Meta — a freely available scalable MTA

Assar Westerlund
Swedish Institute of Computer Science
assar@sics.se
Love Hörnquist-Åstrand
Department of Signals, Sensors, and Systems, KTH
lha@s3.kth.se
Johan Danielsson
Center for Parallel Computers, KTH
joda@pdc.kth.se

Abstract

This paper describes the design and implementation of the mail server Meta. It is intended to be a simple and secure yet efficient and scalable mail server. It only handles receiving mails by SMTP and providing a POP server for the user but tries to be fast at doing that.

1 Introduction

One of the oldest and still most popular applications on the Internet is electronic mail (e-mail). More and more people find e-mail to be a practical form of communication with colleagues, companies, authorities, relatives, and friends. For many people, e-mail has become the preferred means of communication. And there are lots of mailing lists where people interested in a particular topic can discuss it among themselves. Also, people are depending more on e-mail actually working and getting to the recipient in a short amount of time.

Lots of effort has been put into optimising other common Internet applications, like web servers and proxies, news servers, and others. But few have looked at how to handle large volumes of mail for large number of users efficiently. We think this popular application deserve some more attention.

2 What is the Problem?

The problem that is addressed (and/or solved) by Meta is building a high-capacity, scalable, and secure mail-hub. A mail-hub in this context is a server that receives mail destined for users with the Simple Message Transfer Protocol (SMTP)[11] from the Internet and that allow them to retrieve it with the Post Office Protocol (POP)[12] to their local mail clients. In other words, Meta is an embedded SMTP-server and POP-server. This is shown in figure 1. Meta does not try to be a general-purpose MTA. Instead, it tries to fulfil the particular need that we had and doing it efficiently and while remaining simple and secure. While performing a subset of what a general-purpose MTA does, we think that there are a number of sites other than us that require this functionality, such as companies or universities with large number of mail users and above all, large ISPs.

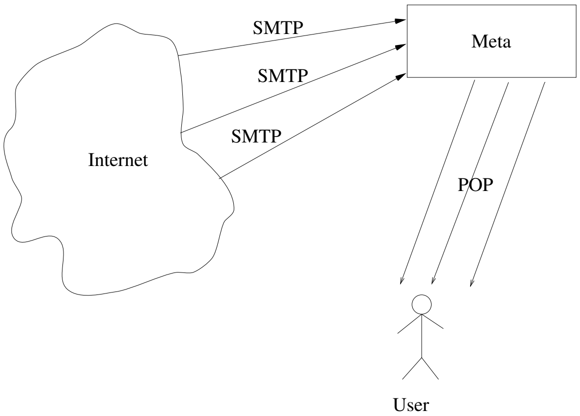

Figure 1: Meta overview

The traditional way of building a mail hub is to run an MTA such as sendmail[10] that receive mails. They are then written in spool files by a local delivery program such as mail.local. There exists a single flat file per user where the incoming mails are stored. Each new mail is appended to the end of this file. The mails are read from these files and eventually deleted by the pop daemon when the users fetch them to their mail clients with POP. But some users and mail clients instead access the spool files directly on the mail hub or through NFS [13]. Or they use IMAP [14] or some other (and new) means of getting the mail. This means that the mails have to be stored in the common mbox-format so that all the programs that can access them will understand the format.

With Meta we decided early that the only way of accessing the mail would be with POP. This simplifies lots of things and does not require storing mail in the mbox format. (The benefits from this are quite similar to those described in [7] regarding NNTP [15] and news spool files.)

3 Goals

- little (or no) configuration

  We would like Meta do be as self-functioning as possible. This means not having to maintain complicated configurations and/or do lots of tuning on the running Meta. The configuration is now kept quite minimal.

- focus on solving a small and well-defined problem

  This is a traditional UNIX approach. Meta itself doesn't relay mail, handle outgoing mail, or manage mailing lists. These functions, which are of course needed, will be implemented in separate programs. Separating different functionality into different programs tend to make them simpler and more efficient.

- simple

  The MTA should be simple to make it easy to review for correctness, and to simplify a security-audit. Lesser code makes it easier to write correct code. It should also be simple to extend and give new functionality.

- secure

  An MTA is exposed to a large number of more or less broken mail applications, as well as malicious people trying to do evil things, but people trust mail to function and depend on it, so it is rather important that it is reliable and secure.

- efficient

  It should be possible to handle a large number of users receiving very large number of mails on rather low-end hardware.

- scalable

  Scalability is an important point, it should not be necessary to buy a faster computer just to be able to handle more users, instead you should be able to cluster mail servers to handle the additional load.

4 Non-Goals

- being a complete MTA

  Which means not handling UUCP, address rewriting, etc, etc.

- being configurable in every detail

  There is not going to be a sendmail.cf to configure Meta. Its sole purpose is to be able to receive mail mail really fast.

- being compatible with old MTAs

  The important point is using the protocols correctly, the mail hub itself should be a rather dark box. Not having to be compatible with old ways of doing things (like /var/spool/mail/*user* and .forward) means being able to try the solve the problems in new (and hopefully better) ways.

- sending and relaying mail

  Separate programs will be written to handle these tasks. They do not even have to run on the same servers.

- handling mailing lists

  There are already lots of programs that do this so we have not found any need for writing a new one.

5 What is the Limit?

We wanted to get a feeling for how good performance it would be possible to get out of a mail server running on a common piece of hardware, say an ordinary PC. For this purpose, we designed a simple benchmark for measuring the time it would take to receive a single mail over an SMTP connection, and also wrote a simple prototype of a mail server. This mail server just receives mail and appends it to spool files, one for each user. The file is also locked at the beginning of reception and unlocked at the end. After having written the complete mail, the file is fsync'ed and close'd. The existence of a user is determined from the existence of a spool file. It uses a state-machine written around select.

The benchmark program sent a variable number of mails of around 500 bytes to one of 10 different users in a round robin fashion over the same SMTP connection. We measured the total elapsed time for this operation.

To get a rough comparison with other MTAs, we also performed the same benchmark against sendmail 8.8.7. We did not spend any time optimising the configuration of the sendmail but just used a stock configuration file. We were not after a heads-to-heads comparison but rather a simple estimate of how Meta compared with sendmail. Both Meta and sendmail ran on a 200 Mhz Pentium Pro with 48Mb of memory, an IDE disk, and under FreeBSD 2.2.

The first tweak we made on our SMTP server was to remove the call to fsync to see how the speed would change with not having to perform synchronous disk writes. As can be seen in figure 2 the change is quite dramatic.

From around 200 mails/second it increases to around 1000 mails/second. And the slope of the curve is smaller a well.

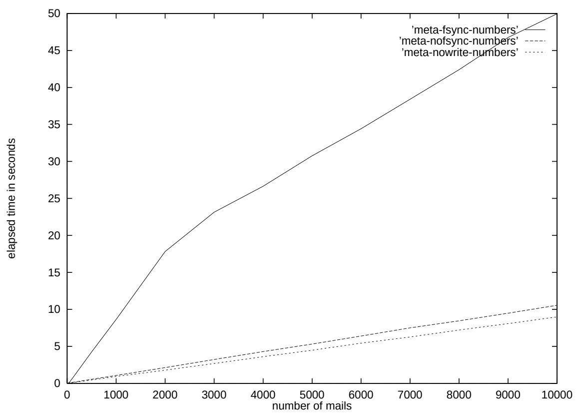

Figure 2: Comparison sync, async, and no-write mail reception

Actually, the extreme case would be to remove the disk I/O completely to see how much of a bottleneck the I/O is. In figure 2 it can be seen that the performance when using asynchronous writes is quite close to that of not having any disk activity at all, and quite far from the synchronous case. Compare the asynchronous case with the no I/O case in this figure and note how much more time the benchmark takes when the writes are synchronous.

A possible optimisation at the SMTP protocol level is to have the mail generator do SMTP pipelining [16]. This means it does not have to wait for every reply to every command before sending the next one and increases the throughput, above all when the sender is fair away. Even in our benchmark case where the sender is very close it makes a difference as can be seen in figure 3. In practice, the savings from doing pipelining is largest when a lot of mails are sent from the same host.

In summary, in table 1 are the approximate number of mails per second it's possible to receive with different configuration, including the one running sendmail.

description	mails/second
meta-fsync	˜ 200
meta-nofsync	˜ 1000
meta-nowrite	˜ 1200
meta-nofsync-pipe	˜ 2000
sendmail-8.8.7	2 – 10

Table 1: Receiving rates for different configurations

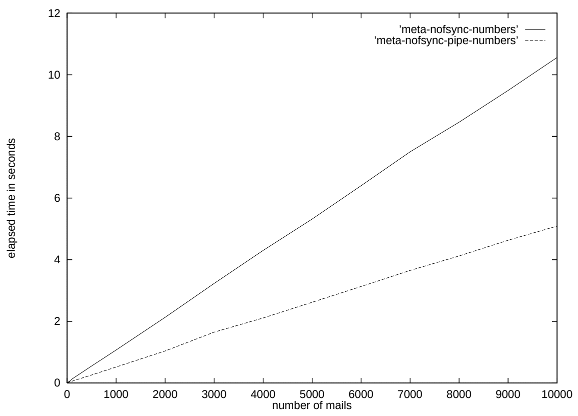

Figure 3: Comparison no-pipelining and pipelining

The conclusions from this experiment was that fsync is going to be important performance-wise and that it is possible to write something that achieves quite a lot better performance than sendmail.

How can we avoid the penalty associated with fsync?

- do not fsync

  That is not a satisfactory solution because when you acknowledge have received a mail you are not supposed to be able to loose it. Users do not like dropped mails.

- fork a different process and let it fsync

  That might be doable but it would require the overhead of fork instead of that of fsync.

- send the file descriptor to a running process

  That seems like one of the best ways of resolving the problem

- clustering

  Have two or more nodes store copies of the mail. With enough nodes running with UPSes storing the mail we might say the system is reliable enough. Clustering is discussed more in section 10.

6 Mail Storage

Meta stores the incoming mails in a series of fix-sized logs. Each message is appended at the tail of a log. When writing the mail is complete, an entry is written for every recipient describing where to find it and in which log. A reference counter is also kept for each message. It is set to the number of recipients when storing the mail and is decremented

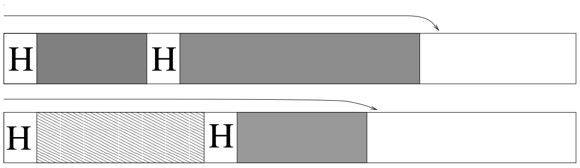

Figure 4: The mail logs

every time an user deletes the mail in a POP session. As the mails are stored consecutively in the logs, only one mail can be written to the same log at any time. The log is locked while writing the mail so that there can only be as many concurrent SMTP sessions as there are logs. On the other hand, the same user can receive several mails at the same time without causing any waiting for locks. When the storage space is too full or Meta is idle, a garbage collector is run that will copy the non-deleted mails from a full log to new logs. To avoid unnecessary coping of the message when it is garbage-collected several times, a generational garbage-collecting algorithm is used. A picture of the logs is shown in figure 4. The arrows point to the first free position in the corresponding log.

The assumptions that we base this choice of data structure on is that mails stay around for a short period of time and are then picked up the client. If the goal was to support something like IMAP where the mails are stored for a long time, some other organisation would probably be better.

Because all mails are received over SMTP and fetched over POP, both of which has the same formatting and quoting rules (CR+LF as line terminator, a single dot on a line as message terminator), the messages are stored in the wire format and are never converted. The CNFS storage for INN can also be configured to behave this way, see [7].

7 Security

Meta runs as an unprivileged user in a chrooted directory. Because no users need to access the spool files, their permissions and owners are not important. All files are owned by the 'meta' user and only read and writable by this user. The mail users need not have accounts and/or shells on the mail hub and in fact that is the recommended configuration. It also makes for a more stable solution as the users have less opportunities to endanger the stability or performance of the mail hub.

The only operation that cannot be done as the meta user is binding the smtp and pop ports. That is done by a small 'nanny' program that just binds the ports, does a chroot to the meta directory, changes the uid, and starts the meta program itself. The nanny also is responsible for re-spawning meta. It is a simple program and consists of only ˜ 70 lines of code and should be easy to audit. That is the only code that run as root. On an operating system that is not that paranoid about ports < 1024 it is not needed either. No program has any set[ug]id bit.

Trying to keep it simple by having it just solve one well-defined problem and not doing any complex stuff should also help making it more secure.

The security model used by Meta is quite different from recent MTAs that have focused on security, typically qmail [9] and postfix [8]. Both of them are composed of a collection of small programs that communicate through IPC and the file systems and that do not trust each other. Meta is a "monolithic" program that handles all in the same program and process.

8 Layers and "back-ends"

Meta has a well-defined API to allow new back ends (called layers) to be added that receive (and store) mails differently. Currently the above described log-based layer, a layer that only sorts mails, and a null layer have been implemented.

9 User Database

Meta keeps a database of all its users. This is completely different and separate from the ordinary `/etc/passwd`. The database can however easily be generated from a standard passwd-file. The database contains the complete mail addresses of the users, including the domain part. This allows Meta to do a lookup on the complete address when checking if it should accept mail for a user. Only if the entire address exists in the database is the mail accepted. Is is therefore unnecessary to configure for what domains Meta should receive mail.

Other per-user information for features that we have not implemented yet (mail quotas, spam filtering, . . .) will also be stored in the user database.

10 Clustering

While it should be possible for large mail-hubs to run a single Meta instance on a single machine, the reliability might not be as high as needed and there might not be room to grow the mail volume gracefully. Therefore we are in the process of implementing support in Meta for running a cluster consisting of collaborating Meta daemons on different machines . We assume that all of them will be presented as SMTP-servers to the Internet at large and to the POP users and therefore be identical. Also, the bandwidth

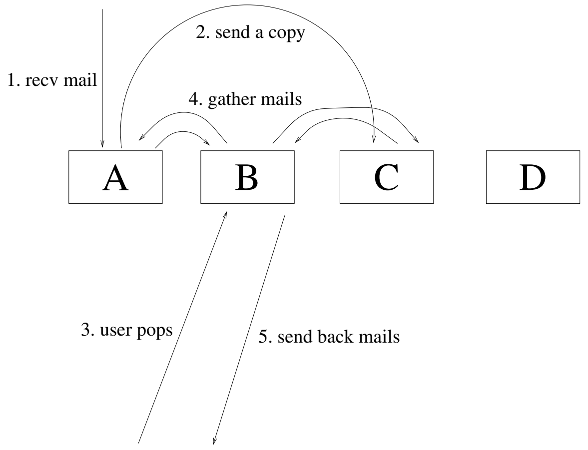

Figure 5: Mail cluster

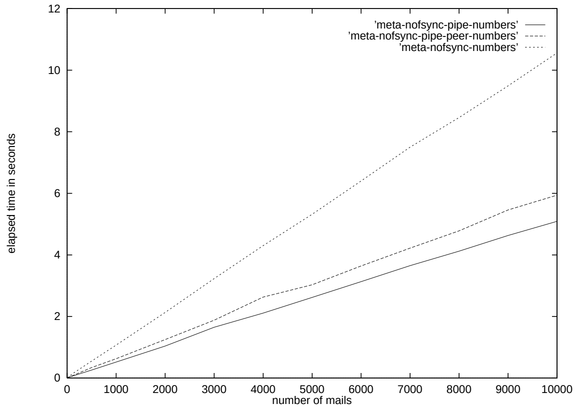

Figure 6: Overhead of sending a copy to a peer

between is considered to be plentiful. The goal here is that Meta should scale, that is, it should be possible to handle more users and larger number of mails by just adding more machines. These machines should share the load and be configured as part of the cluster easily. Meta also tries to store the mails redundantly so that no mails are lost should a single machine crash. The users need not keep tracking of on which machine they have their mails stored but are able to fetch them from any node in the cluster.

A typical scenario is shown in figure 5. First in step (1), the sender of the mail looks up MX records for the Meta cluster. In this case, the first host returned is *A*. A SMTP connection is opened to *A* which will while receiving the mail (2) send a copy of it to *C*. At some time later, a user tries to pop her mail and looks up an A record, getting *B*. In (3) she starts popping from *B*. *B* will then retrieve the mails stored on *A* and *C* in (4) and in (5) send them to the user.

An obvious question is how to choose the other server that will store the redundant copy of a mail. Two policies seem the most appropriate at the moment:

- trying to keep all the mails belonging to a single user on the same host. That way all messages will be stored on the machine that received the mail and the "home" machine for the user.
- choose the least loaded machine, which should give reasonable load-sharing over the nodes in the cluster.

To get some idea as to how much sending an extra copy of the incoming mails to another server would cost, we added that function to the server used in section 5. As is shown in figure 6, the overhead is quite small (around 15%).

10.1 Load Balancing

The load-sharing could be performed by having multiple A records, a load-balancing name server ([17]), or some kind of TCP router ([18, 19]). We have chosen to start with the simplest solution (DNS round-robin) and see if that gives good enough load balance.

11 Usage

Meta is not quite ready to be used in production yet but there are some sites running it experimentally and it is being installed as a front-end to the mail system at a large Swedish ISP. Meta was chosen for its performance at receiving mail.

12 Related Work

Christenson [3] tries to make a scalable mail solution using file servers (NFS) to distribute the mail storage (and thus vulnerable to network-partition). The smtp and pop hosts are not keeping any local data. The local delivery agent is replaced but otherwise a stock sendmail is used. Meta tries to solve a similar problem but from scratch instead of building on sendmail.

Carson[4] discusses the security paradigm "least privilege" applied on mailing. Tries to solve the problem with giving sendmail access to port 25 (SMTP/TCP) by adding a wrapper to sendmail that does that. Meta does exactly that, but do the delivery to the mailbox itself instead of using a local delivery agent.

Knowles[5] focuses on transport issues (inbound, mostly outbound, mailing lists). Arguing that the mail queueing is the slow part of mail delivery, showing techniques to speed-up sendmail's delivery time. Kolstad[6] also tries to tune sendmail to deliver mail faster for mailing lists. Meta is not today trying to solve that problem, it is trying to make in-bound delivery as fast as possible. A outbound MTA could use Meta's logs to avoid most of the problems with the queue-files described in the paper.

Porcupine[1, 2] is architecturely very similar to Meta. They also build a cluster of identical server machines that act as a large mail server. The storage of mails locally on the nodes is however done differently from Meta. The next stage of the Porcupine project underway now is aimed at generalising the ideas that were used for building their mail server.

13 Future Work

There a more work to be done with regards to clustering, to see what are good policies for how to choose server nodes and when to migrate mails between nodes. Load balancing also requires some experimentation and measurements.

There are functionality that we have not implemented but is probably going to be needed like mail quotas, filtering per user (for spam), and other related functionality.

14 More Information

See
`http://www.stacken.kth.se/projekt/meta`.

15 Acknowledgements

Björn Grönvall was responsible for a large part of the initial ideas that led to Meta. Magnus Ahltorp has written part of the code and has also participated in the discussions about Meta.

References

[1] Yasushi Saito, Brian Bershad, Hank Levy, and Eric Hoffman, *The Porcupine Scalable Mail Server*, SIGOPS European Workshop, Sintra, Portugal. September, 1998.

[2] David Becker, David Becker, Brian Bershad, Bertil Folliot, Eric Hoffman, Hank Levy, and Yasushi Saito, *The Porcupine Project*, `http://www.porcupine.cs.washington.edu/`

[3] Nick Christenson Tim Bosserman, and David Beckemeyer , *A Highly Scalable Electronic Mail Service Using Open Systems*, USENIX Symposium on Internet Technologies and Systems, Monterey California (1997)

[4] Mark E. Carson, *Sendmail without the Superuser*, 4th UNIX Security Symposium, Santa Clara California (1993)

[5] Brad Knowles, *Sendmail Performance Tuning for Large Systems*, SANE98, Maastricht, The Netherlands (1998)

[6] Rob Kolstad, *Tuning Sendmail for Large Mailing Lists*, LISA97, San Diego, California (1997)

[7] Scott Lystig Frichie, *The Cyclie News Filesystem: Getting INN To Do More With Less*, LISA XI, San Diego, CA (1997)

[8] Wietse Venema, *Wietse's Postfix Project*, `http://www.postfix.org/`

[9] Dan Bernstein, *qmail: a replacement for sendmail*, `http://www.qmail.org/`

[10] Eric Allman, *Sendmail*, `http://www.sendmail.org/`, `http://www.sendmail.com/`

[11] Jonathan B. Postel, *SIMPLE MAIL TRANSFER PROTOCOL*, Information Sciences Institute, University of Southern California (1982)

[12] J. Myers, M. Rose *Post Office Protocol - Version 3* Carnegie Mellon, Dover Beach Consulting, Inc. (1996)

[13] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, *Design and Implementation of the Sun Network Filesystem*, In Proceedings of the USENIX Summer Technical Conference, 1985.

[14] M. Crispin, *Internet Message Access Protocol - Version 4rev1*, University of Washington (1996)

[15] Brian Kantor, Phil Lapsley, *Network News Transfer Protocol*, U.C. San Diego and U.C. Berkeley (1986)

[16] N. Freed, *SMTP Service Extension for Command Pipelining*, Innosoft (1997)

[17] Roland J. Schemers, III, *lbnamed: A Load Balancing Name Server in Perl*, LISA IX, Monterey, CA (1995)

[18] Cisco, *Cisco LocalDirector*, `http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/index.shtml`

[19] IBM, *SecureWay Network Dispatcher*, `http://www.software.ibm.com/network/dispatcher/`