by Peter Collinson, **Hillside Systems**

DAVID FAUST

Sendmail

It's been quite some time since I wrote about Sendmail–April 1994 to be exact. The computing world has changed dramatically since then and, of course, the Internet has grown immensely. More and more people are now contactable by e-mail, with the expectation that you should be able to send e-mail to everyone on the planet.

People always cite the Web and the Web browser as the "killer app" that launched the Internet. It's hard not to agree with them, and I'll guess that the ability to get at the Web is why people initially connect.

However, once online, I'll bet that many people become disenchanted to discover that the Web is stuffed with sites mostly created by designers who don't understand or care about download speeds; don't realize that some people prefer not to use Internet Explorer; and who spend their lives handing out design awards amongst themselves for Flash-enhanced pages whose antics get in the way of the information they are trying to display. The best button on a Flash page is the one that says "Skip Intro." The Web is fundamentally about information, not advertising, and it looks as if the advertising people are finally beginning to figure this out.

Of course, I am demonstrating the jaundiced view of someone who has had a Web site since the Web started. There's evidence that things are getting somewhat better. I use the Internet regularly to find out about U.K. train times, work out how much it will cost to send a letter by surface mail, manage my bank and credit card accounts, worry about flight times, and buy bits of computers. Most of these facilities have appeared in the last couple of years, and most of them do have reasonably designed usable interfaces.

Anyway, even if new Internet connectees quickly become weary of the Web, it's e-mail that keeps them coming back for more. It is still true to say that Sendmail is responsible for handling most of the e-mail on the planet. There are other systems, of course, but Sendmail's long pedigree tends to mean that it works faster and more reliably than its competitors.

Sendmail is now a supported product. Eric Allman, its author, started Sendmail Inc. a couple of years ago, and that company has migrated Sendmail onto Windows platforms, created a visual support interface, and is selling support for the product. It's also maintaining the UNIX version that remains free to all with an Internet connection from `http://www.sendmail.org/`.

What Does *sendmail* Do?

Many people are hazy about how e-mail works. Basically, e-mail gives you the ability to transmit a file from your machine to another. The destination address is a person's mailbox, a file on a machine somewhere where that person expects to receive the mail. To compose and also read the mail, you will use one of a variety of mail handling programs.

Where does Sendmail fit in? The first thing to understand is that Sendmail doesn't generally talk directly to you, the user sending or receiving mail. It's designed to simply be a switch, taking e-mail in, understanding the headers and passing the mail onto the correct destination. Your mail composition program will connect to the running `sendmail` daemon that passes your mail on, either to an internal mailbox if the recipient is on your machine, or to the Internet if the destination address is offsite. When mail is received on a machine, mail will be dumped in your mailbox, and you will need a mail reading program to access it.

Sendmail mostly deals with external connections using the Simple Mail Transfer Protocol (SMTP) because it's generally talking to the Internet these days. Other ways of routing mail are supported but have dropped out of use as the Internet and its protocols have become ubiquitous. If you are running `sendmail` on your machine, you will see that it spends most of its time listening for incoming SMTP connections. When a remote machine connects to it, it starts a new process to handle that mail and resumes its listening watch. The new process deals with the SMTP connection and stores the incoming mail in a queue. It then decides where to send the mail. For local delivery, it places the mail in a mailbox file. For off-machine delivery, it uses SMTP to connect to the destination machine.

Sendmail is perfectly happy to take a message from anywhere on the Internet and relay it onto anywhere else. In fact, in the days before spam, it was common practice to permit anyone to relay through your machine. The sender could then run a lightweight client and use your more intelligent mail relay to process their mail. Open relay is deeply discouraged now, and relaying is inhibited by the Sendmail releases. You should think hard before turning it on.

Bulk e-mailers use open relays to hide the real origin of their mail. Typically these days, spam mail arrives from a bogus e-mail address and has passed through at least two machines. A single message is sent by the bulk e-mail program using a phone connection to an ISP's mail system. The mail message is often passed onto an open relay which very helpfully replicates it many times and passes it on to a huge batch of recipients.

Bulk e-mailers spend time looking for open relays. My machines receive at least one probe a day from people scanning the Net for some unsuspecting site to help them relay mail onwards. Incidentally, when Sendmail receives mail, it writes a header line into the mail, a `Received: from` line, and you can use these to trace where a particular piece of mail has been, assuming your mail reader program allows you to have access to the full set of headers. Beware that some spam mail programs pre-insert some `Received` lines to further confuse the source of their mail.

Sendmail is really designed to be connected to the Internet at all times, listening for incoming connections and delivering the mail to local mailboxes. If your machine is down or disconnected from the Internet, then remote mail systems trying to send you mail will sit there continually attempting to make a direct connection. My machines are connected to the Internet all the time but I still arrange my world so that my ISP will pick up any mail that's destined for one of my machines if my systems are unreachable.

In fact, most end users that use e-mail don't operate in this fully-connected environment. Most users dial into an ISP for a short time, and would like to pick up their mail using an explicit command at a time when they are connected. They also want to send any pending outgoing mail at connection time, too. It's important to get the mail off their machine, so they tend to upload all outgoing mail to their ISP and let their ISP's directly connected system handle all the delays inherent in sending mail. It's convenient to use SMTP for this outbound mail, because it interfaces well to the ISP's mail system.

However, their mailbox is not really on their machine, it's on a system run by their ISP. Systems have been developed to retrieve this remote mailbox and process it. The Post Office Protocol, POP, or since it's now into version 3, POP3, is one common method. Another is IMAP, which stands for Internet Messaging Access Protocol. One or both of these protocols will be installed in the mail reading program that the user invokes to obtain their messages.

There are also UNIX programs that can obtain mail from a remote mailbox and inject it into the local SMTP listener provided by the `sendmail` daemon and deliver the mail to a local mailbox. The one that springs to mind is `fetchmail`, which is open source and available from `http://www.tuxedo.org/~esr/fetchmail/index.html/`. If you use Sendmail on your machine in a partially connected environment, go to `www.sendmail.org/other-sendmail-links.html/` for advice on configuring Sendmail for outbound mail.

The Mail Trail

It's a common misconception that handling e-mail should be easy. After all, all I want to do is send some mail to my friend and give them the ability to send mail back. How hard could this be? One problem is that any user sending mail can specify a range of addresses and will expect the mail to be split into separate messages, some of which may end up on the local machine in a mailbox, or some of which may be sent out over the Internet to a set of distinct machines.

When sending mail on a machine that runs Sendmail, your composition program can call the `sendmail` program directly to inject mail into its queuing system. The disadvantage of this approach is that it ties your mail-sending program to a particular mail processing system. Alternatively, the program can make an SMTP connection to the local `sendmail` daemon, assuming your mail composition program can speak SMTP.

If your mail composition program *can* speak SMTP, then why doesn't it simply talk directly to remote machines over the Internet? The answer is that Sendmail provides a bunch of message processing facilities, like the ability to send messages to multiple recipients and the ability to keep trying to send the mail when the remote machine is temporarily unavailable. It's better not to burden the mail sending program with these abilities. There has to be intelligence somewhere to handle the mail, and it's easier to have that placed in one program on your system.

On a machine running `sendmail`, create a message using your mail composition program, tell the program to send it, and it ends up in the capable hands of the `sendmail` daemon. What happens next? Well, on a site that contains several machines, it's common to want to lie about where the mail originated. I want to tell you that my e-mail address at my company is `me@somecompany.com`. When I send mail from my *computer* named `solaris.somecompany.com`, I want the mail to be sent as if it came from the e-mail address that I told you and not from `me@solaris.somecompany.com`. Tomorrow I may use a different machine or my original machine might be junked and I still want to receive mail somewhere. Transforming the source e-mail address from a machine name to a local domain name is usually called "masquerading."

Masquerading can lead to a set of interesting problems with mail intended to be delivered locally to my colleagues. They'll be working on a range of separate machines and if all the machines are lying about where mail is originated, then some system is needed to work out where the mailboxes for all the local users are kept. One solution for internal connections is to pass outbound mail from all the machines in the company through a mail concentrator system (often called a "smart host"), and let that host deal with all routing problems, whether they're internal or external. All the machines in the company simply take mail from users and throw it at the smart host. Doing this fits in with the idea that you have a firewall that connects to the Internet–and pass all connections through it. It's also convenient for management purposes; you need to maintain only one more complex system. Actually, Solaris 8 is distributed with two possible configuration files for Sendmail: one for a smart host, one for a slave.

At some point, the mail I've sent will reach a machine that can make decisions about onward routing onto the Internet. The machine will have to be told that it can relay mail from me, which is normally done by enabling the source domain name or IP address range. It now decides what needs to be done to send the mail. There are various active headers in the mail message that are used to contain addresses: `To:`, `Cc:` (carbon copy), and the blind carbon copy version `Bcc:`.

Each of these header fields can contain lists of valid addresses that need to be checked and a decision taken whether the mail is to be delivered locally or sent out over the Internet. If the mail is to be sent out over the Internet, then the domain name part of the address (the bit after the "@") is looked up in the Domain Name Service (DNS). The DNS has a special record for mail hosts, called the MX record. Each MX record tells the sending mail system that when mail is to be sent to a specific domain name, the mail system should make an SMTP connection to a particular machine. It's also possible to specify a range of machines to be contacted for that domain, and give a priority should one or other be unavailable. So the DNS information for `yourcompany.com` might be:

```
yourcompany.com MX 10 mail.yourcompany.com
        MX 50 mail.yourisp.com
```

The entry says that if I want to send mail to you at `yourcompany.com`, then I must first try to send to the machine `mail.yourcompany.com`. This entry assumes that this machine has an IP address somewhere else in the DNS. If a connection to that machine fails, then an attempt is made to send to `mail.yourisp.com`, and if that fails the program must wait. The more usual case is that the mail is sent to your ISP because your machine is unavailable. At some point when your mail host is available again, all the mail that's been stored with your ISP will be sent to your machine. We assume that `mail.yourisp.com` is happy to relay mail into `mail.yourcompany.com`.

If `sendmail` cannot find the domain name in the DNS with an appropriate MX record, it will attempt to look up the name as if it were the name of a computer. It may find an IP address and it will send the mail to that machine. It should not really do this, but is done on the grounds that it gets the mail through even if the DNS is poorly configured, which is not unusual. This means I could send to `you@mail.yourcompany.com` and things will still work. You may or may not regard this as a desirable feature.

Having found somewhere to send the mail, my mail machine reaches out to yours, connects using SMTP and sends the mail. SMTP is a text based protocol and there are limitations on what can be transmitted. It you attempt to send mail that contains long lines, like the stored data from some word processing programs, then you may find that the mail is not received properly. In general, if you are sending something that is not a set of relatively short text lines, then you need to use an encoding mechanism such as one supported by MIME to ensure that the underlying protocol can handle data transmission properly.

We've got my mail message onto your mail machine, and assuming that your mail system is also running Sendmail, local delivery will be attempted. There are various levels of complexity supported here. The simplest case is the mail is delivered to a mailbox file on the machine. Before that is found, the local name is looked up in the `aliases` system, so a local name can be expanded to a list of names, a file where the mail is dumped, or a program that can be run when the mail is received. Any combination of these options can appear as the target for an alias.

The main alias database is a text file, and usually lives on `/etc/aliases` or `/etc/mail/aliases`. There should be an existing alias file on your machine that can handle common aliases like `postmaster` or `webmaster`. Whenever this file is changed, you need to run the `newaliases` command that rebuilds the DBM database file the `sendmail` daemon itself uses to look up aliases. Actually, `newaliases` is really a link to the `sendmail` program.

The format of the text alias file is simple. Comments can be introduced by placing a "#" at the start of a line, and blank lines are ignored. The remaining lines contain colon separated fields, for example:

```
name: replace
namelist: name1, name2, name3@other.com
namefile: :include:/etc/mail/namefile
toafile: /usr/somewhere/mailfilestore
toaprogram: |/usr/bin/program
```

The address `name` is found as `replace` and the mail will be delivered to the mailbox of the `replace` user. The address `namelist` will be expanded to the full list of names and the mail then delivered onwards. Note that the list can contain off-machine addresses. The `namefile` syntax allows you to place a list of aliases, one per line, in a file, `namefile` and then the file contents will be expanded to that full list. A copy of the message will be sent to all the people whose e-mail addresses are in the file. This type of expansion is convenient for mailing lists, you can simply alter the target file without having to rebuild the main `aliases` file.

The `toafile` alias line form allows the mail to be written to an explicit filename somewhere on the system, which is useful if you want to log all mail for a specific user to a file, perhaps for safety or because you need to be "Big Brother." Finally, the `toaprogram` alias allows `sendmail` to run a program rather than deliver mail to a mailbox, and the source mail message is placed on the standard input of the program. This mechanism can be used to employ spam filters, perform some processing based on the contents of the e-mail, or provide an automatic mail bouncer.

You should note that none of these alias expansions are reflected in the mail message itself. The addresses on the right-hand side of the `aliases` file are used by the delivery mechanism to re-route the mail. For off machine routes, an SMTP connection will be made to the destination machine that will be expected to receive and process mail based solely on the address given in the SMTP connection. The destination mailbox may not appear in the `To:` or `Cc:` lines.

If the mail is to be delivered to a local mailbox on a UNIX machine, the name of the mailbox is most often the UNIX login name of the user. Before local delivery is attempted, a file `.forward` on the user's home directory is examined and can be used to divert the mail to some other account, or pass it into a program. The `.forward` file contains a line similar to the `aliases` file. This feature is used by the `vacation` program. When called initially, it sets up your `.forward` file so that inbound mail is stored as usual, but it also includes a call to the vacation program that will send a friendly bounce message when mail is received. It also remembers who has talked to it, and will only send one piece of "I am not here" mail per caller. The `.forward` file can be a security hole, and `sendmail` will not use it unless it is owned by the user and is set so others cannot write to it.

I've missed a couple of "systemy" steps in the discussion above. First, to make the decision about local delivery, it's possible to configure Sendmail to accept mail for a large number of different domains that may be hosted on the machine. I use this quite a bit on my Web server. We are hosting around 90 domains and the `sendmail` on the machine will accept mail for each of those domains. It performs a local delivery for them, usually involving relaying of the mail to some mailbox on the Internet. There is also a mechanism (the `virtualuser` table), which allows intelligent forwarding so, for instance, inbound mail to `auser@otherdom.com` can be relayed to `auser@realdom.com`, where `realdom.com` is the real domain name of the user's mailbox.

Also, it's possible to make recent Sendmail versions use a table of source e-mail addresses, domains and IP addresses to decide how it should react to the address of the person attempting to relay mail into the system (the `access` table). I currently deny access to my systems to over 100 domains and e-mail addresses on the grounds that someone from that address has sent me spam. I also use the list of dialups known to send spam as a filter to reject e-mail. This is a free service run from `mail-abuse.org`.

Logs

Finally, it's always useful to know that Sendmail writes a complex log file that can help you work out what is going on, or more accurately, what has happened. The program uses `syslog` to manage its logs so it's not easy to say where the log file will be on your system. If you are running Solaris, then you'll find the log file in `/var/log/syslog`. BSD systems have tended to move the log into `/var/log/maillog`.

The log file will contain a minimum of two entries for each piece of mail that has passed through the system. Each line is tagged with a Sendmail id number which is useful when trying to trace a particular piece of mail. The id number is written into the "Received" header that's prepended to the message when it is received. For example:

```
Received: from yours.com
    (mta7.yours.com [299.100.100.100])
    by local.com (8.9.3+Sun/8.9.3)
    with ESMTP id GAA29646
    for <person@local.com>; DATE
```

I've split the lines for printing. The `GAA29646` in the line is the id number and can be used as an argument to `grep` to find out when the mail passed through the system and what happened to it.

Further Reading

There are several books on Sendmail. The one on my bookshelf is published by O'Reilly & Associates Inc. and is *Sendmail*, 2nd Edition by Bryan Costales with Eric Allman (ISBN 1-56592-222-0). This is the Sendmail "bible" and is not really for the fainthearted.

Peter Collinson *runs his own UNIX consultancy, dedicated to earning enough money to allow him to pursue his own interests: doing whatever, whenever, wherever ... He writes, teaches, consults and programs using Solaris running on an UltraSPARC/10. Email:* `pc@cpg.com`.