

# Building A Spam-Aware High Performance Mail Server

Abhinav Pathak (student), Ali Jafri (student), Y. Charlie Hu

*Purdue University*

{pathaka, sjafri, ychu}@purdue.edu

Recently there has been a steep rise in the amount of unsolicited emails, i.e. spams. Such emails overwhelm users' mailboxes, consume server resources and cause delays in mail delivery. Till now, mail server performance has largely remained a non-issue as typical email workload is relatively low, for example, compared to that for web servers. However, the situation is determined to change in the near future due to two contributing factors: First, the volume of spams will continue to grow rapidly, imposing higher and higher load on mail servers; Second, an increasing number of anti-spam techniques, with increasing complexity (driven by the increasing sophistication of spammers), have been or will be deployed as add-on modules to mail servers. Together, it is evident that the need to design "spam-aware" mail servers that optimize their resource utilization by expending minimal resources on rogue emails is inevitable.

Modern mail servers were not originally designed with email spams in mind. As such, as the "common case" workload for mail servers has shifted from legitimate emails to spam emails, we argue it is time to revisit mail server architecture design in following the system design principle of "optimizing the common case". In this work, we show how to optimize the performance of three major components of modern mail servers by exploiting the new "common case" workload.

**Concurrency Architecture:** Contemporary mail servers such as postfix fork (or reuse) a separate process to handle every incoming connection. The process isolation gives security, reliability, robustness and extensibility. However, it becomes highly efficient for a class of spam operations. Using a well-known email harvesting technique, random guessing, a spammer host randomly guesses email ids and attempts to spam such guessed mail ids on a mail server, resulting in bounced emails for incorrect guesses. Our measurement of a Purdue mail server farm shows that the bounce ratio in incoming connections stood at about 25% of all mail connections and continues to rise. Our experiment with postfix shows that the bounces significant degrade its goodput, up to 13% and 30% for bounce ratios of 25% and 50%, due to the overhead incurred in non-differential treatment of bounced emails.

Accordingly, we propose a new hybrid concurrency architecture for mail servers, called *fork-after-trust*, where the server commits its resources only after confirming the legitimacy of an incoming connection. All incoming connections are kept in an event loop; as soon as the server is certain the mail will not bounce, the connection is transferred to a separate process for further processing. The hybrid architecture expends minimum resources on bounced emails while preserving the advantages of process isolation.

**Delivery Architecture:** To use their resources efficiently, spammers often send out one spam mail destined to multiple recipients, using just one connection to the mail server. The mail server then writes the same mail to all the recipients' mailboxes. In our long-term spam collection using a sinkhole, we noticed the number of "rcpt to" in a single spam email is commonly around 10 - 15, but can be as high as 2000. Our experiments with running postfix on a PC show that it writes on average 21.1 mails/sec on an IDE disk for single-recipient emails, but on average 5.4 mails/sec when the number of recipients per mail is increased to 15.

We propose a new file system functionality (and associated API) that supports efficient sharing of parts of large files, analogous to how System V shared memory allows two processes to share parts of their address spaces. This new FS functionality exploits the granularity (a mail) of special files (mailboxes) and applications' knowledge (hints) of shared file units (spam emails) to provide efficient support for rapid storing and later manipulation of targeted file units. For example, mail servers can maintain only one copy on disk for an email marked as spam by spam filters and link it to mailbox files of all the intended recipients of the mail. With proper control of access permission to the mailbox files, the mail servers allow sharing the storage of spams without comprising privacy.

**RBL Querying Architecture:** DNS-based blacklisting is a technique increasingly used by mail servers to query known spam hosts. Upon receiving a new connection, a mail server queries the DNSBL server(s) about the blacklist status of the client IP, in the form of a DNS query to the DNSBL. Our experiments show the such queries can incur a high delay (16%–50% of 30,000 queries sent to the six DNSBLs took more than 100 msec.) Subsequent queries for the same IPs in the near future can benefit from DNS caching which happens at various levels in DNS. The caching scheme helped till recently when large amounts of spams were sent out by single end hosts. As botnets are increasingly taking a lead in sending out spams, and each machine in the bot army sends out only a few spams to a mail server per day, DNS caching becomes less effective.

A recent study by Microsoft Research showed that almost all emails originating from a large number of dynamic IP addresses mostly from blocks of /24 prefixes are spams (accounting for over 42% of overall spams received at Hotmail). Exploiting this finding, we propose an extended query-reply scheme for DNSBL and mail servers. In replying to a query, the DNSBL server also includes the blacklist status of IPs neighboring the IP in question. This can be used by the mail servers in anticipation of further spams from nearby IPs, i.e., in a botnet spam attack.