OpenSMTPD : We deliver!

Éric Faurot
eric@openbsd.org

February 8, 2013

Abstract

In this paper we present the OpenSMTPD daemon: a simple, modern and portable mail server implemented using privilege-separation and messaging passing. Among different features, it comes with a notably simple configuration file format, and it offers very powerful deployment options.

We describe the internal organisation of the daemon in different processes with very specific roles. We examine the workflows for the main server tasks: enqueueing mails from external sources, delivering to the local users, relaying to external host and generating bounces. Finally, we discuss the server modularity, especially the table and backend APIs.

1 Introduction

Although several mail server implementations exist, they are not always satisfying for various reasons: complexity of the configuration, aging design which make it difficult to add support for new features, or inappropriate licensing terms.

The aim of the OpenSMTPD project is to provide a simple, robust and flexible implementation of the SMTP protocol, as defined in by RFC 5321[2] and other related RFCs. It is available under the liberal ISC license. It is being developed as part of the OpenBSD project. The development has started a few years ago, and has been very active in the last months. This paper presents an overview of the OpenSMTPD daemon design.

The first section will describe the configuration principles. In the next section we present the internal design of the daemon, based on privileged-separation and message-passing. The following section illustrates the workflow for the five main tasks : enqueueing, scheduling, delivering, relaying and bouncing. In the fourth section we will briefly discuss the flexibility of the daemon through filters and backends.

2 Features and Usage

2.1 Configuration

One of the most distinctive feature of OpenSMTPD is the simplicity of the configuration file, especially in comparison with other alternative mail server implementations. It can describe complex setups in a clear and concise manner, which makes the maintenance easy for the administrator, and helps to prevent misconfigurations with unexpected side-effects. Basic examples of configuration examples are described here.

The philosophy and syntax is largely inspired by the pf[1] configuration. It is based on the definition of a rule set: each *input* passes through the rule-matching engine to decide what action to take. Unlike pf, OpenSMTPD uses a first-match wins strategy.

```
listen on lo0
accept for local deliver to mbox
accept for any relay
```

The first line tells the server to listen for SMTP connections on the loopback interface. Then comes the rule-set definition. The first rule matches mails sent for the local machine, which means that the destination domain is the machine hostname (or localhost). The action to take is to deliver the mail to the user mbox. In this case, the user part of the destination address is expected to be a local user. The second rule matches all mails regardless of the destination domain, and use standard relaying for all domain.
In this example, the rules will implicitly reject mails not originating from the local machine. The updated configuration which follows allows to also listen on external interfaces (egress group), and accept mails from external sources for local deliveries using an alias file. Relaying to external domain is still there but only for mails originating from the local machine :

```
listen on lo0
listen on egress

table aliases file:/etc/mail/aliases

accept from any for local alias <aliases> deliver to mbox
accept for any relay
```

A common use-case is to route mail for outside through a specific host. The following example is a typical setup for a home user relaying its mail through his ISP, with authentication.

```
listen on lo0

table aliases file:/etc/mail/aliases
table secrets file:/etc/mail/secret

accept for local alias <aliases> deliver to mbox
accept for any relay via tls+auth://myisp@smtps.my.isp auth <secrets>
```

2.2 Administration

The administrator may interact with the daemon using a simple `smtpctl` control program. This program connects to the daemon control socket, turns the user request into specific internal control messages, forwards them to the control process and reports the result. Currently, the program supports the following operations:

- inspecting the current state of the queue: list of messages and scheduling information,
- pausing or resuming subsystems such as listeners for incoming mails, outgoing transfers or local deliveries,
- scheduling pending messages for immediate delivery,
- removing specific messages,
- retrieving various internal statistic counters,
- monitoring the server activity in real time.

2.3 Other features

Only a few simple cases have been described here, to give an overview of the philosophy behind the OpenSMTPD configuration. It supports many other features that are expected from any decent SMTP server implementation, among which :

- support for TLS and SMTPS,
- user authentication,
- backup server for a domain,
- primary and virtual domains,
- completely virtual users,
- local delivery to mbox, Maildir or external program.

The following example, adapted from a real-world configuration, shows a more complex setup using many of the OpenSMTPD features. The details won't be discussed here, but it shows how very powerful setups can be achieved, while the configuration file remains reasonably easy to read.

```
listen on lo0
#   Tag traffic from the local DKIM signing proxy
listen on lo0 port 10029 tag DKIM
#   Listen for incoming mail on standard smtp port and smtps port with ssl.
listen on egress ssl certificate mail.mydomain.org
#   Listen on a different port, enable tls and require auth
listen on egress port submission tls certificate mail.mydomain.org auth

table sources           { xxx.xxx.xxx.44, xxx.xxx.xxx.45 }
table aliases           "/etc/mail/smtpd/aliases"
table aliases2          "/etc/mail/smtpd/aliases2"
table pdomains          "/etc/mail/smtpd/primary-domains"
table vdomains          "/etc/mail/smtpd/virtual-domains"
table vusers            "/etc/mail/smtpd/virtual-users"
table bdomains          "/etc/mail/smtpd/backup-domains"

#   Deliver local mails with aliases
accept for local alias <aliases> deliver to maildir

#   Accept external mails for our primary domains.
accept from any for domain <pdomains> alias <aliases> deliver to maildir

#   Accept external mails for a specific primary domain with other aliases.
accept from any for domain "other.dom" alias <aliases2> deliver to maildir

#   Virtual domains, with a mapping from virtual users to real mailboxes
accept from any for domain <vdomains> virtual <vusers> deliver to maildir

#   Act as a backup server for the given domains, only relay to servers with
#   lower MX preference than the one specified.
accept from any for domain <bdomains> relay backup mx1.mydomain.org

#   Relay all signed traffic, using multiple source addresses (round robin).
accept tagged DKIM for any relay source <sources>
#   Relay outgoing mails through a DKIM signing proxy.
accept for any relay via smtp://127.0.0.1:10028
```

3 Internal Design

3.1 Fundamental concepts

The most important "object" around which the OpenSMTPD daemon is organised is the *envelope*. An envelope describes a message that is in transit within the server. It is always associated with a body, or content. Different envelopes can be associated with the same underlying content, to form what we refer to as a *message*, as illustrated in figure 1.

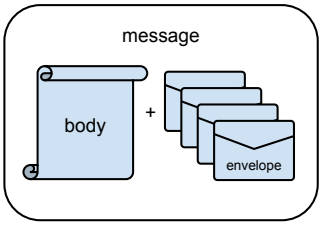

Figure 1: A message

An envelope has an origin, the sender, which is shared between all envelopes within the same message. An envelope also has a single destination, the recipient. A message sent to multiple recipient will have one envelope for each of them. The envelope also contains administrative information like the creation time, the expiry time, some internal flags, information on the enqueueing, etc. It also contains routing information, which describes how the associated content is to be transfered or delivered. An envelope is uniquely identified with its envelope ID, a 64 bits number that is generated internally.

Envelopes are created by and SMTP clients and the associated messages are stored (queued) temporarily in the server, until they can reach their next or final destination. There are three types of envelopes: local envelopes which are to be delivered to a local user on the host, relay envelopes, which must be transfered to another host, and bounce envelopes which are internal, and are meant to generate a report.

The OpenSMTPD server is implemented as a set of dedicated processes communicating using the ISMG(3) framework, and working together, exchanging envelopes, to drain accepted messages out. This design has several advantages. Defining specific roles and isolating functionalities in different processes lead to a design that is globally simpler to understand, and more reliable in the sense that it prevents layer violation. The processes can also run at different privilege levels, providing extra protection in case one of them (especially those directly exposed to the internet) is compromised.

Besides the short-lived ones that are temporarily forked, there are 9 processes :

- scheduler
- queue
- mail transfer agent
- mail delivery agent
- mail filter agent

- lookup agent
- smtp
- control
- parent

In the rest of the section we describe the different processes and their role in the system.

3.2 Process layout

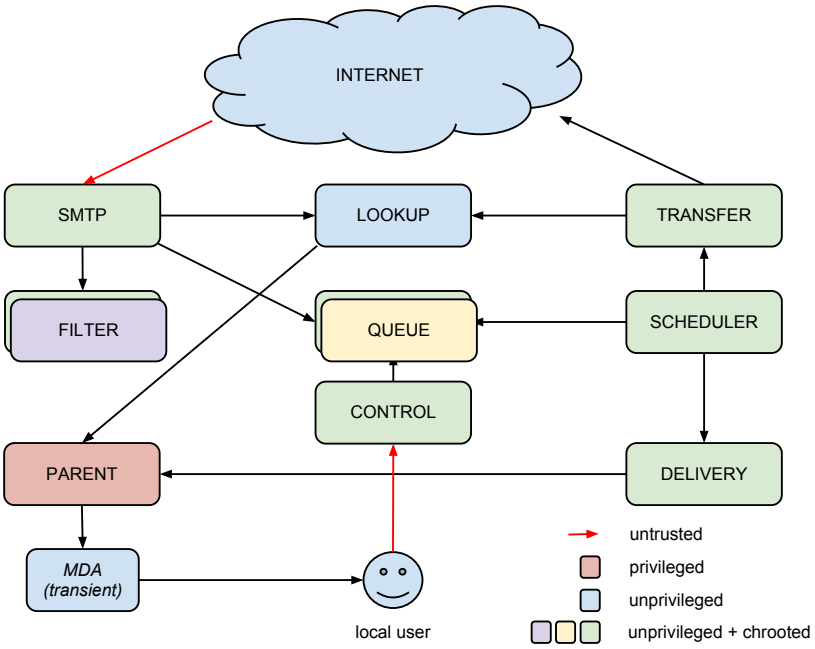

Figure 2: OpenSMTPD process layout

3.2.1 Scheduler

The scheduler is the process that knows all existing envelopes, and which takes the decision of scheduling for delivery, relaying, expiring them. It does not know the full detail of the envelopes, but only the necessary scheduling information: creation date, expiry date, retry count and a few flags which are required to take scheduling decision.

The scheduler only maintains a runtime state: it gets notified about new envelopes, dispatch work to be done when an envelope is schedulable, and wait for notification of the delivery outcome, to update the envelope runtime state.

It runs fully unprivileged and chrooted to an empty directory.

3.2.2 Queue

The queue process is responsible for storing envelopes and messages on permanent storage reliably, and reloading them on demand. For practical reason it is also the process that re-inject bounced envelopes. When the daemon starts, the queue process is also responsible for reloading existing

envelopes off permanent storage and providing them to the scheduler. It is the only process that is supposed to maintain an offline state.

One important duty of the queue process for reliability is to make sure that incoming messages and envelopes have reached permanent storage before being acknowledged, and that they are not removed or altered before being confirmed as delivered.

The queue process runs unprivileged, and chrooted into the smtpd spool directory.

3.2.3 SMTP

The SMTP process implements the SMTP protocol. It manages clients connections and push incoming messages to the rest of the system. As for the rest of the daemon, it runs fully event-based. Due to its design which relies heavily on fd-passing, the daemon is very sensible to file descriptor exhaustion situations. To prevent this from happening, which would impair the already running sessions, the daemon automatically disable input sockets when system limits are reached or close to be. It also implements simple counter-measures against clients that hog a connection by not doing any useful work, i.e. actually delivering mails.

It receives the listening sockets through dedicated imsg from the parent process, which already bound them to reserved ports. So it can run fully unprivileged and chrooted to an empty directory.

3.2.4 Mail transfer agent

The mail transfer agent (mta), is the process that handles relaying to other hosts.

The process managed destination relays to which mails must be sent. It tries to establish connections to these relays and drain pending messages. It handles connections limit, such as the number of mails sent per session, the number of recipients per mail, the number of parallel connections to the same host or domain, the maximum number of connections.

Like the SMTP process, it runs unprivileged and chrooted to an empty directory.

3.2.5 Mail delivery agent

The mail delivery agent (mda) is responsible for managing local deliveries. It does not do the delivery itself, but asks the parent privileged process to execute the actual mail delivery program on behalf of the local user to which a mail is destinated. The process makes sure to limit the number of running delivery processes on a global and per-user basis.

This process also runs unprivileged and chrooted to an empty directory,

3.2.6 Lookup agent

The lookup agent (lka) acts as a proxy for all kinds of lookups that other processes wants to do: credentials for authentication, user information, DNS resolving, which is a very important part of an SMTP daemon. The DNS queries are done asynchronously using an async variant of the resolver API. Multiple queries can run in parallels within the same process, which avoids having to block the whole process or rely on cumbersome work-arounds like thread pools or forks.

The lookup agent is also the process that does the rule-set evaluation for incoming envelopes, as we will see in the next section.

The main reason for having a separate process doing the queries is isolation: it allows other

processes to run with very constrained environment. The lookup agent runs unprivileged, but is not chrooted since it must be able to access system resources like /etc/resolv.conf.

3.2.7 Mail filter agent

The mail filter agent (mfa) manages the pluggable filter processes and drives the filtering process. It maintains a chain of filters through which SMTP requests and events are passed for processing, before being sent back to the SMTP process.

3.2.8 Control

The control process is the one that listens on the UNIX socket for request `smtpctl` program. It forwards requests from the administrator to the relevant process: pausing/resuming some agent, force the scheduling or removal of a message, get information about the runtime state of a message or envelope, etc, and deliver the responses to the client.

It is also responsible for collecting counter updates. All processes can report information about their internal activity for statistical analysis or runtime monitoring, in the form of named counter increments/decrements. For example, the number of active clients connections or queued envelopes.

3.2.9 Parent

This is the master process. It is the only process that needs to run as superuser. After the configuration file is loaded, it forks all the above processes, send them their runtime configuration, and waits for queries from other processes.

This process can perform the following privileged operations on behalf of other process: open a user .forward file for the lookup agent when expanding envelopes, perform local user authentication for the smtp process, execute or kill an external delivery program on behalf of the mail delivery process.

4 Operational workflows

We will describe here the internal exchanges that occurs to fulfill different services that the server must provide.

4.1 Enqueueing

The enqueueing task consists in accepting messages from outside. The sequence is depicted in figure 3. We suppose that all SMTP negotiation is already done, and the client wants to start a new transaction.

When a new MAIL FROM command is received it tells the queue process to create a new incoming message, and gets a unique message id in return. For each recipient (RCPT TO), a query is sent to the lookup agent, which tries find a matching rule. If found the recipient is *expanded* according to that rule (by looking for aliases or forward). The complete expand process is not detailed here, but it creates one or more final envelopes corresponding to the given recipient. These envelopes are sent to queue which stores them for the incoming message, and forward them to the scheduler. When all envelopes for a recipient are expanded, the SMTP process is notified that the

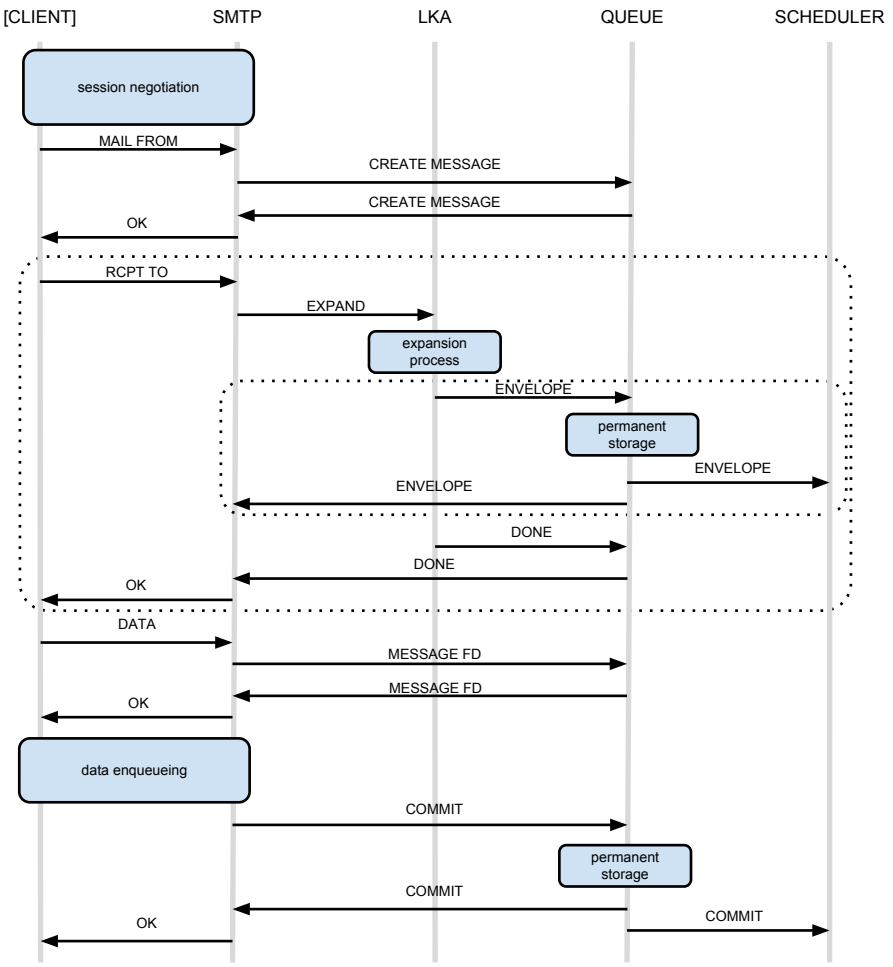

Figure 3: Enqueueing sequence diagram

RCPT is accepted. The client can then issue new RCPT request, repeating the expand sequence.

When the client wants to start sending the message content (using the DATA command), the SMTP process request a file descriptor from the queue process for writing the incoming message. When all data is written, it asks the queue to *commit* the message, which means moving it from incoming to permanent queue, and notifying the scheduler. When the SMTP receives the confirmation from the queue that the message is committed, it notifies the client.

If anything fails at some point of the process, the incoming message is cancelled. The sequence ensures that the envelopes and message have always reached permanent storage before the client and the scheduler are notified. So the whole process is fully transactional.

4.2 Scheduling

The scheduler receives new envelopes for incoming envelopes from the queue process. When a message is committed, it starts scheduling the associated envelopes. The life-cycle of these envelopes in the scheduler is shown on diagram 4.

The scheduler creates batches of schedulable envelopes id which are sent to the queue process

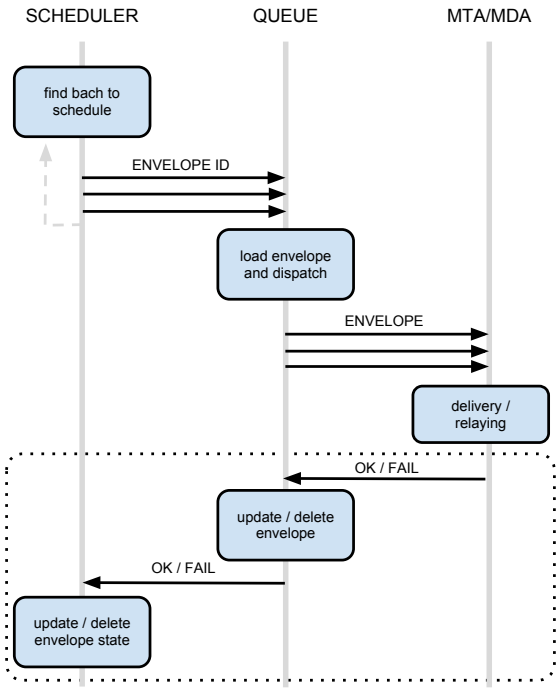

Figure 4: Scheduling sequence diagram

for delivery or relaying, and marked as "in-flight". The queue process loads the full envelopes and it dispatches them between the two mda an mta agent, depending on the envelope type.

When the agent is done with an envelope, it reports the outcome to the queue process, which updates the envelope state or delete it on success. The scheduler is then notified, so that it can update the state of the "in-flight" envelope accordingly: either remove it, or re-schedule later in a new batch.

4.3 Local deliveries

The detail of the local delivery operation is depicted on figure 5. When a message is to be delivered locally, the delivery process first retrieves the user information from the lka. It queries the queue process to obtain a file descriptor to read the message content from. When received, it sends a fork request with given user info to the parent process, which returns a file descriptor to write the message body to. When done, it waits for the notification from parent that the forked mda process has correctly exited, and if so, it reports the envelope as delivered to the queue process, which removes the envelope and informs the scheduler. In case of error, a failure report is sent to the queue.

4.4 Relaying

Relaying is the process by which responsibility for a message in the queued is transfered to another server, normally through the SMTP protocol. The internal interaction between smtpd process is shown on figure 6.

When receiving a envelope to relay, the transfer process first determines the relay to use. If necessary, it requests the credentials from the lookup agent. It also requests the list of MX hosts to use. These information are cached as long as the relay is currently referenced, so new envelopes

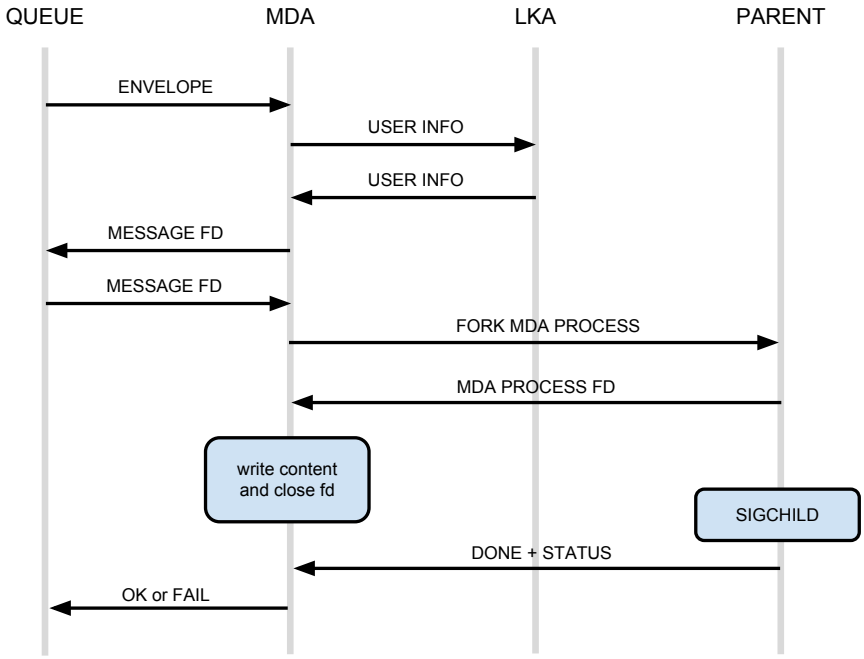

Figure 5: Local delivery sequence diagram

for the same relay in the mean time will not trigger these queries again. Received envelopes are grouped into tasks (messages for the same relay and for the same message) and queued on the relay for delivery.

When ready, the mail transfer agent will try to establish connections for the relay by trying to connect the MXs. Once a session is established, the reverse DNS entry is searched and the initial SMTP negotiation happens. At this point the agent might want to get the specific HELO string to use from the lka. When the session is ready to send mails, it dequeues the next task, retrieves the file descriptor for the message body from the queue, and starts the SMTP transfer. When the remote host has acknowledged (or rejected) the message transfer, the queue is notified for each envelope. The session can than proceed to the next task, if any.

If no connections could be established at all, all envelopes for that relay are sent back to the queue process with a failure report.

4.5 Bounces

Bounces are report messages that are sent by the mailer daemon to report temporary or permanent failure when trying to deliver a mail it has accepted. There are two kinds of bounces which can occurs: either a delay notification that is generated when an envelope has been queued for a certain time, or a failure report when the envelope has been refused or has expired, in which case the associated envelope is discarded.

A bounce is always associated to a message, it is internally handled as a special type of envelope for that message. It is created by the queue process as any other envelopes, expect that it is usually created on an already committed message (as opposed to incoming). Once created, the bounce envelope is sent to the scheduler, as any other envelope.

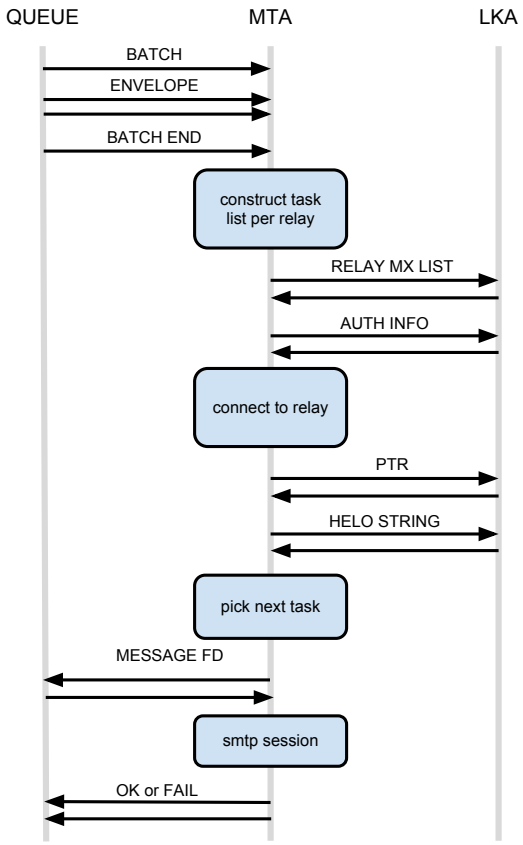

Figure 6: Relaying sequence diagram

It is processed by the queue process which will create regular envelope targeted at the original sender (return path) with a report and the bounced message. So the queue process request an internal socket from the smtp process to establish a local SMTP session, and re-inject the bounce message using the SMTP protocol. It creates a message with a single envelope, which is the original sender for the bounced envelope, and the body of the message contains the failure report for the original recipient.

Bounces are grouped when possible: when receiving a bounce envelope to re-inject, the queue process will delay the SMTP session lightly waiting for other bounces for the same message. The figure 7 illustrates the successive states of the queue when bouncing and discarding two envelopes for the same message.

5 Modularity and deployment options

Besides the flexibility of the configuration which allows the creative minds to easily build complex processing chains (relaying outgoing mail through a DKIM proxy is a 4 liners), OpenSMTPD provides two mean of extending its functionalities: input filters and internal backends.

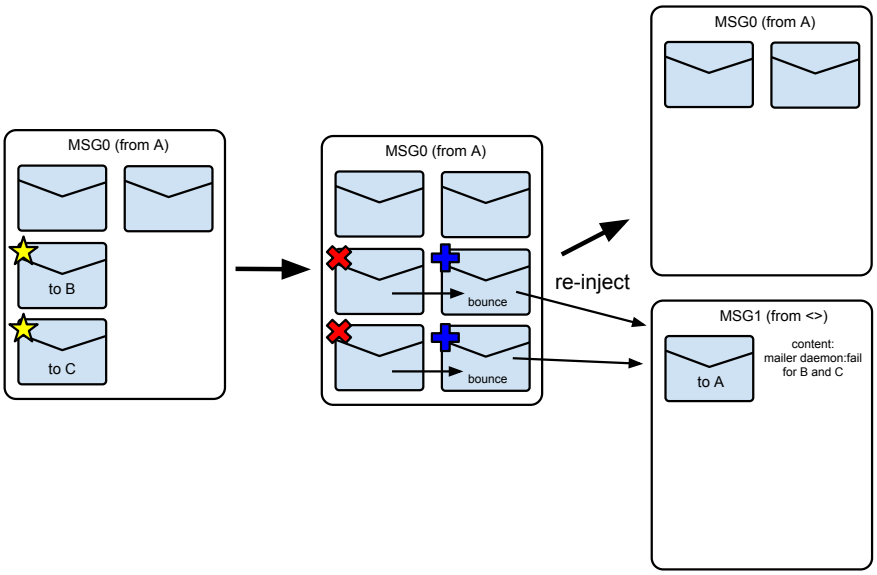

Figure 7: Bounce creation and re-injection

5.1 Filtering

The goal is to provide a powerful way to let external programs filter the incoming SMTP sessions and messages, in order to reject some possibly altering the recipients and/or the message contents.

A filtering mechanism has been in place since the beginning, but it has not been a high priority until more fundamental parts of the daemon were in place. It has been completely re-designed recently to fix some early limitations.

Filters are implemented as external programs which are started with the daemon. They are bound to the `mfa` process, and communicate with it via dedicated IMSG. The filter will registers various *hooks* for events or queries it wants to intercepts. An API and library is provided for writing filters, hiding the internal imsg machinery.

At the moment, the integrated filter support is still under development, and not officially supported yet and will not be part of the first release. It is however possible achieve various kind of filtering using a proxy loop: incoming mails are relayed through a specific SMTP proxy, such as a DKIM signing program or spam filter, and re-injected on a specific tagged listener.

A specific `milter` glue filter is planned, to allow using filters implemented with the milter API to run with OpenSMTPD. The ability to filter outgoing mail (at the transfer agent level) is also planned but not implemented yet.

5.2 Backends

Whereas filters operates on users inputs to alter the smtpd behaviour, the backend mechanism is a mean to completely change the deployment strategy.

Various parts of the daemon internally relies on abstractions for specific operations, which are defined as concise and strict API, which are implemented by *backends*. From the point of view of the calling process, this is completely hidden, so any implementation of these interfaces with

the required semantics, can be used. This has two advantages: it prevents layer violation in the design, and it makes the implementation very flexible, once the correct abstractions are found. The three most important backends are presented here.

5.2.1 Table backends

Almost all lookups in the daemons are done using the *table* abstraction. Depending on the context, a table can be either a set, or an associative array. The types of the elements also depends on the context. Tables are said to provide one or more *service*, and are implemented by a backend. The different services used are :

- ALIAS: association of a user name to mail aliases, used for alias lookups.
- DOMAIN: list of domain names: used to match destination or source domains.
- CREDENTIALS: association of a label to a pair of username/password, for SMTP authentication.
- NETADDR: list of network addresses, used for rule matching on the connection.
- USERINFO: association of a user name to its system uid/gid and home directory.
- SOURCE: list of local IP addresses, used by the MTA for outgoing connections.

The existing backends are :

- static: in memory content, or loaded from a file,
- db: using a db(3) database,
- getpwnam: this special backends implements only the credentials and userinfo service,
- ldap and sqlite: those are experimental backends

Any table backend able to implement a given service can be used in all contexts where this service is expected. It is then trivial to run completely virtual mail servers.

5.2.2 Scheduler backend

The scheduler backend API only have a few function to insert new envelopes fetch the list of envelopes to schedule, which are dispatched to the relevant agents, and update their state depending on the delivery outcome. There is a couple of administrative functions too. The current scheduler operates entirely in memory.

5.2.3 Queue backend

The queue process relies on a storage backend which role is to implement a set of well-defined atomic operation on envelopes and messages. The default out-of-the box backend uses bucket-based disk storage and rely on file-system operations. OpenSMTPD also ships with a RAM-based queue, which keeps every everything in memory. Of course, the storage is not permanent anymore, but it is a good way for testing things, and it is also useful, when designing the API and implementation, to make sure that the abstractions make sense and that no dangerous assumptions are made on the queue implementation.

As another Proof-of-Concept, we have written a completely alternative storage backend for the queue , which used ReST queries to store envelopes on an external *cloud* storage. The performances were of course terrible in the example setup, but the reliability of the whole delivery process was intact, since every thing is still transactional.

6 Conclusion

OpenSMTPD provides a simple and powerful implementation of a mail server, with a relatively small codebase. The design principles behind it make it very easy to use and deploy in a large range of environments. Performances issues have not been discussed here. It is a bit early to focus on performance tuning. Nonetheless, early tests shows that the server behaves correctly under load, and is able to process envelopes at very satisfying rate.

The daemon is actively maintained. Primary development is done on OpenBSD, and a portable version for other UNIX systems is provided too. It's been tested successfully on various flavours of Linux, other BSDs, and MacOS-X.

Some features are not there yet: for example support for the PIPELINING extensions is missing, and we need a way to quarantine some messages/envelopes. However we believe that the current features are enough to handle most situations. We are now focusing on internal cleanups and testing. A first official release is to be expected very soon.

References

[1] Pf: The openbsd packet filter. `http://www.openbsd.org/faq/pf/index.html`.

[2] J. Klensin. Simple Mail Transfer Protocol. RFC 5321 (Draft Standard), October 2008.