Exim configuration at the University of Cambridge

Tony Finch <fanf2@cam.ac.uk> <dot@dotat.at>

University of Cambridge Computing Service

ABSTRACT

Exim was originally written nearly 10 years ago for use on the email servers at the University of Cambridge, and our requirements still have significant influence on Exim's development. Our central email systems provide a number of message transport services, including a general-purpose smarthost, an authenticated message submission service usable by roaming users, and the University's main incoming MX.

This talk will describe some notable features of our Exim configuration, including: call-out address verification; basic anti-spam checks; TLS and AUTH; virtual services; effective use of macros and table lookups; stunt routers; and kamikaze string expansions. This will be explained in the context of a real live service handling over 400 000 messages each day for over 34 000 users.

1. Introduction

The bulk of this paper is a description of the Exim configuration on the central email relay run by the University of Cambridge Computing Service, which is known as ppswitch. This is intended to provide examples of some more advanced Exim features than provided by the default configuration file that comes with Exim. These examples are based on real world experience of running a reasonably large and featureful email system.

Section 2 describes the history and context in which ppswitch operates, together with an overview of the complete central email systems. This should explain the reasons for the various configuration details in the following sections.

The next two sections describe the technical details of the configuration. Section 3 describes the Access Control Lists, which implement ppswitch's various service personalities for incoming and outgoing email, and some of our anti-spam protections. Section 4 describes the email address routeing, which implements features such as virtual domains and mailing lists, and special handling for the Hermes message store and departmental email servers.

Section 5 describes some bits and pieces that do not currently appear in our configuration but which may be of interest. Section 6 describes the style rules used to lay out the configuration file in a readable manner. These rules were developed to make it easier to maintain a large and complicated configuration.

Section 7 outlines some planned and potential work for the future, followed by some concluding remarks in section 8. Finally, the appendices in section 9 contain the text of the configuration files which are described in this paper.

2. Background and context

This section explains the whys and wherefores of ppswitch's configuration file: why it provides the facilities it does, and why its architecture is what it is. Our paper about Hermes[1] provides a slightly different perspective, which might also be of interest.

2.1. History

PPswitch was introduced in 1992[2] to provide gatewaying between Internet email and JANET ''Grey Book'' email. It is named ppswitch after the software it originally ran, PP[3]. At that time most of the University's users used Grey Book email on the old IBM mainframe system ''Phoenix'', which had been the basis of

This paper was presented at the First International Exim Conference and Tutorial on the 23rd & 24th February 2005 in Birmingham and is published on the Web at
`http://www.cus.cam.ac.uk/~fanf2/hermes/doc/talks/2005-02-eximconf/`

the Computing Service for over 20 years. Since 1990 there was also the Central Unix Service (CUS) which provided Internet email. In 1994-1995, the need for Grey Book support declined rapidly. Phoenix was due to close on 1st September 1995[4]; its email-only users were being moved to the new Hermes email service[5] and others were being moved to CUS. UKERNA decided that Grey Book email was officially obsolete at the start of 1995, and by the end of the year ppswitch was handling Internet email only.

Some other early facilities implemented by ppswitch were also influenced by the closure of Phoenix. A mailing list system[6] was implemented to take over from Phoenix's lists. The Computing Service's departmental email domain *@ucs.cam.ac.uk*[7] and the University-wide *@cam.ac.uk* email domain[8] allowed people to be contacted using a consistent address regardless of the system they used for email, which became a consideration when Phoenix was no longer effectively the only computer.

The facilities described so far all predate Exim. Philip Hazel started writing Exim in 1995, and the Computing Service started running it on the major central systems in 1996, first CUS in June and then Hermes in August. Before then these systems had run Smail[9]. One of the initial motivations was to improve upon Smail's access controls, which was becoming an area of concern as spam became common. Another new feature that Exim provided was support for multiple mail domains, which was required by ppswitch. It also started running Exim in August 1996.

After this point, ppswitch acquired more facilities: The ''Managed Mail Domain'' system[10] provides virtual email domains, each consisting of an aliases file which can be edited using a friendly user interface. The mailing list system was upgraded to support restricted and moderated lists[11]. More recently, a full-content anti-spam and anti-virus email scanner was added to ppswitch to provide significantly better central protection against junk email[12].

As well as using the central email facilities, Colleges and departments of the University can run their own email systems. This is particularly useful for institutions who want features that aren't provided by the Computing Service, such as calendaring and other groupware functions. Although these email systems can communicate directly with the public Internet, many of them opt to send and receive email via ppswitch, to provide protection from hackers; this has become more popular since the improvement of the anti-spam and anti-virus protection. Departments can also use ppswitch as a backup MX.

Although ppswitch and Hermes have – until recently – been separate systems, ppswitch has never had a user interface of its own, instead relying on the Hermes menu system for management of mailing lists and virtual domains. This semi-detached arrangement changed with the recent Hermes upgrade[1], and now Hermes and ppswitch are effectively aspects of the same system. A POP and IMAP proxy running on ppswitch makes the multiple Hermes message store machines appear like a single system; more relevant to this paper, however, is that Exim on ppswitch also handles email delivery to the Hermes message store, and provides the message submission service for Hermes users.

2.2. Architecture

Cambridge University's central email systems consist of two large clusters and a few other miscellaneous machines. The clusters are ppswitch and the Cyrus message store; each cluster consists of machines of essentially identical configuration, differing only in the data that they store. The other machines include the central MUA service (comprising a mirrored pair of machines), a backup server with lots of online and offline mass storage, and an administrative support server which is used to manage the configuration of the other machines and control the propagation of changes around the system.

Two services are provided by this setup: Hermes, which is the email service for individual users; and ppswitch, which provides email relay services. The names we use are somewhat ambiguous, so deserve an attempt at clarification. Hermes is the name of the service as a whole; *hermes.cam.ac.uk* provides access to the terminal-based menu system and Pine; and *hermes-1* and *hermes-2* are the machines the central MUAs run on. PPswitch is both the name of the email relay service and the cluster it runs on (which also supports the redirecting proxy for *pop.hermes.cam.ac.uk* and *imap.hermes.cam.ac.uk*). PPswitch's personality – its rules for accepting email – depends on the IP address on which it is contacted (as explained in more detail below) either *ppsw.cam.ac.uk*[1], *mx.cam.ac.uk*, or *smtp.hermes.cam.ac.uk*. PPswitch's individual machines are named *ppsw-0* to *ppsw-9*[2].

Hermes has over 34 000 users, about 27 000 of whom log in at least once a week. They are provided with a basic quota of 250MB (upgradable to 1GB) for storing email, which their MUA software can access via POP or IMAP. Hermes also provides central MUA facilities, so that users can read their email via the WWW, or

[1] ''*ppsw*'' can be pronounced as it is spelt or like ''ppswitch''.

[2] Not all of these names may be in use at once: at the moment we have 7 ppswitch machines active.

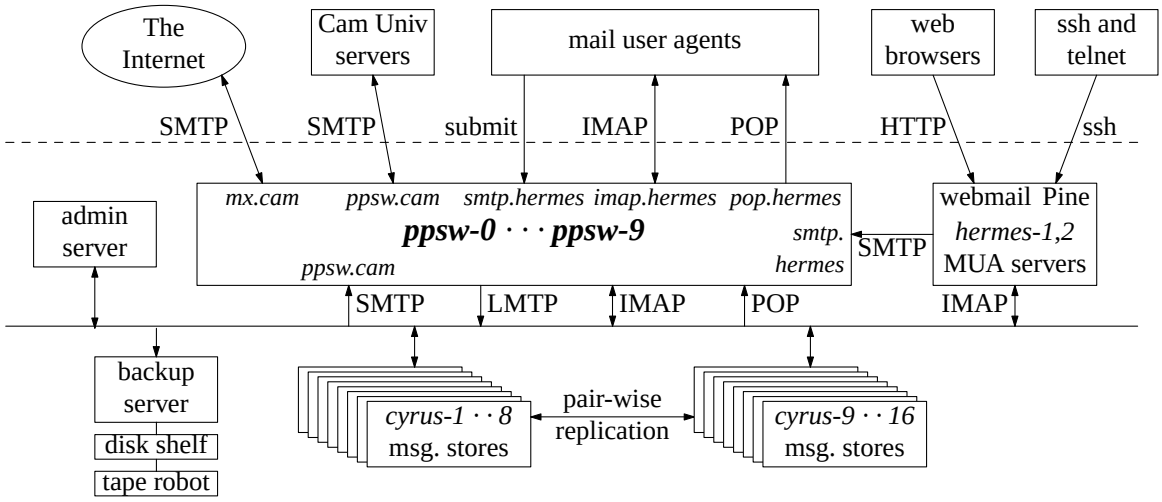

Fig. 1: Cambridge University central email systems architecture

using Pine over `ssh`. Hermes now stores over 1TB of email. Because of the overheads of the performance and reliability features (indexes, replication, retention of deleted data) this actually occupies about 4TB of space on disk, out of the total of over 10TB usable in the whole system.

As well as handling all the email for Hermes, ppswitch handles email for over 160 virtual domains, over 4000 mailing lists, and over 70 departmental email systems. The volume during term is up to 400 000 messages each day, averaging 8 messages per second 9 to 5. This does not include the messages that are rejected at SMTP time, which is presently (Jan 2005) about 700 000 per day but has been as high as 1 300 000 per day (Nov 2004).

The configuration on ppswitch is divided into two differently-managed parts: the mostly-static configuration files and tables under `/opt/exim` controlled by the system administrators (the main topic of this paper); and the more fluid tables under `/opt/dist` for virtual domains, mailing lists, passwords, etc., which users can edit using the Hermes menu system and other user interfaces. Both of these are distributed from the admin server, the former as required by the admins (after being tested and committed to the configuration management system), and the latter automatically every hour or so.

I mentioned above that ppswitch is a cluster of identical machines, which share all of the work equally. There are two reasons for this: The present reason is that it makes system administration simpler because it reduces the number of different configurations that we have to manage. Another view is historical: ppswitch has always been a uniform cluster, and there has not been any reason to change this. This point is worth noting because it is common for sites to split their email relaying machines into incoming and outgoing clusters, or to separate quickly deliverable email from delayed email, etc. This might be to enable the configuration of different email acceptance policies, or for reasons of performance management. I will explain how Exim's extremely flexible ACL system allows ppswitch to present multiple personalities, and how to thoroughly verify addresses at SMTP time in order to avoid clogging queues with undeliverable junk.

3. Multiple personalities

In the mid 1990's, spammers started seriously abusing email servers to hide the source of spam and to provide bandwidth amplification. This led to a change in common practice: email servers no longer relay email for anyone, but are locked down so that they distinguish between incoming email (destined to a local email domain) and outgoing email (coming from a local network). The effect of this is that the email server has two personalities: its rules for accepting email differ depending on where you are sending from. If you are on the public Internet it has an MX personality and if you are on a local private network it has a smarthost personality.

It's common on small systems for these two personalities to exist on the same IP address, and this is what the default Exim configuration implements. Historically, ppswitch has also worked in this way: the host name *ppsw.cam.ac.uk* was used both as a smart host[3] for outgoing email from the University and as the MX host for the University's email domains. At that time, Hermes email – both incoming and outgoing – was handled by a separate system. This arrangement became untenable with the introduction of the new Hermes system: the need

[3] Of course the logical host name *ppsw.cam.ac.uk* has several IP addresses, so refers to several physical machines.

to handle its email on ppswitch implied the addition of a third personality. At the same time the advantages of logically separating incoming and outgoing email had become clear: it would provide architectural flexibility and allow a wider choice of anti-spam/anti-virus technology on the MX service, without interfering with the smarthost service. Therefore we took the opportunity to rename our MX host to *mx.cam.ac.uk*[4]; *ppsw.cam.ac.uk* was retained as the smarthost name because it would be much more difficult to inform everyone who needed to know about the change – as well as for sentimental reasons.

Alternative approaches: Another way of implementing multiple personalities is to choose the personality based on the port number instead of the IP address. The message submission personality runs on ports 465 and 587, and the traditional hybrid MX/smarthost runs on port 25. This may be a simpler way to add a modern secure authenticated message submission service to an existing system. However it loses some of the advantages of providing a clear distinction between incoming and outgoing email.

3.1. Differences between the personalities

type & name *feature*	MX *mx.cam.ac.uk*	smarthost *ppsw.cam.ac.uk*	submission *smtp.hermes.cam.ac.uk*
accept from	anywhere	CUDN only	CUDN or AUTH
accept to	`+our_domains`	anywhere	anywhere
verification	sender + recipient	sender only	sender only
TLS	no	no	yes
ports	25	25	25, 465, 587
fixed address	no	yes	no
fix-ups	none	Date, Message-ID	submission mode
anti-junk	spam + virus	virus	virus

Fig. 2: Comparison of ppswitch personalities

The table in fig. 2 summarizes the differences between ppswitch's three personalities. These are the MX personality on *mx.cam.ac.uk*, which handles incoming email; the smarthost personality on *ppsw.cam.ac.uk*, which handles outgoing email from other email servers, web servers, and miscellaneous machines; and the message submission personality on *smtp.hermes.cam.ac.uk*, which handles outgoing email from Hermes users' MUAs.

As explained in the introduction to this section, the MX personality accepts email from anywhere on the Internet addressed to any of our email domains. (The `+our_domains` notation comes from the name of a domain list in ppswitch's Exim configuration.) Since it is the only way for email to come into the University[5], it is the only personality that implements anti-spam checks. Viruses, on the other hand, are filtered by all personalities so that they can be detected and stopped if they manage to get past our defences.

The smarthost and submission personalities on *ppsw.cam.ac.uk* and *smtp.hermes.cam.ac.uk* are quite similar at a first glance. A primary reason for separating them is that *smtp.hermes.cam.ac.uk* needs to support TLS, so must offer a certificate to clients that matches its name; if a client were to address it as *ppsw.cam.ac.uk* there would be a mismatch that would be likely to cause problems. A more subtle difference is that *smtp.hermes.cam.ac.uk* implements full submission-mode fix-ups to the message header, ensuring that the `Sender:` `Date:` and `Message-ID:` fields are present and correct; *ppsw.cam.ac.uk*, on the other hand, does not have enough information to fix the `Sender:` field so it leaves it alone, in order to avoid making gratuitous changes to outgoing email from other email servers.

The submission personality uses some of SMTP's more advanced features in order to support roaming users. As well as being authorized to send email by virtue of being on the Cambridge University Data Network (CUDN) – the same smarthost logic as *ppsw.cam.ac.uk* – Hermes users can also authenticate themselves in order to send email from foreign networks. This is done using ESMTP AUTH and the SASL PLAIN or LOGIN mechanisms [13,14,15,16]. These mechanisms transmit the user's password in the clear, so we use TLS to encrypt the SMTP session. Two modes of TLS are provided. The standard way is to start the SMTP conversation in the clear as usual, then use the STARTTLS command[17] to turn on encryption. This is offered on ports

[4] This was not the best choice, despite being elegantly simple: it has a built-in assumption that there will not be any need for a secondary MX, which makes the use of certain anti-spam techniques harder. It would have been better to choose *mx0.mail.cam.ac.uk* and at the same time rename the rest of our machines into their own subdomain separate from the general Computing Service internal domain.

[5] To a first approximation; see the implementation sub-section for an explanation of why this is not always true.

25 and 587, the traditional SMTP port and the newer message submission port[18]. The older deprecated way is to start TLS immediately after connection, before starting SMTP. This is offered on port 465 (the old SMTPS port) to support software that does not implement the STARTTLS specification correctly, most notably Microsoft Outlook and Outlook Express. We recommend to our roaming users that they do not configure their software to user port 25, because it is more likely to be blocked than the alternatives[19].

The final point to note in this section is the logic for email address verification. In general we try to check the email addresses in the message envelope as thoroughly as possible[6]. This reduces the quantity of resources wasted by junk email, and makes it more likely that errors will be noticed and fixed before they cause email to be lost. However the SMTP error handling of common MUAs is very bad, such that they report confusing error messages to the user or fail to send the message to some of its recipients, etc. Therefore for outgoing email we only check the `MAIL FROM` address at SMTP time; if any of the `RCPT TO` addresses are invalid, a bounce message is returned instead of an immediate error. We do not do this for the MX personality, since any bounce message sent in response to a forged message would be collateral spam.

3.2. Implementation of multiple personalities

Initially I attempted to implement all the personalities as part of the same ACL, in a similar style to Exim's default configuration. However, this resulted in a configuration that was hard to understand, difficult to maintain, and buggy – it didn't implement the policies I wanted. Individual ACL clauses became more complicated, because they had to check both the current personality and the actual test of interest. The ACL as a whole became more complicated, because various unrelated clauses were interleaved. I decided that it's much simpler to have a separate ACL for each personality. The logic for a given personality becomes clear: it is not cluttered by extraneous tests. This directness of purpose makes it concise: all the principal ACLs in ppswitch's configuration fit on a single side of paper. This also makes it easier to check that they are correct.

The fundamental idea is for each ppswitch machine to have multiple IP addresses, one for each personality. Then Exim is configured to alter its behaviour based on the server IP address that the client connected to. This is available in the expansion variable `$interface_address`. We don't actually use this variable directly: instead it is looked up in a table containing parameters associated with various addresses. (See line 23 of the configuration in Appendix 9.1, and the IP address parameters table in Appendix 9.3.) This allows us to configure Exim in terms of features, somewhat independent of the low-level details of a particular personality. As well as server IP addresses, the table is also used to hold parameters associated with various client IP addresses, specifically other machines in the central email cluster for which we activate special features. (See line 24.) Macro definitions are used to abbreviate these lookups throughout the configuration, most prominently `PARAM`.

The result of the table lookup is a number of parameters, and we use `${extract` to separate them out of `PARAM`. One of the more important personality parameters is the personality's hostname; a macro `NAME` is defined to hold this value (see line 28). This macro also provides an example of how the configuration uses default values to ensure that it remains moderately sane (e.g. not becoming an open relay) in case of misconfiguration. A more elaborate macro is `FULL_HOSTINFO` (see line 30) which includes the individual machine's primary hostname, the personality name, and the server IP address and port number. It is used in the `smtp_banner` and `received_header_text` settings (see lines 248 and 287 respectively) to provide details of which machine and personality the client connected to, for example:

```
ppsw-6.csi.cam.ac.uk (smtp.hermes.cam.ac.uk [131.111.8.156]:25)
```

Most aspects of the personality are determined by the set of ACLs that it uses. These are configured using the `acl_smtp_*` options. The value of these options is expanded, so we can use a personality parameter lookup to select the ACLs we are using – see line 212[7]. Again, a default value is used to choose a reasonably sane ACL for IP addresses (such as 127.0.0.1) which don't have their own personalities. There are also some personality-related settings outside the ACLs, which will be covered in the following subsections.

3.3. The smarthost personality

The preceding subsections might have given the impression that this is the simplest personality. However there are a couple of tricky features which make it more interesting. All of the smarthost personality is

[6] Checking email addresses in the message header is rather problematic because of the high proportion of legitimate but malformed email. We leave that job to SpamAssassin, not Exim.

[7] At the moment we implement a hook for `acl_smtp_data` on like 217, but we do not currently use it. In the past it was used for submission mode fix-ups before these were built in to Exim. In the future they will be used for content scanning.

implemented in its ACLs, starting at line 365. Most of these perform the default accept action, except for the `RCPT` ACL which, as usual with Exim, is where all the checks are performed.

The first clause ensures that port 25 is being used; this should probably be done in the connect ACL, but it makes little difference in practice. The second clause ensures that people who are having configuration or interoperability problems can get through to our support staff. The third clause is a special case for other machines in the central email cluster, in particular the Cyrus message store machines. Hermes users are allowed to set up Sieve filters[20] which are run by the Cyrus `lmtpd`; these can cause email to be forwarded to another destination, and since the Cyrus machines do not have globally-routeable IP addresses, these forwarded messages must go out via ppswitch. Since ppswitch has already done any necessary address verification when the message first passed through, it can skip the checks on outgoing forwarded messages to save work.

The next two clauses are a wart. In an ideal world they could be replaced by:

```
deny
  message        = No SMTP service for unauthorized users
! hosts          = +relay_hosts
```

However, for historical reasons *ppsw.cam.ac.uk* is an email domain as well as the hostname of the smarthost service – it's the domain used for email generated on ppswitch (see line 284). This means that *ppsw.cam.ac.uk* should have an MX record; however if the MX record points to *mx.cam.ac.uk* then some systems will irritatingly try to use *mx.cam.ac.uk* as their smarthost. Therefore if *ppsw.cam.ac.uk* has an MX record it must point to *ppsw.cam.ac.uk*. This has a further implication that the smarthost personality must have a little bit of MX functionality for email to addresses *@ppsw.cam.ac.uk* from the public Internet. Thus the fourth clause checks outgoing email (to addresses other than *@ppsw.cam.ac.uk*) is only from `+relay_hosts`, and the fifth clause ensures that other hosts can only send email to valid addresses *@ppsw.cam.ac.uk*. The `+relay_hosts` are defined at line 114 to be all the machines on the Cambridge University Data Network (CUDN) – see Appendix 9.6. It has essentially the same meaning as the `+relay_from_hosts` setting in the default configuration file that comes with Exim.

The next clause tells Exim to implement the appropriate message header fix-ups, then finally we perform address verification. For normal messages we perform our usual sender address verification, which is explained in more detail below. For bounce messages we check the recipient address instead, because bounces don't have a sender address to which we can send delivery failure reports (i.e. bounced bounces). This bounce recipient verification is stricter than our sender address verification, because it does not include the exemption checks that we implement for sender verification. The aim is to keep failed bounces closer to the systems that generated them, since they are more likely to be noticed by someone who can do something appropriate with them. Note that it uses the same time-out settings as sender verification, via the `CALLTIME` macro.

One feature of this personality, which is not evident from the configuration alone, is that in addition to its name it has a rôle address, 131.111.8.129. This is listed near the start of the `addrparams` table in Appendix 9.3. Unlike the other IP addresses of *ppsw.cam.ac.uk* it is not tied to a particular *ppsw-N* machine, but instead is an extra virtual IP address on one of the currently-active machines. The reason for this is that there are a number of departmental email servers and other SMTP clients which cannot be configured with a hostname for their smarthost, or which run for a long time and only perform DNS lookups once at startup. 131.111.8.129 is guaranteed to be available regardless of which ppswitch machines are currently in use, which gives us more freedom to reconfigure things without having to co-ordinate with departmental Computer Officers.

3.4. The message submission personality

The ACL for this personality is very similar to the smarthost personality's. The differences are that: it isn't restricted to port 25; messages are accepted both from `+relay_hosts` and authenticated users; the wart is missing; and full message submission fix-ups are performed. Note that the message submission domain is extracted from the personality parameters, because it is different from `$qualify_domain`.

Most of the interesting features of this personality are implemented outside the ACLs. The simplest example is the smaller message size limit imposed on Hermes users – see line 192. Slightly more complicated is the lobotomizing of Exim's SMTP implementation; this is partly done via the connect ACL at line 446, and partly via a global option at line 253 (where the reason for it is explained; see also line 446). Note the use of `${if match{PARAM}{acl=submit} {:} {*} }` as a shorter and simpler replacement for the pedantic `${extract {acl}{PARAM} {${if eq{submit}{$value} {:} {*} }} {*} }`.

The most important non-ACL settings for message submission are the TLS setting starting at line 223. These advertise TLS only when there is a certificate available, in a file with the same name as the personality's hostname. Also, port 465 is configured to use TLS-on-connect instead of STARTTLS. These settings are

important because they have the side-effect of enabling authentication – see line 699.

Most of our authentication configuration is very basic: the `LOGIN` and `PLAIN` authenticators are almost straight out of the Exim documentation. However they are followed by a rather unusual `EXTERNAL` authenticator. The SASL EXTERNAL mechanism exists to pull up authentication information from a lower layer (such as TLS) as a kind of formalized layering violation. We're using it for a slightly different purpose: to communicate authentication information from the central MUA service (webmail etc.) to ppswitch. The reason for this it to simplify automated per-user signing of email by allowing it to be implemented in only one place: on ppswitch[8].

The Exim configuration for the MUA service can be found in Appendix 9.2. It is mostly quite straightforward. One tricky part is the fact that Exim has to trust the user that the webmail server runs as (`prayer`) to act faithfully on behalf of users: as well as the `trusted_users` setting it is also given special handling when we are working out which user to authenticate as on line 92. The other niggle is that there's a mismatch between SMTP authentication (which authenticates the whole client connection) and what we want to do (which is authenticate individual messages). Therefore we must transmit only one message per connection to ppswitch so that there's a one-to-one match between authentication and messages – see line 114.

The alert reader may be wondering why email from the MUA service is sent via *smtp.hermes.cam.ac.uk*, whereas email from the Cyrus message store is sent via *ppsw.cam.ac.uk*. The reason is that the former comprises newly submitted messages, so they are sent via the message submission service, but the latter comprise existing messages that are being relayed. Relaying is unauthenticated and does not involve header fix-ups.

3.5. The MX personality

At a first glance this appears to be much more complicated than the other two personalities. However most of the complexity is due to anti-spam measures which can be ignored for the moment. The section we are concerned with in this sub-section is the RCPT ACL starting at line 567. This starts off with the same two clauses as the smarthost personality.

The third clause prevents the MX personality from relaying all email presented to it. Note in particular that this applies to all hosts, whether on the public Internet or on the University network. The reason for not allowing local machines to relay via the MX personality is to thwart a spammer tactic. Although our block on port 25 prevents spammers from simple use of zombies and open HTTP or SOCKS proxies, they can still use a multi-hop technique. They connect to the proxy and use it to send spam to a smarthost which then relays it to the Internet. They often use MX records to discover the network's smarthost: in fact, Spamhaus recently reported[21] that a new version of the widely-used Send-Safe spamming software has an easy-to-use feature for sending spam via the MXes of the hijacked zombie PCs. We observed this technique being used with a machine on our network in September 2004, which led to us locking down *mx.cam.ac.uk* for local machines as well as remote ones. We hope that the combination of strict no-relaying rules on our MX and somewhat obscure smarthost names will protect us from this exploit in future.

The fourth clause performs various anti-spam checks, by calling an auxiliary ACL. This avoids clutter in the main logic of this ACL, and makes it easy to turn off spam checks for University hosts. The anti-spam checks are described below.

The next three clauses perform envelope address verification. First we just check the domain of the sender address, because this is a cheap way of rejecting obvious forgeries. Next we check the recipient address, with a call-forward. If the recipient is on a departmental server we want to find out as soon as possible if the address is valid in order to minimise the amount of undeliverable email we have to handle. The `use_sender` option allows departmental servers to implement their own sender address blacklists and to distinguish between normal messages and bounces. After recipient verification, we perform a full call-back sender verification. We delay this as long as possible in order to avoid bothering the sender's MX server for email we aren't going to accept anyway. The details of call-back verification are described in the next sub-section.

The final check is an overload protector. If email is arriving faster than MailScanner can handle it, the incoming queue will grow and email will be delayed. Therefore, if the queue is longer than a few minutes of scanning time, we defer the message. (The sender should immediately try another ppswitch system which is hopefully not also overloaded.) When the load is high we make some effort to avoid re-scanning the queue length and adding further to the load. We don't perform this check for the other personalities in order to avoid worrying users with a temporary problem.

[8] This is the subject of my paper for the UKUUG Winter Conference which immediately follows the Exim conference this year.

3.6. Call-back verification

All of the personalities' RCPT ACLs call an auxiliary ACL to do sender call-back verification. This starts at line 635 in the configuration file. The purpose of the added complexity is to avoid performing call-back verification when it is known that the result will be incorrect. There are two common reasons for legitimate email to have a sender address that appears to be invalid or which cannot be verified at all: the sender domain's MTA is either misconfigured or incompetently written, and rejects all bounce messages; or the email comes from a web server which is not running an MTA, and its sender address is of the form *httpd@host.name.of.web.server*. The general goal of the ACL is to detect totally invalid email without penalising legitimate email just because of technical incompetence.

Most of the work of maintaining an exemption list has already been done for us: the *rfc-ignorant.org* project keeps a list of domains which are known to reject all bounce messages (or Delivery Status Notifications, DSNs). Slightly unusually, we use this list as a whitelist (not a blacklist!) to exempt domains from sender call-back verification. We also periodically run a script to search our logs for domains that ought to be listed and submit them for inclusion in *dsn.rfc-ignorant.org*.

In addition to that we have a locally-maintained list of problem addresses and domains, for example email addresses that refer to web servers, domains which do not like call-back verification, and other problems which are not sufficiently clear-cut to be listed by *rfc-ignorant.org*. Our local list has less than 100 entries, which is orders of magnitude fewer than the thousands listed by *rfc-ignorant.org*. The somewhat complicated double lookup allows us to be very specific or general in our listings; see the `nocallout` table examples in Appendix 9.6.

Note that this auxiliary ACL does not do any verification of exempted domains. This means that callers must do `verify = sender` to verify the sender's domain, either immediately before as in the smarthost and submission ACLs, or before recipient verification as in the MX ACL. Also, this ACL could be improved somewhat by using the new `$sender_verify_failure` variable to dynamically detect those unlisted RFC-ignorant domains that immediately reject `MAIL FROM:<>`.

3.7. Anti-spam measures

This configuration uses four anti-spam techniques: address verification, which we have already covered; ratware HELO signature detection; DNS blacklists; and pump-and-dump detection. The latter is implemented in the MX personality's connect and HELO ACLs, and the remaining two are implemented in the auxiliary spam check ACL. Together these checks reject about 90% of spam.

The HELO checks start on line 659. These detect a number of SMTP protocol implementation bugs which are unique to spam software (''ratware''), and which are therefore very reliable anti-spam criteria. The first check detects bare IP addresses, such as `192.0.2.54`. Although SMTP allows clients to state an IP address instead of a domain name as the argument to HELO, they must do so in square brackets, like `[192.0.2.54]`. Then we check if the HELO argument is the same as the recipient's local part, for example, `HELO fanf2` / `MAIL FROM:<scott@optinbig.com>` / `RCPT TO:<fanf2@cam.ac.uk>` / `RCPT TO:<spqr1@cam.ac.uk>`. This particular type of ratware uses multiple RCPT commands for each message, so if our checks fail we remember this in the `$ACL_OKHELO` variable for future RCPT commands. Then we check for excessively long HELO commands, or which have a syntax error (double dots) which Exim doesn't detect by itself. These bugs are caused by ratware that randomly concatenates domain names together to create its HELO argument, which seems like a lot of work to make anti-spammers' jobs easier. Finally, we check if the ratware is stating one of our domain names instead of its own host name.

The pump-and-dump protection is based on Exim's SMTP protocol synchronization checks. ''Pump-and-dump'' refers to software which just pumps a load of SMTP commands at a server and ignores the responses. We make Exim's checks more effective by adding delays at various points in order to give it time to detect spewing ratware despite long network round-trip times. These delays are inserted before the SMTP banner by the connect ACL (see line 510), and before the response to HELO by the HELO ACL (see line 547). In addition to the delays, we attempt to confuse shoddy software which doesn't correctly handle multi-line SMTP responses by adding a message to the SMTP banner – see line 248. I'm not sure how effective this is, but it is at least amusing.

After messages have been accepted by Exim, they are processed by MailScanner[22] which passes them through ClamAV[23], McAfee[24], and SpamAssassin[25]. MailScanner works using two spool directories: an incoming queue in which Exim places new messages, and an outgoing queue from which deliveries are performed. It moves messages from one spool to the other after passing them through the scanning software packages. Two corresponding copies of Exim are run: the outgoing Exim daemon just starts queue runners `-q5m`;

the incoming daemon is run in queue-only mode `-odq` so that it doesn't try to deliver messages before MailScanner has handled them, and it uses a command-line macro `-DSPOOL=/spool/exim.in` to override the configuration file's spool directory setting at line 139.

MailScanner passes all email through the virus scanners, but only certain email is passed through SpamAssassin. The aim is to scan email that came from the public Internet, but that is not entirely straightforward since email may come into the University via a departmental email system before arriving at ppswitch. At the moment we maintain a list of systems within the University from which we will scan email, as well as from the Internet; in the future we plan to reduce this maintenance work by simplifying the logic: we will only scan email that comes in via *mx.cam.ac.uk*, based on the assumption that departments which receive email from the Internet will route email to the MX rather than via the smarthost *ppsw.cam.ac.uk*.

3.8. Received header text

The final personality-related part of the configuration is the setting of `received_header_text` at line 287. The purpose of this is to add extra information to the trace fields in the headers of messages handled by ppswitch. We use the `FULL_HOSTINFO` macro to include details of both the personality and the individual ppswitch host which received the message. We include details of the authentication mechanism and username, as well as the TLS information which Exim includes by default. Finally, we record the message's return path, which makes it easier to trace messages that were sent via mailing lists that change the return path. The return path is also used by SpamAssassin for some tests, and it isn't otherwise available if, like us, you run SpamAssassin before the Return-Path: header field is added at final delivery.

The Received: header is difficult to configure. A lot of care is required to stay within the syntax specified in section 4.4 of RFC 2821[13]. It is further complicated by the need to control the format of the header by careful placement of newlines in the string expansion. According to RFC 2822[26], each physical line in the header should be kept to less than 78 characters, though we don't quite manage this if the return-path or recipient addresses are long. The complicated series of nested string expansion items can easily become incomprehensible. Section 6 below describes my formatting rules which help to make it easier to follow.

4. Routers and transports

This section examines what we do with email after accepting it, that is, determining its destination and delivering it. Fig. 3 provides a simplified overview of this process. The architecture diagram in fig. 1 hides the routeing complexity that is internal to ppswitch: there are very few external SMTP[9] connections. Similarly, fig. 3 has few external connections: most of the diagram is concerned with virtual domains and mailing lists, which cause a message to be redirected to a new set of recipients and passed through the routeing process again. Whereas the ACLs do not have complicated table lookups – just `+relay_hosts` and `+our_domains` – the routers depend on a large number of tables maintained by the system administrators and by users. After describing these in the first subsection, I will explain the details of the routers and their associated transports.

4.1. Routeing tables

The tables used in routeing can be divided into two classes. Tables that control higher-level aspects of the configuration, such as which domains we handle, are used to construct a number of named lists starting at line 62. These named lists are used in router preconditions, to determine which domains are handled by which router. The high-level tables are small in number and managed by the system administrators. The lower-level details of routeing, such as the destination of each local part within a domain, are handled by table lookups configured within the routers. There is a very large number of these tables – several for each mailing list and virtual domain – and their management is generally delegated to users.

We have already seen the list of `+our_domains` (line 99), which is used by the ACLs as part of our relaying restrictions. This is the list of all domains handled by ppswitch, which is constructed from the list of domains handled on ppswitch itself, `+local_domains`, and the list of domains handled by departmental servers, `+relay_domains`. These have essentially the same meaning as `+local_domains` and `+relay_to_domains` in the default configuration that comes with Exim. I will explain the purpose of `+postmaster_domains` in the ''postmaster domains'' sub-section below.

There are three kinds of local domain: managed mail domains, special cases, and long form domains. A managed mail domain is a virtual email domain which simply consists of an aliases file that can be edited by the domain managers using the Hermes ssh/telnet menu system. The list of managed mail domains includes some special cases which have an aliases file like other managed mail domains; however the special case domains

[9] Including the SMTP-alikes, LMTP and message submission.

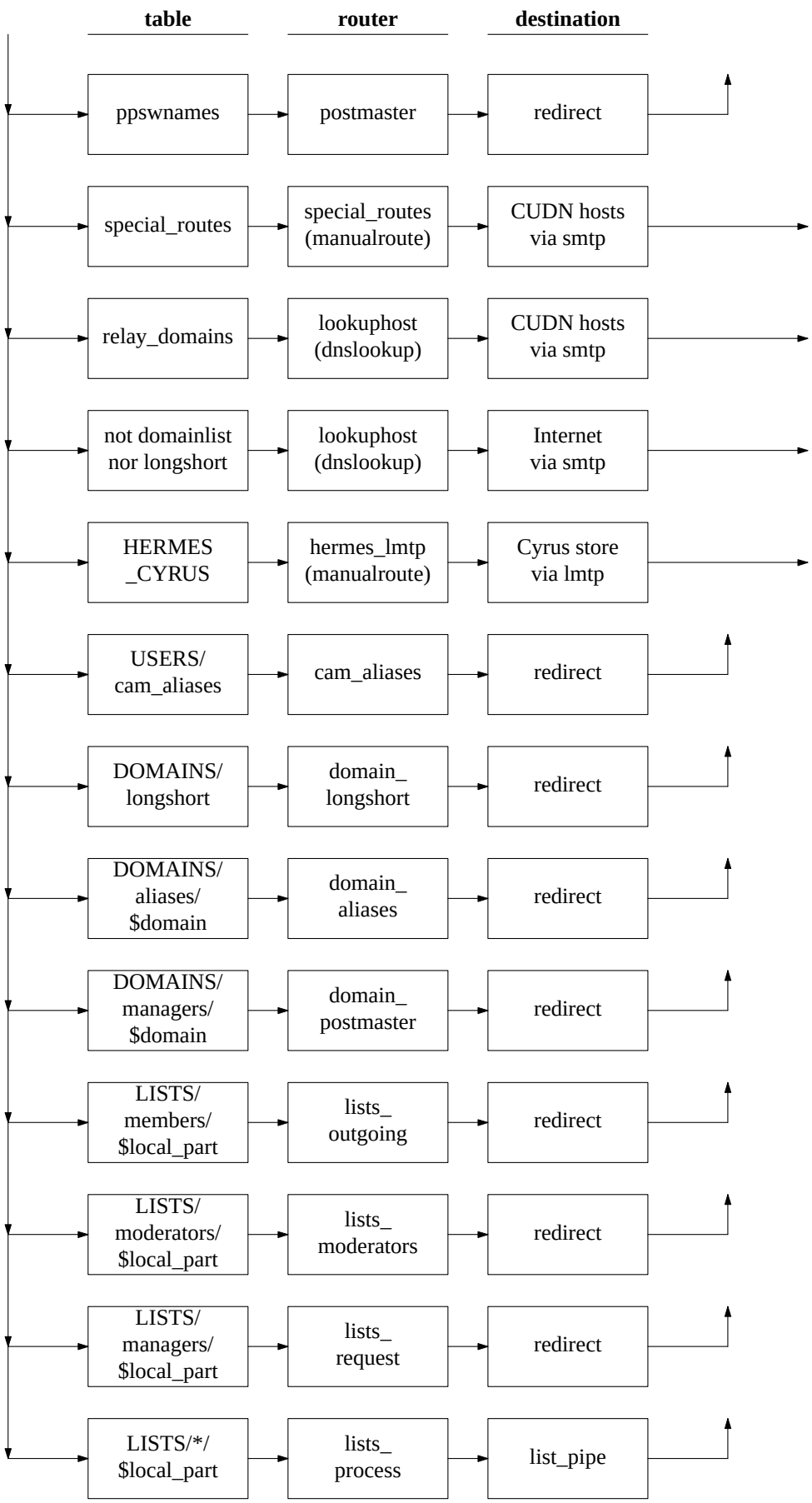

Fig. 3: Address routeing on ppswitch

have extra routers in the configuration, which implement special handling for large numbers of email addresses in addition to the aliases files. Long form domains are synonyms for existing domains, sort of domain-level aliases. See Appendix 9.6 for examples of entries from the `domainlist` and `longshort` tables.

Long form domains are a newer feature than you might expect. In the days of Grey Book email, the NRS (Name Registration Scheme) was JANET's backwards counterpart to the DNS. It generally contained two forms of each name, a short form like *uk.ac.cam.phx* and a long form like *uk.ac.cambridge.phoenix*. Cambridge quickly dispensed with long form names after switching to the Internet protocols,[10] however one lasting effect has been that some of the short form domains are cryptic and ugly. Hence we are currently re-introducing a limited kind of long form domain for use in email addresses and URLs.

There are two kinds of relay domain: domains for which ppswitch will act as the secondary MX, and special routes. Together these list domains handled by departmental email servers; in the first case the department sends and receives email directly to and from the public Internet, whereas in the second case they do so via ppswitch. The special routes table is used instead of the DNS, to route email to the right departmental server when ppswitch is the domain's primary MX. See Appendix 9.6 for examples from the `relay_domains` and `special_routes` tables.

4.2. Postmaster domains

Our configuration on ppswitch includes somewhat over-engineered support for email to ''postmasterish'' local parts (i.e. *postmaster@* and *abuse@* – see line 107). As well as supporting *postmaster@* all our normal domains, we support it at any of ppswitch's host names and IP addresses (for example, *postmaster@*[131.111.8.129]). The aim of this is to support email to postmaster from automated systems or in other cases where intelligent guessing of email domains cannot be expected.

The list of postmaster domains is defined at line 92. It includes all of ppswitch's various names as defined on line 86, and all of the current host's IP addresses via the `@[]` notation. The list of `+our_names` is also used for the `hosts_treat_as_local` setting on line 280, which protects against certain kinds of mail loops. Email to IP addresses (known as ''domain literals'' in email terminology) must be specially enabled in Exim: see line 277. The first two routers starting at line 765 implement the postmaster handling. The first one just redirects all the relevant addresses to *postmaster@ppsw.cam.ac.uk*, and the second one produces a friendly error message for any non-postmasterish address at a postmaster domain.

Note that we go through a number of contortions to handle the domain *ppsw.cam.ac.uk* correctly, because of the overlap between `+our_names` and `+local_domains`. I have resolved this by omitting *ppsw.cam.ac.uk* from `+our_names` and including it separately in the `hosts_treat_as_local` setting. An alternative approach would be to give it a special exemption from the `postmaster_error` router, similar to the handling for *lists.cam.ac.uk* in the `domain_error` router. However given the problems that *ppsw.cam.ac.uk* being an email domain causes in the smarthost ACL, it may be better to reduce it to a pure host name and use another domain – perhaps *hermes.cam.ac.uk* – for ppswitch's `qualify_domain`.

4.3. Remote domains

''Remote domains'' are not local domains, or in Exim syntax `!+local_domains`. This includes relay domains and special routes, as well as all non-Cambridge domains. The next two routers in the configuration handle these domains; see line 787. Both of these routers send email out using a very basic SMTP transport defined at line 1040. Note that I have turned off TLS for outgoing email, since it wastes CPU that would be better used for spam and virus scanning.

The first of these handles special routes, which are straightforwardly handled by the `manualroute` router. If the lookup in the `special_routes` table succeeds, the result is appropriate for the `route_data` setting – see Appendix 9.6 for examples. If the lookup fails the router declines, so processing falls through to the `lookuphost` router; thus the `domains` precondition does not need to be precise. The effect of this is that managed mail domains take precedence over special routes, because the existence of a managed mail domain causes the `domains` precondition to fail. Alternatively we could perform the `special_routes` table lookup in the `domains` precondition and set `route_data = $domain_data`; the effect of this would be to make special routes take precedence over managed mail domains.

The `lookuphost` router is very similar to the `dnslookup` router in the default configuration file that comes with Exim. It handles all routeing to domains by MX record, which includes domains in the `relay_domains` table for which ppswitch is the secondary MX, as well as non-Cambridge domains. The `widen_domains` setting is used to support traditional abbreviated email addresses, such as *fanf2@ucs*. It

[10] Some universities kept both long and short forms; for example see Oxford's documentation[27].

only really covers recipient addresses, so there is also a rewrite rule at line 749 for expanding sender addresses. The `mx_domains` setting enforces the local policy that valid email domains must have an MX record. The final twiddle is the `ignore_target_hosts` setting, which is somewhat more thorough than the default one. It uses a list that is defined on line 120 to contain the appropriate reserved networks (see Appendix 9.5 and [28]), and a list of the IP addresses of hosts referred to by wildcards in top level domains. The latter is a defence against problems caused by Verisign's Site Finder and similar breakage perpetrated by other registries[29].

4.4. Managed mail domains

The handling for managed mail domains (i.e. virtual domains) starts at line 873. The core functionality is implemented by the `domain_aliases` starting at line 891. This is very similar to the `system_aliases` router in the default configuration file that comes with Exim. The most important difference is that the location of the aliases file is based on the domain that we are handling. The aliases file is also translated into a more efficient format. Because managed mail domain aliases files may be edited by untrusted users, all the insecure redirect router options are disabled so that the destination must be another email address. We also use the `check_ancestor` option to improve the handling of certain kinds of redirection loops.

If a local part is not found in the domain's aliases file, Exim falls through to the following routers. The next router at line 905 deals with domain managers that have not defined *postmaster@* or *abuse@* addresses; by default email to these addresses is sent to the domain managers. The `file` just contains a list of the managers' addresses. Any other undefined address is handled by the `domain_error` router at line 920, which just sets up an appropriate error message.

Expert readers might wonder why we have a separate `domainlist` table, rather than, say, using `dsearch;DOMAINS/aliases/`. The reason is that we sometimes need to disable a managed mail domain but keep its aliases file available for editing by the domain managers. This typically happens to domain that is being migrated from a departmental server to a managed mail domain: the managed mail domain is created in disabled mode so that it can be set up during preparation for the cut-over. Changing the priority of managed mail domains and special routes might be another way to solve the problem, though it would not help much with migrating a relay domain to a managed mail domain.

At the start of this section, at line 877, is the `domain_longshort` router which handles long form email domains. A long form domain is a synonym for the corresponding short form domain, so the router just replaces the domain part of the email address based on a lookup in the `longshort` table. (There is an example of its contents in Appendix 9.6.) If the lookup fails, Exim falls through to the subsequent routers which handle other local domains.

4.5. Special case domains

There are three[11] special case domains which have additional routers of their own: *hermes.cam.ac.uk*, *cam.ac.uk*, and *lists.cam.ac.uk*. Most of the addresses in each of these domains are handled by routers that are specific to the domain; however they all have some additional addresses that are not managed in the usual way for that domain. These include system addresses such as *root@*, or temporary work-arounds to compensate for widely-distributed printing errors. These additional addresses are implemented using the managed mail domain system. The *@hermes* and *@cam* routers appear before the `domain_*` routers, so that if they do not handle an address Exim will fall through to check the additional addresses, and if none of them matches the `domain_error` router will handle unknown addresses. The *@lists* routers are discussed in the next sub-section.

Most *@cam.ac.uk* addresses are handled by the router starting at line 854. This is almost exactly the same as the `domain_aliases` router, except that the location of the aliases file is different. The data for the file also comes from a different source: *jackdaw.cam.ac.uk*, the Computing Service's central user administration database. Individual users are able to control the destination of their own *@cam* address[30] rather than requiring the assistance of a domain manager, which is necessary when there are over 35 000 entries in the file. In the past the main *@cam* aliases and the additional aliases were merged into a single table for use by Exim, but the current approach means the table distribution scripts are simpler to maintain.

There are two routers for *@hermes* addresses, starting at line 818. One router is used for verifying addresses and the other for routeing them to their destination. They are separated in order to save the cost of call-forward recipient verification to the Cyrus message store, which is pointless since ppswitch has an accurate list of Hermes users. Both routers use the `HERMES_CYRUS` macro defined on line 55; this does a lookup in two

[11] The domain *ppsw.cam.ac.uk* is special in that it is wired into the configuration in various places, but from the routeing point of view it is just another managed mail domain.

tables, one which maps users to the Cyrus machine that stores their email, and one which lists cancelled accounts.[12] Hermes accounts are usually destroyed about 15–18 months after they are cancelled, because users often return after a year away. If the account doesn't exist or has been cancelled, the `hermes_*` routers pass the address on to the `domain_*` routers.

The transport configuration for Hermes is complicated by two things. Firstly, it uses the Local Mail Transport Protocol[31] which is a variant of SMTP that allows the Cyrus server to provide different accept or reject responses for different users after the message data has been sent. If a user is over quota, Cyrus will defer the message for that individual user while delivering it to any other recipients that do not have quota problems. The message remains on the queue on ppswitch to be retried later, and the Cyrus machines to not have to maintain delivery queues of their own. Secondly, Cyrus is very strict about rejecting messages that contain NUL bytes. In an ideal world this would not be a problem, but buggy email software generates messages that are slightly mangled by NUL bytes surprisingly frequently, and this caused complaints. Therefore we use a transport filter to remove NUL bytes before they get to Cyrus. For performance reasons, we only do this if Exim spotted a NUL byte when receiving the message, using the fact that the `transport` setting is expanded to select the Hermes LMTP transport with or without the filter as appropriate.

4.6. Mailing lists

Cambridge uses home-grown software to implement its mailing list system[32]. Since the software is unique and shortly to be phased out, I will only give a brief discussion of the implementation here.

The lists system is divided between the Exim routers and a support program called `explode_list`. The latter implements the special features of lists, including moderation, approval, posting restrictions, and list footers. All messages sent to the list are passed through this program: see the `lists_process` router at line 1018 which hands off to the `list_pipe` transport at line 1073.

Despite its name, `explode_list` doesn't distribute messages to the list members. Instead it sends the message to the list's `-outgoing` address. The work of sending the message to the list members is entirely handled by Exim's `lists_outgoing` router: see line 934. The file is just a list of the mailing list members' addresses. Postings to a list are forced to go via `explode_list` because the router condition requires outgoing postings to come from the local machine. Only the ppswitch system administrators can forge them.

Moderation messages are also sent by `explode_list`, to the list's `-moderators` address: see line 952. This works very similar to the `-outgoing` address, except that anyone is permitted to send a message to a list's moderators. However, a list might not have a moderation team configured, so this is checked for by the router's condition. If the condition fails – an empty or missing file implies no moderators – the next router at line 972 redirects the message to the list's managers instead, who are the default moderators.

The next two routers implement a couple of aliases for the list managers' address. The `-owner` address is traditional; the `-request` address is used as the return address on outgoing postings to the list (see line 950) so that list managers handle bounces. The actual list managers distribution router is at line 993; it is very similar to (and simpler than) the `-outgoing` and `-moderators` addresses.

The special list verification router at line 1007 is slightly subtle. Instead of verifying that we can deliver the message to `explode_list`, we check that after the message has passed through `explode_list` to the `-outgoing` router, it will be given a valid return address – i.e. that the list manager's address is valid. This is to catch messages to lists that are orphaned by the departure of their managers. In fact, because Exim stops verification when it finds an address has more than one destination, this only deals with lists that have one manager. Other cases are eventually handled by *postmaster@lists*.

The list handling is all kept at the end of the routers section because there is so much of it, and it keeps the interesting logic of the other routers together. The number of domains handled by the routers decreases as you read through the file: all non-local domains, then local domains, and finally *lists.cam.ac.uk* only. The `domain_error` router is last router apart from the lists – see its `domains` precondition at line 925. The lists have their own last-of-all error handling router at line 1027 with a message that talks about lists rather than users.

5. Miscellanea

This section describes some Exim ''hacks''. The first two sub-sections describe a couple of stunt routers that I have used in the past as part of our live configuration to fulfil some special requirements. The final subsection is just for fun.

[12] We plan to improve efficiency by replacing these with a single table of active accounts.

5.1. Draining queues

One thing that must be dealt with when taking a machine out of service is preserving all of its unique data. On ppswitch this means moving the queue data to another ppswitch machine. One way of doing this is with the following router:

```
drain:
  driver        = manualroute
  no_verify
  require_files = TABLES/drain_info
  route_data    = ${readfile{TABLES/drain_info}}
  transport     = smtp
```

Most of the time this router does nothing. It can be activated by creating a file containing the name of the host to which the queue should be drained, for example:

```
echo ppsw-5.csi.cam.ac.uk > /opt/exim/etc/tables/drain_info
```

This router is designed to work in conjunction with the `benice` feature of our ACLs – see line 454. This ensures that messages being transferred from a privileged machine will be accepted without question. However if you look at Appendix 9.3 you will see that ppswitch is nice to the Cyrus machines, but it is not nice to other machines in the ppswitch cluster as would be necessary to support this router. This is because we no longer use this method for draining a queue because it has a few problems. The drain router doesn't preserve authentication information when transferring messages, which is required for some of the anti-spam features we are planning for the future. It also doesn't preserve the message age information, which messes up retry calculations and can cause duplicate delay warning messages.

Our current approach to preserving the queue of a decommissioned machine is to tar it up and copy it to another machine. We use the `localhost_number` feature (see line 196) to ensure that message IDs are unique across the ppswitch cluster, so that when a queue is moved to a new machine there will not be any file name collision. This method preserves authentication information, details of the client machine that sent the message, the message age, and other metadata.

5.2. Scanner testing

Before I deployed MailScanner I did some testing using real email in order to ensure that my candidate configuration did not do anything unexpected. I also wanted to get an idea of its performance and its behaviour under overload conditions. This meant I needed a way to get a copy of part of our email traffic sent to my test server, which would then run it through the scanner before discarding it. I also needed to be able to adjust the volume of the test traffic so that it could be slow enough to observe in real time, or fast enough to load the test machine. The solution was a more elaborate version of the drain router from the previous sub-section.

```
TAP_INFO = ${readfile {TABLES/tap_info}{:} }

# ...

traffic_tap:
  driver        = manualroute
  no_verify
  require_files = TABLES/tap_info
  condition     = ${if eq{a}{${hash {1} \
                                {${extract {2}{:} {TAP_INFO} }} \
                                {$message_headers$message_body} }} \
                       {yes} {no} }
  address_data  = ${if !def:address_data {tapped} fail }
  route_data    = ${extract {1}{:} {TAP_INFO} }
  unseen
  transport     = smtp
```

The file that controls the router has two colon-separated fields: a hostname and a number. We use `${extract` to pull out the individual fields, see for example the `route_data` setting (and compare it with the drain router). The number field is used to select a fraction of the total traffic: if the number is 4 then a quarter of the traffic is tapped. We use `${hash` to generate a random letter from the first *N* in the alphabet (e.g. *a*, *b*,

c, *d* if *N* = 4), and the letter is always the same for a given message so that the decision to tap is consistent. If the letter is ‘‘a’’ then the message is tapped.

In order to ensure that the testing doesn't affect normal email handling, the traffic tap takes a *copy* of the email. This is done using the `unseen` setting. However we also have to ensure that a message isn't repeatedly tapped each time it passes through the routers. This is done using the `address_data` setting: if no address data has been set, it is set to ‘‘tapped’’; if it has already been set then the expansion failure causes the traffic tap router to be skipped. Users don't receive duplicate email because the test system is configured to discard it.

5.3. An Exim quine

Readers of the *exim-users* mailing list might have noticed the cryptic signature I use on my postings:

```
<fanf@exim.org>  <dot@dotat.at>  http://dotat.at/  ${sg{\N${sg{\
N\}{([^N]*)(.)(.)(.*)}{\$1\$3\$2\$1\$3\n\$2\$3\$4\$3\n\$3\$2\$4}}\
\N}{([^N]*)(.)(.)(.*)}{\$1\$3\$2\$1\$3\n\$2\$3\$4\$3\n\$3\$2\$4}}
```

This is an Exim string expansion ‘‘quine’’, i.e. a program whose output is its own source code, named after the logician Willard van Orman Quine. The version in my email signature is laid out to fit the space available, but a shorter and clearer version of the quine is possible:

```
${sg{\N${sg{N\}{(.*?)N(.)(.*)}{\$1\$2N\$0\$2N\$3}}\N}{(.*?)N(.)(.*)}{\$1\$2N\$0\$2N\$3}}
```

Quines are constructed in roughly the same way in any programming language. They consist of a quoted string and some code which prints out two copies of the string, one copy in quotes and one copy bare. If the quoted string contains a copy of the code, then you have a quine.[13] In the Exim quine the quoted string appears between the two `\N` no-expansion markers:

```
\N${sg{N\}{(.*?)N(.)(.*)}{\$1\$2N\$0\$2N\$3}}\N
```

The code part is an `${sg` expansion item, whose first argument is the quoted string (marked with · · · below):

```
${sg{···}{(.*?)N(.)(.*)}{\$1\$2N\$0\$2N\$3}}
```

The second argument is a regex to match against the quoted string. This breaks the string up into a preamble before the first `N`, the `N` itself, the following character, and the rest of the string. The character after the `N` in the quoted string is a backslash, which we capture in the regex match variable `$2` in order to include it in the output without quoting problems. The final `${sg` argument is its output string, which consists of the preamble `$1`, a quote mark `$2N`, i.e. `\N`, the original quoted string `$0`, another quote mark, and a copy of the code consisting of everything from the first `}`. The output string is peppered with backslashes so that the variables are interpolated after the regex match.

Although this is just a bit of intellectual fun, it also has a purpose: it demonstrates that Exim's string expansions are extremely powerful, and at the same time shows that they can become incomprehensible. This is true for real-world string expansions as much as for kamikaze string expansions – see the discussion about the `received_header_text` setting in section 3.8. The following section describes how you can reduce the problem by using a good layout style.

6. Style

I have diverged somewhat from the style used by the default configuration file in the Exim distribution. The aim of style rules is to improve the readability and maintainability of the file. Exim has a very large number of options, not all of which are clearly named, so you can help the reader by ordering options systematically to suggest what area of functionality they affect. Some parts of Exim's configuration language can become very cryptic, but you can mitigate this with careful layout and judicious use of macros. In this section I'll describe the style rules I use and explain the rationale for them.

6.1. Order of sections

The order of the sections in ppswitch's configuration file is intended to correspond roughly to the order in which they apply to a message, as follows:

- Main configuration

  This section must appear first in the file. The order of the items inside it is explained below.

[13] This is explained better by David Madore[33].

- Access Control Lists

  This section controls SMTP input, so it appears near the start. Within the section the ACLs are ordered according to the SMTP conversation, i.e. not_smtp, connect, helo, expn, vrfy, etrn, starttls, auth, mailauth, mail, rcpt, predata, data, quit. However ppswitch's multiple-personality configuration is a little more complicated than that, so the ACLs are ordered first by personality then according to SMTP.

- Local-scan options

  We don't use any `local_scan` extension, but if we did the options would go after the ACLs.

- Authenticators

  Authentication is closely related to access control, which is why it appears next. This makes most sense when Exim is providing authentication as a service; when it is authenticating as a client it would make more sense to put the authenticators after the transports. However Exim muddles client and server authentication together, and since server authentication is more common I have decided to put the section here.

- Rewrite rules

  Addresses are rewritten before they are routed, so the rewrite rules appear immediately before the routers. Rewrite rules can also apply at SMTP time before the ACLs run, but this is a discouraged and rarely-used feature, so this section is most comfortable here.

- Routers

  This section slots in logically between the rewrite rules and the transports.

- Transports

  After a message has been routed it is transported, so the transports section appears next.

- Retry rules

  Finally, if a message is not delivered successfully it is retried at a later time, so the retry rules appear last.

6.2. Main configuration section

As well as general configuration directives, this section contains macro definitions (which I put first) and named list definitions (which I put between the macros and the general directives).

I've used macros fairly heavily, for about three different purposes: to abbreviate pathnames; to hide complicated lookups; and to make ACL variable names more readable. In all cases the macros replace a string which appears a number of times in the configuration, so they also help to ensure that consistent settings remain so. They are also particularly useful for improving readability, especially for complicated string expansions. The idea of using them to provide more meaningful names for ACL variables is due to Matthew Newton of the University of Leicester.

The main configuration directives can easily become unwieldy: ppswitch's configuration has nearly 200 lines of them. Fortunately the Exim specification includes lists of the directives divided into categories, and I have used these categories to divide up the section into groups of related directives. Some directives are listed in more than one category in the specification, and this provides you with some flexibility in ordering your configuration.

6.3. Routers

A large proportion of the logic of an Exim configuration is in its router configuration, so it is important to make the routers easily understandable by people reading the configuration file (including the original author weeks later!). Fortunately routers are usually quite short and simple-looking, but there is a large number of router options some of which have quite subtle behaviour. You can improve the clarity of the configuration by careful ordering of the router options, as follows:

- First declare the router's driver.
- Then put any preconditions, in the order in which they are tested. This is described in the ''Router Preconditions'' section of the Exim specification.
- Some router options can cause a router to decline or pass, for example, if string expansion fails. This is similar to the effect of failed a precondition, so these options should appear next. Generic options should come before per-driver options.
- Most of the per-driver options that can affect whether the router accepts are related to configuring the actions of the router, so immediately after them come the other per-driver options that configure the core

functions of the router.

- Next come the various options that control subsequent routing, such as which router to use instead of the following one when this router does not accept. The `redirect` router also has some per-driver options which fall into this category.
- Finally come the options related to the transport that is set up by the router. A few of these are expanded in a specific order, so they should appear first in the order: `errors_to`, `headers_add`, `headers_remove`, `transport`. The `redirect` option also has a few `_transport` options which can appear next, followed by any of the remaining transport-related options.

6.4. Transports

Transports tend to be rather simpler than routers from the configuration point of view. They have fewer clear criteria for deciding in which order to write the options, but at the same time the order matters less for clarity. I use the following guidelines:

- First declare the transport's driver.
- Then put any options that affect the delivery destination, or whether delivery is successful.
- Closely related to the preceding are the options that can be used to override the result of routing, so they come next.
- Then options that alter the message before delivery, or which affect its format.
- Finally put any options that don't fall into any other category.

6.5. Layout

In the default configuration file, the equals signs in the ACL section are lined up vertically. I have extended this to the other sections of the configuration. This helps to visually separate the option names from their values, making the option names easier to read especially when options of widely differing lengths appear next to each other.

One disadvantage of this is that it tends to make option values march off the right-hand-side of the screen, especially in the ACL section. To reduce this problem I have put the ACL conditions and modifiers under their controlling verb, indented slightly. Two features alter the logic of ACL verbs: the not-operator "`!`" and the `endpass` modifier; I line these up under the ACL verb.

Another ACL style guideline worth noting is to always put any `message` modifier first (unless there are special reasons for putting it later). This avoids confusing mistakes caused by the short-cutting logic of ACLs, especially in the case of the `require` verb.

6.6. String expansions

This is where there is the most scope for making Exim configurations hard to read, let alone understand and maintain! Exim allows you to add white space and line breaks at certain points inside string expansion expressions without affecting the result, and this can be used to improve readability. In general, my rule is to space out the inside of expansion items[14] (where white space is not significant) so that the item's arguments are separated and the closing brace is not cuddled[15]. (Exim requires the opening brace and keyword to cuddle.) Whitespace in nested braces inside expansion items is usually significant, so nested braces are cuddled, as are braces around expansion operators[16].

The general spaced-out rule has exceptions where that improves readability, for example the condition part of a `${if` is not spaced out in order to keep it as a visually distinct unit from the other arguments. In fact there are several expansion items which end with two arguments for the success and failure cases (which are spaced out) following the non-optional arguments (which are not spaced out). If the line needs to be broken for length reasons, this is done before the success argument.

These rules are not very clear when described in the abstract, so I'll list prototype layouts for individual expansion items which aren't simply spaced-out. I have indicated good places to break the line with a backslash "`\`"; if you need more breaks than this, breaking between `{yes}` and `{no}` is always an option – on the other hand the `{yes}` and `{no}` parts (or just the `{no}` part) may always be omitted[17]. Arguments on subsequent

[14] Expansion items start with a bare keyword, for example `${if`.

[15] By "cuddled", I mean `{cuddled}` as opposed to `{ not cuddled }`.

[16] Expansion operators start with a keyword and a colon, for example `${md5:`.

[17] This is a new feature of `${if` in 4.50.

lines should usually be lined up under the first argument. There are two exceptions to this: conditions, which are covered below; and single-key lookups with a long key, where the type should be lined up under the lookup keyword.

```
${extract {key}{string} \ {yes} {no} }
${extract {num}{sep}{string} \ {yes} {no} }
${hash {n}{m} {string} }
${hmac {type}{secret} {string} }
${if cond \ {yes} {no} }
${lookup type{query} \ {yes} {no} }
${lookup {key} \ type {file} \ {yes} {no} }
${substr {n}{m} {string} }
${tr {string} {chars}{replace} }
```

Complicated expansion conditions in `${if` items can easily get unwieldy because of the number of extra pairs of braces required by the `and{}` and `or{}` conditions. As for other conditions, I cuddle up the condition and its braces, except that I do *not* cuddle the braces around each inner condition and I break the line between them. A prototype layout would be:

```
${if cond{{ inner{}{} } \
          { inner{}{} }} \
     {yes} {no} }
```

Similar layout can be used for the `saslauthd` condition:

```
${if saslauthd{ {user} {pass} } \
     {yes} {no} }
```

The general effect of the rules on complicated nested expansion expressions is to cause closing braces to cluster in singles or pairs, which makes them easier to count and match with their opening braces. There are a number of examples of this in the configuration file: see the `HERMES_CYRUS` macro at line 55; the `smtp_banner` setting at line 248; the `received_header_text` setting at line 287; the back-logging check in the ACL at line 612; the callback verification conditions at line 647; the `HELO` verification tests at line 659; the authentication lookups at line 718; and the non-empty file check at line 960.

7. Future work

A system administrator's work is never done. There are a number of areas in which ppswitch's current implementation is not ideal, and of course we are working to improve them.

7.1. Anti-spam

In the anti-spam area the major problem at the moment is that users have no control over SMTP-time checks. This is only OK so long as the one-size-fits-all approach is neither too strict nor too loose. A related problem is that scoring email and allowing users to filter probable spam into a separate mailbox often has the same effect as silently throwing away email, since users rarely check their spam folder. Again, this is only OK if the scoring is reasonably accurate.

Therefore we are currently working on infrastructure that will allow users to control some personal settings on ppswitch. The immediate goal of this is to implement forgery protection and collateral spam detection[34] (which will be a per-user option). When we replace MailScanner (or perhaps supplement it) with the Exiscan content scanning features that are built-in to Exim from version 4.50, users will also be able to set a spam score threshold above which email will be rejected at SMTP time instead of filtered into a spam folder.

Other planned work includes the implementation of Client SMTP Authentication[35]. This will replace most of the HELO heuristics with DNS lookups. There's also scope for better use of Exim's logs to provide data that could be used for a locally-maintained blacklist.

7.2. Mailing lists and domains

The current locally-develop mailing lists system lacks many features that are nowadays considered to be basic requirements. These include automatic subscription and unsubscription, bounce handling, a web-based user interface, and better MIME handling, including proper internationalization.

My colleague David Carter is currently adapting Mailman[36] to integrate with our local web authentication system[37]. I also have a deep background project to improve Exim's handling of large recipient lists, in particular to allow concurrent DNS lookups when routeing addresses, which should make it reasonable for Exim to handle large mailing lists directly instead of relying on the mailing list manager to divide them into smaller batches.

There are similar requests for the Managed Mail Domain service: automated management of the domain's aliases file, and a more modern user interface. However given that this does not affect a large number of users, it is not currently a priority.

7.3. Reliability and availability

We currently rely on DNS round-robin to spread the load across the ppswitch machines. This has a number of disadvantages. If a machine fails we have to perform an emergency DNS update, and even then the broken address may remain cached for up to a day. When we take systems out of service for reconfiguration, we must consider DNS propagation delays, and even after taking them into account some systems will still cache IP addresses for longer than the TTL. This leads to unnecessary support questions, and means we have to implement a special IP address (131.111.8.129) for long-running client hosts.

The solution to this would be to deploy a front-end load balancer which handles configuration changes and failures gracefully, so that users are not disrupted by them.

A further problem is that ppswitch depends on the hardware SCSI RAID controllers in each machine for storage reliability: there is no higher-level replication as there is for the Cyrus message store. Catastrophic failure of a machine would mean the loss of hundreds of messages. Fortunately, in the past we have been able to recover queued messages from the remains of broken machines[38]. A solution to this might involve low-level replication technologies, though we tend to prefer application-level replication since it's easier to understand and debug.

8. Conclusion

Exim is an extremely flexible MTA. This paper has shown how we use this flexibility to provide a richly featured email service, with a very simple architecture that reduces the complexity of system management and scaling. This is especially due to the expressiveness of Exim's SMTP-time policy control, in the form of its Access Control Lists. Although our email routeing and delivery requirements are not especially complicated, Exim allows some nice simplifications and optimizations, and when we do need to perform stunts we can usually do so without extra programming. Exim's configuration can become cryptic unless care is taken with layout, so I have developed a style which makes it easier to read our configuration file.

Our configuration uses several of Exim's latest features, which have been implemented by Philip Hazel at least partly because of local needs. Examples include RFC 2476 message submission mode and some of the call-out options. Other options developed for use in Cambridge which do not appear in this configuration include the `mua_wrapper` mode and the enhanced DNS lookup features. Our plans for the future mean that this is likely to continue to be the case.

9. Appendices: configuration files and tables

The following sub-sections contain the live, unedited configuration files from our service machines. Not all the configuration files have been included; in order to avoid uninformative repetition I have just provided example lines from some tables. The configuration is not quite warts-and-all, because we maintain notes on future configuration plans and testing configurations in a different branch of our revision management system.

9.1. Exim configuration on ppswitch

```
# $Cambridge: hermes/conf/exim/etc/etc.ppsw/configure,v 1.294 2005/02/12 00:38:08 fanf2 Exp $

##############################################################################
#                    MAIN CONFIGURATION SETTINGS                             #
##############################################################################

## Macros for locating various files.

# data that comes from other machines

CERTS   = /opt/dist/certs
DOMAINS = /opt/dist/domains
LISTS   = /opt/dist/lists
USERS   = /opt/dist/users

# configuration that belongs to the Exim package

DB      = /opt/exim/etc/db
TABLES  = /opt/exim/etc/tables

## Behaviour changes based on our name

INTERFACE_PARAM = ${lookup {$interface_address} cdb {DB/addrparams.cdb} }
SENDER_PARAM    = ${lookup {$sender_host_address} cdb {DB/addrparams.cdb} }

PARAM           = INTERFACE_PARAM

NAME            = ${extract {name}{PARAM} {$value} {localhost} }

FULL_HOSTINFO   = $primary_hostname (${if def:interface_address \
                        {NAME [$interface_address]:$interface_port} \
                        {NAME} })

## Some abbreviations for ACLs and routers

# ACL variable names
#
# time to delay if the sending host is dodgy
ACL_DELAY       = acl_c0
#
# the client's HELO name was wrong
ACL_HELO        = acl_c1
#
# We are too busy
ACL_BUSY        = acl_c2

# standard callout timeouts
#
CALLTIME        = 4m,maxwait=4m,connect=30s

# check local port is 25
#
PORT25          = ${if ={25}{$interface_port} }

# Map a Hermes user to their Cyrus store. If the username is invalid,
# return an empty string which causes the routers to decline. Otherwise
# return the name of the user's Cyrus message store.
#
HERMES_CYRUS    = ${lookup {$local_part} cdb {USERS/hermes_cancelled.cdb} \
                     {} {${lookup {$local_part} cdb {USERS/cyrus.cdb} }} }

## Domain and host lists.

# List of domains handled on PPSW itself
#
# This list includes ppsw.cam.ac.uk which is $qualify_domain.
# It is handled in the same way as normal managed mail domains.
#
# This list also includes the special domains
#       cam.ac.uk
#       hermes.cam.ac.uk
#       lists.cam.ac.uk
# whose special-case aliases are handled as managed mail domains but
# which have a lot of additional addresses routed by other means.
#
domainlist local_domains = \
                cdb;DOMAINS/domainlist.cdb :\
                cdb;DOMAINS/longshort.cdb

# List of domains that PPSW will relay to
#
```

```
domainlist relay_domains = \
                cdb;DB/special_routes.cdb :\
                partial-cdb;DB/relay_domains.cdb

# Lists of all host names which might refer to us.
# ppsw.cam.ac.uk aka $qualify_domain is not included.
#
domainlist our_names = \
                cdb;DB/ppswnames.cdb

# Special-case domains for handling postmaster email,
# including domain literals [IP addresses].
#
domainlist postmaster_domains = \
                +our_names :\
                @[]

# List of all domains known to PPSW,
# including the local host to make automated postmaster contact possible
#
domainlist our_domains = \
                +local_domains :\
                +relay_domains :\
                +postmaster_domains

# Local parts which should be present in all domains
# and which should not be filtered.
#
localpartlist postmasterish = \
                postmaster :\
                abuse

# We are prepared to relay outgoing email from these hosts,
# and we give them favourable MX service.
#
hostlist relay_hosts = \
                TABLES/cudn_nets

# We will not deliver email to these hosts, and will reject email with
# an envelope-from domain that resolves to one of these hosts.
#
hostlist bad_hosts = \
                TABLES/bad_nets :\
                net-cdb;DB/badtlds.cdb

##
## Configuration options.
##
## See Chapter 14 of the Exim specification for the categories.
##

## Exim parameters

# to facilitate moving the queue from one machine to another using tar
#
localhost_number        = ${substr_5_1:$primary_hostname}

# special spool handling for MailScanner
SPOOL                   = /spool/exim
spool_directory         = SPOOL
split_spool_directory   = true

## Privileged users

deliver_drop_privilege  = true
never_users             = root
trusted_groups          = exim

## Logging

# log to the same place regardless of spool directory
log_file_path           = syslog:/spool/exim/log/%slog
process_log_path        = /spool/exim/exim-process.info
syslog_facility         = local5

# sensible timestamp handling
log_timezone            = true
syslog_timestamp        = false

# performance and content sanity
message_logs            = false
print_topbitchars       = true
syslog_duplication      = false
```

```
0165
    # adjust logging detail: don't log no-ops; log interface information
0167 # so we can tell the difference between ppsw and smtp.hermes; message
    # reception confirmation (often includes message-ID); more address
0169 # information on each line to reduce the need for exigrep and make the
    # delays caused by MailScanner less of a readability problem.
0171 .ifdef DEBUG
    log_selector    = +all
0173 .else
    log_selector    = -retry_defer -skip_delivery -host_lookup_failed \
0175                   +incoming_interface +incoming_port +smtp_confirmation \
                       +sender_on_delivery +return_path_on_delivery +delivery_size \
0177                   +received_recipients +all_parents +address_rewrite \
                       +tls_certificate_verified +tls_peerdn \
0179                   +smtp_protocol_error +smtp_syntax_error \
                       +deliver_time +queue_time \
0181                   -lost_incoming_connection
    .endif
0183
    ## Resource control
0185
    # These protections need to take into account MailScanner's need to do
0187 # MIME explosion.

0189 check_spool_inodes      = 1000
    check_spool_space       = 1000M
0191
    # ppsw has a generous message size limit, Hermes less so --
0193 # see also the cyrus LMTP limit and the Exim client limit
    message_size_limit      = ${extract {msgsizelim}{PARAM} {$value} {100M} }
0195
    # Note that there are seven concurrent MailScanner processes, and that
0197 # (for SMTP input) we queue_only anyway, so the queue_only_load helpls
    # mostly with mailing list messages.
0199
    smtp_accept_max_per_host = 10
0201
    deliver_queue_load_max  = 20.00
0203 queue_only_load         = 10.00
    queue_run_max           = 20
0205 smtp_accept_max         = 400
    smtp_accept_reserve     = 20
0207 smtp_load_reserve       = 15.00
    smtp_reserve_hosts      = +relay_hosts
0209
    ## Policy controls
0211
    # The default ACL name is based on the default NAME of localhost.
0213
    acl_smtp_connect        = acl_conn_${extract {acl}{PARAM} {$value} {local} }
0215 acl_smtp_helo           = acl_helo_${extract {acl}{PARAM} {$value} {local} }
    acl_smtp_rcpt           = acl_rcpt_${extract {acl}{PARAM} {$value} {local} }
0217 acl_smtp_data           = acl_data_${extract {acl}{PARAM} {$value} {local} }

0219 acl_smtp_vrfy           = accept

0221 ## TLS

0223 # server-side TLS settings
    tls_advertise_hosts     = ${if exists{CERTS/server/NAME} {*} {} }
0225 tls_certificate         = CERTS/server/NAME
    tls_dhparam             = CERTS/dhparam
0227 tls_on_connect_ports    = 465

0229 # Eudora/Outlook bug: if we ask it for a client certificate,
    # it bails out instead of declining gracefully.
0231 # For more info see the interoperability section of
    # http://www.sendmail.org/~ca/email/starttls.html
0233 # This makes client TLS authentication hard to support :-(
    #
0235 # We'll have to think more about the interaction with AUTH
    # advertisement too, since it's currently keyed on the use of
0237 # TLS (rather than the use of the submission service) whereas
    # verify = certificate is more of an smarthost thing.
0239 #
    #tls_try_verify_hosts   = *
0241 #tls_verify_certificates= CERTS/client

0243 ## Incoming SMTP

0245 # see also tls_on_connect_ports above
    daemon_smtp_ports       = 25 : 465 : 587
0247
```

```
# optionally attempt to confuse ratware
smtp_banner             = \
        FULL_HOSTINFO ESMTP Exim $version_number+ppsw+$compile_number $tod_full\
        ${if match{PARAM}{acl=mx} {${run {/usr/bin/fortune -s} {\n$value} }} }

# Make ESMTP PIPELINING available in all cases except when in submission mode.
# This is an attempt to make the error handling of Outlook better, so that it
# reports the response to RCPT instead of the response to DATA when a message
# is rejected. We also turn off synchronization checks to allow for crapware
# that tries to pipeline anyway.
#
pipelining_advertise_hosts = ${if match{PARAM}{acl=submit} {:} {*} }

# a bit of good cop / bad cop with helo
helo_allow_chars        = "_"
helo_try_verify_hosts   = *

# reverse DNS information is useful
helo_lookup_domains     = *
host_lookup             = *

# ident lookups are not and cause firewall problems.
rfc1413_hosts           = :
rfc1413_query_timeout   = 0s

smtp_return_error_details

## Message processing

# only for postmaster
allow_domain_literals

# never send email to another ppsw machine
hosts_treat_as_local    = +our_names : $qualify_domain

# email domain on locally-generated messages
qualify_domain          = ppsw.cam.ac.uk
remote_sort_domains     = *.cam.ac.uk : *.ac.uk : *.uk

# tweaked so that it is clear which way the message arrived
# for reference, the default is:
#
# received_header_text = Received: \
#       ${if def:sender_rcvhost {from $sender_rcvhost\n\t} \
#            {${if def:sender_ident {from $sender_ident } }\
#             ${if def:sender_helo_name {(helo=$sender_helo_name)\n\t} }} }\
#       by $primary_hostname \
#       ${if def:received_protocol {with $received_protocol} } \
#       ${if def:tls_cipher {($tls_cipher)\n\t} }\
#       (Exim $version_number)\n\t\
#       id $message_id\
#       ${if def:received_for {\n\tfor $received_for} }
#
received_header_text = Received: \
        from ${if def:sender_rcvhost {$sender_rcvhost\n\t} \
                  {${if def:sender_ident {$sender_ident } {localhost } }\
                   ${if def:sender_helo_name {(helo=$sender_helo_name) } }} }\
        by FULL_HOSTINFO\n\t\
        ${if def:received_protocol {with $received_protocol } }\
        ${if def:sender_host_authenticated \
             {($sender_host_authenticated:$authenticated_id) } }\
        ${if def:tls_cipher {($tls_cipher)\n\t} }\
        id $message_id (Exim $version_number)\
        ${if def:received_for { for $received_for} }\n\t\
        (return-path <$sender_address>)

## Frozen, bounce, and warning messages

bounce_return_body
bounce_return_size_limit        = 10K
errors_reply_to                 = postmaster@cam.ac.uk

# single warning after 24h (30d will never be reached)
delay_warning                   = 24h:30d

ignore_bounce_errors_after      = 24h
auto_thaw                       = 8h

#####################################################################
#                     ACL CONFIGURATION                             #
#####################################################################

begin acl
```

```
0331
      ###############################################################################
0333  #
      #  ACLs for messages sent via localhost
0335  #

0337  acl_conn_local:
        accept
0339
      acl_helo_local:
0341    accept

0343  acl_rcpt_local:

0345    # Be secure in case of config cock-up.

0347    require
          message       = No SMTP service for unauthorized users
0349      hosts         = : @[] :

0351    # Check all envelope addresses.

0353    require
          verify        = sender
0355      verify        = recipient/callout=use_sender,defer_ok
          acl           = aux_verify_sender
0357
        accept
0359
      # end of acl_rcpt_local
0361
      acl_data_local:
0363    accept

0365  ###############################################################################
      #
0367  #  ACLs for the smarthost service
      #
0369
      acl_conn_smart:
0371    accept

0373  acl_helo_smart:
        accept
0375
      acl_rcpt_smart:
0377
        # This service is only available on port 25.
0379
        require
0381      message        = No SMTP service for unauthorized users
          condition      = PORT25
0383
        # Make it easy to get help.
0385
        accept
0387      domains        = +our_domains
          local_parts    = +postmasterish
0389
        # Accept email from machines we should be nice to without question.
0391
        accept
0393      condition      = ${extract {benice}{SENDER_PARAM} }

0395    # The sender must be allowed to relay through us,
        # or the recipient must be in the smarthost domain.
0397    # The latter is so that the smarthost can be its own MX,
        # to avoid confusion from clients that use the MX instead
0399    # of the A record to route outgoing email.

0401    deny
          message        = No SMTP service for unauthorized users
0403    ! hosts          = +relay_hosts
        ! domains        = NAME
0405
        deny
0407    ! hosts          = +relay_hosts
          domains        = NAME
0409    ! verify         = recipient/callout=use_sender,defer_ok

0411    # Set up submission mode, in case we accept the message.
        # We have to fix up partly-formed messages to support
0413    # certain clients, but since this service may be relaying
```

```
    # messages we leave the Sender: header alone.

    require
      control        = submission/sender_retain

    # Require valid recipient addresses on bounce messages.

    accept
      senders        = :
    endpass
      verify         = recipient/callout=CALLTIME

    # Do return address verification compatible with the mx service.

    require
      verify         = sender
      acl            = aux_verify_sender

    accept

  # end of acl_rcpt_smart

  acl_data_smart:
    accept

  ################################################################################
  #
  #  ACLs for the message submission service
  #

  acl_conn_submit:

    # Turn off synchronization checks, in order to be more forgiving to
    # incompetent SMTP implementations like Outlook, especially when ESMTP
    # PIPELINING is turned off. See pipelining_advertise_hosts above.

    require
      control        = no_enforce_sync

    accept

  # end of acl_conn_submit

  acl_helo_submit:
    accept

  acl_rcpt_submit:

    # Make it easy to get help.

    accept
      domains        = +our_domains
      local_parts    = +postmasterish

    # The sender must be either allowed to relay or authenticated.

    deny
      message        = No SMTP service for unauthorized users
    ! hosts          = +relay_hosts
    ! authenticated  = *

    # Set up submission mode, in case we accept the message.

    require
      control        = submission/domain=${extract {domain}{PARAM} \
                                                   {$value} {$qualify_domain} }

    # Require valid recipient addresses on bounce messages.

    accept
      senders        = :
    endpass
      verify         = recipient/callout=CALLTIME

    # Do return address verification compatible with the mx service.

    require
      verify         = sender
      acl            = aux_verify_sender

    accept

  # end of acl_rcpt_submit
```

```
0497
      acl_data_submit:
0499    accept

0501  ###############################################################################
      #
0503  #  ACLs for messages from the public Internet
      #
0505
      # The delays at the start of the SMTP conversation are to help Exim's
0507  # synchronization checks catch pump-and-dump spamware and viruses.
      # Compare and contrast acl_conn_submit above.
0509
      acl_conn_mx:
0511
        # Be nice to friendly machines.
0513
        accept
0515      hosts          = +relay_hosts

0517    # Assume we won't have to delay.

0519    warn
          set ACL_DELAY  = 0s
0521
        # We delay if the sender is blacklisted.
0523
        warn
0525      dnslists       = list.dsbl.org : \
                           multihop.dsbl.org : \
0527                       rbl-plus.mail-abuse.ja.net : \
                           combined.njabl.org : \
0529                       relays.ordb.org : \
                           dnsbl.sorbs.net : \
0531                       sbl-xbl.spamhaus.org
          set ACL_DELAY  = 5s
0533
        # We delay if the sender has bad DNS.
0535
        warn
0537    ! verify         = reverse_host_lookup
          set ACL_DELAY  = 5s
0539
        # Do whatever delay we worked out.
0541
        accept
0543      delay          = $ACL_DELAY

0545  # end of acl_conn_mx

0547  acl_helo_mx:

0549    # Be nice to friendly machines.

0551    accept
          hosts          = +relay_hosts
0553
        # We delay if the sender says the wrong hello domain.
0555
        warn
0557    ! verify         = helo
          set ACL_DELAY  = 5s
0559
        # Do whatever delay we worked out.
0561
        accept
0563      delay          = $ACL_DELAY

0565  # end of acl_helo_mx

0567  acl_rcpt_mx:

0569    # This service is only available on port 25.

0571    require
          message        = No SMTP service for unauthorized users
0573      condition      = PORT25

0575    # Make it easy to get help

0577    accept
          domains        = +our_domains
0579      local_parts    = +postmasterish
```

```
0581    # We accept email only for domains that we know about.
        # This check is cheap so we do it early to save time.
0583
        require
0585      message        = Relaying is not permitted
          domains        = +our_domains
0587
        # Do some anti-spam checking for non-friendly machines.
0589
        deny
0591    ! hosts          = +relay_hosts
        ! acl            = aux_check_spam
0593
        # Do cheap sender domain verification to avoid further work.
0595
        require
0597      verify         = sender

0599    # All recipient addresses must be valid, more or less.

0601    require
          message        = ${acl_verify_message}\n\
0603                       See http://www.cam.ac.uk/cs/email/bounce.html
          verify         = recipient/callout=use_sender,defer_ok
0605
        # Do more thorough sender address checks. We do this after verifying the
0607    # recipient address to reduce the number of sender callouts.

0609    require
          acl            = aux_verify_sender
0611
        # Don't accept email if we are too busy. We keep this check at the end
0613    # of the ACLs and ensure we do it only once because it can be expensive.

0615    defer
          message        = Sorry, too busy. Try again later.
0617      condition      = ${if or{{ eq{$ACL_BUSY}{yes} } \
                                   { <{300}{${run {/opt/exim/sbin/exim_incount} }} }} }
0619      set ACL_BUSY   = yes

0621    # Every check has been passed.

0623    accept

0625  # end of acl_rcpt_mx

0627  acl_data_mx:
        accept
0629
      ###############################################################################
0631  #
      #  Auxiliary ACLs called by the others
0633  #

0635  aux_verify_sender:

0637    # Only do sender callouts if the sender is not known to be incompetent
        # according to any of the preliminary ACL conditions. We assume that
0639    # the caller has already required verify = sender.

0641    accept
          condition      = \
0643        ${lookup {${lc:$sender_address_domain}} partial-cdb {DB/nocallout.cdb} \
                    {yes} {${lookup {${lc:$sender_address}} cdb {DB/nocallout.cdb} \
0645                                {yes} {no} }} }

0647    accept
          dnslists       = dsn.rfc-ignorant.org/$sender_address_domain
0649
        require
0651      verify         = sender/callout=CALLTIME,defer_ok

0653    accept

0655  # end of aux_verify_sender

0657  aux_check_spam:

0659    # Check for ratware HELO signatures. We don't use the full strictness of
        # verify=helo; if it fails we only check for a few choice stupidities.
0661
        deny
```

```
0663      message        = Please use your name when saying HELO (not $sender_helo_name)
        ! verify         = helo
0665      condition      = ${if or{{ eq{$ACL_HELO}{bad} } \
                                 { isip{$sender_helo_name} } \
0667                             { eq{$sender_helo_name}{$local_part} } \
                                 { match{$sender_helo_name}{\N[.][.]|.{55}\N} } \
0669                             { match_domain{$sender_helo_name}{+our_domains} }} }
          set ACL_HELO   = bad
0671
        # Look up in a few choice blacklists.
0673
        deny
0675      message        = ${sender_host_address} is listed at ${dnslist_domain}; \
                           See ${dnslist_text}
0677      dnslists       = sbl-xbl.spamhaus.org

0679    deny
          message        = ${sender_host_address} is listed at ${dnslist_domain}; \
0681                       See http://mail-abuse.com/cgi-bin/lookup?${sender_host_address}
          dnslists       = rbl-plus.mail-abuse.ja.net
0683
        deny
0685      message        = ${sender_address_domain} is listed at ${dnslist_domain}; \
                           ${dnslist_text}
0687      dnslists       = nomail.rhsbl.sorbs.net/$sender_address_domain

0689    # It has passed the tests.

0691    accept

0693  # end of aux_check_spam

0695  ###############################################################################
      #                   AUTHENTICATION CONFIGURATION                              #
0697  ###############################################################################

0699  # Note that although the authenticators aren't explicitly restricted
      # to the submission service, they are only used in that case because
0701  # only the submission service has a TLS certificate, and the MUA
      # server only sends messages via the submission service.
0703
      begin authenticators
0705
      # We could be vulnerable to password stealing by spammers, so it's
0707  # important that the authentication mechanisms are reasonably secure.
      # We protect passwords from snooping by requiring TLS, and the
0709  # password-changing program checks for basic password security.
      # TLS is only advertised if we have a certificate available, and we
0711  # only have certificates for the message submission service.

0713  LOGIN:
        driver               = plaintext
0715    server_set_id        = $1
        server_prompts       = <| Username: | Password:
0717    server_condition     = \
              ${if crypteq{$2}{${lookup {$1} cdb {USERS/passwd.cdb} }} }
0719    server_advertise_condition = ${if !eq{}{$tls_cipher} }

0721  PLAIN:
        driver               = plaintext
0723    server_set_id        = $2
        server_prompts       = :
0725    server_condition     = \
              ${if crypteq{$3}{${lookup {$2} cdb {USERS/passwd.cdb} }} }
0727    server_advertise_condition = ${if !eq{}{$tls_cipher} }

0729  # This authenticator is used to communicate authentication from the
      # central MUA server to us for bounce address tagging. EXTERNAL is a
0731  # standard SASL mechanism that uses external information to
      # authenticate the stated username; in this case the external
0733  # information is that we trust the client, i.e. the MUA server.
      # The mechanism is only advertised to the MUA server.
0735
      EXTERNAL:
0737    driver               = plaintext
        server_set_id        = $1
0739    server_prompts       = :
        server_condition     = yes
0741    server_advertise_condition = ${extract {mua}{SENDER_PARAM} }

0743  ###############################################################################
      #                      REWRITE CONFIGURATION                                  #
0745  ###############################################################################
```

```
begin rewrite

# This is partly handled by the widen_domains in the lookuphost
# router, but that doesn't handle the envelope return path and
# other addresses in the headers.

*@cam                           $1@cam.ac.uk

# Continue to support broken Hermes user configurations.

*@*.hermes.cam.ac.uk            $1@hermes.cam.ac.uk             hF

#############################################################################
#                      ROUTERS CONFIGURATION                                #
#############################################################################

begin routers

# A special case for postmaster email directed to the local host, to
# allow automated systems to contact postmaster. Although email directed
# to specific hosts is in general not kosher and against local policy,
# the importance of ppsw means that it's probably best to make it easy
# to contact us without any knowledge of email in Cambridge.

postmaster:
  driver                = redirect
  domains               = +postmaster_domains
  local_parts           = +postmasterish
  data                  = postmaster@${qualify_domain}

# Produce a nice error message. Without this router the lookuphost router
# will say "Invalid domain part in email address" which isn't correct.

postmaster_error:
  driver                = redirect
  domains               = +postmaster_domains
  data                  = :fail: \
        "${local_part}@${domain}" is not a known user on this system.
  allow_fail

##
## Remote domains.
##

# List of special local routes that override MX information.
# If the lookup fails the router declines so the address is
# handled by the lookuphost router below.

special_routes:
  driver                = manualroute
  domains               = !+local_domains
  host_find_failed      = defer
  route_data            = ${lookup {$domain} cdb {DB/special_routes.cdb} }
  same_domain_copy_routing
  transport             = smtp

# This router routes to remote hosts over SMTP using a DNS lookup.
# We refuse to deliver email to hosts in Cambridge unless they are
# known email servers, i.e. they have MX records.

lookuphost:
  driver                = dnslookup
  domains               = !+local_domains
  ignore_target_hosts   = +bad_hosts
  mx_domains            = *.cam.ac.uk
  widen_domains         = cam.ac.uk : ac.uk
  same_domain_copy_routing
  no_more
  cannot_route_message  = Invalid domain part in email address
  transport             = smtp

##
## hermes.cam.ac.uk
##

# Verify Hermes addresses that are destined for the Cyrus messages stores
# in a separate router in order to avoid callouts. The HERMES_CYRUS mapping
# either returns a Cyrus hostname (equivalent to true) or an empty string
# (equivalent to false), in which case this router declines and the address
# falls through to the managed mail domain routers for special-case and
# unknown addresses.
```

```
hermes_verify:
  driver                = accept
  local_part_suffix     = +*
  local_part_suffix_optional
  verify_only
  domains               = hermes.cam.ac.uk
  condition             = HERMES_CYRUS

# Deliver most Hermes addresses to the appropriate Cyrus store.
# The HERMES_CYRUS mapping either returns a Cyrus hostname, suitable for
# use in the route_data, or an empty string, which causes this router to
# decline and the address falls through as before.

hermes_lmtp:
  driver                = manualroute
  local_part_suffix     = +*
  local_part_suffix_optional
  no_verify
  domains               = hermes.cam.ac.uk
  host_find_failed      = defer
  route_data            = HERMES_CYRUS
  retry_use_local_part
  transport             = ${if ={0}{$body_zerocount} \
                               {hermes_lmtp} {hermes_lmtp_filter} }

##
## cam.ac.uk
##

# A big special-case extension to the managed mail domain system.
# As for hermes.cam.ac.uk, we fall through to the routers below
# for special-case and unknown addresses.

cam_aliases:
  driver                = redirect
  domains               = cam.ac.uk
  data                  = ${lookup {$local_part} cdb {USERS/cam_aliases.cdb} }
  forbid_blackhole
  forbid_file
  forbid_include
  forbid_pipe
  check_ancestor
  retry_use_local_part

##
## DOMAINS
##

# Redirect long form addresses to their short form equivalents.

domain_longshort:
  driver                = redirect
  domains               = +local_domains
  data                  = ${lookup {$domain} cdb {DOMAINS/longshort.cdb} \
                                   {${local_part}@${value}} fail }
  forbid_blackhole
  forbid_file
  forbid_include
  forbid_pipe
  check_ancestor
  retry_use_local_part

# This includes special-case local parts in cam.ac.uk, hermes.cam.ac.uk,
# and lists.cam.ac.uk, and all addresses @ppsw.cam.ac.uk

domain_aliases:
  driver                = redirect
  domains               = +local_domains
  data                  = ${lookup {$local_part} cdb {DOMAINS/db/${domain}.cdb} }
  forbid_blackhole
  forbid_file
  forbid_include
  forbid_pipe
  check_ancestor
  retry_use_local_part

# Ensure postmaster@ always works.

domain_postmaster:
  driver                = redirect
  domains               = +local_domains
  local_parts           = +postmasterish
  file                  = DOMAINS/managers/${domain}
```

```
  forbid_blackhole
  forbid_file
  forbid_include
  forbid_pipe
  check_ancestor
  retry_use_local_part
  errors_to             = postmaster@ppsw.cam.ac.uk

# This router produces a nice error message for unknown users in any
# local domain other than lists.cam.ac.uk.

domain_error:
  driver                = redirect
  domains               = !lists.cam.ac.uk : +local_domains
  data                  = :fail: \
        "${local_part}@${domain}" is not a known user on this system.
  allow_fail

##
## lists.cam.ac.uk
##

# This router's condition requires that the message is not
# submitted over the network.

lists_outgoing:
  driver                = redirect
  local_part_suffix     = -outgoing
  domains               = lists.cam.ac.uk
  condition             = ${if eq{}{$sender_host_address} }
  file                  = LISTS/members/$local_part
  forbid_blackhole
  forbid_file
  forbid_include
  forbid_pipe
  check_ancestor
  one_time
  retry_use_local_part
  errors_to             = ${local_part}-request@lists.cam.ac.uk

# This router's condition requires that moderators file
# is non-zero in size.

lists_moderators:
  driver                = redirect
  local_part_suffix     = -moderators
  domains               = lists.cam.ac.uk
  require_files         = LISTS/moderators/$local_part
  condition             = ${extract {size} \
                            {${stat:LISTS/moderators/$local_part}} }
  file                  = LISTS/moderators/$local_part
  forbid_blackhole
  forbid_file
  forbid_include
  forbid_pipe
  check_ancestor
  one_time
  retry_use_local_part
  errors_to             = ${local_part}-managers@lists.cam.ac.uk

lists_no_moderators:
  driver                = redirect
  local_part_suffix     = -moderators
  domains               = lists.cam.ac.uk
  data                  = ${local_part}-managers@lists.cam.ac.uk
  check_ancestor

lists_owner:
  driver                = redirect
  local_part_prefix     = owner-
  domains               = lists.cam.ac.uk
  data                  = ${local_part}-managers@lists.cam.ac.uk
  check_ancestor

lists_request:
  driver                = redirect
  local_part_suffix     = -request
  domains               = lists.cam.ac.uk
  data                  = ${local_part}-managers@lists.cam.ac.uk
  check_ancestor

lists_managers:
  driver                = redirect
```

```
  local_part_suffix     = -managers
  domains               = lists.cam.ac.uk
  file                  = LISTS/managers/$local_part
  forbid_blackhole
  forbid_file
  forbid_include
  forbid_pipe
  check_ancestor
  one_time
  retry_use_local_part
  errors_to             = postmaster@lists.cam.ac.uk

# Ensure that a list's bounce address will verify
# before accepting messages to it.

lists_verify:
  driver                = redirect
  verify_only
  domains               = lists.cam.ac.uk
  require_files         = LISTS/members/$local_part
  data                  = ${local_part}-managers@lists.cam.ac.uk
  check_ancestor

# Vanilla list explosion for anything which doesn't match prefix or suffix

lists_process:
  driver                = accept
  domains               = lists.cam.ac.uk
  require_files         = LISTS/members/$local_part
  retry_use_local_part
  transport             = list_pipe

lists_error:
  driver                = redirect
  domains               = lists.cam.ac.uk
  data                  = :fail: \
        "${local_part}" is not a list that is managed on this system.
  allow_fail

#############################################################################
#                     TRANSPORTS CONFIGURATION                              #
#############################################################################

begin transports

# This transport is used for delivering messages over SMTP connections.
# We do not use TLS to send email.

smtp:
  driver                = smtp
  hosts_randomize
  hosts_avoid_tls       = *

# This transport is used when delivering messages to Hermes by LMTP
# The target machines do not appear in the DNS, hence gethostbyname.
# (actually appears to be redundant when parent router is manualroute
# rather than accept, but useful as documentation none the less.)
# We keep any local_part_suffix that was recognised by the router.

hermes_lmtp:
  driver                = smtp
  rcpt_include_affixes  = true
  gethostbyname         = true
  protocol              = lmtp

# This variant of the hermes_lmtp transport strips out any nul bytes in order
# to avoid triggering Cyrus's strict checking. We only use it when necessary
# for efficiency reasons.

hermes_lmtp_filter:
  driver                = smtp
  rcpt_include_affixes  = true
  gethostbyname         = true
  transport_filter      = /usr/bin/tr -d \\000
  protocol              = lmtp

# Mailing list exploder process

list_pipe:
  driver                = pipe
  command               = /opt/exim/sbin/explode_list
  message_prefix        = ""
  message_suffix        = ""
```

```
  return_fail_output
1079
########################################################################
1081 #                      RETRY CONFIGURATION                             #
########################################################################
1083
begin retry
1085
# Large time out for local mail servers so that problems can be fixed.
1087 # This also deals with quota problems on the Hermes LMTP message store.
# We have a short time-out for non-local addresses that get routed via
1089 # an A record (because they have no MX) because these are usually the
# result of fat-fingering.
1091
# Address               Error           Retries
1093 # -------               -----           -------

1095 *@+our_domains          *               F,2h,15m; F,8h,30m; F,7d,60m; F,14d,2h

1097 *@*                     refused_A       F,2h,15m; G,16h,30m,1.5
*@*             timeout_connect_A       F,2h,15m; G,16h,30m,1.5
1099
*@*                     *               F,2h,15m; G,16h,30m,1.5; F,5d,8h
1101
# End of Exim 4 configuration
```

9.2. Exim configuration on *hermes*

```
0001 # $Cambridge: hermes/conf/exim/etc/etc.hermes/configure,v 1.23 2004/11/23 14:00:01 fanf2 Exp $

0003 # This configuration relies as much as possible on the message
# submission server to do the clever stuff, including address
0005 # verification, SMTP error text, message fix-ups, and bounce address
# tagging. We give it a bit of help by passing across the user's
0007 # authenticated identity, if it's available.

0009 ########################################################################
#                    MAIN CONFIGURATION SETTINGS                       #
0011 ########################################################################

0013 qualify_domain                  = hermes.cam.ac.uk

0015 ## Privileged users

0017 deliver_drop_privilege          = true
never_users                     = root
0019 trusted_users                   = prayer

0021 ## Resource control

0023 # see also the cyrus LMTP limit and the smtp.hermes limit
#
0025 message_size_limit              = 25M

0027 ## Policy controls

0029 acl_smtp_rcpt                   = accept hosts = : @[] :

0031 # see discussion of authentication below
rfc1413_hosts                   = :
0033 rfc1413_query_timeout           = 0s

0035 ## Frozen, bounce, and warning messages

0037 auto_thaw                        = 24h
bounce_return_body
0039 bounce_return_size_limit        = 10K
errors_reply_to                 = postmaster@cam.ac.uk
0041
## Logging
0043
log_timezone                    = true
0045 message_logs                    = false
print_topbitchars               = true
0047
.ifdef DEBUG
0049 log_selector    = +all
.else
0051 log_selector    = -retry_defer -skip_delivery -host_lookup_failed \
                       +smtp_confirmation +delivery_size \
0053                   +sender_on_delivery +return_path_on_delivery \
                       +received_recipients +all_parents +address_rewrite \
```

```
                  +deliver_time +queue_time \
                  +smtp_protocol_error +smtp_syntax_error
.endif

# tweaked for consistency with ppsw
#
received_header_text = Received: \
        from ${if def:sender_rcvhost {$sender_rcvhost\n\t} \
                  {${if def:sender_ident {$sender_ident } {localhost } }\
                   ${if def:sender_helo_name {(helo=$sender_helo_name) } }} }\
        by $primary_hostname ${if def:interface_address \
                {(hermes.cam.ac.uk [$interface_address]:$interface_port)} \
                {(hermes.cam.ac.uk)} }\n\t\
        ${if def:received_protocol {with $received_protocol } }\
        ${if eq{prayer}{$sender_ident} {(PRAYER:$sender_address_local_part) } }\
        id $message_id (Exim $version_number)\
        ${if def:received_for { for $received_for} }\n\t\
        (return-path <$sender_address>)

#############################################################################
#                   AUTHENTICATION CONFIGURATION                            #
#############################################################################

begin authenticators

# This authenticator communicates MUA user authentication to the
# message submission server. If the user submitted the message
# via webmail, trust it to pass us the authenticated username.
# If the user wasn't authenticated, we will not authenticate.
#
# (Note that since prayer is a trusted user submitting a message on
# behalf of another user with -f, Exim does not set $authenticated_id
# so we have to use $sender_ident. This is OK so long as we don't do
# ident callbacks and/or don't accept messages remotely.)

EXTERNAL:
  driver                = plaintext
  client_send           = <| ${if eq{prayer}{$sender_ident} \
                                {$sender_address_local_part} \
                                {${if def:authenticated_id \
                                      {$authenticated_id} \
                                      fail }} }

#############################################################################
#                      ROUTERS CONFIGURATION                                #
#############################################################################

begin routers

smtp:
  driver                = accept
  transport             = smtp

#############################################################################
#                     TRANSPORTS CONFIGURATION                              #
#############################################################################

begin transports

# We have to limit the number of messages delivered down a connection
# because SMTP authenticates connections not messages, and we are
# authenticating on behalf of the message's sender not for ourself.

smtp:
  driver                = smtp
  hosts                 = smtp.hermes.cam.ac.uk
  hosts_try_auth        = smtp.hermes.cam.ac.uk
  hosts_randomize
  connection_max_messages = 1

#############################################################################
#                      RETRY CONFIGURATION                                  #
#############################################################################

begin retry

*               *               F,5d,5m

# End of Exim 4 configuration
```

9.3. Table of IP address parameters, `addrparams`

```
#
# The configuration parameters for certain IP addresses.
# Note that the total allocated range for ppsw systems is 129...159.
# Hermes machines have ad-hoc allocations. See hermes/doc/misc/ppsw.txt
#
# Non-ppsw addresses are listed here to provide especially favourable
# email relay service, to keep queues on ppsw not the other systems.
#
#       name            the role in which ppsw acts
#                       and name of TLS certificate
#       acl             suffix to use on acl names
#               smart   - basic smarthost functionality
#               submit  - RFC 2476 message submission service
#               mx      - border email gateway
#       benice          always accept email from the machine
#       domain          email domain to use in submit acl
#       msgsizelim      message_size_limit value
#       mua             if the machine is a trusted MUA server
#
# Trusted MUAs can use AUTH EXTERNAL to inform ppsw of the submitter's
# username so that bounce address tagging can be done correctly.
#
# $Cambridge: hermes/conf/exim/etc/etc.ppsw/tables/addrparams,v 1.4 2005/01/17 15:58:00 fanf2 Exp $
#
# Fixed virtual address.
#
131.111.8.129   name=ppsw.cam.ac.uk  acl=smart
#
# Standard service names.
#
131.111.8.130   name=ppsw.cam.ac.uk  acl=smart
131.111.8.131   name=ppsw.cam.ac.uk  acl=smart
131.111.8.132   name=ppsw.cam.ac.uk  acl=smart
131.111.8.133   name=ppsw.cam.ac.uk  acl=smart
131.111.8.134   name=ppsw.cam.ac.uk  acl=smart
131.111.8.135   name=ppsw.cam.ac.uk  acl=smart
131.111.8.136   name=ppsw.cam.ac.uk  acl=smart
131.111.8.137   name=ppsw.cam.ac.uk  acl=smart
131.111.8.138   name=ppsw.cam.ac.uk  acl=smart
131.111.8.139   name=ppsw.cam.ac.uk  acl=smart
#
131.111.8.140   name=mx.cam.ac.uk  acl=mx
131.111.8.141   name=mx.cam.ac.uk  acl=mx
131.111.8.142   name=mx.cam.ac.uk  acl=mx
131.111.8.143   name=mx.cam.ac.uk  acl=mx
131.111.8.144   name=mx.cam.ac.uk  acl=mx
131.111.8.145   name=mx.cam.ac.uk  acl=mx
131.111.8.146   name=mx.cam.ac.uk  acl=mx
131.111.8.147   name=mx.cam.ac.uk  acl=mx
131.111.8.148   name=mx.cam.ac.uk  acl=mx
131.111.8.149   name=mx.cam.ac.uk  acl=mx
#
131.111.8.150   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.151   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.152   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.153   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.154   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.155   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.156   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.157   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.158   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
131.111.8.159   name=smtp.hermes.cam.ac.uk  acl=submit  domain=hermes.cam.ac.uk  msgsizelim=25M
#
# Real and virtual addresses for Hermes
#
131.111.8.51    mua=yes         # hermes-1.csi.cam.ac.uk
131.111.8.54    mua=yes         # hermes-2.csi.cam.ac.uk
131.111.8.59    mua=yes         # hermes-v.csi.cam.ac.uk
131.111.8.66    mua=yes         # hermes-w.csi.cam.ac.uk
#
# Other machines to which we should be nice.
#
131.111.8.16    benice=yes      # canvas.csi.cam.ac.uk
172.28.13.60    benice=yes      # otanes.csi.private.cam.ac.uk
172.28.13.1     benice=yes      # cyrus-1.csi.private.cam.ac.uk
172.28.13.2     benice=yes      # cyrus-2.csi.private.cam.ac.uk
172.28.13.3     benice=yes      # cyrus-3.csi.private.cam.ac.uk
172.28.13.4     benice=yes      # cyrus-4.csi.private.cam.ac.uk
172.28.13.5     benice=yes      # cyrus-5.csi.private.cam.ac.uk
172.28.13.6     benice=yes      # cyrus-6.csi.private.cam.ac.uk
172.28.13.7     benice=yes      # cyrus-7.csi.private.cam.ac.uk
```

```
172.28.13.8     benice=yes      # cyrus-8.csi.private.cam.ac.uk
172.28.13.9     benice=yes      # cyrus-9.csi.private.cam.ac.uk
172.28.13.10    benice=yes      # cyrus-10.csi.private.cam.ac.uk
172.28.13.11    benice=yes      # cyrus-11.csi.private.cam.ac.uk
172.28.13.12    benice=yes      # cyrus-12.csi.private.cam.ac.uk
172.28.13.13    benice=yes      # cyrus-13.csi.private.cam.ac.uk
172.28.13.14    benice=yes      # cyrus-14.csi.private.cam.ac.uk
172.28.13.15    benice=yes      # cyrus-15.csi.private.cam.ac.uk
172.28.13.16    benice=yes      # cyrus-16.csi.private.cam.ac.uk
172.28.13.17    benice=yes      # cyrus-17.csi.private.cam.ac.uk
#
# EOF
```

9.4. Table of invalid destination networks, `bad_nets`

```
#
# Networks to which we will not send email via the lookuphost router.
# Note that this does not include email routed within Cambridge via
# the special_routes table, so CUDN-private addresses are listed here.
#
# See RFC 3330 for source material. Not all of the networks mentioned
# in it are appropriate for listing here.
#
# $Cambridge: hermes/conf/exim/etc/etc.ppsw/tables/bad_nets,v 1.4 2004/02/16 11:36:58 fanf2 Exp $
#
0.0.0.0/8
10.0.0.0/8
127.0.0.0/8
169.254.0.0/16
172.16.0.0/12
192.0.2.0/24
192.168.0.0/16
198.18.0.0/15
224.0.0.0/3
#
# EOF
```

9.5. Table of ppswitch host names, `ppswnames`

```
#
# List of all the possible names of ppsw, for hosts_treat_as_local.
#
# Note that ppsw.cam.ac.uk is handled separately
# because it is also a managed mail domain.
#
# $Cambridge: hermes/conf/exim/etc/etc.ppsw/tables/ppswnames,v 1.4 2004/11/15 15:03:25 fanf2 Exp $
#
mx.cam.ac.uk
smtp.hermes.cam.ac.uk
ppsw-v.csi.cam.ac.uk
ppsw-0.csi.cam.ac.uk
ppsw-1.csi.cam.ac.uk
ppsw-2.csi.cam.ac.uk
ppsw-3.csi.cam.ac.uk
ppsw-4.csi.cam.ac.uk
ppsw-5.csi.cam.ac.uk
ppsw-6.csi.cam.ac.uk
ppsw-7.csi.cam.ac.uk
ppsw-8.csi.cam.ac.uk
ppsw-9.csi.cam.ac.uk
ppsw-0h.csi.cam.ac.uk
ppsw-1h.csi.cam.ac.uk
ppsw-2h.csi.cam.ac.uk
ppsw-3h.csi.cam.ac.uk
ppsw-4h.csi.cam.ac.uk
ppsw-5h.csi.cam.ac.uk
ppsw-6h.csi.cam.ac.uk
ppsw-7h.csi.cam.ac.uk
ppsw-8h.csi.cam.ac.uk
ppsw-9h.csi.cam.ac.uk
ppsw-0m.csi.cam.ac.uk
ppsw-1m.csi.cam.ac.uk
ppsw-2m.csi.cam.ac.uk
ppsw-3m.csi.cam.ac.uk
ppsw-4m.csi.cam.ac.uk
ppsw-5m.csi.cam.ac.uk
ppsw-6m.csi.cam.ac.uk
ppsw-7m.csi.cam.ac.uk
ppsw-8m.csi.cam.ac.uk
ppsw-9m.csi.cam.ac.uk
#
# EOF
```

9.6. Example entries from other tables

`cudn_nets`:
This contains the list of address ranges used on the Cambridge University Data Network, including:

131.111.0.0 / 16
129.169.0.0 / 16
128.232.0.0 / 16

`domainlist`:
This contains the list of managed mail domains, including the four special-case domains listed to the right of the examples below:

law.cam.ac.uk	*cam.ac.uk*
quns.cam.ac.uk	*hermes.cam.ac.uk*
ucs.cam.ac.uk	*lists.cam.ac.uk*
conferencecambridge.com	*ppsw.cam.ac.uk*

`longshort`:
This is a mapping from long-form domains to their corresponding short forms, for example:

queens.cam.ac.uk`:` *quns.cam.ac.uk*

`nocallout`:
This contains a list of email addresses and domains, and the domains may be wildcarded. For example:

httpd@host.name.of.web.server
domain.rejecting.all.bounces
**.all.subdomains.broken*

`relay_domains`:
This just contains **.cam.ac.uk* and *cambridge.org*.

`special_routes`:
This is effectively a `manualroute` router `route_list`. It contains entries like:

cl.cam.ac.uk	*mta.cl.cam.ac.uk*	`bydns`
emma.cam.ac.uk	*mail.emma.cam.ac.uk*	`bydns`
srcf.ucam.org	*student.cusu.cam.ac.uk*	`bydns`

References

1. David Carter and Tony Finch, *Scaling up Cambridge University's email service* (Feb 2004). `http://www.cus.cam.ac.uk/~fanf2/hermes/doc/talks/2004-02-ukuug/`
2. "Mail to and from Phoenix," *University of Cambridge Computing Service Newsletter,* 164 (Mar/Apr 1992). `http://www.cam.ac.uk/cs/newsletter/1992/nl164.html#19`
3. *ISO Development Environment.* `ftp://ftp.uu.net/networking/osi/isode/`
4. "Closure of Phoenix," *University of Cambridge Computing Service Newsletter,* 183 (Oct 1995). `http://www.cam.ac.uk/cs/newsletter/1995/nl183/phoenix.html`
5. "The Hermes Message Store," *University of Cambridge Computing Service Newsletter,* 172 (Oct 1993). `http://www.cam.ac.uk/cs/newsletter/1993/nl172.html#31`
6. "New mail list software available," *University of Cambridge Computing Service Newsletter,* 178 (Oct 1994). `http://www.cam.ac.uk/cs/newsletter/1994/nl178.html#45`
7. "Computing Service Mail Addresses," *University of Cambridge Computing Service Newsletter,* 171 (Aug 1993). `http://www.cam.ac.uk/cs/newsletter/1993/nl171.html#25`
8. "New @cam Addresses," *University of Cambridge Computing Service Newsletter,* 178 (Oct 1994). `http://www.cam.ac.uk/cs/newsletter/1994/nl178.html#40`
9. Greg A. Woods, *The Smail Project.* `http://www.weird.com/~woods/projects/smail.html`
10. "Managed Mail Domains," *University of Cambridge Computing Service Newsletter,* 192 (Jul 1997). `http://www.cam.ac.uk/cs/newsletter/1997/nl192/services.html`

11. "Hermes mailing lists," *University of Cambridge Computing Service Newsletter*, 195 (Feb 1998). http://www.cam.ac.uk/cs/newsletter/1998/nl195/services.html#s5
12. "Spam and virus filtering," *University of Cambridge Computing Service Newsletter*, 216 (Apr 2003). http://www.cam.ac.uk/cs/newsletter/2003/nl216/mail.html#1
13. John C. Klensin (ed), "Simple Mail Transfer Protocol," RFC 2821 (Apr 2001). http://www.ietf.org/rfc/rfc2821.txt
14. John G. Myers, "SMTP Service Extension for Authentication," RFC 2554 (Mar 1999). http://www.ietf.org/rfc/rfc2554.txt
15. John G. Myers, "Simple Authentication and Security Layer (SASL)," RFC 2222 (Oct 1997). http://www.ietf.org/rfc/rfc2222.txt
16. Chris Newman, "Using TLS with IMAP, POP3 and ACAP," RFC 2595 (Jun 1999). http://www.ietf.org/rfc/rfc2595.txt
17. Paul Hoffman, "SMTP Service Extension for Secure SMTP over Transport Layer Security," RFC 3207 (Feb 2002). http://www.ietf.org/rfc/rfc3207.txt
18. Randall Gellens and John C. Klensin, "Message Submission," RFC 2476 (Dec 1998). http://www.ietf.org/rfc/rfc2476.txt
19. *Mail program settings for Hermes*, University of Cambridge Computing Service. http://www.cam.ac.uk/cs/email/muasettings.html
20. Tim Showalter, "Sieve: A Mail Filtering Language," RFC 3028 (Jan 2001). http://www.ietf.org/rfc/rfc3028.txt
21. The Spamhaus Project, *Should MCI Be Profiting From Knowingly Hosting Spam Gangs?* (Feb 2005). http://www.spamhaus.org/news.lasso?article=158
22. Julian Field, *MailScanner.* http://www.mailscanner.info/
23. Tomasz Kojm, et al., *Clam Anti-Virus.* http://www.clamav.net/
24. Network Associates Technology, Inc., *McAfee VirusScan Command Line Scanner for Linux.* http://www.networkassociates.com/us/products/mcafee/antivirus/desktop/vs_commandline.htm
25. *The Apache SpamAssassin Project.* http://spamassassin.apache.org/
26. Peter W. Resnick (ed), "Internet Message Format," RFC 2822 (Apr 2001). http://www.ietf.org/rfc/rfc2822.txt
27. Oxford University Computing Services, *Oxford Email Addresses.* *http://www.oucs.ox.ac.uk/email/oxford/index.xml.ID=body.1_div.3*
28. Internet Assigned Numbers Authority, "Special-Use IPv4 Addresses," RFC 3330 (Sep 2002). http://www.ietf.org/rfc/rfc3330.txt
29. Internet Corporation For Assigned Names and Numbers, *Verisign's Wildcard Service Deployment.* http://www.icann.org/topics/wildcard-history.html
30. "@cam addresses and the online e-mail directory," *University of Cambridge Computing Service Information Sheets*, 29 (Oct 2004). http://www.cam.ac.uk/cs/docs/infosheets/is29/
31. John G. Myers, "Local Mail Transfer Protocol," RFC 2033 (Oct 1996). http://www.ietf.org/rfc/rfc2033.txt
32. "User-driven mailing lists," *University of Cambridge Computing Service Leaflets*, G90 (Sep 2003). http://www.cam.ac.uk/cs/docs/leaflets/g90/
33. David Madore, *A page about quines* (Dec 2002). http://www.eleves.ens.fr:8080/home/madore/computers/quine.html
34. Tony Finch, *Protecting against email forgery in Cambridge* (Jul 2004). http://www.cus.cam.ac.uk/~fanf2/hermes/doc/antiforgery/cam.txt
35. *Certified Server Validation.* http://mipassoc.org/csv/index.html
36. Free Software Foundation, *Mailman, the GNU Mailing List Manager.* http://www.gnu.org/software/mailman/
37. University of Cambridge Computing Service, *The Raven web authentication service.* http://www.cam.ac.uk/cs/raven/
38. Tony Finch, *A Hermes "thermal event"*. (Jan 2004). http://www-uxsup.csx.cam.ac.uk/~fanf2/hermes/doc/misc/orange-fire/