The Design of a Mail System for the KA9Q Internet Protocol Package

Bdale Garbee, N3EUA
Gerard van der Grinten, PAØGRI

ABSTRACT

The current implementation of the mail manipulation system that has been built by N3EUA and PAØGRI for the KA9Q Internet Protocol Package is described briefly. A proposal for the next generation of network mail handling for the KA9Q Internet environment is described. An important change in the way we can and should think about electronic mail in the amateur packet radio world is discussed.

Background:

Since the conception of "Amateur Packet Radio", quite a bit of evolution has occurred. Many lessons have been learned the hard way, and many are yet to be learned. However, one point which has become increasingly obvious to those of us involved in the development of packet radio software, is that a much more consistent and powerful way of dealing with electronic mail messages is

In the beginning, there was raw data transfer from point A to point B without any protocol, and it was good. Then came a protocol called AX.25, which allowed for error-free point-to-point virtual circuit links, and it was better. Most recently has come an implementation of the DARBA TCP/IP protocol family, and it is the best so far. The fact that this protocol family includes a defacto real-world mail-handling protocol, SMTP, has caused some interesting changes in the way we think about mail in packet radio.

The current state-of-the-practice in amateur digital mail handling is for one user, or ham, to "log into" a local PBBS system, by establishing a virtual circuit to the PBBS's user interface. He can then enter, read, or delete a given message, and then "log off". Whenever users are not connected to the PBBS, it is available to attempt "forwarding" of the mail, a process in which "packets" consisting of entire messages are routed from one PBBS to another. Once the message has arrived at the destination system, the end user or ham can read the message using the same "connect to the PBBS" sequence that the sender used.

There are some fundamental design problems with this process. It works, and has solved. the first part our mail-handling needs. But it requires that both the sending and receiving participants spend a non-trivial amount of their time interacting with a remote user-interface, which implicitly includes waiting for channel throughput on the AX.25 circuit. In addition, there are very few services available to the user, such as message archiving or printing, message forwarding, editing of messages, etc. It is also the case that some mechanism (usually beacons) must be provided to allow the receiving mail user to know that there is a message waiting for him on his local PBBS.

It is interesting to note that these limitations are well-known and understood in the packet community. An interesting trend has been the creation of programs to allow machines such as the Tandy 100 and C64 to serve as "mini-PBBS" systems, or "personal mailboxes". Moving as much of the mail user interface from a remote system to the user's local computer as possible provides several benefits. The user need no longer suffer the delays associated with packet flow on the channel while interacting with the mail system, and it becomes possible to provide features such as message filing, printing, and more extensive reply and forward capabilities. While this may not have been possible in the early days of packet radio due to the use of dumb terminals as packet stations, almost all packet radio stations today include a computer as the end terminal.

While a transfer of the mail user interface from a PBBS to the user's local machine is in theory possible with any underlying network protocol family, the recent development by **KA9Q** and others of a complete implementation of the DARPA TCP/IP protocol family, including the defacto-standard mail transfer protocol SMTP, has hastened the need for such a transfer. Implementing a rational mail user interface and underlying servers for the **KA9Q** Internet Package has been a most challenging task, with the resulting system's design principles and implementation tradeoffs occupying the remainder of this paper.

The Current Implementation :

The current mail system in the **KA9Q** Internet package consists of an SMTP server and SMTP client that are integrated into the NET.EXE package, and a separate mail user-interface. The mail user interface, while currently sporting a very terse user interface, provides facilities for creation and reading of messages, and some crude message filing capabilities.

When a user wishes to send a mail message, he runs the mail user interface program ***BM***, and enters the destination address, subject, and text of his message. The message is queued in a directory on his machine for later transmission. At regular intervals, the SMTP client in NET.EXE scans the queue directory and attempts to process waiting outbound mail messages. Each message is delivered by the client directly to the server at the destination station.

When an incoming message arrives at the local SMTP server, it is appended to a mailbox file, the name of which is based on the **username** given in the destination address. Multiple real or pseudo-users (for mailing lists) are therefore supported on a given machine. At his leisure, the user can run the ***BM*** user interface to read and manipulate the messages in his mailbox file(s), and/or can manipulate them directly as disk text files.

The biggest change from existing PBBS mail handling is that the user never has to wait for a packet to be transferred on the channel. All on-air activity is handled automatically "in the background".

Environment:

As may be obvious, the design concepts and implementation ideas presented here are applicable regardless of the particular computer facilities in use, and/or the specific low-level transport protocol and mechanism available. However, for the sake of comparison and reference, we will describe briefly the hardware and software environment under which our mailer implementation has been developed.

The **KA9Q** Internet Package has been developed on the IBM PC, and close compatibles. It uses the Intel Small Memory Model, which allows 64k of code space and 64k of data space on the 8088 processor. In an effort to avoid the necessity of going to a large memory model with the attendant segment maintenance overhead on the 8088, and as an aid to generality and portability, functions such as the mail user interface are being developed as separate applications, designed to run concurrently with the **KA9Q** Internet Package using a PC multitasking package such as Doubledos. In the future, the ideas and algorithms presented here will most likely be implemented under Minix/Unix as well.

Hardware requirements are essentially the same as for the **KA9Q** package itself, except that mail handling can frequently consume considerably more disk space that is required for the protocol package to run, meaning that a large forwarding site will need a hard disk. A simple single or dual user PC should be able to get by on one 360k floppy drive.

All of the mailer software, as with the rest of the **KA9Q** package, has been written in C using the Aztec MS-DOS compiler. It is therefore extremely portable, with the exception of a small module of PC-specific file-system oriented functions, which should be easily translatable to other operating environments.

Top-Level Design - Our Goals:

The next step in the development of our mail system is partitioning of the mail handling tasks. We choose to separate our operations into four distinct categories: user interface, routing, reception, and delivery.

The user interface level is fairly obvious. This is the hunk of code that the user actually sees, and which allows him to enter, read, print, file, delete, forward, etc., all of the messages he must deal with. There can and should be an ability to support multiple user interfaces, to deal with varied and changing user needs, hardware availability, etc.

In the middle, **a** "routing agent" receives requests from user interfaces, and determines **the** appropriate target address and delivery agent for each message. It is the only place in which knowledge about the mapping between mail addresses and mail delivery techniques should exist.

At the other end of the scale are the (potentially multiple) mail delivery agents. These are the functional blocks designed to take a message in a reasonable format (for them), and actually do the work of deliver-, ing it. In the SMTP world, this might entail opening a TCP connection to the destination host, and engaging in the SMTP protocol to deliver the message. In the UUCP world, this might include looking up and dialing an appropriate telephone number, performing the login dialog, **and then engaging** in the required uucp protocols (similar to the way uucico works under Unix). In the Fido world, this might simply be a matter of reformating the message file into a Fido- compatible message, and moving it to an appropriate directory on a PC's hard disk.

The final class of entity is the mail reception agent. **Each server receives inbound messages from one of** potentially many mail services and puts them in the routing agent's queue, in **the same manner that a user** interface queues a new message for processing.

The flow of information therefore includes an influx of messages to be handled, both locally **created by the** user and his interface, and remotely created and received by a server. All pending messages pass into a queue and are processed by the router. The router places each message, potentially with some additional delivery information, into the queue associated with a particular mail delivery agent. **Each delivery agent** watches its queue for messages to process, and attempts to deliver them.

The Core of the Implementation:

Some simple observation of the mail management system that we have thus far described should reveal the fact that a considerable amount of work will be done enqueueing and dequeueing **messages in a set of** queues, one for the routing agent and one for each delivery agent. To allow for a consistent means of handling queue'ed messages, a combination of file system usage and custom databasing routines has been laid out.

In the most general case, a queue entry will consist of two files, one for the text of the **message (most** likely in RFC822 format), and one for the "work" information required by a particular delivery agent, or the router. A very reasonable idea is to force each of these files to have a completely unique name. Thus, the primary task of the database is to manage file naming, and connection of **a** given text file with it's associated work files. **Exact** details of the database are the topic of current design effort **as of** this writing.

The routing agent is an extremely important part of this design. It must "read" each message and make some determination about which delivery agent should be used, and what address the delivery agent should try to deliver to. In some situations, such as forwarding uucp mail, which is frequently source-routed, there is information external to the message body that will allow the router to '"cheat". In **the** majority of cases, however, the router will have to deduce the appropriate agent and address by reading the **To:** address field in the RFC822-format message body, and comparing it to patterns present in a table of heuristics.

Conclusion:

The intent of this paper is to document the current status and future plans of those working on the mail user interface associated with the KA9Q Internet package. Because a great deal of the **detail about how the** system is and will be put together is still in **a** state of flux, it would seem inappropriate to discuss **that** detail here. Those interested in following the progress of, and/or assisting in the work remaining in this project, are directed to contact one or both of the authors.

References:

1. **RFC821,** Simple Mail Transfer Protocol.
2. RFC822, Standard for the Format of Internet Text Messages.
3. Allman, Eric, "SENDMAIL - An Internetwork Mail Router", supplied with the 4bsd Unix **manual set.**