Mail Systems and Addressing in 4.2bsd

Eric Allman*

Britton-Lee, Inc.
1919 Addison Street, Suite 105.
Berkeley, California 94704.

eric@Berkeley.ARPA
ucbvax!eric

ABSTRACT

Routing mail through a heterogeneous 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 old and new format addresses. The new format is "domain-based," a flexible technique that can handle many common situations. Sendmail is not intended to perform user interface functions.

Sendmail will replace delivermail in the Berkeley 4.2 distribution. Several major hosts are now or will soon be running sendmail. This change will affect any users that route mail through a sendmail gateway. The changes that will be user visible are emphasized.

The mail system to appear in 4.2bsd will contain a number of changes. Most of these changes are based on the replacement of *delivermail* with a new module called *sendmail. Sendmail* implements a general internetwork mail routing facility, featuring aliasing and forwarding, automatic routing to network gateways, and flexible configuration. Of key interest to the mail system user will be the changes in the network addressing structure.

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 characteristics 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 that the route the message takes be explicitly specified by the sender, simplifying the database update problem since only adjacent hosts must be entered into the system tables, while others use logical addressing, where the sender specifies the location of the recipient but not how to get there. Some networks use a left-associative syntax and others use a right-associative syntax, causing

---

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

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, username} triples. Network numbers must be universally agreed upon, and hosts can be assigned locally on each network. The user-level presentation was changed 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 the logical organization of the address space (user "eric" on host "a" in the Computer Center at Berkeley) but not the physical networks used (for example, this could go over different networks depending on whether "a" were on an ethernet or a store-and-forward network).

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 schemes.

Section 1 defines some of the terms frequently left fuzzy when working in mail systems. Section 2 discusses the design goals for *sendmail*. In section 3, the new address formats and basic features of *sendmail* are described. Section 4 discusses some of the special problems of the UUCP network. The differences between *sendmail* and *delivermail* are presented in section 5.

> **DISCLAIMER:** A number of examples in this paper use names of actual people and organizations. This is not intended to imply a commitment or even an intellectual agreement on the part of these people or organizations. In particular, Bell Telephone Laboratories (BTL), Digital Equipment Corporation (DEC), Lawrence Berkeley Laboratories (LBL), Britton-Lee Incorporated (BLI), and the University of California at Berkeley are not committed to any of these proposals at this time. Much of this paper represents no more than the personal opinions of the author.

1. DEFINITIONS

There are four basic concepts that must be clearly distinguished when dealing with mail systems: the user (or the user's agent), the user's identification, the user's address, and the route. These are distinguished primarily by their position independence.

1.1. User and Identification

The user is the being (a person or program) that is creating or receiving a message. An *agent* is an entity operating on behalf of the user — such as a secretary who handles my mail. or a program that automatically returns a message such as "I am at the UNICOM conference."

The identification is the tag that goes along with the particular user. This tag is completely independent of location. For example, my identification is the string "Eric Allman," and this identification does not change whether I am located at U.C. Berkeley, at Britton-Lee, or at a scientific institute in Austria.

Since the identification is frequently ambiguous (e.g., there are two "Robert Henry"s at Berkeley) it is common to add other disambiguating information that is not strictly part of the identification (e.g., Robert "Code Generator" Henry versus Robert "System Administrator" Henry).

1.2. Address

The address specifies a location. As I move around, my address changes. For example, my address might change from “eric@Berkeley.ARPA” to “eric@bli.UUCP” or “allman@IIASA.Austria” depending on my current affiliation.

However, an address is independent of the location of anyone else. That is, my address remains the same to everyone who might be sending me mail. For example, a person at MIT and a person at USC could both send to “eric@Berkeley.ARPA” and have it arrive to the same mailbox.

Ideally a “white pages” service would be provided to map user identifications into addresses (for example, see [Solomon81]). Currently this is handled by passing around scraps of paper or by calling people on the telephone to find out their address.

1.3. Route

While an address specifies *where* to find a mailbox, a route specifies *how* to find the mailbox. Specifically, it specifies a path from sender to receiver. As such, the route is potentially different for every pair of people in the electronic universe.

Normally the route is hidden from the user by the software. However, some networks put the burden of determining the route onto the sender. Although this simplifies the software, it also greatly impairs the usability for most users. The UUCP network is an example of such a network.

2. DESIGN GOALS

Design goals for *sendmail*[1] include:

(1) Compatibility with the existing mail programs, including Bell version 6 mail, Bell version 7 mail, Berkeley *Mail* [Shoens79], BerkNet mail [Schmidt79], and hopefully UUCP mail [Nowitz78]. ARPANET mail [Crocker82] 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 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 Ethernets). This goal requires consideration of the contents of an address as well as its syntax in order to determine which gateway to use.

(5) Configuration information 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.

---

[1]This section makes no distinction between *delivermail* and *sendmail.*

(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.

3. USAGE

3.1. Address Formats

Arguments may be flags or addresses. Flags set various processing options. Following flag arguments, address arguments may be given. 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

Figure 1 — Sendmail System Structure.

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. This might be used to avoid normal aliasing or duplicate suppression algorithms.

Parentheses, angle brackets, and double quotes must be properly balanced and nested. The rewriting rules control remaining parsing[2].

Although old style addresses are still accepted in most cases, the preferred address format is based on ARPANET-style domain-based addresses [Su82a]. These addresses are based on a hierarchical, logical decomposition of the address space. The addresses are hierarchical in a sense similar to the U.S. postal addresses: the messages may first be routed to the correct state, with no initial consideration of the city or other addressing details. The addresses are logical in that each step in the hierarchy corresponds to a set of “naming authorities” rather than a physical network.

For example, the address:

eric@HostA.BigSite.ARPA

would first look up the domain BigSite in the namespace administrated by ARPA. A query could then be sent to BigSite for interpretation of HostA. Eventually the mail would arrive at HostA, which would then do final delivery to user “eric.”

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).

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.

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 local addresses to address lists using a system-wide file. This file is hashed to speed access. Only addresses 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, if an recipient address specifies a local user *sendmail* searches for a “.forward” file in the recipient’s home directory. If it exists, the message is *not* sent to that user, but rather to the list of addresses in that file. Often this list will contain only one address, and the feature will be used for network mail forwarding.

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

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 [Crocker77] 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 body is an uninterpreted sequence of text lines.

The header is formated 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.

4. THE UUCP PROBLEM

Of particular interest is the UUCP network. The explicit routing used in the UUCP environment causes a number of serious problems. First, giving out an address is impossible without knowing the address of your potential correspondent. This is typically handled by specifying the address relative to some “well-known” host (e.g., ucbvax or decvax). Second, it is often difficult to compute the set of addresses to reply to without some knowledge of the topology of the network. Although it may be easy for a human being to do this under many circumstances, a program does not have equally sophisticated heuristics built in. Third, certain addresses will become painfully and unnecessarily long, as when a message is routed through many hosts in the USENET. And finally, certain “mixed domain” addresses are impossible to parse unambiguously — e.g.,

decvax!ucbvax!lbl-h!user@LBL-CSAM

might have many possible resolutions, depending on whether the message was first routed to decvax or to LBL-CSAM.

To solve this problem, the UUCP syntax would have to be changed to use addresses rather than routes. For example, the address “decvax!ucbvax!eric” might be expressed as “eric@ucbvax.UUCP” (with the hop through decvax implied). This address would itself be a domain-based address; for example, an address might be of the form:

mark@d.cbosg.btl.UUCP

Hosts outside of Bell Telephone Laboratories would then only need to know how to get to a designated BTL relay, and the BTL topology would only be maintained inside Bell.

There are three major problems associated with turning UUCP addresses into something reasonable: defining the namespace, creating and propagating the necessary software, and building and maintaining the database.

4.1. Defining the Namespace

Putting all UUCP hosts into a flat namespace (e.g., "...@host.UUCP") is not practical for a number of reasons. First, with over 1600 sites already, and (with the increasing availability of inexpensive microcomputers and autodialers) several thousand more coming within a few years, the database update problem is simply intractable if the namespace is flat. Second, there are almost certainly name conflicts today. Third, as the number of sites grow the names become ever less mnemonic.

It seems inevitable that there be some sort of naming authority for the set of top level names in the UUCP domain, as unpleasant a possibility as that may seem. It will simply not be possible to have one host resolving all names. It may however be possible to handle this in a fashion similar to that of assigning names of newsgroups in USENET. However, it will be essential to encourage everyone to become subdomains of an existing domain whenever possible — even though this will certainly bruise some egos. For example, if a new host named "blid" were to be added to the UUCP network, it would probably actually be addressed as "d.bli.UUCP" (i.e., as host "d" in the pseudo-domain "bli" rather than as host "blid" in the UUCP domain).

4.2. Creating and Propagating the Software

The software required to implement a consistent namespace is relatively trivial. Two modules are needed, one to handle incoming mail and one to handle outgoing mail.

The incoming module must be prepared to handle either old or new style addresses. New-style addresses can be passed through unchanged. Old style addresses must be turned into new style addresses where possible.

The outgoing module is slightly trickier. It must do a database lookup on the recipient addresses (passed on the command line) to determine what hosts to send the message to. If those hosts do not accept new-style addresses, it must transform all addresses in the header of the message into old style using the database lookup.

Both of these modules are straightforward except for the issue of modifying the header. It seems prudent to choose one format for the message headers. For a number of reasons, Berkeley has elected to use the ARPANET protocols for message formats. However, this protocol is somewhat difficult to parse.

Propagation is somewhat more difficult. There are a large number of hosts connected to UUCP that will want to run completely standard systems (for very good reasons). The strategy is not to convert the entire network — only enough of it it alleviate the problem.

4.3. Building and Maintaining the Database

This is by far the most difficult problem. A prototype for this database already exists, but it is maintained by hand and does not pretend to be complete.

This problem will be reduced considerably if people choose to group their hosts into subdomains. This would require a global update only when a new top level domain joined the network. A message to a host in a subdomain could simply be routed to a known domain gateway for further processing. For example, the address "eric@a.bli.UUCP" might be routed to the "bli" gateway for redistribution; new hosts could be added within BLI without notifying the rest of the world. Of course, other hosts *could* be notified as an efficiency measure.

There may be more than one domain gateway. A domain such as BTL, for instance, might have a dozen gateways to the outside world; a non-BTL site could choose the closest gateway. The only restriction would be that all gateways maintain a consistent view of the domain they represent.

4.4. Logical Structure

Logically, domains are organized into a tree. There need not be a host actually associated with each level in the tree — for example, there will be no host associated with the name "UUCP." Similarly, an organization might group names together for administrative reasons; for example, the name

CAD.research.BigCorp.UUCP

might not actually have a host representing "research."

However, it may frequently be convenient to have a host or hosts that "represent" a domain. For example, if a single host exists that represents Berkeley, then mail from outside Berkeley can forward mail to that host for further resolution without knowing Berkeley's (rather volatile) topology. This is not unlike the operation of the telephone network.

This may also be useful inside certain large domains. For example, at Berkeley it may be presumed that most hosts know about other hosts inside the Berkeley domain. But if they process an address that is unknown, they can pass it "upstairs" for further examination. Thus as new hosts are added only one host (the domain master) *must* be updated immediately; other hosts can be updated as convenient.

Ideally this name resolution process would be performed by a name server (e.g., [Su82b]) to avoid unnecessary copying of the message. However, in a batch network such as UUCP this could result in unnecessary delays.

5. COMPARISON WITH 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.

REFERENCES

[Crocker77] 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.

[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.

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

[Nowitz78] 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.

[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.

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

[Su82a] 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.

[Su82b] Su, Zaw-Sing, *A Distributed System for Internet Name Service.* RFC830. Network Information Center, SRI International, Menlo Park, California. October 1982.