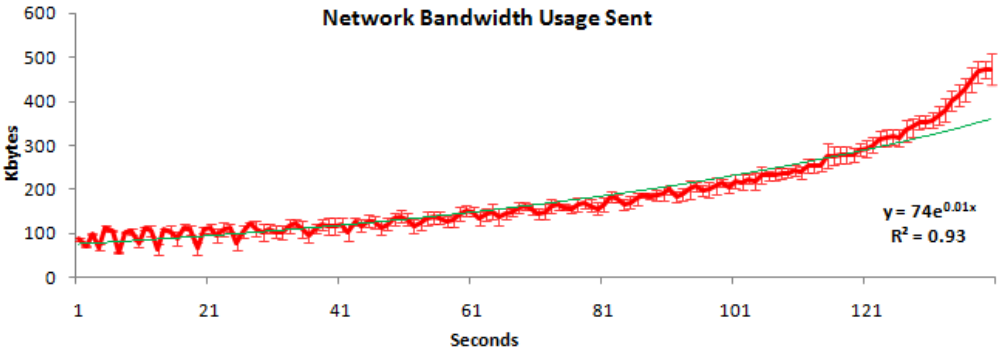

Figure 4.133: Opera's Network Bandwiddth Usage for Sent Packets for 3.4 Kbyte Message Size plotted for Client's Optimized Behavior

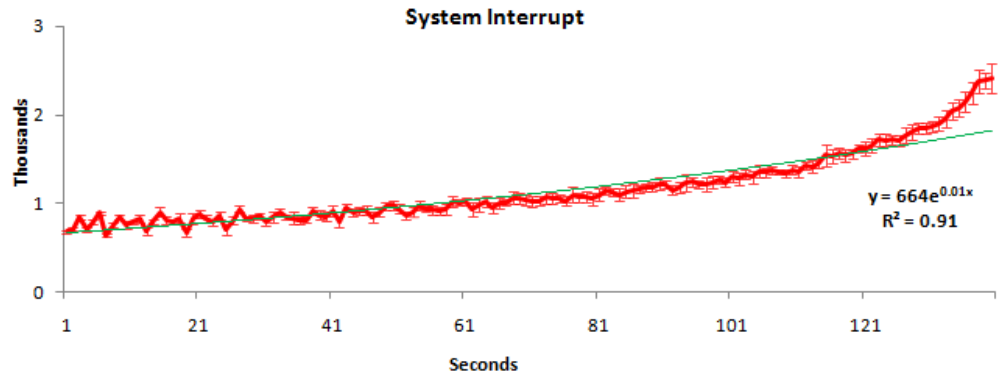

Figure 4.134: Opera's System Interrupt for 3.4 Kbyte Message Size plotted for Client's Optimized Behavior

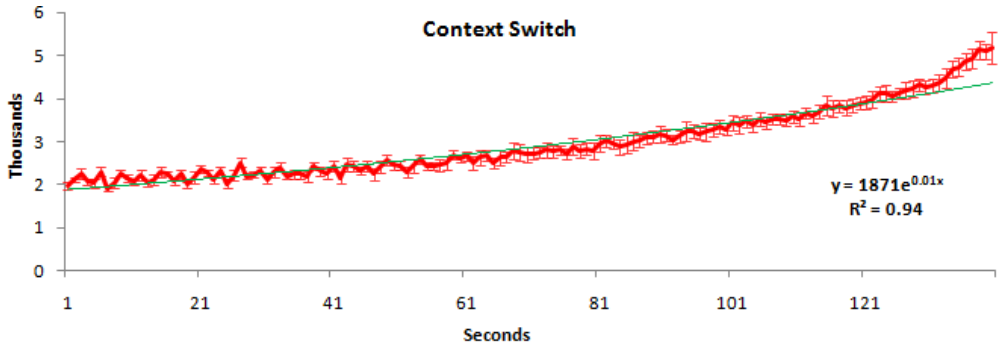

Figure 4.135: Opera's Context Switch for 3.4 Kbyte Message Size plotted for Client's Optimized Behavior

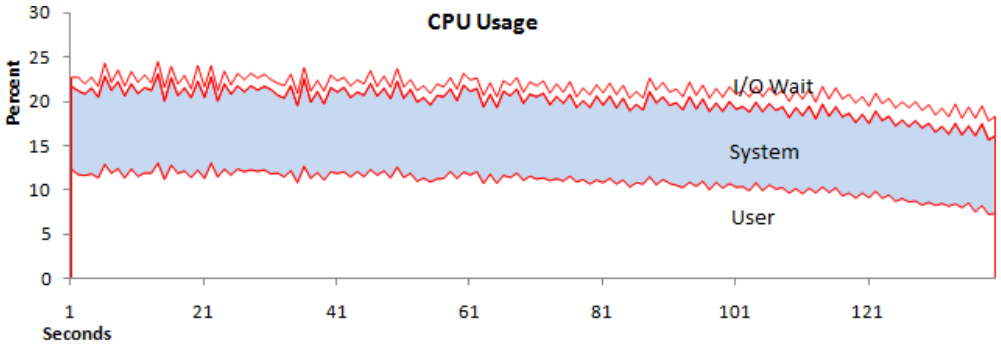

Figure 4.136: Opera's CPU Usage for 3.4 Kbyte Message Size plotted for Client's Optimized Behavior

message group manipulation occurs in the optimized configuration, avoiding the dynamic nature of the client behavior in handling deleted messages beyond each message group manipulation intended for the simulation experiment. The strong correlation coefficient value in Figure 4.137 is good testimony to the direct relationship between the performance metrics.

	read	writ	recv	send	int	csw	usr	sys	idl	wai
read	1									
write	0.94	1								
recv	0.99	0.96	1							
send	0.99	0.95	1.00	1						
int	0.99	0.96	1.00	1.00	1					
csw	0.98	0.95	0.99	0.99	0.99	1				
usr	-0.92	-0.90	-0.94	-0.94	-0.94	-0.93	1			
sys	-0.51	-0.49	-0.54	-0.54	-0.54	-0.53	0.71	1		
idl	0.77	0.75	0.80	0.80	0.80	0.79	-0.94	-0.89	1	
wai	0.88	0.85	0.87	0.87	0.87	0.88	-0.79	-0.35	0.57	1

Figure 4.137: Correlation Coefficients Calculated for Relationships Between Measured Performance Metrics for Opera's Optimized Behavior: 3/4 Kbyte Message Size

4.6.4 Opera's Default vs Optimized Behavior for 3.4 Kbyte Message Size

In this subsection, the comparison between Opera's default and optimized behavior for disk I/O and network bandwidth usage should be presented. The problem in simulating the dynamic nature of the client's IMAP protocol implementation does not allow a scientifically valid comparison between the default and optimized behaviors. However, the plots were prepared in the same manner as for the other clients for consistency, but comparisons must be made with extreme caution. The first two graphs are combined disk I/O read and write performance measured for Opera's default and optimized behaviors. The last two are for network bandwidth usage corresponding to packets received from and sent to the IMAP server.

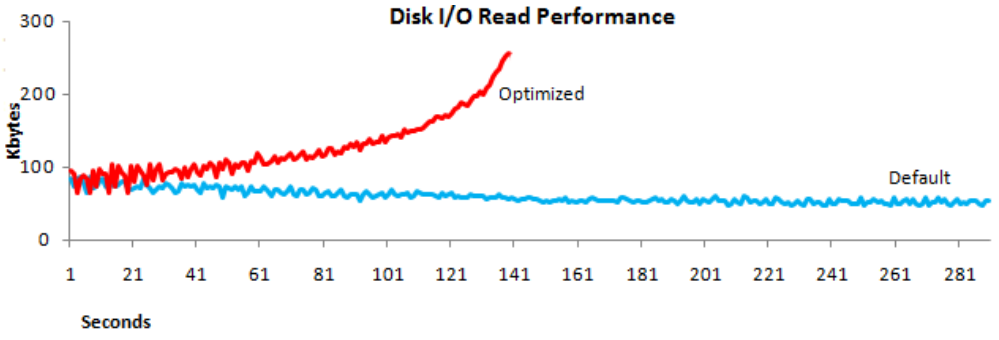

Figure 4.138: Opera's Default vs Optimized Disk I/O Read Performance: 3/4 Kbyte Message Size

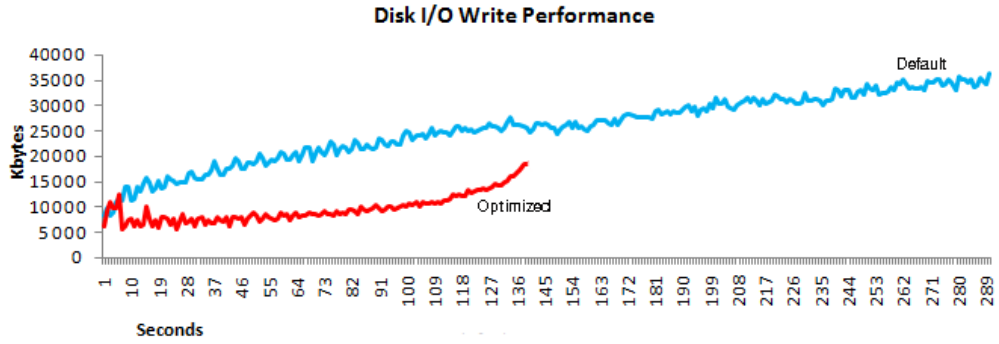

Figure 4.139: Opera's Default vs Optimized Disk I/O Write Performance: 3/4 Kbyte Message Size

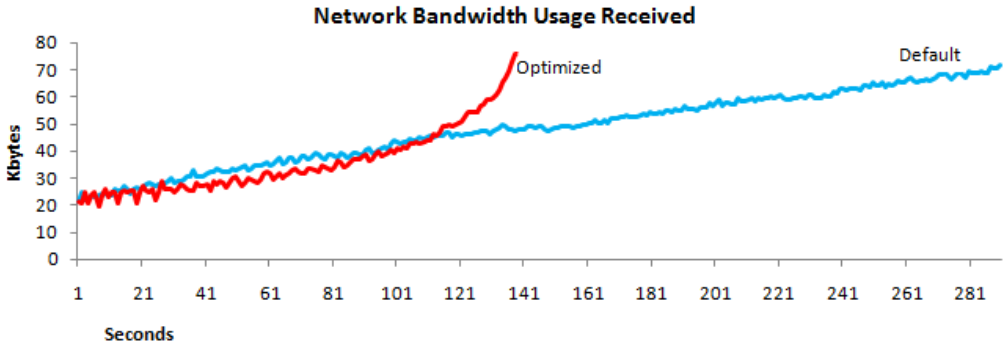

Figure 4.140: Opera's Default vs Optimized Network Bandwidth Usage, Packets Received: 3/4 Kbyte Message Size

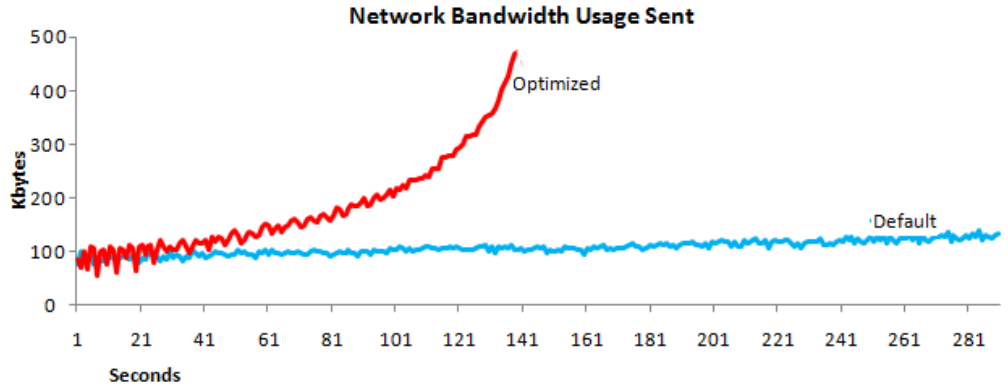

Figure 4.141: Opera's Default vs Optimized Network Bandwidth Usage, Packets Sent: 3/4 Kbyte Message Size

4.6.5 3.4 vs 76 Kbyte Message Size Comparison for Opera Default Behavior

This section presents the plotted graph results from comparisons between Opera's default behaviors for 3.4 Kbyte vs. 76 Kbyte messages sizes. These graphs are included for consistency purposes. However, this comparison is based on default behavior of the client which is considered to be invalid in this thesis, these results should be viewed with extreme caution.

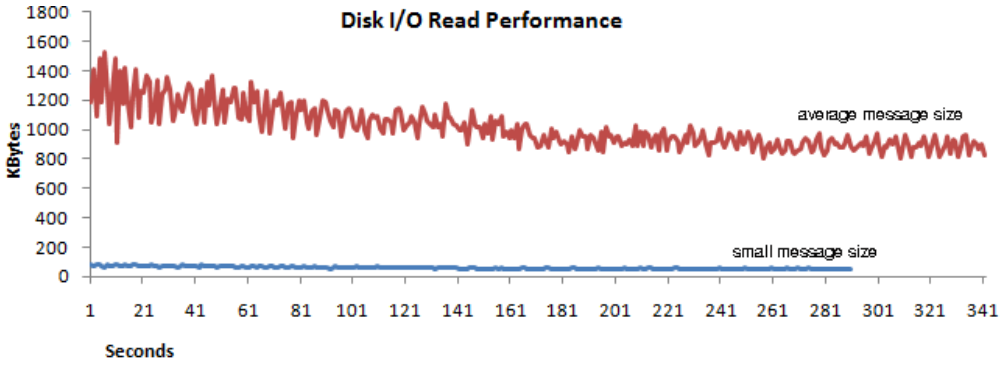

Figure 4.142: Opera Default Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

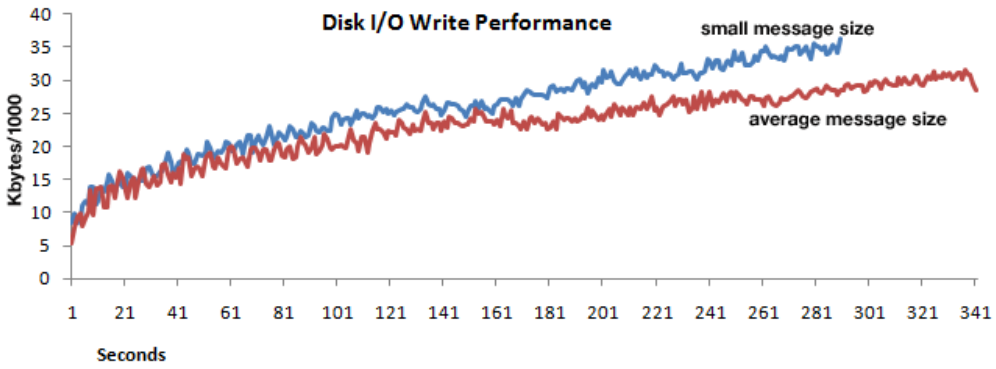

Figure 4.143: Opera Default Behavior Disk I/O Write Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

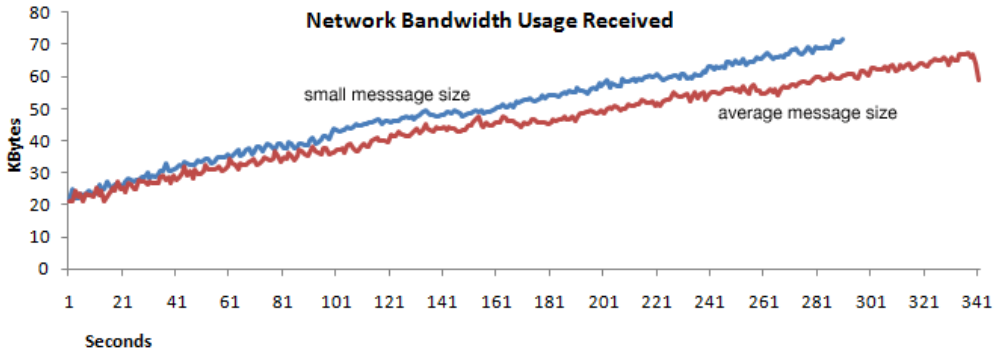

Figure 4.144: Opera Default Behavior Network Bandwidth Usage (Packets Received) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

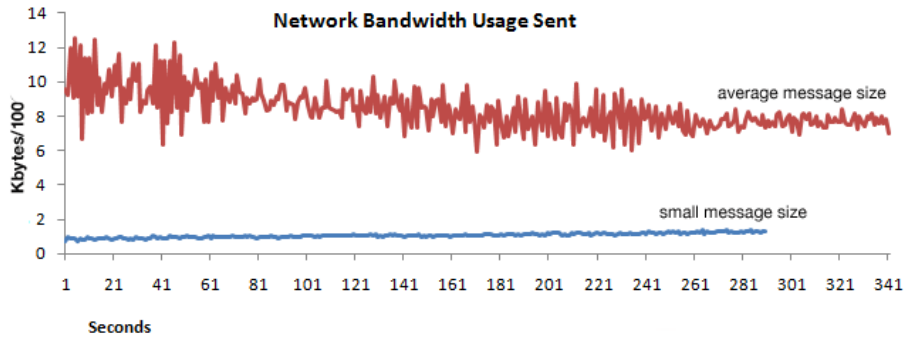

Figure 4.145: Opera Default Behavior Network Bandwidth Usage (Packets Sent) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

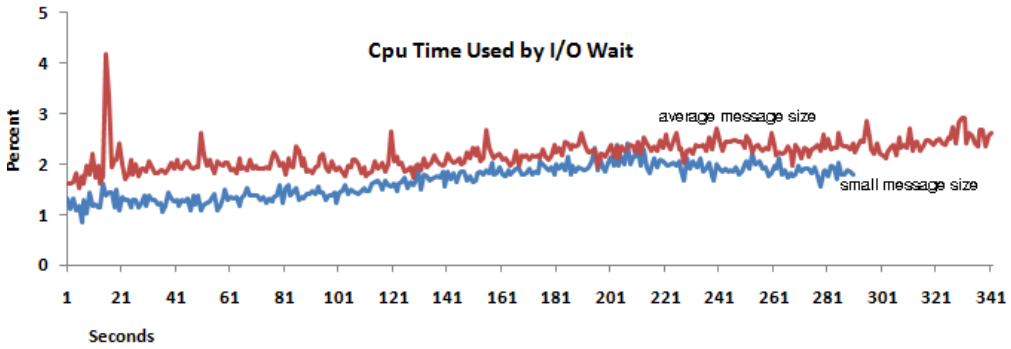

Figure 4.146: Opera Default Behavior Disk I/O Wait CPU Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

4.6.6 3.4 vs 76 Kbyte Message Size Comparison for Opera Optimized Behavior

This section presents the plotted graph results from comparison between Opera's optimized behaviors for 3.4 Kbyte vs. 76 Kbyte messages sizes. Figures 4.147, 4.148, 4.149, 4.150, and 4.151 present the 76 Kbyte message size results (in comparison with the corresponding results for the Kbyte message size) for disk I/O read and write performance, incoming and outgoing network bandwidth, and I/O wait CPU time (respectively).

The high resource consumption as the message size increased to 76 Kbytes seen in other clients is also applicable to Opera.

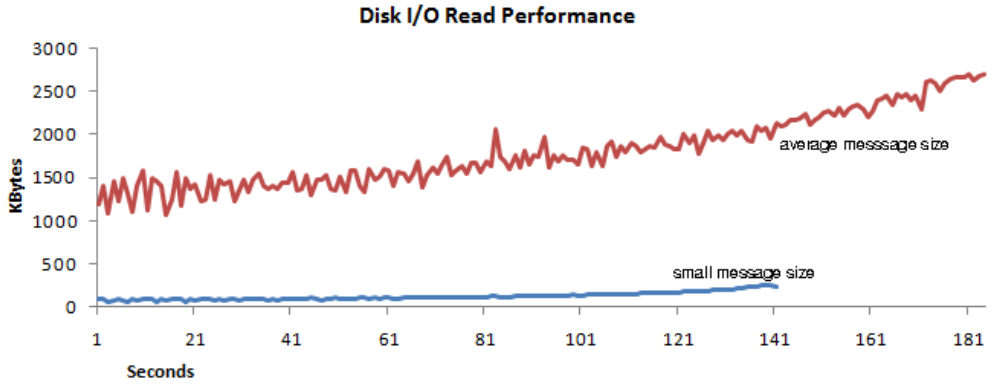

Figure 4.147: Opera Optimized Behavior Disk I/O Read Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

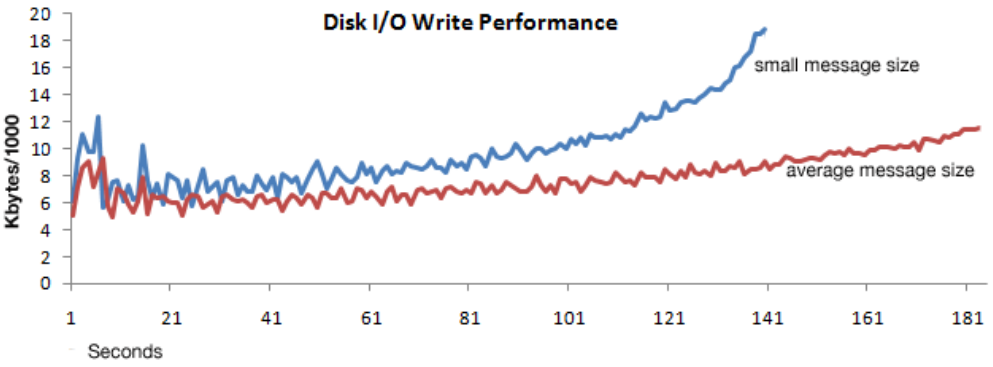

Figure 4.148: Opera Optimized Behavior Network Bandwidth Usage (Packets Received) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

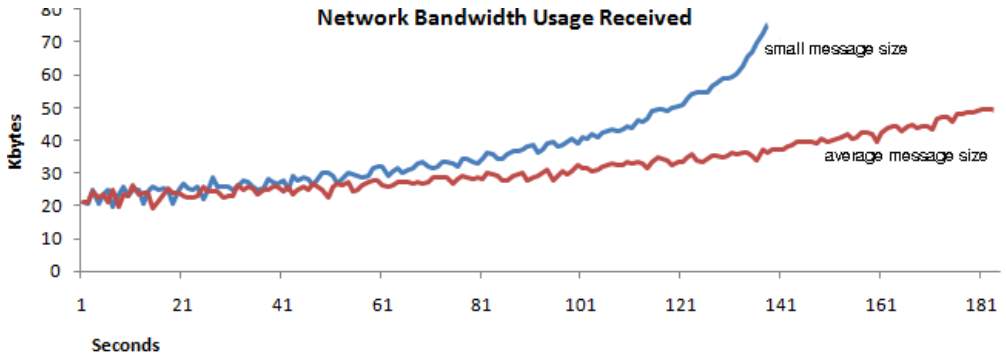

Figure 4.149: Opera Optimized Behavior Network Bandwidth Usage (Packets Received) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

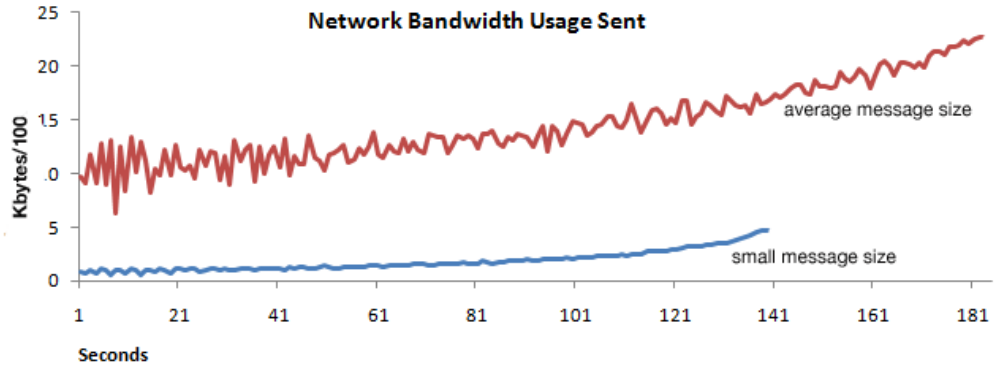

Figure 4.150: Opera Optimized Behavior Network Bandwidth Usage (Packets Sent) Performance Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

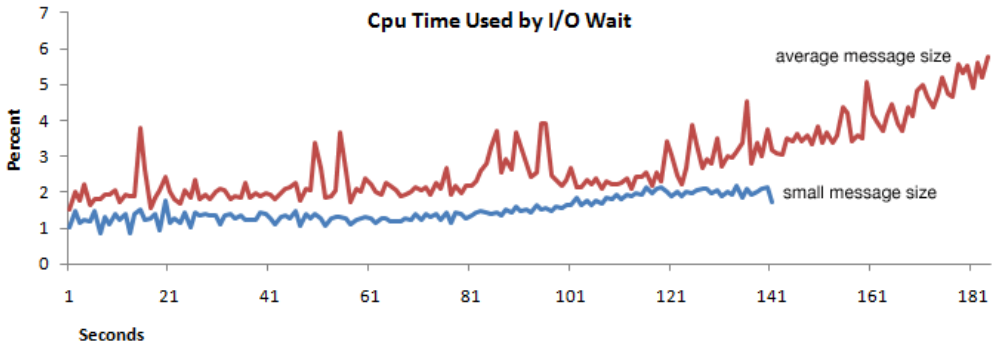

Figure 4.151: opera Optimized Behavior Disk I/O Wait CPU Comparison for 3.4 Kbyte vs. 76 Kbyte Message Sizes

Chapter 5

Discussion and Analysis

This chapter discusses and further analyzes the results presented in the previous chapter. The first section discusses the common performance issues shared by all selected clients while experimenting. Subsequent sections compare the five clients under the relevant scenarios.

5.1 Common Performance Issues and Trends for All Clients

As it has been elaborated in the methodology chapter, Maildir was selected as the message box format for the experiments conducted in this thesis. Although Maildir was developed mainly to support multiple concurrent access by different applications to avoid file locking[14], it has some performance overhead issues.

The major performance bottleneck in this message box format is that the entire directory where the messages resides must be scanned and read to perform any modification or access to an existing file under it[18].

A mail message is stored as a file in a mail directory. As such, it is always checked for existence by executing the stat(2) Unix system call. This makes the file system inefficient because the entire file in the directory must be read for every status change of a message. This means that the frequent demand from clients to change status of a message results in significant performance overhead because the files (which are messages in this case) need to be renamed after every status change request[18].

In this experimental set up, a shared mailbox scenario was implemented. Therefore, there were a frequent rescan of the mailbox directories, especially the Inbox, where significant message manipulation taken place.

Thus, if there are several system calls for several messages access and manipulation, this will be reflected in an increased amount of context switching

and system interrupts [33]. Moreover, as the number of messages in Maildir subdirectories increases, disk I/O becomes an important performance issue for some IMAP protocol commands that demand frequent access and status change of messages.

Network bandwidth usage for packets received and sent is the other performance metric which has a direct relationship to disk I/O performance. The Expect programming language which was used to simulate the clients' behavior is capable of building an interactive session between client and server[30] and [31]. This means the program does not issue the next request unless the server responds for the previous request.

However, the scripts are designed for instant command exchange, so that the speed of the disk I/O is directly proportional to the speed of the network bandwidth usage in both directions. Therefore, if the disk I/O is slow or fast by any reason while reading or writing to disk, then the network bandwidth also becomes correspondingly slow or fast because the script issues the next command as it receives and check the response for the previous one.

Another common performance issue is the IMAP commands' resource demands. Some IMAP commands are disk I/O intensive, and others are network bandwidth intensive. Consider the functioning of the FETCH command. First of all, the FETCH command slows disk I/O as the number of messages increase in the message folder because it must scan and search for specific messages based on their UIDs to meet the clients' request. Secondly, the FETCH command can be issued by client software for different purposes. The common ones observed during this experiment are to fetch the envelope, header, body, UID, UID and flags, and flags (as explained in the background chapter).

In most cases, if the client issued FETCH command to look for the status of each messages flags, then it is disk I/O read intensive and does not consume much network bandwidth since the server responds with a summary. However, if the command is issued to fetch the body part of the messages, then the command is disk I/O read intensive and also consumes significant network bandwidth, depending on the size of the message. Thus, as the number of messages in the message box increases, othe resource demands from FETCH command also increase.

STORE command is another important command with regards to resource consumption. In most cases, the command is issued to change the status of a message flag. As discussed above, status change has a dual resource demand. First, the message has to be located, which requires little disk I/O read resource consumption. Second, the file name (the message in this case) has to be changed as the Maildir implementation enforces this to reflect the status change. Therefore, there will be a lot of disk I/O write resource requirements, depending on the amount of messages that are going to be affected.

Another disk I/O intensive command is COPY. Although most clients provide

for moving messages from one message folder to another, in typical implementations, they just COPY the message and then delete from the old folder.

Other commands like IDLE and NOOP are only network bandwidth intensive because they do not affect messages in message boxes.

5.2 Analysis of the 3.4 Kbyte Message Size, Default Behavior Experiments

The following figures display graphs that compare the performance of the five client programs for the various performance metrics being considered for their default operation modes and using the smaller, 3.4 Kbyte message size.

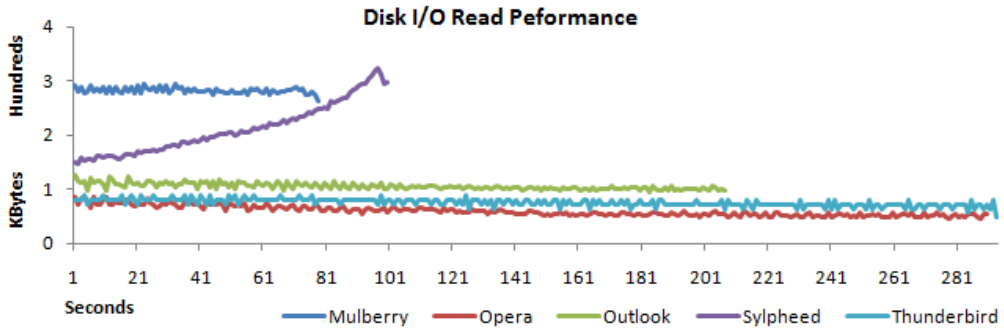

Figure 5.1: Disk I/O Read Performance, 3.4 Kbyte Message Size: Default Behavior

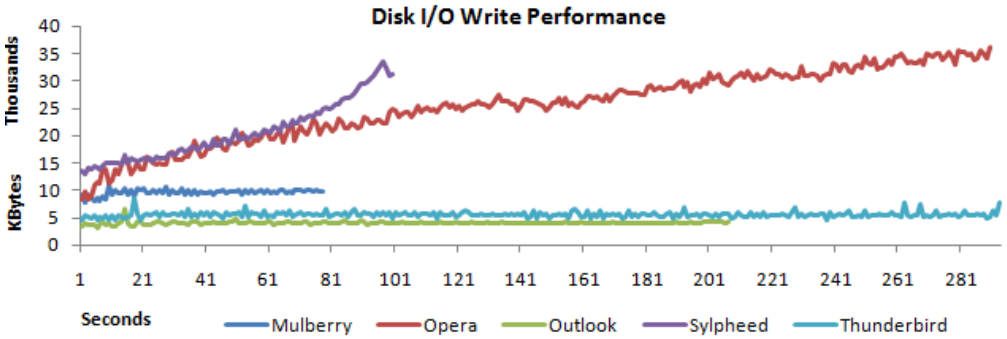

Figure 5.2: Disk I/O Write Performance, 3.4 Kbyte Message Size: Default Behavior

From the preceding results, one can clearly see the significant difference between clients' resource requirements on IMAP server. In disk I/O read, Mulberry completed the task quickly with high resource consumption. Sylpheed showed a low start but finishes quicker than Outlook, Thunderbird and Opera. Thunderbird and Opera took a longer time to complete manipulating the 2800 messages, followed by Outlook.

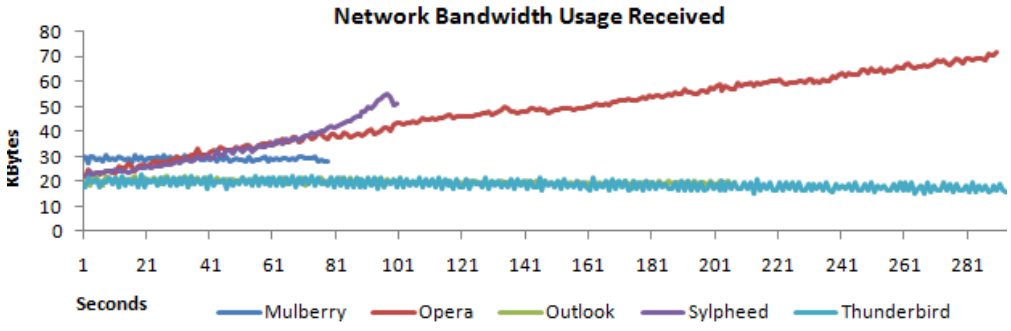

Figure 5.3: Incoming Network Bandwidth, 3.4 Kbyte Message Size: Default Behavior

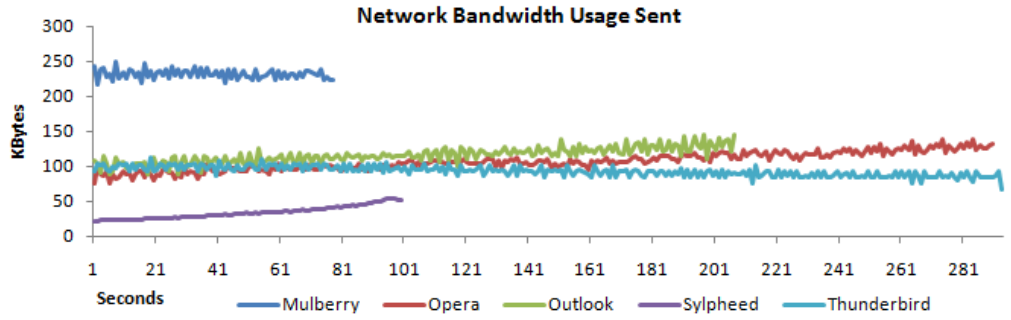

Figure 5.4: Outgoing Network Bandwidth, 3.4 Kbyte Message Size: Default Behavior

When this trend is analyzed in combination with Table 5.6, the sum of total disk I/O read needed to complete the tasks does not show much difference between clients. Mulberry's efficiency is confirmed because it completes the tasks with lowest total amount of disk I/O write (754 MB) and network bandwidth usage: 2 MB and 18 MB for received and sent performance respectively. Outlook is the second best in disk I/O write (844 MB) and network bandwidth usage: 4 MB and 24 MB for packets received and sent respectively followed by Thunderbird. Sylpheed consumed much more disk I/O write resources (2149 MB) and network bandwidth, both for packets sent (35 MB) and received (5 MB).

Opera's behavior simulation problem is clearly seen here, and the values are invalid since the performance is too much exaggerated. However, the frequent use of STORE command by the client and switching of message flags could contribute to the performance because it requires frequent renaming of file names.

Although Sylpheed showed consistency in its default and optimized behavior, a frequent request for status of messages in all available folders using STATUS command might be the case in addition to high number of CLOSE and EXPUNGE commands as shown in Table 5.5.

Thunderbird showed the second worst in disk I/O write resource usage because of the highest amount of STORE command used (265) compared to the others.

As explained earlier, since the disk I/O performance is directly proportional to network bandwidth usage, the higher amount of bandwidth usage for Sylpheed followed by Thunderbird is self explanatory. In the same context, the system interrupt and context switch trends also followed the trend and agreed with disk I/O performance as shown in Table 5.6.

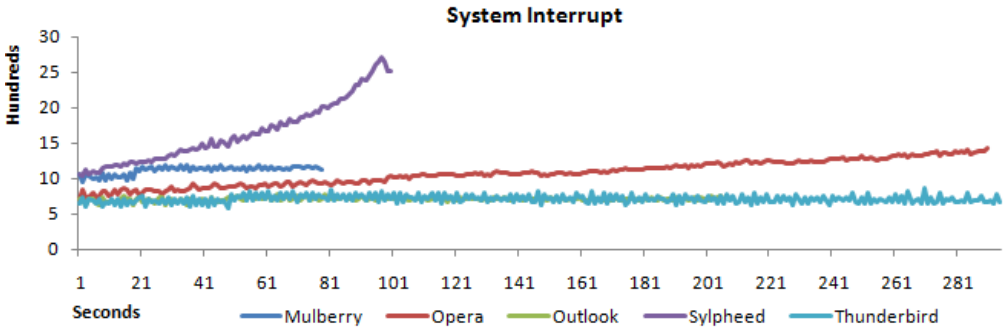

Figure 5.5: System Interrupts, 3.4 Kbyte Message Size: Default Behavior

In Figure 5.7, Sylpheed's performance showed the highest CPU I/O wait time usage. This is the direct reflection of Sylpheed's worst performance in disk I/O read. Thunderbird and Outlook showed approximately comparable results in I/O wait time. Mulberry started with highest I/O wait time usage but showed

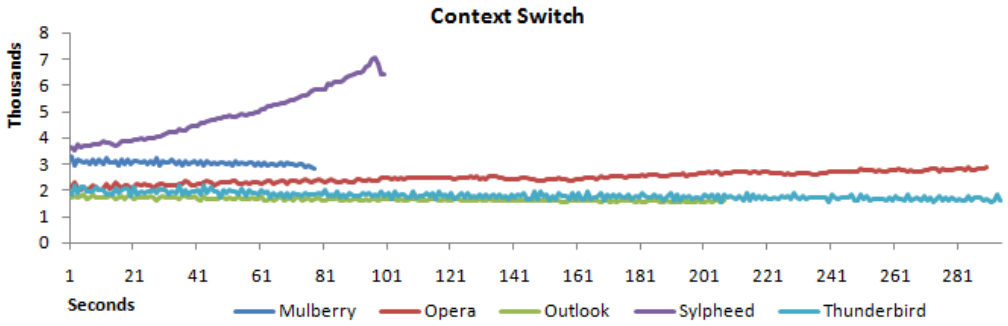

Figure 5.6: System Context Switches, 3.4 Kbyte Message Size: Default Behavior

a dramatic decline over time. Opera's I/O wait time usage is in the midde with slight increase over time.

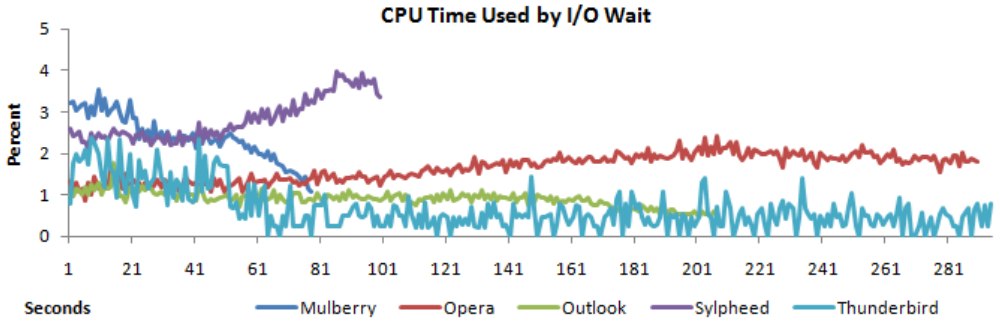

Figure 5.7: I/O Wait, 3.4 Kbyte Message Size: Default Behavior

Figures 5.8 and 5.9 show the size of message boxes and the amount of messages remaining after the completion of each replication. These result shows the disk space usage differences between clients. From Table 5.2, Opera mail utilizes the least disk usage (11.29 MB), followed by Sylpheed (12.78 MB). Thunderbird is the worst (25.54 MB), followed by Outlook (18.25) and Mulberry (17.68 mB). This is because Opera cleaned deleted messages from the Inbox as the experiment went on. These result are the direct reflection factors noted in discussion of the previous chapter.

Opera's best performance with this regard is because of its spam permanent deletion feature without read from the Inbox. Opera also does not copy deleted messages to the Trash. Sylpheed's performance is due to its ability to copy deleted messages from the Inbox and clear them immediately. Outlook and Mulberry leave deleted messages in the Inbox until users clean them. Thunderbird is the worst because it keeps all copied deleted messages in the Trash and Inbox, and messages exist duplicated in to message folders. Table 5.1 provides additional evidence for these conclusions.

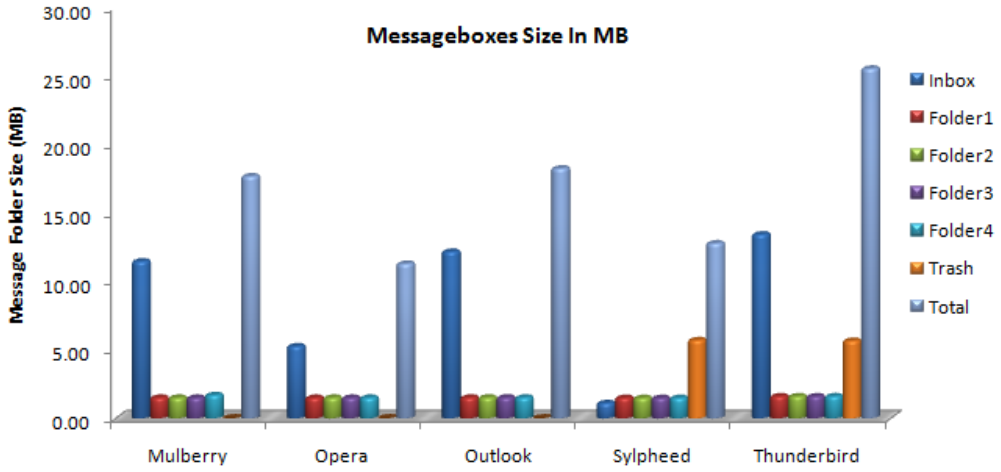

Figure 5.8: Message Box Size After Manipulation of 2800 Messages: Default Behaviour

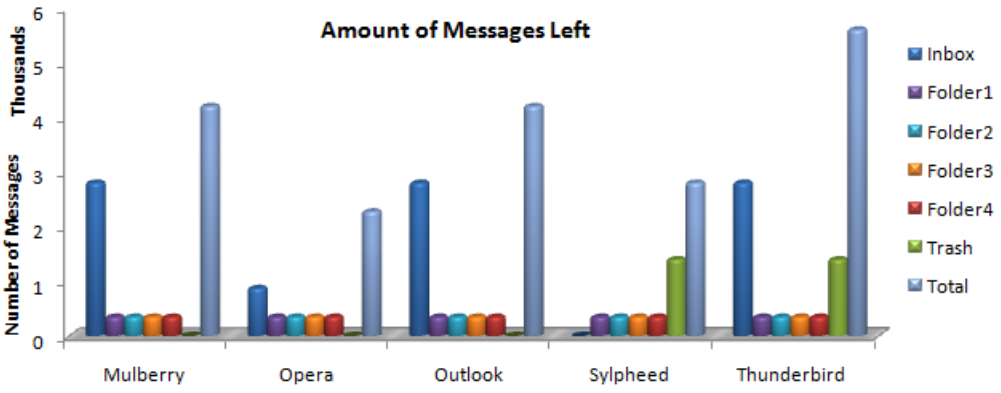

Figure 5.9: Messages Left After Manipulation of 2800 Messages: Default Behavior

	Inbox	Folder1	Folder2	Folder3	Folder4	Trash	Total
Mulberry	2800	350	350	350	350	0	4200
Opera	875	350	350	350	350	0	2275
Outlook	2800	350	350	350	350	0	4200
Sylpheed	0	350	350	350	350	1400	2800
Thunderbird	2800	350	350	350	350	1400	5600

Table 5.1: Number of Messages Left After Manipulation of 2800 Messages: 3.4 Kbyte Message Size, Default Behavior

	Inbox	Folder1	Folder2	Folder3	Folder4	Trash	Total
Mulberry	11.46	1.51	1.50	1.50	1.70	0.00	17.68
Opera	5.26	1.50	1.51	1.50	1.51	0.00	11.29
Outlook	12.17	1.52	1.52	1.52	1.52	0.00	18.25
Sylpheed	1.11	1.50	1.49	1.49	1.49	5.71	12.78
Thunderbird	13.43	1.61	1.61	1.61	1.61	5.66	25.54

Table 5.2: Message Box Sizes (MB) After Experiment Completion: 3.4 Kbyte Message Size, Default Behavior

5.3 Analysis of the 3.4 Kbyte Message Size, Optimized Behavior Experiments

The following figures display graphs that compare the performance of the five client programs for the various performance metrics being considered for their optimized operation modes and using the smaller, 3.4 Kbyte message size.

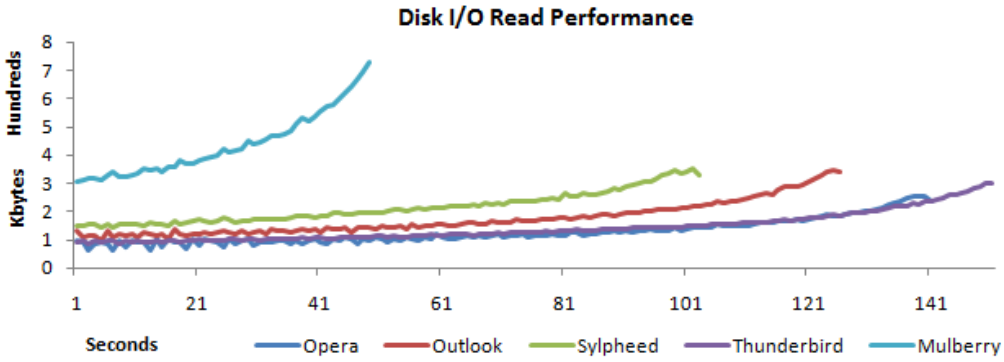

Figure 5.10: Disk I/O Read Performance, 3.4 Kbyte Message Size: Optimized Behavior

Figures 5.10,5.11,5.12, 5.13, 5.14 and 5.15 compare disk I/O read and write performance, network bandwidth usage for packets received and sent, and system interrupts and context switches (respectively) for the five client programs under consideration.

As for the default mode results, Mulberry again completed the tasks first by

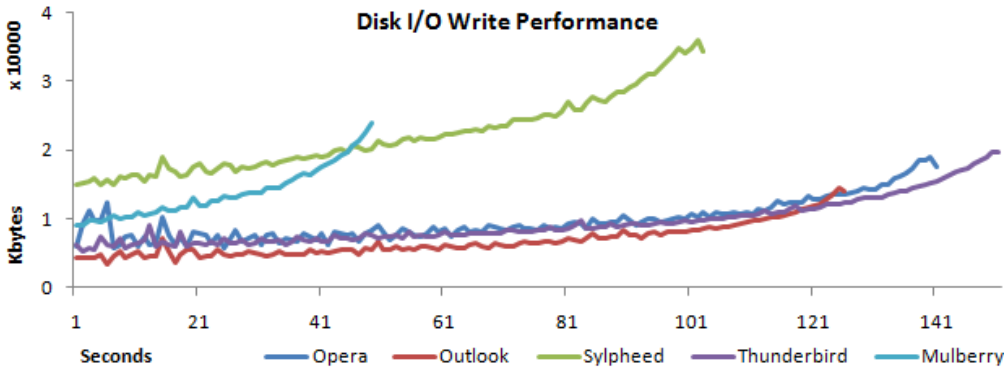

Figure 5.11: Disk I/O Write Performance, 3.4 Kbyte Message Size: Optimized Behavior

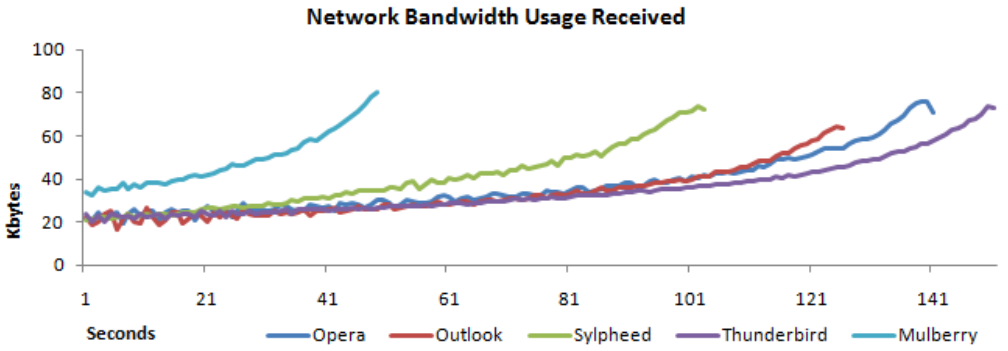

Figure 5.12: Incoming Network Bandwidth, 3.4 Kbyte Message Size: Optimized Behavior

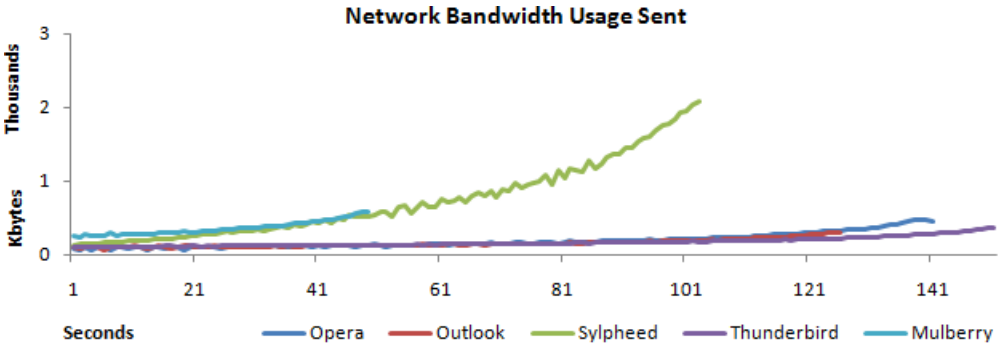

Figure 5.13: Outgoing Network Bandwidth, 3.4 Kbyte Message Size: Optimized Behavior

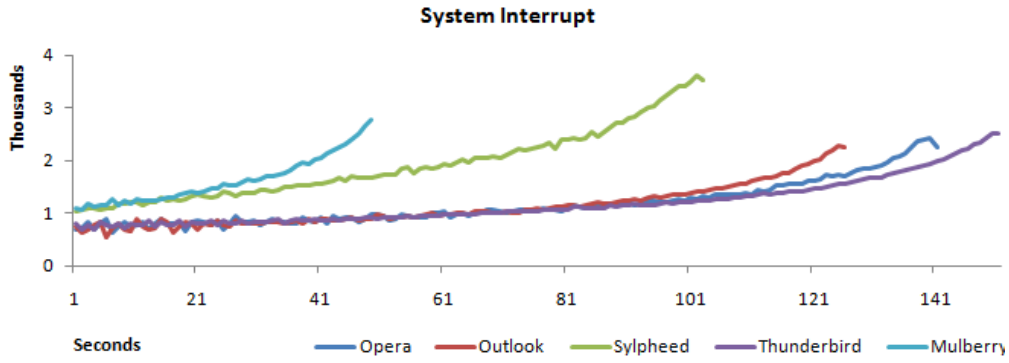

Figure 5.14: System Interrupts, 3.4 Kbyte Message Size: Optimized Behavior

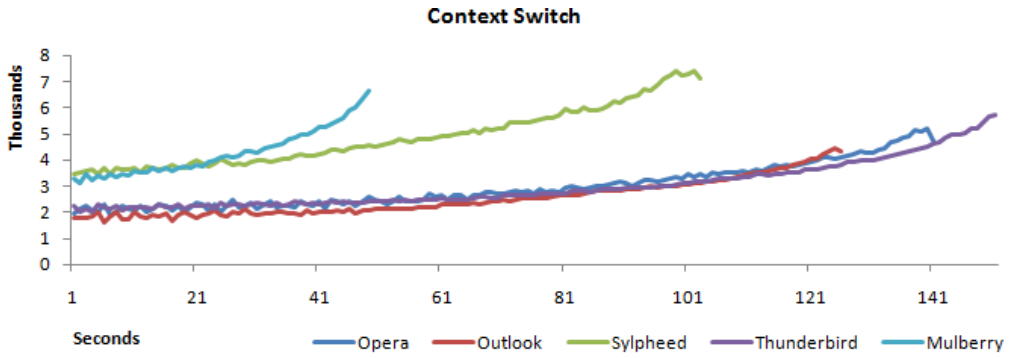

Figure 5.15: Context Switches, 3.4 Kbyte Message Size: Optimized Behavior

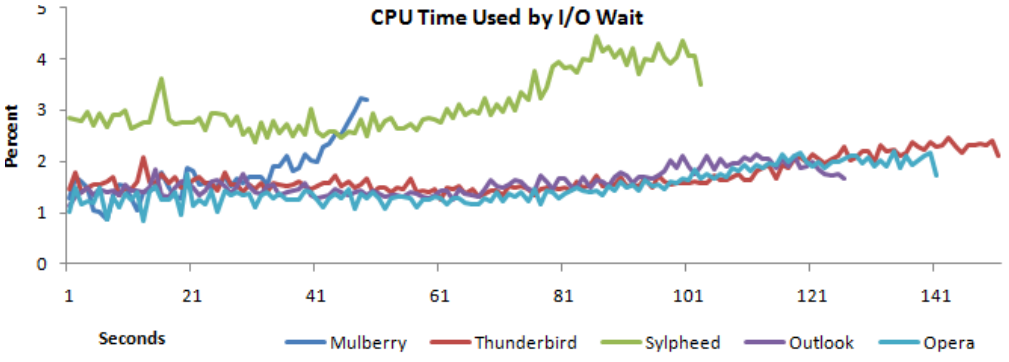

Figure 5.16: I/O Wait CPU Percentage, 3.4 Kbyte Message Size: Optimized Behavior

utilizing higher disk I/O read performance and network bandwidth for packets received. Its results were followed by Sylpheed. The other clients were essentially equivalent in their performance for disk I/O write and network bandwidth for packets sent. The experiments' slight differences in length caused Thunderbird to finish slowest, followed by Opera.

Outlook consumed higher disk I/O read resources as compared to Thunderbird and Opera, which were neck and neck except for Thunderbird's delayed finishing.

When the preceding graphs are interpreted in conjunction with Table 5.7, Mulberry was again an efficient client in resource usage because the sum of total amount of resources consumed for disk I/O write and network bandwidth usage for packets received was the lowest at 697 MB and 18 MB, followed by Outlook, which consumed 864 MB and 19 MB respectively. The number of system interrupts and context switches recorded in this table is another reflection of the disk I/O performance figures.

In Figure 5.16, Sylpheed's performance was the worst of all for CPU I/O wait time usage. This is the direct reflection of Sylpheed's worst performance in disk I/O read (explained earlier). Thunderbird, Outlook and Opera exhibited similar performance in with respect to CPU I/O wait time. Mulberry started with almost similar CPU I/O resource demands with Thunderbird, Outlook and Opera but showed a significant increase after approximately 30 seconds.

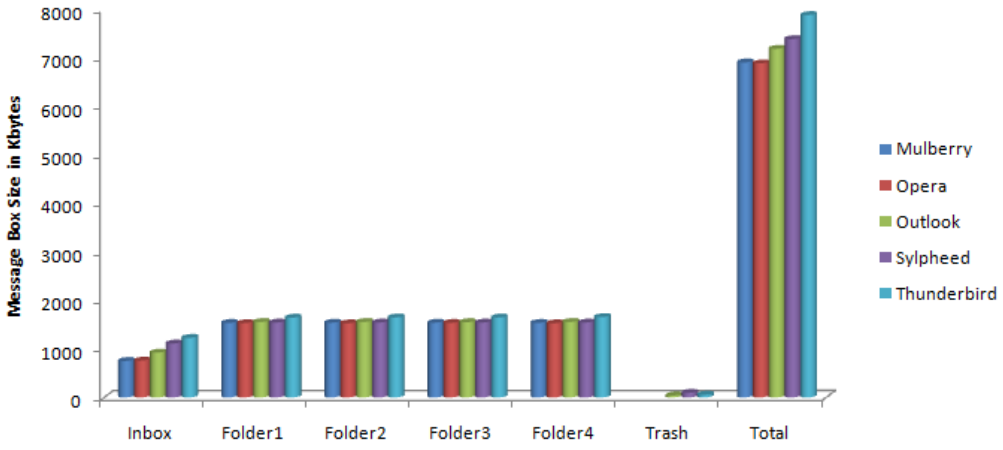

Figure 5.17: Message Box Size After Manipulation of 2800 3.4 Kbyte Messages: Optimized Behavior

Figures 5.17 and 5.18 shows the size of message boxes and amount of messages left after the completion of each replication. The corresponding numerical data is tabulated in Tables 5.3 and 5.4.

In Table 5.4, Opera mail again utilizes the least disk usage (6884 KB), followed by Sylpheed (6904 KB) and Outlook (7184 KB). Thunderbird is the worst (7880 KB). This is because the other clients cleaned deleted messages from the Inbox

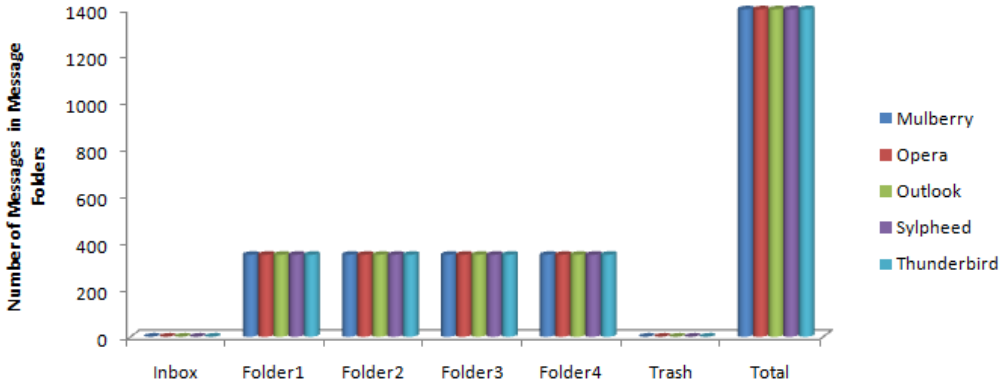

Figure 5.18: Messages Left After Manipulation of 2800 3.4 Kbyte Messages: Optimized Behavior

	Inbox	Folder1	Folder2	Folder3	Folder4	Trash	Total
Mulberry	0	350	350	350	350	0	1400
Opera	0	350	350	350	350	0	1400
Outlook	0	350	350	350	350	0	1400
Sylpheed	0	350	350	350	350	0	1400
Thunderbird	0	350	350	350	350	0	1400

Table 5.3: Messages Remaining After Experiment Completion: 3.4 Kbyte Messages, Optimized Behavior

as the experiment went on. These result are the direct reflection of what we have discussed so far.

Although clients cleaned their messages, the Inbox occupied significant disk space. This is due to Maildir's implementation that inodes are freed and considered to be reused later again. Maildir does not immediately remove deleted messages from the Inbox although the client software cleans them unless the mailbox is permanently removed.

5.4 Analysis of the 76 Kbyte Message Size, Default Behavior Experiments

The following figures display graphs that compare the performance of the five client programs for the various performance metrics being considered for their default operation modes and using the average size, 76 Kbyte message size.

From the preceding graphs, it is clear that Mulberry's finishing time is the shortest, followed by Sylpheed and Outlook. Thunderbird took the longest time to complete the experiment. Almost all clients showed constant resource

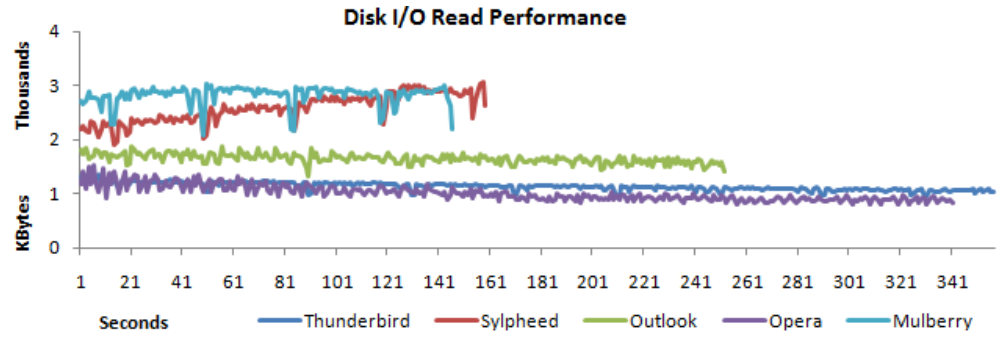

Figure 5.19: Disk I/O Read Performance, 76 Kbyte Message Size: Default Behavior

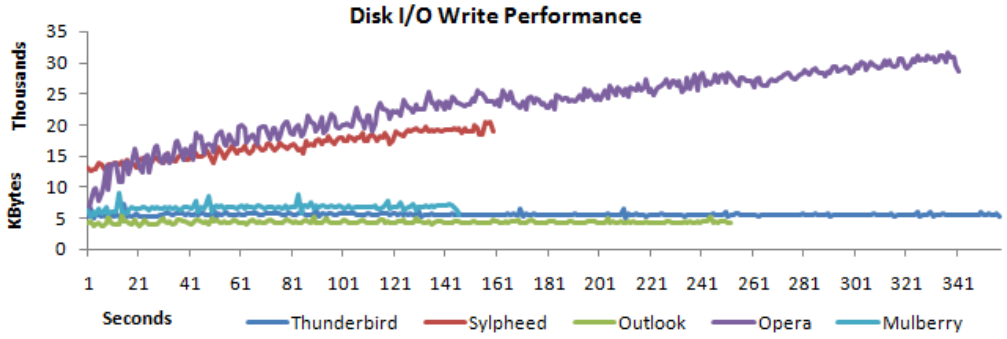

Figure 5.20: Disk I/O Write Performance, 76 Kbyte Message Size: Default Behavior

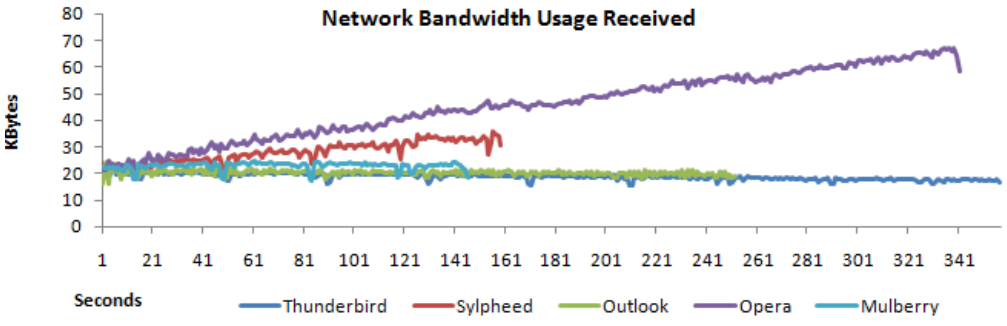

Figure 5.21: Incoming Network Bandwidth, 76 Kbyte Message Size: Default Behavior

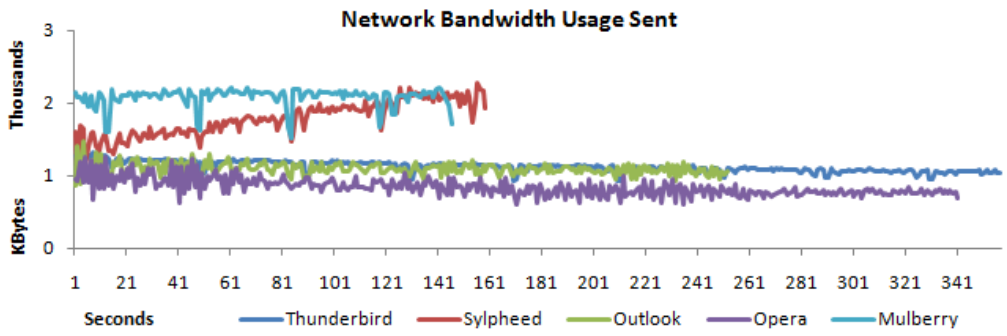

Figure 5.22: Outgoing Network Bandwidth, 76 Kbyte Message Size: Default Behavior

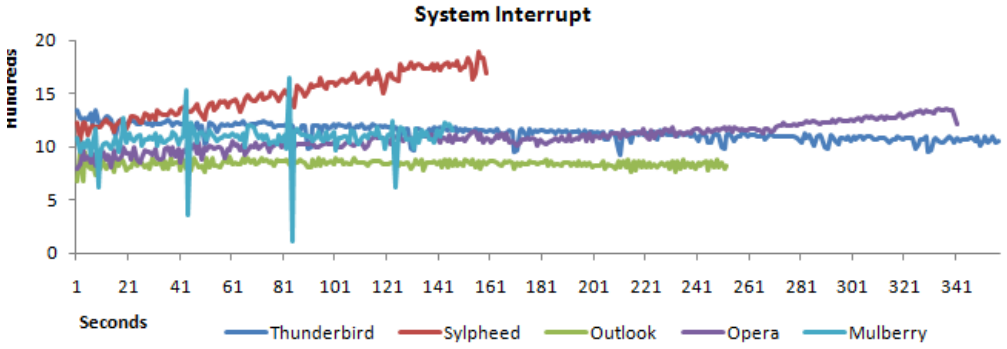

Figure 5.23: System Interrupts, 76 Kbyte Message Size: Default Behavior

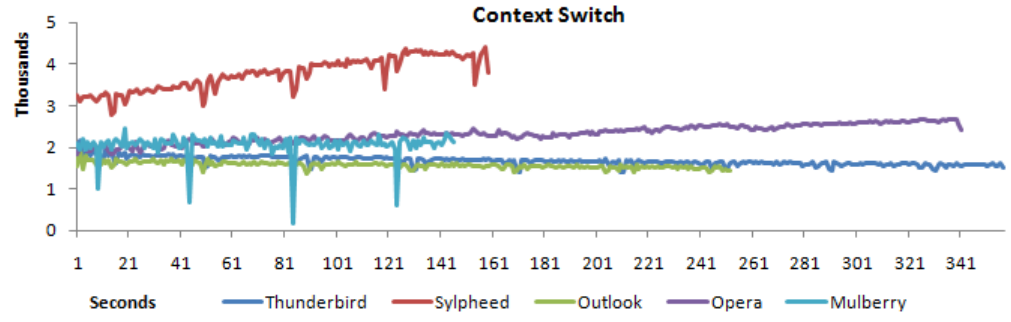

Figure 5.24: System Context Switches, 76 Kbyte Message Size: Default Behavior

	Inbox	Folder1	Folder2	Folder3	Folder4	Trash	Total
Mulberry	752	1536	1540	1540	1536		6904
Opera	760	1532	1528	1536	1528		6884
Outlook	924	1556	1556	1552	1556	40	7184
Sylpheed	1112	1544	1544	1544	1544	96	7384
Thunderbird	1228	1644	1648	1648	1656	56	7880

Table 5.4: Final Mailbox Sizes (MB): 3.4 Kbyte Messages, Optimized Behavior

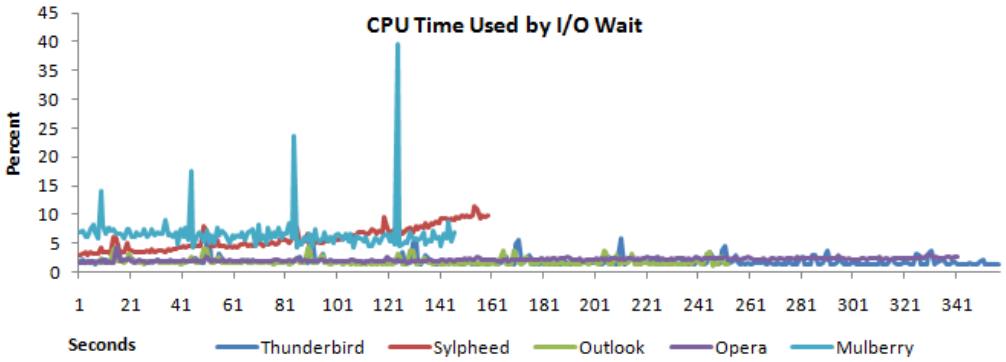

Figure 5.25: I/O Wait CPU Percentage, 76 Kbyte Message Size: Default Behavior

consumption with regards to disk I/O, network bandwidth usage and system interrupts and context switches. All clients showed downward spike on their graphs, although the spikes are not always well visible because of scaling. However, this is reflected in the CPU I/O wait (see Figure 5.25).

This could be due to the disk I/O write requirements from the increased message size. The disk I/O write graph shows an upward spike at similar locations and time intervals for each client. This behavior is most clearly seen in Mulberry's line. Since Mulberry issued a small number of commands to complete the task, disk I/O intensive commands like STORE could be issued closely together, resulting in increases in disk I/O operation. Thus, the I/O wait spikes in Mulberry are high. Except for the disk I/O write graph, the other performance metrics showed downward spikes at the same time interval. This shows that the disk I/O write caused the CPU wait time.

5.5 Analysis of the 76 Kbyte Message Size, Optimized Behavior Experiments

The following figures display graphs that compare the performance of the five client programs for the various performance metrics being considered for their optimized operation modes and using the average size, 76 Kbyte message size.

As the preceding graphs show, all performance metrics for all clients showed an increased performance as the amount of message in inbox was decreasing. From this results, one can easily see that Mulberry finished in short period of time than others as usual. Sylpheed is the next client finished the tasks with little short from Outlook. Thunderbird took the longest time to complete the tasks.

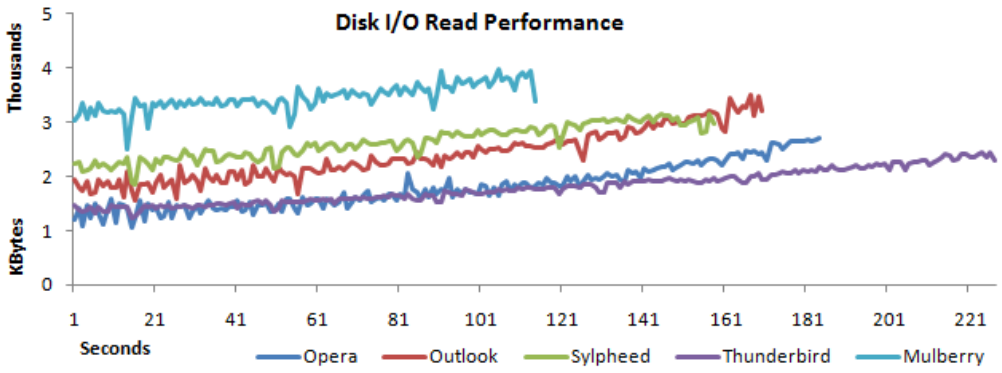

Figure 5.26: Disk I/O Read Performance, 76 Kbyte Message Size: Optimized Behavior

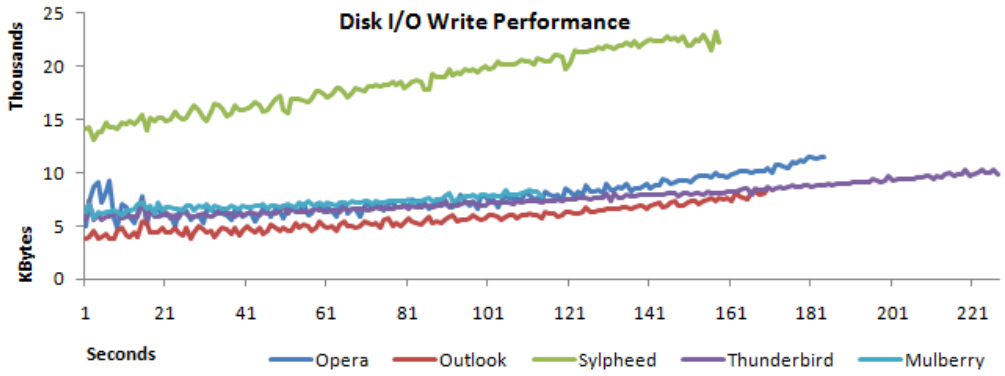

Figure 5.27: Disk I/O Write Performance, 76 Kbyte Message Size: Optimized Behavior

As in case in of the default behavior of clients for same message size, Mulberry is the highest disk I/O read resource consumer followed by Sylpheed. Outlook and Thunderbird were neck and neck except for the extended total experiment time for Thunderbird (see Figure 5.26). However, all clients except Sylpheed showed a similar level of disk I/O write performance over the length of time they took.

Other performance metrics are the reflection of these disk I/O requirements. Clients showed competitive resource demand in network bandwidth usage for packets received (see Figure 5.28). However, Mulberry consumed the highest level of network bandwidth for packets received from the server, followed by Sylpheed. This traffic could be the direct reflection of disk I/O read perfor-

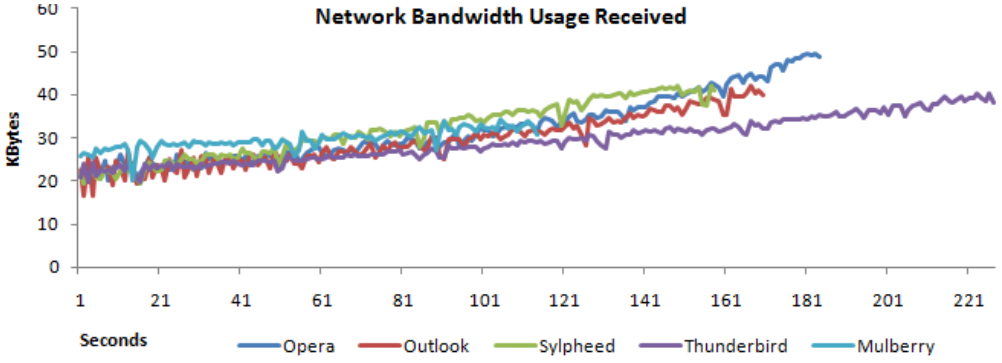

Figure 5.28: Incoming Network Bandwidth, 76 Kbyte Message Size: Optimized Behavior

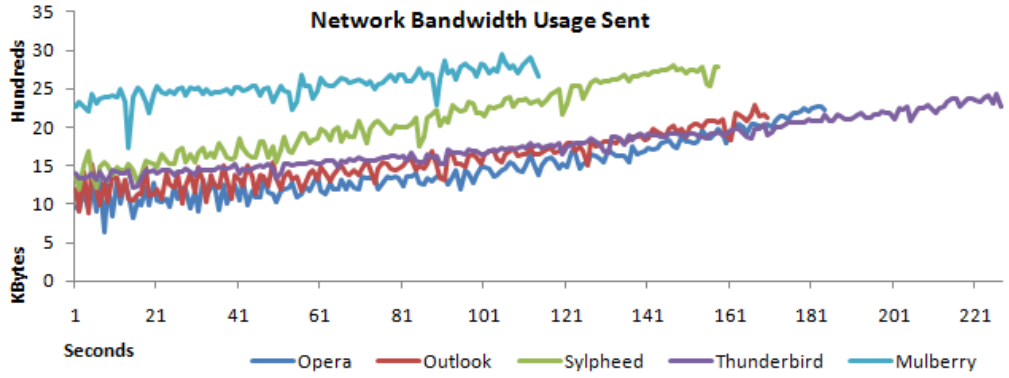

Figure 5.29: Outgoing Network Bandwidth, 76 Kbyte Message Size: Optimized Behavior

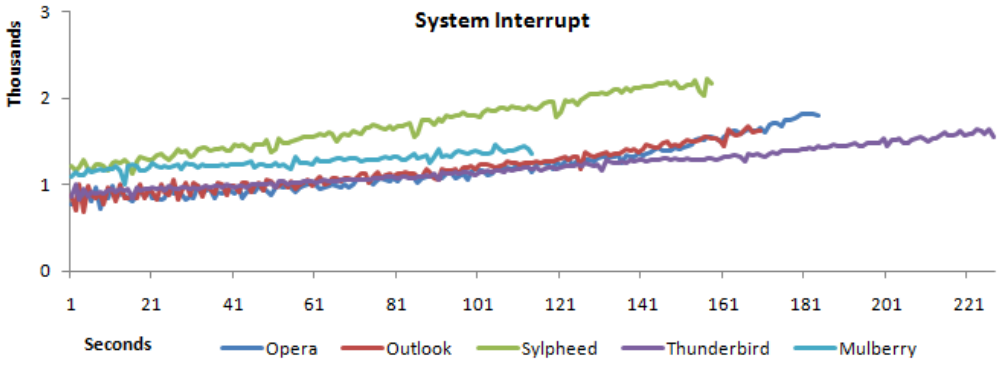

Figure 5.30: System Interrupts, 76 Kbyte Message Size: Optimized Behavior

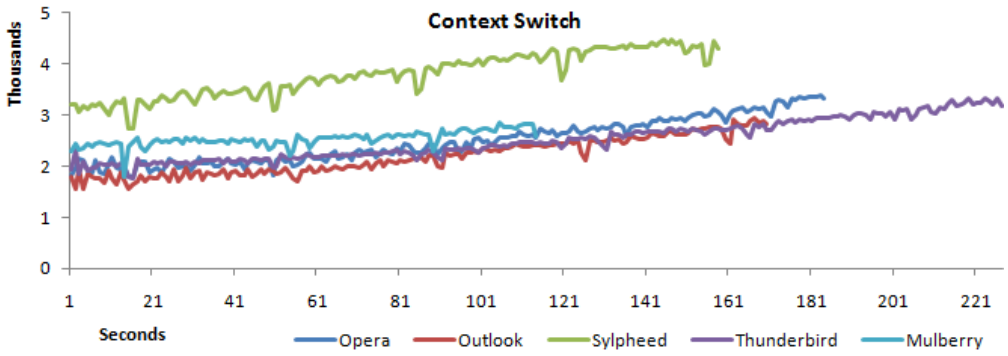

Figure 5.31: System Context Switches, 76 Kbyte Message Size: Optimized Behavior

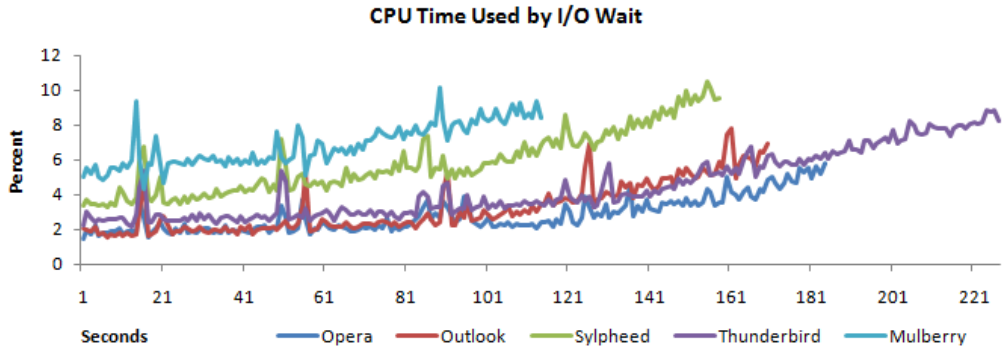

Figure 5.32: I/O Wait CPU Percentage, 76 Kbyte Message Size: Optimized Behavior

mance since the server sends responses back immediately for any disk read operation-related commands like FETCH.

CPU time wait showed in Figure 5.32 is the reflection of the performance difference between clients during disk I/O intensive operations. Mulberry again consumes hight percentage of CPU time followed by Sylpheed. Others are very close, differing only in the length of time required to complete the task.

Command	Mulberry	Outlook	Thunderbird	Sylpheed	Opera
IDLE	0	383	273	0	282
DONE (without tag)	0	383	273	0	282
idling (without tag)	0	383	273	0	282
NOOP (single way)	121	12	168	0	2
FETCH (Envelope)	120	0	0	0	0
FETCH (HEADER)	80	96	96	96	119
FETCH (BODY)	120	105	160	105	105
FETCH (UID)	0	96	0	0	0
FETCH (UID FLAGS)	0	4	0	40	39
FETCH (FLAGS)	0	0	168	0	0
COPY	40	40	80	80	40
STORE (Deleted)	80	0	0	80	180
STORE(Deleted&Seen)	0	80	81	0	0
STORE (Seen)	0	40	104	65	0
STORE (NonJunk)	0	0	80	0	0
SELECT INBOX *	5	6	5	84	81
SELECT Folder1	4	5	4	4	13
SELECT Folder2	4	5	4	4	15
SELECT Folder3	4	5	4	4	13
SELECT Folder4	4	5	4	4	14
SELECT Trash	0	0	1	1	0
SELECT Junk E-mail	0	1	0	0	0
LOGOUT	1	1	1	1	1
LOGIN	1	1	1	1	1
CAPABILITY	1	2	1	1	1
LSUB	1	2	1	0	1
LIST	0	2	1	0	119
NAMESPACE	0	0	1	1	0
STATUS INBOX	0	0	0	4	0
STATUS Folder1	0	0	0	14	10
STATUS Folder2	0	0	0	14	10
STATUS Folder3	0	0	0	14	10
STATUS Folder4	0	0	0	14	10
STATUS Trash	0	0	0	44	0
CLOSE	1	1	17	100	0
EXPUNGE	0	0	2	80	55
Total (OK)	587	1658	1803	855	1685

Table 5.5: This table shows the number of commands that each client issued for one message group manipulation round.

Client	read	writ	recv	send	int	csw	usr	sys	idl	wai	time
Mulberry	27	754	2	18	89,385	244,012	10	8	79	2	1.4
Opera	22	7,710	15	32	333,090	764,500	9	10	79	2	5.3
Outlook	26	844	4	24	151,291	352,903	12	10	77	1	3.6
Sylpheed	26	2,149	4	35	176,162	520,733	9	7	80	3	1.9
Thunderbird	26	1,617	5	27	211,389	535,518	13	9	77	1	5

Table 5.6: This table compares the total sum amount of resource usage under each selected performance metric for the default behavior of the various clients. Columns read and write show disk I/O read and write performance in MB per second. Column recv and send show the sum of total amount of network bandwidth usage for packets receved and sent in MB per second. Columns int and csw show the sum of total amount of system interrupt and context switch. Columns usr and sys show the average percentage CPU time usage for user and system time. Columns idl and wai shows the average percentage of CPU time in the idle and wait states. Column time shows the average elapsed time required to complete the experiment.

Client	read	writ	recv	send	int	csw	usr	sys	idl	wai	time
Mulberry	25	697	2	18	83,284	220,312	9	8	81	2	0.9
Opera	23	1,367	5	26	167,820	427,774	11	9	79	2	2.4
Outlook	27	864	4	19	144,593	328,273	11	9	78	2	2.2
Sylpheed	27	2,263	4	74	200,916	511,759	9	8	79	3	1.8
Thunderbird	27	1,454	5	26	186,692	471,904	10	9	79	2	2.7

Table 5.7: This table compares the total sum amount of resource usage under each selected performance metric for the optimized behavior of the various clients. Columns read and write show disk I/O read and write performance in MB per second. Column recv and send show the sum of total amount of network bandwidth usage for packets receved and sent in MB per second. Columns int and csw show the sum of total amount of system interrupt and context switch. Columns usr and sys show the average percentage CPU time usage for user and system time. Columns idl and wai shows the average percentage of CPU time in the idle and wait states. Column time shows the average elapsed time required to complete the experiment.

5.6 Summary from tables

Table 5.5 the number of commands issued from client side for one message group which was 80 in this thesis experiment.

From the table, one can easily see the difference in IMAP protocol implementation by different email client softwares under same amount of message manipulation. The figures in the table are from network traffic filter while each client's simulated scripts manipulated 80 messages. The table result indicates that Thunderbird issued 1803 commands followed by Opera which was 1685. However, the amount of commands in Opera column are increased by significant amount during simulation. Outlook issued 1658 followed by Sylpheed that amounted 855. Mulberry issued the least amount of commands.

The total amount of commands has direct relation with network bandwidth usage resource requirement for both packets received and sent from the server. The less amount of commands shown in Mulberry and Sylpheed is because they do not implement IDLE command by default.

Sylpheed issued high number of EXPUNGE, CLOSE and SELECT the Inbox commands. This is to achieve a single login implementation unlike other clients in this experiment. However, the frequent request for status of each message folder has contributed the high amount of resource usage discussed earlier.

Thunderbird issued the highest number of STORE command (265) followed by Opera (180) for all types of flag options. Again Thunderbird issued 424 FETCH commands to manipulate 80 messages because of its frequent request for new message and messages flags. Mulberry issued 320 FETCH commands because it fetches envelope part of the message unlike other clients. Outlook, Opera and Sylpheed issued 301, 263 and 241 FETCH commands respectively. Sylpheed and Opera are the only client that utilized STATUS command for 104 and 40 times respectively. Other commands issued by these clients is listed in table 5.5.

Table 5.6 and 5.7 summarize the total sum amount of resource usage under each selected performance metrics for default and optimized behavior of selected clients respectively. The calculation is made from the mean values obtained from 35 replications for each client.

From the tables, Opera is th highest consumer in disk I/O write (7710 MB/sec) followed by Sylpheed (2149 MB/Sec). Thunderbird and outlook are the third and fourth with values 1617 MB/Sec and 844 MB/Sec respectively. Opera's figures are highly distorted because of the additional commands for implementation purpose. This can be seen from the network bandwidth received and sent.

In the optimized behavior of clients, Sylpheed is the worst (2263 MB/Sec) in

disk I/O write followed by Thunderbird (1454 MB/Sec). Opera is the third and this time, the simulation problem is much more reduced in this client's behavior. Thus, it is possible to say this is the close to actual behavior of the client. Outlook and Mulberry are the fourth and fifth with disk I/O usage with values 864 MB/sec and 697 MB/sec respectively.

5.7 Trend Analysis

Trend analysis was attempted to project values in the time series graph and to compare the resource usage based on the growth trend on the graphs. Most client's default behavior graphs showed a linear function trend liine. Graphs from Mulberry, Thunderbird and Outlook are good example for this. Sylpheed graphs behave same for both default and optimized behaviors graph. For cllient's optimized behavior, some showed an exponential function but the slopes for this function were not significant because the slope for theses graphs were not steepy as normal exponential function graphs. Most graphs fitted with polynomial function with varying degree from 2 to 5. The equations with r-square value indicated on the graphs to provide evidence how well the function fits to the actual graph. From these equations, one can easily see the trend by which the performance mertics decreased or increased.

5.8 Conclusion

At the present, the IMAP protocol standards are generally focused on allowing a client to access and manipulate electonic mail messages on a server. However, the diversity of the IMAP protocol commands permits email client developers to implement the protocol differently without realizing its server side consequences.

The effects of different IMAP clients on IMAP server performance and resource usage has been described and quantified through studying how different IMAP clients implement the IMAP protocol. Disk I/O read and write usage was the major performance metric studied, as well as network bandwidth usage. The CPU I/O wait percentage was also a metric that reflected the performance bottlneck inherent in disk I/O write peformance. Network bandwidth usage was also directly influenced by the disk I/O performance. It is possible to say the selected performance metrics appropriatly evaluated clients' behavior under different scenarios.

From the outcomes, Mulberry has shown efficient use of resources in all performance metrics except disk space usage. Outlook has shown mid level performance consumption in most selected performance metrics. Opera and Sylpheed showed consistent IMAP protocol implementation in their default and optimized behaviors. Opera was the best client for disk space utilization but has shown higher disk I/O resource usage in its default behavior, although the additional overhead due to simulation should be considered. Sylpheed's highest resource consumption, especially in its optimized behavior, was a surprising result because it is thought to be lightweight client. Thunderbird's performance with respect to disk space usage was the worst in all scenarios. In most optimized client's behavior results, Opera, Outlook and Thunderbird showed very similar results, with the exception being disk usage.

The default vs. optimized and small vs average message size scenarios helped to clearly see the resource requirement differences among different clients. All clients' comparison between their default vs. optimized behavior clearly showed a significant difference in all selected performance metrics. The results also show significance difference between these selected clients. The selected message size comparison also highlighted the significant difference among clients' default and optimized behaviors.

5.8.1 Limitation and Obstacles in the Experiments

Most of selected clients except Sylpheed manipulated messages on IMAP server via multiple logins concurrently to mailboxes. Thus, any manipulation to the mailboxes is accomplished through the already established connection. However, during simulation for this experiment, it was not possilbe to achieve this

since the telnet connection that this thesis adapted allowed only a single connection. Hence, the simulation was implemented through switching between mailboxes. This problem was significant during Opera mail's simulation since there was a lot of switching between the Inbox and other message folders.

The other major problem was during simulation Opera's mail client behavior for managing deleted messages. The client implements dynamic clearing of messages with deleted flags through the SEARCH command and removing the deleted flag status for listed messages when running the EXPUNGE command. This implementation helped the client not permanently remove deleted messages when the user wants them to remain. However, it was not possible to achieve this implementation within the benchmarking tool developed for this thesis. Thus, the above implementation was achieved through a significant amount of additional command runs.

Another obstacle in this experiment was simulating a single message arrival in a mailbox. In an actual email infrastructure, this is done through the SMTP protocol that delivers the messages to the mailbox. However, in this experiment, since the experiment is only about IMAP protocol, messages were put in mailbox before the experiment began to avoid the complications of concurrent SMTP traffic.

The original plan of this project was to include multiuser and bulk email manipulation experiments. The bulk email manipulation experiment was completed for 3 message folders successfully, but due to limitations of time, the results could not be included for discussion and analysis.

A multiuser experiment was also conducted for 8 users. However, the design could not reflect a real life scenario due to limitations in the current version of the benchmarking facility.

5.8.2 Contributions of the Thesis

The major output of this thesis is its contribution to the study of IMAP clients' behavior. The achieved results from this thesis could be helpful for standard organizations like The Internet Engineering Task Force (IETF). Since the mission of IETF is "to make the Internet work better by producing high quality, relevant technical documents that influence the way people design, use, and manage the Internet," the outcome from this thesis could contribute to look at IMAP protocol implementation from resource usage point of view. The findings might influence the standard organizations to work closely with IMAP client developers.

The other contribution from the outcome is for system administrators who manage email service infrastructure for large organization with thousands customers. The result could provide an insight during resource allocation and preparation. Since availability is crucial to email service, the outcome could

contribute knowledge for a quick understanding of accidental mail server resource requirements due to a client's specific implementation of the IMAP protocol.

The methodology utilized in this thesis can be adapted easily for additional research on the existing organizational email infrastructure. Benchmarking IMAP servers based on clients behavior is a new concept, and it can be developed further. Thus, a system administrator can confidently control the infrastructure if the maximum threshold resource usage can be studied based on the email clients used by its users.

Email client developers should take also such issues into consideration in the process of email client development. Developers should be careful when they decide on the default behavior for their software based on these results since most users utilize software with its default configuration.

Last but not least, the outcome underscore the principle that users should optimized their email client from a resource usage perspective. By doing so, a single user can save a lot of resources such as Internet traffic and other indirect resource consumption. Although this is a different topic, as the amount of message increases in mailbox, the resource requirement from server side and client machine also increases for message manipulation of newly arrived messages. Synchronizing mailboxes is also elongated because of deleted and unwanted messages that should be permanently removed. This could contiribute in a small way power usage in data centers.

5.9 Future Work

Since the approach in this thesis is new, several issues could not be addressed, and they invite further research and analysis.

The benchmarking approach used to test IMAP server is an interesting area that could be developed with more extensive programming and organization since existing benchmarking tools do not address these concerns.

Although this thesis could not research and compare different IMAP servers with the same approach due to time limitations, studying different IMAP servers response to different IMAP protocol implementation could result an interesting outcome.

During the process of this thesis, obtaining resources related to IMAP were the major problem to quickly organize and study documents. Developing a central information center for IMAP and IMAP related developments could help to facilitate and produce better results for research like this. This can be achieved through developing a website and collecting resources in relation to

IMAP protocol, although it requires effort and willingness of concerned parties. Since existing IMAP books are decade old and have not been updated, information included in this thesis could be developed further and used as an alternative resource.

The increasing trend on utilizing IMAP protocol is a good indication for the future of Internet network traffic. Unnecessary resource utilization will lead to unnecessary resource consumption and competition. Most email clients are popular because of GUI features that they come with. However, their true purpose is to implement the IMAP and/or POP protocols. Many users do not select an email client because of their efficincy in resource usage but because of their GUI and other fancy features.

Unwisely used resources are an unnecessary waste, especially in developing countries. Africa and Asia are the most vulnerable continent with this respect because of lack or adequate technology adaptation. Waste resources because of IMAP protocol implementation could be used to balance Internet resource utilization.

The Internet Task Force or another standard agency should consider standardizing part or all of the implementation of the IMAP protocol as well as its features and insist that such requirements be met before clients are made available for use.

Bibliography

[1] Mulberry. Available at http://www.mulberrymail.com/about.shtml.

[2] Pop4 specification. http://pop4.org/, Accessed March 2, 2010.

[3] *Red Hat Enterprise Linux 5.1 Deployment Guide*, 2007. Available at http://www.centos.org/docs/4/4.5/Reference_Guide/index.html.

[4] Email client popularity, 2008. http://fingerprintapp.com/email-client-stats, Accessed in February 2010.

[5] [ms-stanoimap]: Rfc 3501 - outlook imap standards compliance, 2008.

[6] Email client popularity, 2009. http://www.labnol.org/internet/email/most-popular-email-clients/9340/, Accessed in January 2010.

[7] Ajax: The collaborative applicatioin platform, 2010. http://www.ajax.org/, Accessed in January 2010.

[8] Blackberry-business software features, 2010. http://na.blackberry.com/eng/services/business/, Accessed in February 2010.

[9] Yahoo! inc.: Company history, 2010. http://yhoo.client.shareholder.com/press/history.cfm, Accessed in February 2010.

[10] Barzan Tony Antal. Popular email-clients reviewed. http://webhosting.devshed.com/c/a/Web-Hosting-Articles/Popular-Email-Clients-Reviewed/, Accessed in February 2010.

[11] S. Bradner. rfc2119: Key words for use in rfcs to indicate requirement levels, 1997. Available at http://www.ietf.org/rfc/rfc2119.txt.

[12] Joachim Charzinski. Observations in e-mail performance. 2002. Available at http://www.jcho.de/jc/Pubs/itc2002-col.pdf.

[13] RXN Comp. History of samba. Available at http://www.rxn.com/services/faq/smb/samba.history.txt.

[14] Courier. *Maildir E-mail Directory*. Available at http://www.courier-mta.org/maildir.html.

[15] M. Crispin. rfc1064: Interactive mail access protocol: Version 2, 1988. Available at `http://www.ietf.org/rfc/rfc1064.txt`.

[16] M. Crispin. rfc1733: Distributed electronic mail models in imap4, 1994. Available at `http://www.ietf.org/rfc/rfc1733.txt`.

[17] M. Crispin. rfc3501: Internet message access protocol-version 4revl, 2003. Available at `http://www.exim-new-users.co.uk/index2.php?option=com_content&do_pdf=1&id=31`.

[18] Mark Crispin. *Mailbox Format Characteristics*. University of Washington, December 2006. Available at `http://www.washington.edu/imap/documentation/formats.txt.html`.

[19] Mark Crispin. Mark crispin's web page, 2009. `http://panda.com/mrc/`, Accessed in January 2010.

[20] Nick Elprin and Bryan Parno. An analysis of database-driven mail servers. Technical report, 2003. Available at `http://www.usenix.org/event/lisa03/tech/full_papers/elprin/elprin_html/`.

[21] N. Freed and N. Borenstein. rfc2049: Multipurpose internet mail extensions, 1996. Available at `http://www.mhonarc.org/~ehood/MIME/2049/rfc2049.html`.

[22] Aeleen Frisch. *Essential System Administration - 3rd Edition*. O'Reilly, 2002.

[23] R. Gellens, C. Newman, and L. Lundblade. rfc2449: Pop3 extension mechanism, 1998.

[24] Chaos Golubitsky. Toward and automated vulnerability comparison of open source imap servers. In *Proceedings of the 19th conference on Large Installation System Administration Conference*, volume 19, pages 2–2. USENIX Association, 2005.

[25] Terry Gray. Message access paradigms and protocols, 1995. `http://staff.washington.edu/gray/papers/imap.vs.pop.html`, Accessed in January 2010.

[26] Peer Heinlein and Peer Hartleben. *The Book of IMAP: Building a Mail Server with Courier and Cyrus*. O'Reilly, 2008.

[27] Ian Hyslop and Chris Imafidon. *Systems Integration Handbook*. 2002.

[28] Runze Li Kai-Tai Fang and Agus Sudjianto. *Design and Modeling for Computer Experiments (Chapman & Hall/CRC Computer Science & Data Analysis)*. Chapman & Hall/CRC; 1 edition, October 14, 2005.

[29] Lee LeClair. Store-and-forward technology works when link isn't always on. Technical report, 2009. Available at `http://www.azbiz.com/articles/2009/11/13/media_technology/tech_talk/doc4afda883a8d19019498735.txt`.

[30] Don Libes. *Exploring Expect*. O'Reilly, 1994.

[31] Don Libes. Automation and testing of character-graphic programs. 1997.

[32] S. Maes and A. Melnikov. rfc4550: Internet email to support diverse service environments (lemonade) profile, 2006. Available at `http://www.ietf.org/rfc/rfc4550.txt`.

[33] Jorge Manjarrez-Sanchez. Implementing linux system calls. 1999. Available at `http://www.linuxjournal.com/article/3326`.

[34] Dianna Mullet and Kevin Mullet. *Managing IMAP*. O'Reilly, 2000.

[35] J. Myers. rfc1731: Imap4 authentication mechanisms, 1994. Available at `http://www.ietf.org/rfc/rfc1731.txt`.

[36] J. Myers, Carnegie Mellon, and M. Rose. rfc1939: Post office protocol - version 3, 1996. Available at `http://tools.ietf.org/html/rfc2449`.

[37] C. Newman. rfc2595: Using tls with imap, pop3 and acap, 1999. Available at `http://www.ietf.org/rfc/rfc2595.txt`.

[38] Gary Nutt. *Operating Systems, A Modern Perspective*. 1997.

[39] University of Washington. Uw imap server documentation. Available at `http://www.washington.edu/imap/documentation/internal.txt.html`.

[40] Mark Pallen. Guide to the internet: Electronic mail. 1995.

[41] Ruoming Pang, Mark Allman, and Mike Bennett. A first look at modern enterprise traffic. 2005. Available at `http://www.usenix.org/event/imc05/tech/full_papers/pang/pang.pdf`.

[42] Phil Pennock. On imap service for customers. *;login:*, 29(5), 2004. Available at `http://www.usenix.org/publications/login/2004-10/pdfs/pennock.pdf`.

[43] David Pogue and J.D.Biersdorfer. *The Internet: The Missing Manual (Missing Manual)*. O'Reilly, 2006.

[44] Gilbert Ramirez. *Wireshark and Ethereal*. Syngress Publishing Inc., 2007.

[45] Curtis Smith. *Pro Open Source Mail: Building an Enterprise Mail Solution*. Apress, 2006.

[46] Inc. The Radicati Group. Email statistics report, 2009-1013. Technical report, 2009. Available at `http://www.radicati.com/`,`http://findarticles.com/p/articles/mi_pwwi/is_200905/ai_n31620110/`, `http://email.about.com/od/emailtrivia/f/how_many_email.htm`, `http://sip-trunking.tmcnet.com/topics/security/articles/55741-19-billion-e-mail-users-2013-radicati-group.htm`.

[47] Heinz Tschabitscher. What is the average size of an email message? http://email.about.com/od/emailstatistics/. Accessed on March 1, 2010.

[48] Company Website. 10 years of samba!

[49] Company Website. Slamd distributed load generation engine. http://www.slamd.com/. Accessed March 2010.

[50] Dag Wieers. Dstat: Plugin-based real-time monitoring. 2007.

[51] IMAP Wiki. Imap server benchmarking. http://www.imapwiki.org/Benchmarking. Accessed January 25, 2010.

[52] David Wood. *Programming Internet Email*. O'Reilly, 1999.

Appendix A

List of IMAP RFCs and Their Status

RFC Number	Authors	Title	Obsoleted by RFC	Updated by RFC	Year	Authors Organization	Category
1064	M. Crispin	Interactive Mail Access Protocol: Version 2	1176, 1203		1988	SUMEX-AIM	
1176	M. Crispin	Interactive Mail Access Protocol: Version 2	1203		1990	University of Washington	
1203	J. Rice	Interactive Mail Access Protocol: Version 3	1730		1991	Stanford	
1730	M. Crispin	Internet Message Access Protocol - Version 4	2060, 2061		1994	University of Washington	Standard Track

1731	J. Myers	IMAP4 Authentication Mechanisms			1994	Carnegie Mellon	Standard Track
1732	M. Crispin	IMAP4 Compatibility with IMAP2 and IMAP2bis			1994	University of Washington	Informational
1733	M. Crispin	Distributed Electronic Mail Models in IMAP4			1994	University of Washington	Informational
2060	M. Crispin	Internet Message Access Protocol - Version 4rev1	3501		1996	University of Washington	Standard Track
2061	M. Crispin	IMAP4 Compatibility with IMAP2bis			1996	University of Washington	Informational
2062	M. Crispin	Internet Message Access Protocol - Obsolete Syntax			1996	University of Washington	Informational
2086	J. Myers	IMAP4 ACL extension	4314		1997	Carnegie Mellon	Standard Track
2087	J. Myers	IMAP4 QUOTA extension			1997	Carnegie Mellon	Standard Track
2088	J. Myers	IMAP4 non-synchronizing literals	4466		1997	Carnegie Mellon	Standard Track
2095	J. Klensin, R. Catoe, P. Krumviede	IMAP/POP AUTHorize Extension for Simple Challenge/Response	2195		1997	MCI	Standard Track
2177	B. Leiba	IMAP4 IDLE command			1997	IBM T.J. Watson Research Center	Standard Track
2180	M. Gahrns	IMAP4 Multi-Accessed Mailbox Practice			1997	Microsoft	Informational

2192	C. Newman	IMAP URL Scheme	5092		1997	Innosoft	Standards Track
2193	M. Gahrns	IMAP4 Mailbox Referrals			1997	Microsoft	Standards Track
2195	J. Klensin, R. Catoe, P. Krumviede	IMAP/POP AUTHorize Extension for Simple Challenge/Response			1997	MCI	Standards Track
2221	M. Gahrns	IMAP4 Login Referrals			1997	Microsoft	Standards Track
2342	M. Gahrns, C. Newman	IMAP4 Namespace		4466	1998	Microsoft and Innosoft	Standards Track
2359	J. Myers	IMAP4 UIDPLUS extension	4315		1998	Netscape Communications	Standards Track
2595	C. Newman	Using TLS with IMAP, POP3 and ACAP		4616	1999	Innosoft	Standards Track
2683	B. Leiba	IMAP4 Implementation Recommendations			1999	IBM T.J. Watson Research Center	Informational
2971	T. Showalter	IMAP4 ID extension			2000	Mirapoint, Inc.	Standards Track
3348	M. Gahrns, R. Cheng	The Internet Message Action Protocol (IMAP4) Child Mailbox Extension			2002	Microsoft	Standards Track

3501	M. Crispin	INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1		4466, 4467, 4469, 4551, 5032, 5182	2003	University of Washington	Standard Track
3502	M. Crispin	Internet Message Access Protocol (IMAP) - MULTIAPPEND Extension		4466, 4469	2003	University of Washington	Standard Track
3503	A. Melnikov	Message Disposition Notification (MDN) profile for Internet Message Access Protocol (IMAP)			2003	ACI Worldwide/MessagingDirect	Standard Track
3516	L. Nerenberg	IMAP4 Binary Content Extension		4466	2003	Orthanc Systems	Standard Track
3517	E. Blanton, M. Allman, K. Fall, L. Wang	A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP			2003	Purdue University, BBN/NASA GRC, Intel Research, University of Kentucky	Standard Track
3691	A. Melnikov	Internet Message Access Protocol (IMAP) UNSELECT command			2004	Isode Ltd.	Standard Track
4314	A. Melnikov	IMAP4 Access Control List (ACL) Extension			2005	Isode Ltd.	Standard Track
4315	M. Crispin	Internet Message Access Protocol (IMAP) - UIDPLUS extension			2005	University of Washington	Standard Track

4466	A. Melnikov	Collected Extensions to IMAP4 ABNF			2006	Isode Ltd.	Standard Track
	C. Daboo						
4467	M. Crispin	Internet Message Access Protocol (IMAP) - URLAUTH Extension		5092, 5550	2006	University of Washington	Standard Track
4549	A. Melnikov, Ed.	Synchronization Operations for Disconnected IMAP4 Clients			2006	Isode Ltd.	Informational
4551	A. Melnikov	IMAP Extension for Conditional STORE Operation or Quick Flag Changes Resynchronization			2006	Isode Ltd.	Standard Track
	S. Hole						
4731	A. Melnikov, D. Cridland	IMAP4 Extension to SEARCH Command for Controlling What Kind of Information Is Returned			2006	Isode Ltd., Inventure Systems Ltd.	Standard Track
4959	R. Siemborski, A. Gulbrandsen	IMAP Extension for Simple Authentication and Security Layer (SASL) Initial Client Response			2007	Google, Inc., Oryx Mail Systems GmbH	Standard Track
4978	A. Gulbrandsen	The IMAP COMPRESS Extension			2007	Oryx Mail Systems GmbH	Standard Track
5032	E. Burger, Ed.	WITHIN Search Extension to the IMAP Protocol			2007	BEA Systems, Inc.	Standard Track

5092	A. Melnikov, Ed., C. Newman	IMAP URL Scheme		5593	2007	BT	Informational
5161	A. Gulbrandsen, Ed., A. Melnikov, Ed.	The IMAP ENABLE Extension			2008	Oryx Mail Systems GmbH, Isode Ltd.	Standard Track
5162	A. Melnikov, D. Cridland, C. Wilson	IMAP4 Extensions for Quick Mailbox Resynchronization			2008	Isode Ltd., Nokia	Standard Track
5182	A. Melnikov	IMAP Extension for Referencing the Last SEARCH Result			2008	Isode Ltd.	Standard Track
5232	A. Melnikov	Sieve Email Filtering: Imap4flags Extension			2008	Isode Ltd.	Standard Track
5267	D. Cridland, C. King	Contexts for IMAP4		5465	2008	Isode Limited	Standard Track
5464	C. Daboo	The IMAP METADATA Extension			2009	Apple, Inc.	Standards Track

5465	A. Gulbrandsen, A. Melnikov, C. King	The IMAP NOTIFY Extension			2009	Oryx Mail Systems GmbH, Isode Ltd.	Standards Track
5466	A. Melnikov, C. King	IMAP4 Extension for Named Searches (Filters)			2009	Isode Ltd.	Standards Track
5530	A. Gulbrandsen	IMAP Response Codes			2009	Oryx Mail Systems GmbH	Standards Track
5593	N. Cook	Internet Message Access Protocol (IMAP) - URL Access Identifier Extension			2009	Cloudmark	Standards Track

Appendix B

Automate Scripts for Benchmarking

```
==========================================
Main Automate Script
==========================================
#!/bin/bash
# mkdir /root/imap-client/program/ #this file tree should exist before running this script and
#all scripts sould be put under /program directory
# make sure you have enabled ssh connection between server and client machines through ssh-keygen
# make all packages and softwares installed and configured as explained in the methodology chapter.
# hostaddress for IMAP server and IMAP client is imap-server and imap-client
# test users must be created with name "client 1, 2, 3, 4 etc" on IMAP-server. If already exists
#and is not possible to change, edit the script accordingly
PS3='Choose the imap-client for test: '
echo "Enter the number of client users you want to test: "
read numberofusers #input
echo "Enter desired number of email message in a mailbox: "
read messageinmailbox #iinput
echo "Enter desired size of message for test: "
read messagesize #iinput
echo "How many times you want to repeat the experiment: "
read replication #iinput
echo
#choose client for test
choice_*of()
{
select client #iinput
```

```
do
echo
echo "You have choosen $client for test purpose and the simulation for this
client will run soon";
echo
break
done
} #available clients for choice
choice_*of Outlook Thunderbird Opera Sylpheed Mulberry Exit
#environment variable
clients=$numberofusers
messages=$messageinmailbox
size=$messagesize
echo "$clients";
echo "$messages";
if [ $client == "Outlook" ] —— [ $client == "Thunderbird" ] —— [ $client ==
"Opera" ] —— [$client == "Sylpheed" ] —— [ $client == "Mulberry" ];
then
#clean files from aborted test
rm -r /root/imap-client/$client/*
for i in 'seq 1 $replication';
do
#clean files from aborted test
rsh imap-server rm -r /root/imaptest_*results
#prepare directory for test on server
rsh imap-server mkdir -p /root/imap-server/program
#copy scripts to server
scp -r /root/imap-client/program/sendmail_*wrapper.sh sendmail.sh imap-
server:/root/imap-server/program/
#prepare directory for test on client
mkdir -p /root/imap-client/$client/results/replication$i/loginfo
#clean files from aborted test
rm -r /root/imap-client/program/${client}fivefolder.expect*
#clean Maildir directory on server
sh /root/imap-client/program/remove.sh $numberofusers
for f in 'seq 1 $numberofusers';
do
#prepare message folders on IMAP server
expect /root/imap-client/program/${client}foldermgmtsinglefivefolder.expect
imap-server 143 client$f Teshu02Dagim $numberofusers
done;
echo " Test replication $i of $replication Started. Preparing messages to send
to users. Please wait...";
#collect message box disk space
rsh imap-server du /home/client*/
>/root/imap-client/$client/results/replication$i/${client}Messageboxsizeemptyreplication$i.txt
#send messages to mailbox
```

```
rsh imap-server sh /root/imap-server/program/sendmail_*wrapper.sh $num-
berofusers $messageinmailbox $messagesize >& /dev/null
echo " Sending $messageinmailbox messages to $numberofusers users each.
Wait..."
sleep 5
queuewait=0
postqueue="'rsh imap-server postqueue -p'"
maxwait=180
delay=0
#check message queue
while [ "$postqueue" != "Mail queue is empty" ];
do
if [ $queuewait -gt $maxwait ];
then
echo "Sending message took more than $maxwait minutes. Aborting the test.
Please try again."
rsh imap-server postqueue -p >>/root/imap-client/$client/results/replication$i/loginfo/errorrep
rsh imap-server postsuper -d ALL
exit 1
fi
echo "There are messages in queue. Please wait..."
sleep 10
queuewait=$((queuewait + 1))
delay=$((delay + 10))
echo "Message queue delayed for $delay seconds. If delay more than an hour
test will abort."
postqueue="'rsh imap-server postqueue -p'"
done
echo " $messageinmailbox messages sent to $numberofusers users success-
fully."
# collect Maildir directory disk space usage
rsh imap-server du /home/client*/
>/root/imap-client/$client/results/replication$i/${client}Messageboxsizebeforemanipulationrep
# collect disk volume space usage
for n in 'seq 1 $numberofusers';
do
#copy original simulation scripts
cp /root/imap-client/program/backup${client}fivefolderoptimized.expect ${client}fivefolder.exp
&
done;
#prepare directory on server
rsh imap-server mkdir -p /root/imaptest_*results/$client/
#restart IMAP server
rsh imap-server reboot
echo "Rebooting your imap-server. Wait for some minutes..."
# wait for 3 minutes
sleep 180
```

```
# shutdown external interface
rsh imap-server ifdown eth1
# collect perfomance metrics data
rsh imap-server dstat –output /root/imaptest_*results/$client/${client}fivefolderoptimizedReplicatio
–noheaders -dnyc -D total -N total -C total &
sleep 10;
for m in ‘seq 1 $numberofusers‘;
do
# run simulation scripts
expect /root/imap-client/program/${client}fivefolder.expect$m imap-server
143 client$m Teshu02Dagim >& /dev/null & done;
checkprocess=‘pgrep expect —wc -l‘
maxprocess=$(( $numberofusers + 10 ))
minprocess=5
counthigh=0
# check processes for simulation scripts. This is helpful for multiuser tests
if [ $checkprocess -gt $maxprocess ]; then
echo ”$i Process $checkprocess” >>/root/imap-client/$client/results/replication$i/loginfo/Reason1
echo ”$i Process $checkprocess” >>/root/imap-client/$client/results/replication$i/loginfo/errorrep
kill -9 $(pgrep expect)
echo ”There is problem with the test”
exit 1
fi
while [ $checkprocess -gt $minprocess ] && [ $checkprocess -le $maxprocess ];
do
checkprocess=‘pgrep expect —wc -l‘
echo ”$i Process $checkprocess” >>/root/imap-client/$client/results/replication$i/loginfo/Reason2
if [ $checkprocess -gt $maxprocess ]; then
echo ”$i Process $checkprocess” >>/root/imap-client/$client/results/replication$i/loginfo/Reason1
echo ”$i Process $checkprocess” >>/root/imap-client/$client/results/replication$i/loginfo/errorrep
kill -9 $(pgrep expect)
echo ”There is problem with the test”
exit 1
fi
sleep 3
done
count=0
while [ $numberofusers -gt 5 ] && [ $checkprocess -gt 0 ] && [ $checkprocess
-le $minprocess ];
do
checkprocess=‘pgrep expect —wc -l‘
if [ $count -gt 5 ]; then
echo ”$i Process $checkprocess and $count”
>>/root/imap-client/$client/results/replication$i/loginfo/errorreplication$i
kill -9 $(pgrep expect)
exit 1
fi
```

```
count=$(( $count + 1 ))
echo "$i Process $checkprocess" >>/root/imap-client/$client/results/replication$i/loginfo/Reasc
echo "$i Count $count" >>/root/imap-client/$client/results/replication$i/loginfo/Reason3
sleep 3
done
while [ $numberofusers -le 5 ] && [ $checkprocess -gt 0 ];
do
checkprocess=`pgrep expect —wc -l`
echo "$i Process $checkprocess" >>/root/imap-client/$client/results/replication$i/loginfo/Reasc
sleep 3
done
sleep 10
#stop data collection on IMAP server
rsh imap-server killall python;
sleep 3
#pull collected data from IMAP server
scp -r imap-server:/root/imaptest_*results/$client/${client}fivefolderoptimizedReplication$i.csv
/root/imap-client/$client/results/replication$i/
#clean earlier created directory
rsh imap-server rm -r /root/imaptest_*results
#collect message box size
rsh imap-server du /home/client*/
>/root/imap-client/$client/results/replication$i/${client}MessageboxsizelastReplication$i.txt
for c in `seq 1 $numberofusers`;
do
# collect amount of messages left in message folders
expect /root/imap-client/program/${client}checkmailbox5folder.expect imap-
server 143 client$c Teshu02Dagim
$numberofusers >/root/imap-client/$client/results/replication$i/messagesleftinmailboxreplicati
done;

make server external interface available
rsh imap-server ifup eth1
sleep 5
#clean copied simulation scripts from client server
rm -r /root/imap-client/program/${client}fivefolder.expect*
echo " Test replication $i of $replication completed.";
#clean Maildir directory
sh /root/imap-client/program/remove.sh $numberofusers
#remove earlier created folders from IMAP server
rsh imap-server rm -r /root/imap-server
done;
echo "Rebooting client server..."
sleep 5
reboot
else echo "These are the opetions we have. If you want to try again run the
main program";
```

```
echo ”None of the options selected”>>/root/imap-client/$client/results/replication$i/loginfo/error
exit 1
fi
exit 0
==========================================
Mailbox Cleaner Script
==========================================
#! usr/bin/bash
# cleaning mailbox and message folders
echo ”Clearing Mailbox and Folders...”
for u in $(seq 1 $1);
do
rsh imap-server rm -r /home/client$u/Maildir
done
echo ”Clearing Mailbox and Folders Completed”
==========================================
Message Sender Script
==========================================
#! usr/bin/bash
#send messaes to test users and should be run on IMAP server
i=1
while [ $i -le $clients ];
do
for m in ‘seq 1 $messages‘; do dd if=/dev/urandom bs=$sizek count=1 —
uuencode - — mail -s ”Test message” client$iimap-server.vlab.iu.hio.no &
done;
i=‘expr $i + 1‘
done
==========================================
Wrapper Script
==========================================
#wrapper script to run sendmail.sh
clients=”$1”
messages=”$2”
size=”$3”
export clients messages size
sh /root/imap-server/program/sendmail.sh ”$”
```