2009 29th IEEE International Conference on Distributed Computing Systems

The Case for Spam-Aware High Performance Mail Server Architecture

Abhinav Pathak, Syed Ali Raza Jafri and Y. Charlie Hu
Purdue University, West Lafayette, IN 47907

Abstract

The email volume per mailbox has largely remained low and unchanged in the past several decades, and hence mail server performance has largely remained a secondary issue. The steep rise in the amount of unsolicited emails, i.e. spam, in the past decade, however, has permanently disrupted this tranquility of largely steady email volume and turned mail server performance into an increasingly important issue.

In this paper, we point out that modern mail servers were not originally designed with email spam in mind, and 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 particular, we show how to optimize the performance of three major components of modern mail servers, the concurrency architecture, the disk I/O, and DNSBL lookups, by exploiting the new "common case" workload. An evaluation of our prototype implementation of the enhanced postfix architecture shows that the optimizations significantly reduce the CPU, disk, and network resource consumptions, and improves the throughput of the mail server by 18% under a university departmental mail server workload and by 40% under a spam sinkhole workload.

1. Introduction

Recently there has been a steep rise in the amount of unsolicited emails, i.e. spam [7]. Such emails overwhelm users' mailboxes, consume server resources and cause delays in mail delivery. A number of techniques have been developed to stop or mitigate spam. Such techniques include content-based filtering [5], IP-based blacklisting [21], [22], quota enforcement [32], and relationship inference between senders and receivers [6]. Such techniques focus on reducing the impact of spam on the end-user, as opposed to the mail servers. With the increasing trend witnessed in the amount of spam, it is evident that the need to design "smart" mail servers that can optimize their resource utilization by expending minimal resources on rogue emails becomes inevitable.

In this paper, we argue that as the volume of mail spam increases at an unprecedented pace, mail spam are well on their way to constitute the new "common case" workload for mail servers, and hence there is a pressing need to revisit the mail server design by following "optimizing the common case" system design principle.

To make the case for optimizing mail servers to be *spam-aware*, we present a case study of three components of a modern mail server, postfix, for which the high percentage of mail spam has changed the "common case" workload. The three instances of workload shift includes increasing amount of bounced spam mails (mails destined to non-existing mailboxes), large amount of spam mails destined to multiple mailboxes which is the outcome of a simple technique that spammers use to conserve their resources, and large amount of spam originating from botnets which reduce the spam per origin host to evade spam origin detection. The three components of modern mail servers that are affected by these new workloads (and hence can exploit them in improving their design) include the concurrency architecture which controls how to handle many concurrent connections from mail clients, the file system for mailboxes which affects the efficiency in writing the high volume of mails to individual mailboxes, and the DNSBL lookup operations which are widely employed in today's mail servers to query blacklist databases for spam origin detection.

We discuss how the shift in common-case workload exposes inefficiencies in the current design of the components being studied and present an optimized design that exploits the new common-case workload for each of the components. First, our new concurrency architecture improves the performance of mail servers by early differential treatment of a class of mail spam from the rest of the mails. In particular, a server's resource is committed to an incoming email only after confirming the legitimacy (w.r.t to bounce/non-bounce nature) of the email. Second, we propose a new file system that targets applications (e.g. mail servers) that have explicit knowledge of the sharing of parts of different files and the sharing granularity (e.g. a mail), and leverages such knowledge to provide efficient sharing of different files, analogous to how System V shared memory allows two processes to share parts of their address spaces. This specialized file system can be used to avoid duplicate disk I/Os for spam sent to multiple mailboxes. Third, we propose a new scheme for DNSBL servers in replying to blacklist queries that works well under the new trend of spammers using botnets for sending spam to improve the effectiveness of DNS caching.

We have implemented the set of optimizations by modifying the postfix [17] server and evaluated the performance improvement using a spam trace collected over a two-month

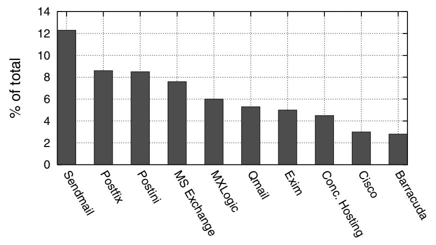

Figure 1. Distribution of mail servers used, from fingerprinting 400,000 company domains remotely.

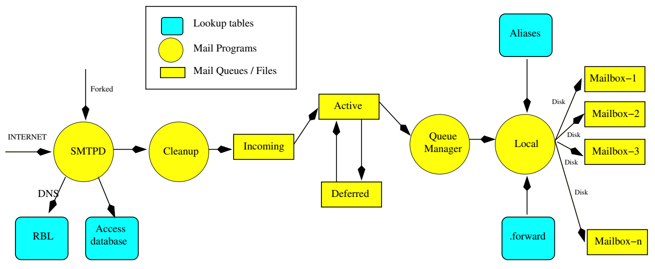

Figure 2. Postfix Architecture.

Table 1. Measurement testbed, software, and traces.

Server & Client Machine	Intel(R) Xeon(TM) CPU 3.00GHz,1 MB cache 2 GB RAM,146 GB U320 10K SCSI drive Linux Kernel - 2.6.20 - Ext3 Journal FS Gigabit switch: 30 ms network delay emulated using traffic control
Server Software	Postfix version: 2.4.1, configurations: smtpd_client_connection_count_limit = 0
Client Program 1	C program that speaks SMTP and maintains a configurable number of concurrent connections. This implements the closed-system model [24].
Client Program 2	C program that speaks SMTP and initiates new connections at a configurable rate. This implements the open-system model [24].
Spam trace	Collected at a spam sinkhole: May-June 2007 Number of connections: 101,692 Number of unique IP addresses: 19,492 Number of unique /24 Prefixes: 8,832
Univ trace	University department mail server: Nov 2007 Number of connections: 1,862,349 Number of unique IP addresses: 621,124 Number of unique /24 Prefixes: 344,679 Spam Ratio: 67% (flagged by Spam-Assassin)

period at a spam sinkhole and a one-month email trace of a large EECS university department. Our evaluation shows that the set of optimizations significantly reduce the CPU, disk, and network resource consumptions, and improve the throughput of the mail server by up to 40%. As the amount of spam continues to increase [7], we expect the performance improvement of our optimizations will become even larger.

2. Background: Mail Server Architecture

In this section, we briefly review the architectures of some popular email servers or mail transfer agents (MTAs). Figure 1 shows a distribution of mail servers currently used [25] in the Internet (measured in January 2007). Sendmail , written in the early 1980's, is still widely used today. The sendmail program is a monolithic piece of code providing all functionalities of the mail server. These functionalities are evoked by calling the sendmail program using different parameters. Sendmail creates a new heavy-weight process for each new mail delivered and thus incurs considerable overhead. Over the years, sendmail has accrued a long list of security vulnerabilities. The fact that sendmail must run with root privileges also makes it a prime target for attacks [9].

Several architectures have been proposed to counter the security loop holes and performance flaws in sendmail, such as qmail and postfix [18]. Since postfix has been shown to outperform qmail [12], we choose postfix as the representative MTA in our study.

The Postfix architecture. The postfix mail server is implemented using a dozen semi-resident, mutually-cooperating processes, each of which performs specific tasks. These processes communicate using UNIX domain sockets. An inetd-like resident process, called "master", runs the postfix server. It creates processes on demand to accept connections from clients, receive mails from the network, send mails, process delivery, manage various queues, and so on.

An incoming mail passes through several processes and several queues before being finally delivered to a mailbox or relayed back to another mail server in the Internet. Figure 2 shows a stripped down version of the path followed by an incoming mail. Upon receiving a new connection request, the master process forks a new process, "smtpd", to handle the connection. This smtpd process performs all the SMTP transactions with the client starting from accepting the connection till shutting it down. This process can be configured to query the DNS for blacklist check. Smtpd also queries the local access database to find if the recipients of the mails exist or not. The master process forks up to a pre-configured maximal number (by default 100) of smtpd processes to handle incoming connections. If an smtpd process is idle for a pre-configured period of time or has served a pre-configured number of requests, it terminates itself after sending a notifying message to the master. The reuse of smtpd processes to handle connections saves on process forking overhead. We skip the details of the remaining processes of postfix due to page limitation.

3. Experiment Methodology

We now describe the testbed, the software, and the workload used in the experiments described in this paper.

Testbed and workload. Table 1 lists the configurations of the measurement system, i.e., the server and client, used throughout the paper.

A mail server's performance depends on the characteristics of mail workload, such as the mail size, the number of recipients each mail is destined to, and whether the mail is a bounce, i.e., destined to a non-existing mailbox. In order to evaluate the performance of various aspects of a mail server for today's "common case", we collected real traces from mail servers to tune the input workload.

The "Univ" trace is a real trace of mails collected at a university department mail server serving over 400 mailboxes during the month of November 2007. The trace contains a mix of spam and legitimate mails, and is representative of a typical university department mail workload.

Since the Univ trace contains no information about unfinished SMTP connections, to measure the extent of bounced mails and unfinished SMTP connections (see Section 4.1 for detail), we collected bounce statistics at the mail server maintained by the Engineering Computer Network (ECN) at Purdue over a period of about one year starting from December 15, 2006. The ECN mail server hosts about 20,000 mail users.

To study the impact of different characteristics of spams such as the bounce ratio and the number of destination mailboxes per SMTP session on mail server performance, we derived a synthetic trace from the Univ trace which follows the mail sizes in the Univ trace while varying other parameters such as the bounce ratios or the number of destination mailboxes per SMTP session.

To measure the IP-level characteristics of spammers and the distribution of the number of recipients their mails are destined to, we also collected a spam trace from a spam sinkhole in the months of May and June 2007.

Tuning postfix. Postfix uses a default of 100 "smtpd" processes to handle incoming SMTP connections. On a given server hardware, the performance of postfix depends on this configurable parameter. A low value for this parameter can result in low CPU utilization while a very high value can result in performance degradation due to high context switching overhead. We experimentally searched for the optimal process limit value on our server using client program 1 driving the Univ trace, and found the throughput of postfix peaks at about 180 mails/sec with the process limit configured at 500.

4. The New Common Case: Spam

In this section, we make the observation that the rise and prevalence of mail spam in the last decade has permanently changed the "common case" workload of mail servers, and discuss the performance implications of this workload change to the three components of a popular mail server architecture.

4.1. Bounces and Unfinished SMTP Transactions

One of the heavily used spamming techniques is random guessing (RG) [19], where spammers send mails addressed to commonly used user names (mailboxes) with the hope of hitting valid ones. When the mail client tries to deliver a mail to an invalid user on the mail server, the mail server generates a "550 User unknown" response. For this reason, we call such mails *bounces*. In addition to trying out their luck, spammers also exploit this feedback mechanism of mail servers to learn the legitimate mail addresses at a domain, turning random guessing into a form of mail address harvesting [1].

Figure 3 shows the fraction of bounces to the total number of mails delivered during the measurement period. We observe that about 20% to 25% of the mails reaching the ECN mail server at Purdue are bounced, i.e., there are no real users associated with the "rcpt to" field of the mail. With increasing amount of resources, spammers are likely to send more and more mails to randomly guessed mailboxes, turning such bounce mails into the "common case" among all the mails received. In fact, Figure 3 shows a slight increase in the percentage of bounces within a year's time frame.

Another type of SMTP sessions observed today are the ones that do not last till completion. In such SMTP transactions, a client connects to an SMTP server, may send a few SMTP handshake messages and then quits the connection. These connections do not deliver any mails to any user. Figure 3 shows such incomplete connections account for 5% to 15% of the connections received by the ECN mail server during the same time frame. For simplicity, from now onwards, we refer to both bounce and unfinished SMTP transactions as bounce connections unless otherwise stated.

Implications. The server architectures used in popular mail servers delegate a new process for each SMTP connection received. Such a concurrency architecture can waste significant server resources in case of bounces. According to the ECN data, bounces and rogue connections currently stands between 25 and 45%, which motivates the need to revisit the mail server concurrency architecture.

4.2. Multiple Recipients

Since setting up a new connection to a mail server is costly, spammers frequently make use of the multi-recipient feature of SMTP by making one connection to a mail server and addressing a spam mail to several recipients. Figure 4 shows the distribution of the number of recipients per connection for the spam in the two-month spam trace (Section 3). We observe the number of "rcpt to" fields in a single spam mail is commonly between 5-15. In contrast, the average number of RCPTs in the good mail in our Univ trace is only 1.02, consistent with Clayton's study [3].

Implications. While the use of the multiple-recipient feature of SMTP for legitimate mails have largely constituted the

1. The original intent of the bounce mechanism is to help the sender to detect mistakes with mail addresses, e.g., from mis-typing.

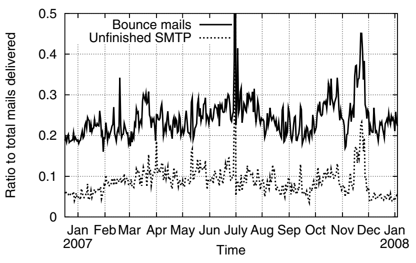

Figure 3. Daily bounce ratio and ratio of unfinished SMTP transactions on the ECN mail server during year 2007.

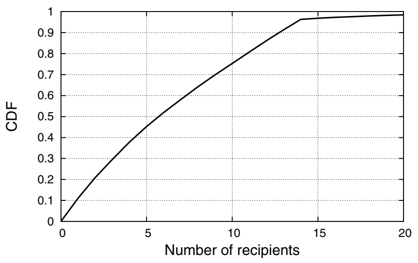

Figure 4. CDF of the number of recipients per mail in the sinkhole trace.

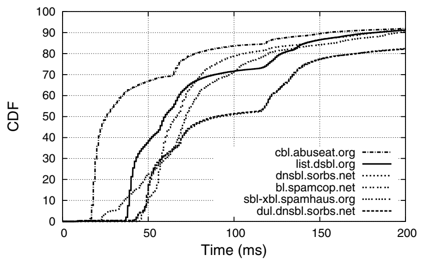

Figure 5. CDF of time to query various DNSBL servers for over 19,000 spam IPs.

rare case in the past, the wide-spread use of this feature by spammers along with the increasing volume of spam in the Internet justify the need for mail servers to optimize the duplicated disk I/Os. While disk space is not a major issue with falling disk prices, duplicated disk writes can consume significant server resources, reducing the mail server throughput.

4.3. DNSBL Lookups

One of the most effective and widely used techniques in the ongoing battle against spam is DNS-based blacklisting. The DNS-based blacklists maintain the IP addresses of hosts that are known to be spam origins/relays. Upon receiving a new connection, a mail server queries DNSBL[2] server(s) about the blacklist status of the client IP, in the form of a DNS query. The IP of the client is reversed and appended to the blacklist server's name to query. For example, if a client with IP x.y.z.w contacts a mail server, the server issues a DNS query for w.z.y.x.blacklistserver. This query is routed to the blacklist server (a DNS server). If the IP is blacklisted, the response to this DNS query will be of the form 127.0.0.x where x suggests the form of spamming activity done by the corresponding IP; otherwise, the DNS query will return with empty answer field.

As the amount of spam continues to increase, DNSBL querying will constitute a growing portion of DNS lookups. For example, Jung and Sit [10] accounted 14% of all DNS lookup at CSAIL, MIT to be DNSBL queries in 2004 compared to only 0.4% in 2000.

Implications. Querying DNSBL servers consumes resources of both the mail server and the DNS servers. Due to heavy load from DNSBL queries, several ISPs decided to block out DNS requests for major DNSBL lists. For example, in April 2004, AT&T's business department blocked queries to several major spam-blocking DNSBL lists because of the "extra" load on its DNS servers [26].

DNSBL queries also take significant time to get resolved which adds to the delay in completing an SMTP session. To find the delay overhead incurred, we compiled a list of over 19,492 IP addresses that spammed our sinkhole, then performed DNSBL queries for these IPs. Figure 5 shows the distribution of the time taken to query six DNSBLs for the blacklisting status of these IPs. We observe between 16%–50% of 19,000 queries sent to the six DNSBLs took more than 100 msec.

DNS caching has helped to reduce the cost of querying against the DNSBL servers in the early days of mail spam when most of the spam were sent out by a few individual end hosts. However, in recent years there is a growing trend that spammers employ a large number of compromised machines, known as bots, to carry out mail spamming on their behalf [27]. Each machine in the botnet sends out relatively few spam to a particular mail server (domain) to avoid detection. This transition from high-volume spammers to low-volume spammers renders DNS caching of per-IP based DNSBL lookups much less effective.

5. Hybrid Concurrency Architecture for Processing Bounces

In this and the next two sections, we present optimizations to the three components of mail servers that target the three new "common case" workloads discussed in Section 4. Together, these optimizations constitute a first step towards "spam-aware" mail server architecture design.

5.1. Design

To optimize the processing of the increasing amount of bounce connections, we propose a new concurrency architecture, called "fork-after-trust", shown in Figure 7. The basic idea is to reduce the overhead of processing bounces by *delaying* delegating such a connection to an smtpd process. All new and unconfirmed (yet) connections are kept by the master process (master) in a list of sockets S. Any event happening to the sockets in S is processed by an event loop using "select/poll". The event-based architecture is used until the validity of the "rcpt to" address is determined, after which the socket is delegated to a separate process which will finish the SMTP transaction. Thus, the overhead

2. Experience from our spam testbed and [22] suggests that IP-based blacklisting works well if many blacklists are queried simultaneously for the same IP.

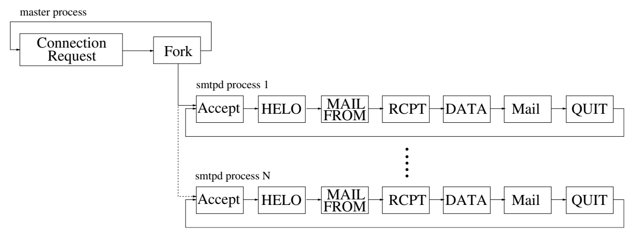

Figure 6. Process-per-connection architecture.

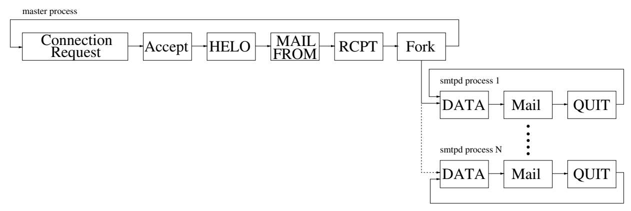

Figure 7. "Fork-after-trust" postfix architecture.

of process creation and context switching for processing bounces is avoided.

5.2. Security Consideration

We argue that in moving from a pure process-based architecture to the "fork-after-trust" architecture, we have compromised minimally on the security of the mail server.

In our hybrid architecture, only part of each SMTP transaction, i.e., till receiving a valid "rcpt to", is handled in the event loop. Since the initial part of the SMTP handshake, e.g. "mail from" and "rcpt to", are handled with a fixed-size receive buffer, the event loop in the master process does not introduce new buffer overflow opportunities.

After a valid recipient is confirmed, the processing of the data part of the mail and the subsequent file I/O of the mail is handed over to a separate smtpd process. Such processing involves heavy operations, and the implementation is much more likely to contain security loopholes than the initial handshaking steps of the transaction. Furthermore, after receiving the data part of the mail, many body tests are performed by various third-party spam filter modules such as keyword matching, image spam testing, which are widely employed by mail servers nowadays. In our hybrid architecture, such add-on functionalities to mail servers remain handled by separate processes, and hence continue to enjoy the security benefit from process isolation.

5.3. Implementation

We implement the "fork-after-trust" model by directly modifying the master and smtpd processes in the postfix mail server. Figure 6 depicts the operations of the current postfix mail server. Briefly, the master process uses events to receive notification of an incoming connection request. Upon the arrival of a connection request, an smtpd process is forked. This smtpd process inherits the master process' listening socket and calls accept() to create the connection. The SMTP dialog is then carried out between the client and the smtpd process.

Figure 7 shows the operations of the new hybrid concurrency architecture. Upon arrival of a connection request, the new master process performs accept and continues the SMTP transaction using the event loop until receiving "rcpt to" fields. If no valid recipient address is received, the connection is terminated and all associated data is freed. If even a single recipient address is confirmed to be valid, the master process delegates the connection to one of the smtpd processes. As in the original postfix, the new postfix is also pre-configured with a maximal number of smtpd processes to be forked. For each forked smtpd process, the master creates a UNIX domain socket connection between itself and the smtpd. Upon creation, the smtpd process immediately starts listening on this connection (using event notification) for SMTP connections delegated from the master process.

The information collected for an SMTP connection prior to its delegation of work to an smtpd process comprises the sender IP address, the sender's mail address and the recipient's mail address. Aside from this data, the connection socket with the client also needs to be transferred from the master process to the smtpd process. Upon receiving this data, the smtpd continues the mail transaction with the client till completion and returns to its state of listening on the UNIX domain connection with the master process.

There is an overhead involved for an smtpd to notify the master process every time it is idle and ready to process the next connection. To avoid this, the master process exploits vector sends to queue multiple tasks (connections) delegated to an smtpd process. Specifically, since the vector send takes as a parameter the size of the payload, even if multiple tasks consecutively are sent into the UNIX domain connection to an smtpd process, that process will be able to read out one task at a time. Based on the default kernel socket buffer size (64 KB) and assuming a single mail is on average addressed to 7 recipients, we estimate on average the socket between the master and an smtpd can hold up to 28 tasks. To balance the load on the multiple smtpd processes, the master process uses nonblocking writes to these sockets in a round-robin fashion. In case the mail server reaches its capacity limit, the master process cannot dispatch more connections to the smtpd processes. Hence the finite-sized socket buffers for the sockets also act as a natural throttle for the master process.

5.4. Evaluation

We implemented the new architecture by modifying postfix version 2.4.1 and evaluated the goodput improvement of the new version on our testbed. Since the modified postfix is expected to outperform the original postfix which achieves the maximal throughput when using 500 processes, we configured the modified postfix to use up to a maximum of 700 sockets. Client program 1 was used to drive the

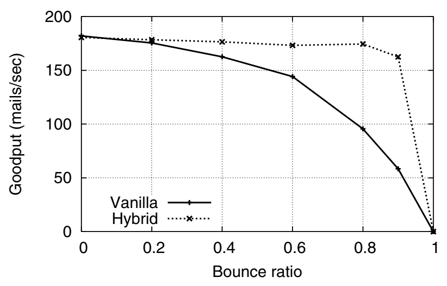

Figure 8. Performance of the current mail server architecture (postfix) and our model.

server for 5 minutes using the synthetic trace with varying bounce ratios. We measured goodput as the number of good mails per second received by postfix. Figure 8 compares the goodput of the new postfix architecture ("Hybrid") to that of the vanilla postfix ("Vanilla") as the ratio of the bounces in the mix of mails sent is increased. We observe the goodput of postfix steadily declines as the percentage of bounce mails is increased. In contrast, the goodput of new postfix stays almost constant until the bounce ratio reaches 0.9, indicating that minimal amount of resources were spent on handling bounced mails. Since smtpd processes are recycled in both versions, the efficiency of the hybrid architecture comes from avoiding context switches in processing bounces; the total number of context switches is reduced by close to a factor of two.

6. Single-copy File System for Mailboxes

In Section 4.2, we discussed that the wide-spread use of the multi-recipient feature by spammers for resource efficiency reasons together with the increasing volume of spam in the Internet justify the need for optimizing duplicated disk I/Os. While numerous single-copy file systems or storages systems have been previously proposed [20], [13], [28], [1], these systems do not leverage the applications' explicit knowledge of potential duplicated disk I/Os. As a result, they often avoid duplicates at the disk block level, for example, by revamping the underlying storage system to be content-addressable [20], [28], or perform post-processing to eliminate duplicated files on disk [1]. In this section, we propose a simple specialized file system that leverages the applications' explicit knowledge of sharing parts of different, typically large, files (e.g. mailboxes containing all mails). It provides efficient sharing of the common parts of different files to avoid duplicated disk I/Os in writing them.

In principle, the specialized file system can be used to optimize disk I/Os for the class of applications that have explicit knowledge of sharing of different files and the sharing granularity, such as mail servers, syslogs, etc. For ease of illustration we use mail server as a concrete example to illustrate the design and the API of the file system, and name the file system MFS for short. The API and the functionality can be trivially generalized to the class of applications.

6.1. Design

MFS is designed for use by applications that have explicit knowledge of sharing among files as well as the sharing granularity. For example, in the case of mail server applications (mail server/POP/IMAP servers), all the writing, reading, and deletion are done in units of mails. Other types of accesses to a mailbox file are generally not performed. Consequently, the natural sharing granularity is a mail, for example, when a mail is destined to multiple recipients. This in turn implies MFS is record-oriented, where a record is simply a mail in the case of mail servers, as opposed to byte-oriented.

To manipulate files at the record-granularity, we need an addressing scheme to uniquely identify each individual record. In the case of mail server applications, every mail has its unique ID labeled by the MTA, e.g. postfix[3], when it was received [11], which can conveniently serve as the unique index key for that mail.

For simplicity and ease of deployment, in this paper, we design MFS to be a simple application-level extension to any conventional byte-oriented file system, as shown in Figure 9. A file in MFS is made up of two conventional files. The primary "key" file contains a list of (key, offset, reference count) tuples, where the keys uniquely identify the records that constitute the content of the file, and the offset for each key stores the offset of the record actually stored in the companion shadow "data" file. Only records that are unique to this MFS file are stored in the companion "data" file. The records that are shared among this and other MFS files are stored in one (or multiple if necessary) "shared" MFS file. Since a shared record cannot be deleted until it is deleted from all MFS files that share it, a 4-byte reference count is maintained for each shared record in the shared MFS file.

To support permissions of files belong to different users, the "shared" MFS file will be implemented in the kernel, i.e., hidden from the users. The only way to access them is through the system call APIs exported by MFS.

Figure 9 shows how a mail server makes use of MFS as follows. First, all the mailboxes of a mail server are MFS files. We denote the key file and the data file for a mailbox as $mailbox\_key$ and $mailbox\_data$. The $mailbox\_key$ file contains the mail-ids of all the mails in the mailbox, and the $mailbox\_data$ file contains the individual mails destined only to that mailbox, i.e., single-recipient mails. A special mailbox is used by the mail server to store mails destined to multiple recipients. We denote its key file and data file as $shmailbox\_key$ and $shmailbox\_data$. When a mail is destined to a single recipient $mailbox1$, the mail is appended to $mailbox1\_data$ and the tuple of (mail-id, offset, 1) is

3. We do not trust mail id as sent by client and rely on mail id generated by mail server.

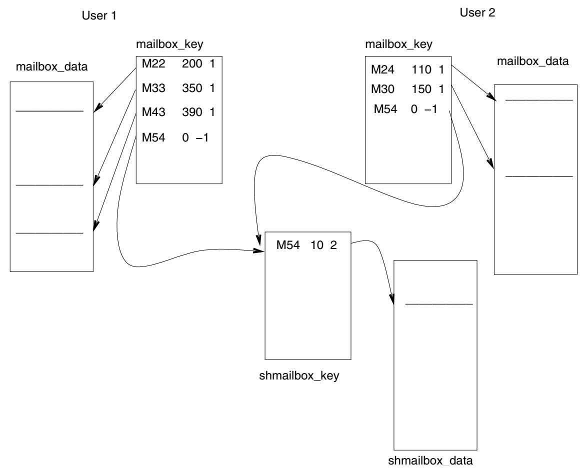

Figure 9. File structure in MFS.

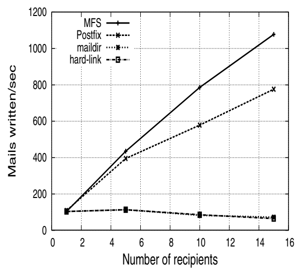

Figure 10. Throughput of the four versions of postfix on Ext3 Filesystem, with increasing # of recipients.

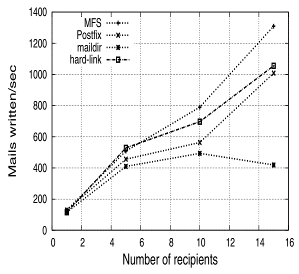

Figure 11. Throughput of the four versions of postfix on Reiser Filesystem, with increasing # of recipients.

appended to $mailbox1\_key$. When a mail is destined to multiple recipients, the mail is appended to $shmailbox\_data$, and the correct (mail-id, offset, num_of_recipients) tuple is appended to $shmailbox\_key$, and a tuple (mail-id, offset, -1) is inserted into all the relevant $mailbox\_key$ files.

6.2. API and Implementation

We now describe the set of APIs exported by MFS to the applications:

• `mail_file *mfd = mail_open(char *filename, char *mode)` opens a mailbox file with the specified filename and the mode to be opened, and sets the seek pointer to point to the first mail in the file. The call returns a pointer to the `mail_file` structure. In our implementation, the `mail_file` structure that implements the file descriptor in MFS is no different from an inode in UFS. If the file does not exist, the proper $mailbox\_key$ and $mailbox\_data$ files are created.

• `int err = mail_seek(mail_file *mfd, int offset, int whence)` performs seek in the specified file and operates at the granularity of a mail instead of a byte, since all read, write, and delete operations to a mail file are assumed to be in units of mails. `whence` specifies the starting location from where to apply offset.

• `int err = mail_nwrite(mail_file **mfd, int nmfd, char *buf, char *mail_id, int buf_len, int msg_id_len)` writes a mail to the `nmfd` mailboxes specified by the mfd file descriptor array. The `mail_id` is passed as a parameter by the mail server application. The file system skips the steps of writing data to the $shmailbox\_data$ file if it finds that mail-id already exists in the $shmailbox\_key$ file.

• `int err = mail_read(mail_file *mfd, char *buf, char *mail_id, int *buf_len, int *mail_id_len)` reads the next mail in the user mailbox pointed by the seek pointer. On return, the interface fills in the input buffer, length and `mail_id` parameters based on the mail read. The API may need to be called multiple times to read a mail if the provided buffer is smaller than the mail.

Due to page limitation, we skip the details of the above APIs and of the remaining APIs mail_delete() and mail_close().

6.3. Evaluation

We implemented MFS as a user-space library that exports the API discussed above. We created a new version of postfix that has a modified local process module that uses the MFS API. The new postfix is then linked with the MFS library. The modified postfix continues to use regular files for temporary files, such as those in the incoming queue, which are never shared.

We measure the impact of the number of recipients (“rcpt to”) in a mail transaction on the performance improvement of the modified postfix as follows. We first performed controlled experiments by running Client Program 1 with a modified synthetic trace, and compared (1) modified postfix (MFS in figure 10) with (2) vanilla postfix which uses one mailbox file per user (Postfix), (3) a version of postfix that uses individual files for storing individual emails (maildir), and (4) an optimized version of (3) that avoids storing duplicate copies of the same mail by storing one copy of the mail in a file and hardlinking additional copies (hard-link). The modified input trace has a zero bounce ratio, and contains repeated sequences of mails destined to 15 distinct mailboxes, i.e., each sequence of 15 mails share the same size, and the sequence of sizes are taken from the Univ trace.

Figure 10 plots the mail throughput, i.e., how many mails are written to the mailboxes per second, for the four postfix versions as the number of recipients per connection is varied. For example, using 5 “rcpt to” fields per connection, a client needs to use 3 separate connections to send each sequence to the server. The base filesystem in this case is Ext3 Journal File System. We make the following observations. First, as the number of recipients per connection is increased from 1 to 15, the mail throughput of the vanilla postfix is increased by a factor of 7.2. This is because while using the multiple-recipient feature to send a mail to multiple mailboxes reduces the network processing per mailbox by a factor of 15, the mail is still written to the mailbox 15 times, one for each destination mailbox. Second, the MFS further improves the mail throughput by 39% (for 15 recipients per connection case) compared to the vanilla postfix. This improvement comes from reduced disk I/Os to the file mailboxes. Third, both maildir version and its optimized version (hard-link) perform much worse when compared to one-file-per-mailbox postfix and MFS, as the number of recipients increases. [16] has shown that for workloads consisting of multiple file creations of small sizes, Ext3-Journal Filesystem performs poorly while the Reiser Filesystem performs the best.

We also compared all four versions of postfix with the Reiser Filesystem as the base file system. Figure 11 shows that, although the maildir format still performs the worst, its optimized version, hard-link, improves significantly. However, MFS still outperforms all other versions, hardlink, vanilla postfix, and maildir, by about 29.5%, 31% and 212%, respectively, for the 15 recipient case.

Finally, we evaluated mail throughput of the two versions of postfix under the two-month spam trace collected by our sinkhole. The average number of recipients per connection in this trace is about 7. Our measurement shows that the MFS throughput outperforms the vanilla postfix by 20% in terms of mail throughput.

6.4. Security Consideration

A potential attack on MFS is the random guessing attack, typically used in content addressable systems (for example, [28]), where a malicious user may attempt to write junk with guessed mail-ids to its own mailbox, in hope of hitting an existing mail in the shared mailbox file and then accessing the mail. However, in MFS, all mail-ids are assumed to be unique as they are generated by the same mail server, and hence writing a mail with a mail-id colliding with an existing mail-id will be identified by the mail_nwrite API as an attack.

7. Prefix-based DNSBL Lookups

In this section, we present an optimization technique to improve the effectiveness of DNS caching whose effectiveness is critical to DNS blacklist lookups, in the presence of the new trend of spamming using botnets.

7.1. Design

Our optimization is based on the following observations:

Spam origins exhibit spatial locality. While the spam origins in a botnet appear to have distinct IP addresses, the compromised machines, i.e. bots, tend to be concentrated in blocks of /24 IP prefixes. To validate this, we measured the spatial locality of the IPs in our spam trace. The two-month spam trace collected at our spam sinkhole consists of spam originating from about 19,000 IP addresses which fall into 8,832 unique /24 prefixes. Figure 12 plots CDF of the number of blacklisted IPs belonging to these prefixes. We observe 40% of the prefixes contained more than 10 IPs blacklisted in cbl.abuseat.org [2], and about 102 of these /24 prefixes (about 3%) contained more than 100 IPs blacklisted in CBL. This suggests that there exists significant spatial locality among the blacklisted IP addresses.

Spam origins exhibit temporal locality. Ultimately the effectiveness of caching is determined by the amount of temporal locality in the sequence of queries. Using the two-month spam trace collected by our sinkhole, we measured the inter-arrival time of spam originated from the same /24 IP prefix versus that originated from the same IP. The results, plotted in Figure 13, show that the inter-arrival time in terms of IP prefix origins is shorter than in terms of individual IP origins, suggesting significant temporal locality in /24 prefixes among the spammers.

Spatial and Temporal localities of botnet activities have been exploited previously by researchers for their detection [8] and prediction of future target networks [4]. In this paper, we use these properties to design efficient mail server and DNSBL architectures. Our proposed new DNSBL system exploits the above spatial and temporal locality in spam originating IPs by aggregating the reputation of the neighboring IPs in the same IP prefix. For each DNSBL query, the DNSBL replies with blacklisting information for the queried IP as well as for its neighboring IPs in the same /25 prefix.

To implement the new DNSBL scheme under unmodified DNS, we make the DNSBL server return a bitmap of blacklist status of the neighboring IPs for each DNSBL query. We observe that a DNS query for an IPv4 address returns 32 bits information, whereas a similar query about an IPv6 address results in 128 bits information. Hence an IPv6 address can correspond to a bitmap of 128 IPs, and we construct our new DNSBL over the IPv6 address space (and hence call it DNSBLv6). For an incoming mail from an IP(v4) address x.y.z.w, a mail server queries the DNSBLv6 by issuing a DNS query for domain name 0.z.y.x.blacklistserver if the number w is less than 128 and 1.z.y.x.blacklistserver otherwise. Effectively each query returns the bitmap of IPs belonging to the same /25 prefix of the IP being queried. The mail server can then simply look up the bit corresponding to the IP being queried, and cache the bitmap for resolving subsequent queries for any IP in the same /25 prefix.

Note that in our scheme the bitmap uniquely identifies each blacklisted IP address; it does not punish any IP not blacklisted.

7.2. Evaluation

To assess the performance impact of prefix-based DNSBL lookups, we evaluated the performance of postfix using prefix-based and IP-based DNSBL lookups, by driving the

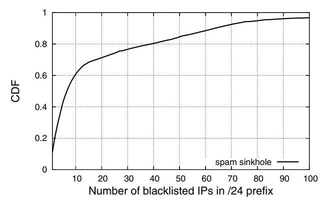

Figure 12. CDF of number of blacklisted IPs in a /24 prefix.

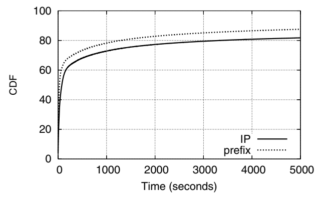

Figure 13. Interarrival times for IP addresses and /24 prefixes for spam sent to our sinkhole.

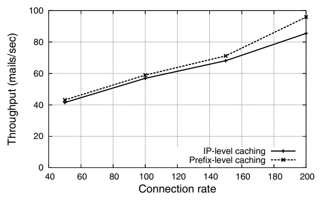

Figure 14. Throughput for DNSBL responses for prefix- and IP-based DNSBL queries vs. the connection rate.

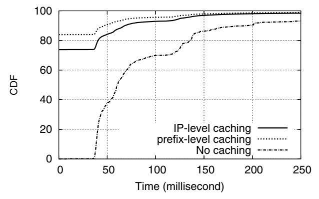

Figure 15. CDF of DNSBL lookup time under prefix- and IP-based queries for spam sinkhole workload.

client with our two-month spam trace. Since DNSBLv6 is not implemented, we emulated DNS caching and consequently the DNSBL query time for each mail received by the mail server. We used a 24-hour expiration time for the DNSBL query replies since in practice these lists are updated rather infrequently. Since the potential performance difference between the two versions comes only from the different DNSBL lookup delays incurred to the SMTP transaction processing, we used Client program 2 (open-system) [24] to drive the postfix server and configured the process limit of postfix to 1000.

Figure 14 shows that when the client-offered connection rate is low, the throughput of postfix under the two lookup schemes are largely the same. However, the performance gap starts to appear at 150 connections/sec and widens as the connection rate is further increased. In particular, postfix with prefix-based DNSBL lookups achieves 10.8% higher mail throughput at 200 connections/sec.

Figure 15 shows the CDF of the DNSBL lookup times under prefix- and IP-based queries for the spam arriving at the sinkhole. Out of the 101,692 client connections from our spam trace, prefix-based DNSBL lookups achieve a cache hit ratio of 83.9%, compared to 73.8% under IP-based DNSBL lookups. The 10.1% increase in cache hit ratio reduces the number of DNSBL lookups and contributes to 10.8% improvement of postfix's throughput. Figure 15 shows that using prefix-based queries would reduce the number of DNS queries issued from 26.22% to 16.11%, i.e., by about 39%.

8. Combined Performance Improvement

Finally, we modified postfix to incorporate all three optimizations and measured their performance. Under a workload that combines our two-month spam trace with the bounce ratio witnessed in the ECN mail server, the modified postfix achieved 40% higher mail throughput than the vanilla postfix and also cuts down DNSBL queries by 39%. Under the Univ trace, we observed a gain of 18% in mail server throughput and a 20% reduction in DNSBL queries using the prefix-based DNSBL. These numbers are lower than those from using the spam trace and can be explained by the 33% of legitimate mails. On average, a legitimate SMTP session contains fewer recipients as compared to a spam, and hence such mails lead to less gain from MFS compared to spam. Additionally, legitimate mails are shown to originate from long lasting static IPs [30], and therefore prefix-based DNSBL lookup does not provide as much gain over IP-based lookup for such mails. Looking forward, we anticipate the percentage of spam (e.g. at an organization) to continuously increase and hence the performance gain achieved by our optimizations to rise.

9. Related Work

Mail and Web Server Architecture. Section 2 already discussed the architectures of popular mail servers. Porcupine [23] is a scalable mail server that provides highly available email service using a large cluster of commodity PC's. Our work optimizes individual mail servers for the new common case workload, i.e. spam, and hence is complementary to Porcupine. The authors of [29], [30] propose to improve the performance of a mail server under load by probabilistically delaying/denying mail service to clients with spamming behavior in the past. Our solution does not delay/deny mail service to any client, which may send both bad and good mail [30]; it gains performance by optimizing the mail server to handle the common case workload more efficiently.

Despite the evolution of web server concurrency architecture [14], [33], [31], [15], mail servers have largely stayed with the process-per-connection architecture. The primary reason is security and reliability required in handling mails. Mail server performance has been a secondary issue due to the steady low-volume of mails in the past.

Single-copy file and storage systems. Several single-copy file and storage systems have been proposed that share the same objectives with MFS, to avoid duplicated disk I/Os and save on disk space. The Single Instance Storage (SIS) [1] component in Windows 2000 removes duplicated files. Since SIS detects duplicates at the file level, it will not benefit mail server applications as different mailbox files only share parts of their content. Several Content Addressable Systems (CAS)-based network file systems have also been proposed [20], [13], [28] to avoid transmitting duplicate data over the network.

A common theme of the above systems is that they do not leverage the application's knowledge of duplicated I/Os, and attempt to uncover the similarity of different files via compute-intensive content-hashing. Effectively, they trade off CPU processing for reduced disk operations. In contrast, MFS targets applications such as mail servers that have explicit knowledge of potential duplicated disk I/Os and leverages such information to avoid duplicated I/Os via a set of simple APIs. Such explicit knowledge also allows for much simpler implementation than previous single-copy file systems.

10. Conclusion

In this paper, we pointed out that modern mail servers were not originally designed with email spam in mind, and the mail spam are increasingly becoming the new "common case" workload. Using a modern mail server, postfix, we presented a case study of how the performance of its three major functional components, the concurrency architecture, the disk I/O, and DNSBL lookups, can be optimized by exploiting the new "common case" workload. We implemented the optimizations in a modified postfix and our evaluation shows that these optimizations significantly reduce the CPU, disk, and network resource consumptions, and improve the throughput of the mail server. The proposed optimizations are general and applicable to other popular mail servers such as qmail.

The battle against mail spamming is ongoing, with both spammers and spam filter providers developing increasingly sophisticated solutions, which, along with the increasing sheer volume of spam, will put increasing pressure on mail server performance. In our future work, we plan to explore other aspects of mail servers that are affected by the changing workload and develop optimization techniques that are also general to different mail server implementations.

Acknowledgment We thank Wietse Venema, author of postfix, for insightful comments on this work.

References

[1] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. Single instance storage in Windows 2000. In *Proc. of USENIX WSS*, 2000.

[2] Cbl: Composite blocking list. http://cbl.abuseat.org/.

[3] R. Clayton. Email traffic: a quantitative snapshot. In *Proc. of CEAS*, 2007.

[4] M. P. Collins, T. J. Shimeall, S. Faber, J. Jaines, R. Weaver, and M. D. Shon. Using uncleanliness to predict future botnet addresses. In *Proc. of IMC*, 2007.

[5] G. Cormack and A. Bratko. Batch and online spam filter comparison. In *Proc. of CEAS*, 2006.

[6] S. Garriss, M. Kaminsky, M. J. Freedman, B. Karp, D. Mazieres, and H. Yu. Re: Reliable email. In *Proc. of NSDI*, 2006.

[7] J. Goodman, G. V. Cormack, and D. Heckerman. Spam and the ongoing battle for the inbox. *Commun. ACM*, 50(2):24–33, 2007.

[8] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of NDSS'08*, February 2008.

[9] M. Hafiz. Security patterns and evolution of mta architecture. In *Proc. of OOPSLA*, 2005.

[10] J. Jung and E. Sit. An empirical study of spam traffic and the use of dns black lists. In *Proc. of IMC*, 2004.

[11] Rfc 822 - standard for the format of arpa internet text messages. http://www.faqs.org/rfcs/rfc822.html.

[12] Postfix vs. qmail - performance. http://www-dt.e-technik.uni-dortmund.de/ ma/postfix/vsqmail.html.

[13] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In *Proc. of ACM SOSP*, 2001.

[14] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *Proceedings of the USENIX ATC*, 1999.

[15] D. Pariag, T. Brecht, A. Harji, P. Buhr, A. Shukla, and D. R. Cheriton. Comparing the performance of web server architectures. In *Proc of EuroSys*, 2007.

[16] J. Piszcz. Benchmarking filesystems part ii. http://linuxgazette.net/122/TWDT.html#piszcz.

[17] Postfix. http://www.postfix.org.

[18] Postfix architecture. http://www.postfix.org/receiving.html.

[19] M. Prince, B. Dahl, L. Holloway, A. Keller, and E. Langheinrich. Understanding how spammers steal your email addresses: An analysis of the first six months of data from project honeypot. In *Proc. of CEAS*, 2005.

[20] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *USENIX FAST*, Monterey,CA, 2002.

[21] A. Ramachandran, D. Dagon, and N. Feamster. Can dns-based blacklists keep up with bots? In *Proc. of CEAS*, 2006.

[22] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proc. of SIGCOMM*, 2006.

[23] Y. Saito, B. N. Bershad, and H. M. Levy. Manageability, availability and performance in porcupine: A highly scalable, cluster-based mail service. In *Proc. of SOSP*, 1999.

[24] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: A cautionary tale. In *NSDI*. USENIX, 2006.

[25] K. Simpson and S. Bekman. Fingerprinting the world's mail servers. http://www.oreillynet.com/pub/a/sysadmin/2007/01/05/fingerprinting-mail-servers.html.

[26] Njabl: Frequently asked questions (faq). http://www.njabl.org/faq.html#Q12.

[27] Botnets cause significant surge in spam. http://arstechnica.com/news.ars/post/20061030-8111.html.

[28] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. Bressoud, and A. Perrig. Opportunistic use of content addressable storage for distributed file systems. In *Proc. of USENIX ATC*, 2003.

[29] R. D. Twining and et.al. Email prioritization: reducing delays on legitimate mail caused by junk mail. In *Proc. Usenix ATC*, pages 45–58, 2004.

[30] S. Venkataraman, S. Sen, O. Spatscheck, P. Haffner, and D. Song. Exploiting network structure for proactive spam mitigation. In *Proc. of 16th Usenix Security Symposium*, pages 149–166, 2007.

[31] R. von Behren, J. Condit, F. Zhou, G. Necula, and E. Brewer. Capriccio: Scalable threads for internet services. In *Proc. of SOSP*, 2003.

[32] M. Walfish, J. D. Zamfirescu, H. Balakrishnan, D. Karger, and S. Shenker. Distributed quota enforcement for spam control. In *Proc. of NSDI*, 2006.

[33] M. Welsh, D. E. Culler, and E. A. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Proc of ACM SOSP*, 2001.