SENDMAIL — An Internetwork Mail Router

Eric Allman*

University of California, Berkeley
Mammoth Project

ABSTRACT

Routing mail through a heterogenous internet presents many new problems. Among the worst of these is that of address mapping. Historically, this has been handled on an *ad hoc* basis. However, this approach has become unmanageable as internets grow.

Sendmail acts a unified "post office" to which all mail can be submitted. Address interpretation is controlled by a production system, which can parse both domain-based addressing and old-style *ad hoc* addresses. The production system is powerful enough to rewrite addresses in the message header to conform to the standards of a number of common target networks, including old (NCP/RFC733) Arpanet, new (TCP/RFC822) Arpanet, UUCP, and Phonenet. Sendmail also implements an SMTP server, message queueing, and aliasing.

Sendmail implements a general internetwork mail routing facility, featuring aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

In a simple network, each node has an address, and resources can be identified with a host-resource pair; in particular, the mail system can refer to users using a host-username pair. Host names and numbers have to be administered by a central authority, but usernames can be assigned locally to each host.

In an internet, multiple networks with different characterstics and managements must communicate. In particular, the syntax and semantics of resource identification change. Certain special cases can be handled trivially by *ad hoc* techniques, such as providing network names that appear local to hosts on other networks, as with the Ethernet at Xerox PARC. However, the general case is extremely complex. For example, some networks require point-to-point routing, which simplifies the database update problem since only adjacent hosts must be entered into the system tables, while others use end-to-end addressing. Some networks use a left-associative syntax and others use a right-associative syntax, causing ambiguity in mixed addresses.

Internet standards seek to eliminate these problems. Initially, these proposed expanding the address pairs to address triples, consisting of {network, host, resource} triples. Network numbers must be universally agreed upon, and hosts can be assigned locally on each network. The user-level presentation was quickly expanded to address domains, comprised of a local resource identification and a hierarchical domain specification with a common static root. The domain technique separates the issue of physical versus logical addressing. For example, an address of the form ‘‘eric@a.cc.berkeley.arpa’’ describes only the logical organization of the address space.

Sendmail is intended to help bridge the gap between the totally *ad hoc* world of networks that know nothing of each other and the clean, tightly-coupled world of unique network numbers. It can accept old arbitrary address syntaxes, resolving ambiguities using heuristics specified by the system administrator, as well as domain-based addressing. It helps guide the conversion of message formats between disparate networks. In short, *sendmail* is designed to assist a graceful transition to consistent internetwork addressing

*A considerable part of this work was done while under the employ of the INGRES Project at the University of California at Berkeley and at Britton Lee.

schemes.

Section 1 discusses the design goals for *sendmail*. Section 2 gives an overview of the basic functions of the system. In section 3, details of usage are discussed. Section 4 compares *sendmail* to other internet mail routers, and an evaluation of *sendmail* is given in section 5, including future plans.

1. DESIGN GOALS

Design goals for *sendmail* include:

(1) Compatibility with the existing mail programs, including Bell version 6 mail, Bell version 7 mail [UNIX83], Berkeley *Mail* [Shoens79], BerkNet mail [Schmidt79], and hopefully UUCP mail [Nowitz78a, Nowitz78b]. ARPANET mail [Crocker77a, Postel77] was also required.

(2) Reliability, in the sense of guaranteeing that every message is correctly delivered or at least brought to the attention of a human for correct disposal; no message should ever be completely lost. This goal was considered essential because of the emphasis on mail in our environment. It has turned out to be one of the hardest goals to satisfy, especially in the face of the many anomalous message formats produced by various ARPANET sites. For example, certain sites generate improperly formated addresses, occasionally causing error-message loops. Some hosts use blanks in names, causing problems with UNIX mail programs that assume that an address is one word. The semantics of some fields are interpreted slightly differently by different sites. In summary, the obscure features of the ARPANET mail protocol really *are* used and are difficult to support, but must be supported.

(3) Existing software to do actual delivery should be used whenever possible. This goal derives as much from political and practical considerations as technical.

(4) Easy expansion to fairly complex environments, including multiple connections to a single network type (such as with multiple UUCP or Ether nets [Metcalfe76]). This goal requires consideration of the contents of an address as well as its syntax in order to determine which gateway to use. For example, the ARPANET is bringing up the TCP protocol to replace the old NCP protocol. No host at Berkeley runs both TCP and NCP, so it is necessary to look at the ARPANET host name to determine whether to route mail to an NCP gateway or a TCP gateway.

(5) Configuration should not be compiled into the code. A single compiled program should be able to run as is at any site (barring such basic changes as the CPU type or the operating system). We have found this seemingly unimportant goal to be critical in real life. Besides the simple problems that occur when any program gets recompiled in a different environment, many sites like to ‘‘fiddle’’ with anything that they will be recompiling anyway.

(6) *Sendmail* must be able to let various groups maintain their own mailing lists, and let individuals specify their own forwarding, without modifying the system alias file.

(7) Each user should be able to specify which mailer to execute to process mail being delivered for him. This feature allows users who are using specialized mailers that use a different format to build their environment without changing the system, and facilitates specialized functions (such as returning an ‘‘I am on vacation’’ message).

(8) Network traffic should be minimized by batching addresses to a single host where possible, without assistance from the user.

These goals motivated the architecture illustrated in figure 1. The user interacts with a mail generating and sending program. When the mail is created, the generator calls *sendmail*, which routes the message to the correct mailer(s). Since some of the senders may be network servers and some of the mailers may be network clients, *sendmail* may be used as an internet mail gateway.

2. OVERVIEW

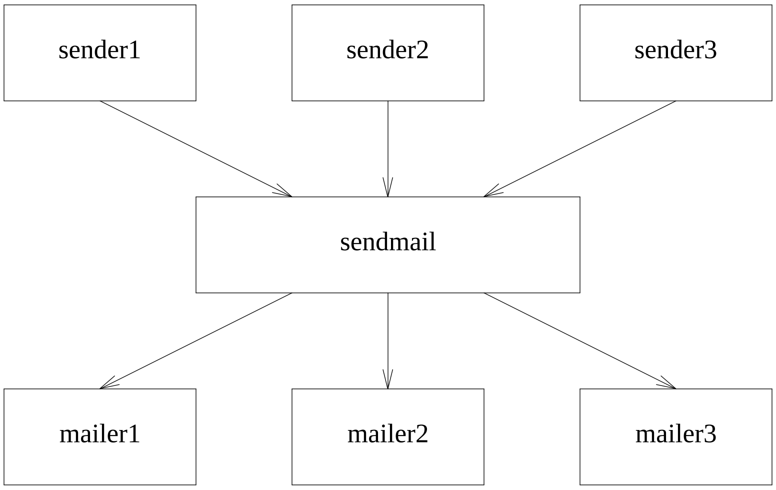

Figure 1 — Sendmail System Structure.

2.1. System Organization

Sendmail neither interfaces with the user nor does actual mail delivery. Rather, it collects a message generated by a user interface program (UIP) such as Berkeley *Mail*, MS [Crocker77b], or MH [Borden79], edits the message as required by the destination network, and calls appropriate mailers to do mail delivery or queueing for network transmission[1]. This discipline allows the insertion of new mailers at minimum cost. In this sense *sendmail* resembles the Message Processing Module (MPM) of [Postel79b].

2.2. Interfaces to the Outside World

There are three ways *sendmail* can communicate with the outside world, both in receiving and in sending mail. These are using the conventional UNIX argument vector/return status, speaking SMTP over a pair of UNIX pipes, and speaking SMTP over an interprocess(or) channel.

2.2.1. Argument vector/exit status

This technique is the standard UNIX method for communicating with the process. A list of recipients is sent in the argument vector, and the message body is sent on the standard input. Anything that the mailer prints is simply collected and sent back to the sender if there were any problems. The exit status from the mailer is collected after the message is sent, and a diagnostic is printed if appropriate.

2.2.2. SMTP over pipes

The SMTP protocol [Postel82] can be used to run an interactive lock-step interface with the mailer. A subprocess is still created, but no recipient addresses are passed to the mailer via the argument list. Instead, they are passed one at a time in commands sent to the processes standard input. Anything appearing on the standard output must be a reply code in a special format.

[1]except when mailing to a file, when *sendmail* does the delivery directly.

2.2.3. SMTP over an IPC connection

This technique is similar to the previous technique, except that it uses a 4.2bsd IPC channel [UNIX83]. This method is exceptionally flexible in that the mailer need not reside on the same machine. It is normally used to connect to a sendmail process on another machine.

2.3. Operational Description

When a sender wants to send a message, it issues a request to *sendmail* using one of the three methods described above. *Sendmail* operates in two distinct phases. In the first phase, it collects and stores the message. In the second phase, message delivery occurs. If there were errors during processing during the second phase, *sendmail* creates and returns a new message describing the error and/or returns an status code telling what went wrong.

2.3.1. Argument processing and address parsing

If *sendmail* is called using one of the two subprocess techniques, the arguments are first scanned and option specifications are processed. Recipient addresses are then collected, either from the command line or from the SMTP RCPT command, and a list of recipients is created. Aliases are expanded at this step, including mailing lists. As much validation as possible of the addresses is done at this step: syntax is checked, and local addresses are verified, but detailed checking of host names and addresses is deferred until delivery. Forwarding is also performed as the local addresses are verified.

Sendmail appends each address to the recipient list after parsing. When a name is aliased or forwarded, the old name is retained in the list, and a flag is set that tells the delivery phase to ignore this recipient. This list is kept free from duplicates, preventing alias loops and duplicate messages deliverd to the same recipient, as might occur if a person is in two groups.

2.3.2. Message collection

Sendmail then collects the message. The message should have a header at the beginning. No formatting requirements are imposed on the message except that they must be lines of text (i.e., binary data is not allowed). The header is parsed and stored in memory, and the body of the message is saved in a temporary file.

To simplify the program interface, the message is collected even if no addresses were valid. The message will be returned with an error.

2.3.3. Message delivery

For each unique mailer and host in the recipient list, *sendmail* calls the appropriate mailer. Each mailer invocation sends to all users receiving the message on one host. Mailers that only accept one recipient at a time are handled properly.

The message is sent to the mailer using one of the same three interfaces used to submit a message to sendmail. Each copy of the message is prepended by a customized header. The mailer status code is caught and checked, and a suitable error message given as appropriate. The exit code must conform to a system standard or a generic message (‘‘Service unavailable’’) is given.

2.3.4. Queueing for retransmission

If the mailer returned an status that indicated that it might be able to handle the mail later, *sendmail* will queue the mail and try again later.

2.3.5. Return to sender

If errors occur during processing, *sendmail* returns the message to the sender for retransmission. The letter can be mailed back or written in the file ‘‘dead.letter’’ in the sender’s home directory[2].

2.4. Message Header Editing

Certain editing of the message header occurs automatically. Header lines can be inserted under control of the configuration file. Some lines can be merged; for example, a ‘‘From:’’ line and a ‘‘Full-name:’’ line can be merged under certain circumstances.

2.5. Configuration File

Almost all configuration information is read at runtime from an ASCII file, encoding macro definitions (defining the value of macros used internally), header declarations (telling sendmail the format of header lines that it will process specially, i.e., lines that it will add or reformat), mailer definitions (giving information such as the location and characteristics of each mailer), and address rewriting rules (a limited production system to rewrite addresses which is used to parse and rewrite the addresses).

To improve performance when reading the configuration file, a memory image can be provided. This provides a ‘‘compiled’’ form of the configuration file.

3. USAGE AND IMPLEMENTATION

3.1. Arguments

Arguments may be flags and addresses. Flags set various processing options. Following flag arguments, address arguments may be given, unless we are running in SMTP mode. Addresses follow the syntax in RFC822 [Crocker82] for ARPANET address formats. In brief, the format is:

(1) Anything in parentheses is thrown away (as a comment).

(2) Anything in angle brackets (‘‘< >’’) is preferred over anything else. This rule implements the ARPANET standard that addresses of the form

    user name <machine-address>

will send to the electronic ‘‘machine-address’’ rather than the human ‘‘user name.’’

(3) Double quotes (") quote phrases; backslashes quote characters. Backslashes are more powerful in that they will cause otherwise equivalent phrases to compare differently — for example, *user* and *"user"* are equivalent, but *\user* is different from either of them.

Parentheses, angle brackets, and double quotes must be properly balanced and nested. The rewriting rules control remaining parsing[3].

3.2. Mail to Files and Programs

Files and programs are legitimate message recipients. Files provide archival storage of messages, useful for project administration and history. Programs are useful as recipients in a variety of situations, for example, to maintain a public repository of systems messages (such as the Berkeley *msgs* program, or the MARS system [Sattley78]).

Any address passing through the initial parsing algorithm as a local address (i.e, not appearing to be a valid address for another mailer) is scanned for two special cases. If prefixed by a vertical bar (‘‘|’’) the rest of the address is processed as a shell command. If the user name begins with a slash mark (‘‘/’’) the name is used as a file name, instead of a login name.

Files that have setuid or setgid bits set but no execute bits set have those bits honored if *sendmail* is running as root.

---

[2] Obviously, if the site giving the error is not the originating site, the only reasonable option is to mail back to the sender. Also, there are many more error disposition options, but they only effect the error message — the ‘‘return to sender’’ function is always handled in one of these two ways.

[3] Disclaimer: Some special processing is done after rewriting local names; see below.

3.3. Aliasing, Forwarding, Inclusion

Sendmail reroutes mail three ways. Aliasing applies system wide. Forwarding allows each user to reroute incoming mail destined for that account. Inclusion directs *sendmail* to read a file for a list of addresses, and is normally used in conjunction with aliasing.

3.3.1. Aliasing

Aliasing maps names to address lists using a system-wide file. This file is indexed to speed access. Only names that parse as local are allowed as aliases; this guarantees a unique key (since there are no nicknames for the local host).

3.3.2. Forwarding

After aliasing, recipients that are local and valid are checked for the existence of a ‘‘.forward’’ file in their home directory. If it exists, the message is *not* sent to that user, but rather to the list of users in that file. Often this list will contain only one address, and the feature will be used for network mail forwarding.

Forwarding also permits a user to specify a private incoming mailer. For example, forwarding to:

"|/usr/local/newmail myname"

will use a different incoming mailer.

3.3.3. Inclusion

Inclusion is specified in RFC 733 [Crocker77a] syntax:

:Include: pathname

An address of this form reads the file specified by *pathname* and sends to all users listed in that file.

The intent is *not* to support direct use of this feature, but rather to use this as a subset of aliasing. For example, an alias of the form:

project: :include:/usr/project/userlist

is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

It is not necessary to rebuild the index on the alias database when a :include: list is changed.

3.4. Message Collection

Once all recipient addresses are parsed and verified, the message is collected. The message comes in two parts: a message header and a message body, separated by a blank line.

The header is formatted as a series of lines of the form

field-name: field-value

Field-value can be split across lines by starting the following lines with a space or a tab. Some header fields have special internal meaning, and have appropriate special processing. Other headers are simply passed through. Some header fields may be added automatically, such as time stamps.

The body is a series of text lines. It is completely uninterpreted and untouched, except that lines beginning with a dot have the dot doubled when transmitted over an SMTP channel. This extra dot is stripped by the receiver.

3.5. Message Delivery

The send queue is ordered by receiving host before transmission to implement message batching. Each address is marked as it is sent so rescanning the list is safe. An argument list is built as the scan proceeds. Mail to files is detected during the scan of the send list. The interface to the mailer is performed using one of the techniques described in section 2.2.

After a connection is established, *sendmail* makes the per-mailer changes to the header and sends the result to the mailer. If any mail is rejected by the mailer, a flag is set to invoke the return-to-sender function after all delivery completes.

3.6. Queued Messages

If the mailer returns a ‘‘temporary failure’’ exit status, the message is queued. A control file is used to describe the recipients to be sent to and various other parameters. This control file is formatted as a series of lines, each describing a sender, a recipient, the time of submission, or some other salient parameter of the message. The header of the message is stored in the control file, so that the associated data file in the queue is just the temporary file that was originally collected.

3.7. Configuration

Configuration is controlled primarily by a configuration file read at startup. *Sendmail* should not need to be recomplied except

(1) To change operating systems (V6, V7/32V, 4BSD).

(2) To remove or insert the DBM (UNIX database) library.

(3) To change ARPANET reply codes.

(4) To add headers fields requiring special processing.

Adding mailers or changing parsing (i.e., rewriting) or routing information does not require recompilation.

If the mail is being sent by a local user, and the file ‘‘.mailcf’’ exists in the sender’s home directory, that file is read as a configuration file after the system configuration file. The primary use of this feature is to add header lines.

The configuration file encodes macro definitions, header definitions, mailer definitions, rewriting rules, and options.

3.7.1. Macros

Macros can be used in three ways. Certain macros transmit unstructured textual information into the mail system, such as the name *sendmail* will use to identify itself in error messages. Other macros transmit information from *sendmail* to the configuration file for use in creating other fields (such as argument vectors to mailers); e.g., the name of the sender, and the host and user of the recipient. Other macros are unused internally, and can be used as shorthand in the configuration file.

3.7.2. Header declarations

Header declarations inform *sendmail* of the format of known header lines. Knowledge of a few header lines is built into *sendmail*, such as the ‘‘From:’’ and ‘‘Date:’’ lines.

Most configured headers will be automatically inserted in the outgoing message if they don’t exist in the incoming message. Certain headers are suppressed by some mailers.

3.7.3. Mailer declarations

Mailer declarations tell *sendmail* of the various mailers available to it. The definition specifies the internal name of the mailer, the pathname of the program to call, some flags associated with the mailer, and an argument vector to be used on the call; this vector is macro-expanded before use.

3.7.4. Address rewriting rules

The heart of address parsing in *sendmail* is a set of rewriting rules. These are an ordered list of pattern-replacement rules, (somewhat like a production system, except that order is critical), which are applied to each address. The address is rewritten textually until it is either rewritten into a special canonical form (i.e., a (mailer, host, user) 3-tuple, such as {arpanet, usc-isif, postel} representing the address ‘‘postel@usc-isif’’), or it falls off the end. When a pattern matches, the rule is reapplied until it fails.

The configuration file also supports the editing of addresses into different formats. For example, an address of the form:

ucsfcgl!tef

might be mapped into:

tef@ucsfcgl.UUCP

to conform to the domain syntax. Translations can also be done in the other direction.

3.7.5. Option setting

There are several options that can be set from the configuration file. These include the pathnames of various support files, timeouts, default modes, etc.

4. COMPARISON WITH OTHER MAILERS

4.1. Delivermail

Sendmail is an outgrowth of *delivermail*. The primary differences are:

(1) Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also allows easy debugging of new mailers.

(2) Address parsing is more flexible. For example, *delivermail* only supported one gateway to any network, whereas *sendmail* can be sensitive to host names and reroute to different gateways.

(3) Forwarding and :include: features eliminate the requirement that the system alias file be writable by any user (or that an update program be written, or that the system administration make all changes).

(4) *Sendmail* supports message batching across networks when a message is being sent to multiple recipients.

(5) A mail queue is provided in *sendmail.* Mail that cannot be delivered immediately but can potentially be delivered later is stored in this queue for a later retry. The queue also provides a buffer against system crashes; after the message has been collected it may be reliably redelivered even if the system crashes during the initial delivery.

(6) *Sendmail* uses the networking support provided by 4.2BSD to provide a direct interface networks such as the ARPANET and/or Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

4.2. MMDF

MMDF [Crocker79] spans a wider problem set than *sendmail*. For example, the domain of MMDF includes a ‘‘phone network’’ mailer, whereas *sendmail* calls on preexisting mailers in most cases.

MMDF and *sendmail* both support aliasing, customized mailers, message batching, automatic forwarding to gateways, queueing, and retransmission. MMDF supports two-stage timeout, which *sendmail* does not support.

The configuration for MMDF is compiled into the code[4].

[4]Dynamic configuration tables are currently being considered for MMDF; allowing the installer to select either compiled or dynamic tables.

Since MMDF does not consider backwards compatibility as a design goal, the address parsing is simpler but much less flexible.

It is somewhat harder to integrate a new channel[5] into MMDF. In particular, MMDF must know the location and format of host tables for all channels, and the channel must speak a special protocol. This allows MMDF to do additional verification (such as verifying host names) at submission time.

MMDF strictly separates the submission and delivery phases. Although *sendmail* has the concept of each of these stages, they are integrated into one program, whereas in MMDF they are split into two programs.

4.3. Message Processing Module

The Message Processing Module (MPM) discussed by Postel [Postel79b] matches *sendmail* closely in terms of its basic architecture. However, like MMDF, the MPM includes the network interface software as part of its domain.

MPM also postulates a duplex channel to the receiver, as does MMDF, thus allowing simpler handling of errors by the mailer than is possible in *sendmail*. When a message queued by *sendmail* is sent, any errors must be returned to the sender by the mailer itself. Both MPM and MMDF mailers can return an immediate error response, and a single error processor can create an appropriate response.

MPM prefers passing the message as a structured object, with type-length-value tuples[6]. Such a convention requires a much higher degree of cooperation between mailers than is required by *sendmail*. MPM also assumes a universally agreed upon internet name space (with each address in the form of a net-host-user tuple), which *sendmail* does not.

5. EVALUATIONS AND FUTURE PLANS

Sendmail is designed to work in a nonhomogeneous environment. Every attempt is made to avoid imposing unnecessary constraints on the underlying mailers. This goal has driven much of the design. One of the major problems has been the lack of a uniform address space, as postulated in [Postel79a] and [Postel79b].

A nonuniform address space implies that a path will be specified in all addresses, either explicitly (as part of the address) or implicitly (as with implied forwarding to gateways). This restriction has the unpleasant effect of making replying to messages exceedingly difficult, since there is no one ‘‘address’’ for any person, but only a way to get there from wherever you are.

Interfacing to mail programs that were not initially intended to be applied in an internet environment has been amazingly successful, and has reduced the job to a manageable task.

Sendmail has knowledge of a few difficult environments built in. It generates ARPANET FTP/SMTP compatible error messages (prepended with three-digit numbers [Neigus73, Postel74, Postel82]) as necessary, optionally generates UNIX-style ‘‘From’’ lines on the front of messages for some mailers, and knows how to parse the same lines on input. Also, error handling has an option customized for BerkNet.

The decision to avoid doing any type of delivery where possible (even, or perhaps especially, local delivery) has turned out to be a good idea. Even with local delivery, there are issues of the location of the mailbox, the format of the mailbox, the locking protocol used, etc., that are best decided by other programs. One surprisingly major annoyance in many internet mailers is that the location and format of local mail is built in. The feeling seems to be that local mail is so common that it should be efficient. This feeling is not born out by our experience; on the contrary, the location and format of mailboxes seems to vary widely from system to system.

---

[5] The MMDF equivalent of a *sendmail* ‘‘mailer.’’

[6] This is similar to the NBS standard.

The ability to automatically generate a response to incoming mail (by forwarding mail to a program) seems useful (‘‘I am on vacation until late August....’’) but can create problems such as forwarding loops (two people on vacation whose programs send notes back and forth, for instance) if these programs are not well written. A program could be written to do standard tasks correctly, but this would solve the general case.

It might be desirable to implement some form of load limiting. I am unaware of any mail system that addresses this problem, nor am I aware of any reasonable solution at this time.

The configuration file is currently practically inscrutable; considerable convenience could be realized with a higher-level format.

It seems clear that common protocols will be changing soon to accommodate changing requirements and environments. These changes will include modifications to the message header (e.g., [NBS80]) or to the body of the message itself (such as for multimedia messages [Postel80]). Experience indicates that these changes should be relatively trivial to integrate into the existing system.

In tightly coupled environments, it would be nice to have a name server such as Grapvine [Birrell82] integrated into the mail system. This would allow a site such as ‘‘Berkeley’’ to appear as a single host, rather than as a collection of hosts, and would allow people to move transparently among machines without having to change their addresses. Such a facility would require an automatically updated database and some method of resolving conflicts. Ideally this would be effective even without all hosts being under a single management. However, it is not clear whether this feature should be integrated into the aliasing facility or should be considered a ‘‘value added’’ feature outside *sendmail* itself.

As a more interesting case, the CSNET name server [Solomon81] provides an facility that goes beyond a single tightly-coupled environment. Such a facility would normally exist outside of *sendmail* however.

ACKNOWLEDGEMENTS

Thanks are due to Kurt Shoens for his continual cheerful assistance and good advice, Bill Joy for pointing me in the correct direction (over and over), and Mark Horton for more advice, prodding, and many of the good ideas. Kurt and Eric Schmidt are to be credited for using *delivermail* as a server for their programs (*Mail* and BerkNet respectively) before any sane person should have, and making the necessary modifications promptly and happily. Eric gave me considerable advice about the perils of network software which saved me an unknown amount of work and grief. Mark did the original implementation of the DBM version of aliasing, installed the VFORK code, wrote the current version of *rmail*, and was the person who really convinced me to put the work into *delivermail* to turn it into *sendmail*. Kurt deserves accolades for using *sendmail* when I was myself afraid to take the risk; how a person can continue to be so enthusiastic in the face of so much bitter reality is beyond me.

Kurt, Mark, Kirk McKusick, Marvin Solomon, and many others have reviewed this paper, giving considerable useful advice.

Special thanks are reserved for Mike Stonebraker at Berkeley and Bob Epstein at Britton-Lee, who both knowingly allowed me to put so much work into this project when there were so many other things I really should have been working on.

REFERENCES

[Birrell82] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M. D., ‘‘Grapevine: An Exercise in Distributed Computing.’’ In *Comm. A.C.M. 25,* 4, April 82.

[Borden79] Borden, S., Gaines, R. S., and Shapiro, N. Z., *The MH Message Handling System: Users' Manual.* R-2367-PAF. Rand Corporation. October 1979.

[Crocker77a] Crocker, D. H., Vittal, J. J., Pogran, K. T., and Henderson, D. A. Jr., *Standard for the Format of ARPA Network Text Messages.* RFC 733, NIC 41952. In [Feinler78]. November 1977.

[Crocker77b] Crocker, D. H., *Framework and Functions of the MS Personal Message System.* R-2134-ARPA, Rand Corporation, Santa Monica, California. 1977.

[Crocker79] Crocker, D. H., Szurkowski, E. S., and Farber, D. J., *An Internetwork Memo Distribution Facility — MMDF.* 6th Data Communication Symposium, Asilomar. November 1979.

[Crocker82] Crocker, D. H., *Standard for the Format of Arpa Internet Text Messages.* RFC 822. Network Information Center, SRI International, Menlo Park, California. August 1982.

[Metcalfe76] Metcalfe, R., and Boggs, D., ‘‘Ethernet: Distributed Packet Switching for Local Computer Networks’’, *Communications of the ACM 19,* 7. July 1976.

[Feinler78] Feinler, E., and Postel, J. (eds.), *ARPANET Protocol Handbook.* NIC 7104, Network Information Center, SRI International, Menlo Park, California. 1978.

[NBS80] National Bureau of Standards, *Specification of a Draft Message Format Standard.* Report No. ICST/CBOS 80-2. October 1980.

[Neigus73] Neigus, N., *File Transfer Protocol for the ARPA Network.* RFC 542, NIC 17759. In [Feinler78]. August, 1973.

[Nowitz78a] Nowitz, D. A., and Lesk, M. E., *A Dial-Up Network of UNIX Systems.* Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. August, 1978.

[Nowitz78b] Nowitz, D. A., *Uucp Implementation Description.* Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. October, 1978.

[Postel74] Postel, J., and Neigus, N., Revised FTP Reply Codes. RFC 640, NIC 30843. In [Feinler78]. June, 1974.

[Postel77] Postel, J., *Mail Protocol.* NIC 29588. In [Feinler78]. November 1977.

[Postel79a] Postel, J., *Internet Message Protocol.* RFC 753, IEN 85. Network Information Center, SRI International, Menlo Park, California. March 1979.

[Postel79b] Postel, J. B., *An Internetwork Message Structure.* In *Proceedings of the Sixth Data Communications Symposium,* IEEE. New York. November 1979.

[Postel80] Postel, J. B., *A Structured Format for Transmission of Multi-Media Documents.* RFC 767. Network Information Center, SRI International, Menlo Park, California. August 1980.

[Postel82] Postel, J. B., *Simple Mail Transfer Protocol.* RFC821 (obsoleting RFC788). Network Information Center, SRI International, Menlo Park, California. August 1982.

[Schmidt79] Schmidt, E., *An Introduction to the Berkeley Network.* University of California, Berkeley California. 1979.

[Shoens79] Shoens, K., *Mail Reference Manual.* University of California, Berkeley. In UNIX Programmer's Manual, Seventh Edition, Volume 2C. December 1979.

[Sluizer81] Sluizer, S., and Postel, J. B., *Mail Transfer Protocol.* RFC 780. Network Information Center, SRI International, Menlo Park, California. May 1981.

[Solomon81] Solomon, M., Landweber, L., and Neuhengen, D., ‘‘The Design of the CSNET Name Server.’’ CS-DN-2, University of Wisconsin, Madison. November 1981.

[Su82] Su, Zaw-Sing, and Postel, Jon, *The Domain Naming Convention for Internet User Applications.* RFC819. Network Information Center, SRI International, Menlo Park, California. August 1982.

[UNIX83] *The UNIX Programmer’s Manual, Seventh Edition*, Virtual VAX-11 Version, Volume 1. Bell Laboratories, modified by the University of California, Berkeley, California. March, 1983.