MeTA1 README

Claus Aßmann

December 30, 2019

Contents

Chapter 1

Introduction to MeTA1

1.1 MeTA1 is a Message Transfer Agent

This distribution contains the source code for MeTA1 which implements a message transfer agent (MTA). It supports the Simple Mail Transfer Protocol (SMTP) as specified by RFC 2821 [Kle01] and various extensions, e.g., STARTTLS [Hof99], AUTH [Mye99], PIPELINING [Fre00], as well as other protocols, e.g., LMTP [Mye96].

MeTA1 is intended to be used as a secure and efficient mail gateway. It does not provide any mail content modification capabilities itself, e.g., masquerading of addresses or changing (addition, removal) of headers. However, those features are available via (policy) milters.

1.1.1 Main Components of MeTA1

MeTA1 is a modularized message transfer agent consisting of five (or more) persistent processes, four of which are multi-threaded. A central queue manager (QMGR) controls SMTP servers (SMTPS) and SMTP clients (SMTPC) to receive and send e-mails, an address resolver (SMAR) provides lookups in various maps including DNS for mail routing, and a main control program (MCP) starts the others processes and watches over their execution. The queue manager organizes the flow of messages through the system and provides measures to avoid overloading the local or remote systems by implementing a central control instance.

More information about each component will be given in the appropriate sections. Complete documentation and background information can be found in [Aßmb]. Section 7.1 describes the data flow in MeTA1, the following is a brief summary. Figure 1.1 shows the interaction of the various components and databases[1]. Incoming messages are accepted by an SMTP server (SMTPS) which stores the messages in the content database (CDB). The envelope information, i.e., sender and recipients, is stored by the queue manager in an incoming queue (IQDB) and written to disk to the incoming queue backup database (IBDB). For a delivery, the envelope information must be transferred into the active queue (AQ). The scheduler in QMGR takes recipient envelopes from AQ and creates transactions which are given to an SMTP client (SMTPC) for delivery. An SMTP client takes the transaction information and tries to send a message whose content is read from CDB. After a successful delivery attempt a record is written to IBDB that logs this information. The deferred envelope database (DEFEDB) is only used if a message

[1] the term *database* is used loosely here

cannot be delivered during the first attempt.

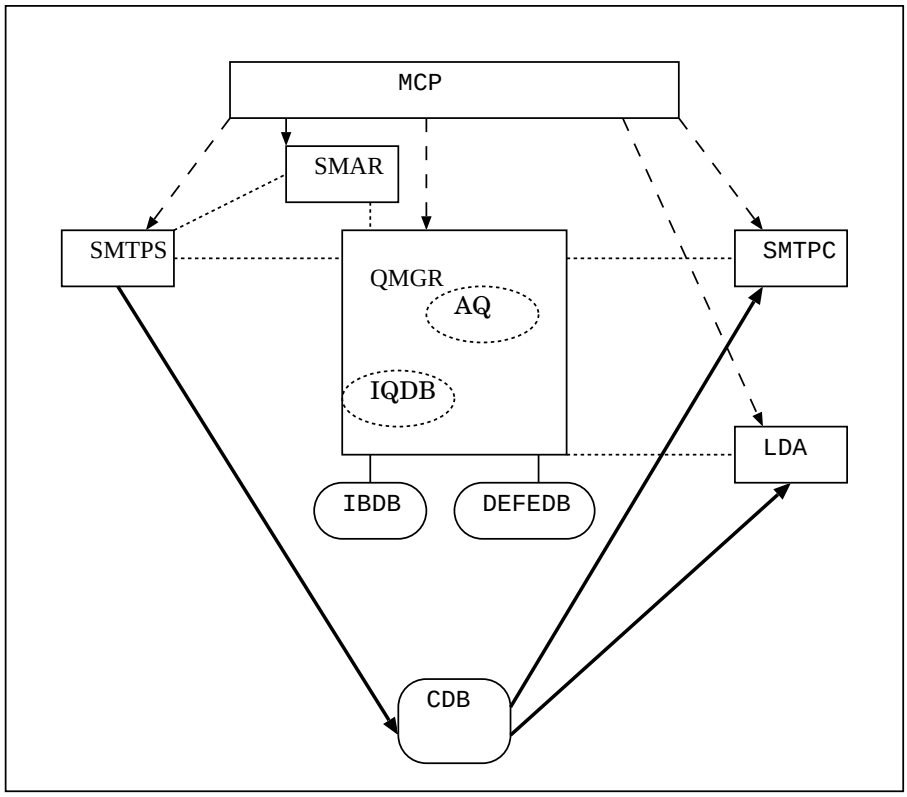

Figure 1.1: MeTA1: Overall Structure

This version of MeTA1 does not come with a local delivery agent nor a mail submission program. See Sections 4.3.1 and 4.2 which programs can be used to achieve the desired functionality.

1.2 Documentation

The document "sendmail X: Requirements, Architecture, and Functional Specification" [Aßmb] provides the background about the MeTA1 design, its architecture, as well as the functional specification, and details about the implementation.

1.2.1 Typographical Conventions

In this documentation, a command written as

```
$ command
```

should be executed as an unprivileged user. Only a command written as

```
# command
```

should be executed as the superuser.

If a command contains components that need to be replaced by values that depend on the environment or the local configuration, then it is usually written as a macro, e.g., `$LOGFILE`.

A number in parentheses behind a command or function refers to the manual section, e.g., `syslog(3)`, as it is usual for Unix documents.

1.3 Version

This document has been written for MeTA1-1.1.Alpha14.1, see also the greeting of the SMTP server and the version output of the main components. See Section 6.8 for information about version naming.

1.3.1 Providing Feedback

Please report bugs and provide feedback either to the developers list[Aßma] (if you are subscribed) or directly to[2]:

```
< MeTA1 + feedback (at) MeTA1 . org >
```

Feedback about the code, the documentation (including typographical, syntactical, and grammatical errors, pointing out parts that are not well enough explained, etc.), as well as patches and enhancements are highly appreciated.

1.4 For the Impatient

For those who do not want to read the entire documentation, it is adviced to read at least sections 2.2 and 2.4, and the appropriate section of Chapter 4.

[2]Sorry for the obfuscation, replace `(at)` with `@` and remove the spaces, but not the plus sign.

Chapter 2

Building, Testing, and Installing MeTA1

2.1 Verifying the Source Code Distribution

The source code is distributed as a (compressed) tar file and is accompanied by a PGP signature file which has the same name as the tar file plus the ending `.sig`. To verify the integrity of the source code PGP [PGP] or GPG [Gnu] are required as well as the MeTA1 PGP signing key [MeT]:

```
$ gpg --verify meta1-$VERSION.tar.gz.sig
```
or:
```
$ pgp meta1-$VERSION.tar.gz.sig meta1-$VERSION.tar.gz
```

Further information, especially about warnings or possible errors, can be found in the documentation for PGP or GPG.

2.2 Building MeTA1

MeTA1 uses a `configure` file generated by GNU autoconf for configuration. Hence you can build it (after verifying and unpacking the distribution) as follows:

```
$ mkdir obj.$OS && cd obj.$OS && $PATHTO/meta1-$VERSION/configure $OPTIONS \
  && make && make check
```

Obviously you have to replace `$OS`, `$VERSION`, `$OPTIONS`, as well as `$PATHTO`. It is also possible to build MeTA1 in the source tree, however, this is discouraged:

```
$ ./configure && make && make check
```

Notes: do *not* run this as `root`; this is not just a basic security measure (*only* use a privileged account if it is really required), but most of the programs refuse to run with `root` privileges. It might be useful to save the output of these commands[1] for later inspection.

[1] using `script(1)` or redirecting it to some file.

2.2.1 Compile-Time Configuration Options

Beside the usual `configure` options like `--prefix` a few MeTA1 specific configuration options are available:

`--enable-TLS` Enable check for STARTTLS support. The default is `yes`, i.e., `configure` tries to determine whether OpenSSL is available on the machine. Requires OpenSSL 0.9.8 or newer [Ope]. Note: check the OpenSSL website [Ope] for security announcement and be aware that due to the complexity of the software it may cause (security) problems.

`--enable-SASL` Enable check for AUTH support. The default is `yes`, i.e., `configure` tries to determine whether Cyrus SASL v2 is available on the machine. Requires Cyrus SASL version 2.1.18 or newer [Cyr]. Notes:

1. check `http://asg.web.cmu.edu/cyrus/` and `http://asg.web.cmu.edu/sasl/` for security announcement and be aware that due to the complexity of the software it may cause (security) problems.
2. If Cyrus SASL uses Berkeley DB then it is necessary that the version which has been used during compilation matches the version that it is linked against.

`--with-sasl-libdir=`path Path to directory containing Cyrus SASL v2 library.

`--with-sasl-incdir=`path Path to directory containing Cyrus SASL v2 include files.

`--disable-included-bdb` MeTA1 ships with a modified version of Berkeley DB 4.3.28 which is built and used by default. To use a different version of Berkeley DB (it must be 4.3, 4.2, or 4.1), e.g., one that is part of the host OS, specify `--disable-included-bdb`.

Notes:

1. If you do not use the Berkeley DB version that comes with MeTA1, make sure you run all the tests. For example, with Berkeley DB 4.2.50 on OpenBSD 3.2/i386 at least one of the test programs fails and hence this combination must *not* be used. Moreover, if you encounter a problem using some other BDB version then you must try to reproduce the problem with the shipped version before reporting a possible bug.
2. Do not use Berkeley DB 4.3.27/28 in 64 bit mode on Solaris 5.8/9 as it crashes at least in those configurations[2]. This bug is fixed in the version that is distributed with MeTA1.

`--with-bdb-libdir`=path Path to directory containing Berkeley DB library. This option is only needed if `--disable-included-bdb` is used and Berkeley DB is not installed in a location that the compiler or linker use by default. Note: `configure` currently checks only for a static library.

`--with-bdb-incdir`=path Path to directory containing Berkeley DB include files. This option is only needed if `--disable-included-bdb` is used and Berkeley DB is not installed in a location that the compiler uses by default. Example:

```
$ B=/usr/local/BerkeleyDB.4.3
$ $PATHTO/meta1-$VERSION/configure --with-bdb-libdir=$B/lib \
  --with-bdb-incdir=$B/include --disable-included-bdb
```

`--enable-pmilter` Enable policy milter protocol, see Chapter 5.

[2] "Private database environments on 64-bit machines no longer drop core because of 64-bit address truncation. [11983]" [Slea]

`--enable-msp` Enable a simple mail submission program (MSP) that is currently not supported (located in `contrib/`). This is just a helper program for those who do not want to install a different MSP but need only some basic functionality (which does *not* include a queueing mechanism). Note: this will install the MSP as `sendmail` thus overriding any existing program of that name (as well as a man page).

`--enable-DKIM` Enable DKIM [DKI] signing support in the SMTP server.

`--enable-tinycdb` Enable support for `cdb` map type, based on `tinycdb 0.75` [Tok].

To get the current list of configuration options, use `configure --help`.

2.3 Test Programs

```
$ make check
```

will run all test programs; currently those tests take about eighty minutes to run on a standard workstation. For each of the test programs one line is printed to denote whether the test succeeded, i.e., the output consists of lines with the marker `PASS:` or `FAIL:` and the name of the test program program. Additional output might be generated by the test programs themselves, e.g.,

`2 of 2 tests completed successfully`,

or some debug output. The debug output may even indicate an error, but only a final `FAIL:` indicates a test failure. Some tests depend on compilation options and are only conditionally enabled; others may depend on environment variables, see 2.3.1. For disabled tests `SKIP` is shown.

Since some of the tests may fail (see Section 2.3.2) and `make` will usually stop after encountering an error, it might be required to use

```
$ make -i check >check.out 2>&1
```

to perform all tests.

2.3.1 Environment Variables used by Test Programs

Environment variables can be used to disable some test programs if required or change the behaviour of some test programs. These environment variables and their effects are:

- `MTA_NO_DNS_TEST`: if not set to `false` then most tests which perform DNS lookups are disabled to avoid misleading failures. These lookups may use domains that are under control of the MeTA1 author.
- `MTA_DNS_TIMEOUT`: can be used to set a different timeout than the default, however, it may not be obeyed by all DNS test programs.
- `MTA_TEST_DNS_TIMING`: run DNS tests that are timing dependent and may fail under certain conditions (e.g., network too slow).
- `MTA_TIMING`: run MTA tests that are timing dependent and may fail under certain conditions.

- `MTA_STOPONERROR`: causes most test scripts that perform multiple checks to stop on the first error that occurs instead of performing all checks.
- `MTA_NO_LOG_TEST`: disables some tests that use `syslog(3)`.
- `MTA_NO_SLOW_TEST`: disables some tests that take a very long time.

In this example the DNS timeout is set to 60 seconds and tests that take a very long time are disabled:

```
$ MTA_DNS_TIMEOUT=60
$ MTA_NO_SLOW_TEST=1
$ export MTA_DNS_TIMEOUT MTA_NO_SLOW_TEST
$ make -i check
```

2.3.2 Known Test Program Problems

- `connctl.sh` will fail on systems that have neither `inet_pton(3)` nor `inet_aton(3)`. Fix: upgrade your OS or write a replacement function and put it into `librepl/`.
- `t-evthr-slp` can fail in some circumstances if the OS is busy with other tasks as it depends on the OS scheduler. Fix: just rerun the test.
- `t-evthr-sig.sh` fails on Linux systems that use a thread implementation that is not POSIX compliant. The test is currently disabled on all Linux versions. Note: if you know a simple way to figure out whether the OS actually provides POSIX compliant pthreads, please let me know.
- `t-hostname` fails on systems where `gethostname()` does not return any FQHN at all (e.g., default SunOS 4/5 installations). Add the FQHN as alias to `/etc/hosts` (see `hosts(5)` to solve this problem, e.g.,

  ```
  10.1.2.3    myname myname.my.domain
  ```

  or

  ```
  10.1.2.3    myname.my.domain myname
  ```

- `t-parsesockstr` fails on systems like AIX which treat an empty string as a valid IP address in `inet_addr(3)`.
- `t-mts-icr.sh` and `t-mts-ocr-?.sh` try to test incoming/outgoing rate control. They rely on the time it takes to send/receive mails which may not work on machines that are significantly slower or faster than the machines available to the author.
- `t-smar-0.sh`, `t-smar-3.sh`, and `t-dns-1.sh` may fail sometimes due to DNS timeouts. Run the tests again or increase the DNS timeout, see Section 2.3.1.

Note: DNS related test programs may fail if the first nameserver entry in `/etc/resolv.conf` does not respond properly (and reasonably fast) to DNS queries. See Section 12.3.1 how to override the default nameserver selection: `MTA_NAMESERVER`. Currently most of the DNS related test programs are not *robust* with respect to handling nameserver problems, e.g., slow or unreliable DNS servers. That is, running the tests multiple times may *resolve* the errors due to DNS caching etc.

For more information about possible test program problems see Section 12.3.2. For problems with programs in the `contrib/` directory, see `contrib/README`.

2.4 Installing MeTA1

MeTA1 needs several users to provide separation of privileges and to enhance security. Currently there are four required accounts (the numbers for uid and gid are examples only); the last one listed below (`meta1`) is not really required:

```
meta1s:*:260:260:meta1 SMTPS:/nonexistent:/sbin/nologin
meta1q:*:261:261:meta1 QMGR:/nonexistent:/sbin/nologin
meta1c:*:262:262:meta1 SMTPC:/nonexistent:/sbin/nologin
meta1m:*:263:263:meta1 misc:/nonexistent:/sbin/nologin
meta1:*:264:264:meta1 other:/nonexistent:/sbin/nologin
```

with the corresponding groups:

```
meta1s:*:260:
meta1q:*:261:
meta1c:*:262:meta1s
meta1m:*:263:meta1s,meta1q
meta1:*:264:
```

Note: on some operating systems the star character is not a valid value for the password field. Check `passwd(5)`[3] to determine which value to use to disable the password.

To check whether the required users and groups exist, run

```
$ ./misc/sm.check.sh -p
```

(in the build directory); see below how to override the default values for the user and group names.

A shell script to setup the directories, files, etc. as described below is available in `misc/sm.setup.sh.in`. This script is modified by `configure` to create `misc/sm.setup.sh` (in the build directory) which is invoked when

```
# make install
```

is called. Most defaults in the installation script `misc/sm.setup.sh` can be overridden with environment variables (default is listed in square brackets):

- `MTACONFDIR`: [`/etc/meta1`] configuration directory.
- `MTAQDIR`: [`/var/spool/meta1`] queue directory; communication sockets are created in this directory by default too.
- `MTALOGDIR`: [`.`] logging directory (relative to `MTAQDIR`). If logging is done via `syslog(3)` then this directory is not used.
- `MTAS` [`meta1s`] SMTP Server user and group.
- `MTAC` [`meta1c`] SMTP Client user and group.
- `MTAQ` [`meta1q`] QMGR user and group.

[3] On some systems the man page is in another section, e.g., 4.

- `MTAM` [`meta1m`] address resolver (misc) user and group.
- `MTA` [`meta1`] generic (configuration etc) user and group.
- `MTALG` group for logfiles; the install program tries `operator`, `sysadmin`, and `root`.

Important Notes:

1. The users and groups *must* be created before `make install` is invoked.
2. `misc/sm.setup.sh` will not overwrite existing files or directories, hence it does not work for upgrading a system if configuration files or directory/file owners need to changed.

2.4.1 Directories, Files, and Permissions

`make install` (i.e., `misc/sm.setup.sh`) will create all the required directories and files with the correct permissions provided the users and groups have been set up properly. This section explains what the created structure looks like.

The CDB directories (`0-9`, `A-F`) must be owned by `meta1s` and have group `meta1q` with the permissions `0771`:

```
drwxrwx--x  2 meta1s  meta1q        0/
```

Note: this means that everyone with access to the machine can guess the name of content files (see Section 10.3 for the format; the names can also be read from the logfiles if those are world-readable) and list (`ls(1)`) them, however, they cannot access the content files as those are owned by `meta1s` with mode `0640` and group `meta1c`, e.g.,

```
-rw-r-----  1 meta1s  meta1c  1993 Jul  9 21:19 2/S000000000006B1D200
```

The main (DEFEDB) and incoming queues (IBDB) must belong to `meta1q` and should not accessible by anyone else:

```
drwx------  2 meta1q  meta1q        defedb/
drwx------  2 meta1q  meta1q        ibdb/
drwx------  2 meta1q  meta1q        ibdb/ibdb/
```

Mailertable, aliases map, and other maps for SMAR (see Section 3.9.3) should belong to `meta1m` and can be readable as local conventions require:

```
-rw-r--r--  1 meta1m  meta1m         mt
-rw-r--r--  1 meta1m  meta1m         aliases.db
```

In general, maps should be owned by the user id of the program that uses them, e.g., `meta1q` owns the QMGR configuration map `qmgr_conf.db` (see Section 3.8.1).

The meta1 configuration file can either belong to `root` or the generic meta1 user:

```
-rw-r--r--  1 meta1   meta1          meta1.conf
```

The directories in which the communication sockets between QMGR and the other programs are located must belong to `meta1q` and be group accessible for the corresponding program:

```
drwxrws---  2 meta1q  meta1m        qmsmar/
drwxrws---  2 meta1q  meta1c        qmsmtpc/
drwxrws---  2 meta1q  meta1s        qmsmtps/
```

The directory in which the communication socket between MCP and SMTPS is located must belong to `meta1s`:

```
drwxr-x---  2 meta1s  meta1s        smtps/
```

The logfiles must be owned by the corresponding user and may have relaxed group (or even world) read permissions:

```
-rw-r-----  1 meta1q  operator   qmgr.log
-rw-r-----  1 meta1m  operator   smar.log
-rw-r-----  1 meta1c  operator   smtpc.log
-rw-r-----  1 meta1s  operator   smtps.log
```

To check whether an installation was successful, run

```
# ./misc/sm.check.sh -P
```

(in the build directory). Note: this script uses the same environment variables as the installation script.

2.4.2 Upgrading from earlier MeTA1 Versions

Currently there is no support for automated upgrades. If you have an earlier version of MeTA1 installed and want to upgrade, here are some tips (note: all programs should be run from the build directory unless mentioned otherwise):

- To check whether an installation was successful, run

  ```
  # ./misc/sm.check.sh -P
  ```

- To check whether the configuration file needs changes, run

  ```
  $ ./misc/smconf /etc/meta1/meta1.conf
  ```

  If the file is syntactically invalid for this version of MeTA1 the program will show those errors. Use `-h` as argument to see the available option, e.g., `-u` might be useful.

Chapter 3

Run-Time Configuration of MeTA1

3.1 Overview

Configuration of MeTA1 can be done via command line parameters or via a configuration file (the latter is preferred, the former offers only a small subset of the available configuration options). If a configuration file and command line options are specified, then the options are currently processed in order, i.e., later settings override earlier ones for the same options. Information about the former is available by invoking a program with the option `-h` (MCP currently uses `syslog(3)` instead of `stderr`), it will show the usage as well as the default values. The syntax of the configuration files is specified in the following sections. To actually use a configuration file, the option `-f $CONFIGFILE` must be used, otherwise the programs use only the builtin default values, but not a configuration file. Option '`-V`' can be used to show version information, specifying '`-V`' multiple times shows more detail, e.g., '`-VVVVV`' will show the configuration data including the default value for (almost) every option, and '`-VVVVVV`' will also show all available flags.

Some configuration options can be set via maps, these maps are: `qmgr_conf` for QMGR (see Section 3.8.1) and `access` for SMTPS (indirectly via the address resolver, see Section 3.9.3).

3.2 Configuration File Syntax

The grammar for a MeTA1 configuration file is very simple:

conf	::=	entries
entries	::=	entry *
entry	::=	option | section
section	::=	keyword [name] "{" entries "}" [";"]
option	::=	option-name "=" rhs
rhs	::=	value ";" | "{" value-list "}" [";"]

A configuration file consists of *entries*, each *entry* is either an *option* or a *section*. An option has a *name*, an equal sign, and a *value* terminated by a semicolon or a (bracketed) list of values separated by comma[1]. A *section* consists of a *keyword*, an optional *name*, and a (bracketed) sequence of *entries*. Keywords and options are not case sensitive. The layout of a configuration file does not matter, i.e., indentation and line breaks are irrelevant (in general, but see below for strings).

[1] A trailing comma is allowed to make writing of lists simpler.

3.2.1 Configuration File Values

Values in a configuration file are usually strings or numbers. If a string is used, then it should be quoted, unless it contains no special characters which are treated specially by the grammar. If a string is very long it can be broken into substrings spread out over several lines (just like strings in ANSI C), e.g.,

```
somemessage = "this is a very long string which is spread "
   "out over several lines because otherwise it is too "
   "hard too read.";
```

Numeric values can have the usual prefixes (known from the programming language C) of `0x` for hexadecimal (with digits `0` to `9`, `A` to `F`, and `a` to `f`) and `0` for octal (with digits `0` to `7`). Valid boolean values are `0`, `false`, `off` for false, and `1`, `true`, `on` for true (case insensitive).

In some cases it is possible to have *units* for values. Currently time and size values make use of this feature. Valid time units are `w` for weeks, `d` for days, `h` for hours, `m` for minutes, and `s` for seconds. Valid units for size are `B` for bytes, `KB` for kilo bytes, `MB` for mega bytes, and `GB` for giga bytes. It is allowed to specify a sequence of numbers and units, e.g., `1h 5m 12s`. Unless otherwise specified, the default units for times and sizes in a configuration file are `s` and `B`, respectively; for those values these units can be used.

3.3 Example Configuration File

The installation script creates the file `meta1.conf` in the configuration directory (`/etc/meta1`), see Section 2.4). Check the comments in the file and edit it if required. A configuration file for meta1 contains several sections: a global section which specifies the locations of sockets and directories that are used by multiple components, and one section each for QMGR, SMAR, SMTP server, and SMTP client. Other sections may define services that are started by MCP, e.g., a local mailer.

```
CDB_base_directory = "/var/spool/meta1/";

qmgr {
  AQ_entries_max = 8192;
  smtpc { initial_connections = 19; connections_max = 101; }
  smtps { connections_max = 5; connection_rate_max=160; }
  errors_per_DSN_max=16;
  wait_for_server = 4; wait_for_client = 4;
  start_action = wait; user = meta1q;
  restart_dependencies = { smtps, smtpc, smar };
  path = "/usr/libexec/qmgr"; arguments = "qmgr -f /etc/meta1/meta1.conf";
}

smtps { flags = {8bitmime}; CDB_gid = 261; IO_timeout = 5m3s;
  listen_socket { type = inet; port = 25; }
  start_action = pass; pass_fd_socket = smtps/smtpsfd;
  user = meta1s; path = /usr/libexec/smtps;
  arguments = "smtps -f /etc/meta1/meta1.conf"; }

smtpc {
```

```
  Log_Level = 12; IO_timeout = 6m; wait_for_server = 4;
  start_action = wait; user = meta1c; path = "/usr/libexec/smtpc";
  arguments = "smtpc -f /etc/meta1/meta1.conf"; }

smar {
  Log_Level = 12;
  DNS {
    nameservers = {10.10.10.9, 127.0.0.1};
    timeout = 6;
  }
  start_action = wait; user = meta1m; restart_dependencies = { smtps, qmgr };
  path = "/usr/libexec/smar"; arguments = "smar -f /etc/meta1/meta1.conf";
}
```

3.4 Common Global Configuration

All of the following options have defaults and should only be changed if necessary.

1. `hostname`: set the hostname to use for the various components. This can be set if `gethostbyname(3)` does not return a valid (fully qualified) hostname (format: string).
2. `CDB_base_directory`: base directory of CDB (format: string); this should either be empty (which is the default) or a path to a directory including a trailing slash; the CDB library currently simply appends the directory names (see Section 2.4.1) to it. It might be useful to move some subdirectories to different disks (by creating (symbolic) links (`ln(1)`)) to spread the I/O load.
3. `SMAR_socket`: socket created by the address resolver over which clients (SMTPS, QMGR) can send requests (format: string).
4. `SMTPC_socket`: communication socket between SMTPC and QMGR (format: string).
5. `SMTPS_socket`: communication socket between SMTPS and QMGR (format: string).

The sockets are currently Unix domain sockets only, hence the value is simply the pathname of the socket.

3.5 Common Configuration Options

There is currently one configuration option which is the same across all modules but is not specified in the global section because it is specific to the individual modules.

1. `log`: this is a section with the following options:
   (a) `facility`: see `syslog(3)` for valid facilities, here are some valid options provided the OS offers them: `daemon`, `mail`, `auth`, `local0`, etc.
   (b) `ident`: identification string for `openlog(3)`, defaults to name of the modules. It might be useful to chose other identifiers, e.g., `MeTA1` or `MeTA1QMGR`.
   (c) `options`: options for `openlog(3)` (without the leading `LOG_`) as provided by the OS, e.g., `pid` or `ndelay`.

Example:

```
qmgr { log { facility = daemon; ident=meta1-qmgr; } }
smtps { log { facility = mail; ident=meta1-MTA; } }
```

Note: debug output is currently sent to `stdout` or `stderr`; `syslog(3)` is not used for debugging.

All modules have an option to set the amount of logging (`log_level`) that should be done. The larger the value the more information is logged. For normal operation a value of 9 is recommended. During testing values of 12 to 14 are useful.

3.6 Pathnames for Files, Directories, and Maps

Most names of files (including maps) and directories in the configuration file have a default name (compiled into the binary) without an absolute path, e.g., `aliases.db`. If a pathname is not explicitly set in the configuration file or does not use a absolute path (i.e., begins with a slash), then the default is relative to either

1. the configuration directory: maps and configuration files, e.g., `aliases.db` and `cert_file`.
2. the main queue directory: pathnames of sockets, and databases to store envelope information (IBDB, DEFEDB) or message contents (CDB).

The paths for files mentioned in case 1 are taken relative to the path of the configuration file which is passed via the `-f` option to the various modules. For example: if SMAR is started as

```
/usr/libexec/smar -f /etc/meta1/meta1.conf
```

then the pathname used for the aliases map is `/etc/meta1/aliases.db`. This applies to the SMAR maps aliases, mailertable, and access (3.9.2), the QMGR `qmgr_conf` map (3.8.1), and the STARTTLS related files and directories used by the SMTP server (3.10) and client (3.11).

The paths for files mentioned in case 2 are taken relative to the execution directory. All MeTA1 modules should be started (via MCP) in the main queue directory (default: `/var/spool/meta1`, see Section 2.4).

See the various configuration options explained below how to override the defaults. Note: relative pathnames specified in the configuration file are (currently) always relative to the main queue directory.

3.7 Configuration for MCP

Every section in a MeTA1 configuration file that refers to one of its four main components (QMGR, SMTPS, SMTPC, and SMAR; see Section 1.1.1) has some options that are relevant for MCP. These MCP options are:

1. `start_action`: one of `nostartaccept`, `accept`, `pass`, `wait` (required).
2. `listen_socket`: this is a subsection that specifies the socket on which a process should listen. It must be specified for any `start_action` except `wait`. There are three different socket types available (IPv6 requires the compile time option `MTA_NETINET6`):

(a) `type = inet`

i. `port`: port number on which process should listen (format: numeric).

ii. `address`: IP address on which process should listen, if none is specified the process listens on all local (IPv4) addresses (format: IPv4 address).

(b) `type = inet6`

i. `port`: port number on which process should listen (format: numeric).

ii. `address`: IP address on which process should listen, if none is specified the process listens on all local (IPv6) addresses (format: IPv6 address).

(c) `type = unix`

i. `path`: pathname of Unix Domain socket on which process should listen (format: string).

ii. `umask`: `umask` for socket (format: numeric).

iii. `user`: owner of socket (format: string).

iv. `group`: group of socket (format: string).

3. `pass_fd_socket`: pathname of Unix Domain socket to pass a file descriptor to the process.

4. `user`: user name to run process.

5. `group`: group name to run process.

6. `restart_dependencies`: list of other MeTA1 components that need to be restarted when this one is restarted.

7. `path`: path to program to execute (required).

8. `arguments`: arguments (argv), must start with name of program, see `execv(2)` (required).

9. `pass_id`: this configuration option specifies the command line option for the process to be spawned that MCP must use to pass a unique, numeric identifier. The command line option will be inserted as first argument. It is useful if more than one process should be started and the invocations need a unique id. Example:

```
smtpc { pass_id = "-i"; processes_min = 4; processes_max = 4;
  path = /usr/libexec/smtpc; arguments = "smtpc -f meta1.conf"; }
```

will cause MCP to start four `smtpc` processes, each with the options `-i`ID `-f meta1.conf` where ID is replaced with a unique identifier. Note: currently the option must start with a hyphen and have exactly one character after that.

10. `failure_max`: override the builtin limit for service failures. This can be useful for services that are started for individual requests but may fail in various ways. By setting the value to 0, MCP will still restart the service for each request.

Notes about `start_action`:

- For `start_action = pass` the option `pass_fd_socket` must be specified; in this case MCP binds to the specified socket (`listen_socket`) and passes it via the Unix domain socket (`pass_fd_socket`) to the started process.
- For `start_action = nostartaccept` MCP waits for incoming connections, and then starts a process to handle a single connection.

- For `start_action = accept` MCP binds to the socket and then starts a process to handle the connections without waiting for an actual request.
- For `start_action = wait` MCP simply starts the requested number of processes without passing them any open connections. This is intended for processes that do not communicate with external clients.

MCP is currently a generic control program that does *not* have any *builtin knowledge* about the various MeTA1 modules. Hence the MCP options for each MeTA1 component must be specified properly, there are no builtin defaults that could be associated with the functionality of the various MeTA1 modules. The default configuration file created by the installation program contains the correct defaults. These should only be changed if really necessary.

3.8 Configuration for QMGR

The following configuration options are valid for QMGR:

1. `AQ_entries_max`: maximum number of entries in AQ (active queue) (unit: entries). Note: this value must be larger than the largest number of recipients accepted by a single transaction.
2. `conf`: name of configuration map (including extension), see Section 3.8.1 for details. See also Section 3.6 about relative pathnames.
3. `control_socket`: specify pathname of "control" socket (for querying and making requests). This socket can be used by the query/control program `qmgrctl`, see Section 4.6.3.
4. subsection `DEFEDB`:
   (a) `base_directory`: home directory for DEFEDB.
   (b) `log_directory`: log directory for DEFEDB. For better performance, this directory can be set to point to a different disk than the base directory of DEFEDB.
5. subsection `DSN_handling`:
   (a) `merge_delay_max`: maximum time to wait for merging multiple DSNs into one (unit: s).
   (b) `flags`: configuration flags:
      i. `header_only`: include only the headers in a DSN; by default the first bounce includes the entire message and subsequent ones include only the headers.
      ii. `MIME_Format`: use MIME to structure a DSN. Note: this is not a DSN in the format specified by RFC 3464 [MV03], use `RFC_Format` (see 5(b)iii) instead.
      iii. `RFC_Format`: create DSNs in the format specified by RFC 3464 [MV03].
      iv. `no_LMTP_delay_DSN`: do not generate a delay DSN if the RCPT is delivered via lmtp.
   (c) `errors_per_DSN_max`: maximum number of error messages (failed recipients) in a bounce (DSN) (unit: entries).
6. `double_bounce_address`: RFC 2821 address for double bounces; defaults to `<doublebounce@`hostname`>`. The SMTP server discards mails to this address if the sender address is `<>`.
7. subsection `IBDB`:
   (a) `commit_delay_max`: maximum time between commits to IBDB (unit: μs)

(b) `size`: maximum size of each IBDB file (unit: B).

(c) `open_TAs_max`: maximum number of open transactions in IBDB before a commit is performed (unit: entries).

Note: the configuration file offers no way to specify a base directory for IBDB, however, the directory can be easily moved elsewhere and a (symbolic) link (`ln(1)`) can be added.

8. subsection `IQDB`:

   (a) `cache_entries_max`: maximum number of entries in IQDB cache (unit: entries). This must be larger than the sum of all recipients in open transactions.

   (b) `hash_table_entries`: size of hash table for IQDB (unit: entries). This must be larger than `cache_entries_max`.

9. `log_level`: logging level.

10. `disk_space_min`: minimum amount of free disk space (unit: KB). This value should be significantly larger than the maximum size of a message to be accepted by the SMTP server, it should be as large as the maximum message size multiplied by the maximum number of incoming connections.

11. `disk_space_ok`: amount of free disk space at which normal operation continues (unit: KB). Must be larger than `disk_space_min`.

12. `OCC_entries_max`: size of outgoing (SMTPC) connection cache (unit: entries). This should be large enough to keep track of outgoing connections over a time span that is at least as long as the maximum retry time.

13. `queue_return_timeout`: maximum time in queue (unit: s).

14. `queue_delay_timeout`: send delay warning ("delayed DSN") if the mail is still in the queue after at least this duration (unit: s). To turn off delayed DSNs set this 0. Note: based on the retry schedule the delayed DSN might be sent later than the option specifies.

15. `retry_delay_max`: maximum time for retrying a delivery (unit: s).

16. `retry_delay_min`: minimum time for retrying a delivery (unit: s).

17. subsection `smtpc`:

    (a) `initial_connections`: maximum initial number of outgoing connections to a single host (unit: entries). The sliding window for the slow start algorithm (see Section 3.8.1) is initialized with this value.

    (b) `connections_max`: maximum number of outgoing connections to a single host (unit: entries).

    (c) `transactions_per_session_max`: maximum number of transactions per session (unit: entries).

    (d) `lmtp_rcpts_per_transaction_max`: maximum number of recipients per transaction for mail sent via LMTP (unit: entries).

    (e) `smtp_rcpts_per_transaction_max`: maximum number of recipients per transaction for mail sent via (E)SMTP (unit: entries).

    (f) subsection `connection_limits`: This named section can be specified multiple times. An entry is selected via the configuration map for QMGR (Section 3.8.1) using the tag `smtpc_conf_rcpt_domain` with the domain part of the recipient as argument.

i. `initial_connections`: maximum initial number of outgoing connections to a single host (unit: entries).

ii. `connections_max`: maximum number of outgoing connections to a single host (unit: entries).

iii. `transactions_per_session_max`: maximum number of transactions per session (unit: entries).

Example:

Configuration file:

```
connection_limits low {
  initial_connections = 1;
  connections_max = 2;
}
```

Map entry:

```
smtpc_conf_rcpt_domain:example.com connection_limits=low;
```

(g) `flags`: configuration flags:

i. `lookup_rcpt_conf`: Look up recipient configuration data (see Section 3.11.1) in the access map (see Section 3.9.3)

ii. `lookup_session_conf`: Look up session configuration data (see Section 3.11.1) in the configuration map (see item 2). This flag is needed if there is no `session_features` section in smtpc (see Section 3.11, item 5), but only entries in the map containing other configuration option, e.g., `tls_requirements`.

(h) `rcpt_conf_lookup_flags`: If recipient configuration data (see item 17(g)i) is looked up in the access map, then these flags determine which kind of lookups should be performed.

i. `full_address`: use the full address as key.

ii. `detail_plus`: look up also "`user++@subdomain`".

iii. `detail_star`: look up also "`user+*@subdomain`".

iv. `star`: look up also "`user*@subdomain`".

v. `domain`: look up domain part.

vi. `dotsubdomain`: iterate through subdomains.

vii. `dot`: look up also ".".

The default is to perform all lookups.

18. subsection `smtps`:

(a) `connection_rate_max`: maximum incoming connection rate from a single host (unit: connections/60s).

(b) `connections_max`: maximum number of open incoming connection from a single host (unit: entries).

19. subsection `VERP`: If VERP is turned on then the original MAIL address is modified for an outgoing transaction to make tracking simpler. The replacement is as follows: if MAIL is `<lm@mail>` and RCPT is `<lr@rcpt>` then the new sender address is `<lm+lr=rcpt@mail>`.

(a) `when`: This option has two possible values: `always` and `never`. If neither of these is set, then VERP is done on demand as explained in Section 3.14.

(b) `delimiter1`: This delimiter is used to separate the local part of the original MAIL address from the (modified) RCPT address that is added (default: '+').

(c) `delimiter2`: This delimiter is used to replace the '@' sign in the RCPT address before it is added to the new sender address (default: '=').

20. `wait_for_client`: maximum amount of time to wait for a client to become available (unit: s)
21. `wait_for_server`: maximum amount of time to wait for a server to become available (unit: s)

3.8.1 Configuration Map for QMGR

QMGR implements a "slow start" algorithm to control the number of concurrent connections to one IP address. Initially, it will at most create a (small) number of open connections up to a specified initial limit. For each successful delivery, the allowed number is increased up to specified maximum limit. Most delivery failures[2] in turn will cause a decrease of the allowed number, even down to zero. If zero is reached, new connections will only be tried after the connection cache timeout (see below) is exceeded.

For incoming connections, QMGR establishes two limits: the connection rate and the number of open connections.

The Berkeley DB hash map `qmgr_conf.db` (the file should be owned by `meta1q`) can have the following entries:

1. `oci:` this key specifies the initial number of concurrent outgoing connection to an IP address.
2. `ocm:` this key specifies the maximum number of concurrent outgoing connection to an IP address.
3. `ocr:` this key specifies the maximum rate of outgoing connection to an IP address in connections per minute (requires compile time option `OCC_RATE`).
4. `octo:` specify the timeout for an entry in the outgoing connection cache.
5. `icr:`, `icl:` specify the maximum rate for incoming connections (per minute).
6. `icm:`, `icc:` specify the maximum number of concurrently open incoming sessions.
7. `smtpc_conf_rcpt_domain:` see Section 3.8, item 17f.
8. `smtpc_session_conf:` see Section 3.11.1.

`oci:`, `ocm:`, `icr:`, `icm:`, and `smtpc_session_conf:` take an IP address/net as parameter such that the limits can be imposed per IP address/net. `icl:` and `icc:` take a host/domain name as parameter. For example:

```
oci:127.0.0.1      5
ocm:127.0.0.1     10
oci:10            10
ocm:10            50
oci:               1
ocm:               4
icr:10             5
```

[2]Delivery errors that are specific to a transaction, e.g., unknown recipients, are not considered.

```
icr:127.0.0.1    100
icm:127.0.0.1    120
icl:meta1.org.    10
smtpc_conf_rcpt_domain:example.com connection_limits=low;
```

Note, however, that the limits apply only to single IP addresses, they are not aggregated for nets. That is, for the example every single host in the IP net 10.x.y.z can have a maximum incoming connection rate of 5 messages per minute.

The default values for these configuration options are set in the binary and can be changed via command line options or the configuration file (see Section 3.8):

1. `-C n` maximum number of concurrent connections to one IP address [default: 100]
2. `-c n` initial number of concurrent connections to one IP address [default: 10]
3. `-O R=n` maximum connection rate per 60s (SMTPS) [default: 100]
4. `-O O=n` maximum number of open connections (SMTPS) [default: 100]

Note: the limits set via the RCPT domain in `connection_limits` (see Section 3.8, 17f) are overridden by these limits.

3.9 Configuration for SMAR

3.9.1 Declaring Maps for SMAR

In general, maps must be declared before they can get used. Each map declaration in a configuration file is a named subsection – the name is used for later references – `map` in the `smar` section with the following options:

1. `type`: type of the map; currently one of `hash` (Berkeley DB hash), `cdb` (tinycdb), `regex`, `sequence`, `socket`, and `passwd`.
2. `file`: the filename of the `db` file (including the extension) (for type `hash`, `cdb`, `regex`).
3. `mapname`: name of the map used in the protocol (type `socket` only).
4. `address`: IPv4 address of inet socket. (type `socket` only).
5. `path`: the pathname of the Unix domain socket (for type `socket`).
6. `port`: port for inet socket (type `socket` only).
7. `maps`: list of map names to use in the map (type `sequence` only).

Note: for `socket` maps either a Unix domain socket (`path`) or an inet socket (`address` and `port`) must be specified.

Example:

```
map localusers { type = hash; file = "/etc/meta1/localusr.db"; }
map otherusers { type = cdb; file = "/etc/meta1/otherusr.cdb"; }
map password { type = passwd; }
map seq1 { type = sequence; maps = { localusers, otherusers }; }
map seq2 { type = sequence; maps = { password, otherusers }; }
```

3.9.2 Configuration Options for SMAR

The following configuration options are valid for SMAR:

1. `access_map`: this is a subsection that specifies the access control map. See Section 3.9.3 for details.

   Note: only one of `file` (1a) and `name` (1b) must be specified.

   (a) `file`: filename of access map (including extension) [default: `access.db`].

   (b) `name`: name of access map . This can be used if a different map type should be used, in which case the map must be declared as explained in Section 3.9.1.

2. `address_delimiter`: list of delimiters (specified as string) for address extensions in local part, [default: `"+"`]. Note: if `address_delimiter` has more than one character, the first one that is found in the local part of an address is used as delimiter in map lookups (see Section 3.12). For example: if the following option is used in the configuration file:

   ```
   address_delimiter = "/_-";
   ```

   then for the address "<`user/ext-list@dom.ain`>", the delimiter for map lookups is "`/`" and the address detail is "`ext-list`", while for the address "<`user-ext_list@dom.ain`>", the delimiter for map lookups is "`-`" and the address detail is "`ext_list`".

3. `aliases`: this is a subsection that specifies the parameters for aliases.

   Note: only one of `file` (3a) and `name` (3b) must be specified.

   (a) `file`: filename of aliases map (including extension) [default: `aliases.db`].

   (b) `name`: name of aliases map . This can be used if a different map type should be used, in which case the map must be declared as explained in Section 3.9.1.

   (c) `flags`:

      i. `localpart`: the aliases map contains only localparts of addresses and those are only looked up for local addresses.

      ii. `local_domains`: the aliases map contains fully qualified addresses which are only looked up for local addresses. This can be used similar to virtual users in sendmail 8, e.g.,
      ```
      vuser1@virt1.tld:   user1
      vuser2@virt1.tld:   user2
      vuser3@virt2.tld:   user3
      ```

      iii. `all_domains`: the aliases map contains fully qualified addresses which are looked up for all domains.

      iv. `implicitly_match_detail`: the items are looked up according to the algorithm specified in Section 3.12.1. and additionally `+detail` is implicitly matched when the pattern is "`user@hostname`". That is, it overrides the default matching explained in case 1e in Section 3.12.1.

      v. `replace_macros`: replace macros in the RHS of the map entries by the appropriate value, see Section 3.12.3.

vi. `preserve_domain`: if the RHS of an entry is an unqualified address, do not append the local hostname to it but the domain of the original address, i.e., preserve the original domain.

4. `DNS`: this subsection contains DNS related options.

   (a) `nameservers`: list of up to four IPv4 addresses[3] of nameservers. Note: it is important that all of these nameservers work properly, as the method to exclude unresponsive nameservers is not very efficient.

   (b) `retries`: maximum number of retries. A value of 0 means one query only, i.e., no retry.

   (c) `timeout`: the default timeout for a single DNS query (unit: s). Notes:

   - the timeout for a DNS request is the product of the number of tries and the individual timeout, i.e., `(retries + 1) * timeout`.
   - this value is only the default timeout which can be overridden by an application. For example, QMGR dynamically increases the timeout for addresses which did not resolve in earlier tries.

   (d) `flags`: The flag `use_resolvconf` causes the list of nameservers (see 4a) to be read from `/etc/resolv.conf`. This flag is set by default unless the `nameservers` option is used. Note: the list of nameservers is not updated when `/etc/resolv.conf` is changed, `smar` needs to be restarted to achieve that. If the option `nameservers` is not used and the flag `use_resolvconf` is cleared, then smar uses `127.0.0.1` as default.

   More flags can be found in Section 8.4.

   (e) `port`: Specify the port on which the `nameservers` are listening. By default (or if set to `0`), port `53` is used (`domain`).

   (f) `Use_DNSSEC`: Use DNSSEC for DNS queries.

5. `dnsbl`: specify a DNS based blacklist[4]. This section can be specified multiple times[5]; it has the following required options:

   - `domain`: specify the domain to use for DNS lookups, e.g., `dnsbl.tld`.
   - `tag`: specify the tag to use for lookups in the access map (which must be enabled, see Section 3.10, 4b).

   The client IPv4 address *A.B.C.D* is looked up via DNS as *D.C.B.A*.`domain` querying for an A record. If an A record *W.X.Y.Z* is found, then it is looked up in the access map as `tag`:*W.X.Y.Z.* for temporary and permanent DNS lookup failures the entries that will be checked in the access map are `tag:temp` and `tag:perm`, respectively.

   Notes:

   - DNS lookups in blacklists can be disabled via entries in the access map using the tag `cltaddr`, see Section 3.9.3.
   - Some DNS blacklists return multiple A records. For those the A records are checked (in the order returned by the DNS server) until an access map entry is found. This is an intermediate solution as it may cause random results if multiple access map entries for a DNS blacklist exist. However, as long as all of them have the same return code type (i.e., temporary or permanent), this behaviour is sufficient.

[3] 4 is the default value for the compile time option `MTA_DNS_MAX_TSKS`
[4] This option is modelled after `dnsblaccess` written by Neil Rickert for sendmail 8.
[5] Compile time option `MTA_MAX_DNSBL`: currently set to 8.

- currently a colon is added as delimiter after `tag`, this may be removed in later versions to allow for more flexibility; e.g., the configuration option itself can include a delimiter.

The access map entry should have one of the usual rejection RHSs as explained in 3.9.3. Example: configuration file:

```
smar { dnsbl { domain = dnsbl.tld; tag = dnsbltld; } }
```

access map:

```
dnsbltld:127.0.0.1  error:550 5.7.1 listed at dnsbl.tld as open relay
dnsbltld:127.0.0.2  error:550 5.7.1 listed at dnsbl.tld as spam source
dnsbltld:127.0.0.9  error:451 4.7.1 listed at dnsbl.tld as suspicious
dnsbltld:temp       error:451 4.7.1 temporary lookup failure at dnsbl.tld
```

If multiple DNS based blacklists are specified, the DNS queries are made concurrently but the lookups in the access map are performed in the order in which the blacklists are given; the first successful lookup is used as result, no further priorization is performed.

6. `greylisting`: specify greylisting options, see Section 3.9.4 for details.
   (a) `grey_wait`: how long before greylisted can be confirmed.
   (b) `grey_expire`: timeout for greylisted entries (did not confirm within that time).
   (c) `white_expire`: expire whitelisted entries after this time if necessary.
   (d) `white_timeout`: force whitelisted entries to reconfirm after this time.
   (e) `main_DB_name`: name of main database (including .db extension).
   (f) `secondary_DB_name`: name of secondary database (including .db extension).
   (g) `expire_limit`: try to expire entries when this limit is reached.
   (h) `netmask`: by default the entire IPv4 address is used as a key, however, by specifying a netmask, e.g., `0xFFFFFF00`, the least significant bits can be cut off. This can be used to deal with server farms, see Section 3.9.4, e.g., if those are in the same class C subnet.
7. `local_user_map`: this is a subsection that specifies a map of valid local addresses.
   (a) `name`: Name of the map of valid local addresses; the map must have been declared as explained in Section 3.9.1.
   (b) `flags`:
      i. `implicitly_match_detail`: `+detail` is implicitly matched when the pattern is "`user@hostname`". That is, it overrides the default matching explained in case 1e in Section 3.12.1.
8. `log_level`: logging level.
9. `mailertable`: this is a subsection that specifies a mailertable, currently you can specify exactly one of the following two options:
   (a) `file`: filename of mailertable [default: `mt`]. In this case a plain text file is read during startup and placed in an internal hash table. Note: it is strongly suggested to use lower case keys in this case, see Section 6.5, item 2.
   (b) `name`: name of a mailertable map that has been declared before (see Section 3.9.1).
   (c) `flags`: these flags can be used to select a subset of the matching described in Section 3.12.1.

i. `full_address`: use the full address as key.
ii. `detail_plus`: look up also "`user++@subdomain`".
iii. `detail_star`: look up also "`user+*@subdomain`".
iv. `star`: look up also "`user*@subdomain`".
v. `domain`: look up domain part.
vi. `dotsubdomain`: iterate through subdomains.
vii. `dot`: look up also ".".

The default is `domain`, `dotsubdomain`, `dot`.

The format of entries in the map is explained in Section 3.9.3. Note: reloading mailertable (Section 4.7) while SMAR is running can be done only if it is declared as Berkeley DB (case 9b with the proper map)

3.9.3 Configuration Maps for SMAR

SMAR requires a mailertable, and it can make use of an alias map as well as an access map, all of which are described in the subsequent sections.

Access Map

To activate the access map the flag `access` (see Section 3.10, item 4b) (or the option `-a`) must be given to the SMTP servers. All entries consist of a left hand side (LHS, key) which in turn has a tag and a (partial) address and a right hand side (RHS, value). Valid tags are:

Tag	refers to
`from:`	envelope sender address (MAIL)
`to:`	envelope recipient address (RCPT)
`cltaddr:`	client IP address
`cltname:`	client host name
`cltresolve:`	result of forward and reverse client lookup
`mxbadip:`	IPv4 addresses that are not allowed for MX - A records
`certissuer:`	DN of CA cert that signed that presented cert
`certsubject:`	DN of presented cert
`protectedrcpt:`	restrictions for recipient address (see Section 3.10.3)
`smtps_session_conf:`	configuration options for a session in the SMTP server (IP address/net) (see Section 3.10.1)
`smtps_se_conf:`	configuration options for a session in the SMTP server (hostname)
`smtps_rcpt_conf:`	envelope recipient address (RCPT) (see Section 3.10, 4i)
`smtpc_rcpt_conf:`	configuration options for recipient in the SMTP client (see Section 3.11.1)
`smtpc_session_conf:`	configuration options for a session in the SMTP client
`ehlo:`	EHLO/HELO parameter (see Section 12.1.4)

Valid addresses for `from:`, `to:`, and `smtps_rcpt_conf:` are RFC 2821 addresses without the angle backets (`localpart@domain`) as well as partial addresses in the form `localpart` and `@domain`, i.e., domains must be preceeded with an at (`@`) sign. `cltaddr:` requires IPv4/IPv6 addresses and (sub)nets, `mxbadip:` requires IPv4 addresses and (sub)nets, and `cltname:` requires host names. The client host name is determined by performing a reverse lookup (PTR record) for its IP address. The resulting names are looked up as A/AAAA records. Only if one of the A/AAAA records matches the client IP address, the host name is set. Note: the host name has a trailing dot after DNS resolution, this dot must be included in the entry. The result of these lookups can be used for `cltresolve:` where the following keys are valid:

`ok`	reverse and forward lookup match
`no`	reverse and forward lookup do not match
`tempptr`	reverse lookup (PTR) caused a temporary error
`tempa`	forward lookup (A/AAAA) caused a temporary error

Valid values for RHS are

`relay`	allow relaying; currently only for `to:`, `cltaddr:`, `cltname:`, `certissuer:`, and `certsubject:`
`ok`	accept command
`error:`*XYZ A.B.C.D text*	return an error consisting of SMTP reply code *XYZ*, enhanced status code *A.B.C.D*, and *text*, i.e., the part after `error:` is returned to the client.
`reject`	same as `error:550 5.7.0 Rejected.`
`discard`	accept command but silently discard its effects.
`cont`	stop current check (e.g., map lookup), but continue others.

Some tags may allow for other RHS values, these are explained when those tags are discussed in more detail.

Optionally a RHS can be preceeded by the modifier `quick:`. For an `error:` entry it causes an immediate rejection when the entry matches. Otherwise rejections can be delayed to the `RCPT` stage – if SMTPS is configured appropriately, see Section 3.10, item 4e – and can be overridden using the modifier `quick:` together with `ok` or `relay` in the access map for the recipient address with the `to:` tag; or with `quick:ok` for the sender address with the `from:` tag. Using the modifier `quick:` together with `relay` for an entry with the `cltaddr:` tag causes it to override all other access map checks. `quick:ok` for an entry with the `cltaddr:` tag causes it to override other access map checks unless they are necessary to allow relaying.

Domain names (`@domain`) must have an exact match, subdomain matching can be specified with a leading dot, i.e., `@.domain`, see Section 3.12.1.

Examples:

```
cltresolve:tempptr        error:451 4.7.1 reverse lookup failed
mxbadip:127.0.0.1         error:551 5.7.1 Bad IP address 127.0.0.1 in MX/A list
mxbadip:192.168.255.255   error:551 5.7.1 Bad IP address 192.168.255.255 in MX/A list
from:@spammer.domain      error:551 5.7.1 No spammers
from:@.spammer.domain     error:551 5.7.1 No spammers in subdomains either
to:root                   error:551 5.7.1 No mail to root
to:abuse                  quick:ok
cltaddr:10.               error:551 5.7.1 No direct mail from 10.x.y.z
cltname:spammer.domain.   quick:error:551 5.7.1 No mail from spammers
to:@primary.domain        relay
cltaddr:10.               relay
cltaddr:127.0.0.1         quick:relay
```

Discard The effect of `discard` depends on the protocol stage in which it is returned. If it is returned for a session, e.g., when a client connects, all transactions in the session are discarded. If it is returned for `MAIL` only that transaction is discarded. If it is returned for `RCPT` only that recipient is discarded; however, if no valid recipients are left, the entire transaction is discarded. Moreover, if `quick:discard` is returned for one recipient the entire transaction is discarded too.

Mailertable

The address resolver implements an asynchronous DNS resolver and by default it uses a file called `mt` (mailertable) (see Section 3.9.2, item 9) which consists of domain parts of e-mail addresses and corresponding IP addresses (in square brackets) or domain/host names. An entry consists (as usual in a map) of a LHS and a RHS; in the case of a flat text file, i.e., case 9a of Section 3.9.2, those are separated by one or more whitespace characters.

```
LHS      ::=  [ local "@" ] [ "." ] hostname | "."
RHS      ::=  [[ port "^" ] ["esmtp:"]] hosts | "lmtp:" | port "^lmtp:" hosts
              | "{" ["protocol" "=" "esmtp" ";"] [portdef] hostdef [flagdef] "}"
              | "{" "protocol" "=" "lmtp" ";" [portdef hostdef] "}"
port     ::=  integer
hosts    ::=  hostname | iplist
iplist   ::=  "[" IPv4-address "]" [ " " iplist ]
portdef  ::=  "port" "=" int ";"
hostdef  ::=  "host" "=" hostname | "ip" "=" "{" ip-list "}"
ip       ::=  IP-address
flagdef  ::=  "flags" "=" "MX_lookup" ";"
```

The key (LHS) is an address (without angle brackets), a hostname, or a dot (denoting the default entry). The value (RHS) is

- either a specification following the same syntax as the configuration file (in this case the specification must be enclosed by curly braces),
- or it uses a syntax specific to mailertable consisting of an optional port number, an optional (`esmtp`) protocol and a hostname or a list of IP addreses (in square brackets) which are separated by spaces.

If LMTP should be used, then the `lmtp` protocol must be selected. There are two cases: just { `protocol=lmtp;` } (or `lmtp:`) by itself means the delivery agent will use the Unix domain socket specified in the configuration file (see Section 3.11, item 2), if an inet socket should be used then a port and a host must be specified. By default, a hostname is subject to MX lookups. However, in a structured RHS the flag `no_MX_lookup` can be used to suppress the MX lookup.

Example:

```
localhost           lmtp:
SPAM.FILTER.DOMAIN  { port=2525; protocol=esmtp; ip={127.0.0.1} }
LMTPHOST.MY.DOMAIN  { protocol=lmtp; port = 525; ip= { 10.11.12.13 } }
OTHER.DOMAIN        { host = SERVER.OTHER.DOMAIN; flags=no_MX_lookup; }
MY.DOMAIN           esmtp:[10.1.2.3]
ANOTHER.DOMAIN      esmtp:MTA.SERVER
.TLD                { host = GATE.WAY }
.                   esmtp:SMART.HOST
```

Note: currently this file must exist, even if there are no entries (it is created during installation).

Aliases

To specify aliases for local addresses the map `aliases.db` (Section 3.9.2, item 3a) is used. The key in the map must be

- the local part of a valid (local) e-mail address,
- or a complete local e-mail address,
- or any e-mail address,

based on the `flags` of the `aliases` option (see Section 3.9.2, 3c). The value (RHS) for an alias entry is a list of one or more RFC 2821 addresses (including the angle brackets) separated by spaces (*not* commas). If the RHS has only a single address which does not start with an angle bracket, then it is converted into an RFC 2821 address by SMAR, i.e., SMAR will append the hostname of the machine and put angle brackets around the string. Example:

```
myalias:  localuser
mylist:  <user1@my.dom> <user2@my.dom> <localuser@local.host>
owner-mylist:  someuser
```

For mailing lists, the `owner-` notation is supported, i.e., if there are aliases *list* and `owner-`*list* then mail sent to *list* will use `owner-`*list* as envelope sender address; the original domain will be preserved.

Example for the flag `local_domains` (see 3.9.2, 3(c)ii). Let two domains be local, i.e., in mailertable:

```
first.dom   lmtp:
second.dom  lmtp:
```

and these entries be in aliases:

```
myalias@first.dom:  user1
another@second.dom:  user2
```

Then mail to `<myalias@second.dom>` and `<another@first.dom>` would be rejected while mail to `<myalias@first.dom>` or `<another@second.dom>` would be accepted.

Aliases can be nested (currently up to 5 levels, see `smar/rcpts.c`).

3.9.4 Greylisting

MeTA1 supports two forms of greylisting [Hara]

1. simple greylisting: only uses the client IP address as key. [Posa].
2. full greylisting: uses a tuple consisting of client IP address, envelope sender, and envelope recipient as key.

The idea behind greylisting is simple: do not accept mail from an unknown source on the first connection, but reject it with a temporary error. Any MTA that conforms to RFC 2821 [Kle01] will try to send the mail later on, however, spamming systems often do not do that. An IP address can be in three different states: *unknown*: the client has not connected before or the entry is expired from the database, *greylisted*: the client has connected before but it did not yet connect again within the configured time interval, *whitelisted*: the client has connected before and it connected again within the configured time interval. The time interval is specified by its lower limit `grey_wait` and its upper limit `grey_expire`. A lower limit is used to prevent system from getting accepted that just send a single message within a few seconds again and again. The upper limit is used to avoid filling up the database. If an entry has made it to the *whitelisted* state, it will stay there (at least) up to the timeout specified by `white_expire`. The greylisting algorithm implemented in MeTA1 uses another timeout `white_timeout` after which a *whitelisted* entry is considered stale and must go through the greylisting stages again, i.e., it is considered

to be in state *unknown*. Each time a mail is sent from a whitelisted host, the entry is updated, to avoid that systems which regularly sent mail become greylisted again.

Greylisting is performed at the `RCPT` stage of the SMTP dialogue. It is only done when a valid recipient is specified, i.e., all other checks must have been successful. Hence clients that do not try to send mail or just try invalid recipient addresses will not be added to the greylisting database. If a transaction is subject to greylisting then the session is aborted with an 421 error. If a server uses callbacks to verify the sender address, then the option `delay_greylisting_error_until_DATA` (see Section 3.10, item 4f) is useful to avoid unnecessary delays. Here is an example: host A is the main MX server for domain `example.com` and it uses greylisting, host B is the main MX server for domain `example.net` and it uses sender callbacks. If a mail is sent to host B for `<rcpt@example.net>` with the sender address `<user@example.com>` then host B will connect to host A to test whether `<user@example.com>` is a valid recipient. However, if host A does not have host B in its whitelist, it will return a 421 error after the `RCPT To:<user@example.com>` command, which (depending on the implementation of the sender callback) will cause host B to temporarily reject the mail for `<rcpt@example.net>`. By specifying the option `delay_greylisting_error_until_DATA` on host A the `RCPT` command will succeed and the original mail to `<rcpt@example.net>` will go through without delay.

The greylisting implementation uses two persistent databases (specified by `main_DB_name` and `secondary_DB_name`), where the second DB is just a secondary index (by expiration time) for the main DB. These databases should be on a filesystem with sufficient free disk space depending on how many connections from different clients the MTA receives. Entries are only removed from the DB if there are more than `expire_limit` elements. However, if none of the entries are expired yet, then the number of elements can exceed that limit.

Greylisting: Whitelisting

Greylisting can be disabled for selected hosts by adding them to the access map (see Section 3.9.3), e.g.,

```
cltaddr:10.         relay
cltaddr:127.0.0.1   quick:relay
```

Possible Problems with Greylisting

Some *legitimate* mailers do not behave properly and will not retry a mail that had a temporary error. This can cause mail loss in various situations, e.g., because the receiving system is currently out of some resources. However, to minimize the impact of greylisting on these misbehaving mailers it might be useful to explicitly whitelist them as:

```
cltaddr:12.107.209.244   ok
cltaddr:64.12.137.       ok
```

A list of such broken mailers can be found at `http://projects.puremagic.com/web-svn/wsvn/greylisting/trunk/schema` [Harb]. A related problem are server farms where a mail might be resent from a different IP address. These should probably be whitelisted too; some of these can be found at the URL given before. However, entries in that file which have the comment "unique sender per attempt" do not need to be whitelisted as this implementation does not use the sender address.

For another possible problem see Section 3.11.1.

Note: if a client authenticates via STARTTLS or AUTH such that relaying is allowed then greylisting is disabled for that client.

3.10 Configuration for SMTP Server

The following configuration options are valid for SMTPS:

1. `auth`: this is a subsection that specifies the parameters for `AUTH` support. It is only available if the system has been configured with the option `--enable-SASL`, see Section 2.2.1.

   (a) `flags`: flags for SMTP AUTH
   See the Cyrus SASL documentation for the meaning of these flags: `noplaintext`, `noactive`, `nodictionary`, `forward_secrecy`, `noanonymous`, `pass_credentials`, `mutual_auth`.

   (b) `trusted_mechanisms`: list of SASL mechanisms for which relaying is allowed if a client successfully authenticated using one of those

   (c) `user_realm`: the following description is taken from the Cyrus-SASL documentation for the `sasl_server_new()` call: The realm the connected client is in. The Kerberos mechanisms ignore this parameter and default to the local Kerberos realm. A value of NULL (here: if not set) makes the library default, usually to the serverFQDN; a value of `""` specifies that the client should specify the realm; this also changes the semantics of `@` in a username for mechanisms that don't support realms.

   Note: the name for the Cyrus-SASL configuration file is currently `meta1.conf`. That file can be used to adjust the list of mechanisms (option `mech_list`) that should be advertised (besides many other things), hence this option is not in the SMTP server itself.

2. `CDB_gid`: (numeric) group id for CDB files, i.e., the group id of `meta1c`, see Section 2.4.1.

3. `DKIM_signing`: this is a subsection that specifies the parameters for DKIM signing support. It is only available if the system has been configured with the option `--enable-DKIM`, see Section 2.2.1. For details about DKIM see RFC 4871 [ACD+07].

   (a) `signature_method`: signature method: `rsa-sha256` (default) or `rsa-sha1`.

   (b) `body_canonicalization`: canonicalization algorithm for mail body: `simple` (default) or `relaxed`.

   (c) `header_canonicalization`: canonicalization algorithm for mail header: `simple` (default) or `relaxed`.

   (d) `ignore_errors`: if set, the server will ignore errors that occur during signing and accept the mail nevertheless. By default, any error is returned to the SMTP client as corresponding reply code.

   (e) `sign_header`: specifies a list of header names which will be signed by DKIM. See below for a description of the algorithm.

   (f) `skip_header`: specifies a list of header names which will not be signed by DKIM. The match is done via a prefix comparision against the header name plus a trailing colon, which allows to also perform a full match. For example, `skip_header = { "X-", "Received:" };` will skip all headers that start with `X-`, and all `Received` headers.
   The decision whether or not to sign a header is made as follows (any decision is final, i.e., it is not overridden by subsequent steps):

      i. if a header is listed in `skip_header` it will not be signed.
      ii. if `sign_header` is empty, the header will be signed.
      iii. if `sign_header` is not empty, the header will be signed if it is in the list.
      iv. in all other cases the header will not be signed.

(g) `over_sign_headers`: specifies a list of header names which will be *over*-signed by DKIM, i.e., each specified header name will be listed in the header list (`h=`) one more time than it is specified in the header of the mail itself, thus preventing an attacker to add another header of the same name (without breaking the DKIM signature).

(h) A subsection called `sign` must be specified at least once:

i. `keyfile`: name of file containing the private (RSA) key (required).

ii. `domain`: the domain of the signing entity (required). This is the domain that will be queried for the public key by a DKIM verification program and is places in the DKIM-Signature header as `d=`.

iii. `selector`: selector name (required).

iv. `from`: specifies (part of) an address that must match the `From:` address in the mail. This option is required in all but one entry, obviously different entries must use different `from` values. An entry without this option is the default entry that will be used for signing if none of the `from` values matches. The `From:` address is checked against the specified `from` options based on the algorithm described in Section 3.12.1, however, no `+detail` handling is (currently) performed. The flags `full_address` and `dotsubdomain`, are set for this map, thus influencing the lookups that are performed. Moreover, domain lookups are done without leading `@`.

Example: sign messages that have a sender address of `sender@example.net` with the domain `example.net` and selector `net`, messages which have a sender domain of `example.org` with the domain `example.org` and selector `org`, all others are signed with the domain `example.com` and selector `default`:

```
DKIM_signing {
  sign {
   domain=example.com; selector=default; keyfile=com.key; }
  sign {
   from=sender@example.net;
   domain=example.net; selector=net; keyfile=net.key; }
  sign {
   from=example.org;
   domain=example.org; selector=org; keyfile=org.key; }
 }
```

Notes:

- DKIM signing is done unconditionally if this section is specified. Hence you *must not* use this on a publically accessible SMTP server. Instead you should define a separate SMTP submission server (see Section 3.10.2) and restrict access to authenticated users or to some trusted IP range.
- If a pmilter changes the mail body or a header that has been signed, then the DKIM signature becomes invalid. Currently no attempt is made to deal with this situation.

4. `flags`:

(a) `8bitmime`: offer `8BITMIME`: MeTA1 is 8 bit transparent, but it does not perform any conversion, so this option should only be used if all communication partners can deal with 8 bit data.

(b) `access`: use access map (in SMAR). Note: currently this flag is required to perform a reverse lookup for a client IP address to get the hostname of the client which then can be used for logging and the `Received:` header.

(c) `allow_data_before_greeting`: allow a client to send data before the initial 220 greeting.

(d) `allow_space_after_colon_for_MAIL_and_RCPT`: Unfortunately some people are not able to read RFCs but try to implement mail programs nevertheless. To deal with one of their errors, this option has been created. It allows a space after the colon in the `MAIL` and `RCPT` commands, e.g.,

```
MAIL FROM: <banktransfers@cashedge.com>
```

(e) `delay_checks`: delay acceptance check until `RCPT` stage (unless explicitly overridden, see Section 3.9.3).

(f) `delay_greylisting_error_until_DATA`: if greylisting (4g) is enabled then wait until the `DATA` command to return an error; see Section 3.9.4 for details.

(g) `greylisting`: enable simple greylisting; this also requires the access map (see 3.9.3) and must be enabled in SMAR, see Section 3.9.2 item 6), see Section 3.9.4 for details.

(h) `fullgreylisting`: enable full greylisting (which must also be enabled in SMAR, see Section 3.9.2 item 6), see Section 3.9.4 for details.

(i) `rcpt_conf`: request SMAR lookup of `RCPT` address in the access map using the tag `smtps_rcpt_conf`. Currently the RHS for this kind of entry is a `flags` option and the only possible value is `greylisting`, which allows to turn on greylisting per `RCPT`, e.g.,

```
 smtps_rcpt_conf:some@local.domain  flags = greylisting;
```

Note: this will change in a subsequent version (syntax and features) as soon as some basic concept has been determined.

(j) `lmtp_does_not_imply_relaying`: even if a domain in the mailertable has `lmtp:` as RHS do not implicitly allow relaying to it, i.e., do not consider the domain as "local" with respect to relaying. This is useful for an MSA to avoid external mail to local domains without authentication.

(k) `lookup_session_conf`: Look up session configuration data (see Section 3.10.1) in the access map (which must be activated, see 4b).

(l) `require_EHLO_before_MAIL`: require `EHLO` (or `HELO`) before a `MAIL` command.

(m) `soft_bounce`: change permanent (5xy) SMTP error replies into temporary (4xy) errors. This is a useful feature for testing to avoid bounces due to misconfigurations.

(n) `strict_EHLO_checks`: perform a strict syntax check on the argument for `EHLO` (or `HELO`).

(o) `smtps`: enable *smtps* submission service, i.e., implicit TLS handshake, as described in RFC 8314.

(p) `xverp`: offer `XVERP` extension to turn on `VERP` [Ber97] support for a transaction. This is a parameter for the `MAIL` command, e.g.

```
MAIL From:<sender@some.domain> XVERP
```

(q) `prdr`: `PRDR` is an SMTP extension [Hal07] to return individual `RCPT` status after the end of data (similar to LMTP). This feature is turned on if a policy milter is configured (see item 19).

5. `greeting`: initial `ESMTP` greeting text. Unless the greeting string starts with the text "220" and ends with CRLF (\r\n), it will be appended to the greeting code `220`, and the host name (separated by a space each). Otherwise, the entire string will be used verbatim. To specify a multiline greeting, use something like:

```
greeting = "220-host.name Welcome to my fancy ESMTP server\r\n"
   "220-Sending UBE is forbidden.\r\n220 Violators will be prosecuted.\r\n";
```

The default greeting text consists of the code 220, the host name, the word "ESMTP" and the MeTA1 version (separated by spaces). Note: RFC 2821 requires that the greeting has the form

```
220 FULLY.QUALIFIED.HOST.NAME optionally more text
```

that is, the first word after the 220 code *must* be the host name, hence these two words are usually prepended to the specified greeting text.

6. `id`: unique identifier for SMTP server (0); see Section 3.10.2.
7. `IO_timeout`: timeout for SMTP operations.
8. `threads_max`: maximum number of threads.
9. `bad_commands_per_session_max`: maximum number of bad, i.e., unknown, SMTP commands per session accepted by server. After this limit is reached the connection is terminated with an 421 error.
10. `invalid_addresses_per_session_max` maximum number of invalid, e.g., unknown, RCPT addresses per session accepted by server. After this limit is reached the connection is terminated with an 421 error.
11. `nop_commands_between_transactions_max`: maximum number of NOOP, RSET, and related SMTP commands between two successful transactions accepted by server. After this limit is reached the connection is terminated with an 421 error.
12. `bad_commands_per_transaction_max`: maximum number of bad, i.e., unknown, SMTP commands per transaction accepted by server. After this limit is reached the connection is terminated with an 421 error.
13. `nop_commands_in_transaction_max`: maximum number of NOOP and related SMTP commands in a single transaction accepted by server. After this limit is reached the connection is terminated with an 421 error.
14. `invalid_addresses_per_transaction_max` maximum number of invalid, e.g., unknown, RCPT addresses per transaction accepted by server. After this limit is reached the connection is terminated with an 421 error.
15. `recipients_per_session_max`: maximum number of recipients per session.
16. `recipients_per_transaction_max`: maximum number of recipients per transaction.
17. `hops_max`: maximum number of hops (Received: headers). If this value is exceeded the incoming mail is rejected because it is considered a possible mail loop.
18. `message_size_max`: maximum message size (unit: KB).
19. `policy_milter`: this is a subsection that specifies the parameters for pmilter support (see Section 5). It is only available if it has been enabled during `configure` (`--enable-pmilter`, see Section 2.2.1).
    (a) `socket`: this is a subsection that specifies the socket to communicate with policy milter. The type (option `type`) of the socket must be either `inet` or `unix`.
        i. `type = inet`
            A. `port`: port number for connection.

B. `address`: IP address for connection.

ii. `type = unix`

A. `path`: pathname of Unix Domain socket.

(b) `timeout`: maximum amount of time to wait for a reply from a policy milter.

(c) `flags`: policy milter flags. If the connection to pmilter fails then SMTPS will ignore pmilter by default. This behaviour can be changed by setting one of following two flags:

i. `abort`: if the connection to pmilter fails then abort the current session with a 421 error.

ii. `accept_but_reconnect`: if the connection to pmilter fails then continue the current session but try to reconnect for the next session.

20. `processes`: number of processes to start.

21. `protected_recipients`: this is a subsection which provides a few simple options to *protect* recipients by restricting who can send mail to them.

(a) `allow_by`: this is a required subsection which has two possible flags (at least one must be specified).

i. `sender`: allow sending mail based on the envelope sender (`MAIL`) address. Even though this address can be forged it provides some basic protection.

ii. `client_ip`: allow sending mail based on the client IP address.

(b) `match_type`: this specifies what type of matching should be done. By default, exact matches are required. Alternatively, one of the following two options can be selected:

i. `generic_lookup`: the items are looked up according to the algorithm specified in Section 3.12.1.

ii. `implicitly_match_detail`: the items are looked up according to the algorithm specified in Section 3.12.1. and additionally `+detail` is implicitly matched when the pattern is "`user@hostname`". That is, it overrides the default matching explained in case 1e in Section 3.12.1.

See Section 3.10.3 for details.

22. `transactions_max`: maximum number of transactions per session.

23. `tls`: this is a subsection that specifies the parameters for `STARTTLS` support. It is only available if the system has been configured with the option `--enable-TLS`, see Section 2.2.1. See Section 11.1 for some background information about these options. Note: this description lists all options, but not all of them are available for the SMTP server and client.

(a) `cache_size`: size of TLS session session cache (0: disable cache) (see Section 11.3).

(b) `cache_timeout`: timeout for entries in TLS session session cache (only smtpc).

(c) `cert_file`: file with certificate in PEM format.

(d) `key_file`: file with private key for certificate in PEM format.

(e) `CAcert_file`: file with CA certificate in PEM format.

(f) `CAlist_file`: file with CA certificates in PEM format (for `SSL_CTX_set_client_CA_list(3)`; only smtps unless TLSv1.3 is used and the TLS library supports `SSL_CTX_set0_CA_list(3)`). Notes: if `CAlist_file` is not set, then `CAcert_file` will be used, if `CAlist_file` is set to `""`, then `NULL` will be passed to (for `SSL_CTX_set_client_CA_list(3)`), otherwise the specified value will be used.

(g) `CAcert_directory`: directory with (symbolic links for) CA certificates in PEM format.

(h) `options`: TLS related options, see `SSL_CTX_set_options(3)` and `SSL_set_options(3)`.

(i) `CipherList`: specify list of ciphers to use, see `SSL_CTX_set_cipher_list(3)` and `SSL_set_cipher_list(3)`. Note: this only works for protocol versions before TLSv1.3. The format of the string (list) is explained in `ciphers(1)`.

(j) `CipherSuites`: specify list of ciphers to use, see `SSL_CTX_set_ciphersuites(3)` and `SSL_set_ciphersuites(3)`. Note: this only works for protocol version TLSv1.3.

(k) `CRL_file`: Name of file that contains certificate revocation status information.

(l) `CRL_directory`: Name of directory that contains certificate revocation status information.

(m) `Digest`: specify the digest to use to calculate the fingerprint of a cert. Default: sha1. This affects logging but more importantly the possible restriction of STARTTLS connections, see section 3.13, item 1. See `dgst(1)` for a list.

(n) `DHParam`: specify DH parameters for the server. This is a section which provides one of the follow options:

i. `bits` to specify the size of the DH parameters to be generated which must be one of 2048, 1024, or 512, or

ii. `file` to specify a file containing the DH parameters to use.

(o) `flags`: some flags are available to influence the behaviour of the SMTP server with respect to STARTTLS.

i. `allow_relaying_if_verified`: if the client presented a certificate that can be verified by the CA certificates that are available to the server (see above: `CAcert_file` and `CAcert_directory`), then relaying is allowed for the SMTP session.

ii. `check_access_map_for_relaying`: if this flag is set then the access map (which must be activated, see 4b) is checked to see whether relaying should be allowed for a client which presented a certificate that has been verified (see above). For this purpose, the DN of the cert issuer is looked up in the access map using the tag `certissuer:`. If the resulting value is `relay`, relaying is allowed. If it is `cont`, the DN of the cert subject is looked up next in the access map using the tag `certsubject:`. If the value is `relay`, relaying is allowed; every other value is currently ignored.
To avoid problems with the DN names in map lookups, they are modified as follows: each non-printable character and the characters '<', '>', '(', ')', '"', '+', ' ' are replaced by their hexadecimal ASCII value with a leading '+'. For example:
`/C=US/ST=California/O=endmail.org/OU=private/CN=`
`Darth Mail (Cert)/emailAddress=darth+cert@endmail.org`
is encoded as:
`/C=US/ST=California/O=endmail.org/OU=private/CN=`
`Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org`
Examples:
To allow relaying for everyone who can present a cert signed by
`/C=US/ST=California/O=endmail.org/OU=private/CN=`
`Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org`
simply use:
```
 certissuer:/C=US/ST=California/O=endmail.org/OU=private/CN=
 Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org  relay
```
To allow relaying only for a subset of machines that have a cert signed by
`/C=US/ST=California/O=endmail.org/OU=private/CN=`
`Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org`
use:

```
certissuer:/C=US/ST=California/O=endmail.org/OU=private/CN=
Darth+20Mail+20+28Cert+29/emailAddress=darth+2Bcert@endmail.org   cont
CertSubject:/C=US/ST=California/O=endmail.org/OU=private/CN=
DeathStar/emailAddress=deathstar@endmail.org                      relay
```

Notes:

- if a CN contains a bogus value (e.g., an embedded NUL), then it will be represented by a textual error description instead:

BadCert:TooLong	CN is too long
BadCert:ContainsNUL	CN contains an embedded NUL
BadCert:Unknown	other, unspecified error while extracting CN

- if a DN contain an unprintable character, then it will be replaced by a backslash, an 'x', and the hexadecimal value (two digits), e.g., an embedded NUL is shown as `\x00`.
- line breaks have been inserted after `CN=` for readability, each tagged entry must be one (long) line in the access map.

iii. `request_cert`: request a (client) cert. This flag is set by default and can be turned off via the usual negation methods, (see section 8.1.1) e.g., `dont_request_cert`. This can be useful for broken clients that fail the TLS handshake if a cert is requested.

iv. `use_chain`: Use `SSL_CTX_use_certificate_chain_file(3)` to load a certificate chain instead of just `SSL_CTX_use_certificate_file(3)`.

(p) `violation`: This option decides how to (globally) handle TLS requirement violations (see Section 3.10.1). Possible values are: `permfail`: generate permanent failure, `tempfail`: generate temporary failure, `abort`: abort session. These options take effect for session/recipient restrictions when a `MAIL`/`RCPT` command is (about to be) issued by the client, respectively.

(q) `verify_depth`: specify the limit up to which depth certificates in a chain are used during the verification procedure. If the certificate chain is longer than allowed, the certificates above the limit are ignored, see `SSL_CTX_set_verify_depth(3)`).

24. `session_features`: This section can be used to define classes of features for an SMTP server session which then can be referenced via the access map.

    There are currently two options allowed:

    - `flags` Available flags are a subset of those listed in item 4: `starttls`, `auth`, `delay_checks`, `allow_data_before_greeting`, `allow_space_after_colon_for_MAIL_and_RCPT`, `require_EHLO_before_MAIL`, `strict_EHLO_checks`, `check_EHLO`.
    - `tls` this subsection can specify some parameters for `STARTTLS` support (see 23 for details). Valid options are: `cert_file`, `key_file`, `CAlist_file`, `flags`, `options` (note: this only sets additional options, see `SSL_set_options(3)`), and `CipherList`, as well as the `flags` section.

    Example: to turn off `STARTTLS` for some clients, declare a `notls` class in the configuration file:

    ```
    session_features  notls { flags = { -starttls }}
    ```

    and reference it in the access map for those clients:

    ```
    smtps_session_conf:217.126.135.148 session_feature=notls;
    ```

3.10.1 SMTP Server Session Configuration

Some options can be set via the access map (see Section 3.9.3, tag `smtps_session_conf`) because they apply to a session, not globally. Currently available are: STARTTLS requirements (`tls_requirements`) (see Section 3.13) and `session_features` (see Section 3.10, item 24).

Note: there are two tags to look up entries for a server session: `smtps_session_conf:` and `smtps_se_conf:`, they are tried in the specified order, which means the client IP address is checked first and only if no entry is found its hostname is checked next.

3.10.2 Multiple SMTP Servers with different Configurations

The normal way to run multiple SMTP servers is to let MCP start several SMTP servers. Each SMTP server must given a unique identifier (see Section 3.10, item 6) and each SMTP server section in `meta1.conf` must have a unique name (e.g., MTA and MSA), which is passed via the option `-N` *name* to `smtps`. Examples: `meta1.conf`:

```
smtps MTA {
  id = 0;
  listen_socket { type=inet; port = 25; }
  start_action = pass; pass_fd_socket = smtps/mtafd;
  user = meta1s;
  path = /usr/libexec/smtps;
  arguments = "smtps -N MTA -f /etc/meta1/meta1.conf";
  log { facility = mail; ident=meta1-MTA; }
}

smtps MSA {
  id = 1;
  DKIM_signing { domain=example.net; selector=test;
    keyfile=DKIM.private.key; }
  listen_socket { type=inet; port = 587; address=127.0.0.1; }
  start_action = pass; pass_fd_socket = smtps/msafd;
  user = meta1s;
  path = /usr/libexec/smtps;
  arguments = "smtps -N MSA -f /etc/meta1/meta1.conf";
  log { facility = mail; ident=meta1-MSA; }
  auth { trusted_mechanisms = "CRAM-MD5 DIGEST-MD5";
         flags = { noplaintext }; } }
```

Or for IPv4 and IPv6 on a dual-stack OS (note: on a hybrid dual-stack OS a single inet6 listener can be sufficient, see also 12.1.1):

```
smtps IPv6 { id = 0;
  listen_socket { type=inet6; port = 25; }
  start_action = pass; pass_fd_socket = smtps/ipv6;
  user = meta1s;
  path = /usr/libexec/smtps;
  arguments = "smtps -N IPv6 -f /etc/meta1/meta1.conf";
```

```
}

smtps IPv4 { id = 1;
  listen_socket { type=inet; port = 25; }
  start_action = pass; pass_fd_socket = smtps/ipv4;
  user = meta1s;
  path = /usr/libexec/smtps;
  arguments = "smtps -N IPv4 -f /etc/meta1/meta1.conf";
```

For tests it is also possible to let MCP start only one SMTP server which creates several copies of itself if multiple daemon addresses are specified (see Section 3.10, item 1). Note: this only works for unprivileged ports because the SMTP server does not run as root.

3.10.3 Protecting Recipients

A few simple features are available to *protect* recipients by restricting who can send mail to them. To do this the configuration section `protected_recipients` must be turned on and at least one of the two flags `sender` and `client_ip` must be selected (see Section 3.10, item 21a). If this is done, then every recipient is looked up in the access map (which must be activated, see Section 3.10, item 4b), using the tag `protectedrcpt:`. If a matching entry is found, it must have a list of (one or more) restrictions, each of which must be one of the following:

restriction	required `allow_by` flag
`from:`sender	`sender`
`cltaddr:`IP-address	`client_ip`
`list:`alias	`sender`

The meaning of the first two restriction types should be obvious, the third one is interesting: it refers to an alias (in the aliases map, see Section 3.9.3) and requires that the sender address matches one of the entries to which the alias expands. This can be used to allow only subscribed members of a mailing list to send mail to it.

The restrictions are evaluated sequentially, if there is a match, the recipient is accepted (*sequential OR*). If none of them matches, the recipient is rejected. By default a permanent error is returned, but if the entry `temp:` is in the RHS, a temporary error is used.

By default exact matches are required. However, if the flag `generic_lookup` is set (see Section 3.10, item 21b), the items are looked up as specified in Section 3.12.1. The flag `implicitly_match_detail` is useful for the `list:` restriction if a sender uses `+detail` without having that specified during subscription.

Examples: consider the following aliases map:

```
list1:  <user1-1@l1-1.dom> <user2-1@l1-1.dom> <list2@local.dom>
list2:  <user1-2@l2-1.dom> <user2-2@l2-2.dom>
list3:  <user1-3@l3-1.dom> <user2-3@l3-2.dom>
```

together with this access map:

```
protectedrcpt:list1@local.dom  list:<list1@local.dom>
protectedrcpt:list3            from:<moderator3@local.dom> cltaddr:1.2.3.4 cltaddr:10.
```

The mails to `<list1@local.dom>` are only accepted from `<user1-1@l1-1.dom>` and `<user2-1@l1-1.dom>`. Note: the list is not recursively expanded, i.e., members of `list2` are not allowed, that restriction must be listed in the access map. Mails to `<list3@local.dom>` are only accepted from `<moderator3@local.dom>`,

the client with the IPv4 address 1.2.3.4, or clients in the IPv4 net 10. The latter requires that the flag `generic_lookup` is turned on too.

3.11 Configuration for SMTP Client

The following configuration options are valid for SMTPC:

1. `IO_timeout`: timeout for SMTP operations (unit: s).
2. `LMTP_socket`: Unix domain socket to use for LMTP [default: `lmtpsock`].
3. `log_level`: logging level.
4. `tls`: this is a subsection that specifies the parameters for `STARTTLS` support. It is only available if the system has been configured with the option `--enable-TLS`, see Section 2.2.1. See Section 11.1 for some background information about these options. The options are basically the same as for the SMTP server (see section 3.10, item 23).
5. `session_features`: This section can be used to define classes of features for an SMTP client session. This is only available if the system has been configured with the option `--enable-TLS` and also requires the compile time option `SC_SESS_FEATS`.

   (a) `flags`:
   - `starttls` can be used to turn off (see Section 8.1.1) the use of `STARTTLS` for servers with a broken implementation.

   For other flags see Section 8.6, item 3.

   (b) `tls`: this is a subsection that can be used to set some STARTTLS related options, which are a subset of those available globally (see Section 3.10, item 23): `cert_file`, `key_file`, `flags`, `options`, `CipherList`, and `CipherSuites`.

   Session features can be selected via the configuration map for QMGR (Section 3.8.1) using the following tags and arguments:

   (a) tag `smtpc_session_conf`, argument: IP address of server to which smtpc connects.
6. `wait_for_server`: maximum amount of time to wait for a server (QMGR) to become available (unit: s).

3.11.1 SMTP Client Session/Recipient Configuration

Some options can only be set via the access map (see Section 3.9.3, tag `smtpc_rcpt_conf`) or the configuration map for QMGR (see 3.8.1, tag `smtpc_session_conf`) not in the SMTP client configuration section itself, as they apply to a session or recipient, not globally. Currently mostly STARTTLS requirements are available which are documented in Section 3.13, more options might be added later on. Note: as more options might be added, the structure of this may change a bit.

For smtpc the additional flag `delay` is available for `violation` when used with `smtpc_rcpt_conf`. If set, smtpc sends a `RCPT` to the server even if its specified TLS requirement is violated, but then aborts the entire transaction before `DATA`. This also changes the behaviour for multiple `RCPT`s: without the flag a message for `RCPT`s without TLS requirement violations would be sent, but with the flag a single `RCPT` with

a TLS requirement violation will cause the entire message to be aborted. This flag has been added to deal with a potential *deadlock*: If a server uses greylisting without STARTTLS and MeTA1 is configured with TLS requirements for a `RCPT` at that server, then the mail will never go through unless the `delay` flag is set because otherwise the greylisting server will never receive that `RCPT`. Note: such a setup should be very unlikely because specifying explicit TLS requirements for a `RCPT` address usually means there is some agreement between the client and the server about those and hence the server would not apply greylisting to the client.

3.12 Lookup Orders

3.12.1 Lookup Orders in Maps

In many cases an item is not just looked up verbatim in a map, but it may be split into logical parts and then less significant parts are iteratively removed and the remaining data is looked up until either a match is found or the data is empty; in the latter case a default key may be looked up depending on the map. These steps can be controlled by flags that are specified in the configuration file for the map. These flags are listed below for the various lookup steps.

For domain names of the form "`sub2.sub1.tld`" the lookup order is "`sub2.sub1.tld`", "`.sub1.tld`", "`.tld`", and "`.`" (without the quotes). The subdomains are tried if the flag `dotsubdomain` is set. The last lookup ("`.`") is only done if the flag `dot` is set, as it is the default for mailertable. Obviously this schema is extended if more components are specified. As the sequence shows there is no implicit "match all subdomains" lookup, instead entries in a map must have a leading dot for subdomains matches. To reiterate: "`sub2.sub1.tld`" does neither match the entry "`sub1.tld`" nor "`tld`".

For IPv4 addresses of the form "`A.B.C.D`", the lookup order is "`A.B.C.D`", "`A.B.C.`", "`A.B.`", and "`A.`" (without the quotes). For IPv6 addresses the corresponding algorithm is used for lookups. Note: IPv6 addresses are represented in the *non-compressed* form, i.e., `::` to denote a sequence of zeros is not used. This allows the map lookup mechanism of removing less significant parts to work.

For RFC 2821 addresses of the form "<`user+detail@domain`>", where "`+detail`" is optional and "`+`" is the first delimiter in the localpart that was found in the `address_delimiter` string (see Section 3.9.2), the lookups are done according to the following sequence:

1. Repeat the following lookups for each subdomain of domain (as explained above):
   (a) "`user+detail@subdomain`" if "`+detail`" exists; this is a verbatim match. Flag: `full_address`.
   (b) "`user++@subdomain`" if "`+detail`" exists and "`detail`" is not empty; this matches any non-empty "`+detail`". Note: the second "`+`" character is a fixed metacharacter, it does not depend on `address_delimiter`; it is a modelled after the "`+`" operator in regular expressions etc to denote a non-empty sequence of items. Flag: `detail_plus`
   (c) "`user+*@subdomain`" if "`+detail`" exists; this matches any "`+detail`" (including just "`+`"). Flag: `detail_star`
   (d) "`user*@subdomain`"; this matches "`user@subdomain`" as well as "`user+detail@subdomain`" ("`detail`" can be empty). *Note*: "`*`" is not a generic metacharacter here, it matches only a token beginning with `address_delimiter` or an empty sequence, it does *not* match any other character sequence. For example: the input "`user1@subdomain`" does not match the LHS "`user*@subdomain`". Flag: `star`
   (e) "`user@subdomain`"; this does *not* match if "`+detail`" exists, unless the flag `implicitly_match_detail` is selected for the map to implicitly match a detail even if there is no wildcard in the pattern.

(f) "`@subdomain`".

2. If nothing has been found and the map type requests it, then try localpart only (with the same meaning as above):
   (a) "`user+detail`" if `detail` exists
   (b) "`user++`" if `detail` exists and is not empty,
   (c) "`user+*`" if `detail` exists,
   (d) "`user*`",
   (e) "`user`"

3.12.2 Lookup Orders for Anti-Spam Measures

Map lookups for anti-spam measures are performed according to the SMTP dialogue, i.e., connection information (`cltaddr:` and `cltname:`), `MAIL` command (`from:`), and `RCPT` command (`to:`). Whether a rejection has an immediate effect depends on the result of the lookup, e.g., the `quick:` modifier, and whether the option `delay_checks` is set. If multiple checks are performed during a single stage of the SMTP dialogue then they are done sequentially until one of them returns something else than `cont`.

Note: in the description of the algorithms below some items are marked as *check:*. Only those can change the result value, other steps perform just operations that may be needed later on but have no immediate effect on the outcome of the checks.

Connect

During connect the following operations are performed if the `access` flag (see Section 3.10, item 4b) is enabled:

1. start PTR lookup for client IP address,
2. *check:* look up client IP address using tag `cltaddr:` (as explained in 3.12.1)
3. map client IP address to client hostname returning the tuple (`clientresolve`, `clientname`).
4. *check:* look up `cltresolve:clientresolve`
5. *check:* look up `cltname:clientname` (as explained in 3.12.1)
6. start DNS blacklist queries,
7. *check:* look up results of DNS blacklists in access map.

MAIL

After a `MAIL` command has been received the following checks are performed unless the address is `<>` or a session check resulted in `quick:relay` or `quick:ok`:

1. *check:* is the address routeable? That is, if the sender address would be used as a recipient address (as it would be necessary if a DSN must be sent) is it possible to find a host that will deal with the address? This means that the domain part must have a valid MX or A record or that routing is specified via mailertable.

2. *check:* if the domain of sender address is local: is the local part valid?
3. *check:* look up the address in the access map (provided it is enabled) with the tag `from:`.
4. *check:* look up the IP addresses that were found when trying to determine whether the address is routeable with the tag `mxbadip:` in the access map.

RCPT

A `RCPT` command causes different checks (note: these are sequential checks, not exclusive, i.e., if the first one applies and it does not return a decision, the second one is performed):

1. is this a relaying attempt and if so, is it authorized? Relaying can be allowed for the entire session, e.g., due to the client IP address or other authorization based on some authentication (STARTTLS, AUTH).
2. are there any other restrictions for the recipient address?

For case 1 the following tests are performed:

1. *check:* is the recipient local and does the address exist? If yes, it is not a relaying attempt and hence allowed.
2. *check:* look up the recipient address with the tag `to:` (provided the access map is enabled) (provided the access map is enabled) and check whether the RHS is `relay`, otherwise reject the `RCPT` command as unauthorized relaying attempt unless the RHS is an `error` entry which is then used as reply.

For case 2 the following steps are taken:

1. look up address with tag `protectedrcpt:`, if found perform all the necessary checks as explained in Section 3.10.3.
2. *check:* look up the address using the tag `to:` if all of the following three conditions are met:
   (a) the access map is enabled.
   (b) a session check did not result in `quick:relay`.
   (c) a session check did not result in `quick:ok` and relaying is allowed by other means.
3. *check:* perform greylisting unless the session is marked as `ok` or `relay` or the `RCPT` lookup returned `quick:ok` or `quick:relay`.

3.12.3 Macro Replacements in RHS

The alias map allows the use of macro in the right hand side of map entries. Macros have the form "`${name}`" (without the quotes). Available macros are: `user`, `detail`, `domain`, `tag`, `delimiter`, `subdomain`, `extension`. They have the obvious meaning; `subdomain` refers to the part of the domain before the dot, i.e., if the pattern is `@.domain` and the input is `user@host.domain` then `subdomain` refers to `host`, `extension` is the delimiter and the detail together (provided the address contains them).

Example:

```
alias*@.domain     user${extension}@${subdomain}.domain
```

provides the following mappings:

```
alias@host.domain              user@host.domain
alias+detail@host2.domain      user+detail@host2.domain
```

3.13 STARTTLS Restrictions

STARTTLS requirements can be set for the SMTP server and SMTP client via map entries (see Sections 3.10.1 and 3.11.1). These restrictions are listed in a section called `tls_requirements`.

1. `cert_fps` A list of fingerprints, one of which must match the fingerprint of the presented TLS certificate. By default, SHA1 is used to calculate the fingerprint, but this can be changed via the `Digest` option (see section 3.10, item 23m).
2. `cert_issuer` require that the DN (Distinguised Name) of the issuer of the presented TLS certificate matches the specified value.
3. `cert_subject` require that the CS (CERT subject) of the presented TLS certificate matches the specified value.
4. `cipher_bits_min` require that the effective keylength (in bits) of the symmetric encryption algorithm used for a TLS connection is at least as big as the specified value.
5. `common_name`: require that the CN (Common Name) of the presented TLS certificate matches the specified value.
6. `flags`: available flags are currently:
   (a) `verified`: the TLS connection must be verified, i.e., the cert issuer must be listed in `CAcert_file` or `CAcert_directory` (see Section 3.10, item 23).
   (b) `encrypted`: the TLS connection must be encrypted, i.e., the effective keylength (in bits) of the symmetric encryption algorithm used for a TLS connection is greater than zero.
7. `hostnames` A list of hostnames, one of which must match a DNS value in subjectAltNames or the CN of the presented TLS certificate. This list can contain FQHN, or a domain name with a leading dot. A DNS name in subjectAltName can contain a leading wildcard * which matches exactly one level in the FQHN. The rules for matching are (first name refers to an entry in `hostnames`, the second name refers to an entry in the cert):
   (a) exact match.
   (b) `host.tld` matches `*.host.tld`. Note: `*.tld` in a cert is not accepted.
   (c) `.host.tld` matches `anything.host.tld` where `anything` can be "*" or one or more subdomains.

   Note: This restriction is only useful in conjunction with the `verified` flag.

8. `root_ca_subjects` A list of CNs, one of which must match the CN of the root CA cert that signed the presented TLS certificate. This feature is useful to restrict the list of CA that should be accepted for a server as the list of CAs is currently only global and hence applies to all TLS connections. Otherwise a compromised CA could be abused to create a bogus intermediate cert that is used to sign a cert that is required by other restrictions, e.g., `common_name`, `cert_subject`, or `cert_issuer`, and thus circumvent the security of a connection.

9. `violation` See Section 3.10, item 23p, for possible values.

Note: The options `cert_fps`, `hostnames`, and `root_ca_subjects` are currently only implemented for the SMTP client (smtpc).

Examples (note: the entry must be currently written in a single line to be properly processed by `createmap(8)`):

```
smtpc_rcpt_conf:@meta1.org tls_requirements { cert_issuer="/C=US/ST=Berkeley/L=Endmail+20Org/O=MTA/CN=C
smtpc_rcpt_conf:@example.org tls_requirements { flags={verified}; hostnames={ server.example.org, serve
```

3.14 VERP

There are three ways to enable VERP [Ber97]:

1. Turning it always on in QMGR (see Section 3.8, item 19). Note: this option overrides the next two.

2. Via the XVERP extension in the SMTP server (see Section 3.10, item 4p).

3. Via an entry in aliases marking a list with `verp-`:

```
otherlist:  <user5@my.dom> <user6@other.dom> <user7@local.host>
verp-otherlist:  something
```

Chapter 4

Running MeTA1

4.1 Starting MeTA1

All components of MeTA1 are under control of the MCP which must be started as `root` in the directory `/var/spool/meta1` (i.e., the main queue directory, see Section 2.4: `MTAQDIR`) using

```
# ./mcp.sh start
```

The script contains the runtime path for MCP based on the data used by `configure` as well as a reference to the MeTA1 configuration file.

To stop all MeTA1 components use

```
# ./mcp.sh stop
```

or simply terminate the MCP, it will forward the signal to all processes it started.

The MCP provides some restart functionality: if a process terminates MCP will restart it depending on the `exit(3)` code and prior behaviour. `exit(3)` codes are classified in several categories:

1. *fatal* (e.g., `EX_USAGE`): the service will not be restarted.
2. *hard* (e.g., `EX_UNAVAILABLE`): the service will be restarted, unless a *hard* restart limit is exceeded within a certain time.
3. *soft* (e.g., `EX_TEMPFAIL`): the service will be restarted. unless a *soft* restart limit is exceeded within a certain time.
4. *ok*: the service will be restarted, unless a restart rate is exceeded.

Moreover, the processes listed in the restart dependencies will be stopped and started too.

4.2 Using MeTA1 only for Outgoing Mail

MeTA1 can be used in combination with a MUA that speaks (E)SMTP directly or with the sendmail 8 MSP (Mail Submission Program) for outgoing mail. For the latter add this to your sendmail 8 `submit.mc`

file (see also misc/sm8.submit.mc):

```
LOCAL_RULE_0
R$* + X<@$*>    $#meta1 $@ localhost $: $1 <@$2>

LOCAL_RULESETS
SHdrToSMTP
R$+                     $: $>PseudoToReal $1            sender/recipient common
R$+                     $: $>MasqSMTP $1                qualify unqual'ed names
R$* + X<@$*>            $: $1 < @ $2 >
R$* < @ *LOCAL* > $*    $: $1 < @ $j . > $2

MAILER_DEFINITIONS
Mmeta1,   P=[IPC], F=kmDFMuXa, S=EnvFromSMTP/HdrFromSMTP, R=EnvToSMTP/HdrToSMTP,
        E=\r\n, L=990, T=DNS/RFC822/SMTP,
        A=TCP $h 2009
```

and run the SMTP server of MeTA1 as listener on `localhost:2009`. Then mail to <user+X@domain> will be sent via MeTA1, i.e., by adding the sequence `+X` to the address <user@domain> the mail will be redirected to MeTA1 (and `+X` will be removed). After initial testing the `relay` mailer can be changed to use port 2009 by default hence the local additions shown above can be removed.

There are also other programs available as substitute for the command line invocation of `sendmail` as mail submission program, e.g., `mini_sendmail` [Posb].

4.3 Using MeTA1 for Incoming Mail

4.3.1 Local Delivery and Specifying Local Domains

If the domain of a recipient address matches an entry in mailertable (see Section 3.9.3) with the right hand side `lmtp:`[1] then SMTPC talks LMTP over the local socket `lmtpsock` (see 3.11). If you have an LDA that runs as daemon and can talk LMTP over a local socket you can use it for local delivery. It is also possible to use `procmail` [vdBG] in LMTP mode and start it from `mcp`, see `meta1.conf`. See `contrib/procmail.lmtp.p0` for a patch[2] for procmail 3.22 to allow handling of addresses with extensions (`+detail`) in LMTP mode. A mailertable for local delivery via LMTP should look like this:

```
localhost     lmtp:
MY.DOM        lmtp:
HOST.MY.DOM   lmtp:
```

By default mail to addresses whose domain part is listed in mailertable with RHS `lmtp:` is allowed, i.e., those domains are considered local and hence relaying (even though technically this might not be called relaying) to them is allowed. This behaviour can be turned off (see Section 3.10, item 4j) in which case it is necessary to also allow relaying to these domains which can be done either via the access map (see Section 3.10, 4b), or the command line option `-T` for SMTPS. This allows for treating (some of) these domains as *private* by not allowing relaying to them, hence they will be only reachable from systems from which relaying is allowed.

[1] currently internally encoded as `127.0.0.255`

[2] if the patch fails to apply, make sure you use a working `patch(1)` program, e.g., most SunOS 5.x versions ship with a broken program

4.3.2 Specifying Valid Local Addresses

To validate addresses for local domains, SMAR uses the map `aliases.db` (Section 3.9.2, item 3a), which can be created using `createmap`, or a map specified by the option `local_user_map` (see Section 3.9.2, item 7). The key in the map must be the local part of a valid (local) e-mail address. If the local part cannot be found in either map, the address is rejected.

To list valid local addresses in the alias map the right hand side must be the string "`local:`", e.g.,

```
postmaster:  <user@host.domain>
abuse:       user+abuse
user++:      local:
user:        local:
```

Note: local addresses are checked for the envelope recipient and sender.

4.4 Using MeTA1 as Gateway

MeTA1 can easily be used as an internet gateway. To override routing, mailertable entries (see Section 3.9.3) can be specified. A list of valid addresses can be made available via the access map by allowing relaying to those addresses instead of entire domains, e.g.,

```
to:user1@my.domain       relay
to:user2@my.domain       relay
to:postmaster@my.domain  relay
cltaddr:10.12.           relay
```

4.5 Using MeTA1 as Backup MX Server

The previous section showed how to specify valid remote addresses if all of them are known. However, for systems that act as backup MX servers it might not be simple to always keep such a list up to date. In that case, a default entry for a domain should be made, e.g.,

```
to:user1@other.domain       relay
to:user2@other.domain       relay
to:postmaster@other.domain  relay
to:@other.domain            error:451 4.3.3 Try main MX server
cltaddr:10.12.              quick:relay
cltaddr:127.0.0.1           quick:relay
```

The last two entries allow local systems to send mail to any user at `other.domain`; without those entries mail to unlisted users will be (temporarily) rejected and hence cannot be delivered via this system.

4.5.1 Note about Backup MX Servers

It is not a good idea to run a backup MX server `B` for a host `A` that has stronger anti-spam measures; if mails are sent to `A` via `B`, then `B` may accept them for delivery, but `A` may reject them and hence `B` has to sent bounces, which, in case of spam, are most likely to forged addresses, hence those bounces will only cause additional problems. The opposite case (`B` has stronger anti-spam measures than `A`) can

cause the rejection of mail that A actually wanted to receive. Hence B and A should have the same anti-spam measures; i.e., a system that acts as backup MX server for another one should perform the same anti-spam checks as the main MX server(s).

4.6 Miscellaneous Programs

4.6.1 Do not run programs as root User

Almost all MeTA1 programs (except for MCP) refuse to run with root privileges. To run a program as a different user the utility misc/runas can be used, e.g., after installation in /usr/local/bin/

```
# /usr/local/bin/runas meta1q mailq -V
```

(specify -h to see the usage).

4.6.2 Displaying Content of Mail Queues

The program mailq displays the content of the mail queues (defedb and ibdb). Currently its output is in a similar format as the sendmail 8 version. The option -h shows how to use the program; see the previous section about using runas for mailq. A simple shell script wrapper mailq.sh is available that invokes mailq via runas. Note: the output of this program might not be accurate due to internal buffering by QMGR. Moreover, this program reads DEFEDB in such a way that only entries that have been *checkpointed* (see Section 8.2 about options for checkpoints) are shown. This is done to avoid interference with the operation of QMGR.

4.6.3 Interacting with QMGR

The program qmgrctl allows to interact with the QMGR via the control socket (see Section 3.8, item 3). Invoke qmgrctl -h to see the available options. By default the program will show the current status of QMGR. If QMGR has been compiled with the option QMGR_STATS (see Section 12.1) then additional statistics is available, e.g., the number of transactions and recipients that has been handled.

Enhancements to this program are welcome to provide more functionality.

4.7 Reloading Maps

Maps (for SMAR and QMGR) can be reloaded by moving the old db file out of the way, creating a new file and then sending a USR1 signal to the appropriate process to reopen the map.

```
# mv $MAP.db $MAP.old.db
# /usr/local/bin/runas meta1q createmap -F $MAP.db < $MAP
# kill -USR1 $PID
```

Note: for QMGR it is also possible to use qmgrctl -r instead, see Section 4.6.3.

4.8 Logging

Logging is done via `syslog(3)` (see Section 3.5, 1) or to `stdout/stderr`, which is redirected by the default MCP configuration to `PROG.log`. The logging format is not yet completely consistent across programs. Moreover, the logging entries might not be easy to understand because they contain some details which are not interesting to a potential postmaster, but to developers. Nevertheless, the logging entries should show the flow of mail through the system. See Section 10.4 for an explanation of the format of logfile entries.

Note:

- logfiles *must* exist with the proper owner and permissions to be used. Neither MCP nor the modules will currently create logfiles. This is done by `make install`, i.e., `misc/sm.setup.sh`, which parses `meta1.conf` to extract the section titles/names and `user` entries to create the logfiles with the correct name and owner. This does not (yet) properly work if unique logfile names are created, see Section 8.2, 3.
- MCP uses facility `LOG_DAEMON`; this is currently not configurable. Hence log entries from MCP may end up in a different logfile, e.g., `/var/log/messages` (depending on `syslog.conf`).

4.8.1 Logfile Rotation

Unless `syslog(3)` is used (see Section 3.5, 1), logfile rotation can be achieved by copying the existing logfile to a backup file, e.g.,

```
# cp qmgr.log qmgr.log.0
```

and sending a `USR2` signal which will cause the processes to rewind the logfile. Note: the author is aware that this is not an optimal solution, however, using `syslog(3)` will usually provide a better way.

4.9 Regular Checks

There are at least two things that should be done regularly:

1. Check the logfile for errors[3]:

   ```
   $ egrep 'sev=(ALERT|CRIT|ERR|FAIL|FATAL)|\<assertion\>' $LOGFILE
   ```

2. Keep track of the size of the processes, e.g,

   ```
   $ date >> $MTAPROCS
   $ ps axuww | grep '^meta1' | sort >> $MTAPROCS
   ```

   If one of the processes continuously grows then MeTA1 should be compiled with `-DSM_HEAP_CHECK` (see Section 12.1.5) and a heap dump should be taken regularly by sending the `USR1` signal to the process. By comparing subsequent heap dumps it should be possible to locate a possible memory leak.

Please report problems that cannot be resolved locally, see Section 1.3.1.

[3] See `egrep(1)` for the correct syntax for word delimiters on your OS.

4.10 Dealing with Errors

4.10.1 Resource Problems

Resource problems in certain parts of the code can lead to a stop of the involved program. In such a case it will be restarted automatically but if the resource problem has not been taken care of the MTA may stop again. In that case manual interaction is required. The simple solution to a resource problem is of course to add more resources (RAM/disk) or to free up some resources, e.g., stopping programs that do not need to run or deleting unused files. There are also ways to control resource usage within MeTA1:

- memory shortage: the memory usage of MeTA1 can be controlled by restricting the size of various data structures, see Section 3.8, e.g., `AQ_entries_max`, `IQDB`, and `OCC_entries_max`. However, setting these values too low will result in a very slow MTA that may operate in a degraded state which is not acceptable.
- disk shortage: MeTA1 has options that let it decide how much free disk space is needed for operation, see Section 3.8: `disk_space_ok` and `disk_space_min`. However, if there is not enough space to store the envelope databases (DEFEDB and IBDB) then the system cannot work, hence sufficiently free disk space is essential for proper operation.

4.10.2 Database Problems

See Section 7.1 for some background information about the usage of the various databases before trying to fix any possible problems.

If the deferred database is corrupted then the Berkeley DB utilities to deal with such situations should be tried [Sleb], e.g., `db_recover`.

Currently messages stored in CDB have the transaction identifier (`ss_ta`, see Section 10.4) as filename. In the worst case, i.e., if IBDB or DEFEDB are destroyed, this allows to reconstruct the envelope data together with the logfile entries. See the script `misc/rcvrenvfromlog.sh` for an example, here is a description of its operation. First, check which messages are still in CDB: in the CDB directory (3.4: `CDB_base_directory`) issue:

```
# ls -1 [0-9A-F]/S*
```

Then search for each of those transaction identifiers (`$TAID`) in the logfile (`$LOG`):

```
$ egrep "ss_ta=$TAID, (mail|rcpt)=" $LOG | \
  sed -e 's;^.*\(mail=<.*>\), .*;\1;' -e 's;^.*\(rcpt=<.*>\), .*;\1;'
```

will show the sender (`mail=`) and the recipients (`rcpt=`). Based on this data it is possible to resend the messages.

Note: contributions in this area are welcome, e.g., better scripts that perform more checks and maybe allow for completely automatic recovery.

4.10.3 Writing Core Dumps

By default, all MeTA1 processes are executed in the main queue directory. As those processes are running with different user and group ids not all of them can write a core dump into that directory if a fatal error occurs. Some operating systems have features (e.g., `coreadm(1M)` on SunOS 5.x, `/proc/sys/kernel/core_pattern` on newer Linux versions[4]) to specify a different directory in which a core dump is written. On operating systems where such a command is not available, the option `working_directory` can be used (see Section 8.2, item 4).

4.11 Replacements for Features available in other MTAs

MeTA1 does currently not offer some of the features that are available in other MTAs. This section describes replacements or workarounds for some of those features.

1. Address Masquerading: The best way to use the correct e-mail addresses is to properly configure your MUA. Some MUAs offer more flexibility for this than the default masquerading features of sendmail 8, e.g., mutt [mut] allows to select sender addresses based on recipient addresses.

   Alternatively a mail submission program (MSP) can be used which offers address rewriting capabilities, e.g., the MSP from sendmail 8.

2. `.forward`: `procmail` can be used as LDA (see Section 4.3.1) and its configuration file `.procmailrc` allows to implement the same functionality as a `.forward` from sendmail 8 and some other MTAs.

3. Sending mail to programs: see previous item 2: this can be done with the help of `procmail`.

4. Appending mails to files: see item 2: this can be done with the help of `procmail`.

[4] The content of `/proc/sys/kernel/core_pattern` should be something like `/var/core/core.%e.%p.%s` and `/var/core` should have permissions `0177`.

Chapter 5

Policy Milter

5.1 Policy Milter Overview

MeTA1 has support for a policy milter which is similar to a milter in sendmail 8. The API is slightly different, however, it should be possible to write an compatibility layer to emulate the sendmail 8 API. Another difference is that MeTA1 itself only talks to a single pmilter. Support for multiple pmilters should be implemented in a *multiplexor* that connects to multiple pmilters and coordinates their responses.

Details on the API can be found in Appendix A.

Chapter 6

Miscellaneous

6.1 Troubleshooting

If something goes wrong then the component which fails usually logs an error message. Depending on the configuration, an error is either logged via `syslog(3)` or printed into a logfile (as explained in Section 4.8). Note: even if the system is configured to use `syslog(3)` (Section 3.4, item 1) errors at startup are printed to the logfile if those errors occur before the configuration is read, hence those files need to be checked too.

6.1.1 Startup Problems

If MeTA1 fails to start properly the reason should be logged as explained before. Some possible reasons are

1. invalid configuration file: use `misc/smconf` to check the syntax before deploying a new configuration file.
2. wrong permissions: check the permissions as explained in Section 2.4:

   ```
   $ ./misc/sm.check.sh
   ```

3. missing or invalid maps: make sure maps are created properly with `createmap(8)`.

6.1.2 Logfile Entries

Most logfile entries should be self-explaining. However, some are more subtle and indicate only indirectly what might be wrong. Example:

```
smtps: ... client_name=Hostname_Not_Determined
```

indicates that the access map (Section 3.10, 4b) is not used which might point to a misconfiguration.

6.2 Caveats

The following problems exist in this version of MeTA1:

- If the system runs out of memory then the MTA may not act gracefully in all cases, see Section 4.10.1.
- If a disk that is used for one of the queues becomes full, some errors may not be handled gracefully, see Section 4.10.1. To avoid this, MeTA1 has some limits for the amount of available disk space that is required to accept mail (see Section 3.8).

6.3 Checks in SMTP Server

The SMTP server has some builtin checks which are explained in the following.

6.3.1 Strict RFC Compliance

The SMTP server currently enforces fairly strict RFC 2821 compliance. For example, a `MAIL` command must be given in the following format

```
MAIL From:<user@some.domain>
```

i.e., the angle brackets are required, there must be no space after ":", etc. This has the useful side effect of catching some spam programs:

```
5.5.0 Syntax error., input=MAIL FROM: <blafwhoyqjywvu@asia.com>
```

Moreover, the server requires that lines end in CRLF (\r\n), it will not accept command input without the correct line ending, i.e., trying to do that will cause a read error.

Another requirement is that MX records *must* point to hostnames, not IP addresses [Moc87]. This applies to receiving mail – a `MAIL` address using a domain whose MX record points to an IP address will be rejected (`553 5.1.8 Sender address does not exist`) – as well as to sending mail – a `RCPT` address with a domain whose MX record points to an IP address is not resolved by SMAR.

6.3.2 Various Checks

The `EHLO` parameter is checked against the local hostname unless the connection comes from localhost (IP address 127.0.0.1) or the access map returned `quick:ok` or `quick:relay`, see Section 3.9.3. Violations will be logged with a status text of `Identity_Theft`.

The SMTP server checks for "illegal pipelining", i.e., whether a client sends commands before it is allowed to do so. Moreover, it also checks whether the client sends a command before the initial greeting. Note: according to RFC 2821 the client SHOULD wait for the greeting, but this is not (yet) a requirement. To turn this off, use `allow_data_before_greeting` (Section 3.10, item 4c.)

6.4 Security Checks

There are currently no additional security checks when creating/accessing files or directories besides those provided by the operating system. This could be a problem if MCP is misconfigured because it runs as `root`. Hence it will simply overwrite existing files if those are specified in the configuration file. The other modules run as non-privileged users, hence the OS provides sufficient access checks – unless the system is misconfigured and the MeTA1 accounts are misused for other purposes too.

6.5 Restrictions

Besides the obviously missing functionality there are some other things that may restrict the use of MeTA1 in certain environments. Here is an incomplete list:

1. DNS lookups currently use only UDP by default, hence answers that exceed the UDP packet size will cause problems. However, such DNS packets are really rare (because they cause operational problems in various places, e.g., some firewalls may block TCP for port 53). A possible workaround might be to force TCP (see Section 8.4, item 1(a)i), the correct way is a change in the DNS library to retry with TCP, but this has not yet been implemented.
2. Map lookups convert keys to lower case before checking an entry. In general this is not a problem unless local addresses rely on preserving the case of the local parts of addresses. That is, local addresses which require upper case characters do not work.
3. Multi-line replies from an SMTP server are currently neither stored (for a possible DSN) nor logged, instead just the last line is used for that purpose.

Everything that is not described in the documentation does either not exist in the current version of MeTA1, or is unlikely to work. However, there may be omissions in the documentation, please inform the author of such bugs.

6.6 Code Review, Enhancements, Patches

Source code inspection as well as patches and suggestions are very welcome.

Enhancements and extensions are very welcome too, especially to extend the basic functionality of the current MeTA1 release.

6.7 Porting

Porting to currently unsupported platforms including non-Unix systems is encouraged. Note that the destination system must support statethreads [SGI01] and Berkeley DB 4.x. It might be necessary to port those first.

6.8 Version Naming

Each MeTA1 version has a name in the following format:

MeTA1-major.minor.[qualifier]qualifier-version.patchlevel

The major number changes between releases when new features are introduced (*major* changes, but see below about the development phases). The minor number changes when no new features are introduced, but bugfixes and (portability) enhancements are made. That is, no configuration changes are needed when going from one minor version to the next. The patchlevel number is used for intermediate patches between releases, e.g., if something is broken but it is not important enough for a new release because it is barely used or encountered.

There are several different qualifiers:

1. PreAlpha: This means the software is not feature complete and hence might be missing some functionality that is considered important by different users. Additionally, there might be no compatibility in data structures stored on disk between different pre-alpha versions, e.g., when upgrading from PreAlpha16 to PreAlpha17 the main queue format may have changed without checks in the software for this. Hence old queues must be drained before upgrading. Moreover, the protocols used for communication between MeTA1 modules may have changed without providing backward compatibility, therefore modules from different releases must *not* be used together. Such incompatibilities are usually stated in the list of changes `ChangeLog`.

   Do not run this on a production server unless you are aware of the possible consequences. The software is still under development and not fully functional. Moreover, it may not be sufficiently tested.

2. Alpha: In this state the software is ready for public testing but its features may still change.

3. Beta: Feature changes are unlikely, but still possible if required. Usually only bugfixes occur between beta versions.

4. Gamma: This is a release candidate. Usually only critical bugfixes occur between gamma versions. There might be no gamma versions at all if beta testing was considered successful and sufficient.

5. A release version does not have an explicit qualifier.

The qualifier-version is used to distinguish between different version of the same qualifier, e.g., PreAlpha16 and PreAlpha17. It is 0 for a release version.

Examples for version names: MeTA1-1.0.Alpha19.0, MeTA1-1.0.0.0.

See the file `include/sm/version.h` how the version string is converted into a 32 bit number that denotes the version number.

6.8.1 Snapshots

From time to time snapshots may be made available. Those are marked with a date in the distribution file name, e.g., `meta1-0.0.16.0-20040928.tar.gz`. The name indicates that it is a snapshot of what will become version `meta1-0.0.16.0`, i.e., the next release will have the given version number (without the date). The only other indication in the distribution is the inclusion of an `s` in the version number that is shown in the version output of the main components. A snapshot did not go through the usual release cycle and is made available as *technology preview*.

Chapter 7

Data and Control Flow in MeTA1

7.1 Data Flow in MeTA1

This section explains how MeTA1 stores information about messages that are transferred. It gives some background information which is useful for troubleshooting. Details about the operation of MeTA1 can be found in [Aßmb].

MeTA1 uses two different databases on disk to store envelope information (sender and recipients): IBDB: incoming backup database, DEFEDB: deferred envelope database, and one database to store message contents: CDB: content database. See Section 2.4.1 about the location and layout of these databases[1]. The queue manager additionally uses two internal envelope databases: IQDB (Incoming Queue DataBase) and AQ (Active Queue).

Incoming mails are accepted by the SMTP servers which store the content in the CDB (complete messages including headers in the format as received). The envelope information, i.e., sender (`MAIL`) and recipients (`RCPT`), is stored by the queue manager in IQDB and written to IBDB which is just a log of envelope data and what happened to it. That is, the files in IBDB are written sequentially and are continuously growing. If a file reaches its size limit (see Section 3.8: `IBDB`), then it is closed and a new file is opened. For a delivery, the envelope information must be transferred into AQ. For incoming mail this happens as soon as a transaction is accepted, in which case the data is moved from IQDB to AQ. A transaction is only accepted if the message is safely written to CDB and the envelope information has been committed to IBDB, i.e., all information is committed to persistent storage[2].

The scheduler in QMGR takes recipient envelopes from AQ and creates transactions which are given to the SMTP clients for delivery. An SMTP client takes the transaction information and tries to send a message whose content is read from CDB. After a successful delivery attempt a record is written to IBDB that logs this information. A cleanup task removes periodically old IBDB files which contain only data that is no longer referenced.

The deferred envelope database is only used if a message cannot be delivered during the first attempt. In that case the appropriate envelope data is added to DEFEDB and a record is written to IBDB stating that the data has been transferred to DEFEDB. Entries in DEFEDB contain a timestamp called next-time-to-try at which QMGR reads them from the database into AQ and the scheduler tries another

[1] the term *database* is used loosely here, only DEFEDB is a real database, the others are just ways to store some information and access them in some way.

[2] If non-persistent storage is used for these databases mail can of course be lost.

delivery attempt. The key for DEFEDB entries is the transaction (or recipient) identifier. In order to read entries sorted by next-time-to-try QMGR keeps an internal cache (called EDBC). If that succeeds, the entries are removed from DEFEDB, otherwise they are either requeued with a new next-time-to-try (in case of a temporary error) or a DSN (bounce message) is generated (in case of a permanent error).

7.2 Caching of Routing Information

MeTA1 uses routing information it retrieved from DNS per recipient. This routing information (i.e., the IP addresses of servers that handle the recipient's domain) is stored together with the recipient in DEFEDB. It is used for delivery attempts as long as specified by the TTL[3] that was returned by the DNS queries.

Routing information retrieved from mailertable (Section 3.9.2, item 9) is treated as having a TTL of 3600s.

[3]Time To Live

Chapter 8

Advanced Configuration Options

8.1 Overview

Some configuration options are only needed in special situations and may require background knowledge of the involved systems. Those advanced configuration options are explained in the subsequent sections.

8.1.1 Flags

Usually flags are not set and hence a configuration file only needs to turn on flags (if required). However, in some cases flags are set by default and under some rare circumstances need to be disabled. To achieve this, the name of the flag can be prefixed with one of `not_`, `dont_`, `no_`, `-`, `!`, or `~`, e.g., `~remove_unused_logfiles` or `dont_remove_unused_logfiles`.

8.2 Advanced Configuration for MCP

1. `processes_max`: maximum number of processes to start [default: 1].
2. `processes_min`: minimum number of processes to start [default: 1].
3. `use_id_in_logfile_name`: if more than one process can be started then it might be useful to have unique logfiles unless the processes use `syslog(3)`. If set, this (boolean) option causes MCP to include a unique identifier (the same as for `pass_id`, which must be used too) in the logfile name. By default the logfile has the name of the section (or the section keyword if no section name is given), preceeded by the log directory (option `-L` for MCP), and `.log` appended. If `use_id_in_logfile_name` is turned on, then the numeric id is added before the extension, e.g., `/var/log/meta1/mailer0.log` for `-L /var/log/meta1/` and a section with the name `mailer`.
4. `working_directory`: perform a `chdir(2)` to the specified directory before executing the process. Note: this option essentially requires that all relevant pathnames in the configuration file are absolute, otherwise it is very easy to misconfigure some pathnames, especially those shared between different processes.

Note: the number of processes for almost all MeTA1 modules should be 1. It *must* be 1 for QMGR and SMAR, it can be larger than 1 for SMTPC. For SMTPS it should be 1 in the default setup as the file descriptor to which MCP binds on behalf of SMTPS can be passed to only one process.

8.3 Advanced Configuration for QMGR

1. `debug_level`: debug level (only if compiled with QMGR_DEBUG).
2. subsection `DEFEDB`: Note: The Berkeley DB documentation [Sleb] should be consulted before modifying any of these options.
   (a) `page_size`: DB page size (this can only be set when the DB is initially created).
   (b) `cache_size`: DB cache size.
   (c) `KBytes_written_for_checkpointing`: If non-zero, a checkpoint will be done if more than the amount of KBytes of log data have been written since the last checkpoint (unit: KB).
   (d) `delay_between_2_checkpoints`: Minimum delay between two checkpoints (unit: s).
   (e) `flags`: flags for DEFEDB:
      i. `remove_unused_logfiles`: this is on by default, hence to turn it off one of the forms explained in Section 8.1.1, e.g., `dont_remove_unused_logfiles`, can be used. This should only be done if the Berkeley DB logfiles are removed some other way, e.g., after archiving.
   (f) `hash_table_entries`: Initial size for the in-memory cache of entries in DEFEDB in order of the next time to try. This should be as large as the estimated number of entries in DEFEDB and should be a power of 2; the valid range is 128 to 67108864 (2^{26}). Note: this is not a Berkeley DB configuration option.
3. `delivery_timeout`: timeout for a single delivery attempt (unit: s). This value should be large enough that even big mails can be delivered over a slow link before the QMGR considers the delivery attempt a failure because the delivery agent did not return a result yet.
4. `flags`: configuration flags:
   (a) `reuse_connection`: try to reuse open SMTP connections for delivery. Note: this feature is still experimental.
5. `fds_max`: maximum number of file descriptors. This sets an upper limit on the number of clients that can connect to QMGR.
6. `threads_max`: maximum number of threads.
7. `threads_min`: minimum number of threads.
8. `scheduler_timeout`: as a safety measure against unforseen problems an item is removed from AQ after the specified timeout. This timeout must be large enough to allow for scheduling delays if all delivery agents are busy which can happen if deliveries are slow or if there are fewer delivery agents available than entries in the active queue.
9. `SMAR_timeout`: timeout in address resolver, i.e., how long to wait for a result from SMAR (unit: s). Note: this value must be larger than the total DNS timeout and it must take alias expansion into account.
10. subsection `smtps`:

(a) `connection_control_cache_size`: size of connection control hash table.

(b) `connection_control_hash_table_size`: size of the hash table used for connection control, i.e., number of incoming connections and connection rate (see Section 3.8, 18a and 18b)

11. `tests`: testing only (available if QMGR is compiled with `-DQMGR_TEST`). See the source code for details.

8.4 Advanced Configuration for SMAR

1. `DNS`: this subsection contains DNS related options.

   (a) `flags`: valid flags are:

      i. `use_TCP`: use TCP instead of UDP for connections to a nameserver.
      Notes:

         A. currently the DNS resolver does not automatically fall back to a TCP connection if the reply was too big. This may be added in a later version.

         B. the current implementation immediately reconnects if a DNS server closes a TCP connection, e.g., due to a timeout. Hence the DNS server should be configured such that it does not close the connection.

      ii. `use_connect`: use `connect(2)` even if using UDP. This is required on systems like FreeBSD `jail(8)`.

2. `fds_max`: maximum number of file descriptors. This sets an upper limit on the number of clients that can connect to SMAR.

3. `threads_max`: maximum number of threads.

4. `threads_min`: minimum number of threads.

8.5 Advanced Configuration for SMTP Server

1. `daemon_address`: address for daemon to listen on; this should not be used in normal operation. Current (preliminary) format is: `host:port`, `:port` (listen on 0.0.0.0) `host` (port defaults to 8000). Up to 16 addresses[1] can be specified. See the notes below.

2. `flags`:

   (a) `background`: `fork(2)` after start; this should not be used in normal operation.

   (b) `serialize_accept`: serialize `accept(2)` calls, see the statethreads documentation [SGI01] for details.

3. `DKIM_signing`:

   Note: the maximum length of the DKIM-signature header is currently 1000 byte.

   (a) `total_header_len`: DKIM signing requires to buffer the entire header before it can be fed into the hashing algorithm. Hence it is necessary to restrict the amount of memory consumed for this purpose. The default is 1 MB, the valid range is 257 B to 1 GB.

[1] Compile time option `SS_MAX_BIND_ADDRS`

(b) `header_val_len_max`: Maximum length of an individual header value. The default is 64 KB, the valid range is 257 B to 1 MB.

4. `listen_queue`: length of `listen(2)` queue; this must not be used in normal operation, i.e., if MCP is used.
5. `wait_threads_max`: maximum number of waiting threads.
6. `wait_threads_min`: minimum number of waiting threads.
7. `module_timeout`: timeout for communication with other modules.
8. `processes`: number of processes to start.
9. `wait_for_smar`: maximum amount of time to wait for a reply from SMAR.

Notes: only one of `daemon_address` and `pass_fd_socket` must be specified. In normal operation it is almost always `pass_fd_socket` because the SMTP server cannot bind to privileged ports, hence the file descriptor must be passed from MCP.

See Section 11.2 for more STARTTLS related configuration options.

8.5.1 Experimental Configuration for SMTP Server

If SMTPS is compiled with `SS_COND` then a section called `check` becomes available. This section can be specified repeatedly and must have a name. The name must be one of `connect`, `mail`, `rcpt`, `data`, or `eom`, which correspond to the SMTP stages initial connection, MAIL, RCPT, DATA, and end-of-message (final dot). These sections have two options:

1. `action`: currently only one of `permfail`: generate permanent failure, `tempfail`: generate temporary failure, `abort`: abort session.
2. `condition`: boolean expression that specifies when to apply the action. Note: for implementation-specific reasons this requires `==` instead of a single `=` after the option, e.g,

   ```
   condition == mail =~ "example.com";
   ```

Example:

```
check mail {
   condition == mail =~ "example.com";
   action = permfail; }
check mail {
   condition == mail =~ "example.org"; action = tempfail; }
check rcpt { condition == rcpt =~ "example.org";
  action = tempfail; }
```

The syntax for conditions follows common programming languages, e.g., C:

```
cond      ::=  b-expr
b-expr    ::=  a-expr rel-op a-expr
               | "!" b-expr
               | b-expr b-op b-expr
               | "(" b-expr ")"
               | match "(" s-expr "," s-expr ")"
               | s-expr "=~" s-expr
rel-op    ::=  "<" | "<=" | "==" | "!=" | ">=" | ">"
b-op      ::=  "&&" | "||"
a-expr    ::=  a-expr a-op a-expr
               | int | int-id
               | "-" a-expr
               | "(" a-expr ")"
               | comp "(" s-expr "," s-expr ")"
a-op      ::=  "+" | "−" | "*" | "/"
comp      ::=  "strcmp" | "strcasecmp"
s-expr    ::=  string | string-id
```

The available identifiers are listed below. Note: some identifiers are only defined if the corresponding configuration option, e.g., `--enable-TLS`, `--enable-SASL`, is used. Moreover, identifiers are only defined after a certain stage in the SMTP dialogue, e.g., `rcpt` cannot be used during the `MAIL` stage.

```
int-id    ::=  "alg_bits"
               | "cipher_bits"
               | "message_size"
               | "session_rcpts_total"
               | "transaction_rcpts_ok"
               | "transaction_rcpts_total"
               | "transactions"

string-id ::=  "auth_authen"
               | "auth_author"
               | "auth_type"
               | "cert_issuer"
               | "cert_subject"
               | "cipher"
               | "client_ip"
               | "client_name"
               | "client_resolve"
               | "cn_issuer"
               | "cn_subject"
               | "hops"
               | "hostname"
               | "mail"
               | "message_id"
               | "rcpt"
               | "session_id"
               | "tls_verify"
               | "tls_version"
               | "transaction_id"
```

8.6 Advanced Configuration for SMTP Client

1. `connect_only_to`: Specify an IP address to which all outgoing mail is sent. This can be used for testing with otherwise real data, i.e., addresses, by running an SMTP sink[2] on a computer and specifying its IP address. Then all mails that should be sent via SMTP will go to that host instead of the addresses determined by SMAR. Note: it is nevertheless a good idea to use firewall rules to prevent mail going out to the internet, i.e., prohibit connections to port 25 to external hosts.
2. `debug_level`: debug level (only if compiled with SMTPC_DEBUG).
3. `flags`: Note: all of these flags can all be set via `session_features` (see Section 3.11, 5)
   (a) `smtps`: use implicit TLS handshake.
   (b) `timing`: log the time it takes to complete most steps the SMTP dialogue (in microseconds), e.g., `func=sc_data, ..., t_connect=2792, t_ehlo=2240, t_starttls=396355, t_mail=420, t_rcpts=20, t_data=7333, t_msg=326, t_eom=10948, size=219, stat=0, reply=250 ok`
   (requires compile time option `SC_STATS`). Note: PIPELINING causes distortions of the timings, as the times are the interval between SMTP commands, which means that the time for synchronization points includes reading all replies for previous commands in a pipelined sequence. For example, if `MAIL`, `RCPT`, and `DATA` are issued together, the largest time will be shown for `t_data`.
   (c) `read_QUIT_reply`: read the reply to the `QUIT` command instead of just closing the connection after sending it. Note: this option causes problems with `procmail` [vdBG] as it will exit with an OS error (`EX_OSERR`) when sending the `221` reply, which in turn will triger the error handling of mcp. Hence this option should be turned off for LMTP delivery to `procmail`, or `procmail` should be patched to ignore that error.
   (d) `separate_final_dot_and_QUIT`: send the final dot of a message and the `QUIT` command in different TCP packages even if `PIPELINING` is offered. This can avoid problems with some broken servers or firewalls.
   (e) `talk_to_myself`: do not check whether server greets with the hostname of the machine on which smtpc runs.
4. `wait_threads_max`: maximum number of waiting threads.
5. `wait_threads_min`: minimum number of waiting threads.
6. `module_timeout`: timeout for communication with QMGR.
7. `remote_port`: port to which connections should be made. Note: if multiple SMTP clients are specified, all of them *must* use the same value for `remote_port`. Currently the scheduler requires that all SMTP clients behave the same. If different ports are required, then those must be listed in mailertable entries.

See also Section 11.2 for more STARTTLS related configuration options.

[2] For example, `statethreads/examples/smtps2`

Chapter 9

Tuning

9.1 Size of Queues, Caches, and Databases

All data structures in QMGR have some maximum size. This is not just done to avoid resource exhaustion in high load situations but also to provide a feedback loop between SMTP servers (producers) and SMTP clients (consumers). This feedback loop helps to avoid flooding the system with mails that it cannot deliver fast enough. The incoming queue (IQDB) and the active queue (AQ) implement this feedback loop. As explained in Section 7.1 the data from the SMTP servers is stored in the incoming queue first which has a fixed size. If more data is produced than taken out (by the scheduler into the active queue) the queue will fill up and the QMGR will throttle the SMTP servers by dynamically reducing the number of available threads. Throttling the SMTP servers is done based on various resources, e.g., IQDB, AQ, available disk space, and much more. Hence by limiting the size of IQDB (see Section 3.8, item 8a) and of course the maximum number of threads in the SMTP servers the incoming flow of messages can be controlled. The size of IQDB should be greater than the maximum number of threads in the SMTP servers multiplied by the average number of recipients, otherwise transaction will be rejected before all threads are busy.

The active queue should be large enough to provide enough work for all SMTP clients (threads) and it must be larger than the largest number of recipients accepted by a single transaction (see Section 3.8, item 1).

9.2 Disk I/O

In most MTAs disk I/O is the limiting factor unless special hardware is used, e.g., disk controllers with battery backed RAM cache or SSD (Solid State Drives), to achieve high I/O rates (IOPS: I/O operations per second). If multiple disks are available, they can be used to spread the load. Disk files (see Section 2.4.1) are used for:

1. IBDB: the directory can be linked to a different disk.
2. DEFEDB: the base directory can be changed via an option (Section 3.8, item 4a), as well as the directory for logfiles (Section 3.8, item 4b).

3. CDB: the base directory can be changed via an option (see Section 3.4, item 2). Individual subdirectories (see Section 2.4.1) can be linked to different disks.

9.3 Processes and Threads

The main MeTA1 processes are multi-threaded. However, two different threading implementations are used: POSIX threads (pthreads) for QMGR and SMAR and statethreads [SGI01] for SMTP server and client. Statethreads only switch between threads on network I/O operations as it is a threading implementation in user space without kernel support. Hence operations that can take a long time, e.g., computations for asymmetric cryptography (as required during the STARTTLS handshake) or in some cases even synchronous disk I/O, will not just stop a single thread but the entire process. If this happens it is possible to start multiple SMTP servers, see Section 8.5, item 8. If it becomes necessary to start multiple SMTP clients, then the MCP can be instructed to do so, see Section 8.2, item 1.

9.4 Process Statistics

The four MeTA1 processes (QMGR, SMTPS, SMTPC, and SMAR) print some information about their internal state when they receive a `USR1` signal, e.g., how much entries are in some data structures, how many threads are currently used, and what were the maximum values reached during process execution. This information is not logged via `syslog(3)` but printed to the `PROG.log` files (see Section 4.8).

Chapter 10

Format Specifications

10.1 Regex Map

A regular expression (regex) map is initialized by reading lines from a file. Each line in the file must have the following format:

```
/regex/ value
```

That is, the line must start with a separator character which does not appear in the regex itself as it also marks the end. This can be followed by a list of flags (see `regex(3)`):

- `e` REG_EXTENDED: extended regex
- `i` REG_ICASE: ignore case

Next is a non-empty sequence of white space characters, and then the value which will be used if a match occurs (note: this means the value cannot start with a white space character). Lookups are done by sequentially applying all regexs in the listed order against a key until a match occurs in which case the value is returned.

Note: regex maps can have *interesting* interactions with the sequence of lookups performed as specified in Section 3.12.1.

Example:

```
/bounce-[0-9]+-[0-9]+@/ei error:550 5.7.1 no numbered bounces
;user@friendly.domain;i   OK
,sender\+[a-z]+@.*\.other.domain,e   OK
```

10.2 Socket Map

The socket map uses a simple request/reply protocol over TCP or UNIX domain sockets to query an external server. Both requests and replies are text based and encoded as netstrings, i.e., a string "hello there" becomes:

```
11:hello there,
```

Note: neither requests nor replies end with CRLF.

The request consists of the database map name and the lookup key separated by a space character:

mapname ' ' *key*

The server responds with a status indicator and the result (if any):

status ' ' *result*

The status indicator is one of the following upper case words:

`OK`	the key was found, result contains the looked up value
`NOTFOUND`	the key was not found, the result is empty
`NOMORE`	the key was not found, stop further search
`TEMP`	a temporary failure occured
`TIMEOUT`	a timeout occured on the server side
`PERM`	a permanent failure occured

In case of errors (status `TEMP`, `TIMEOUT` or `PERM`) the result field may contain an explanatory message.

The `NOMORE` status indicates that the key has not been found and also instructs smar to stop any further searches using this key and its derivatives.

For example, consider lookups in the alias map. Suppose that the current alias map is configured as:

```
aliases {
     name = aliasmap;
     flags = { localpart, local_domains };
}
```

where `aliasmap` is a socket map declared earlier in the configuration. This means that when looking for the key `foo@bar.net`, smar will perform the following lookups (see Section 3.12):

```
aliases foo@bar.net
aliases foo*@bar.net
aliases foo
```

Suppose that the alias `foo@bar.net` does not exist, but the local alias `foo` does exist. Now, when presented the lookup key `foo@bar.net`, the socket map cannot simply return `NOTFOUND`, because then smar will go on searching until the search for `foo` returns a false positive. In this case, the map should return `NOMORE`. When returned this reply, smar discontinues further searches and acts as if `NOTFOUND` was returned.

The `NOMORE` return works also in sequence map. If any of the maps in the sequence returns `NOMORE`, the remaining maps are not tried.

Example replies:

```
31:OK resolved.address@example.com,
```

in case of a successful lookup, or:

```
8:NOTFOUND,
```

in case the key was not found, or:

```
55:TEMP this text explains that we had a temporary failure,
```

in case of a failure.

10.3 Format of Session/Transaction/Recipient Identifiers

The format of session and transaction identifiers is specified in `include/sm/mta.h`. For the SMTP server it consists of a leading 'S', a 64 bit counter and an 8 bit "process" identifier, both of which are printed in hexadecimal format. For the SMTP client it consists of a leading 'C', an 8 bit "process" identifier, a 32 bit counter, and a 32 bit thread index, all of which are printed in hexadecimal format.

Examples: `S00000000407CE49200`, `C010000137D00000000`.

SMTP server session/transaction identifiers are unique until the 64 bit counter wraps around, SMTP client session/transaction identifiers are unique only within a single invocation of QMGR.

A recipient identifier (`rcpt_id`) consists of its transaction identifier, a hyphen, and its index in base64 encoding. For example, `S00000000407CE49200-000000`, denotes the first recipient of the transaction `S00000000407CE49200`.

Note: the format may change between different release of MeTA1, hence the identifiers should be considered opaque.

10.4 Logfile Format

The general format of entries in a logfile is a sequence of named fields which are separated by commas. Each field consists of a name, an equal sign, and a value. If the value is a text field that is received from an external (untrusted) source, then all non-printable characters, commas, and percent signs are shown as their two digit hexadecimal ASCII representation with a leading percent sign. For example, the text

```
550 5.7.1 no, not now, 99% usage
```

is encoded as

```
550 5.7.1 no%2C not now%2C 99%25 usage
```

This encoding allows a logfile analyzer to use the comma symbol as a delimiter of fields without having to perform complicated parsing, e.g, the Unix `awk` utility can be used with comma as field separator. Note: suggestions for a better encoding or different solution for the problem are welcome (more details can be found in [Aßmb]).

Logfiles use the identifiers described earlier (see Section 10.3) such that transactions and sessions can be easily recognized. For the following examples logfile entries have been slightly edited and line breaks have been inserted.

Here is one example of a session in an SMTP server:

```
ss_se=S00000000407EAE3800, client_ip=127.0.0.1,
  client_name=localhost.endmail.org.
ss_se=S00000000407EAE3800, starttls=TLSv1
ss_se=S00000000407EAE3800, ss_ta=S00000000407EAE4E00,
  mail=<SENDER@meta1.org>, stat=0
ss_se=S00000000407EAE3800, ss_ta=S00000000407EAE4E00,
  rcpt=<RECIPIENT@meta1.org>, idx=0, stat=0
ss_se=S00000000407EAE3800, ss_ta=S00000000407EAE4E00,
  rcpt=<SOMEONE@SOME.DOMAIN>, idx=1, stat=0
ss_se=S00000000407EAE3800, ss_ta=S00000000407EAE4E00,
  msgid=<20040916050457.GG54961@endmail.org>, size=1177, stat=0
```

The first entry shows a successful session creation including the IPv4 address and the hostname of the client. The second entry indicates that STARTTLS has been used. A new transaction is shown in the third entry and two recipients are given thereafter (along with the index `idx`). The last entry shows that the transaction was successful (`stat=0`; 0 is used instead of 250 or other SMTP reply codes that indicate success) and the size of the received mail (in bytes) as well as its Message-Id.

Here is one example of a session in an SMTP client:

```
da_se=C01000006C800000002, status=connected, port=25, addr=64.81.247.36
da_se=C01000006C800000002, starttls=TLSv1
da_se=C01000006C800000002, da_ta=C01000006C900000002,
  ss_ta=S00000000407EAE4E00, mail=<SENDER@meta1.org>, stat=0,
  reply=250 2.5.0 MAIL command succeeded
da_se=C01000006C800000002, da_ta=C01000006C900000002,
  ss_ta=S00000000407EAE4E00, rcpt=<RECIPIENT@meta1.org>, stat=0,
  reply=250 2.1.5 RCPT ok
da_se=C01000006C800000002, da_ta=C01000006C900000002,
  ss_ta=S00000000407EAE4E00, where=final_dot, size=1177, stat=0
```

This is very similar to the format of the entries entries in the SMTP server and should not require an explanation. In addition to the delivery agent session and transaction identifiers (`da_se` and `da_ta`) the SMTP server transaction identifier (`ss_ta`) is logged too. This makes it simple to track a message through the MTA. Obviously `ss_ta` can be used for multiple outgoing messages if the incoming message has been sent to multiple recipients (maybe indirectly via an alias), hence this is not a unique identifier in the SMTP client log.

QMGR can also log the delay time for each recipient, e.g.,

```
func=q_upd_rcpt_ok, rcpt_id=S00000000407EAE4E00-000000,
  rcpt=<RECIPIENT@meta1.org>, xdelay=0, delay=1
```

where `xdelay` is the time for this delivery attempt, and `delay` is the total delivery time (unit is seconds).

`iid`, `ip`, and `reply` are only logged if values are available.

10.5 Format of Received Header

The format of the `Received:` header added by the SMTP server is specified in `smtps/smtps.c`.

```
Received: from EHLO-NAME (CLIENT-NAME [CLIENT-ADDR])
        by HOST-NAME (MTA-VERSION) with PROTOCOL
        id SMTP-TA-ID; DATE
```

where `PROTOCOL` is one of `ESMTP`, `ESMTPS`, `ESMTPA`, `ESMTPSA`, or `SMTP` [New04]. If STARTTLS is active, then `(TLS=TLSVERSION, cipher=CIPHERSUITE, bits=CIPHERBITS, verify=VERIFYRESULT)` is placed before `id`, where TLSVERSION is the TLS protocol version, e.g., TLSv1.2, TLSv1; CIPHERSUITE is the cipher suite that was in use, e.g., AES256-SHA, EDH-DSS-DES-CBC3-SHA, EDH-RSA-DES-CBC-SHA, CIPHERBITS denotes the effective keylength (in bits) of the symmetric encryption algorithm of the TLS connection, and VERIFYRESULT is one of the following:

OK	verification succeeded.
NO	no cert presented.
NOT	no cert requested.
FAIL	cert presented but could not be verified, e.g., the signing CA cert is missing.

Note: the name of the client is only shown if the access map feature is activated (see Section 3.10, 4b), otherwise the time-consuming DNS lookups (PTR and A records) are not performed.

10.6 Format of DSNs

DSNs (*bounces*) are only compliant to RFC 3464 [MV03] if the configuration option `RFC_Format` (Section 3.8, item 5(b)iii) is used. Otherwise the format looks like this:

```
From: Mailer-Daemon@HOST.NAME
Subject: Undeliverable mail

A mail from you could not be delivered. See below for details.
```

and then a list of recipients and the reasons for the failure, e.g.,

```
Recipient:
<user@example.com>
Remote-MTA:
10.2.3.4
Reason:
550 5.7.1 <user@example.com>... Access denied
during RCPT
```

Chapter 11

Setup for STARTTLS

11.1 Certificates for STARTTLS

When acting as a server, MeTA1 requires X.509 certificates to support STARTTLS: one as certificate for the server, at least one root CA (`CAcert_file`), i.e., a certificate that is used to sign other certificates, and a path to a directory which contains certs of other CAs (`CAcert_directory`). The file specified via `CAlist_file` can contain several certificates of CAs. The DNs of these certificates are sent during the TLS handshake as the list of acceptable CAs to the

- client (as part of the CertificateRequest) (`SSL_CTX_set_client_CA_list(3)`),
- server if TLSv1.3 is used (`SSL_CTX_set0_CA_list(3)`).

Note: do not list too many root CAs in that file, otherwise the TLS handshake may fail; e.g.,

```
error:14094417:SSL routines:SSL3_READ_BYTES:
sslv3 alert illegal parameter:s3_pkt.c:964:SSL alert number 47
```

You should probably put only the CA cert into that file that signed your own cert(s), or at least only those you trust. The directory specified via `CAcert_directory` must contain the hashes of each CA certificate as filenames (or as links to them). Symbolic links can be generated using `c_rehash` (preferred; part of the OpenSSL distribution), or otherwise with the following two (Bourne) shell commands:

```
C=FileName_of_CA_Certificate
ln -s $C `openssl x509 -noout -hash < $C`.0
```

An X.509 certificate is also required for authentication in client mode, however, MeTA1 will always use STARTTLS when offered by a server. The client and server certificates can be identical. Certificates can be obtained from a certificate authority or created with the help of OpenSSL. The required format for certificates and private keys is PEM. To allow for automatic startup of MeTA1, private keys must be stored unencrypted. The keys are only protected by the permissions of the file system, hence they should not be readable by anyone but the owner. If server and client share the same key it is ok to make the key group readable however. Never make a private key available to a third party.

```
-r--r--r--  1 root     wheel       CAcert.pem
-r--r--r--  1 meta1s   meta1c      smcert.pem
-r--r-----  1 meta1s   meta1c      smkey.pem
drwxr-xr-x  2 root     wheel       certs/
```

11.2 Advanced Configuration for STARTTLS

The `tls` subsection in smtps and smtpc can have more options (some of the options are useful only for the server):

1. `DSA_cert_file`: file with DSA certificate in PEM format.
2. `DSA_key_file`: file with private key for DSA certificate in PEM format.
3. `EC_cert_file`: file with EC certificate in PEM format.
4. `EC_key_file`: file with private key for EC certificate in PEM format.
5. `EC_curve`: Name of elliptic curve to use, should be one of `prime256v1` and `secp384r1`.

Note: the elliptic curve options (`EC*`) require support by the OpenSSL implementation, usually available since version 1.0.0, but some OS have their own implementation which should be detected by the `configure` script.

11.3 TLS Session Cache

If a TLS session cache is configured, then information about TLS sessions are stored and retrieved from that cache. Doing so avoids computationally expensive parts of the TLS handshake, and hence can speed up creating sessions with hosts for which the corresponding TLS information is cached.

The cache size should be sufficient to hold information for the commonly used connections. Selecting a very large cache size may just waste memory. Entries in the cache are timed out based on the time they were added to the cache, not on their last use.

Chapter 12

More About Configuration, Compilation, Debugging, and Testing

12.1 Compile Time Options

There are several compile time options which might be useful in some situations that are listed below. Compile time options to turn on additional debugging are listed in section 12.1.5.

12.1.1 Generic

To further restrict the length of `syslog(3)` messages the compile time options `MTA_LOG_LEN` and `MTA_LOG_LEN_MAX` can be used (see `libmta/log.c` for the defaults). The macro `MTA_LOG_LEN_MAX` sets the maximum length of a `syslog(3)` message, `MTA_LOG_LEN` needs only be set if `MTA_LOG_LEN_MAX` is less than the default value of `MTA_LOG_LEN`.

IPv6 is available if the compile time option `MTA_NETINET6` is set. To enable IPv4 mapped addresses (provided the OS supports a hybrid dual-stack architectures) set the option to 2, e.g., `-DMTA_NETINET6=2`. For OSs that strictly separate IPv4 and IPv6, e.g., OpenBSD, see 3.10.2 for how to set up IPv6 and IPv4 servers.

12.1.2 QMGR

To enable QMGR statistics, e.g., number of transactions and recipients that have been handled, set `QMGR_STATS`.

12.1.3 SMAR

The address resolver imposes limits on the number of MX and A records that it accepts when it does routing lookups. These macros are:

- SM_DNS_MX_MAX: maximum number of MX records for a domain,
- SM_DNS_A_PER_MX_MAX: maximum number of A records for one MX record,
- SM_DNS_A_MAX: maximum number of A records for a domain (after performing MX lookups).

12.1.4 SMTPS

- SS_EHLO_ACCESS_CHK enables lookups of the EHLO/HELO argument in the access map (Section 3.10, 4b) using the tag ehlo:. This must be enabled at runtime via the flag check_ehlo.

12.1.5 Debugging Compile Time Options

There are several compile time parameters to support debugging. An option that applies to all modules (as they use the same libraries) is MTA_HEAP_CHECK which turns on various heap checks and keep track of memory usage.

Other options are specific to a module and can be used to turn on debugging output. Since currently no logging abstraction is in use, the output is done on a per-module basis (whatever is simplest for the individual module). These compile time options are:

SC_DEBUG	SMTPC debugging
SSQ_DEBUG	SMTPS - QMGR communication debugging
SS_DATA_DEBUG	SMTPS DATA stage debugging
QMGR_DEBUG	QMGR debugging
SMAR_DEBUG	SMAR debugging
MTA_LIBDNS_DEBUG	libdns debugging

For details see the source code.

Note: it is possible to set different debug levels for different debug categories in QMGR. For a list of categories see include/sm/qmgrdbg.h. To set a debug level n for a category c use the option -x$c.n$. The general syntax for the parameters is:

debugoptions	::=	debugoption ["," debugoptions]
debugoption	::=	range ["." level]
range	::=	first ["-" last]

If level is omitted, it defaults to 1. Example: -x1-3.4,5.3,9-11

A simple way to set compile time options is to use:

```
$ CFLAGS="-DSM_HEAP_CHECK" $PATHTO/meta1-$VERSION/configure
```

A more complicated example is:

```
$ CFLAGS="-O -g -DSM_HEAP_CHECK -I/usr/local/include" \
 LDFLAGS="-L/usr/local/lib" \
 $PATHTO/meta1-$VERSION/configure
```

Hint: it is useful to write the command line into a local file that can be reused for subsequent builds and versions.

Note: if `configure` has problems with OpenSSL because you do not have KerberosV installed, add

```
$ CPPFLAGS="-DOPENSSL_NO_KRB5"
```

12.2 Possible Compilation Problems or Warnings

1. If gcc is used as C compiler and full checking is turned on warnings like these are produced:

   ```
   warning: unknown conversion type character 'N' in format
   warning: unsigned int format, sm_str_P arg (arg 3)
   warning: too many arguments for format
   ```

   Unfortunately gcc cannot be told about additional format specifiers and hence the misleading warning might be generated.

2. On some systems the following warning is generated by the compiler:

   ```
   'sys_nerr' is deprecated; use 'strerror' or 'strerror_r' instead
   ```

   `sys_nerr` is used to determine whether it makes sense to invoke `strerror(3)` at all. The systems that generate this error do not provide an alternative way to perform this check. Just ignore the warning or ask the authors of that warning for an alternative way to determine the range of defined error codes.

12.3 More About Test Programs

12.3.1 More Environment Variables used by Test Programs

- `MTA_NAMESERVER`: can be used to set a specific nameserver (IPv4 address) in case the simple script which extracts the first line beginning with `nameserver` from the file `/etc/resolv.conf` does not give the desired result.
- `MTA_PMILTER_REGEX_TEST`: can be used to enable the tests (`make check` in `chkmts/`) for the policy milter `milter-regex` provided it is enabled and compiled. Notes:
  - this requires that `make check` is executed in `contrib/` before `make check` in `chkmts/`.
  - `make check` in `contrib/` may fail due to a compilation error for `milter-regex.c`. If your OS has `yacc(1)` (or `bison(1)` installed then remove that file (`milter-regex.c`) and try again.
- Some test programs use SMTP servers and sinks listening on an INET port. The default values for these ports are specified in `chkmts/common.sh`. If one of the default ports is used by another program, then the corresponding environment variable must be set as otherwise all related test programs will fail.
  - `MTA_SNKPORT`: set the port on which the SMTP sink is listening.
  - `MTA_SRVPORT`: set the port on which the SMTP test server is listening.
- `MTA_NO_IPV6_TEST`: skips IPv6 related tests even if the compile time option `MTA_NETINET6` is set.

12.3.2 Other Potential Problems with Test Programs

Some of the test programs may generate warnings, e.g., most of the tree related programs cause compilers on 32 bit systems to emit a warning `integer constant too large` which can be ignored.

Known Test Program Problems specific to an OS/setup

FreeBSD systems when running in a `jail(8)` exhibit the following problems:

- The test programs for SMAR which perform DNS lookups can fail because UDP does not work in a `jail(8)` as expected. A workaround for this is to use the `-U` option for `smar` which can be achieved by setting the environment variable `SMAROPTS` to that value.
- Connections from localhost to the SMTP server do not have `127.0.0.1` as source IP address, but the IP address of a NIC. Hence relaying must be allowed for it by setting the environment variable `MTA_SERVER_OPTIONS` to the option `-C` and the IP address, e.g., `-C 10.2.3.4`. Moreover, because the tests `chkmts/t-mts-icr.sh` and `chkmts/t-mts-ocr.sh` rely on connections coming from `127.0.0.1` they will fail too.

MacOS 10.3.4 has a problem with `sigwait(3)`, see Apple's bug 3675391; hence MeTA1 does not work on this OS (and other versions that have the same bug).

Chapter 13

Licenses

The main licenses for MeTA1 can be found in the file `LICENSE` and in the directory `license/`. Additionally, MeTA1 contains code from other projects whose licenses can be either found in the respective source files or in `statethreads/README` for the statethreads library and `db-4.3.28.NC/LICENSE` for Berkeley DB. Some source code is licensed under a BSD license which can be found at the begin of those files.

Chapter 14

Experimental Features

There are some experimental features in MeTA1 which are usually excluded from compilation using an `FFR_name` C preprocessor macro. This chapter lists those that might be ready for testing, use at your own risk.

14.1 Cert Pinning

MeTA1 implements an experimental form of *cert pinning* when compiled with `FFR_CERT_PINNING`. This applies to the SMTP client: when it established a TLS session with a server, it stores the fingerprint of the presented cert (if available) as well as its expiration time and passes that information to QMGR. QMGR in turn stores this information in the outgoing connection cache (OCC). When a new session to that server (IP address) is required, QMGR passes the cert fingerprint (if the cert hasn't expired yet and the information has not been removed from OCC due to its size limit) to SMTPC which will check it against the fingerprint of the cert that the server presents this time. If they do not match, SMTPC will log a warning `status=cert_fp does not match previous value`. Currently, there are configuration options to change this behaviour. However, if `tls_requirements` for that server have been violated, then cert pinning is not performed.

This compile time option also causes MeTA1 to store TLS errors in OCC. smtpc will first try different TLS options if a previous connection to the same server (based on IP address) failed during the TLS handshake. If all of those variations fail, smtpc will finally not use STARTTLS. Errors are cleared after the OCC timeout is reached (see section 3.8.1, item 4).

14.2 DANE

Partial support for DANE has been implemented in form of a hack. To test this functionality the following compile time options are required: `-DMTA_USE_DNSSEC` for DNSSEC, `-DMTA_TLS_DANE` to enable support of DANE-EE TLSA entries. It is strongly recommended to run a local DNS resolver that supports DNSSEC and configure MeTA1 to use that (including setting the flag `Use_DNSSEC`). smar has a new option `DANE` with the possible values:

1. `secure`: only check TLSA records if they can be retrieved using DNSSEC (default).

2. `never`: turn off DANE.

3. `always`: check TLSA records even without DNSSEC (this is mostly for testing).

There is currently only support for TLSA entries of type 3-1-x, unless OpenSSL 1.1 with DANE support is used.

- Certificate usage: 3: specify a certificate, or the public key of such a certificate, that MUST match the end entity certificate given by the server in TLS.
- Selector 1: SubjectPublicKeyInfo: DER-encoded binary structure.
- Matching Types 0, 1, and 2: exact match, or SHA-256 or SHA-512 hash of selected content.

The hack is implemented by using a `dane` section with `pubkey_fps`, `base_domains`, and some other information (requires `lookup_rcpt_conf` and `lookup_session_conf`, see Section 3.11.1), which is similar to `cert_fps`, but added by smar.

Currently the port for TLSA records is hardcoded to 25 (SMTP).

If DANE verification of the server was successful and DNSSEC was used, then the value for `verify` in the log is set to `DANE_SEC`.

Appendix A

Policy Milter

A.1 Native Policy Milter API

Note: this API may evolve over time.

Naming conventions: A *policy milter* (also called *pmilter*) is a program that uses the API provided by *libpmilter*. The latter interacts with the SMTP servers via an internal protocol, i.e., this protocol can be changed without changing the visible API and should not directly be accessed by a user application.

A.1.1 Data Structures

libpmilter itself uses three context structures all of which must be treated by a milter as opaque.

1. `pmg_ctx`: "global" libpmilter context (only one per process).
2. `pmss_ctx`: libpmilter context per SMTP server that connects to this instance. There can be multiple SMTP servers connecting to one libpmilter instance.
3. `pmse_ctx`: libpmilter context per SMTP session.

Any of the libpmilter functions takes one of these contexts as parameter; e.g., all SMTP session oriented functions have a parameter of type `pmse_ctx_P`.

A milter can have its own contexts for each of these three environments, see Section A.1.5.

A.1.2 Start and Stop

The functions in this section return `SM_SUCCESS` (0) on success and a negative value in case of an error.

First libpmilter must be initialized; a pmilter must specify a variable `pmg_ctx_P pmg_ctx;` which is passed per reference to the initialization function:

```
sm_ret_T sm_pmfi_init(pmg_ctx_P *pmg_ctx)
```

The pmilter global context must be treated as opaque data structure, it is passed to subsequent libpmilter function calls.

Next pmilter starts libpmilter by handing control over to the library; the pmilter passes a description of its requirements and functionality:

```
sm_ret_T sm_pmfi_start(pmg_ctx_P pmg_ctx, pmilter_P pmilter)
```

A milter can stop by calling:

```
sm_ret_T sm_pmfi_stop(pmg_ctx_P pmg_ctx)
```

There are various functions to set some options which can be called after libpmilter is initialized but before it is started. To set the path of the Unix domain socket over which the SMTP servers (see Section 3.10, item 19) and libpmilter communicate:

```
sm_ret_T sm_pmfi_setconn(pmg_ctx_P pmg_ctx, const char *path)
```

The backlog parameter of the `listen(2)` function can be set:

```
sm_ret_T sm_pmfi_setbacklog(pmg_ctx_P pmg_ctx, int backlog)
```

The debug level of libpmilter might be set via (this requires knowledge of the internals of the library which can be acquired by looking at the source code):

```
sm_ret_T sm_pmfi_setdbg(pmg_ctx_P pmg_ctx, int debuglevel)
```

To set the communication timeout:

```
sm_ret_T sm_pmfi_settimeout(pmg_ctx_P pmg_ctx, int timeout)
```

A.1.3 New SMTP Server

Whenever an SMTP server connects to a milter an option negotiation is performed (similar to ESMTP itself). A pmilter can check whether server capabilities are acceptable and return the options that it wants:

```
sm_ret_T pmfi_negotiate(pmss_ctx_P pmss_ctx, uint32_t srv_cap, uint32_t srv_fct, uint32_t srv_feat,
uint32_t srv_misc, uint32_t *pm_cap, uint32_t *pm_fct, uint32_t *pm_feat, uint32_t *pm_misc)
```

Currently only the capabilities field is used: `srv_cap` is set by the SMTP server to a list (implemented as bit field) of phases of the ESMTP dialogue that can be passed to a pmilter. In turn the pmilter must set `*pm_cap` to includes those phases of the ESMTP dialogue that it wants to receive. For details, see `include/sm/pmilter.h`. For each of those phases a callback is invoked (see Section A.1.4) which must be set by the pmilter in its description structure `struct pmilter_S` (see `include/sm/pmfapi.h`).

A.1.4 SMTP Session and Transaction

The protocol steps from ESMTP are forwarded to the policy milter which can decide to accept or reject them.

- New SMTP session:

  ```
  sfsistat_T pmfi_connect(pmse_ctx_P pmse_ctx, const char *hostname, sm_sock_addr_T *hostaddr)
  ```

  `hostname`: host name, as determined by a reverse lookup on the host IP address; `hostaddr`: host address, as determined by a `getpeername(2)` call on the SMTP socket.

- SMTP `HELO`/`EHLO` command:

`sfsistat_T pmfi_helo(pmse_ctx_P pmse_ctx, const char *helohost, bool ehlo)`

`helohost`: Value passed to `HELO`/`EHLO` command, which should be the domain name of the sending host. `ehlo`: `true` iff `EHLO` was used.

- `MAIL` (envelope sender):

  `sfsistat_T pmfi_mail(pmse_ctx_P pmse_ctx, const char *mail, char **argv)`

  `mail`: envelope mail address; `argv`: null-terminated `MAIL` command arguments.

- `RCPT` (envelope recipient):

  `sfsistat_T pmfi_rcpt(pmse_ctx_P pmse_ctx, const char *rcpt, char **argv)`

  `rcpt`: envelope recipient address; `argv`: null-terminated `RCPT` command arguments.

- `DATA`:

  `sfsistat_T pmfi_data(pmse_ctx_P pmse_ctx)`

- unknown/not implemented SMTP command:

  `sfsistat_T pmfi_unknown(pmse_ctx_P pmse_ctx, const char *cmd)`

  `cmd`: SMTP command. Note: this is not yet implemented.

- For each chunk of a message:

  `sfsistat_T pmfi_msg(pmse_ctx_P pmse_ctx, unsigned char *msgp, size_t msglen)`

  `msgp`: pointer to message data; `msglen`: length of message data. There may be multiple message chunks passed to the filter. End-of-lines are represented as received from SMTP (normally Carriage-Return/Line-Feed; CRLF). Notes:

  - the last message chunk contains the final dot of the SMTP transmission, i.e., "CRLF.CRLF"
  - the message is not modified in any form, i.e., dots at the begin of a line are duplicated (by the SMTP client) as specified in section 4.5.2 of RFC 2821 [Kle01] which must be undone by the application if so desired.
  - the message is *streamed* while being received. That is, the mail is not first stored on disk and then sent to the filter, but each part received from the client is sent directly to the filter (at the same speed as received from the network which might be slow). This may mean that the filter does not receive the entire message as the transmission may get interrupted or the SMTP server may decide to skip the rest of the message because it exceeds the maximum size.
  - the return code is ignored unless `SM_SCAP_PM_MSG_RC` is set, see Section A.1.11, item 2.

- End of message (final dot of message has been received):

  `sfsistat_T pmfi_eom(pmse_ctx_P pmse_ctx)`

- Message is aborted outside of the control of the filter, for example, if the SMTP client issues an RSET command.

  `sm_ret_T pmfi_abort(pmse_ctx_P pmse_ctx)`

- `QUIT` (end of an SMTP session):

  `sm_ret_T pmfi_close(pmse_ctx_P pmse_ctx)`

  This is called when an SMTP session ends.

A.1.5 Set and Get pmilter Contexts

As explained in Section A.1.1 a milter can have a "global" context `pmilter_g_ctx`, a context per SMTP server `pmilter_ss_ctx`, and a context per SMTP session `pmilter_se_ctx`. The following functions are provided to set and get these contexts.

Set the "global" context `pmilter_g_ctx`:

```
sm_ret_T sm_pmfi_set_ctx_g(pmg_ctx_P pmg_ctx, void *pmilter_g_ctx).
```

This must be done after libpmilter has been initialized but before control is transferred to it.

To retrieve the "global" context invoke:

```
void *sm_pmfi_get_ctx_g(pmg_ctx_P pmg_ctx)
```

Note: this requires the "global" libpmilter context which is not usually passed to pmilter functions in callbacks. See below how to access the "global" context `pmilter_g_ctx` from other places.

To set the pmilter context per SMTP server `pmilter_ss_ctx` use:

```
sm_ret_T sm_pmfi_set_ctx_ss(pmss_ctx_P pmss_ctx, void *pmilter_ss_ctx);
```

to retrieve it call:

```
void *sm_pmfi_get_ctx_ss(pmss_ctx_P pmss_ctx)
```

The "global" pmilter context `pmilter_g_ctx` can be retrieved from the libpmilter context per SMTP server:

```
void *sm_pmfi_get_ctx_g_ss(pmss_ctx_P pmss_ctx)
```

At the lowest level a context per SMTP session `pmilter_se_ctx` can be set via:

```
sm_ret_T sm_pmfi_set_ctx_se(pmse_ctx_P pmse_ctx, void *pmilter_se_ctx)
```

and retrieved by:

```
void *sm_pmfi_get_ctx_se(pmse_ctx_P pmse_ctx).
```

Just as before there is a function to retrieve the pmilter context per SMTP server `pmilter_ss_ctx` from the libpmilter context per SMTP session:

```
void *sm_pmfi_get_ctx_ss_se(pmse_ctx_P pmse_ctx)
```

Note: if a pmilter uses these contexts, then it is useful that each "lower level" context contains a link to its "higher level" context. That is, each pmilter context per SMTP session `pmilter_se_ctx` should have a pointer to its pmilter context per SMTP server `pmilter_ss_ctx` which in turn should have a pointer to the "global" pmilter context `pmilter_g_ctx`. This allows access from a function that is specific to a SMTP session to each relevant context.

A.1.6 Accessing MTA Symbols

A pmilter can set a list of symbols it wants to receive from the MTA by calling one of following functions:

```
sm_pmfi_setmaclist(pmss_ctx_P pmss_ctx, uint where, ...)
```

```
sm_pmfi_setmacs(pmss_ctx_P pmss_ctx, uint where, uint32_t macros[])
```

during the option negotiation, i.e., in `pmfi_negotiate()`. The parameter `where` denotes the stage of the

ESMTP dialogue when the value of the symbol should be sent. It must be one of

PM_SMST_CONNECT	Session start
PM_SMST_EHLO	EHLO or HELO command
PM_SMST_MAIL	MAIL command
PM_SMST_RCPT	RCPT command
PM_SMST_DATA	DATA command
PM_SMST_DOT	Final dot of mail body

Note: these functions can only be called once for each value of `where`.

A sequence of up to PM_MACROS_MAX macros can be requested which must end with PMM_END. Valid values are:

1. PMM_SRVHOSTNAME hostname of SMTP server.
2. PMM_SEID session id.
3. PMM_CLIENT_RESOLVE result of client lookups.
4. PMM_MAIL_TAID transaction id.
5. PMM_DOT_MSGID Message-Id.
6. PMM_TLS_VERSION TLS/SSL version used.
7. PMM_TLS_CIPHER_SUITE cipher suite used.
8. PMM_TLS_CIPHER_BITS effective key length of the symmetric encryption algorithm.
9. PMM_TLS_ALG_BITS maximum key length of the symmetric encryption algorithm. This may be less than the effective key length for *export controlled* algorithms.
10. PMM_TLS_VRFY the result of the verification of the presented cert.
11. PMM_TLS_CERT_SUBJECT the DN (distinguished name) of the presented certificate.
12. PMM_TLS_CERT_ISSUER the DN (distinguished name) of the CA (certifcate authority) that signed the presented certificate (the cert issuer).
13. PMM_TLS_CN_SUBJECT the CN (common name) of the presented certificate.
14. PMM_TLS_CN_ISSUER the CN (common name) of the CA that signed the presented certificate.
15. PMM_AUTH_TYPE the AUTH mechanism used.
16. PMM_AUTH_AUTHEN the client's authentication credentials as determined by authentication.
17. PMM_AUTH_AUTHOR The authorization identity, i.e. the AUTH= parameter of the MAIL command if supplied.

Notes:

- PMM_MAIL_TAID cannot be requested before PM_SMST_MAIL and PMM_DOT_MSGID can only be requested at stage PM_SMST_DOT.
- All macros beginning with PMM_TLS are only valid after a STARTTLS command.

To retrieve the value of a symbol the function

```
sm_pmfi_getmac(pmse_ctx_P pmse_ctx, uint32_t macro, char **pvalue)
```

can be used in the various callback functions of the ESMTP dialogue. If the macro was not in the request list, an error will be returned. If the macro has not yet been received, `*pvalue` will be `NULL`. Otherwise `*pvalue` will point to the value of the macro. Note: the string to which `*pvalue` points must *not* be changed.

A.1.7 Sender Modification

The sender address (`MAIL`) can be replaced:

```
sm_ret_T sm_pmfi_mail_rplc(pmse_ctx_P pmse_ctx, const char *mail_pa, char **argv)
```

This function must only be called during `pmfi_eom()`. The address `mail_pa` must be in RFC 2821 format. The argument `argv` can be used to specify SMTP parameters for the sender address, however, this is currently not implemented, hence it must be set to `NULL` for now.

A.1.8 Recipient Modifications

Recipients can be added:

```
sm_ret_T sm_pmfi_rcpt_add(pmse_ctx_P pmse_ctx, const char *rcpt_pa, char **argv)
```

or deleted:

```
sm_ret_T sm_pmfi_rcpt_del(pmse_ctx_P pmse_ctx, const char *rcpt_pa, rcpt_idx_T rcpt_idx)
```

These functions must only be called during `pmfi_eom()`. The addresses `rcpt_pa` must be in RFC 2821 format. The argument `argv` can be used to specify SMTP parameters for the recipient address, however, this is currently not implemented, hence it must be set to `NULL` for now. As the MTA does not remove identical recipient addresses, the address itself is not sufficient to identify one `RCPT`, but its index must be specified too. This index can be retrieved during a `RCPT` command (`pmfi_rcpt()`) using

```
sm_ret_T sm_pmfi_getrcpt_idx(pmse_ctx_P pmse_ctx, rcpt_idx_T *prcpt_idx)
```

Note: it is invalid to remove all recipients of a transaction. To discard a transaction, return `SMTP_R_DISCARD` as a result of one of the transaction oriented callbacks, e.g., `pmfi_eom()`.

A.1.9 Header Modifications

To request modifications of the header of a mail being sent, the function

```
sm_pmfi_hdr_mod(pmse_ctx_P pmse_ctx, uint type, uint pos, const unsigned char *header)
```

can be used. This function must only be called during `pmfi_eom()`. The argument `type` specifies which kind of modification is requested, legitimate values are defined in `include/sm/hdrmoddef.h`; these are: `SM_HDRMOD_T_PREPEND`, `SM_HDRMOD_T_INSERT`, `SM_HDRMOD_T_REPLACE`, `SM_HDRMOD_T_REMOVE`, and `SM_HDRMOD_T_APPEND`.

The argument `header` must be a complete header line including the proper line ending (CRLF). The argument `pos` specifies the position for the types `SM_HDRMOD_T_INSERT`, `SM_HDRMOD_T_REPLACE`, and `SM_HDRMOD_T_REMOVE`.

The first header of the original mail has position one; zero is the `Received:` header added by the SMTP server.

A.1.10 Message Replacement

If a pmilter wants to replace the entire message, the function `pmfi_eom()` must return the value `SMTP_R_RPLCMSG`. This will cause the invocation of the callback

```
sfsistat_T pmfi_msg_rplc(pmse_ctx_P pmse_ctx, const unsigned char **pmsgchunk, size_t *pmsglen)
```

which then must set a pointer to a message chunk and its length. Multiple chunks can be sent by returning `SMTP_R_CONT` as result of the callback. For the last chunk, `SMTP_R_OK` should be returned. The size of each chunk (`pmsglen`) must be less than `PMILTER_CHUNK_SIZE` as defined in `include/sm/pmfapi.h`.

libpmilter will thereafter invoke the callback

```
sfsistat_T pmfi_msg_rplc_stat(pmse_ctx_P pmse_ctx, sm_ret_T status)
```

to give the pmilter a chance to clean up after the transaction, and to let it know whether the message replacement was successful.

Notes:

- as the entire message is replaced and by default only the data that is received from the SMTP client is sent to a pmilter, the `Received:` header field that smtps generates is lost. To avoid this, a pmilter can request to receive that header field by setting `SM_SCAP_PM_SND_RCVD` and sending it as first chunk of the message replacement.
- the message must be in SMTP format, i.e., lines must end in CRLF and the final chunk should have the usual SMTP end of message indication: CRLF dot CRLF (`\r\n.\r\n`), however, it can also just end in CRLF.
- if `pmfi_msg_rplc()` encounters an error, e.g., due to an API violation or due to a communication error with smtps, then it will invoke `pmfi_msg_rplc_stat()` directly without waiting for the entire message even if it consists of more chunks.

A.1.11 Further Capabilities

In addition to selecting which SMTP commands to send to pmilter (see Section A.1.3), there are some more capabilities available:

1. `SM_SCAP_PM_RCPT_ST` causes the MTA to send `RCPT` information even if the command has been rejected, e.g., because the recipient is unknown, the recipient has been rejected due to access map checks, or relaying has been denied. Note: `RCPT` commands that are rejected for other reasons, e.g., because the address is syntactically invalid, or some limit (maximum number of recipients) is exceeded, will not be sent to pmilter.

   The function

   ```
   sm_ret_T sm_pmfi_getstatus(pmse_ctx_P pmse_ctx, sfsistat_T *pstatus)
   ```

   should be used in that case to access the current SMTP reply code for the command. This functionality is useful for a pmilter that wants to keep track of all recipients, not just those which are accepted, e.g., to deal with dictionary attacks.

2. SM_SCAP_PM_MSG_RC allows a pmilter to return a reply code as specified in A.1.13 from pmfi_msg(). This is useful if a pmilter can make a decision about the mail without having to read the entire message. If this capability is turned on, pmfi_msg() must return SMTP_R_CONT for each message chunk by default to receive subsequent parts. Otherwise pmfi_eom() will not be called but the return code from pmfi_msg() will be used at the end of the message (in response to the final dot).

A.1.12 Miscellaneous Functions

To set a reply text in an SMTP session or transaction oriented callback in addition to the reply code use:

```
sm_ret_T sm_pmfi_setreply(pmse_ctx_P pmse_ctx, const char *reply)
```

Note: the reply string *must* contain the full SMTP reply, i.e., it must be of the form

```
XYZ D.S.N text\r\n
```

where XYZ is a valid SMTP reply code (see RFC 2821 [Kle01]) which *must* match the return code of the function from which sm_pmfi_setreply() is called, D.S.N is an enhanced status code as defined in RFC 3463 [Vau03] and the rest is an explanation of the status including CRLF (\r\n). The text can be a multi-line reply in the form:

```
XYZ-D.S.N text1
XYZ-D.S.N text2
XYZ D.S.N text3
```

which must be specified in the format required by SMTP:

```
XYZ-D.S.N text1\r\nXYZ-D.S.N text2\r\nXYZ D.S.N text3\r\n
```

To set reply codes for commands that need multiple reply values the function:

```
sm_ret_T sm_pmfi_setreplies(pmse_ctx_P pmse_ctx, uint nreplies, int *rcodes, const char **rtexts)
```

must be used. Currently this function makes only sense if PRDR is available in the SMTP server and actually used by the client. A pmilter can determine the latter by parsing the arguments of the MAIL command (see pmfi_mail()). Note: currently the argument rtexts is ignored, only the array of reply codes (rcodes) is used. The size of this array must be nreplies which must match the number of valid RCPTs for this transaction. The reply codes in that array must be in the same order in which the RCPTs have been received.

Return version number of libpmilter:

```
sm_ret_T sm_pmfi_version(pmg_ctx_P pmg_ctx, uint32_t *major, uint32_t *minor, uint32_t *patchlevel)
```

This can be used to compare the version number of the library against which pmilter is linked with the version number against which pmilter is compiled. The major version numbers must match otherwise the program will not run.

Signal handler callback:

```
sm_ret_T pmfi_signal(pmg_ctx_P pmg_ctx, int sig)
```

This function will be called when a USR1 or USR2 signal is received; it is not called within a signal handler, i.e., the code does not have to be signal-safe. Note: this is not yet implemented.

A.1.13 Return Values

SMTP Session and transaction oriented functions use `sfsistat_T` as return type. Allowed values for this type are (as defined in `include/sm/smreplycodes.h`):

- `SMTP_R_OK`: accept command.
- `SMTP_R_ACCEPT`: accept entire transaction or session depending on where this value is returned. Note: this is just a shortcut for `SMTP_R_SET_QUICK(SMTP_R_OK)`.
- `SMTP_R_DISCARD`: discard effect of command.
- `SMTP_R_CONT`: continue other checks.
- `SMTP_R_SSD`: shut down SMTP session.
- `SMTP_R_TEMP`: reject command with a temporary error.
- `SMTP_R_SYNTAX`: syntax error.
- `SMTP_R_PERM`: reject command with a permanent error.
- other valid SMTP reply codes [Kle01].

Additionally return values can be modified by using `SMTP_R_SET_QUICK(returnvalue)`. See Section 3.9.3 for the effects of this.

For functions that use `sm_ret_T` as return type a successful call returns `SM_SUCCESS` (0) and a negative value in case of an error.

A.1.14 Implementation Notes

As `libpmilter` currently does not keep track of the status of a transaction or session, the functions `pmfi_abort()` and `pmfi_close()` may be called even if no transaction or session is currently active. This can happen if an SMTP server unexpectedly aborts the connection to a policy milter. An application must be aware of this and keep track of its state properly.

A.2 Policy Milter Examples

The program `libpmilter/example-pmilter-0.c` is a simple example how to write a policy milter. It might be useful as a template for other milters. For some operating systems it might be necessary to change the list of system include files (see also Section A.2.1).

Also available is a policy milter `contrib/milter-spamd.c` that offers an interface to `spamd(1)` which is a daemonized version of `spamassassin(1)`. `milter-spamd.c` is written by Daniel Hartmeier [Harc] (see the file itself for the Copyright) for sendmail 8 and modified to work with the policy milter API of MeTA1.

A.2.1 Compiling Policy Milters

The program `libpmilter/example-pmilter-1.c` shows which `.h` files need to be included from the MeTA1 distribution: those are referenced as "`sm/`*name*`.h`". As a minimum, a pmilter also needs definitions for `bool` (usually available via `stdbool.h`) and `int16_t`, `int32_t`, `uint16_t`, and `uint32_t` (usually available via `stdint.h` or `inttypes.h`). If those type definitions are not available, the file "`sm/generic.h`" contains default definitions that are suitable for most systems. Those can be activated via the compile time options `META1_NEED_INTN` and `META1_NEED_BOOL`, respectively. The file `libpmilter/makefile.pmilter` is an example `makefile` (for `make(1)`) that works on systems like SunOS 5.10. It can be easily adapted to other operating systems; it shows the list of libraries that are needed from the MeTA1 distribution.

Bibliography

[ACD+07] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. DomainKeys Identified Mail (DKIM) Signatures. RFC 4871, Internet Engineering Task Force, 2007.

[Aßma] Claus Aßmann. Sendmail X. `http://www.sendmail.org/%7Eca/email/sm-9-rfh.html`.

[Aßmb] Claus Aßmann. Sendmail X: Requirements, Architecture, Functional Specification, Implementation, and Performance. `http://www.sendmail.org/%7Eca/email/sm-X/`.

[Ber97] Dan Bernstein. VERP: Variable Envelope Return Paths, 1997. `http://cr.yp.to/proto/verp.txt`.

[Cyr] Project Cyrus. `http://asg.web.cmu.edu/cyrus/`, `http://asg.web.cmu.edu/sasl/`.

[DKI] DKIM. `http://www.dkim.org/`.

[Fre00] N. Freed. SMTP service extension for command pipelining. RFC 2920, Internet Engineering Task Force, 2000.

[Gnu] GnuPG. GNU Privacy Guard. `http://www.gnupg.org/`.

[Hal07] Eric A. Hall. Smtp service extension for per-recipient data responses (prdr). Draft, Internet Engineering Task Force, 2007.

[Hara] Evan Harris. The next step in the spam control war: Greylisting. `http://greylisting.org/articles/whitepaper.shtml`.

[Harb] Evan Harris. Whitelisting. `http://greylisting.org/whitelisting.shtml`.

[Harc] Daniel Hartmeier. benzedrine.cx - milter-spamd. `http://www.benzedrine.cx/milter-spamd.html`.

[Hof99] P. Hoffman. SMTP service extension for secure SMTP over TLS. RFC 2487, Internet Engineering Task Force, 1999.

[Kle01] Simple mail transfer protocol. RFC 2821, Internet Engineering Task Force, 2001.

[MeT] PGP keys. `http://www.MeTA1.org/security/PGPKEYS`.

[Moc87] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035, Internet Engineering Task Force, 1987.

[mut] mutt. `http://www.mutt.org/`.

[MV03] K. Moore and G. Vaudreuil. An Extensible Message Format for Delivery Status Notifications. RFC 3464, Internet Engineering Task Force, 2003.

[Mye96] John Myers. Local mail transfer protocol. RFC 2033, Internet Engineering Task Force, 1996.

[Mye99] J. Myers. SMTP service extension for authentication. RFC 2554, Internet Engineering Task Force, 1999.

[New04] Chris Newman. ESMTP and LMTP Transmission Types Registration. RFC 3848, Internet Engineering Task Force, 2004.

[Ope] OpenSSL. `http://www.openssl.org/`.

[PGP] PGP. `http://www.pgp.com/`.

[Posa] Jef Poskanzer. graymilter - simple graylist mail filter module. `http://www.acme.com/software/graymilter/`.

[Posb] Jef Poskanzer. mini_sendmail - accept email on behalf of real sendmail. `http://www.acme.com/software/mini_sendmail/`.

[SGI01] SGI. State threads for internet applications, 2001. `http://state-threads.sourceforge.net/`.

[Slea] Berkeley DB 4.4.XX Change Log. `http://www.sleepycat.com/update/4.4.XX/if.4.4.XX.html`.

[Sleb] Berkeley DB Tutorial and Reference Guide, version 4.3.28. `http://www.sleepycat.com/docs/`.

[Tok] Michael Tokarev. tinycdb: A package for maintenance of constant databases. `ftp://ftp.corpit.ru/pub/tinycdb/`.

[Vau03] G. Vaudreuil. Enhanced mail system status codes. RFC 3463, Internet Engineering Task Force, 2003.

[vdBG] S.R. van den Berg and Philip Guenther. procmail. `http://www.procmail.org/`.