Electronic Mail Addressing in Theory and Practice

with The IDA Sendmail Enhancement Kit

(or The Postmaster's Last Will and Testament)

*Lennart Lövstrand**
<lel@ida.liu.se>

Department of Computer and Information Science
University of Linköping
S-581 83 Linköping
SWEDEN

ABSTRACT

This paper discusses theoretical and practical aspects of handling electronic mail addresses in a heterogeneous environment. It argues for more intelligent Mail Transport Agents that are able to fully format addresses according to different formats and that does not unnecessarily complicate header addresses. Also described is a set of enhancements to the UNIX® *sendmail* program and accompanying rewriting rules used to fulfill our two main goals: (1) To provide a canonical format for handling all electronic mail addresses in which "replying" regularly will work and where local users do not have to depend on the recipient's explicit route or addressing syntax when submitting a message. (2) To design and implement a method for managing mail to and from local users in a machine independent way, allowing them to change their preferred actual mailboxes while maintaining the same visible surface addresses at all times.

May 27, 1987

* New address from July 1987: Xerox EuroPARC, 61 Regent Street, Cambridge CB2 1AB, U.K.

Report no. LiTH-IDA-Ex-8715

Electronic Mail Addressing in Theory and Practice

with The IDA Sendmail Enhancement Kit

(or The Postmaster's Last Will and Testament)

*Lennart Lövstrand**
<lel@ida.liu.se>

Department of Computer and Information Science
University of Linköping
S-581 83 Linköping
SWEDEN

1. INTRODUCTION

> *While some computer-based mail addressing systems are actually easier to deal with than the paper-based model, they are the exception—and not the rule.*
>
> *Why, you might ask, has electronic mail service become so very complex? Most of the problems are simply inherent in reaching beyond a local system to connect with another.*
>
> [Crocker87a]

Sending electronic mail is not always as easy as it ought to be. Too many incompatible mail addressing formats exist, forcing the presumptive user sending a message to know a great deal more than can be thought reasonable about the recipient mail system's idiosyncrasies. This is a widely recognized problem, which can be seen as a consequence of the ever increasing interconnectivity between different computer systems, each subscribing to a different addressing standard. There are gateways that do address transformation on messages passing from one network to another, but it is normally done in a too insufficient manner to get rid of the unintelligible hybrid addresses that often infest us. Even worse are the many systems that assault these mixed format addresses by rewriting them to malformed or incomplete ones. A hybrid address passing several network boundaries is often transformed in such a way that it no longer is possible to use it as a "reply" or error return address; not even for a human being, much less for a machine.

These problems are especially frequent in the UNIX world. Networks like the ARPANET and CSNET have the advantage of being more internally coherent; both follow the Internet mail syntax specifications, described in RFC822 [Crocker82a]. The UNIX world used to practice the '!'-path addressing syntax in which all addresses are relative routes, but has recently been moving over to the domain address standard of the Internet. The present problems concern nodes that has not yet done the transition and those that *cannot* change, because their standard mailer software is unable to handle these new format addresses. A typical example of the latter are the System V systems. Berkeley systems have the freedom of *sendmail*(8), which unfortunately not always turns out as a blessing. In a way, it is too easy to rewrite addresses using *sendmail*, but too hard to control the transformations. This often leads to strange and incompatible formats that don't belong in either standard.

This paper discusses the most common formats and functions electronic mail addresses have. It argues for more intelligent Mail Transport Agents that are able to fully format addresses according to different formats and that does not unnecessarily complicate header addresses. In the end, it moves over to describe the *IDA Sendmail Enhancement Kit* and the work and rationale that lies behind it. The Kit is made up of two parts: First, the configuration file setup and the rewriting rules contained in it. These implement a rewriting strategy based on always *completely* resolving addresses instead of being content by looking at the immediate host. The addresses are then fully transformed again according to the respective mailer's and

* New address from July 1987: Xerox EuroPARC, 61 Regent Street, Cambridge CB2 1AB, U.K.

Report no. LiTH-IDA-Ex-8715

expected ultimate recipient's format. Second, we describe a set of modifications to the *sendmail* source, giving it an extended functionality that in the opinion of this author should have been implemented long ago. Typical additions are: Direct Access to Dbm(3) Files, Separate Envelope/Header Rewritings, and Multi-Token Class Matches. The configuration file is heavily dependent of these modifications and will not function without them.

We have also developed a way of handling mail to or from local users in a machine independent way by hiding their actual sender and recipient addresses behind generic organization oriented addresses. This way, one may have a fixed visible address which is dynamically associated with one or more physical mailboxes. Mails sent from any of a person's "well known" accounts will appear to come from his generic address. Similarly, mail to any of his generic address will be forwarded to his preferred mailbox(es). Note that the generic addresses as a group have no connection to any particular machine. Instead, they are merely database entries on one or more nodes.

2. NAMES, ADDRESSES, AND ROUTES

Larry Kluger and John Shoch has in an excellent article [Kluger86a] described the distinction between *names*, *addresses*, and *routes*, in short:

> *The name of a resource refers to what we seek, an address indicates where the resource is, and a route tells us how to get there.*

When dealing with electronic mail, *names* are typically used in identifying three kinds of entities: (1) The mailbox associated with the sender (originator) and recipient of a message, (2) The name space (domain) in which the sender/recipient is known, and (3) The computer system that houses a Mail Transfer Agent (MTA) able of delivering or forwarding messages. Often, the two latter coincide by associating the domain of a set of mailboxes with the actual machine that implements them. Furthermore, an *address* would be the data structure used in directly connecting to another MTA over a computer network, such as a four-byte Internet number + TCP port number, or an ordinary telephone number. It may well happen that many names map to the same address, or that the same name have more than one address. Lastly, a *route* consists of an ordered sequence of two or more MTA names or addresses, forming an explicit path that the message should take to reach its recipient. Routes can be further divided into *system routes,* where the MTA itself is the responsible of constructing a useful path and *source routes,* where that responsibility lies on the person sending the message.

The mapping from *names* to *addresses* is essentially beyond the scope of this paper, and will only briefly be mentioned in the following sections. Thus, we have taken the liberty of using the general meaning of the word *address* to it denote both mailbox/domain name pairs as well as complete routes. Also, we are using the words *system*, *host*, and *node* to all denote MTAs somewhere in a network. It is our hope that the reader should not be confused because of this.

3. MAIL ADDRESS FORMATS

The absolute majority of today's mailing systems use addresses,[1] represented by a simple string of characters. Some of these characters implement operators that are used to divide the address into mailbox/domain/route parts when parsed by an MTA. Different operators have different directions of associativity, making it increasingly difficult to unambiguously parse addresses produced by combining incompatible operators of different mail address syntaxes. It is hoped that at least some of these problems will be solved with the emergence of the structured attribute list addresses of X.400. In the mean time, we have a variety of different formats in use, each subscribing to a different set of delimiting operators. It is not uncommon to see addresses like:

mcvax!enea!liuida!obelix!p_e%seismo.css.gov@relay.cs.net

or even

enea!seismo.CSS.GOV!!OZ.AI.MIT.EDU,!MC.LCS.MIT.EDU:ebg!REAGAN.AI.MIT.EDU

turn up in message envelopes and headers. The last example comes from the envelope sender address found on a message in which the RFC822 route was incompletely translated into '!'-path syntax. Now,

[1] That is, routes or mailbox/domain name pairs.

before delving into a discussion about how these may be resolved or preferably avoided, let's take a look at what kind of addressing formats currently exist.

3.1. Relative Addresses

These types of addresses are by necessity all implemented as *routes*. In purely relative addresses, all node names are relative to each other, making path optimization or system routing difficult, if not impossible. For the sender of a message, this means that addresses will look different depending on his location in the network, forcing him to recompute all addresses each time he changes his location. Even worse, in a rapidly growing network, it might even happen that an address becomes invalid overnight because some link far away has been disconnected or replaced by another. All this makes it difficult for a presumptive user to continuously keep his addresses correct and up to date.

Relative addresses have since long been in use within the UNIX community, but a great deal of work has been done by an organization called *The UUCP Mapping Project* in eliminating duplicate host names, thus making it possible to use absolute addresses[2] in a flat name space. It is presently moving towards utilizing full domain names but is delayed by the fact that some systems, notably *System V* systems, cannot handle anything but UUCP source routes with standard mailer software. The addressing syntax for UNIX UUCP '!'-paths is as follows:

node!...!node!user

The route sequence is read from the left to the right, with the ultimate recipient on the rightmost end. Other systems that have similar addressing formats are the Berknet and VAX/VMS mail systems, which use:

node:...:node:user

and

node::...::node::user

respectively. RFC822 also specifies a way of constructing explicit paths using the somewhat complicated syntax:

<@node,@node,...:user@node>

Here, the message should be passed through each successive node from left to right, ending up in the last user@node's mailbox. Note that the less than and greater than brackets are included in the syntax. Another widely used but undocumented format is *Ye Olde ARPANET '%'-Kludge:*

user%node%...%node@node

which is interpreted from the right to the left by delivering the message to the node after the atsign and then instantiating the rightmost percent sign into a new atsign, etc.

3.2. Absolute Addresses

The Tao that can be told of is not the Absolute Tao;
The Names that can be given are not Absolute Names.

The Nameless is the origin of Heaven and Earth;
The Named is it the Mother of all Things.

[Laotse00a]

Absolute addresses have the advantage of being universally unique and thus applicable by any MTA[3] independently of where it is located. Since the names should be uniquely identified, some way of distributing them within their name space needs to be accomplished. The simplest way of doing this is by registering plain node names with some central name directory on a first-come-you-get-it service. The *UUCP Project* tried this to avoid duplicate UUCP node names. However, maintaining such a directory and propagating its changes easily becomes too heavy a burden to handle. Another strategy was first adopted by the ARPA Internet community, the hierarchical domain naming system described by RFC882 [Mockapetris83a], RFC920 [Postel84a] and others.

[2] See the following section.

[3] At least in theory—not all MTAs necessarily know about how to deliver to all addresses.

In this system, a labelled tree is built with each node in the tree denoting a specific domain. Some nodes correspond to actual hosts, typically the leaves in the tree, while others simply map to some organizational entity, like a group, department, or institution. The purpose of the domain naming system is to distribute the naming authority throughout the tree. Letting each domain have the responsibility of naming the domains immediately beneath it guarantees the uniqueness of all simple domain names relative to their parents. The full, qualified domain names are constructed by concatenating each level's simple domain name with a dot in between. For example, there might exist a certain mail computer named "MC" within the Laboratory of Computer Science of the Massachusetts Institute of Technology, an Educational organization. A possible domain name for this computer would be:

MC.LCS.MIT.EDU

There might be many hosts named "MC," but only one within the "LCS.MIT.EDU" domain. The same goes for the "LCS" domain within the "MIT.EDU" domain. The global uniqueness of each fully qualified domain is thus guaranteed by its parentage.

The domain system is currently in use within the ARPA Internet, CSNET, and is in progress within the UUCP world. Under its anonymous root domain, it presently has six three-letter organizational domains registered and a continuously increasing number of national two-letter domains. The organizational domains are mainly used within the U.S., and the national domains in Europe and Asia. There are also a set of *de facto* network based domains in use, although not officially registered. These are really mock domains used to incorporate hosts on physical networks that cannot or do not want to handle domain addresses. Examples of these are BITNET and still most of the UUCP world. Appendix D lists all domains currently registered with the SRI Network Information Center together with a set of otherwise frequently recognized network based domains.

3.3. Attribute Addresses

With the CCITT[4] X.400 [Malaga-Torremolinos84a] series standard for electronic mail in emergence, a new kind of addressing system is being proposed. In this format, recipients are uniquely identified using a list of attribute-value pairs. Some of these, like the Organization and Country attributes, are obligatory while others may be supplied only if known by the sender. The idea is that the base attributes should be able to guide the message to a relevant directory server, while the others then are used to select the actual recipient. Attribute sets that select no or more than one recipient will probably be considered erroneous, but could be used in selecting multiple recipients.

It will yet take several years before the attribute addressing scheme has come to widespread use. It will, however, surely come—if nothing else, then because it has the force of the united PTTs behind it. Already, there exists guidelines for mapping between RFC822 based addresses and X.400, such as RFC987 [Kille86a].

3.4. Hybrid Addresses

With all this in mind, let's take a look at how different formats sometimes are combined and how we can resolve them. The three major addressing formats for routing messages are:

[1]	The UUCP '!'-path	<*node$_1$!node$_2$!node$_3$!user*>
[2]	Ye Olde ARPANET '%'-Kludge	<*user%node$_3$%node$_2$@node$_1$*>
[3]	The RFC822 route syntax	<*@node$_1$,@node$_2$:user@node$_3$*>

where the latter mostly is used for envelope senders.

Combinations of the above usually appear in messages crossing one or more network boundaries with different addressing formats. Since each of these formats were independently developed, it may not be obvious how they should be interpreted when combined. Still, by reasoning a little, much can be inferred from how they incrementally are constructed.

[4] *Comité Consultatif International Téléphonique et Télégraphique,* i.e. the International Telegraph and Telephone Consultive Committee

Starting with the Domainist's approach to the matter, we have to give '@' precedence over '!' since this is implied by RFC822. This means that addresses like:

$node_2$!$node_1$!user@domain

will be interpreted as:

domain → $node_2$ → $node_1$ → user

Now, since '%' is often the *de facto* standard routing operator on top of '@', an address like:

host!user@domain

that is autorouted through *relay* will probably end up looking as:

host!user%domain@relay

meaning:

relay → domain → host → user

This forces us to give '%' priority over '!'. However, a '!'-path address ending with a "user%node," cannot be a domain address (no '@') and should therefore be interpreted using UUCP semantics by prioritizing '!' over '%'. Thus,

$node_1$!$node_2$!user%domain

should be read as:

$node_1$ → $node_2$ → domain → user

Mixtures with RFC822 routes may look hard to read, but are actually easy to parse. A fairly complicated address like:

$node_1$!$node_2$!@$domain_1$,@$domain_2$:host!user%relay@$domain_3$

has to be interpreted as:

$node_1$ → $node_2$ → $domain_1$ → $domain_2$ → $domain_3$ → relay → host → user

since RFC822 like '!'-paths associate left-to-right, and since the last "localpart@domain" can be unambiguously found after the colon.

Now, not all of us are Domainists. Many nodes can and will only be able to interpret UUCP '!'-paths, which leads to complications with mixed '!'- and '@'-style addresses. The only workable solution to this is to try and avoid such mixtures altogether. The easiest way of doing this is to write them as '!'- and '%'-style combinations, but even better would be to wrap them wholly around to the '!'-path format. They should then turned back into '%' and '@' combinations when breaking the Domain Land boundary.

4. A SHORT ANATOMY OF THE ELECTRONIC MESSAGE

In analogy to the written letter, there are two major parts of a message: The envelope and the contents. The envelope is there specifically for the MTAs to handle and contains the sender address together with the message's actual recipients. The contents are usually further subdivided into the header lines and the actual body, where only the latter is under the sender's full control. The headers are used by the MTAs and MUAs[5] to store various information of interest to the recipient, such as sender, all official recipients, posting date, etc. Although the body usually is left uninterpreted, some mail systems put constraints by limiting the length of each line or the whole message, or by only allowing printable ASCII characters.

4.1. The Envelope

The envelope contains the physical message's actual recipients, which very well may be different from those in the headers. Typically, a message sent to more than one recipient will be split into *n* copies, one for each network. These messages will have the original's all recipients listed in their header lines, but each copy's envelope should only have those being delivered over the network in question. There is usually also the option of *Blank Carbon Copy* recipients, which per definition never shall show up in the headers.

[5] Mail User Agent, the program that the user directly interacts with when reading or composing messages.

The envelope will also contain the explicit path back to the sender for error messages and tracing purposes. This path should formed by having each node that forwards the message incrementally add its name to the route, thus avoiding routing problems that otherwise may appear. The result of each rewriting should be a full route in a suitable format leading from the current node back to the originator.

If the envelope recipient(s) are routes, they are handled in an analogous manner to the senders by removing the local node's name from each address before propagating it further. Optionally, the address can be made fully relative to the immediate receiving node by removing its name from the route as well. This should be determined on a mailer dependent basis. The MTA has the full freedom of at any point turning a simple envelope recipient address into a route if it sees reason to do so. This could be done on the grounds that the immediate recipient node cannot perform automatic routing. It should, however, be avoided if possible since it is hard to keep routing tables fully updated with topological changes in distant parts of the network. Turning envelope routes into simple addresses should also be avoided since there usually exists a good reason for a route to be there.

4.2. The Headers

Header addresses are not normally used by the MTA. Exceptions may be when headers such as "Return-Receipt-To:" exists and the MTA is doing the final delivery or when the delivery of a message fails and there exists a "Errors-To:" header.[6] The MTA is also allowed to rewrite, or "munge," header addresses when a message is forwarded from one network to another. This is done by first removing the addressing idiosyncrasies of the transmitting network to obtain some internal canonical format and then applying the receiving network's idiosyncrasies to produce a conforming address [Rose83a]. Of course, this should be done to both envelope and header addresses.

Even within one world, like the UUCP pseudo-network, it may be necessary to "munge" addresses for them to be understandable by the recipient system. For instance, many mail systems does not recognize all domains or perhaps cannot even handle anything but pure and fully routed UUCP '!'-paths. If the transmitting MTA does not take this into consideration, the user sending the message has to submit full source routes with each receiving network's addressing syntax embedded. Except in the most simple cases, this task requires great knowledge[7] about how networks are interconnected, much more than can be considered reasonable by any casual or even experienced user.

In our opinion, this is currently the greatest obstacle in making electronic mail usable. On from bad to worse, these user supplied source routes that are fully contained in the headers often get rewritten into further complicated routes. When such a message is received by its recipient, its header addresses may very well be too unintelligible to be understandable by a human being, much less by a machine. In the best case, they will just have routes with incorrect points of reference, forcing "reply" messages to the other recipients to first be (automatically) routed to the first node of the path before it can start on the actual route. Then often in the opposite direction, leading half way back again.

5. ADDRESS REWRITING STRATEGIES

Now, given the freedom and flexibility of *sendmail*, our project's task has been to construct a configuration file that, with the necessary enhancements to the *sendmail* source, will completely resolve and canonicalize all envelope and header addresses to an internal format. All unqualified addresses are then officialized using the TCP/IP name server function and a local *dbm*(3) based domain name table, and a route is found using a direct interface to a *pathalias*(1) routing file. Finally, using a static *dbm*(3) mailer table together again with the TCP/IP name server function, the message is dispatched to the appripriate mailer which fully rewrites the addresses according to its own idiosyncrasies.

5.1. Sneak-In Preview

To give a taste of how the complete system performs with a realistic case, consider at the following only partly imaginary example:

[6] These are *sendmail* specific; other MTAs may have other exceptions.

[7] That is, a case for a *guru*!

Envelope:
Sender: enea!seismo!relay.cs.net!cate%busch%pany.com
Recipient: obelix!p_e
Headers:
From: enea!relay.cs.net!cate%busch%pany.com
To: mcvax!enea!liuida!obelix!p_e%seismo.css.gov@relay.cs.net
cc: ree.pete%fidelio.uu.se%seismo.css.gov@relay.cs.net

A user *cate* on the Company Inc's local host *busch* has sent a message to two Swedish recipients: *p_e* on the UUCP host *obelix* in Linköping and to *ree.pete* on the Uppsala node *fidelio.uu.se.* If the headers would be left untouched, a reply from *p_e* to both *cate* and *ree.pete* would force *ree.pete*'s copy to go all the way back to *relay.cs.net* before it could return to Sweden and Uppsala. Clearly, this is a waste of both resources and time when there might (and does) exist a much shorter path within the country. With The Kit's rewriting heuristics, the same header lines will look like the following when leaving the local node:

Envelope:
Sender: @majestix.liu.se,@enea.se,@seismo:cate%busch%pany.com@relay.cs.net
Recipient: p_e%obelix.liu.se@asterix.liu.se
Headers:
From: cate%busch@pany.com
To: p_e@obelix.UUCP
cc: ree.pete@fidelio.uu.se

Here, our local node's name has been added to the envelope sender path, which also has been transformed into a RFC822 route[8]. Other options would be to have it as a '!'-path or '%'-path. The envelope recipient has been routed via *asterix.liu.se,* and changed into a '%'-path, on the basis that the message is forwarded over a TCP/IP connection and this is the preferred route format for most such systems.

Also, the route has been removed from the header "From:" line, leaving the first universally qualified node there together with a '%'-path from that point to the recipient. The "To:" line has undergone even more drastic changes. First, the route to *seismo.css.gov* was removed since this is the first universally qualified node. Then a table of well-known UUCP relays was consulted to further compress the path. *Mcvax*, *enea*, and *liuida* were all members of that list. This gave "obelix!p_e" as a result, which then was turned into the domain form "p_e@obelix.UUCP." In the last line, "ree.pete@fidelio.uu.se" simply had its path removed since *SE* is a registered top domain.

5.2. The Configuration File

The IDA Sendmail Master Configuration File should be sent through the *m4*(1) macro processor to produce an actual configuration file. Several *m4* identifiers are used to customize the file; each of them is described in *Appendix C: Customization Parameters*. Unlike the Berkeley version, it was not designed as a set of *m4* fragments that "sources" each other to form a full configuration, but rather as a single master configuration file which holds a *bank* of all possible mailers and corresponding rewriting rulesets. The instance's actually available mailers are enabled by giving values to their corresponding *m4* identifiers. The current version include mailer definitions for a TCP/IP mailer, three kinds of UUCP mailers depending on the remote node's address handling capabilities, a mock DECnet mailer, as well as the LOCAL and PROG mailers. Their design has been kept as clean as possible to make the construction of e.g. BITNET or CSNET mailers using these as templates straight-forward.

The rewriting rules of the Kit's configuration file are explicitly oriented towards the domain naming syntax. They will resolve all input addresses to an internal domain based format and then rewrite them according to the selected mailer's preferences. Internally, all addresses have the same

user@.domain

format. Note the dot after the atsign; it is there to make it easier to rewrite the address. Also note that this differs substantially from the Berkeley "whatever<@host>whatever" format. For historical reasons, both the RFC822 route syntax and *Ye Olde ARPANET '%'-Kludge* are used internally to represent routes when only one of them should be sufficient.

[8] Save for the '<' and '>' brackets.

5.3. Canonicalizing the Address

Ruleset 3 canonicalizes all addresses, making them conform to our internal format. After the canonicalization, the "user" part may end up containing a route in either standard RFC822 format or using the '%'-path format. '!'-, ':'-, and '::'-style paths are rewritten into RFC822 routes. Reasonable mixtures of route formats are resolved using the strategies described in the section about *Hybrid Addresses*. As an option, the (untested) UUCPPRECEDENCE switch may be turned on in the configuration master file. This will enable some simple heuristics that will decide between domain style and UUCP '!'-path prioritized unpacking depending on whether the *domain* is qualified or not. In any case, ruleset 3 will make sure that the *domain* part of all "user@.domain" addresses are mapped to their full, official domain names whenever possible using both the TCP/IP name server and a dbm domaintable. It also goes through some effort to repair malformed addresses, but much of this is probably too site specific to be generally useful.

Since '!'-paths are internally represented as RFC822 routes, you should not be surprised when you see an address like:

foo!bar!baz!user

first be transformed into:

@foo.UUCP,@bar:user@baz

and then to:

bar:user@baz@.foo.UUCP

The UUCP domain of *foo* has been inferred from the '!'-style syntax. If *foo* had been known by the domaintable to have specific domain name, that had been used instead. Nothing can be inferred about the nodes *bar* and *baz*, since we they may be local to *foo*. Now, since the pure RFC822 route doesn't conform to our internal format, i.e. it does not have a "user" part followed by an atsign-dot and a "domain," we had to rearrange it a little. The closest node of the route was thus extracted and added the right side of the rest of the route together with the atsign-dot. It may not be very pretty to look at, but it is easier to handle this way.

Note that there is a risk of confusing UUCP node names with local hosts using the domaintable lookup. For example, if you had a local node *linus* with a full domain name of *linus.liu.se* and received an address like "linus!user," this would be interpreted as the local *linus* and rewritten into "user@linus.liu.se." This is probably right for envelope recipients, but not so surely in header lines. You can define BANGIMPLIESUUCP if you want to disable the domaintable qualification.

5.4. Finding Route and Mailer

"Would you tell me, please, which way I ought to go from here?"
"That depends a good deal on where you want to get to," said the Cat.

[Carrol96a]

Before ruleset 0 tries to find an applicable mailer, it digests all routes through the local host by stripping off its own name and sending the address through ruleset 3 again. It then has four strategies of finding a suitable mailer for the address:

[1] Try to find a mailer that will connect to the immediate host in the address.

[2] Try to find a route to the address' domain using a *dbm*(3) routing table and a mailer that will connect to the route's closest node.

[3] Use the firm-wired RELAY_MAILER and RELAY_HOST pairs to automatically forward the message.

[4] Give up; send the address to the ERROR mailer.

The code that determines if a mailer directly can deliver to a certain domain is found in ruleset 26.[9] It does this on a per mailer bases with the following order of priority:

LOCAL If the supplied domain is any of local host's names (member of the **$w** class), or if the complete address is found in the *aliases*(5) file, the message is delivered locally. The latter type of

[9] Yes, I too wish that named rulesets would be available in *sendmail*. Perhaps somebody should convert this configuration file into *ease*.

local delivery will cause the address to be expanded to the RHS of the alias entry and the complete process to recurse.

\k:*Special*\h'|\n:u'\v'+1'*Mailers*\v'-1'

In order to override the standard mailer selection, a special dbm *mailertable* may be used to force addresses to be delivered using specific mailers. If the address' domain is found in the *mailertable*, the associated mailer will be used. The mailer table should map official domain names to "mailer:host" pairs, with a colon between the mailer and the host.

TCP/IP With the new *default* argument of the TCP/IP nameserver lookup function, it is possible to determine if an address can be delivered using this protocol family without relying on static host tables. If the address' domain is known to the TCP/IP nameserver, it is returned together with its canonicalized host name.

DECnet The DECnet mailer does not share the network based nameserver facilities of the TCP/IP mailer, and thus has to rely on a host table. This is done with a two-phase operation—first the domain is mapped to a DECnet name, if known, then the the DECnet host name is checked in the list of connectable DECnet hosts before it is returned. This is because some DECnet nodes cannot talk across area boundaries, forcing recipient addresses to be explicitly routed over an intermediary host. *Note:* The supplied DECnet mailer uses a TCP/IP connection to a DECsystem-20 acting as gateway. A real implementation should remove the immediate node from routes before returning them, but we cannot do this.

UUCP The UUCP mailer is also determined with a two-phase operation—first the domains is mapped through the UUCP translation table, returning the UUCP node name, if known. The UUCP mailer will then be selected only if the UUCP name is known to be directly connectable by us (normally determined using the /usr/lib/uucp/L.sys file). All nodes found this way will be sent to through the "dumb" UUCP mailer. Delivery using either the UUCP-A or the UUCP-B mailer has to be determined using the special mailertable previously mentioned.

If an address needs to be routed, i.e. if the first pass through ruleset 26 fails, it is given to ruleset 22 where its domain is looked up in a *pathalias*(1) type routing table. Routes to explicit domain/host names are preferred over general (parent) domain routes. Before the new address is returned, it is sent through the canonicalization routines of ruleset 3. This makes specific *pathalias* route syntax effectively ineffective. The normal way would be not to specify any special routing syntax at all to *pathalias*, but to invariably let it produce '!'-paths.

5.5. Externalizing the Address

After a mailer has been chosen, addresses are rewritten using rulesets 1 and 2 for envelope senders/recipients and rulesets 5 and 6 for header senders/recipients. Envelope senders are left untouched by this process, but envelope recipients will have RFC822 routes turned into '%'-paths. Header RFC822 routes will also be turned into '%'-paths and then gently compressed by having paths to fully qualified domains and UUCP relay-to-relay paths removed. Header senders will furthermore have their host names hidden by HIDDENNAME, if defined, and their addresses filtered through the GENERICFROM table, if available.

When this is done, the mailer specific rewriting phase starts. The LOCAL and PROG mailers does not do any further rewriting as supplied, but could be convinced to produce '!'-paths for UUCP routes if preferred [using ruleset 15 or a variant thereof].

The TCP/IP and DECnet mailers will add a call to ruleset 24 for all envelope recipients. This will turn domains corresponding to DECnet nodes into flatspaced DECnet host names, since domains are not supported there. This should really not be done in the TCP/IP mailer, but all our DECnet traffic is presently routed over a TCP/IP link. Since no special rewriting is done for envelope senders, this means that they normally will appear in RFC822 route format using these as well as any of the previous mailers.

There are three variants of the UUCP mailer depending on the remote node's address handling capabilities. The "dumb" version, simply called UUCP, corresponds closely to the class 1 mailer of RFC976 [Horton86a]. It will rewrite all addresses into '!'-format, and makes all header addresses '!'-relative the

recipient node, routed through the transmitting node if necessary.[10] The UUCP-A is closer to the RFC976 classes 2 and 3 mailers in that it will let all header addresses stay in '@'-format, but change envelope addresses to '!'-paths whenever applicable. The UUCP-B mailer, finally, functions as the UUCP-A mailer but will in addition supply envelope senders in RFC822 route format and transmit the message to a *bsmtp* program on the remote node.

Ruleset 4 will as usual make the address truly external. In our case, this means by removing the dot after the atsign and by moving the immediate domain to the head of RFC822 routes.

6. MANAGING GENERIC MAIL ADDRESSES

While sending mail to people by specifying a mailbox on a named computer may fulfill the technical requirements of an electronic mail system, it may not always be very convenient. With people having accounts on many different hosts and often moving between them, it is hard to keep track of where a person presently will read his or her mail. For external senders, remembering strange host names for lots of people is even worse. Just knowing that Person A, working in the same group as Person B, has a mailbox on Computer C does not mean that B also can be reached on that machine. It may very well be that Computer C is A's personal workstation with no other accounts than A's.

Simply setting up as many forwarding addresses as possible on all sorts of machines does not solve the problem. First of all, there might be reasons for a user to want to actually receive mail on any of these machines—e.g. because this may be the easiest way of copying files between remote machines. Also, there is still no visible connection between the person and his group, which makes it hard to remember his address.

Another solution would be to have a special machine for mail forwarding, preferably with a name corresponding to the local organization. Since all mail has to go through this node in order to reach its recipients, it may prove to be an expensive solution if this means that the extra load will restrain users from doing other useful work on the machine. Also, if this node goes down, no mail will be delivered until it comes up again.

Our proposed solution to the problem is to extend *sendmail*'s notion of aliases to include forwarding of non-local addresses as well as those specific to the local host. This way, several hosts may share the non-local part of a aliases table and any of them can do the forwarding. Users may change their preferred mailbox and computers may be renamed at any time with no change in the way their inbound addresses look like. As an option, users' "From:" line addresses may be rewritten to correspond to the organizational address by using the same database. For example, this author's mail address:

lel@ida.liu.se (*aka* lel@liuida.UUCP)

is implemented by an entry in the aliases files on set of Sun systems and a Gould. All of them will forward messages for that address to my preferred mailbox on a DECsystem-20. Any message sent from either the Suns, the Gould, or the DECsystem-20 will have "lel@ida.liu.se" as the "From:" line return address. The envelope sender, however, will at all times point to the actual user and host that originated the message.

This is implemented using an *aliases* file with extended syntax, called *xaliases*. Entries in this file are marked for either inbound or outbound aliasing. Inbound is the normal function, and may be used for non-local addresses as well as local ones. Outbound aliasing is done on header sender lines by rewriting them to the LHS value in the *xaliases* entry. The syntax for entries in this file is:

alias, alias, ...: prefix address, prefix address, ...

where the first *alias* is the generic address that should be substituted for those in the header sender lines matching the RHS outbound *addresses*. All LHS *aliases* are mailboxes that should be forwarded to the RHS *addresses* marked inbound. The *prefix* before each RHS *address* marks it as either inbound, outbound or both according to the following table:

(*no mark*)	Inbound
<	Outbound

[10] See the new M_RELATIVIZE mailer flag in the following section.

>	Inbound
<>	Both Inbound and Outbound

The *xaliases* file is then parsed by the *xalparse*(8) program, producing a normal *aliases*(5) file as output together with a *dbm*(1) input file, consisting of the outbound aliases. An example would be:

Fooey.Barbaz@dept, bar@dept, foo: <> fooey@besthost, < fobar@otherhost

which means that mails to either "Fooey.Barbaz@dept," "bar@dept," or "foo@localhost" will be forwarded to "fooey@besthost" and that mail from either "fooey@besthost" or "fobar@otherhost" will have their header sender lines substituted for "Fooey.Barbaz@dept." See the supplied *xaliases* file for more examples.

An *aliases* file with non-local aliases should be processed by running *sendmail* with a configuration file having the NEWALIASES identifier defined. This makes it parse all addresses as local and return them to the LOCAL mailer when building the dbm alias tables. A delivering *sendmail* will then lookup all addresses in the *aliases* file and return any found to the LOCAL mailer [ruleset 26]. The normal aliasing mechanisms of *sendmail* are then used to distribute the messages further.

The substitution of "physical" senders with "generic" addresses, is as previously mentinioned done in the header/sender specific ruleset 5 by lookup in the GENERICFROM database.

7. SENDMAIL SOURCE MODIFICATIONS

Sendmail gives a lot of flexibility to the maintainer and developer of electronic mail. Still, it lacks certain functionality for which it was determined that our project's goals could not be fulfilled without. Just [sic] developing a new configuration file was not adequate, but changes to the source code itself had to be done. Still, we did this with the philosophy that it is best to avoid source code modifications whenever it is possible to obtain the same results by merely changing parameters in the configuration file.

The following features have been implemented in our current version of *sendmail*(5.51++):

7.1. Nameserver Default Argument

Previously, there was no way of knowing whether a nameserver lookup was successful or not, thus making that feature of limited value. This version will allow you to add a *default* argument to the nameserver lookup function, which will be returned if the match fails. A typical usage for this is to determine if a host is accessible using the TCP/IP protocol family. The extended syntax is as follows:

$[** hostname **$: default **$]**

where the **$:***default* part is optional.

7.2. Direct Access to Dbm(3) Files

The configuration file syntax has been expanded to include the declaration and usage of general *ndbm*(3) databases. The option 'K' (for *Keyed database*) has been added. It takes two arguments, a one character internal name for the database and the corresponding *dbm*(3) file(s), as in:

OKP/usr/lib/mail/pathtable

which defines the internal database 'P' to be associated with the dbm files /usr/lib/mail/pathtable.dir and pathtable.pag (or pathtable.map and pathtable.dat if you are using Maryland's *mdbm*(3) package).

The 'P' database may now be used to lookup arbitrary strings in the RHS of rewriting rules. The syntax is as follows:

$(**x key **$@ arg **$:** default **$)**

where *x* is corresponds to a previously declared database, *key* is the string that should be searched for in the database. The *arg* and *default* arguments are optional. The *default* string is returned is if the *key* could not be found in the database. If neither *default* string, nor a matching *key* is to be found, the whole expression expands to the value of *key*. However, if a result is found, it is used as the format string of a *sprintf*(3) expression, with the *arg* as extra argument. Thus, database values with "%s" strings embedded in them can be useful when rewriting expressions. This could typically be used in cooperation with the *pathalias*(1) program to expand routes without leaving *sendmail*.

The *aliases*(5) file is automatically available using the '@' database and should **not** be declared with a option 'K' statement.

7.3. Batched SMTP Support

Sendmail already speaks SMTP over interactive channels, but because it both will drop errors occurring when acting as server as well as hang indefinitely when talking to a non-responding channel as client, this is not sufficient to process or produce SMTP batches. Still, since the SMTP code already is there, it was considered easier to add batching support internally in *sendmail* than to write new front-end programs.

The new code defines a new MD_BSMTP mode, which is activated by the **–bb** option or by making a link to *sendmail* named *bsmtp*. The normal way of digesting SMTP batches is to execute *bsmtp* with no arguments.

To produce SMTP scripts, the M_BSMTP (**B**) mailer flag has been added. It is used as in the following example:

```
MUUCP-B, P=/usr/bin/uux, F=BDFMSXhmpu, S=0, R=15/0, A=uux – –z –r $h!bsmtp
```

This defines the UUCP-B mailer to send SMTP scripts on the standard input to the *uux*(1) program.

7.4. Separate Envelope/Header Rewriting Rulesets

Envelope and header addresses does not always look the same. For example, it is often desirable to have envelope return addresses formed using RFC822 route syntax, while this format more rarely is understood by users' front-end mail programs. Another case is when the envelope recipient address is expanded to a system route, while the header recipient should be kept simple.

With this package, the mail system administrator has the option of separating rewriting control for envelope/recipient addresses. Normally, all sender/recipient addresses are passed through rulesets 1 and 2, but if the *SplitRewriting* option '/' is set, only envelope addresses are handled that way; header addresses are given to rulesets 5 and 6, which should be properly defined.

Mailer dependant rewriting may also be controlled in an envelope/header specific way. This is accomplished by extending the syntax for the 'R' and 'S' attributes of the mailer definition statement:

```
R=re/rh, S=se/sh
```

with the envelope and header rulesets divided by a slash. If no slash is found, it functions as before by using the same ruleset for both types of rewriting. A zero or missing ruleset indicates that no rewriting should be done. In the previous example of the *Batched SMTP Support*, no mailer dependant sender rewriting is done, recipient envelope addresses are rewritten using ruleset 15, but recipient header addresses are left untouched.

7.5. Separate Local UUCP Host Name

With the extensive, structured world of domains on one side and the flatspaced, shortnamed UUCP world on the other, it may be desirable for a node to have a UUCP node name separate from its normal host name. For this purpose, the **$k** macro has been introduced to hold the local node's specific UUCP host name. It defaults to the node's "normal" host name (as returned by *gethostname*(3)), if not explicitly defined in the configuration file. It is used when rewriting headers in the *UUCP Relativization Routines* as well as when producing UUCP "From_" lines with the M_FROMPATH (**p**) turned on.

7.6. Return Path for UUCP Mailers

The M_FROMPATH (**p**) mailer flag is used by the SMTP routines to add the local host to the envelope sender in the MAIL FROM: command. This is a useful option for producing trustworthy routes back to the sender for receipts and error messages. This capability has now been added the the code that produces UUCP "From_" lines. The sender's address is simply prefixed using the local host's UUCP name and an exclamation mark—the canonical way of constructing paths in the UUCP world. (For mailers with the M_UGLYUCP flag (**U**) set, the local host's name is added after the "remote from" string.)

7.7. UUCP Header Address Relativization

A new M_RELATIVIZE (**V**) mailer flag has been added, which relativizes header lines with respect to the immediate recipient host. This means that *paths* through the remote host will have the remote node's name removed (local recipients at the remote host are untouched) and that other addresses are rewritten to have paths through the local host.

Specifically, "**$h**!...!user" addresses are stripped down to the "...!user" part, "**$h**!user" are left untouched, and "others" rewritten into "**$k**!others" (the initial value of **$w** is used if **$k** is undefined).

7.8. Support for Multi-Token Class Matches

When *sendmail* tried to match a LHS **$=**X class expression, it used to be the case that it only looked for matches with one token. If the period is a delimiter and if "foo.EDU" is a member of the 'X' class, it would not find the string "foo.EDU" in the class since it contained three tokens ("foo" "." and "EDU"). This was considered such a great inconvenience that the expression matching code was rewritten to allow multi-token class matches on the expense of being somewhat slower. With the above example, the current version will first try to find a match for "foo" then "foo." then "foo.EDU" and so on, each time incorporating the next successive token of the expression in the class match.

7.9. Support for Embedded Subruleset Calls

Being a LISP hacker of heart, the author couldn't refrain from making *sendmail* handle embedded ruleset calls. The previous version had the very annoying restriction of only allowing one ruleset call per rule and only allowing the **$@** and **$:** macros to appear on its left side. The current version handles both embedded ruleset calls of the form ">4>10$>6 $1@.$2" as well as arbitrarily positioned calls within the RHS expression (although the latter has not been fully tested).

7.10. Elaborate Matching Algorithm for Unknown Local Recipients

Based on the idea that it always is a bad idea to throw mail back with a "User Unknown" error message if a human operator might be able to guess the actual recipients identity, we have added a more elaborate search algorithm that matches unknown recipients with the personal name field of the /etc/passwd file. The previous version of *sendmail* offered a similar functionality, but would only find strings that were exact copies of that field. This version uses an algorithm that will return a number corresponding to the degree of similarity between the two strings. Strings are considered match best if as many mutual substrings as possible are found in sequence. Substrings are delimited by any non-alphabetic character and completely equal substrings are better matches than just prefixes.

The best match of such a search through the /etc/passwd file is returned as the local recipient, provided that only one such match exists. If no best match is found (i.e., if more than one *passwd* entry have the same highest matching degree), the search fails and the mail is returned with an error message.

7.11. Support for Maryland's Mdbm Package

Sites that do not yet have the new *ndbm*(3) functions of BSD 4.3 may still use the *Database Access Functions* described above using the University of Maryland's public domain multi-dbm routines, which are available from your nearest comp.sources.unix archive. Note that these use different extensions of their database files (.map and .dat instead of .dir and .pag) as well as a different internal format than *dbm*(3), so it is still recommended to use *ndbm*(3) whenever possible.

7.12. Improved Test Mode Output

It is hard to remember what all these "^X"s and "^V"s stand for when debugging sendmail rewriting rules, using its address rewriting test mode. The changes in this version will make macros print in their symbolic form, i.e. "**$:**" for "^X," etc.

In addition, at least this *sendmail.cf* developer often wanted to send addresses directly into a specific ruleset without having them automatically rewritten by ruleset 3. Thus, the initial call to ruleset 3 has been **deleted** in this version. You will have to send your addresses manually through ruleset 3 henceforth if you adopt this change.

Finally, the ruleset rewriting output often became too wide to be easily read due to the excess of quotation marks around the tokens. These have now been removed from the test output, leaving just a space between each token. The author thinks this is much more convenient and hopes that you agree.

7.13. Better To: and Cc: Headers of Returned Messages

The original code would produce multiple To: header lines in returned messages if the message was sent to more than one recipient. A typical reason for this to happen was if you used the PostMasterCopy option in your *sendmail.cf* file. This version will put the PostMasterCopy on a separate Cc: line and all others together in a comma-separated list on the To: line.

7.14. Queue Bug Fixed

A nasty little queue bug bit us hard several times last year. It occurred when, for whatever reason, the queue daemon ended up processing a locked queue entry as its last. It immediately discovered that it was locked and left it alone, but then exited and while doing general cleanup in *finis*, removed the locked queue file as well. That way, we frequently started ending up with broken queue files and lost messages as a consequence. The fix is to explicitly set the current envelope's id to NULL before leaving the current queue pass, since it should not exist anyway.

7.15. Shared Input SMTP Bug Tentatively Fixed

The SMTP server routines read delivery requests from stdin. Each message is handled by a different process by forking after the MAIL FROM: command is read. The parent then sleeps while the child delivers the message and continues to process more SMTP commands thereafter. Now, both parent and child read from stdin and share the same file descriptor. However, they do not share the same *_iobuf* that represents the stream, so when the parent starts reading after the child has died, it reads the very same input that the child has processed.[11]

There are only two ways out of this. Either, the parent process don't fork and delivers all messages itself, or they all read unbuffered. The former solution (or a variant thereof) is probably better, but the latter was easier to implement, so that is what presently have been done. Since this causes unnecessary overhead as each character has to be read with a system call, someone ought to make a better fix for this eventually.

7.16. Optional BSD 2.9 and 4.2 Compatibility Code

The Kit includes a set of changes that optionally may be added to make the 5.51 version of *sendmail* run under the Berkeley 4.2 or 2.9 release of UNIX. The changes neccesary to make it run under 4.2 are very minor and mostly deal with adding undefined symbols. The 2.9 changes are fewer than you would expect, although it still is necessary to drastically trim down the size of *sendmail*'s buffers if you intend to run it on a PDP-11. The 2.9 changes are all enclosed under the BSD29 define.

7.17. Miscellaneous Changes

In addition to what already has been mentioned, a set of changes has been included that only add minor functionality or deal with less important bugs in the original source. They are further described in *Appendix A: List of Affected Files*.

8. CONCLUSION

The development of the IDA Sendmail Enhancment Kit has been going on more or less continuously for over two years. Many were the interim versions and many were the nights spent testing the newly configured system. Still, it is our feeling that the current version is complete enough to be considered generally useful, and it is our sincere hope that you have found your time spent reading this document worthwhile.

The problems with complex hybrid addresses are parts of the growing pains of a rapidly expanding world of interconnecting computer networks. It is this author's opinion that it ultimately will be necessary to agree on some common way of identifying message originators and recipients. In the mean time, we

[11] Unless stdin is connected to a terminal.

will have to face the world as it is and try to relieve users' pains as much as possible by letting the MTAs worry about routing and foreign addressing format issues.

An excellent source for further reading is John Quarterman's and Hosiah Hoskins' article *Notable Computer Networks* [Quarterman86a], which gives a thorough overview of existing major computer networks and supplied services around the world, with an emphasis on electronic mail. Their survey has been an invaluable source of information for this author among others.

Electronic mail is truly an interesting medium. It combines the speed of the telephone with the asychronacity of the written letter. In addition, it gives possibilities for developing all sorts of interesting computerized communication services, such as electronic conferencing systems and database or expert systems consultation services. But above all, it communicates across all boundaries with no respect to operating systems or computer brands. It may not always be painless, but that can be improved. As long as we communicate, everything can be improved.

The Mail Connectivity Conspiracy Continues... [Crispin86a]

APPENDIX A: LIST OF AFFECTED FILES

This is description of all changes made to the *sendmail* source files.

Nameserver Default Argument

daemon.c Changed to return TRUE if *gethostbyname*(3) succeeds and FALSE if not.

parseaddr.c Changed to interpret the **$:** *default* argument and to take care of the returned value of *maphostname*.

Direct Access to Dbm(3) Files

alias.c Changed to allow access using the '@' database.

conf.c Changed to initialize all databases to DB_NOTYETOPEN.

daemon.c The lookup function *mapkey* goes here. It takes four arguments: *db*, the character denoting the database, *key*, the lookup string and buffer in which a result is returned, *keysiz*, the maximum size of the key buffer, and *arg*, which either should be a character string or NULL. The function returns TRUE if a match could be found and FALSE otherwise. The debugging flag 60 may be used to trace database lookups.

main.c Added mappings of **$(** and **$)** to KEYBEGIN and KEYEND.

parseaddr.c Added code to interpret **$(** . . . **$)** constructs.

readcf.c Added understanding of the 'K' option.

sendmail.h Added definitions for the macro characters KEYBEGIN and KEYEND. Added the declaration of the global database file table DbmTab.

Batched SMTP Support

main.c Added the MD_BSMTP option and the usage of *bsmtp* as an alternate name of *sendmail* that automatically will turn on the MD_BSMTP mode.

sendmail.h Added the MD_BSMTP define.

srvrsmtp.c Added the *batched* argument (boolean) to the *smtp* function and changes that will make it mail back errors is *batched* is set.

usersmtp.c Changed the code to automatically generate internal SMTPGOODREPLY (**250**) reply codes to all SMTP commands if the M_BSMTP mailer flag is set.

Separate Envelope/Header Rewriting Rulesets

headers.c Changed to propagate a flag telling if this is an envelope or header address for *remotename* to rewrite.

main.c Trace statement changed to display mailers' envelope and header specific rulesets.

parseaddr.c Added a boolean *headeraddress* argument to *remotename* and code to distinguish between envelope and header rewriting.

queue.c Set both envelope and header rewriting rulesets to -1 in *nullmailer*.

readcf.c Made it parse the '/' option and the extended mailer ruleset specification syntax.

sendmail.h Extended the mailer declaration to include both envelope and header specific rulesets.

Separate Local UUCP Host Name

deliver.c Used when producing "From_" lines.

main.c Added the definition of **$k** to the initial value of **$w**.

parseaddr.c Used when making addresses UUCP relative.

Return Path for UUCP Mailers

deliver.c Changed to look for the mailer flag M_FROMPATH when producing the UUCP "From_" lines.

UUCP Header Address Relativization

parseaddr.c Changed to *uurelativize* addresses after ruleset 4 has been applied if the "mailer flag is set. The actual *uurelativize* is here too.

sendmail.h Defined the M_RELATIVIZE flag to be 'V'.

Support for Multi-Token Class Matches

parseaddr.c Crude code added to enable multi-token class matches.

Support for Embedded Subruleset Calls

parseaddr.c Wrote a separate *callsubr* function to take care of subruleset calls.

Elaborate Matching Algorithm for Unknown Local Recipients

recipient.c Added the *partialstring* matching routine and code that calls it for unknown local recipients.

Support for Maryland's mdbm Package

alias.c Changed to be independent on which package is being used.

conf.h Added the MDBM define, to be used if the mdbm routines should be used instead of the ndbm. Note that NDBM still should be defined.

mdbm_compat.h
A mdbm compatibility file, used to define macros which map ndbm functions to their mdbm equivalents.

sendmail.h Added mdbm compatibility macros and generalized the dbm code in general.

Improved Test Mode Output

main.c Changed it to export the macros' symbolic names.

parseaddr.c Changed it to call *printcav* instead of *printav* when tracing the rewriting rules.

sendmail.h Included main.c's macro table among the global variables.

util.h Changed it to print macros using their symbolic names. Added the *printcav* function, which prints argument vectors without enclosing quotation marks.

Better To: and Cc: Headers of Returned Messages

savemail.c Changed the code to produce the above result.

Queue Bug Fixed

queue.c Set the current envelope's id to NULL before exiting.

Shared Input SMTP Bug Tentatively Fixed

main.c Turned off buffering from standard input before the call to *smtp*.

Optional BSD4.2 Compatibility Code

conf.h Added the definition of *sigmask* if left undefined by <signal.h>. Added mock definitions for LOG_MAIL, TRY_AGAIN, and *h_errno*.

Optional BSD2.9 Compatibility Code

conf.h Added the definition of EPROCLIM (not really applicable under BSD 2.9) and the inclusion of ../lib/libndir/dir.h.

conf.c Changed to include <a.out.h> instead of <nlist.h>, and to search /unix instead of /vmunix for kernel symbols.

daemon.c Changed to use *gethostname*(3) instead of <whoami.h> to find out its local host name.

deliver.h Changed not to include <netdb.h>

err.c Changed not to include <netdb.h>

Miscellaneous Changes

deliver.c An array of verbose mailer error messages has been added together with code that prints out the error in text instead of just giving the code in numeric form. [Incorporated from USENET]

main.c A new **−Z** command line option has been added, which defines the name of the frozen configuration file in analogy with **−C**.

parseaddr.c Has been changed to compare aliased users with *sameword* instead of *strcmp* to make up for differencing case.

sendmail.h The reference to <sys/syslog.h> has been replaced by <syslog.h> unless sendmail is compiling on a VAX. I'm not sure that this is correct, but neither our Suns, nor the Gould had syslog.h in that directory. Also, *s_host*, has been forced undefined if compiling on a Sun. This is because it is defined in one of the Sun's include files as well.

srvrsmtp.c The **$s** macro was set to be the name of the remote host, but then cleared before it could be used. It is now being set *after* the relevant cleanup routines have been run. An unnecessary additional rewriting of envelope recipient addresses in the SMTP routines has been removed. It could even be harmful if it was to be left alone.

../doc/op.me Added text to describe our new, wonderful features.

APPENDIX C: CUSTOMIZATION PARAMETERS

The following is a list of all *m4* identifiers used in the configuration file. All of them are optional.

ALIASES
: Name of the aliases file, defaults to sendmail's default.

BSD29
: Activates various hacks for usage on BSD 2.9 systems.

BANGIMPLIESUUCP
: If defined, will inhibit domaintable lookups for unqualified nodes first in '!'-paths and always interpreting them to reside in the UUCP pseudo-domain.

DECNETNODES
: A file containing DECnet host names. Used in combination with DECNETXTABLE to determine delivery through the DECnet mailer and when to expand flatspaced DECnet host names into domains.

DECNETXTABLE
: The DECnet translation table. Returns a node's DECnet host name if given its domain name. (*Dbm* file, see ruleset 24 for more info).

DEFAULT_HOST
: Explicit host name, replaces automatic definition of **$w**. [Not normally used]

DEFAULT_DOMAIN
: The string that (+ '.') will be attached to **$w** to form **$j**, this node's official domain name. Should only be left undefined when your hostname (**$w**) already contains its domain.

DOMAINTABLE
: *Dbm* database used for hostname canonicalization, i.e. to find the official domain name for local or otherwise unqualified hosts.

GENERICFROM
: A database mapping actual user names to generic user names. Used instead of HIDDENNET in a heterogeneous environment.

HIDDENNET
: Points to a file containing a list of host names, one per line. Mail from users on any of these hosts will have their host names substituted for our host, **$w**.

LIBDIR
: The directory that will hold most data files, including sendmail.{hf,st}; defaults to /usr/lib/mail.

PATHTABLE
: The heart & soul of this sendmail configuration—the pathalias routing table in *dbm*(3) format, as produced by the *pathalias*(1) program. If you want some kind of routing capabilities, you either define this or rely on RELAYHOST/RELAYMAILER.

MAILERTABLE
: A *dbm* table mapping node names to "mailer:host" pairs. It is used for special cases when the resolving heuristics of ruleset 26 aren't enough.

NEWALIASES
: If defined, will make ruleset 26 return all addresses as local. This should be used by the newaliases program only when parsing the aliases file if you want to handle non-local aliases as well as local.

PSEUDONYMS
: Additional names that we are known under (in addition to the nicknames returned by *gethostbyname*(3)).

RELAY_HOST & RELAY_MAILER
: Name of the host and mailer to ship unknown recipient addresses to. Not necessary to define if you have a complete PATHTABLE.

RSH_SERVER
: If defined, do local deliveries by *rsh*(1)'ing /bin/mail on the RSH_SERVER host.

SPOOLDIR
: Directory for sendmail queue files; defaults to /usr/spool/mqueue.

UUCPNAME
: This node's UUCP host name, if different from **$w**.

UUCPNODES
: A file containing names of directly connectable UUCP nodes, normally /usr/lib/uucp/L.sys.

UUCPPRECEDENCE
: If defined, will change the interpretation of mixed '!'- / '@'-addresses to use heuristics instead of always preferring RFC822 style.[12]

UUCPRELAYS
: Name of file containing names of known (UUCP) relays. Header addresses containing paths through any of these will be shortened by having the path to the relay removed. (It is assumed that paths to each of these are known to everybody)

UUCPXTABLE
: A table mapping domain node names to UUCP node names. Used in envelope addresses sent using UUCP/rmail.

[12] Not fully tested.

APPENDIX D: LIST OF DOMAINS

The following is a list of all top-level domains officially registered with the SRI Network Information Center as of May 13, 1987:

Organizational Domains

Domain	*Organizations*
COM	Commercial
EDU	Educational
GOV	Government
MIL	Military
NET	Network Administrations
ORG	Other Organizations

National Domains

Domain	*Country*
AU	Australia
DE	Germany
FI	Finland
FR	France
IL	Israel
JP	Japan
KR	Korea
NO	Norway
NL	The Netherlands
NZ	New Zealand
SE	Sweden
UK	The United Kingdom
US	The United States of America

Network Domains

Domain	*Network*
ARPA	The Advanced Research Projects Agency Network

which is the only officially registered network domain, but others more or less widely recognized are:

BITNET	IBM Network, includes NETNORTH and EARN
CSNET	The Computer Science Network
JUNET	The Japanese University Network
MAILNET	A now deceased(?) networking project centered around MIT-MULTICS
OZ	The Australian Computer Science Network (ACSnet)
SUNET	The Swedish University Network
UNINETT	The Norwegian University Network
UUCP	The UNIX-to-UNIX-Copy Network

APPENDIX F: LIST OF DATA FILES

The following is a list of all data files used by the *sendmail* program in conjunction with the supplied configuration file. Not all of them are used in the sample setup. Not all of them will probably be needed by your setup either.

aliases . . .is better described in *aliases*(5).

domaintable . . .is domain name table used when finding a node's canonical name. It is written in *dbm*(1) **parse** format.

generics . . .is automatically produced by the *xalparse*(8) program from the xaliases file. It contains entries mapping sending user's real addresses to generic ones.

hiddennet . . .may contain a list of node (domain) names that should be hidden by the local host's name on header sender addresses.

mailertable . . .defines the mapping from node (domain) names to *Special Mailers*. It is written in *dbm*(1) **parse** format, with each value being a "mailer:host" tuple.

newaliases.cf . . .is a configuration file produced by having defined the NEWALIASES identifier in the master file. This should only be used when producing a new *aliases*(5) dbm table, because it returns all addresses to the LOCAL mailer in order to accept non-local aliases.

pathtable . . .is the basis for all explicit routing decisions. It is written in *pathalias*(1) format and describes how systems connect to each other and where to forward according to (parent) domains.

xaliases . . .is the extended aliases file that is used to produce the *aliases*(5) file together with the *generics* file. It is further described in the section about *Managing Generic Mail Addresses*.

\k:*network***/nodes\h'|\n:u'\v'+1'***network***/xtable**\v'-1'

. . .are used when translating node names from one network to another. The **nodes** file contain all internally known node names for the network and **xtable** is a translation table that maps official domain names to the network's internal node names.

uucp/relays . . .contains node names of well-known UUCP relays. It is used when compressing header paths in the header rewriting rulesets.

APPENDIX I: INSTALLATION INSTRUCTIONS

These instructions will tell you step-by-step how to install and bring the Kit's *sendmail* system up. The source code modifications are given as context *diff*(1)'s, based on the BSD 4.3 release of *sendmail* (version 5.11), ready to be installed using Larry Wall's eminent *patch* program. You will also need Maryland's *mdbm* library if you intend to use this instead of *ndbm*. Finally, in order to automatically produce routing tables, you will need Peter Honeyman's *pathalias* program. All of these are available from your nearest USENET **comp.sources.unix** archive.

Now, assuming that you have read this far and made up your mind to try it all for yourself, do the following:

[1] Unpack the Kit in the **sendmail** directory (preferably). This should give you a new **ida** subdirectory with all the Kit's files.

[2] Goto **sendmail/ida** and check that you agree with the Makefile's definitions. If you change anything, do a "make configure" to propagate those changes to the subdirectories' Makefiles.

[3] Goto **sendmail/ida/doc**. Doing "make doc" will print out this paper but since you already are reading it, this might be unneccesary. Anyway, do "make man" to print the manual pages. Do "make install" to install them in your manuals directory.

[4] Goto **sendmail/ida/patches** and do "make backup." This will create a backup copy of **sendmail/src/*.[hc]** and **sendmail/doc/op.me** in **Backup.tar**. You can restore them if necessary by performing "make restore," still in the patches directory.

[5] Do one of "make bsd43," "make bsd42," or "make bsd29" to patch the *sendmail* source to the required compatibility level. You will need *patch*(1), for this or else edit the files by hand. Look out for rejected patches.

[6] Goto **sendmail/src** and recompile *sendmail*. See that it still works. Your old configuration file *should* still work unless you depend on some obscure side effects. Note that a BSD 4.2 configuration file might not work with *sendmail* 5.11.

[7] Goto **sendmail/ida/aux** and do "make" to compile the auxiliary programs. Try them out, guided with the newly printed manual pages.

[8] Do "make install" to install the programs in BINDIR (**/usr/local/bin** by default; but another choice would be **/usr/lib/mail** if you only intend to use them with this kit). It's also about time to (manually) do a (symbolic) link from **/usr/ucb/bsmtp** to **/usr/lib/sendmail** if you intend to receive batched SMTP mail.

[9] Goto **sendmail/ida/cf** and inspect the supplied *m4*(1) configuration definitions. Send **Sendmail.mc** to your line printer and study it. Do "make" to see how the sample configurations look expanded.

[10] Goto **sendmail/ida/lib** and inspect the supplied sample data files. Try applying the *xalparse* program on the *xaliases* file if you feel like it.

[11] Determine your site's routing capabilities and create your corresponding data files in LIBDIR. Go back to **sendmail/ida/cf** and create your own *m4*(1) configuration file using the samples as templates. Produce an actual, personal **sendmail.cf** file.

[12] Try out your new *sendmail* system.
Good Luck!

[13] Mail problems or comments to lel@ida.liu.se.

APPENDIX P: LIST OF AUXILIARY PROGRAMS

All programs but *scanf* are further described by their manual pages. What you find here is only a short overview of their functionality.

dbm A general *dbm*(3) database management tool. Clears, loads, and dumps complete databases. Stores, fetches, and deletes individual keys. Also includes a special parser that produces key/value pairs from a compressed but easily readable format.

mkdomext Make the extended set of domain names. This program acts as a filter that, when supplied with fully qualified domain names on stdin, will generate all legal abbreviations of these on stdout with respect to the parent domains given on the command line. It is used to be able to recognize unqualified local subdomains and sibling domains from one or more positions in the domain hierarchy. See RFC822, section 6.2.2 for a rationale.

rmail Yet another implementation of the remote mail receiving program. This time with raw header line logging together with *dbm* lookup of remote node names and a more liberal parsing of "From_" lines. Needs to run "seteuid root" to be able to use the macro setting options of *sendmail*. No new manual supplied, your old one will have to do. Actually, your old rmail itself will probably do too.

scanf Quick hack to scan and extract substrings from input lines using the *scanf*(3) function. No manual page, the source code is self documenting.

xalparse A program that parses an extended aliases file and produceces an ordinary aliases file together with a file with generic address translations. It's all described in the manual page and in the section about *Managing Generic Mail Addresses*. Quod vide.

APPENDIX R: REFERENCES

References

Carrol96a.
Lewis Carrol, *Alice in Wonderland* (1896).

Crispin86a.
Mark Crispin, *(Private Communication)* (1986).

Crocker82a.
David Crocker, *Standard for the Format of ARPA Internet Text Messages,* RFC822 (1982).

Crocker87a.
David Crocker, "Networking Considered Harmful," *Unix Review* **5**(3) (1987).

Horton86a.
Mark Horton, *UUCP Mail Interchange Format Standard,* RFC976 (1986).

Kille86a.
Steven Kille, *Mapping Between X.400 and RFC822,* RFC987 (1986).

Kluger86a.
Larry Kluger and John Shoch, "Names, Addresses, and Routes," *Unix Review* **4**(1) (1986).

Laotse00a.
Laotse, *Tao Te Ching,* Book 1, Verse 1 (ca 500 BC).

Malaga-Torremolinos84a.
Malaga-Torremolinos, *Message Handling Systems: System Model—Service Elements,* X.400 (1984).

Mockapetris83a.
Paul Mockapetris, *Domain Names—Concepts and Facilities,* RFC882 (1983).

Postel84a.
Jon Postel and Joyce Reynolds, *Domain Requirements,* RFC920 (1984).

Quarterman86a.
John Quarterman and Hosiah Hoskins, "Notable Computer Networks," *Communications of the ACM* **29**(10) (1986).

Rose83a.
Marshall Rose, *Proposed Standard for Message Header Munging,* RFC886 (1983).

APPENDIX T: TABLE OF CONTENTS

Sections

Appendices