

A Case for a Spam-Aware Mail Server Architecture

Abhinav Pathak
Dept. of ECE
Purdue University
West Lafayette, IN 47907
pathaka@purdue.edu

Sabyasachi Roy
Dept. of ECE
Purdue University
West Lafayette, IN 47907
roy0@purdue.edu

Y. Charlie Hu
Dept. of ECE
Purdue University
West Lafayette, IN 47907
ychu@purdue.edu

ABSTRACT

The current mail server architecture spawns a new process upon every new connection it receives. The new process deals with the handling of the mail from accepting “Helo” information till the end of the connection. While forking a new process for each separate connection has a lot of advantages in terms of security and modularity, this architecture has severe performance problems in view of increasing unsolicited emails - spams and emails with rogue connections. For example, as spammers resort to guessing email user ids, the number of emails that bounce off a mail server is increasing. For such emails, the mail server spawns a process which wastes its resources.

In this paper we propose a new architecture for mail servers, which keeps all the advantages of the process architecture has for receiving mails, but at the same time wastes little server resources in case of bounced emails/rogue connections. Essentially, the new architecture does not fork off a new process until it is certain that the mail would not get bounced. We present detailed evaluation of our architecture and show that the new architecture use server resources efficiently.

1. INTRODUCTION

Recently there has been a steep rise in the amount of unsolicited emails (spams) [13]. Such emails overwhelm users’ mailboxes, consume server resources and cause delays in mail delivery. Numerous techniques have been proposed to stop or mitigate spams. Such techniques include content-based filtering [3], IP-based blacklisting [1, 11], quota enforcement [16], relationship inference between senders and receivers [4], [18]. Although such techniques focus on reducing the impact of spams on the end-user, they assume mail servers to be over-provisioned. With the increasing trend witnessed in the amount of spams [13], it is evident that no matter how much over-provisioned the mail servers are, the need to design “smart” mail servers that can optimize their resource utilization by expending minimal resources on rogue emails is inevitable.

In this paper, we focus on a class of spams and make the case for optimizing mail servers to be *spam-aware*. Nevertheless, the same idea can be extended to other forms of illicit mails.

Figure 1 shows the percentage of the total number of

emails that arrive at a representative mail server that eventually get bounced. The data is collected at the mail server maintained by the Engineering Computer Network (ECN) at Purdue University over a period of three months starting from Dec 2006. The ECN network hosts about 20,000 mail users. We can observe that about 20 to 25% of the emails reaching the ECN mail server are bounced, i.e., there are no real users associated with the “to address” of the mail. Such a large number of bounces largely result from a well-known spamming technique: random guessing (RG) [9]. Using RG, spammers send mails to commonly used email addresses with the hope of hitting on valid ones. It is likely that in the future this percentage will continue to increase with the increase of spams, and become the “common case” among all the emails received. In fact, Figure 1 shows a slight increase in the percentage of bounces from 19% to 23% within a period of 3 months.

The server architectures used in popular mail servers delegate a new process for each SMTP connection received. Such an architecture has a myriad of advantages but particularly wastes a lot of server resources in case of bounced emails and is potentially vulnerable to DDoS. For instance, Figure 2 depicts the degradation of the postfix mail server’s *goodput performance* as the fraction of bounces received increases (details in Section 3). Using experiments on a random set of mail servers in the Internet, we show that at least 20% of the mail servers are vulnerable to very small scale DoS attacks.

Based on these observations, we propound a novel hybrid mail server architecture that improves the performance of mail servers by *early differential treatment* of legit and illicit emails. We commit server’s resources to an incoming email only after knowing the legitimacy (w.r.t to bounce/non-bounce nature) of the email.

2. CURRENT MAIL SERVER ARCHITECTURE

Current mail server [12, 8, 10] architectures have a process-based model and differ only in the degree of multiprocessing. Specifically, on receiving an SMTP connection, each of them forks a new process or uses a pre-forked process to handle the incoming connection. Such an architecture is inherently inefficient, particularly under high load. However, there are myriad advantages of using such an architecture. Below we describe the pros and cons associated with such an architecture.

2.1 Pros

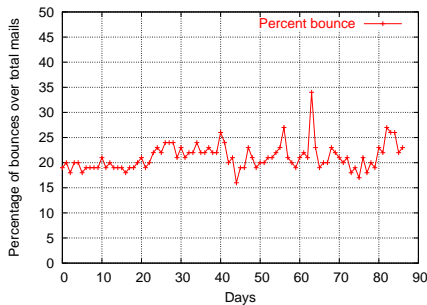


Figure 1: Percentage of the bounces w.r.t total emails at ECN mail server at Purdue University.

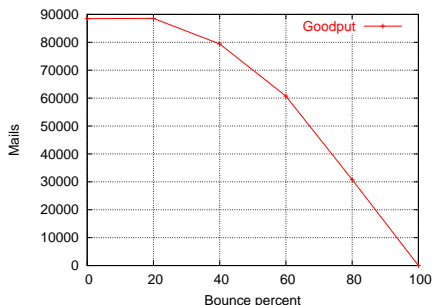


Figure 2: Performance degradation of postfix due to bounces.

The current architectural requirements of mail servers are security, reliability, robustness, performance, availability, extensibility, maintainability, testability and portability in decreasing order of importance [5]. A process-based model naturally captures the requirements of security, reliability, robustness and extensibility better than a thread/event-based model. In fact, the *sendmail* architecture that was composed of a single monolithic root process per connection was modified to spawn multiple small processes in *postfix* and *qmail*. Such a modification was necessary to get around security loopholes in *sendmail*. Specifically, using many small processes with different privileges for different tasks led to a more modular architecture. Moreover, such a design limited security breaches to the small underprivileged processes and obviated the need for subtle security checks thereby reducing the risk of coding lapses. Postfix optimized the multi process architecture by creating a process pool apriori, thus getting rid of the process creation and termination overhead.

2.2 Cons

Web servers have already switched to multithreaded architecture to satisfy the ever growing performance demands [6, 14, 2, 15]. However, due to the less strict latency and response time requirements in case of SMTP traffic, a process-based architecture has sufficed for mail servers till now. But, the downsides in terms performance and availability are getting pronounced with the increase in email traffic. The poor performance of this architecture in turn makes it vulnerable to DoS attacks. To prevent such attacks, current mail servers put a source IP based limit on the number and rate of connections served at a time. But such limits are configurable parameters and finding the correct parameter value is a challenge in itself. Also, using the same parameter value

for all source IPs may not be appropriate. For instance, a large amount of SMTP traffic is likely between the mail servers of Yahoo and Google. In such a case a predefined limit is not desirable. All such concerns justify the need to rethink the mail server architecture.

3. PERFORMANCE PROBLEM

We consider the problem of performance penalty that accompanies a process-based mail server architecture from two perspectives. First, we estimate the gravity of the problem by quantifying the performance degradation due to a process-based architecture such as process creation and context switching overhead. Second, we show that the above problem is widespread in today’s Internet by using data collected from our Internet experiments.

3.1 How serious is the problem?

For the purpose of demonstrating the seriousness of the problem, we focus on a case where a mail server receives a large number of bounces. By allocating processes to handle such bounces, mail servers often unnecessarily waste resources. We conducted our experiments on postfix [8] as a representative MTA. We hosted our mail server on a machine with Intel Celeron 3.06GHz processor, 256 MB of RAM and the client sending mails was running on a similar machine. A large number of mails were sent continuously over a period of 15 minutes. Postfix was configured to handle only 100 (by default) connections at any time. The client generating mails always maintained at least 200 connections to keep the MTA busy. The client sent a mix of mails to valid and invalid users (bounces). The proportion of the two types of mails was varied. Figure 2 plots the goodput of the postfix MTA, i.e., number of valid mails that postfix could accept during the 15-minute period. We can observe that as the percentage of bounces increases, the goodput decreases. After the mix reaches a point where more than 50% of mails are bounces, the goodput drops alarmingly. Ideally, the processing of bounces should take little time. However, due to the context switches involved among processes processing bounces, a large portion of the time is wasted.

3.2 How prevalent is the problem?

Because of the high overhead associated with the process-based model, most of the mail servers have a configurable limit on the number of processes they can fork for accepting incoming connections. When the upper limit on the number of processes to be forked is reached, mail servers either drop down the new connections or send back a message “421 Too many connections” and close the connection. A practical scenario when the above can happen is when a mail server is hit by spammers, leading to a large number of connections being served for the emails from the spammer which may eventually get bounced and thus, leaving little room for valid emails to connect to the mail server.

To demonstrate the prevalence of the performance problem, we did a simple experiment on a set of randomly selected mail servers currently running in the Internet. We randomly picked a list of one thousand mail servers in the Internet from the “whois” [17] database. For each mail server in the list, we tried to achieve the maximum possible number of simultaneous SMTP connections to it as follows. We ran an SMTP client on one machine that tries to create up to 1000 connections to that mail server for 20 seconds. At

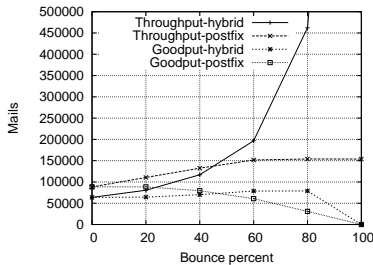


Figure 3: Performance of the current mail server architecture (postfix) and our model.

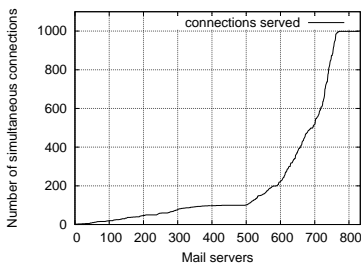


Figure 4: Limit on number of processes forked by mail servers.

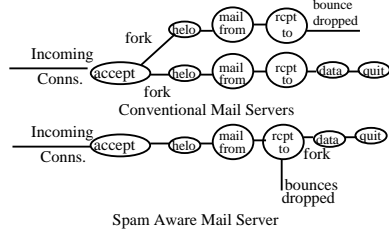


Figure 5: The conventional multi-process and the proposed hybrid architectures.

the same time, we ran a second SMTP client from a second machine (and hence different IP) that tries to create 5 connections to the same mail server. For each mail server that did not accept connections from the second machine, we concluded that it was choked by connections from the first machine. For such mail servers, we recorded the number of connections that were successfully served before getting choked.

Figure 4 plots a distribution of the number of connections that were successfully served by the mail servers. We see that out of about 800 mail servers plotted, about 300 choked under 100 connections. About 200 of the mail servers choked at 100 connections (300 to 500 on x - axis). This shows that a lot of SMTP servers deployed use the default setting of 100 (common for postfix). We also see that a few of them were able to serve all the 1000 connections. Importantly, at least 20% of the mail servers were vulnerable under a spam attack from the spammers which eventually would lead to legitimate connections suffering.

4. A NEW ARCHITECTURAL MODEL FOR SPAM-AWARE MAIL SERVERS

In Section 3.1 we explained how the number of bounces seen by mail servers are expected to increase and how such an increase would degrade the performance of traditional mail servers. Citing such imminent problems, we make a case for a new mail server architecture. The basic idea behind a new architecture is derived from the observation that as the percentage of bounces increases, mail servers spend increasingly more resources processing illicit mails. The current mail servers delegate the responsibility of each incoming connection to a new process. This leads to a significant amount of *pure* CPU overhead for processing invalid mails such as bounces. Our solution aims to reduce this overhead by *delaying* the event of such a delegation. In particular, a new email is not delegated to a new process until the validity of the “to address” is determined. Until that time the mail server keeps all such new and undecided connections in a list of sockets (S). Any event happening due to data arriving at a socket in S is caught by waiting on them in a “select/poll” loop. Thus, the overhead of process creation and/or context switching for bounces is avoided by using a hybrid of an event-based architecture and a process-based architecture. The event-based architecture is used until the validity of the “to address” is determined, after which the process-based architecture is invoked as described in Figure 5. Such a hybrid architecture keeps the pros and removes the cons,

as discussed in Section 2, of the process architecture.

We implemented a simple prototype of the above architecture by designing a wrapper [7]. The wrapper is a front-end of the mail server running on port 25 of a public IP address. The wrapper has a select-based architecture that accepts connections and completes the entire SMTP transaction. The wrapper determines if the “to address” is valid and forwards it to the traditional mail server, the “backend”, which in our case was postfix running on another machine connected to the same switch. This approximates our intended architecture. In this way, postfix never sees a bounced email and hence does not fork out a new process for it. However, note that this architecture incurs extra overhead due to the extra level of indirection every mail goes through (once to the wrapper then to the final mail server).

Figure 3 compares the goodput (Goodput-hybrid) of our architecture to the goodput of vanilla postfix (Goodput-postfix) as the ratio of the bounces in the mix of emails sent is increased. We observe that the goodput of our architecture is less than that of the original postfix MTA initially (up to 50% mark). We attribute this to the extra overhead our scheme incurs (store and forward) and we consider it as a future work to investigate and mitigate this overhead. After the mix of bounces is increased above 50%, the number of valid emails received by our scheme outperforms the postfix. This is because the backend never sees an email that would get bounced and hence does not waste resources on it. We also plot the total number of emails (bounced + non-bounced) that were handled for the case of our architecture (throughput-hybrid) and for postfix (throughput-postfix). We see that our architecture handles more emails compared to the postfix after the mix contains more than 50% bounces.

5. CONCLUSION AND FUTURE WORK

In this paper, we demonstrated that a significant and ever increasing portion of current mail servers processing power is wasted due to processing of invalid mails. We also demonstrated that despite several advantages, current mail server architectures are vulnerable to DDOS attacks owing to their process-based model. Hence, we proposed a hybrid mail server architecture that combines the strengths of event-based (efficiency) and process-based (security) architectures. Using a prototype implementation, we demonstrate that our architecture performs better in comparison to the state-of-the-art mail server architecture such as postfix, as the percent of bounces increases. As our future work,

we are developing an integrated implementation of the hybrid architecture to eliminate the overhead from the wrapper implementation.

6. REFERENCES

- [1] N. F. A. Ramachandran, D. Dagon. Can dns-based blacklists keep up with bots? In *Proc. of CEAS*, 2006.
- [2] B. Bradel and C. Drula. A study of the thread and event concurrency models for web servers. 2003.
- [3] G. Cormack and A. Bratko. Batch and online spam filter comparison. In *Proc. of CEAS*, 2006.
- [4] S. Garriss, M. Kaminsky, M. J. Freedman, B. Karp, D. Mazieres, and H. Yu. Re: Reliable email. In *Proc. of NSDI*, 2006.
- [5] M. Hafiz. Security patterns and evolution of mta architecture. In *Proc. of OOPSLA*, 2005.
- [6] V. S. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *In Proceedings of the USENIX 1999 Annual Technical Conference*, 1999.
- [7] K. Park and V. S. Pai. Connection conditioning: Architecture-independent support for simple, robust servers. In *Proc. of NSDI*, 2006.
- [8] Postfix. <http://www.postfix.org>.
- [9] 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.
- [10] Qmail. <http://www.qmail.org>.
- [11] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proc. of SIGCOMM*, 2006.
- [12] Sendmail. <http://www.sendmail.org>.
- [13] Spammation. <http://www.spammation.info/stats/>.
- [14] R. von Behren, J. Condit, and E. Brewer. Why events are a bad idea (for high-concurrency servers). In *Proc. of HotOS*, 2003.
- [15] R. von Behren, J. Condit, F. Zhou, G. Necula, and E. Brewer. Capriccio: Scalable threads for internet services. In *Proc. of SOSP*, 2003.
- [16] M. Walfish, J. D. Zamfirescu, H. Balakrishnan, D. Karger, and S. Shenker. Distributed quota enforcement for spam control. In *Proc. of NSDI*, 2006.
- [17] Whois. <http://www.whois.ws>.
- [18] J. Yeh and A. Harnly. Email thread reassembly using similarity matching. In *Proc. of CEAS*, 2006.