

Frugal E-mail

Adam Lewis

Student at Lancaster University
Lancaster, United Kingdom
2009-2013
a.lewis3@lancaster.ac.uk

Adrian Friday

School of Computing & Communications
Infolab21, Lancaster University
Lancaster, LA1 4WA
adrian@comp.lancs.ac.uk

Abstract

This paper discusses our research into the area of efficient e-mail storage. We discuss the many potential areas for improvement on currently employed storage formats and explore how effective our proposed improvements perform in comparison through an analysis of our prototype system “PortalMail”. We also present the results of a small experiment to infer the popularity of SMTP with respect to ESMTP and provide an explanation of how modern e-mail systems operate, analysing efficiency from the networking perspective right down to the formatting and storage of messages.

Categories and Subject Descriptors A.0 [General]: Exploration of how compression technologies can be applied to e-mail

General Terms Design, Experimentation.

Keywords E-mail compression; Lossless compression

1. Introduction

E-mail as a technology is commonly associated with networks, but it actually pre-dates both the Internet and ARPANET. E-mail was never “invented” per se [1], it evolved over time in harmony with supporting technologies, its potential expanding and growing.

In its earliest beginnings, e-mail was a small program capable of appending notes to a mailbox file/folder. These files were stored and written to the work space of a colleague on the same local machine. Back during the many dumb terminals to one mainframe era this worked exceptionally well. All users shared the same physical machine to carry out their work and from the user's perspective, e-mail was conceptually equal to leaving a note on a person's desk.

This soon changed with the introduction of the ARPANet experiment however and these mainframe computers were suddenly networked together and able to communicate. Although ARPANet is mostly associated with the creation of the Internet, sometime “in late 1971, a computer engineer named Ray Tomlinson sent the first email message” [2]. “First, he chose the @ symbol to distinguish between messages addressed to mailboxes in the local machine and messages that were headed out onto the network” [3]. He then extended file transfer protocols

he had recently developed to allow for the appending of data to remote files over the network. Finally, he tested his creation and what followed was described as a “Natural Phenomenon” [4].

E-mail instantly became an incredibly popular application, proving itself to be one of the Internet's “killer applications” [10]. Even in today's social centric Internet, e-mail remains incredibly popular [9] which is an impressive feat for technology that was invented roughly forty years ago “because it seemed like a neat idea” [8].

“It soon became obvious that the ARPANET was becoming a human-communication medium with very important advantages over normal U.S. mail and over telephone calls. One of the advantages of the message systems over letter mail was that, in an ARPANET message, one could write tersely and type imperfectly, even to an older person in a superior position and even to a person one did not know very well, and the recipient took no offense. The formality and perfection that most people expect in a typed letter did not become associated with network message probably because the network was so much faster, so much more like the telephone” - J.C.R. Licklider [5].

Having proved itself to be such a popular application, standardisation of the e-mail system soon followed to encourage co-operation between service providers. This co-operation was of paramount importance to create systems that could be used to send and read e-mail on a global scale over the rapidly expanding Internet. Protocols such as X.400, SMTP, ESMTP, IMAP and POP are the backbone of modern e-mail transportation; The Internet Message Format RFC [6] and the Multipurpose Internet Mail Extensions RFC [7] also define how e-mail messages are formatted.

Due to the incredible popularity the system holds globally and its evolution based history, we believe that exploration should be undertaken to unveil if and where we could improve the design of e-mail. With this in mind, we began researching the system from the top down to discover if it is possible to improve efficiency and reduce the data footprint produced from the operation of e-mail systems.

2. E-mail – The Network Perspective

The transition from local scope to networked scope e-mail systems required both a change in design and a change in technology. It was no longer possible to have only one program capable of writing messages to a user's mail box, instead we were required to create several new components to make the system work. The basic e-mail model is composed of four parts as described by David Wood's “Programming Internet Email” [37]. These four components are Mail User Agents, Mail Transfer

Agents, Mail Delivery Agents and Mail Retrieval Agents; It is the flow of messages between these components and the roles that they play that underpins any recognisable modern e-mail system.

“Mail User Agent. A program responsible for displaying received mail messages and composing and sending new messages” [27].

“Mail Transfer Agent. A program responsible for delivering and routing email through the Internet from source to destination” [28].

“Mail delivery agents are used to write messages into a user's mailbox, to forward messages to other MTAs, and to forward messages to other mail systems” [29].

“Mail Retrieval Agent. A program that moves and manages the movement of mail from an MTA to an MUA” [30].

Sending Mail

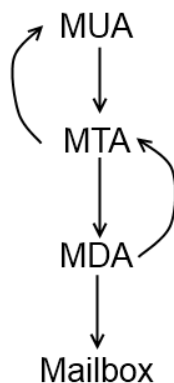


Figure 1. Diagram showing the flow of sending e-mail from sender to a receiver's mailbox

The above figure displays how a newly composed e-mail message is sent from the sending user to the target mailbox. First in the chain is the Mail User Agent (MUA). This component is the most visible and is utilised by end users every time they access their e-mail through web browsers/mobiles/applications et cetera. It is this component that is used to compose, retrieve and read e-mail over the network.

Next in the chain is the Mail Transfer Agent. The purpose of this component is to route accepted message. When attempting to send an e-mail through an MTA it will either accept or reject the request. Once the message is accepted and transferred, the MTA is fully responsible for delivering the message onwards through the network to the correct mailbox. Rejection is a slightly more complex procedure however as a MTA will either reject the message outright, or suggest a better MTA for the connected client to use in its place.

As an example, in larger organisations there will be several mail servers dedicated to storing user e-mail; It is the job of an MTA in this organisation to accept/route incoming messages onwards to the correct mailbox on the Intranet. By accepting and forwarding mail, the messages eventually reach the server that hosts the recipient's mail box and are locally stored.

To actually send/store messages, the MTA utilises a Mail Delivery Agent. This component usually delivers the incoming message to a locally held mailbox or forwards the mail onwards to

another MTA but could perform other tasks such as sending the mail to a printer. These tasks do not have to be carried out by a single MDA program however; It is common for a MTA to have two or more mail delivery agents available to it, each designed for differing purposes. Generally speaking, modern mail delivery agents either record the message to a locally held mailbox or forward the message onwards through the network.

To summarise, when sending a new e-mail message, several steps are taken. The first step is for the user to utilise a mail user agent to compose their message and send this message onto one of the mail transfer agents provided by public DNS records. This mail transfer agent then stores the message locally in cache and routes the message to its next destination. The message is then either forwarded by the MTA through the use of an appropriate mail delivery agent where the message may be sent to a locally held mailbox or onwards to another mail transfer agent somewhere on the Internet/Intranet.

Retrieving Mail

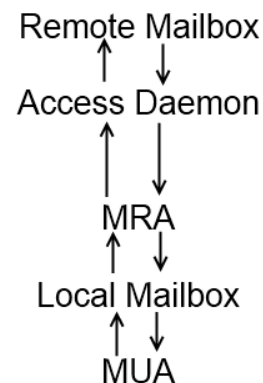


Figure 2. Diagram showing the flow of data when fetching mail

“When the Internet mail system was conceived, most users read mail directly from their mailboxes.” but now, “... messages written to a server side mailbox may need to be read by a machine across the Internet” [31]. This is a task that the Mail Retrieval Agent enables us to perform. A mail retrieval agent must at least be capable of at sourcing and copying mail from a networked mailbox to a user's local machine. This enables users that do not own a mail server or are intermittently connected to the Internet to access and use e-mail services.

To provide this service to such users, a mail server remains online and accepts mail on their behalf. This server is responsible for hosting the primary copy of the mailbox and can be read later on when the user is connected to the network. To access their mail, the MRA finds the users mailbox on the network, gains access to the mailbox & downloads the necessary files to the local machine. By providing these functions, the user is able to connect, download and read their e-mail as though they had their own locally set up mail server.

In terms of design, the system is incredibly simplistic and robust. The sending and downloading of e-mail is already efficient in terms of high level network usage and as such there is no clear scope for immediate improvements in design. As such, the task was to consider what state modern protocols are in. Being a core component of the e-mail system, how efficient the protocols can transport data and whether they can be improved

would make for a highly credible area of future research if there is a need for it.

3. Sending Mail – The Protocol Perspective

When studying the Internet and how it operates, there is no one system common to all providers but it all still fits together. Vendors follow the mantra of “think global, act local” [11], encouraging them to implement their own systems but in such a manner that they will be able to co-operate with each other for the good of the whole. Protocols used for transferring e-mail also follow this mantra and draw a perfect parallel to the Internet Protocol; Both the Internet Protocol and the protocols used to send e-mail provide a common ground for systems, allowing individual implementations with differing designs to interconnect and co-operate regardless of how they operate internally.

The protocols used by vendors makes for a perfect area of study as the efficiency of mail transfer has a large impact on the overall global system. To transfer emails from a sender to the recipient(s) over a network, there are currently three standardised protocols capable of performing this task. First of the three protocols is the International Organization for Standardization/International Telecommunication Union standard X.400, second is SMTP, and third is SMTP's successor ESMTP.

3.1 X.400

First standardised in 1984 [13] and being a “feature rich” [14] protocol, X.400 was thought by most experts to be the definitive future protocol for transporting e-mail. X.400 however was never accepted as the primary protocol for sending e-mail messages and is not widely used on the modern Internet.

There are many reasons people speculate as to why X.400 did not become the de facto standard and the lack of success it experienced makes for a compelling case study. Proposed reasons put forward include the use of an unfriendly addressing scheme [38], complaints at the complexity of the protocol, and the perceived threat of telecommunications companies trying to charge users for the use of e-mail services [15]. However, the general consensus today is that X.400 was just too late for the inter-networked personal computer revolution and had already lost the user base to SMTP.

“X.400 came into standardization at roughly the same time that the growth of the Internet was causing a huge installed base of SMTP/MIME implementations to be fielded. The PC revolution of the 1980s had already occurred, resulting in many proprietary commercial email systems. In short, X.400 was never widely adopted because it would have required expensive and widespread changes in email systems on a huge scale” [12].

With X.400 essentially dead in the water without user support, very few organisations actually utilised the X.400 e-mail protocols. X.400 is best suited to “environments where its additional benefits are needed, such as Military, Financial Data, Intelligence, Aviation and Government” [16]. Outside of these environments however, such as the less constrained public domain, users grew impatient of waiting and SMTP took the world by storm by filling the void while X.400 standards were set.

3.2 SMTP and ESMTP

SMTP or Simple Mail Transfer Protocol was initially put forward by the late Jonathan B. Postel as RFC 821 in 1982 [17]. The purpose of SMTP according to its author “is to transfer mail reliably and efficiently” [18] and is written to be a human friendly protocol to operate. A quick evaluation of these goals shows that they are perfectly reasonable; The SMTP protocol was defined

during the era of command line interfaces and low speed networks. This made simplicity a necessity to help users navigate basic interfaces effectively and efficiency was vital to prevent overloading the network.

“An SMTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is meant for the human user. It is intended that the three digits contain enough encoded information that the sender-SMTP need not examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be receiver-dependent and context dependent, so there are likely to be varying texts for each reply code” - Postel on SMTP replies and how they provide information for users [19].

In fact, SMTP is that simplistic to use, the envelope to send an e-mail message can be created using no more than six case insensitive commands. These commands are “HELO”, “MAIL FROM”, “RCPT TO”, “DATA” and the message termination string <Carriage Return Line Feed>. <Carriage Return Line Feed> or “<CRLF>”. In order to explain SMTP's full operation, consider the following example of an SMTP e-mail exchange taken from the book “Programming Internet Email” by David Wood.

```
HELO lovelace.staff.plugged.com.au
250 mailhost.plugged.net.au Hello [203.20.51.50], pleased to
meet you
MAIL FROM: dwood@lovelace.staff.plugged.com.au
250 dwood@lovelace.staff.plugged.com.au... Sender ok
RCPT TO: dwood@plugged.net.au
250 dwood@plugged.net.au... Recipient ok
DATA
354 Enter mail, end with “.” on a line by itself
```

Figure 3. Example of a SMTP conversation used to transfer mail to a receiving mail transfer agent [20]

Once connected to a mail server, usually on port 25 for a plain text connection or port 465 for an SSL connection, all of the commands and responses are sent over a TCP connection to ensure reliable error free delivery. To close the connection, the client can do so by entering the QUIT command at any point during the exchange.

Instead of quitting however, the client in the above example greets the server with the HELO command and provides the server with their domain, “lovelace.staff.plugged.com.au”. If the server accepts this greeting and is willing to allow e-mail from this domain, it replies in a positive manner through the use of response codes. In this case, the server is willing to accept mail so it replies with the code “250” along with a textual confirmation message.

```
250 Requested mail action okay, completed
500 Syntax error, command unrecognised
501 Syntax error in parameters or arguments
504 Command parameter not implemented
521 <domain> does not accept mail [rfc1846]
421 <domain> Service not available, closing transmission
channel
```

Figure 4. List of potential SMTP replies to HELO [21]

The client then provides the mail server with the e-mail address of the user trying to send mail. Again this is checked by the receiving server and is confirmed by using a reply code and a textual confirmation message.

```

250 Requested mail action okay, completed
552 Requested mail action aborted: exceeded storage
allocation
451 Requested action aborted: local error in processing
452 Requested action not taken: insufficient system storage
500 Syntax error, command unrecognised
501 Syntax error in parameters or arguments
421 <domain> Service not available, closing transmission
channel

```

Figure 5. List of potential SMTP replies to MAIL FROM: [22]

The next step when using the protocol is to provide the mail server with the e-mail address(es) the message is for; Each address is sent and confirmed individually and one at a time using the command “RCPT TO:”. In the above example figure 5, only one recipient is specified by the sender but if there were others, each address would be individually sent and verified using individual “RCPT TO:” commands. Again, if the response “250” is returned, the recipient has been accepted by the mail server and it has agreed that it will deliver the message for the connected client.

```

250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
550 Requested action not taken: mailbox unavailable
551 User not local; please try <forward-path>
552 Requested mail action aborted: exceeded storage
allocation
553 Requested action not taken: mailbox name not allowed
450 Requested mail action not taken: mailbox unavailable
451 Requested action aborted: local error in processing
452 Requested action not taken: insufficient system storage
500 Syntax error, command unrecognised
501 Syntax error in parameters or arguments
503 Bad sequence of commands
521 <domain> does not accept mail [rfc1846]
421 <domain> Service not available, closing transmission
channel

```

Figure 6. List of potential SMTP replies to RCPT TO: [23]

Finally, the DATA command is sent to signal that the connected client is ready to send the body of the e-mail message to the server. The server signals that it is ready with response code “354” and informs the client to end their message with a full stop on a line by itself.

```

354 Start mail input; end with <CRLF>.<CRLF>
451 Requested action aborted: local error in processing
554 Transaction failed
500 Syntax error, command unrecognised
501 Syntax error in parameters or arguments
503 Bad sequence of commands
421 <domain> Service not available, closing transmission
channel

```

Figure 7. List of potential SMTP replies to DATA [24]

Once the connected client has then finished transferring the body of their e-mail message, the <CRLF>.<CRLF> command is sent and the message has been successfully transferred to the MTA for delivery. The connected client is then able to close the connection through the “QUIT” command or specify another message by following the above process again from the “MAIL FROM:” step onwards.

Sending an SMTP e-mail message in this way is as simple as using Telnet to connect to the correct port of a server and following the steps above. Most modern servers however may prove a little difficult to Telnet onto. Fuelled by a need to prevent spam e-mail, most main providers have put into place measures to prevent malicious mail entering their system. Microsoft's Hotmail for example currently times out a connection on port 25 after approximately 2 seconds. Hypothetically this could be interpreted as a simple attempt by Microsoft to prevent human users attacking their system. Other more complex measures put in place include trusted IP lists [25] and challenging connected clients for proof of authentication.

To challenge clients for authentication requires additional protocol functionality and the Extended Simple Mail Transfer Protocol, known as ESMTP, is the protocol that provides this. By expanding on the features of SMTP and bringing some more specific functions to the protocol, vendors are able to provide a much richer service to their users. The full extended list of commands is extensive and varies from vendor to vendor as to which commands have been implemented [26] and it is the use of the “SMTP AUTH” command which allows vendors to challenge connected clients.

Essentially, ESMTP it is best thought of as SMTP where different providers can provide differing functionality based on what they believe to be in their best interest. There are no rules stating what functions must be implemented, only that they must be preceded by the greeting “EHLO” in place of “HELO” and that the server must provide a full list of the commands it supports.

If a server supports ESMTP it will respond to the “EHLO” command with a success code and conversely, if the server does not support ESMTP it will respond with an error code. This provides connected clients with a mechanism to differentiate between which of the two protocols to use when transferring their mail.

4. SMTP vs ESMTP – A Small Experiment

4.1 Motivation

While researching ESMTP it was soon discovered that no research has been performed exploring ESMTP's success with respect to SMTP. Being the protocol designed to supersede SMTP and with no definitive answer, we undertook a small experiment to try and infer how successful ESMTP has been at replacing SMTP as an Internet standard.

4.2 Experimental Procedure

To perform the experiment, four popular e-mail providers were selected and a new account set up with each. The four providers selected were Gmail, Hotmail, AOL and Yahoo, and all four providers had a new account set up on their service with the user name “smtppopularityexperiment”. Two emails were then sent under each newly set up account addressed to all four accounts. One message contained only text and one the other contained both text and an image attachment. This resulted in each account having 8 identical emails, two from themselves and six from the other three accounts, four containing text and an image and four just containing plain text.



Figure 8. The image sent as an attachment in the experiment

Once all of the emails had been exchanged, the message bodies were downloaded and the headers of the messages were examined. Through inspection of the “Received:” headers in a message, it is possible to determine the direction e-mail messages flow in from server to server and what protocol was used to transfer these messages between them. By solely interpreting the unambiguous headers, we produced the following results.

4.3 Results

	Text Only		Attachment	
	SMTP	ESMTP	SMTP	ESMTP
AOL	1	7	1	7
GMAIL	11	2	11	2
HOTMAIL	0	0	0	0
YAHOO	1	3	1	3

Figure 9. Protocol used for incoming mail

	Text Only		Attachment	
	SMTP	ESMTP	SMTP	ESMTP
AOL	0	7	0	7
GMAIL	10	2	10	2
HOTMAIL	0	3	0	3
YAHOO	3	0	3	0

Figure 10. Protocol used for outgoing mail

4.4 Analysis

In total 48 e-mail transfers were made using ESMTP and 52 mail transfers occurred using SMTP. This resulted in 48% of total traffic sent over ESMTP to 52% of total traffic sent via SMTP.

These results are surprising as ESMTP was introduced by RFC 1869 in 1995 [39], meaning it has taken 17 years for the protocol to be accepted by vendors to handle nearly half of the mail sent amongst some of the Internet's most popular providers.

Examining the data closer also revealed further interesting discoveries too. For example, AOL appears to be the most enthusiastic to adopt the newer standards, with all Internet bound and internal traffic shown to be sent over ESMTP. Google and Microsoft also appear to send their Internet bound mail by ESMTP. Yahoo however appears to be the least enthusiastic of all providers, as all out going mail is sent via the older SMTP protocol.

Yahoo conversely to this however, appears perfectly capable of receiving ESMTP mail from the Internet. This suggests the possibilities that there are technical restrictions at Yahoo or perhaps even a conscious unwillingness to co-operate and share the benefits of ESMTP with competitors. Aside from this strange behaviour, both AOL & Google show that they have ESMTP capability for incoming mail with four of six external source messages being transferred via ESMTP (The other two messages can be accounted for by Yahoo's exclusive use of SMTP for externally destined mail).

Finally, the protocols used for internally networked e-mails are also of interest. The discovery that AOL again uses ESMTP for all mail transfers whether external or internal is not ground breaking but Google's mail transfer patterns however are. Google based on how it signs headers appears to accept ESMTP mail from external sources but internally all mail is sent over SMTP. This is confusing behaviour to uncover but most importantly it could be potentially worrying. This pattern suggests that Gmail does not utilise ESMTP's advanced functionality, namely “SMTP AUTH”, on their internal mail network. This could potentially be evidence of Google leaving their network unintentionally open to attack by spammers, giving them access to sending unsolicited e-mail through Gmail as a relay.

A short paragraph must be given to note that not much has been said about Microsoft's externally visible technical capabilities. This is due to all of Microsoft's incoming mail and internal routing being impossible to classify as the “Received:” headers are only signed by “SMTPSVC” and a version number. This unfortunately provides no definitive answer as to whether ESMTP or SMTP has been used and as such, this data was excluded from the results.

Additionally, inspection of Yahoo's internal mail headers has revealed that Yahoo utilises its own bespoke non-standard protocol for internal traffic called NNFMF or “Newman No-Frills Mail Protocol” [43]. Analysing how efficient this protocol performs can not be explored however as no information is available about the protocol in the public domain.

4.5 Conclusion

Having analysed the e-mail traffic between four of the major service providers, some notable results were discovered. Most importantly within the scope of our paper is that ESMTP only holds approximately 50% of the visible e-mail traffic and this figure should be tracked over time to generate a trend. Secondly, further research should be undertaken into exploring why those using SMTP are still doing so as Google and Yahoo are potentially opening security holes in the global mailing system.

5. E-mail – The Message Perspective

Having explored the protocols to send e-mail in quite some depth, problems with these protocols were soon discovered. The standardisation of ESMTP was intended to solve these issues but

unfortunately, due to ESMTP's poor uptake and the bloated range of options, the probability that the problems introduced by SMTP having been solved are low. With this in mind, perhaps it is time to redesign the mail transfer protocols with efficiency built into the core of the protocol and with less focus on user centric operation. It would however be impossible to define a protocol capable of transferring messages in a lean & efficient way without first understanding the format of the messages themselves. Due to this, we looked at how the messages themselves are formatted and stored to see if improvements can be made locally to aid any improvements we may propose to be used globally.

With the task of transporting e-mail messages over a network formally updated and standardised, the format of messages also had to be updated to fit this model. This format was first defined in RFC 822 by David Crocker, August 13th 1982 [32].

RFC 822 “revises the specifications in RFC #733, in order to serve the needs of the larger and more complex ARPA Internet. Some of RFC #733's features failed to gain adequate acceptance. In order to simplify the standard and the software that follows it, these features have been removed. A different addressing scheme is used, to handle the case of inter-network mail; and the concept of re-transmission has been introduced” [33].

To create an Internet formatted message we define two sections, messages headers and message body. These two sections are separated from each other by a single blank line of text and were designed with different purposes in mind. Headers are found at the top of the message and define properties of the message, such as the sender, recipient(s) and the date. The message body however defines the actual content of the message to be displayed on the recipient's screen.

Every line of the message terminates with the sequence carriage return line feed “<CRLF>” and must be no longer than 1000 characters including the <CRLF>. The line should also be no longer than 78 characters including the <CRLF> for reasons of the notion of “netiquette” [36].

5.1 Internet Message Headers

```
Message-ID: <945878.112546296-sendEmail@adam-5750>
From: The big H <harry@portalmail.com>
To: "My dearest friend Ivy and her lovely daughter Eve"
    ivy@portalmail.com
Subject: Hello
Date: Fri, 23 Mar 2012 00:58:15 +0000
X-Mailer: sendEmail-1.56
```

My test message

Figure 11. Small example an e-mail message's content to explain header formatting rules

As headers are the first section of any e-mail message, we will discuss these first. Headers are represented as paired values and are in the format “HEADER FIELD HEADER VALUE”. For example, the header “Subject: Hello” from figure 11 above is defined as the header field “Subject:” and the header value “Hello”.

Header fields are only permitted to be single words, containing no white space and always end with the colon character. It is also possible to create non-standard headers by applying the prefix “X-” to the header. Additionally, all headers fields are case insensitive. “For example, the field-names “From”, “FROM”, “from”, and even “FroM” are semantically equal and should all be treated identically” [34]. Header values however are case sensitive and should never be altered without good cause. Some examples

of the most common headers found in e-mail messages are “To:”, “From”, “Subject:”, “X-Mailer:”, “Received:” and “Date:”.

The rules denoting the formatting of header value rules are even more complex, extensive and generally differ from header to header. For example, the formatting rules for headers storing dates will naturally differ to the formatting rules applied to addresses. One formatting rule common to all headers however is known as header folding.

The simplest headers fit onto a single line of text and are terminated by the <CRLF> sequence; Several examples of such headers can be seen in figure 11. When a header value is too long to fit onto a single line however, we have to employ header folding to make the message adhere to standards.

Most mail user agents fold headers when the header field and the header value together is longer than 78 characters including <CRLF> sequence but all mail user agents must perform folding if the header is longer than 1000 characters including <CRLF>. Although this sounds absurd as a possibility, e-mails with multiple recipients can quite easily overstep this value. To perform header folding, the composing program ends the header value prematurely with the <CRLF> sequence. The MUA then inserts white space at the beginning of the next line before continuing the header value as though no folding had occurred.

This folding allows for headers of a potentially infinite length and the “general rule is that wherever there may be linear-white-space (NOT simply LWSP-chars), a CRLF immediately followed by AT LEAST one LWSP-char may instead be inserted” [35]. An example of header folding can be seen in figure 11; The “To:” header has been folded to make the header fit within the constraints of this paper's formatting.

5.2 Addressing Format

The formatting of internet addresses in e-mail headers is also an area of particular interest, as exploring this area can give a good feel for how many differing ways header values in an e-mail can be formatted. An address consists of two parts, the user name and the domain name, which are separated by the “@” symbol. An address in this format is to be read as the user X located at the domain Y and can be represented in one of two formats in an e-mail header.

An address can either be simply shown in a plain text format or be represented as an encapsulated string using angular brackets. For example the address “ivy@portalmail.com” can either be represented in the form “To: ivy@portalmail.com” or as “To: <ivy@portalmail.com>” when placed into the header by the composing program.

Address headers also allow us to add comments to provide further information about the sender/recipient(s). For example, it's not uncommon to see e-mail signed by a person's name in place of the sender's address. How we attach these comments to address headers depends on the format used for the address however. If the e-mail address is represented as plain text for example, comments have to be placed between quotation or parenthesis markers. Alternatively, if the e-mail address is represented between angle brackets, any external text is interpreted as a plain text comment.

5.3 Message Body Format

The message body in comparison to the header partition is far simpler in terms of formatting rules. Separated from the headers by a blank line of text, the message body is only restricted by netiquette. Again, rules command that no more than 1000 characters including the <CRLF> is to be permitted and that limiting a line to 78 characters including the <CRLF> is

recommended. Other than this however the contents of the message body is not pinned down by any standardised rules and is able to contain any format of data.

5.4 Multimedia E-mail And Encoding Issues

When discussing the Simple Mail Transfer Protocol earlier in the text, some of the protocol's major flaws were neglected and were purposefully not discussed. Other than having authentication issues, SMTP also has problems transferring non-textual data with e-mail messages. Under SMTP, and even ESMTP, there is no native mechanism for the transferral of additional data with mail such as file attachments.

An additional problem also lies in SMTP's definition as a protocol. SMTP, as it was defined in 1982, is incapable of transferring binary in the modern 8 bit standard. The protocol is defined to be 7 bit clean and to operate exclusively within the US-ASCII character set. Even SMTP's successor ESMTP only allows for 8 bit data transfer if it is specifically supported as an extension. Due to this, sending 8 bit data or using other character sets in e-mail requires translation into a format that SMTP/ESMTP is capable of transferring.

"Commands and replies are composed of characters from the ASCII character set [1]. When the transport service provides an 8-bit byte (octet) transmission channel, each 7-bit character is transmitted right justified in an octet with the high order bit cleared to zero" [40].

Sending multimedia as a part of our e-mail messages however is a feature that we as users frequently take advantage of, and the use of other scripts such as Chinese is also permitted. As there is no built in protocol mechanisms to transfer this data, we have had to get creative to make this functionality fit. To achieve multimedia e-mail and different character sets, the data itself gets pushed into the body of an e-mail message. Doing so allows us to utilise other data types in such a manner that they transfer cleanly under already existing protocols. To achieve these features, mail user agents adopt the Multipurpose Internet Mail Extensions standard, alternatively known as "MIME".

5.5 Multipurpose Internet Mail Extensions

The purpose of MIME is to provide a recipient's MUA with a method to decompose a message's body into sections which can each be treated as their own individual message. As previously stated, the highest perspective of a message consists of headers and a message body; The message body itself can then be split into further sections of headers and body through the use of the MIME standard. These sections are processed individually by the recipient's MUA based on parameters specified by the sender's MUA.

It is this splitting of content that allows the receiver's MUA to decode the message and display each piece of content in an appropriate manner. To flag that this splitting occurs, the sending mail user agent tags the e-mail with standardised MIME Headers. First of these headers, and found in all MIME based e-mail, is the header "MIME-Version: 1.0". Coincidentally, this header is viewed by one its creators as a total waste of space. "The intention was to allow MIME to change, to advance to version 2.0 and so forth, but this decision led to the opposite outcome, making it nearly impossible to create a new version of the standard" [41].

Next and more important of MIME's specification is the provision of the "Content-Type:" header. The purpose of this header is to define the data-type and rules for the content that follows it. For example, if the header field value is set to "image/jpg", the receiving mail user agent knows to treat and display the following content as a jpg image. A full list of content

types can be found online at the site for the Internet Assigned Numbers Authority [42].

The use of MIME types only partially solves the problem of sending non-textual data however. The issue of how to encode non US-ASCII data to be SMTP safe still persists and we again look to MIME to provide further functionality to circumvent this restriction. To attach non US-ASCII data, we utilise the MIME header "Content-Transfer-Encoding:". Found shortly after MIME's "Content-Type:" header, the content transfer encoding header specifies the encoding scheme used to represent the content in the section. A few standardised encoding types are available, "7bit", "quoted-printable", "base64", "8bit" and "binary", each with their own rules. How these schemes operate can be easily researched through the publicly available RFC documents and as before, we are permitted to define non-standard forms of encoding through use of the prefix "X-" [44].

MIME as discussed thus far only permits the transportation of one data type in a message and no mechanisms to implement multimedia e-mail have been discussed. To explain how this is achieved requires a more detailed study of the "Content-Type:" header. Not only does this header allow the composing mail user agent to specify how the recipient needs to handle the following content at a data-type level, specifying "multipart" as the content type allows us to handle the trailing content as multimedia mail.

How multipart achieves multimedia mail is through the specification of additional information within the header. To do so, MIME permits the composing MUA to specify parameters in headers. These parameters are separated from each other by using the semi-colon as a delimiting character.

By specifying the boundary parameter, a mail user agent can successfully determine where the each piece of content starts and ends. When the type "multipart" is found, the MUA immediately knows that there must be several data types within the scope of the content. To separate these sections from each other requires a boundary and this boundary must be specified in the same header as the content's type as a parameter. Once the boundary value has been found, the mail user agent is fully prepared to decompose the message into sections that can be individually processed as individual MIME format messages.

Content-Type: multipart/mixed; boundary=paper

--paper

Content-Type: text/plain; charset=US-ASCII

This is some text

--paper

Content-Type: image/png;

Content-Transfer-Encoding: base64

(Image encoded as Base64)

--paper--

Figure 12. Example of boundaries used to split content types and create a multimedia message

A boundary's value is unique within the scope of the content the header precedes, and is prefixed by "--" at the start of each content's section. When finishing the last section of content, the boundary is again inserted but is instead both prefixed and suffixed by the string "--". Through use of these boundaries, the recipient's mail user agent is able to treat each section as having its own headers and message body, effectively creating messages inside messages.

Another commonly used parameter to circumvent SMTP's shortcomings and of a worthwhile mention is the "charset" parameter. It is the use of this parameter that permits us to use scripts other than US-ASCII when composing e-mail and opens the system up to the non-English speaking world.

5.6 Base64

With the basics of MIME briefly covered and explanations given for how it circumvents some of SMTP's shortcomings, it would be wise to cover Base64. Base64 is currently the most popular encoding scheme used to transfer files in both SMTP and ESMTP as it provides a simple method to convert modern 8 bit data into a form that is protocol safe. To achieve this, the binary data of a file is read 6 bits at a time & mapped onto an US-ASCII printable character.

0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Figure 13. Full table of Base64 mappings

The process of recovering the original binary is as simple as converting the US-ASCII character back to its 6 bit binary value. To cater for instances where the binary is not cleanly divisible by 6, the binary is padded with zeroed bits to make a full triplet. The padding character "=" is then used to flag how many six bit patterns were added to complete the Base64 string when it was encoded. Note, it is impossible to have a Base64 string terminated by three padding characters as the minimum input length is 8 bits, creating at least two valid Base64 characters.

```
00000000 00000001 00000010 >> 0 0 4 2 >> AAEC
00000000 00000001 >> 0 0 4 >> 00E=
00000000 >> 0 0 >> 00==
```

Figure 14. Example of binary, 6 bit values and Base64 values

The data is formatted into lines of 78 characters including the <CRLF> sequence, and is placed into the message body as US-ASCII text. This results in a completely safe and simple method for transferring binary data over SMTP/ESMTP, and reinforcing this design for simplicity is the purposeful exclusion of the characters "-" and "." from the encoding scheme. This prevents the encoded data from interfering with control characters used by SMTP, ESMTP and MIME protocols.

Base64 is not without its faults however as employing this encoding scheme results in an increase in file size of approximately 133%. This is due to every 6 bits of data requiring 8 bits of space on modern day computers to store. You may wonder why 7 bit or even 8 bit encoding schemes are not more popular for their better efficiency, and the reasoning for this is simple. SMTP, ESMTP and MIME all rely on patterns of characters for their correct operation, and using 7/8bit encoding schemes can accidentally break these standards by introducing illegal patterns into the data stream. For example, the message termination pattern "<CRLF>.<CRLF>" must be avoided at all costs when performing data transmission. If this pattern accidentally occurred in our data, it would signal to the receiving MTA that the message transfer has finished and the rest of the message would be lost.

6. Current Storage Formats

Knowing the internal structure of the data in an e-mail message and designing redundancy combating techniques here is not enough. These standards were designed with data efficiency in mind and though they are showing their age, Base64 encoding for example, we can not do anything radical in terms of changing the format of these messages. This is again due to standards being difficult to alter, enforcing requirements upon PortalMail so that it conforms to the norm and supports hypothetical users.

Any compression techniques designed and employed would need to be completely reversible. Any message entering our system and undergoing compression would need to be returned in its original format when retrieved by users. In the end, we were able to apply novel compression techniques to minimise the message content stored by PortalMail, but the true gains made originates from the techniques used to stores files.

Technically speaking, any form of database can store e-mail but two approaches have asserted themselves as standard over time. First and oldest of these two approaches is known as "mbox". The ideology behind this approach dates back to e-mails first beginnings in the mainframe era and conceptually the system is incredibly simple.

Under mbox, every user is assigned a mailbox file that holds their e-mail. The mailbox file is simply a text file where each message in the file is separated by a single blank line of text. When a new message is sent to the user, the mail delivery agent responsible for delivering the message simply appends the new data onto the end of the file. Aside from a few issues, the mbox approach is very simple and effective.

For those interested, some of the issues mbox faces includes locking problems for maintaining data consistency, mail management problems such as deletion, read speed issues as the newest messages are at the bottom of the file, and how to successfully flag the state of a message - new, read, junk etc.

The second commonly used approach is known as "Maildir". When employing Maildir systems, a user's mailbox is represented locally as a folder as opposed to a file. Inside of this folder are individual files, each representing an e-mail the user has sent/received. Systems based on this approach are simpler to lock, maintain, flag and navigate when compared to mbox as messages

are stored as manageable units. Unfortunately, Maildir systems also perform less efficiently in terms of their disk footprint.

6.1 Analysis of efficiency – mbox vs Maildir

Originally, our research was concerned with exploring e-mail with respect to overall data efficiency and has since been reduced in scope to the local storage of messages. As such, it was a logical step to examine how mbox and Maildir differentiate from each other with respect to their footprints on disk. Examining both of these systems provided us with invaluable information that aided us in designing the storage solution PortalMail utilises.

To analyse both systems, a collection of e-mail was downloaded from personal accounts using Microsoft's Windows Live Mail [45]. The messages were natively stored in the Maildir format and were analysed by using a small, simple Java program. The purpose of the program was to cycle through all of the downloaded files, calculating statistics for each block size group (4096 bytes, 8192 bytes etc). These statistics were then printed out in tab delimited format to be imported and analysed using spreadsheet software.

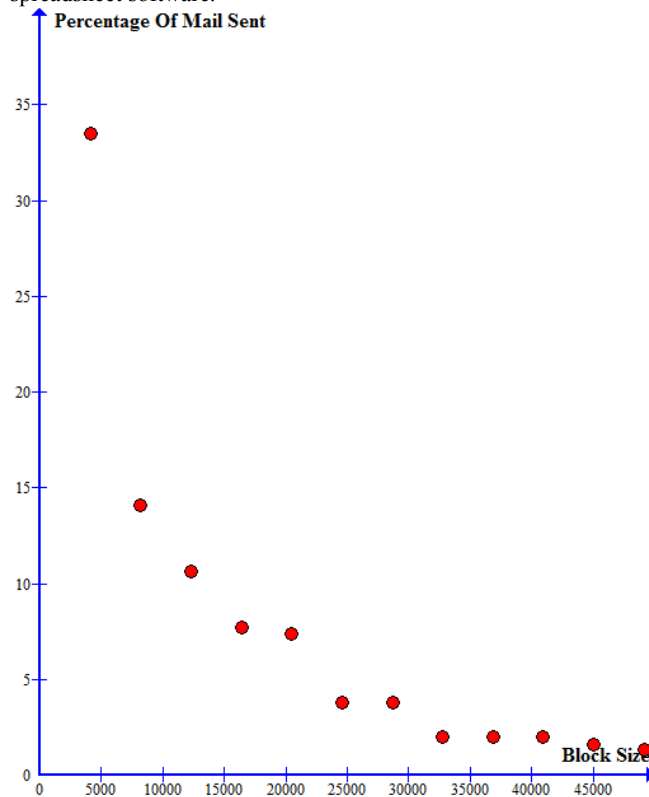


Figure 14. Distribution of e-mail by block size

One of the first things that was immediately clear is that out of the 2127 messages downloaded, most of them were short. A whole 33% of the mail was in fact small enough to fit onto a single 4KB disk block, averaging 2684 bytes of content per email and having a data storage efficiency of 65.5%. Additionally, as the length of a message increased, the frequency that messages of that length were sent also decreased.

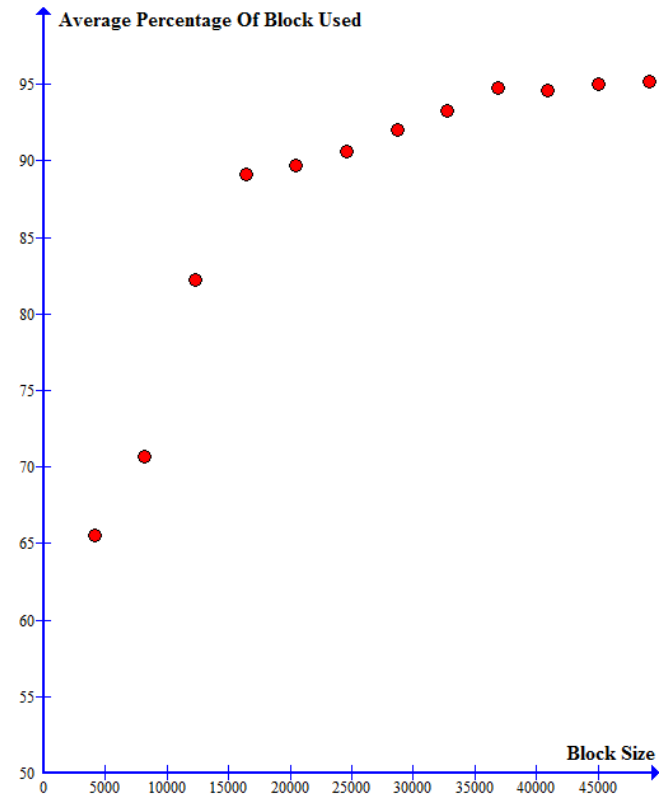


Figure 15. Average % used by blocks used

Another important trend visible in the data is that the average percentage of disk utilisation increases with the length of messages. This trend can be clearly seen in figure 15 above where as the block size is increased, the average % of block usage also increases. To store all 2127 messages in total required the use of 30789 4KB blocks, 126111744 bytes and of these consumed blocks, only 122007872 bytes were actually used. This calculates to be a total efficiency of 96.7% overall when using Maildir formatted storage.

With the efficiency of Maildir calculated, we also needed to analyse our data under mbox. To do so, further calculations were carried out upon our already existing data as opposed to undertaking a conversion process. As mentioned previously however each message in mbox is separated by a blank line in the file, <CRLF CRLF>, and this discrepancy needs accounting for. Multiplying the number of Maildir files by 4 bytes and increasing the total data count by this value removes this difference giving the total bytes mbox would require.

Our end results showed that if the e-mails were stored under mbox, the mailbox file would have required 29790 4KB blocks, 122019840 bytes of disk space. Of that space, 122016380 bytes would have been used up by data resulting in a total efficiency of 99.9%.

7. PortalMail

Having given a background of how e-mail operates and the areas where improvements can be made, it is time to introduce our prototype system "PortalMail". To summarise the paper thus far, there are four scopes to consider when judging the efficiency of modern e-mail systems. These are the network model level, the protocol level, the message content level and the file storage level.

We soon discovered problems in three of these levels, with only the network model perspective deemed as having no immediately visible flaws. As we can not easily change and test potential future standards however, we are left with only one area for immediate exploration: local storage techniques.

This is not negative news however as how we store e-mail locally can have a large impact on an e-mail system's performance. By compressing modern e-mail formats down to use as little disk space as possible, many benefits can be enjoyed. Operating a mail system with efficient storage for example would bring benefits to the corporation through reduced operating costs and benefits to end users through improvements in quality of service, such as being able to offer larger mailboxes.

PortalMail is our prototype system that attempts to achieve server-side data reduction. Written in C on the UNIX platform, PortalMail provides SMTP connectivity and is capable of receiving and delivering mail from the network to hypothetical local users. These messages are then saved in Maildir format, mbox format and our experimental format to allow for the fair testing of our proposed improvements.

8. File Storage Exploits

PortalMail's core technical edge, the inspiration for its name, is based upon the file storage techniques it utilises. Inspired by Dropbox, PortalMail is able to vastly reduce the data footprint of storing e-mail through the exploitation of file system functions provided by the local operating system.

Dropbox is a popular cloud based file hosting service that users employ to easily back up files. Dropbox's service is known for utilising data sharing algorithms to keep operating costs down [46], and PortalMail is designed to use a similar tact. When a file is uploaded to Dropbox, it is split by the client into chunks and hashed. If another user has already uploaded the same data chunks, the Dropbox client-side program does not copy these chunks to servers. Instead, Dropbox shares the already uploaded chunks between all of the users who own the same data, substantially reducing the amount of data they are required to host. PortalMail is designed with the aforementioned concept of data sharing in mind, but does not exercise an identical design however.

Why the two services are not identical in design is due to how Dropbox distributes processing load. When uploading files, Dropbox places the load of chunking and hashing files upon client machines in place of their own servers. To provide an identical chunk and hash set-up, without changing the mail transfer protocols, a PortalMail server would be required to actively chunk and hash all incoming messages on behalf of the client. This approach clearly would struggle to scale and as such, when designing PortalMail different techniques were taken to exploit and remove redundant data in mail.

Because actively searching for redundancy would be a heavy process to undertake, PortalMail uses passive systems to remove implicit redundancies automatically. How we achieve this through applying what we term "portal" and "cluster" messages when storing a e-mail messages on disk.

8.1 File Linking

As previously stated, PortalMail was built on top of the UNIX platform. This platform operates an inode based file system that when exploited, allows PortalMail to reduce the data footprint to 1/nth of the traditional size when compared to mbox/Maildir. How this is achieved through the promotion of sharing data between local users and the use of the "ln" system command.

The purpose of the "ln" system command is to create hard links on the UNIX file-system. When a hard link is formed on the file system, it creates a pointer to the data of the input file. This hard link looks and behaves identically to the original file. When the hard link is opened, it takes the user to the original file. This is analogous to creating a website where several hyper-links point to the same content.

Additionally, if changes are made to the data through one hard link, the changes that are made will be visible when the file is opened through another. This functionality is commonly used by system administrators to create file descriptors that to point to the same data put are placed under different file-names and directories. PortalMail however utilises these hard links to share e-mail data between local users.

8.2 Cluster Messages

The use of hard links allows us to achieve something that is unique to PortalMail, the compression of e-mail to 1/nth of its normal disk requirements (where n is the number of locally hosted recipients the message is intended for). As a worked example, consider an inbound e-mail targeted for ten local users that is 3 kilobytes long (the approximate average e-mail length for 33% of sent messages).

This e-mail if stored using mbox would require between 30-40 kilobytes of disk space and 40 kilobytes of space under Maildir. As there are 10 recipients, ten individual copies would be stored onto our server's disk drive. PortalMail in comparison requires only 4 kilobytes of data as the system only stores one copy of the message. If the message were sent to tens, hundreds or even thousands of our local users, it would still only require 4 kilobytes of server side disk space to store.

These 1-to-many messages are termed "cluster" messages, as much like networking clusters the contents spreads out in multiple directions. These cluster messages are stored in Maildir format and are hard linked to every recipient the message was intended for.

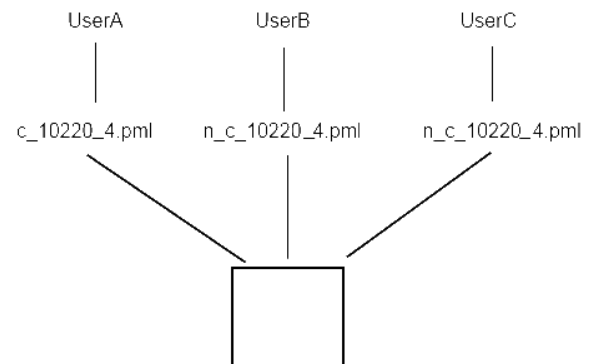


Figure 16. Example of a hard linked cluster message to three local users

The naming of the messages in figure 16 above is also of importance. When the message is received it is assigned a unique numerical identifier, a combination of system time and socket number. Flags are also appended to the file name for the purposes of checking mail, "n_" for new and "c_" for cluster message. Hence, in reference to figure 16 above, a cluster file has been hard linked to the mail boxes of users A, B and C as they are common recipients. Additionally, user A has already read the new message and as such the new message flag on their hard link is removed.

8.3 Portal Messages

Cluster messages as discussed above are stored using the Maildir system to prevent rapid message growth. As cluster messages are shared between multiple recipients, storage as mbox format files would lead to rapidly growing mailboxes that increasingly become more complex. Cluster class shared mbox storage would eventually lead to problems that are difficult to solve, such as how to make a system that would be able to successfully filter one user's inbox from another's in the mass pool of data.

Mbox based systems however have been shown to be clearly favourable in terms of their disk footprints. This led us to redesign how we store our shared files and to the definition of "portal" class messages. In contrast to cluster messages, a portal message is strictly a 1-to-1 connection between two participants.

These messages are stored using mbox formatting, giving portal class messages the same storage efficiency benefits, and for each and every e-mail address that the user has had any contact with there is a unique portal file. To clarify, whenever a new contact is made, the system creates a new portal file named after the other participant in the conversation. This occurs for all contacts whether local or external, bringing all 1-to-1 messages in line with mbox in terms of disk efficiency.

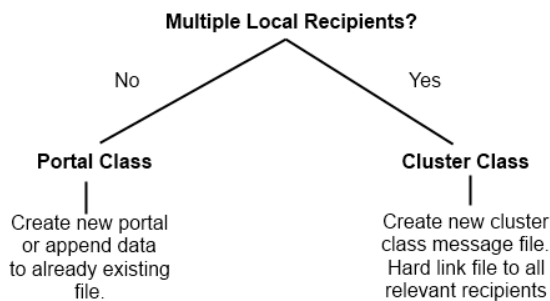


Figure 17. Decision chart showing what action to take when dealing with a new e-mail message

Figure 17. Flow chart of message type

The use of portal class messages in this manner can reduce the data storage requirements by half if the exchange is made between two locally held users. Alternatively, if the sender/recipient is not local to PortalMail, the system again uses portal class messages to maintain a consistent system design. Note though that as no data sharing can occur when this happens, no disk savings are made.

Additionally, portal class messages are named differently to cluster messages. Whereas cluster messages are named by at least the "c_" flag and a unique identifier, portal class messages are named after the other participant in the conversation, containing the contact's entire e-mail address.

9. E-mail File Compression

Although we could not change how e-mail is formatted when it is transmitted, nor how users expect to receive it, we did have room to implement lossless compression techniques to reduce the size of the e-mail itself.

There are two types of algorithms to consider when discussing the subject of compression, lossless and lossy. Lossless is a form of compression such that when data is compressed, the original data is fully recovered exactly how it was before compression when decompressed. These techniques are commonly used to compress files into zip folders. When operating such programs,

the input files are compounded into a single file & redundancies across the files are substituted for shorter codes, reducing the quantity of data required to store the file(s).

Lossy compression techniques however lose data through their execution. We as users however usually accept this loss as the data remains of a high enough quality that it can be deemed acceptable. These algorithms are most commonly found in most modern image/video formats. When using these formats, data is often smoothed out using wavelets to produce data that can be compressed with better results when compared to the original. The result is a visually observable decrease in image quality, but we accept this as we are still able to view and enjoy the image with enough clarity and detail to make the benefits outweigh the losses.

It is immediately clear when compressing e-mail data, lossless techniques have to be used. Any loss of data could prove disastrous to end users trying to read their mail. The loss or transformation of a single byte can result in the complete corruption of an e-mail message, undermining any perceived value users had in our service.

Additionally, any compression techniques applied have to be lightweight in terms of system resource usage. Full file compression algorithms, such as deflate [47], are to be avoided due to the sequence of processes that would be needed to append new mail to a user's portal class inbox. If full file compression were employed, a mailbox would need to be decompressed, the data appended and then the file re-compressed for each and every new message the user received. Any techniques we designed would need to be fast, fully reversible and capable of providing substantial amounts of compression to make it a worthwhile.

9.1 Header Compression

One of the first ideas we came up with was to use one of SMTP's own flaws against it. As previously mentioned, the simple mail transfer protocol is exclusive to the transferral of 7 bit US-ASCII data. This leaves our modern 8 bit system with 128 values (128-255) unused and open for exploitation. In reality, we limited ourselves to 127 of these values as under UNIX, as the value 255 marks the end of file.

These non-US-ASCII characters were then used to substitute common header fields. For every mail header PortalMail finds, the compression code for that header is written instead. To reverse the process is as simple as performing a look-up to resolve the original header field that was used. In total, PortalMail is capable of substituting 97 commonly used header fields at run time while delivering messages to recipients. Note, when using this technique it is possible to lose the original case of the headers in the message, but as header fields are case insensitive this is an invalid issue.

9.2 Cross Message Compression in Portal Messages

The use of mbox format 1-to-1 portal messages also provides scope for the removal of cross message redundancies. Initially, we hoped to have designed a system capable of removing replies within replies. When using the term "replies within replies", we are describing the data that results from the appending a previous e-mail message to the end of a newly composed reply. Such behaviour leads to unnecessary message growth with each response, but protocols as they stand don't support us when trying to enable our system with functions to remove this waste data. Every MUA appends old data differently and without clear and definitive boundaries to aid our search for common material, we would be required to use longest common substring algorithms to actively search out repeated sections of text.

Unfortunately, such algorithms run in $O(nm)$ time and if used would place an unnecessarily high strain on the system. This same problem presents itself in several places if we attempt to remove other cross message redundancies. For instance, the removal of signatures cross-message would again require us to actively perform substring comparisons. As such, we had to again find implicit redundancies present across messages to squeeze data out.

At the top of every portal class message is the full e-mail address of the user who made first contact. This address is hereafter known in the context of the portal message as the originator and the recipient is known as the subscriber. When appending messages, any string equivalent to the originator's e-mail address can be substituted for the originator non-US-ASCII character. Likewise, instances of the subscriber's e-mail address are substituted by the subscriber non-US-ASCII character.

When reading e-mail, the user immediately knows if it is the originator or the subscriber by inspecting the first line of the file. The system can then infer which address it needs to place whenever the originator/subscriber characters occur. If the reader is the originator, the subscriber's e-mail address is stored in the portal message's file name. Alternatively, if the reader is not the originator, the system knows that the readers own e-mail address must be the subscriber value.

9.3 Message Body Compression

As well as being able to remove redundancies in headers and across messages, PortalMail is able to remove redundancy inside message bodies. To do so, the system switches into MIME compression mode once the MIME-Version header has been found and compressed during header compression.

The system when in this mode compresses common MIME headers/parameters in the same way it performs header compression. By again substituting common strings using the 127 non-US-ASCII characters we can further the reduce data footprint with a static dictionary. Note, the 127 characters can be reused from the header compression stage due to the change in mode upon discovery of the MIME-Version header.

When operating in MIME compression mode, the system also searches for MIME boundaries and non-US-ASCII substitutes these boundaries whenever they occur within the message body. This technique alone cuts down data by roughly 30 bytes for each instance of the MIME boundary.

PortalMail also beings to show what can be achieved when processing MIME content types individually. Clearly, it would be beneficial to capable of applying differing compression techniques when processing MIME content. Each technique could be designed and specialised for differing types of data. For example, we could design a technique that is dedicated to the compression of HTML data. We can then go on creating other techniques and so forth for each data type we wished to compress. The area we believed to be the most advantageous to explore when dealing with the content of MIME e-mails however was the parsing of content sections to extract attachments.

As discussed previously in the text, MIME enables us to attach files to e-mails and these files are most commonly found as Base64 encoded data. When recording a new message, PortalMail parses the MIME sections to find, remove and share Base64 encoded files. Initially, the Base64 data is stored into a temporary file instead of the message in mid-delivery. Upon finishing the MIME section, the Base64 is then decoded back to binary to return the data back to its pre-encoded size. Finally the system uses a MD5 hash on the binary file. This result is appended to the incoming message in place of the Base64 data and also to name

the binary file. The hash is then used to check for collisions with previously decoded attachments held within a repository. If the file already exists and a collision has occurred, both the temporary Base64 and the binary files are deleted. Alternatively, if the file is new to the system, the binary is copied to the repository and preserved for potential future file transfers with identical attachments.

10. Evaluation

Unfortunately, we were unable to test the PortalMail system through real world usage. As the system is very new and designed to be an exploratory prototype, the robustness of the system is not capable of handling the full scope of message formats that system providers can produce. Additionally, we were also limited as to how many accounts worth of e-mail we could forward to our system. As PortalMail relies on the sharing of data between local accounts, and without a widespread data source, we can not hope to discover if the system is practical in the real world. As such, we assume that PortalMail is at least as efficient as Mbox/Maildir as it employs the same storage techniques. We additionally limit ourselves to performing tests that create fake messages designed to explore the repercussions of our design on storage efficiency.

First of these tests was simply to test if shared 1-to-1 files had tangible benefits. By creating and sending ten messages between two locally held users and simulating a "conversation", the following results were produced.

Compressed					
Bytes:	3758	Bytes	as	Blocks	4096
Maildir					
Bytes:	13754	Bytes	as	Blocks	81920
MBox					
Bytes:	13754	Bytes	as	Blocks	16384

Figure 18. Results gathered from portal messages experiment

It is immediately clear that the PortalMail system has gained strong benefits from employing 1-to-1 data sharing design. After applying compression techniques and enforcing data sharing, the total length of data in bytes has been reduced by 72% in comparison to its mail storage counterparts. Additionally, the disk footprint has been reduced by 95% and 75% for Maildir and Mbox respectively.

Next, we tested how well the system performed when using 1-to-many cluster messages. In this experiment, the e-mail utility was used to send 10 e-mail messages from an external address that were addressed to all of our 26 hypothetical users. Once run, the following results were produced.

Compressed					
Bytes:	16186	Bytes	as	Blocks	40960
Maildir					
Bytes:	495352	Bytes	as	Blocks	1064960
MBox					
Bytes:	495352	Bytes	as	Blocks	532480

Figure 19. Results gathered from cluster class message experiment

Again, it is immediately clear that PortalMail has the upper-hand in terms of storage technique. By sharing the message contents between all 26 recipients, the total file size has been reduced to 3% of the normally required space. Additionally, disk block usage is again lower when using portal mail, requiring 96/92% less in terms of storage space.

Finally we wanted to test and evaluate our third and final feature, Base64 removal, decoding & sharing. To do this, the image in figure 8 from above was attached to ten e-mails and sent from an external address. Of these ten e-mails, 5 were sent to one local user & five to the other. The results are as follows. (Note, these results also include the size of the extracted file)

Compressed

Bytes: 915688 Bytes as Blocks 921600

Maildir

Bytes: 12469534 Bytes as Blocks 12492800

MBox

Bytes: 12469534 Bytes as Blocks 12476416

Figure 20. Results gathered from attachment experiment

Again, it is immediately clear that PortalMail has advantageous traits. Requiring 92% less space than its counterparts, the extraction & removal of the Base64 data has proved incredibly effective. The PortalMail approach requires only 7% of the disk space when compared to Maildir and Mbox.

It is clear from these results that the PortalMail design is highly capable of scaling to high storage demand. With the potential to achieve 1/n message compression, the use of 1-to-1 sharing, 1-to-n sharing and system wide file sharing plays a key part to the success of a system modelled on the designs discussed in this paper.

11. Future Areas of Research

There is much to be done to improve the current state of e-mail systems. Surviving on decades old protocols and with little scope for improvements, the development of e-mail as a global system has seen many problems. In terms of future research, I suggest a global overhaul of the system and a true re-imagining of what e-mail can become.

In today's modern world, we can not afford to exclude the advancements we have made in technology. With social media rife on the Internet, more users online than ever before and an ever changing move to mobile networking, e-mail needs to again evolve to remain a central force in the future of the Internet.

With only imagination to restrict, there are countless areas where e-mail could be improved. For example, the allowance of mail-merged messages would allow for the transmitting of templates and then data, saving valuable bandwidth for companies that send messages in bulk. As caution however, such improvements should be made as future proof as possible to prevent the same problems of age arising again.

12. Conclusions

In this paper we have given a wide over-view of the modern mail system as a whole and brought to attention some of the issues that e-mail systems face daily. To begin the reform of e-mail, we have presented our prototype system "PortalMail" and provided results showing what the system is capable of. Unfortunately, there is still much scope for improvements in the system. Any improvements made must complement and support each other however. It is vital to ensure that any changes made do not interfere and produce the same problems through their extended use and development over time.

References

All hyper links were last known to be active 04/05/2012

[1]<http://www.nethistory.info/History%20of%20the%20Internet/email.html> - Age of email, first line

[2]http://socrates.berkeley.edu/~scotch/innovation/inventing_email.pdf - Article quote, first page 5th paragraph

[3]http://socrates.berkeley.edu/~scotch/innovation/inventing_email.pdf - Article quote, second page 7th paragraph

[4]http://socrates.berkeley.edu/~scotch/innovation/inventing_email.pdf - Heading quote, bottom of second page

[5]<http://www.livinginternet.com/e/ei.htm> - J.C.R Licklider quote

[6]<http://tools.ietf.org/html/rfc5322> - Internet Message Format RFC

[7]<http://tools.ietf.org/html/rfc2049> - Multipurpose Internet Mail Extensions RFC

[8]<http://www.computinghistory.org.uk/det/1787/Ray-Tomlinson/> - Ray Tomlinson quote, last paragraph

[9]<http://www.forbes.com/sites/marketshare/2011/07/27/even-in-a-social-world-email-is-still-the-killer-app/> - E-mail as a killer application, stated in headline

[10]<http://www.youtube.com/watch?v=RhD5IIHxCN0&t=1m7s> - Definition of killer application by Bill Gates

[11]<http://www.pkc.gov.uk/Planning+and+the+environment/Countryside+and+Wildlife/Public+Space+Management/Understanding+the+character+of+Perth+and+Kinross/Sir+Patrick+Geddes.htm> - The origins of "think global act local"

[12] O'Reilly - Programming Internet Email First Edition, David Wood, August 1999, 1-56592-479-7, page 4, 7th paragraph

[13]<http://www.isode.com/whitepapers/x400-messaging.html> - First paragraph, proof of 1984 origins

[14]<http://www.isode.com/whitepapers/x400-messaging.html> - First paragraph of "High Functionality" subsection

[15]<http://www.isode.com/whitepapers/x400-messaging.html> - Second section, second paragraph

[16]<http://www.isode.com/whitepapers/x400-messaging.html> - Where X.400 is used subsection, first paragraph

[17]<http://james.apache.org/server/rfclist/smt/rfc0821.txt> - Cover Page

[18]<http://james.apache.org/server/rfclist/smt/rfc0821.txt> - First paragraph of introduction

[19]<http://james.apache.org/server/rfclist/smt/rfc0821.txt> - Section 4.2 SMTP REPLIES, third paragraph

[20] Programming Internet Email First Edition, David Wood, August 1999, 1-56592-479-7, page 31

[21,22,23,24]<http://www.greenend.org.uk/rjk/tech/smtpreplies.html> - HELO, MAIL FROM, RCPT TO, DATA commands

[25]<http://www.lancs.ac.uk/email/rbl.php?ip=90.212.157.189&rbl=sh> - Lancaster Webmail blocks SMTP attempts off campus

[26] <http://smtpfilter.sourceforge.net/esmtp.html> - list of functions

[27] Programming Internet Email First Edition, David Wood, August 1999, 1-56592-479-7, page 354

[28] Programming Internet Email First Edition, David Wood, August 1999, 1-56592-479-7, page 354

[29] Programming Internet Email First Edition, David Wood, August 1999, 1-56592-479-7, page 13, fourth paragraph

[30] Programming Internet Email First Edition, David Wood, August 1999, 1-56592-479-7, page 354

[31] Programming Internet Email First Edition, David Wood, August 1999, 1-56592-479-7, page 19, third paragraph

[32]<http://www.ietf.org/rfc/rfc0822.txt> - RFC 822 cover page

[33]<http://www.ietf.org/rfc/rfc0822.txt> - RFC 822 preface, 2nd paragraph

[34]<http://www.ietf.org/rfc/rfc0822.txt> - RFC 822, page 14, third paragraph

[35]<http://www.ietf.org/rfc/rfc0822.txt> - RFC 822, section 3.1.1

[36]<http://networketiquette.net/> - Concept of netiquette

[37]Programming Internet Email First Edition, David Wood, August 1999, 1-56592-479-7
[38]<http://www.alvestrand.no/x400/debate/addressing.html> – Debate on the X.400 addressing standards
[39]<http://www.ietf.org/rfc/rfc1869.txt> - ESMTP RFC
[40]<http://james.apache.org/server/rfclist/smtp/rfc0821.txt> - RFC 821, page 3 paragraph 7
[41]<http://www.networkworld.com/news/2011/020111-mime-internet-email.html?page=3> - “The only version of MIME, ever” section, third paragraph

[42]<http://www.iana.org/assignments/media-types/index.html> - List of MIME Types
[43]http://en.wikipedia.org/wiki/Yahoo!_Mail#NNFMP - Yahoo's NNFMP
[44]<http://www.freesoft.org/CIE/RFC/1521/5.htm> – List of encoding types
[45]<http://windows.microsoft.com/en-US/windows-live/essentials-other-programs> - Windows Live Mail
[46]<http://www.fozworks.com/blog/2009/12/19/dropbox-knows-quite-a-lot-actually/> - DropBox chunking Tech
[47]<http://www.zlib.net/feldspar.html> – Deflate algorithm

All documents, programs and results can be found at <http://www.lancs.ac.uk/ug/lewisa3>