The following paper was originally published
in the Proceedings of the
Tenth USENIX System Administration Conference (LISA X)
Chicago, IL, USA, Sept. 29 - Oct. 4, 1996

How to Get There From Here: Scaling the Enterprise-Wide Mail Infrastructure

Michael Grubb
Duke University

How to Get There From Here: Scaling the Enterprise-Wide Mail Infrastructure

Michael Grubb – Duke University

ABSTRACT

Recent explosions in volume of mail traffic and in numbers of people making use of Internet email has had a rapidly negative effect on many sites built with assumptions about low mail volume. Older solutions such as NFS are found not to scale upwards to the new email demands of the enterprise-wide email infrastructure. This paper examines this widely-experienced scenario as it unfolded at Duke University, discusses the problems experienced along the way, and describes in detail a solution for migration to a larger-scale infrastructural model that is incremental, transparent to users, and cost-effective.

The Way We Were

Like many installations, electronic mail handling at Duke began in earnest with Unix systems implementing widely-available and well-understood local mail delivery via a local filesystem, and using sendmail [Allman] for long-haul mail transport. When a distributed cluster system was installed for academic use, it was only natural to extend this paradigm using NFS [RFC1094] to provide remote access to a single mail spool for the whole set of machines, and leaving other elements of the local mail system in place.

This cluster mailhub arrangement [Cuccia] is a common scenario and appears to be functional for relatively small numbers of client machines, users, and messages delivered. This level of functionality is deceiving, however, because it does not scale upwards well.

The first problem area we noticed was in the time needed for directory name lookups and file reads of mailboxes via NFS. As the number of users grew, the number of files in the NFS-exported mail spool grew to the point that the server's directory name lookup cache was largely ineffective in preventing frequent sequential seeks for name information in the very large mail spool directory. This caused a slowdown in NFS accesses that caused increased load on the nfsd processes running on the mailhub machine.

At the same time, the number of incoming messages per day to the system was increasing dramatically. Each incoming message was handled by a forked sendmail child to handle the SMTP connection; at peak times (typically between noon and 4 pm local time on weekdays) the sheer number of incoming simultaneous SMTP connections, in conjunction with the increased load already being experienced by the nfsd processes, would overwhelm the available processing time, and the length of the pending run queue would grow precipitously. The resulting excessively high load average would be noticed by sendmail, which would then begin to refuse incoming connections. The pending messages would be processed and the system load average would begin to fall, to the point that load was sufficiently low for sendmail once again to begin accepting incoming SMTP connections, and the cycle would start anew almost immediately.

It was clear in this scenario that the destabilizing factor was the spiking behavior caused by the unpredictable numbers of incoming SMTP connections. The solution adopted was to serialize incoming SMTP connections to the mailhub machine by setting up a mail gateway system. The MX records for the domain address in our BIND name server tables were altered so that all incoming connections would be routed to the mail gateway, like so:

```
acpub.duke.edu. IN MX
        9 gateway.acpub.duke.edu.
```

Sendmail on the mail gateway was configured to queue all messages. This can be done with the "-odq" command line option, accompanied by periodic queue runs with a separate invocation of sendmail. At our site, however, forced queuing of outgoing SMTP messages was done by setting the "e" (for "expensive") flag on the SMTP mailer definitions in sendmail.cf, as in:

```
Msmtp, P=..., F=mDFMuXe, S=...,
       R=..., E=..., L=...,
       M=..., T=..., A=...
```

This can be done simply with one command using sendmail v8's m4 configuration facility:

```
define(`SMTP_MAILER_FLAGS', `e')
```

If the 'c' option (or "HoldExpensive" in sendmail 8.7) is set in the sendmail.cf of the mail gateway, then messages bound for delivery agents with the expensive flag set will be automatically enqueued and delivery attempted at the next queue run rather than immediately. This can be set using sendmail v8's m4 command:

```
define(`confCON_EXPENSIVE', `True')
```

The combination of the expensive flag on the SMTP delivery agents and the HoldExpensive option results in all SMTP messages being enqueued and delivered to the mailhub host serially. Frequent queue runs then safely pass messages to the mailhub without spiking load as specified at the command invocation, as in "/usr/lib/sendmail -bd -q5m" to run the queue every 5 minutes. Sendmail 8.7's "MinQueueAge" option allows such a system to be further configured with a minimum time in queue before delivery of a message is retried. We set "MinQueueAge" at our site to one hour so that down external sites would not clog up queue runs of critical messages to the mailhub.

In order to prevent mail on the mail gateway from looping back or from causing sendmail to notice the loop and bounce the message, a rule was added to ruleset 3 to automatically pass all recognized local messages to the mailhub (defined as $H):

```
R$-     $: $1 < @ $H >
```

The mail gateway can be made to know about local users by carrying passwd table entries for each user; a dummy impossible password encryption such as "nopass*" prevents the user from using the system for other purposes. As long as the domain name is in class w in the mail gateway's sendmail.cf and the mail gateway knows about local users, the sendmail.cf rule above results in addresses of the form "user@acpub.duke.edu" being rewritten by the mail gateway as "user@mailhub.acpub.duke.edu". A similar rule in ruleset 3 on the mailhub system rewrote addresses back to the form "user@acpub.duke.edu":

```
R$* < @ $* .$M > $*  $: $1 < @ $M > $3
```

As long as the domain name is also in class w in the mailhub's sendmail.cf and the sendmail v8 masquerading feature is used to set $M, message headers are rewritten in such a way that users are blissfully shielded from the particulars of the mailhub/mailgateway arrangement. The domain name can be added to class w with a "Cwacpub.duke.edu" directive in the sendmail.cf, or by adding it to an /etc/sendmail.cw file pointed to by a "Fw/etc/sendmail.cw" directive in the sendmail.cf. The sendmail v8 masquerading feature can be easily made use of with this m4 command:

```
MASQUERADE_AS(acpub.duke.edu)
```

We quickly learned that the mail gateway system was not able to keep up with its frequent queue runs due to the volume of incoming mail. This was evidenced by incoming messages remaining in the queue for longer than the queuing interval before being delivered to the mailhub. This would occur when messages came in at a rate faster than the mailhub could dequeue them (roughly one message per second). Multiple mail gateways were then set up, and an MX round robin was created with multiple MX records at the same precedence level, in order to spread the incoming load across multiple systems; see Figure 1. We still found that roughly 10% of our incoming traffic continued to flow directly to the domain host from off-campus sites, presumably due to MTAs not handling MX records correctly, so we set the A record to a gateway system (gateway2, which received significantly less traffic over time than gateway1 and gateway3, presumably due to side-effects of the DNS round robin implementation) as well, as in:

```
acpub.duke.edu.          IN A 152.3.233.10
gateway2.acpub.duke.edu. IN A 152.3.233.10
```

The end result was that incoming SMTP connections went to one of several mail gateway systems, were enqueued, and were then passed on to the mailhub system in a small number of serialized connections rather than in the wild spikes of numbers of connections previously experienced by the mailhub.

This situation lasted stably for a short time until the continuing increase in mail usage began to overwhelm the mailhub. At that point, the system was experiencing peaks of ~300 simultaneous mail readers, there were more than 20,000 user accounts, and 120 client machines of the single NFS server, which was handling more than 1.5 million NFS accesses per day as measured by nfsstat. Disk I/O was abysmal due to the extreme contention among the nfsd processes (write wait times were in the dozens of seconds according to iostat -x). Users would wait for minutes between running a mail application and actually seeing their waiting messages.

The syslog facility was one of the few processes on the machine separable from the mail handling; the logging file system was moved to a new disk on a different SCSI interface in order to minimize I/O conflicts between the constant syslogging activity and the constant mail spool activity. A significant (> 10 second) improvement in disk access times resulted, but was not enough to be considered any kind of a solution.

At this point we were becoming desperate for any and all changes to make that would improve performance in any increment.[1] A number of kernel parameters on the mailhub system were manipulated for this purpose. On a Solaris system many kernel

[1] [1] See [Cockroft] and [NFSTuning] for excellent discussions of the particulars of Solaris performance tuning.

```
acpub.duke.edu. IN MX 9 gateway1.acpub.duke.edu.
                IN MX 9 gateway2.acpub.duke.edu.
                IN MX 9 gateway3.acpub.duke.edu.
```

Figure 1: Spreading incoming load

tables are sized automatically but can be increased to take advantage of available real memory by increasing the maxusers kernel parameter. We also maximally increased the size of the kernel's inode and buffer caches for the same purpose. On a Solaris 2.3 system (the exact numbers to use vary from version to version of Solaris), the /etc/system entries looked like this:

```
set maxusers=1024
set ufs_ninode=34906
set ncsize=34906
set bufhwm=10240
```

NFS mount options on the client hosts were also manipulated in order to maximize efficiency of the NFS connections. In order to minimize storms of NFS retransmissions from the clients, the NFS request timeout and attribute caching periods were dramatically increased. [Stern] An /etc/vfstab entry to accomplish this for Solaris 2.3 systems looked like this:

```
mailhub:/var/mail - /var/mail \
  nfs - yes bg,hard,intr,timeo=90,\
  acregmin=60,acregmax=300,\
  acdirmin=300,acdirmax=600
```

Some of these rather extreme values only make sense because the filesystem was a mail spool and thus directory and file attributes were changing very little. Don't try this with an ordinary multipurpose NFS-mounted file system!

Ultimately, the system ground to a halt. The active attempt of Solaris' rpc.statd to honor file locking via NFS and the impact of 20 or more NFS operations per second imposed such load on the server and introduced such instability, that the nfsd processes under extreme load experienced total meltdown. The system froze, and in a horrible set of experiences we described as "infocalypse", was unable to be restarted with locking honored between server and clients. The restarted server would receive storms of rpc.statd requests when it became available, would be unable to handle the load, and would die or freeze. On these occasions it was necessary to turn off NFS on the entire distributed system manually by killing lockd and statd processes on all client machines in order to bring the server back up stably, when NFS exporting could once again be turned on. This sort of experience is time-consuming and extremely disruptive to a busy system, and resulted in demands from users and management alike for a different way of doing things.

The Way We Wanted to Be

We knew where we wanted to wind up in our changes to the mail system, a system that had inadvertently become the central campus-wide email infrastructure. We were victims of our own success; the stable and fast system we had created for 5,000 users attracted 15,000 more users, who created load beyond the ability of the system to handle. By tweaking parameters on the mailhub, investing in larger and more powerful mailhub hardware, and imposing mail gateways to handle incoming SMTP connections, we had grown the system as far as it could go with the present architecture.

Our desire was to have a truly scalable system that could keep up with future growth for a number of years. We wanted to replace mailbox access via NFS with the POP and IMAP protocols.

POP and IMAP are Internet protocols for remote mail access. POP (current version POP3 [RFC1725]), which has been around in various forms for much more than a decade, is very stable and widely implemented. POP provides access to a single remote mailbox. IMAP, which is somewhat newer and less widely-implemented, is much more flexible. IMAP provides access to multiple remote folders, as well as to individual messages within those folders, as opposed to the bulk mailbox operation of POP.

IMAP2 [RFC1176] (with extensions referred to as "IMAP2bis" [Crispin]) is currently the most widely-used version of the IMAP protocol, although implementations of IMAP4 [RFC1730] clients and servers now becoming available provide greatly expanded and long-awaited functionality such as shared folders and per folder access control lists. IMAP4 servers can also handle IMAP2bis requests from older clients for backward-compatibility. [RFC1732] IMAP4 server implementations typically also provide a daemon for POP3 access to mailboxes for additional backward-compatibility with the already-installed base of mail client software.

We wanted our new system to be able to make use of as many post office machines (that is, machines storing users' mailboxes) as necessary to handle load. Our desired architecture should be scalable to thousands of hosts and millions of messages per day. It should provide an infrastructure that accommodates mobile and disconnected operation. Points of failure should be spread out rather than concentrated in a single machine, as long as mean time between failures can be kept low. The new system should continue to work with the client hardware and software already in place, and it should provide additional functionality such as bulletin boards, shared mailboxes, and mailbox access control lists (ACLs). The IMAP4 protocol looked like a good candidate to provide the backwards compatibility (via widely-used IMAP2bis clients such as pine) and the new features we were looking for.

With a clear idea of what was wrong with our system and a clear idea of what new system we wanted to put in place, we were still paralyzed by the problem of "You can't get there from here." There was no clear, documented, or well-understood path for migrating from the one paradigm to the other. One option for migration appeared to be shutting down our mail systems for an extended downtime, then coming back up with the new servers in place. This option was deemed unacceptable for mission-critical mail services. Another option was to set up the new system in parallel to the old system and offer users the choice of moving their mail to the new system with its added functionality. This option was deemed unacceptable because it would have been costly to implement and maintain two mail systems in parallel for an extended period of time, and because the user community was for the most part not technically inclined, costly to support, and not willing to abandon an already-familiar configuration, even when it was failing under load. The decision was made in this environment to attempt to perform the miracle of salvaging the existing system by converting it to the new paradigm without significantly interrupting mission-critical mail services.

How to Get There From Here

One key insight that made our migration possible was to implement multiple post office machines while still using NFS for mailbox access. This made it possible to move to a multiple post office architecture independently of other changes to the mail system.

The second key insight that allowed us to "get there from here" was the creation of a large number of CNAME DNS records that would allow users to configure their mail software to access their post office by a single unchanging name related to their userid, without having to know or care which post office was actually handling their mail. The most flexible way to do this is to set up a separate domain for your post offices, such as "mail.domain.com", and to have a CNAME for each userid, such as "user1.mail.domain.com", "user2.domain.mail.com", etc. This allows mailboxes to be rearranged on post office servers individually, and this large number of CNAME records is certainly workable on today's BIND name servers. Due to purely political considerations, a slightly different approach was followed at our site, using the same theory. CNAME records were created for each two-letter combination at the beginning of a userid. Thus, for a userid "hiro", his mail would no longer necessarily be available from "acpub.duke.edu", but would be guaranteed to be available from his post office, which would be "mail-hi.acpub.duke.edu", "h" and "i" being the first two letters of his userid. With these post office CNAMEs in place, it became possible to make changes to mail handling for small groups of users at a time without having an effect on all users simultaneously.

The overall migration plan was fairly simple[2]:

1. Create the mail-xx CNAMEs. Route mail from mail gateways to correct post offices using the CNAMEs and the sendmail v8 userdb.
2. Set up parallel post offices, exporting mail spools via NFS. Move groups of users to new post offices to spread load. Leave symbolic links from original mail spool to allow old mail clients to continue to work. For each group moved, update mail-xx CNAMEs accordingly.
3. Turn on POP3 and IMAP2bis servers on each post office machine. Provide users with clients that access mail via POP or IMAP.
4. Turn off NFS exporting from post offices.
5. Set up IMAP4/POP3 servers, and move users group-by-group to the new post offices. Convert vacated post offices to the new IMAP4/POP3 setup, and continue by moving additional mailboxes to the new post offices, repeating until all mailboxes and post offices are converted.

The beauty of this migration path is that each step is completely incremental. It is possible to take each step as slowly or as quickly as necessary, and it is possible to back out of each step along the way. There are no "flag days" for users, no single events of system-wide change that require all users to alter their behavior – most changes are transparent to end users, and those that are not can be easily coordinated with users on an individual basis.

From the point of view of the users, there is a single change: the configuration of their mail reading software must be updated to access their mailbox via the new mail-xx CNAME. Once NFS access to mailboxes is turned off on the post offices, POP and IMAP clients that haven't been updated will no longer be able to access a user's mailbox, although mail delivery will continue uninterrupted. This change can be announced as far in advance and as repeatedly as necessary. By following the logs of POP and IMAP accesses, it is possible to determine if necessary those users who have not reconfigured their clients. These users can then be contacted individually for assistance in reconfiguration. The deadline can be moved back as many times as necessary to avoid seriously inconveniencing any users.

The IMAP4/POP3 server software we decided to use was the Cyrus[3] mail software from Carnegie-Mellon University. This software suite was designed for maximum efficiency for very large sites like ours with dedicated, secured mail servers. Use of this software would minimize the load created by the imapd processes handling client IMAP connections because single messages are loaded into memory rather than entire mailboxes[4]. However, switching to use of the Cyrus software introduced a complication to our migration path. Since the Cyrus server uses a different

[2]The original design of this migration plan grew out of a series of conversations between Karl Ramm and the author.

[3]The Cyrus Project at Carnegie-Mellon University is described in detail at <http://andrew2.andrew.cmu.edu/cyrus/cyrus.html>.

[4]This is also possible with the University of Washington imapd using a non-Berkeley mailbox format; however, this solution presents the same problem of mailbox conversion without gaining the security and administrative simplicity of a sealed server that doesn't know about user accounts.

format for storing mailboxes on disk it was necessary to create new Cyrus servers and convert mailboxes to the new format as they were moved from old servers to the new servers.

Proper sizing of the post office servers during and after this migration is critical to maintaining continuous highly-reliable mail service. One large difference between most users accessing mailboxes via NFS and most users accessing the same Berkeley-style mailboxes via an IMAP2 server is in memory utilization: one or two copies of the entire mailbox are loaded into memory while the imapd is running. The habit of some users of accumulating very large mailboxes, combined with another common habit of opening a mail reader (and thus an imapd on the post office) and leaving it open for a long period of time, can result in very large memory requirements on the post offices. In our environment 512MB of RAM and five times that amount in swap space was barely sufficient to prevent some crashes at peak times. Luckily the memory requirements decrease after the conversion is completed due to the post-conversion IMAP daemon being able to access mailbox messages individually.

Once the IMAP4 servers are in place it is possible to begin making new clients available to users that take advantage of the new features of this protocol. IMAP2 and IMAP2bis clients should remain minimally functional with the new servers, although they will not provide access to the expanded IMAP4 capabilities such as shared message folders, per-folder access control lists, resynchronization for disconnected clients, and so forth.

Once the migration is complete, users have configured their POP and IMAP clients to access their mailboxes using the mail-xx CNAMEs. An IMSP [Collyer] service is also made available for clients that can make use of that service for mailbox location discovery. IMSP is a deprecated protocol for remote authenticated service of mail client configuration information such as mailbox location and addressbook information. IMSP is currently being reworked by the IETF working group on ACAP [Wall], which is more broadly targeted to supply configuration information for other types of clients as well. It is to be hoped that ACAP implementations will continue to provide the mailbox discovery mechanism provided by the IMSP server.

Conclusions

This paper details one odyssey from a small mail system to an enterprise-wide mission-critical infrastructure. There are many ways to tackle this problem, thanks in large part to the power and flexibility of sendmail as a mail transport agent. Some of the particular examples given here are very specific to our primary operating system at the time, Solaris 2.3, but the principles at stake should carry over to other Unix implementations. By providing what we did not have when embarking on this project, one clearly-documented example of migration from a small to a large-sized mail system, it is to be hoped that this paper helps at least one other site to avoid some of the same mistakes. It is possible to ''get there from here'', but it is best to plan ahead carefully rather than to wait for infocalyptic disasters to occur and push you forward.

Author Information

Michael Grubb is a Senior Systems Programmer for Duke University's Office of Information Technology. He is also a licensed attorney. He can be reached via email at <mg@duke.edu> or via post at Box 90132, Durham NC 27708-0132 USA.

References

[Albitz] Albitz, Paul, and Cricket Liu. *DNS and BIND*. Sebastopol, CA: O'Reilly & Associates, Inc., 1992.

[Allman] Allman, Eric. ''Sendmail, An Internetwork Mail Router'' in the *BSD Unix Documentation Set*, Berkeley, CA: University of California, 1986-1993.

[Avolio] Avolio, Frederick M., and Paul A. Vixie. *Sendmail: Theory and Practice.* Boston, MA: Digital Press, 1995.

[Cockroft] Cockroft, Adrian. *Sun Performance and Tuning*. Mountain View, CA: SunSoft Press, 1995.

[Collyer] Collyer, Wallace. *IMSP – Internet Email Scales to the Enterprise*. Available from http://andrew2.andrew.cmu.edu/cyrus/imsp/imsp-white.html .

[Costales] Costales, Bryan, with Eric Allman and Neil Rickert. *sendmail*. Sebastopol, CA: O'Reilly & Associates, Inc., 1993.

[Crispin] Crispin, M. IMAP2bis: ''Extensions to the IMAP2 Protocol, 1992''. Available from ftp://ftp.cac.washington.edu/mail/IMAP2bis.TXT .

[Cuccia] Cuccia, Nichlos H. ''The Design and Implementation of a Multihub Electronic Mail Environment''. San Diego, CA: *USENIX Proceedings – Lisa V*; October 3, 1991.

[Darmohray] Darmohray, Tina M. ''A sendmail.cf Scheme for a Large Network''. San Diego, CA: *USENIX Proceedings – Lisa V*; October 3, 1991.

[Harrison] Harrison, Helen E. ''A Domain Mail System on Dissimilar Computers: Trials and Tribulations of SMTP''. Colorado Springs, CO: *USENIX Proceedings – LISA IV*; October 19, 1990.

[Myers] Myers, J. G. ''IMSP – Internet Message Support Protocol'', 1995. Available from http://andrew2.andrew.cmu.edu/cyrus/rfc/imsp.html .

[Nemeth] Nemeth, Evi, Garth Snyder, Scott Seebass, and Trent R. Hein. ''Electronic Mail'', Chapter

21 in *UNIX System Administration Handbook, 2d ed.* Englewood Cliffs, NJ: Prentice Hall, 1995.

[NFSTuning] Sun Microsystems. *SMCC NFS Server Performance and Tuning Guide*. Mountain View, CA, 1994.

[RFC821] Postel, Jonathan B. *RFC 821: Simple Mail Transfer Protocol*, 1982.

[RFC822] Crocker, David H. *RFC 822: Standard for the Format of ARPA Internet Text Messages*, 1982.

[RFC974] Partridge, Craig. *RFC 974: Mail Routing and the Domain System*, 1986.

[RFC1094] Sun Microsystems, Inc. *RFC 1094: NFS: Network File System Protocol Specification*, 1989.

[RFC1123] Braden, R., ed. *RFC 1123: Requirements for Internet Hosts – Application and Support*, 1989.

[RFC1176] Crispin, M. *RFC 1176: Interactive Mail Access Protocol – Version 2*, 1990.

[RFC1725] Myers, J. and M. *Rose. RFC 1725: Post Office Protocol – Version 3*, 1994.

[RFC1730] Crispin, M. *RFC 1730: Internet Message Access Protocol – Version 4*, 1994.

[RFC1732] Crispin, M. *RFC 1732: IMAP4 Compatibility with IMAP2 and IMAP2bis*, 1994.

[Stern] Stern, Hal. *Managing NFS and NIS*. Sebastopol, CA: O'Reilly & Associates, Inc., 1991.

[Wall] Wall, Matthew. *The Application Configuration Access Protocol and User Mobility on the Internet*, 1996. Available from http://andrew2.andrew.cmu.edu/cyrus/acap/acap-white-paper.html .

Appendix A: Pictorial Diagrams

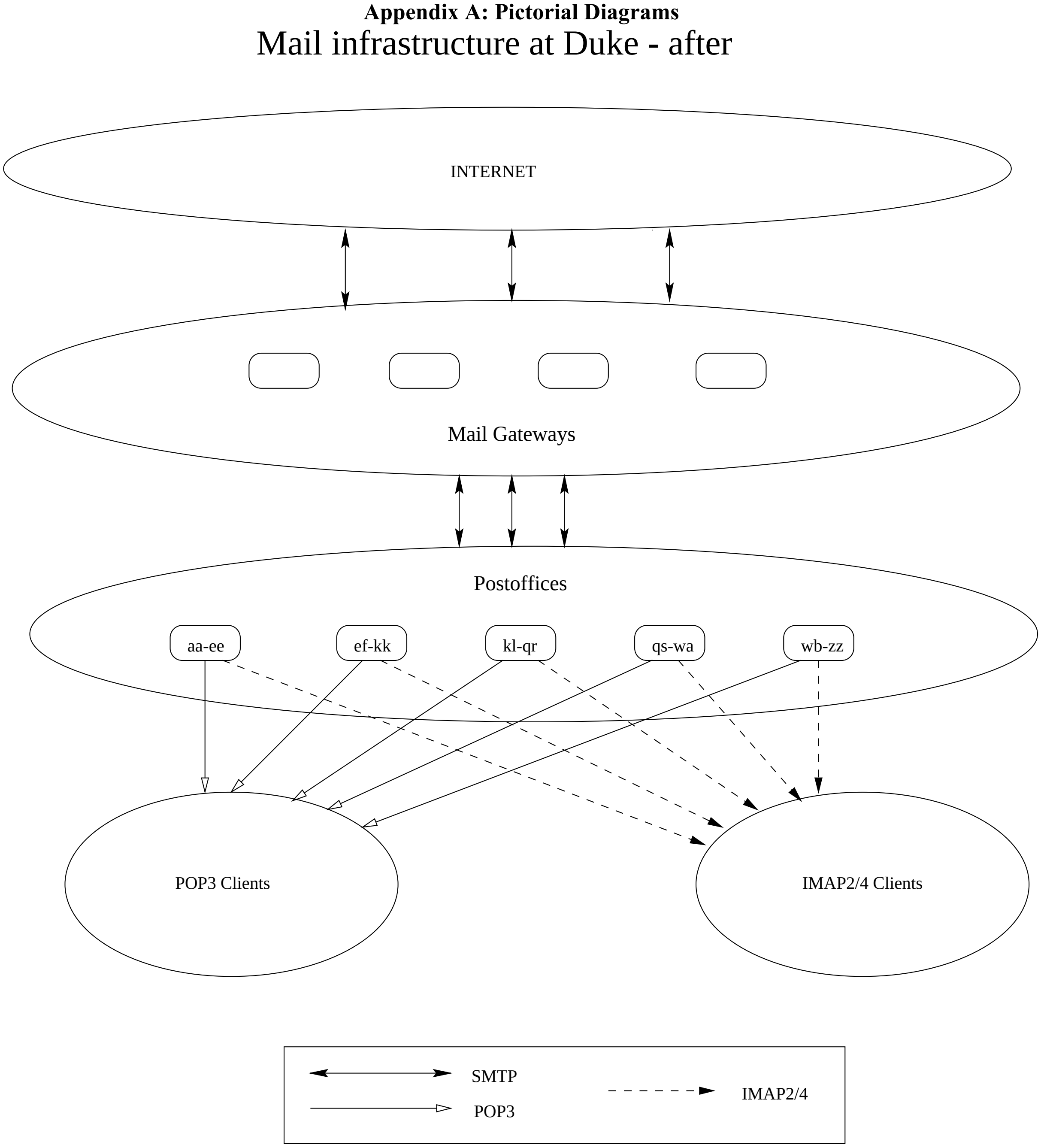

Figure 1: After the Scaling

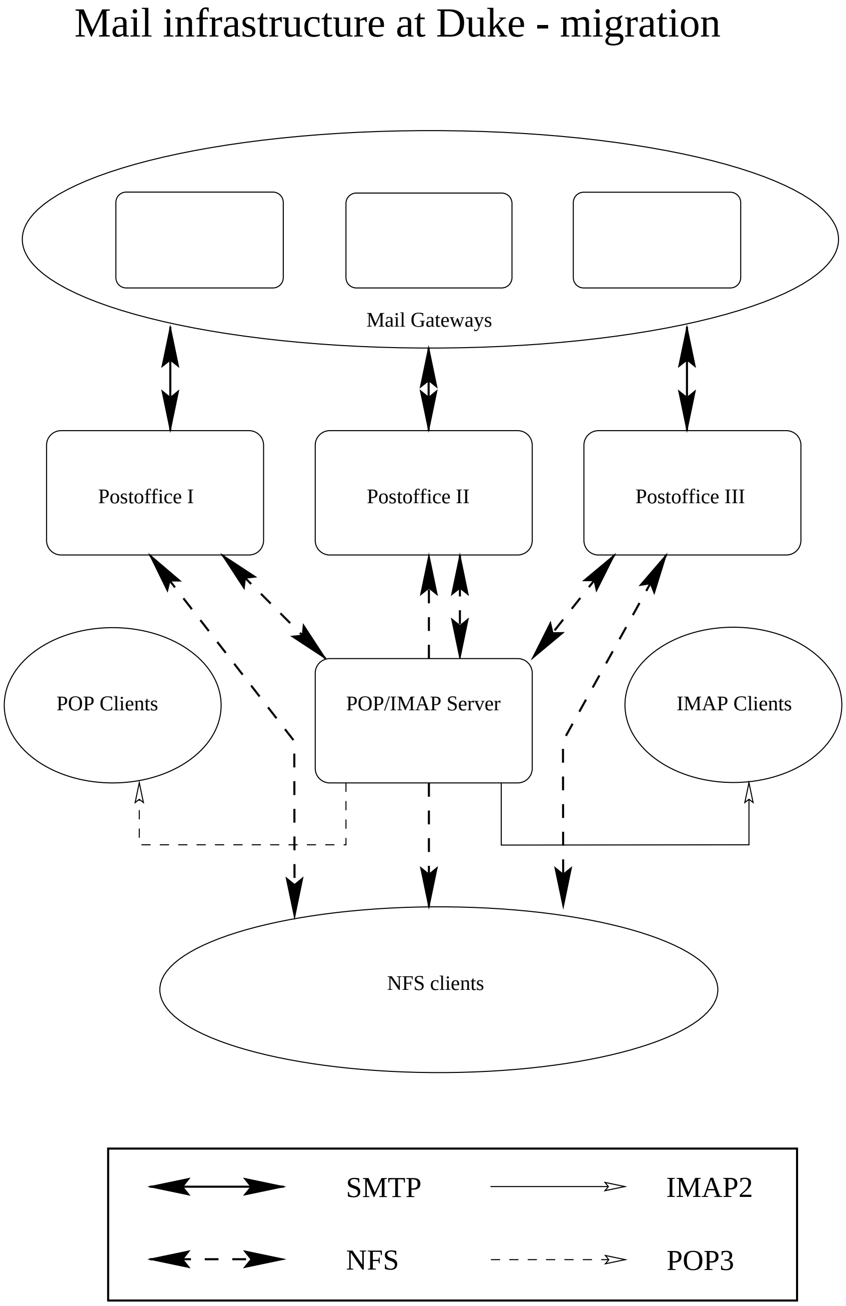

Figure 2: Migration