Upas - a simpler approach to network mail

David L. Presotto (research!presotto)

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

*Upas** is a mail interface that routes messages between existing network specific mailers, users, and user mailboxes. It uses a regular expression based mini-language to convert mail addresses into the commands needed to route the mail to the intended destination. Upas is the mail interface for the Eighth Edition of the Unix† Time-Sharing System.

Introduction

Our entry in the 'mail race' sprang from events similar to those motivating the development of many mail systems. For many years a short and simple mailer was used to deliver local mail and to route mail via our home grown networks. Although its user interface left a little to be desired, its reliability was so high that great trust was put into it. However, as we gained access to more and more networks, particularly ones over which we had no control, the situation quickly deteriorated. Each of these networks had their own mail 'standards' and addressing conventions. With some trepidation, we absorbed these standards into our mailer. Its simplicity was quickly lost along with its fabled reliability. Realizing our danger, we decided to step back and see if there was a way to get back to a simple, well understood, and thereby reliable mail system.

The job to be performed by a network mail system is illustrated by figure 1. A mail system is essentially a large switch for handling the routing and delivery of messages. As a router it must be conversant in the various network protocols, be able to decipher destination addresses, and pass messages along to the next network. Sometimes it actually gets to deliver a piece of mail to a mailbox. Also, since there is no common mail format, the mail system must convert messages from one format to another as it routes them from network to network. Because of the number of networks and mail formats, this can easily lead to thousands of lines of code. Our task was to decide how to partition the task in order to create a manageable yet efficient mail system.

Some Observations

The task of interfacing to a particular network is often a messy and arbitrary thing. Fortunately, most entities (corporations, governments, committees) that design network protocols also provide code (i.e. mail programs) that understand these protocols. In our experience, it has always been easier to interface one of these mailers to our mail system than to incorporate the new protocols into our existing mailer. Also, code provided by someone else is supported by someone else. As network protocols change it is easier to pick up the new version of the network mailer than to rewrite our mailer.

Although there are many networks, there are far fewer message formats. It is clear that a message needs a destination address and possibly even a reply address. However, the imposition of further structure on the message is at best distasteful, at worst obstructive. Imagine what postal delivery would be like if the

---

* **upas**,[1] *u´pas, n.* (in full **u´pas-tree´**), a fabulous Javanese tree that poisoned everything for miles around; Javanese tree (*Antiaris toxicara*, of the mulberry family): the poison of its latex. [Malay, poison.]

† Unix is a trademark of AT&T Bell Laboratories

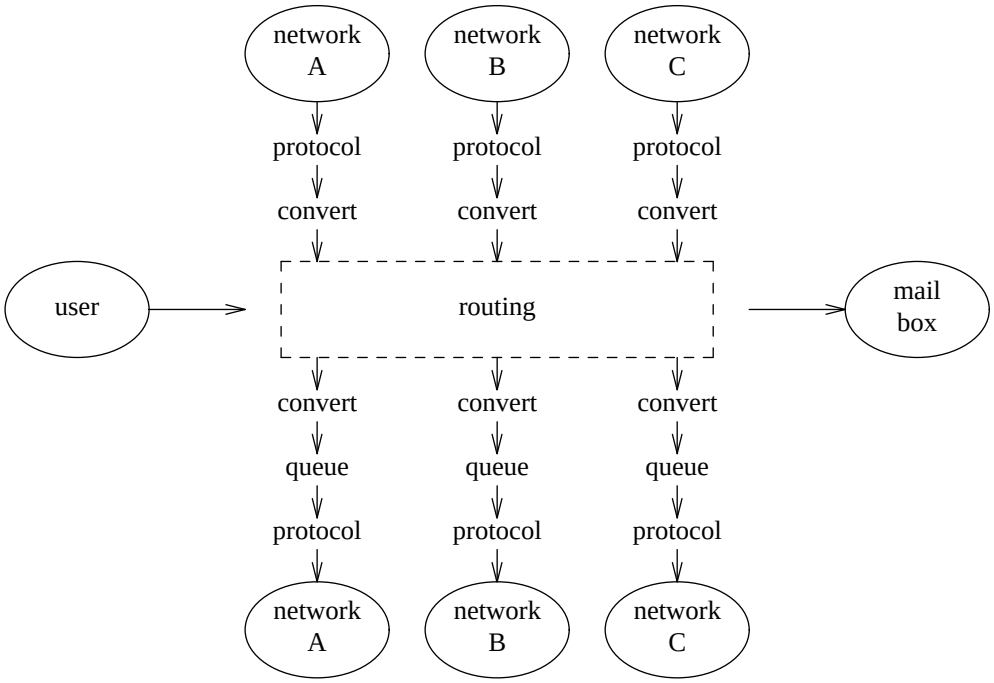

Figure 1. The functions to be performed to route network mail.

Postal Service opened each piece of mail to ensure that it is correctly dated and signed, that the form of address is correct, and that the company letterhead obeys some preconceived format, refusing delivery if any of these conditions are not met. Unfortunately, some networks impose such requirements. For a message to obey one standard is difficult enough. To expect it to survive a number of conversions between restrictive standards constitutes wishful thinking. Because of this, most networks adopt standards established by older or larger networks. Therefore, although there are many networks, there are relatively few message formats.

A network address describes a path through a number of machines and networks. This path may be rather simple, consisting of a single machine and user name. Often, however, the path crosses a number of administrative domains. Each such domain imposes some rules for structuring paths within the domain. Unfortunately, there is no adhered-to standard for binding the path segments from each domain into a single address. The networks differ on direction of binding (person@machine vs. machine!person), delimiters (‘.’ vs. ‘@’ vs. ‘%’), quotation marks, and even case sensitivity. Therefore, there is no fixed way to correctly parse and understand a network address. Instead, there are conventions which tend to be very short lived, usually until someone issues a new RFC or a new network appears. As a relatively simple example, consider a message sent from one UUCP[2] network, through ARPAnet, to another UUCP network. The address format might be something like:

```
placeA!placeB!"placeD!placeE!person"@placeC
```

The rules for parsing such an address are easily defined. Unfortunately, the conventions underlying the rules change from day to day. Once you’ve managed to write your code, the administrator at placeB may decide that he won’t accept quotation marks in an address and would really like the address to look like:

```
placeA!placeB!@placeC!placeD!placeE!person
```

A new set of parsing rules now have to be defined. In our experience these changes happen with maddening frequency. They are the direct result of there being no single comprehensive standard or administrative authority. Therefore, we have to treat address parsing rules as ephemeral. Any network mailer should be able to change its address parsing rules frequently and with little difficulty. Tying them to one particular standard such as this week’s ARPA rules is equivalent to planned obsolescence.

Finally, we should make a point about reversibility that many other mail designers seem to have missed. In addition to parsing destination addresses, mailers are expected to maintain some form of return address attached to the message. This often involves changing the current return address to one that the mailer will accept as a reply destination. A mailer should parse and modify return addresses using the same rules as it does for destination addresses. Otherwise, as is too often the case, the mailer will reject the very addresses that it has provided for replies.

A Solution

The best solution would have been to throw out all the so-called standards and create a single coherent scheme for formatting and addressing mail.[3] However, since we have no power to impose such a scheme, we have tried to use the above stated requirements and observations to build a mail system that makes the best of a bad situation.

The structure of our mail system is depicted in figure 2. Each network has its own interface program for message reception and transmission. In general these are the network specific mailers provided with the networks. When a message enters from a network, the network specific mailer gives it to Upas. Upas then either deposits the mail in a local mail box or routes the mail to the next network. A format specific filter may be called to convert the message from network format to one Upas understands or vice-versa; RFC 822[4] and Unix formats are built in. The rest of this paper describes how Upas preforms its message routing and the ease with which new networks can be added.

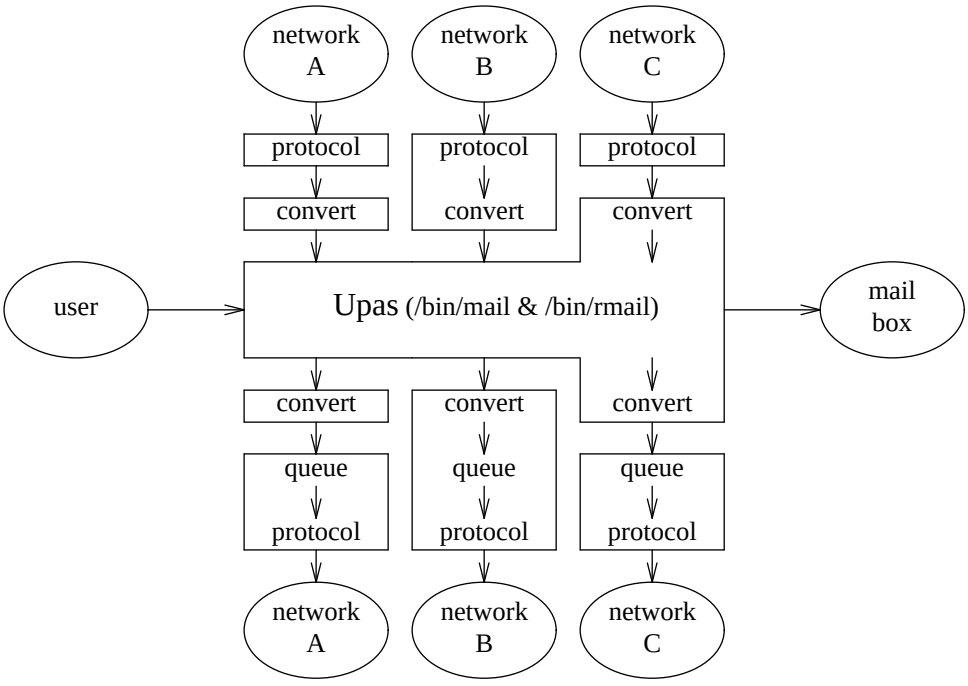

Figure 2. The structure of Upas. Each solid box represents a separate program.

Message Routing

The routing of messages is determined by a destination address and by a set of rewriting rules kept in a configuration file, /usr/lib/upas/rewrite. Each line of the file is a rule. Blank lines and lines beginning with ´#´ are ignored.

Each rewriting rule consists of four strings:

pattern An *ed*(1)-like regular expression, with simple parentheses () playing the role of \(\) and with the

+ and ? operators of *egrep*(1). The *pattern* is applied to mail addresses.

machine An *ed*(1) style replacement string that picks the source or destination string from an address matched by the *pattern.* The substring, '*\s*', is replaced by the login id of the sender.

command An *ed*(1) style replacement string to generate a command to deliver messages to the destination matched by the *pattern.*

conversion The name of a conversion to be performed.

When delivering a message, Upas starts with the first rule and continues down the list until a pattern matches the destination address. If the rule contains no command, the mail is appended to a local user's mailbox. If the rule does contain a command, Upas starts the command and pipes the message to it, performing any requested conversion. For example, the following rule can be used to specify that all RFC 822 style addresses should be passed to a particular gateway machine (research):

```
^[^!]+[.@%].+$  ""  "uux - research!rmail \\(&\\)"
```

The first expression, `^[^!]+[.@%].+$`, is used to match the destination address and the third, `"uux - research!rmail \\(&\\)"`, is used to form the command. The second expression (blank here) relates to forwarding restrictions and is explained below. The gateway machine can have more restrictive rewrite rules to pass a message onto a particular network. For example, if a message were a CSNET message, the following rule might exist on the gateway machine:

```
^(.+)[.@]csnet$  ""  "/usr/mmdf/lib/mail \1.csnet-relay"
```

Here /usr/mmdf/lib/mail is the program provided by csnet to interface to their network.

Rules for most networks can be specified in one or two lines. In addition, the rules are in a language familiar to most experienced Unix programmers — the regular expressions seen in many editors, languages, and utilities. By using such a mini-language, it becomes an easy task to build or modify Upas configuration files. The result is that configuration files rarely contain gross mistakes and take very little time to create and to edit when addressing conventions change.

Message Format Conversion

Upas provides for both incoming and outgoing conversions between internal format and those of the networks. For incoming mail, the conversion to be performed is specified by the name with which Upas is started. For example, starting Upas as 'rmail' causes it to perform no conversion. Starting it as 'cmail' causes it to convert from RFC822 format to its own (Unix) format. For outgoing mail, the conversion to be performed is specified by the optional last field in the rewrite rules. For example, the CSNET rule in the previous section might more correctly be written:

```
^(.+)[.@]csnet$ "" "/usr/mmdf/lib/mail \1.csnet-relay" rfc822
```

The 'rfc822' requests that the message be converted to the RFC 822 format before being piped to /usr/mmdf/lib/mail. For formats not recognizable to Upas, a filter process can be interspersed to perform the conversion either on input or output. Upas currently understands only RFC 822 and Unix message formats.

Discouraging Anti-social Behavior

It is often desirable to control the use of a machine as a forwarding site. Such is especially the case when the forwarding of mail has a devastating effect on either available cycles and telephone usage. The second field in a rewriting rule is used to allow only *friendly* machines to forward mail through Upas. This field, the *machine* field, is a replacement string matching the source and destination machine names from the reply and destination addresses. These names are checked against a file, /usr/lib/upas/forwardlist, which contains a list of friendly machines. If neither name is found in the list, the message is returned to the sender. For example, a UUCP address may have the following rewriting rule:

```
^([^!]+)!([^!]+)$   \1  "uux – –a \s \1!rmail \\(\2\\)"
```

Mail sent to ''placeA!person'' by ''placeB!person'' will be refused if neither placeA nor placeB is in the

forwarding list.

If the file of friendly machines doesn't exist, the mail is forwarded unconditionally.

User Control

Users often wish to specify alternate ways to dispose of their mail. Upas allows this in a manner similar to Sendmail.[5] The first line of a user's mail file is interpreted as a command to the mail system. If the line is of the format

Forward to list-of-addresses

the mail is forwarded to each recipient in *list-of-addresses.* While this can be used to forward a single user's mail, it can be also be used to create mailing lists. To do this, one creates a file in the mail directory whose name is that of the mailing list and which consists of ''Forward to'' followed by the list of recipients.

If the first line is of the format is

Pipe to shell-command

shell-command is executed when mail is delivered, with the message as standard input.

Loop Detection

Detecting forward loops, like those provoked by 'Forward to' is difficult. It involves combining the forwarding lists of all involved machines into a single directed graph and then performing a search or partitioning to detect cycles. However, if we allow a detection algorithm to reject some legal although highly unlikely cases along with real loops, we greatly simplify the problem.

In the case of a single machine, an infinite forwarding loop corresponds to infinite recursion of the mailer. If a mailer rejects any message that results in recursion past a certain depth, it will reject all loops and some small number of legal but very long mail redirections. In our case a depth is 32 and to date, no legal forwarding loop has been more than 3 steps long.

In the case of a multi-machine loop, the recursion technique is not valid. However, we can still use a similar method. Instead of counting recursion, we scan the From line to see the number of times the local machine name occurs in the path. If this exceeds a limit (in our case 8), the mail is returned to the sender.

A Comparison With Sendmail

Upas is an attempt to solve the same problem previously attacked by Sendmail. Upas owes much of its design and success to Sendmail. The idea of designing Upas as a central switcher communicating with network specific mailers comes directly from Sendmail. The reasons we wrote Upas and didn't just adopt Sendmail are:

- We strongly favor messages whose only formatted portion are the destination and reply addresses. Sendmail has an unfortunate predilection for verbose and rigidly structured messages that we would like to avoid.
- Sendmail configuration files are famous for their inscrutability. We wanted a system that had simpler and therefore more easily verifiable rewriting rules.
- Sendmail is over designed for our needs including queuing (performed by our network specific mailers), SMTP support, batching of mail transmissions, aliasing, inclusion, etc. This extra design makes Sendmail more complicated and harder to support.

Summary

We have presented a simple yet flexible network mail system. It gains its simplicity from a number of assumptions which are valid in most networked computers. By using existing network specific mailers as expert systems that deal with network details, Upas itself remains relatively simple and understandable. By converting the most common message formats internally while allowing other formats to be converted by filter processes, Upas remains efficient yet extensible. Finally, by using a mini-language already

familiar to most Unix programmers, Upas is easily modified to respond to changes in the name space and topology of the network.

References

1. *Chambers Twentieth Century Dictionary*, W & R Chambers Ltd., Edinburgh, GB (1976).
2. Nowitz, D. A., ‘‘Uucp Implementation Description,’’ *Unix Programmer’s Manual, Seventh Edition, Volume 2*, Bell Laboratories (October 1978).
3. Pike, R. and Weinberger, P. J., ‘‘The Hideous Name,’’ *USENIX Summer Conference Proceedings* (June 1985).
4. Crocker, D. H., ‘‘Standard for the Format of Arpa Internet Text Messages,’’ *RFC 822*, Menlo Park, California, Network Information Center, SRI International (August 1982).
5. Allman, Eric, ‘‘SENDMAIL - An Internetwork Mail Router,’’ *Unix Programmer’s Manual, 4.2 BSD, Volume 2C*, University of California, Berkeley (July, 1983).