WORKLOAD CHARACTERIZATION OF SPAM EMAIL FILTERING SYSTEMS

Yan Luo

[1]Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Massachusetts, USA
yan_luo@uml.edu

ABSTRACT

Email systems have suffered from degraded quality of service due to rampant spam, phishing and fraudulent emails. This is partly because the classification speed of email filtering systems falls far behind the requirements of email service providers. We are motivated to address this issue from the perspective of computer architecture support. In this paper, as the first step towards novel architecture designs, we present extensive performance data collected from measurement and profiling experiments using representative email filtering systems including CRM114, DSPAM, SpamAssassin and TREC Bogofilter. We provide detailed analysis of the time consuming functions in the systems under study. We also show how the processor architecture parameters affect the performance of these email filters through simulation experiments.

KEYWORDS

Workload Characterization, Email Filtering, Anti-Spam

1. INTRODUCTION

Email and instant messaging systems have suffered from degraded quality of service due to spam, phishing and fraudulent emails and messages. A surprising fact is that 9 out of 10 emails are spam [25]. Gartner estimated that 3.5 million Americans would give up sensitive information to phishers and the total financial losses amounted to $2.8 billion in 2006 [22]. Widely spreading spam and phishing emails have drawn attention from both the academia and industry. There is a plethora of research on spam detection, filtering and elimination [1], [2], [13], [12], [18], [33], [21], [10]. Some anti-spam appliances have been introduced to the market [32], [28]. Moreover, fighting spam has gone beyond the technical regime and had its social, legal and economical impacts on people's life [7], [30].

However, the growth of spam messages remains rampant. There exists a strong call to design high-performance email filtering systems. A careful analysis of spam shows that the requirements of an efficient filtering system include: (1) accuracy: low false positive and low false negative filters are desirable; (2) self-evolving capability: the systems should be able to adapt themselves to new spam; and (3) high-performance: the detection of spam incur complex operations including regular expression matching and statistical computation, which need to be completed quickly especially in large email or messaging systems.

Most of the existing research focuses on the design of protocols, authentication methods, neural network based self-learning and statistical filtering. In contrast, we address the spam filtering issues from another perspective - improving the filtering speed through computer architecture support. We are motivated by the inadequate classification speed of current anti-spam systems. Data have shown that the classification speeds of current spam filters fall far behind the growth of messages handled by servers. In this paper, we analyze the workload of spam filters and identify the critical operations in spam filtering. We intend to provide insight to the critical workloads that must be accelerated by novel architecture designs.

The contribution of this paper is as follows.

- We first qualitatively compare and analyze representative open source spam filtering systems including CRM114 [1], DSPAM [2], TREC [11], SpamAssassin [4], etc.
- We then conduct extensive measurement and profiling experiments on the above email filtering systems using realistic email samples. We collect statistical data on the frequently executed time-consuming functions of these filtering systems.
- We further simulate the CRM114 in a processor architecture simulator, SimpleScalar [8], to evaluate how the architecture parameters affect its performance.

The paper is organized as follows. Section II introduces the background and motivation of this research. Section III describes the representative email filtering systems and their main operations. The methodology of the experiments is also described in Section III. Section IV and V show the performance data we collect from the extensive measurement and profiling experiments, respectively. Simulation results are reported in Section VI. Finally the paper is concluded in Section VII.

2. Background and Motivation

The following formatting rules must be followed strictly. This (.doc) document may be used as a template for papers prepared using Microsoft Word. Papers not conforming to these requirements may not be published in the conference proceedings.

2.1. Related Work

Spam flooding has outpaced the growth of legitimate emails. Data have shown that spam now represents nearly 93 percent of all email, and over 2006, the number of spam messages grows by 147 percent, up 73 percent in the last quarter [25]. Graham-Cumming maintains an archive of different categories of spam emails [17], and such a list keeps expanding constantly. It is also evident that approaches taken by spammers become more versatile and spam emails are harder for filters to capture. Recently, a number of actions are taken to address the spam issue. We briefly summarize them as follows.

Email Authentication Standards

Extensions to SMTP protocol introduce new methods to enhance security. Identified Internet Mail [12] determines if the sender of the message has authorization (from the administration of the domain) for the use of an email address by associating the signature with the message itself. The signature is calculated with cryptographic hash such as SHA1 and encapsulated in the message header. DomainKeys Identified Mail (DKIM) [18] defines a domain-level authentication framework for emails using public-key cryptography to generate the signature of a domain. The Sender Policy Framework (SPF) [33] explicitly authorizes the hosts that are allowed to use its domain name by associating SPF entries to the domain names as Resource Record in Domain Naming Systems (DNS). Such SPF entries can be multiple and use wildcard. The Sender ID Framework [21], working together with SFP, addresses the problem of spoofing and phishing by verifying the domain name or IP address from which an e-mail is sent against a registered list of servers that the domain owner has authorized to send e-mails.

Software Email Filters

The continuous anti-spam efforts have resulted in several popular software tools. CRM114 [1] is an open source software tool that supports a programming language to write filters. CRM114 is primarily based on regular expressions [5]. Its criteria for classification of data can be a set of methods, including regular expressions, approximate regular expressions, a Hidden Markov Model [26], etc. DSPAM [2] is another open-source content based spam filter designed for large enterprise systems. Its core engine, libdspam library, uses various filtering algorithms such as Concept Identification [35], Neural Networking and Bayesian Noise Reduction[34]. POPFile [3] classifies emails into spam or not spam using a naive Bayes algorithm.

Statistical Filtering Algorithms

Most anti-spam systems use statistical filtering algorithms, which determine the accuracy. There is an increasing number of statistics algorithms used in current systems. Bayesian algorithms are a category of algorithms that are widely used in spam filters [24], [27], [16]. The representative one, Naive Bayes classifier selects the most likely classification Vnb given the attribute values a1, a2, ...an., using Vnb = $\text{argmax}_{vj\in V}$ P(vj)ΠP(ai|vj). P(ai|vj) is practically estimated with m-estimates as P(ai|vj) = $(n_c+mp)/(n+m)$, where n is the number of training examples for which v = vj , n_c is the number of examples for which v = vj and a = ai, p is a priori estimate for P(ai|vj), and m is the equivalent sample size. Other popular statistical filters include Markov Chain [26], Fisher-Robinsons' Inverse Chi Square[36], etc.

Image Processing Algorithms

As the spammers have begun to use image based spam emails, the filtering process relies on complex image processing algorithms to extract useful information from obscured images similar to CAPTCHA [37]. For example, recent work in [38] first computes the perimetric complexity of *p* °— *q* image cells of a suspicious image and then computes three quantitative metrics on the broken characters and their interference level with background noise.

2.2. Common Steps in Fighting Spam

There is a growing set of software tools, hardware appliances and statistical filtering algorithms for fighting spam [1], [2], [24], [3], [32], [28]. Although these systems differ in deployment, formats and algorithms, they follow very similar processing procedure as depicted in Figure 1.

First, SMTP protocol stack on mail servers handles network packets and parses them into email headers and bodies. An unknown email will then go through several steps before it is classified as a spam or a legitimate email. Tokenization breaks the email, including header and body, into interesting tokens (keywords) to prepare for further processing. A subset of the tokens, e.g. domain names or URLs, are then checked against whitelists and blacklists for immediate acceptance or rejection (explained later). The majority of the tokens are counted to calculate the statistical metrics. Then various statistical filtering algorithms, such as Bayesian and Chi-Square, are applied to compute the probability of the email under investigation being a spam. Finally a decision is made and the email is classified as either a legitimate email or a spam. The essential operations and techniques within this classification procedure are summarized as follows.

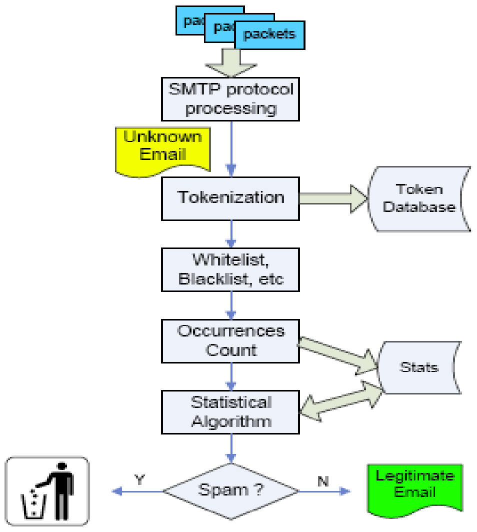


Figure 1. Common Work Flow of Spam Filtering

• SMTP Protocol Processing

Mail servers communicate interactively with mail transfer agents through SMTP that is primarily based on TCP/IP protocol. The header and body of an email are received by the server after a series of SMTP commands and replies. During such a session, the server needs to process various security extensions such as SHA1 decryption, and handle certain message formats such as MIME [15], in addition to the underlying TCP/IP protocol processing.

• Tokenization

Tokenizer or parser extracts interesting and significant terms from both the header and body of emails. This procedure is called tokenization [36]. The tokenizer shall understand HTML tags as well as different URL encoding methods such as Base64 [19] that are used in various standards [20], [15]. The tokenizer removes whitespace characters but allows a small set of them to appear within terms (e.g. '.', '+', '-', ' ', '$'). Pure numbers, IP addresses and money amounts are also differentiated.

• Checking Against Blacklist, Whitelist and Graylist

Blacklist, whitelist and graylist are checked against to classify unknown emails before statistical filters are leveraged. A whitelist is a list of accepted items, confirming that the item being analyzed is acceptable. A blacklist is, on the contrary, a list of entities that are being denied a particular privilege, or marked as "spam". Entries in a graylist cause a mail transfer agent to "temporarily reject" an email from a sender on the list and wait for a retry from the originating server, which a spammer usually does not do. These lists are created from the past classification results. The interested fields, including From, ReturnPath and URL, are extracted from an incoming email and checked against the lists.

• Hashing Operations

Hashing functions are used to find the attributes and count the occurrences of a token. The number of tokens and their attributes keep expanding, which results in the growth of token databases and hash tables. Hashing functions are widely used to speed up the searching of a token. They are usually implemented on general purpose processors using linked lists.

2.3. Challenges

Most popular anti-spam systems focus on the accuracy of the filter, in terms of false positive and false negative rates. However, the classification speed, in units of messages per second, is as critical as accuracy to the effectiveness of the system, but often overlooked. Existing email servers are already heavily loaded and subject to denial of service attacks. When spam filtering becomes another significant workload for email servers to carry out, the servers can be easily overloaded and become more vulnerable to attacks.

Current anti-spam applications have not met the performance requirements. The majority of the current antispam tools [1], [2], [3], [11] are software running on general purpose processor based platforms. Their performance suffers from both the overhead of protocol processing in operating systems and the inefficiency of regular expression matching and hashing operations on general purpose CPUs. We have measured the performance of a representative anti-spam tool, TREC Spam Filter Kit with bogofilter [11]. The experiment results show that classifying 6034 emails on a 2.2GHz Pentium 4 takes 20 minutes and 18 seconds, i.e., less than 5 emails per second, shown as the second right-most bar in Fig. 2. Another spam filtering software, DSPAM, reported similar performance (0.25 second per message on average, or 4 emails per second as shown as the rightmost bar in Fig. 2) [2]. On the other hand, various sources [29], [6], [23], [9] show that the emails processed by service providers such as Hotmail, America Online and Microsoft are several magnitude more than the capacity of current anti-spam systems, as depicted in Figure 2. The left four bars depict the number of emails processed by the email service providers. Both Hotmail and AOL process over 10,000 emails per second. Instant Messengers such as AOL, Yahoo and MSN each process over 10,000 messages per second (shown as the "IM" bar.) An enterprise level email server (e.g. Microsoft) needs to handle about 100 emails per second. Such performance gap is not acceptable for enterprise level or ISP email systems where both the number and size of emails keep increasing.

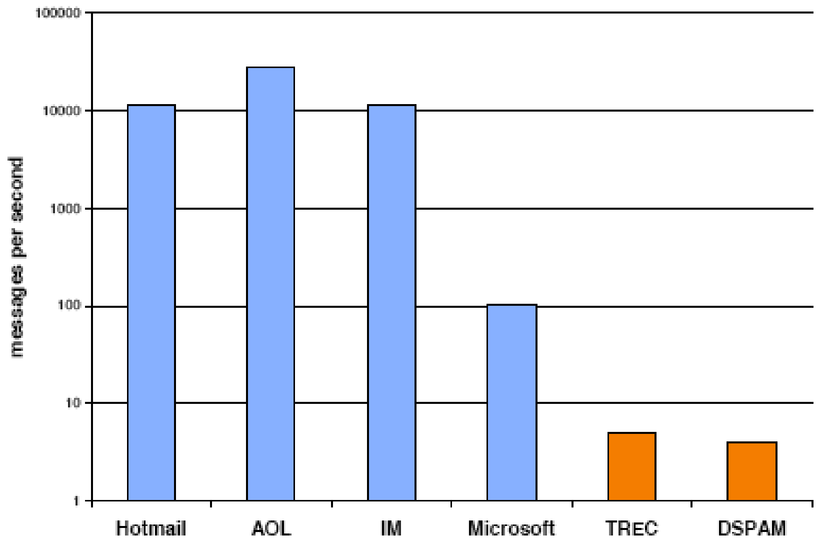


Figure 2. ISP Email/Message Load vs Performance of Current Anti-Spam Systems (IM: instant messaging)

Tokenization takes considerable time in spam filtering. Tokenization depends heavily on regular expression techniques. However, regular expression matching is a memory access intensive task. We profiled the CRM114 spam filter system and found that over 51% of the processing time is on pattern matching functions (e.g. tre_tnfa_run_parallel()). Whitelist (and other lists) checking is another performance bottleneck. Various whitelists and blacklists are currently used and they contain an increasing number of entries . For example, the URL blacklist [31] contains 2,208,371 entries as of Dec 19, 2006, and this list keeps expanding as new domains and URLs are registered. The entries in a list can be either explicit URLs, domain names or regular expressions, which make the matching a non-trivial task.

Hashing, an essential operation in finding attributes of tokens and updating statistics data, incurs large overhead as well. Software-based hashing on general purpose CPUs is popular but not efficient. A common implementation of hashing uses linked lists to store hash nodes and resolve collisions. The CPU computes hash value using a large number of arithmetic or logic instructions. Such prolonged instruction execution and dynamic memory allocation/deallocation degrade the performance of hash operations. For instance, profiling results show that hash functions in TREC kit take significant portion of time (30%) in accessing message counters [11].

Protocol stack processing on email servers requires a significant amount processing power from processors. Most email protocols (e.g. SMTP) and popular messaging protocols (e.g. AOL and MSN) are based on TCP/IP protocol stack. The processing overhead of TCP/IP is known to be deteriorating on general purpose processors. The rule of thumb is that 1GHz processing power is needed to sustain 1Gbps TCP/IP network traffic [14]. With the processing of SMTP protocol and its security extensions, email servers are not able to filter spam and keep up with the growth of email volume. As a result, email servers are vulnerable to denial-of-service attacks when they need to handle both message forwarding and spam classification. The same risk is present in instant messaging servers.

Therefore, it is indispensable to seek novel architecture designs to improve the performance of spam filter. The first step towards new design is to thoroughly analyze the workload of spam filtering systems. We choose to conduct measurement, profiling and simulation experiments on representative spam filter tools.

3. Email Filtering Systems Under Study and Experiment Methodologies

We focus our research on email filtering systems that run on or are tightly coupled with email servers, although there are filter plug-ins embedded in email clients. The filters at end systems require end users to understand email filtering policies and update their filter rules frequently. This imposes management overhead and is often ineffective for users with inadequate knowledge on email filters. On the other hand, centralized email filters ease the management and reduce the cost of supporting general users. Therefore we focus on server-side centralized email filtering systems in our study.

3.1. Email Filtering Systems Under Study

We summarize the features of four representative open source email filters in Table 1 including the language in which the source code is written, whether the filter software includes training functions and sample emails, and which software library the filter relies on. We briefly introduce these email filter applications as follows.

CRM114 is a filtering system that can examine emails, system log files, user data files and other data streams. It can scan or alter the data stream through user defined scripts. It supports various statistical algorithms such as Hidden Markov Model, Orthogonal Sparse Bigrams, etc. CRM114 is written in C language and has been ported to Linux, BSD, MacOS and Windows. It relies on TRE [5], the POSIX compliant regular expression matching library for scanning and tokenization. CRM114 can be used at both server and client side.

DSPAM is an open-source content-based spam filter that is designed to be used at server side. The software is written in C and supports different mathematical paradigms including Bayes, Chi-Square, Geometric, and Markovian Discrimination. DSPAM does not rely on heuristic rules or regular expressions.

SpamAssassin is part of the Apache Software Foundation projects. It attempts to identify spam through text analysis, Bayesian filtering, DNS blocklists, and collaborative filtering databases. SpamAssassin is written in Perl, which well supports regular expressions. It provides APIs that are can be used on a wide variety of email systems including procmail, sendmail, Postfix, qmail, etc.

TREC spam filter evaluation kit is a tool to assess the effectiveness of spam filters. The goal is to increase the availability of appropriate evaluation techniques for use by industry and academia [11]. The default filter included in TREC kit is “bogofilter”. The TREC kit is written in C.

Table 1. Representative Email Filtering Systems

Name	**Source Code Language**	**Trainers included**	**Email Samples Included**	**Software Libraries Used**
CRM114	C	Yes	No	TRE
Bogofilter	C	Yes	No	Berkeley DB
SpamAssassin	Perl	Yes	Yes	Perl reg-ex
Dspam	C	Yes	No	libdspam

3.2. Spam and Ham Email Samples

Spam is known as the “unsolicited commercial emails” while ham refers to the emails that users expect to receive. Some of the filter systems include archives of spam and ham emails for the purpose of training the system initially. For example, SpamAssassin contains an archive of 4150 ham and 1898 spam emails. Such emails are collected from various sources and reflect realistic ones received by general users. They are textual and without any attachments. These emails serve well for training spam filters.

We utilize the emails from SpamAssassin as the sample inputs to be fed into the filtering systems. There are three categories of these samples, namely “Easy Ham”, “Hard Ham” and “Spam”. These emails are categorized by the characteristics of their content. Easy Ham emails are legitimate ones one receives from expected senders. These emails often contain plain text only. Hard Ham emails are also expected legitimate emails, however, they contain a fairly large number of HTML tags and URL links which make it harder for the email filters to recognize them as good emails. Spam emails are those unsolicited ones. Table 2 lists the characteristics of these emails. Furthermore, for each category, we construct synthetic email samples of a range of sizes (e.g. 8KB, 32KB, up to 2MB) by concatenating multiple randomly selected emails so that we can evaluate how the size of an email affects the filtering performance. We use these original and synthetic email samples in our following experiments.

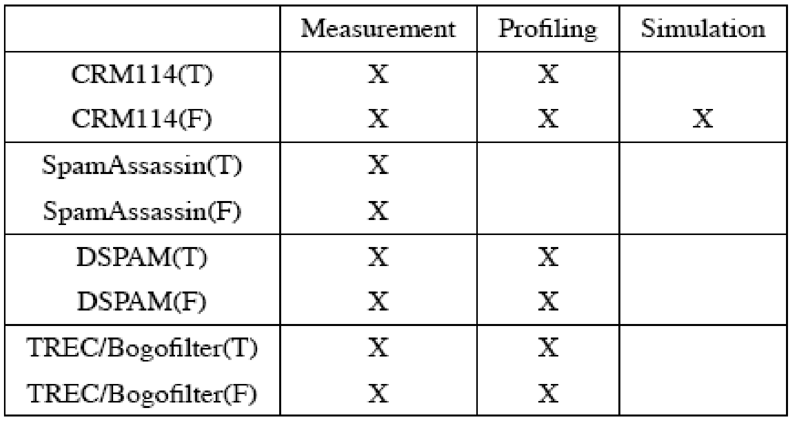
3.3. Experiment Methodology

Table 2. Email Samples

Sample Name	**Total Emails**	**Total Size (Mbytes)**	**Average Size (bytes/email)**
Spam	1898	11.9	6270
Easy Ham	3900	14.1	3615
Hard Ham	250	5.5	22000

In this paper, we conduct three categories of experiments: measurement, profiling and simulation as listed in Table 3. For measurement experiments, we set up a testbed of a Pentium 4 2.6GHz based Linux box running Fedora Core 5. The Linux box has 1GB DDR DRAM memory. We install the software packages of the email filters under test as user level applications. For profiling study, we compile the C language based filters with "-pg" option for gcc such that "gprof" can be used to collect profiling data. We use "gprof -s" to combine the data from several runs for avoiding statistical inaccuracy. Finally we use SimpleScalar simulator to investigate how the architecture parameters affect the performance of CRM114.

Table 3. Experiments Conducted (T: Training, F: Filtering)



	Measurement	Profiling	Simulation
CRM114(T)	X	X	
CRM114(F)	X	X	X
SpamAssassin(T)	X		
SpamAssassin(F)	X		
DSPAM(T)	X	X	
DSPAM(F)	X	X	
TREC/Bogofilter(T)	X	X	
TREC/Bogofilter(F)	X	X	

Each application has both training process and filtering process. Most email filtering systems rely on the past knowledge of known emails, either spam or ham. As the initial step, known emails are used to train the filters before working on classification of incoming unknown emails. Such training is performed frequently either explicitly by issuing training commands or implicitly after receiving and classifying new emails. After the training, the filters can start classification of incoming emails.

4. MEASUREMENT RESULTS

4.1. Training of Filters

Fig. 3 shows the training of the four email filters under test. The training input is a set of emails (4150 ham emails and 1898 spam emails.) SpamAssassin consumes over 500 seconds in the training process while CRM114 and DSPAM consume as little as 82.5 and 103.6 seconds, respectively. Bogofilter has the fastest training procedure taking only 10.8 seconds. On average, the training process takes about 179.3 seconds.

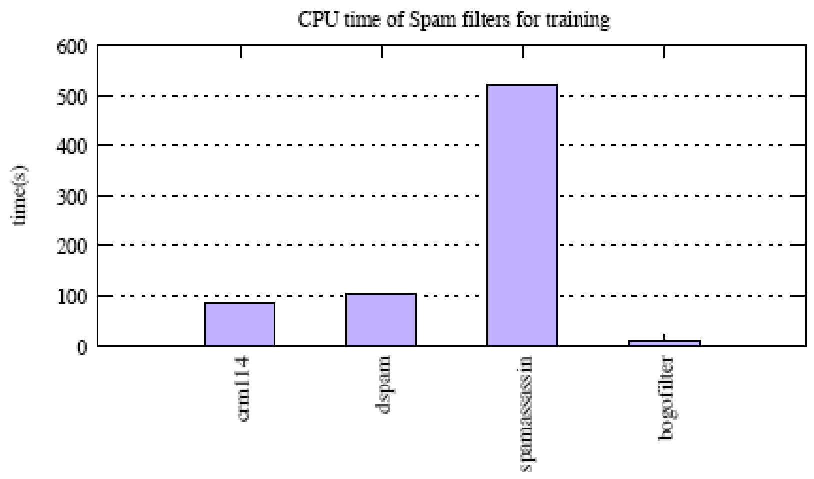


Figure 3. CPU Time of Email Filters Running in Training Mode.

4.2. Filtering Email Samples

Fig. 4 shows the performance of four filters on a set of Easy Ham email samples. The x axis of the figure is the size of Easy Ham email samples and y axis is the CPU time spent by the filters under test. The CPU time increases as the size of email samples increases, which is intuitive. CRM114, DSPAM and Bogofilter show scalable performance: the increase of CPU time is proportional to the size of samples. SpamAssassin consumes one order of magnitude more time than the other three filters. The contributing factor is that SpamAssassin is written in Perl, which inherently has much larger runtime overhead.

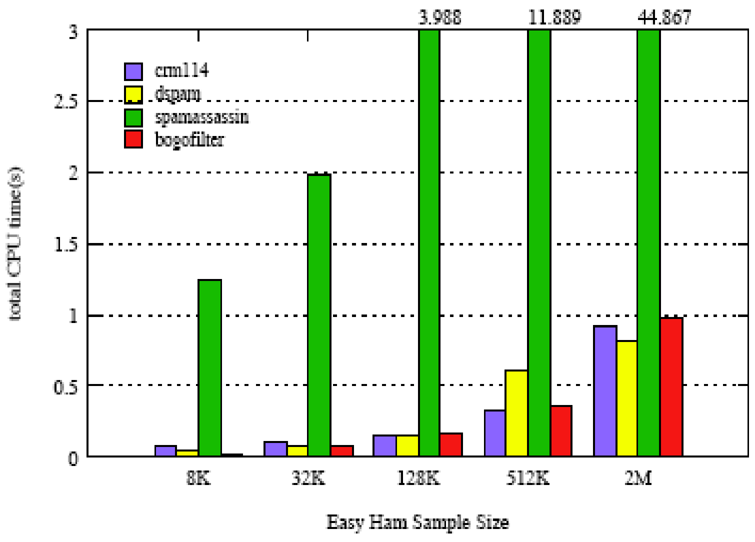


Figure 4. CPU time of filters on Easy Ham email samples

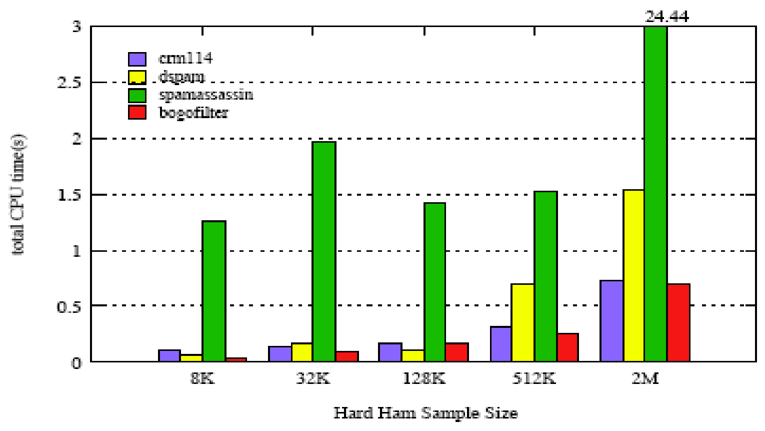


Figure 5. CPU time of filters on Hard Ham email samples

Fig. 5 shows the performance of four filters on a set of "Hard Ham" email samples. The CPU time of CRM114, DSPAM and Bogofilter increases with email size. SpamAssassin differs in the cases when the email size is 128KB and 512KB: the increase in CPU time is not significant. This is due to the runtime overhead of Perl libraries. Fig. 6 shows the performance of four filters on a set of "Spam" email samples. All the four filters show similar performance to that on Easy Ham: the time spent is proportional to the size of email samples.

Comparing the CPU time spent on the Easy Ham, Hard Ham and Spam Emails with the same size, there is no universal conclusion can be drawn from the measurement results: each filter behaves differently. For example, DSPAM spends more time on 2MB Hard Ham than 2MB Easy Ham while Bogofilter is the opposite. Hard ham does not necessarily incur more processing time than easy spam because hard ham is named after the probability of false positive.

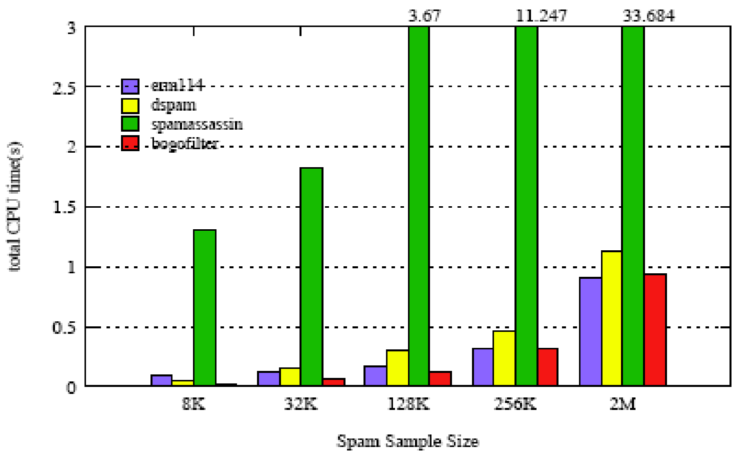


Figure 6. CPU time of filters on Spam email samples

5. PROFILING RESULTS

We present in this subsection the profiling results obtained with "gprof". The three filtering systems under test, CRM114, DSPAM and Bogofilter, are compiled with "-pg" options for gcc. SpamAssassin is written in Perl thus we exclude it from the profiling experiments.

5.1. CRM114

Training

Fig. 7 shows the profiling result of CRM114 during the training process. The x axis depicts the functions sorted by the CPU time and y axis is the percentage of total CPU time that a function consumes. The input of the training is a set of ham and spam emails. The most time consuming function is crm_expr_osb_bayes_classify() that takes 37.5% of the total time. Following that is tre_tnfa_run_parallel(), taking 22.2% of the total. Then crm_vht_lookup() consumes 12.1% of the time. The operations of these three functions are as follows.

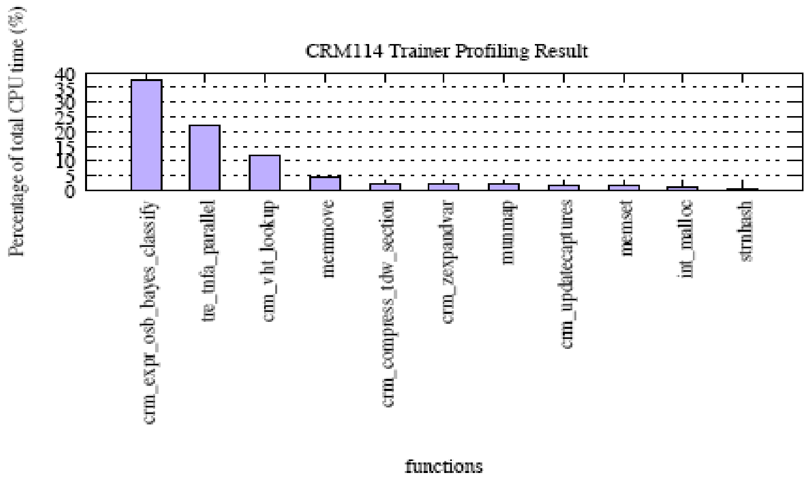


Figure 7. Profiling results of CRM114 in training mode

Tre_tnfa_run_parallel() function is the key function to match regular expression patterns. Such matching procedure is based on a Nondeterministic Finite Automata (NFA) structure. First, the function allocates a chunk of memory that is used for matching the current input stream. Such a memory space is used to store the NFA states which contain tag bits for the current state and next states. Next, the function starts to search the first matching character in the input string, from which the complete NFA matching process will begin. The main loop in the function migrates from one NFA state to the other one based on the input character and the tag bits of the current state. Finally, the function reports a match or not match based on the ending NFA state and the tag bits it reaches.

Crm_expr_osb_bayes_classify() is the main Bayesian classification function with 1069 lines of C code. This function serves as the “probabilistic evaluator” that computes the probability of an email being spam. It relies on hash functions that are stored in at least two offline files, one for spam (spam.css) and one for nonspam (nonspam.css). Before the first filtering procedure, the hash functions are constructed from the files and mapped into memory. This mapping processing involves validation of these files, hash function normalization and calculation of compensation correction. After the hash function is constructed, input stream is searched for any matched regular expression patterns. Such search is to extract interesting tokens from the input email. Next, the tokens are fed into the hash functions to examine the “hits” to the features of spam and ham. That is to say, if the token has a feature of spam, the corresponding counter will be incremented and the probability will be calculated with Baysian statistical formula subsequently, where some floating point computation is needed.

Crm_vht_lookup() is responsible for looking up a variable in the hash table. CRM114 has its internal script language and CRM114 actually interprets and executes CRM scripts. The scripts may contain any number of variables as long as they conform to the syntax of CRM114. The function manages a hash table where a variable can be looked up or a new hash slots can be allocated if it does not exist in the table.

Filtering

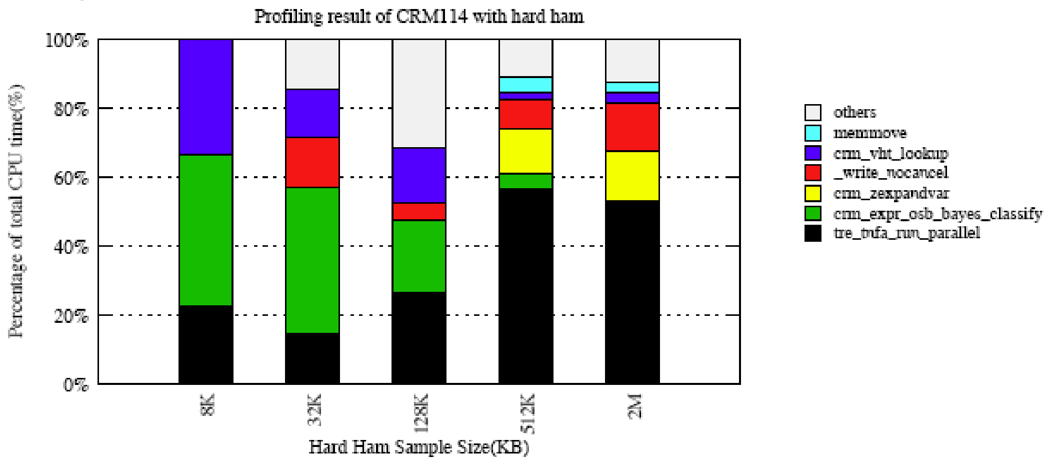


Figure 8. Profiling results of CRM114 using Hard Ham samples

Fig. 8 to 9 depict the profiling results of CRM114 on “Hard Ham” and “Spam” email samples. The three most time consuming functions are the same as the ones in training process. In addition, function crm_zexpandvar() consistently consumes over 15% of the CPU time when the size of an email is over 512KB. Analysis of the CRM114 source code reveals that crm_zexpandvar() takes care of the variable expansion operations supported by CRM114 script language.

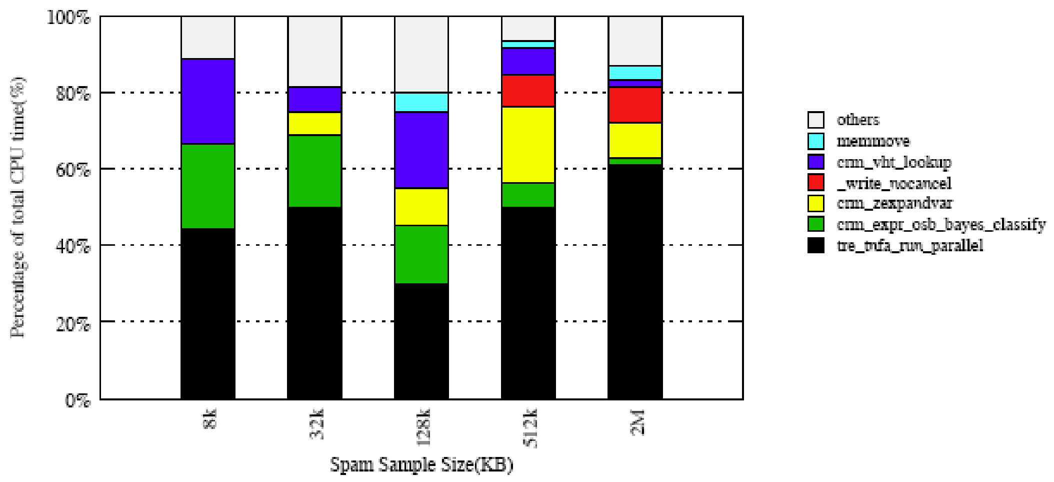


Figure 9. Profiling results of CRM114 using Spam samples

5.2. Bogofilter

Training

Fig. 10 shows the profiling result of Bogofilter during the training process with the percentage of total CPU time that a function consumes. The reason of having two bars is because Bogofilter use has two training phases that use spam emails and ham emails separately. The most time consuming function is yylex() that takes 31.9% of the total time. Following that is wordhash_search(), taking 29.5% of the total. Then word_cmp() consumes 5% of the time. The operations of these three functions are as follows.

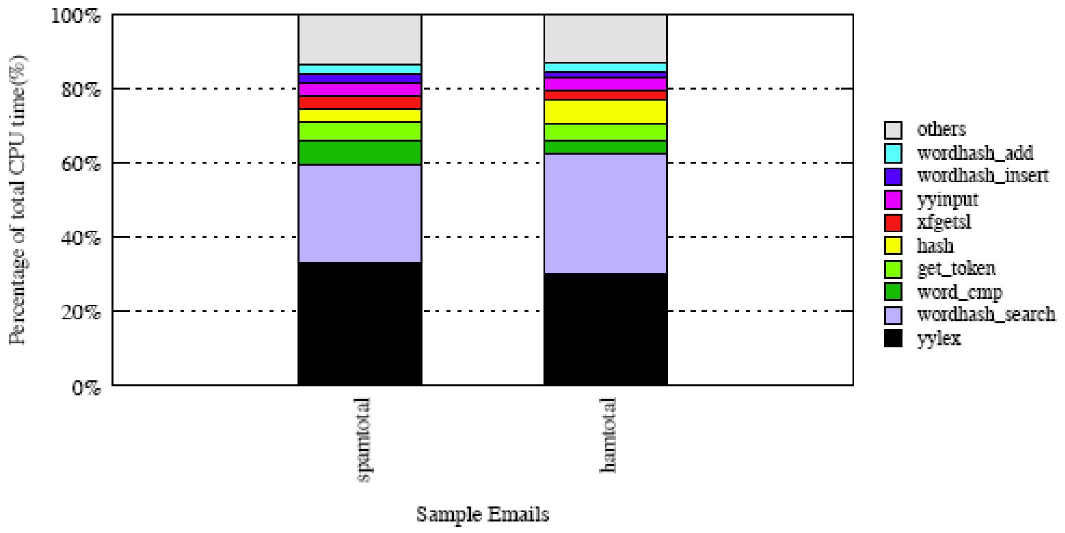


Figure 10. Profiling results of bogofilter in training mode

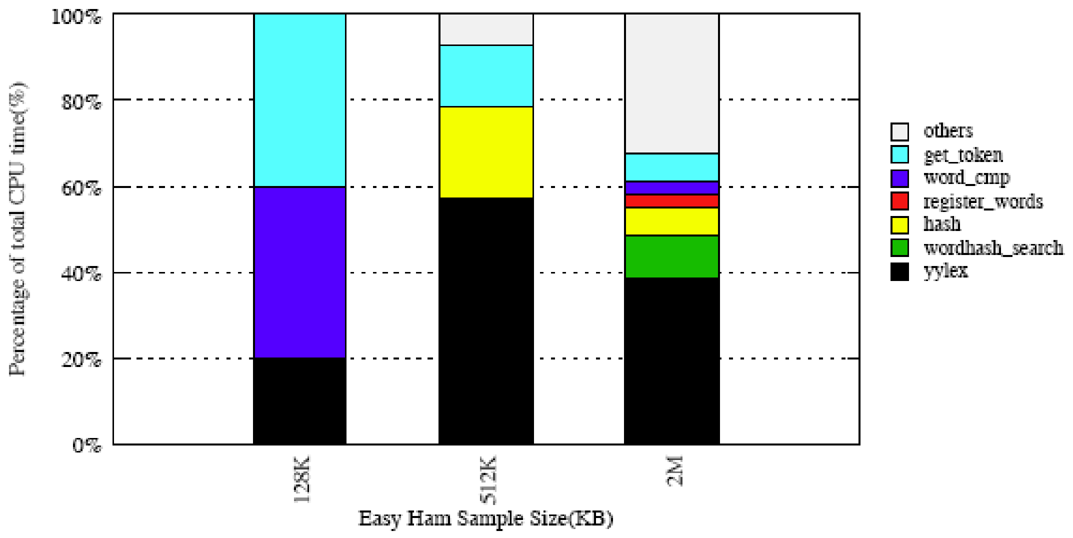


Figure 11. Profiling results of bogofilter with Ham samples

The lexical analyzer function, yylex(), recognizes tokens from the input stream and returns them to the parser. The regular expression matching procedure in this function uses is based on a Deterministic Finite Automata (DFA), taking in a string of input symbols. For each input symbol it will then transit to a next state. When the last Input symbol has been received, it will either accept or reject the input stream depending on whether the DFA is in an accepting state or a non-accepting state. Wordhash_search() is a function of only with 16 lines of C code. It searches a keyword in the hashing data structure. The functions returns with hash buffer address or NULL depending on the comparison of the input parameter with the keyword stored in hashing entries that organized in a linked list. The linked list traversing makes this function very time consuming.

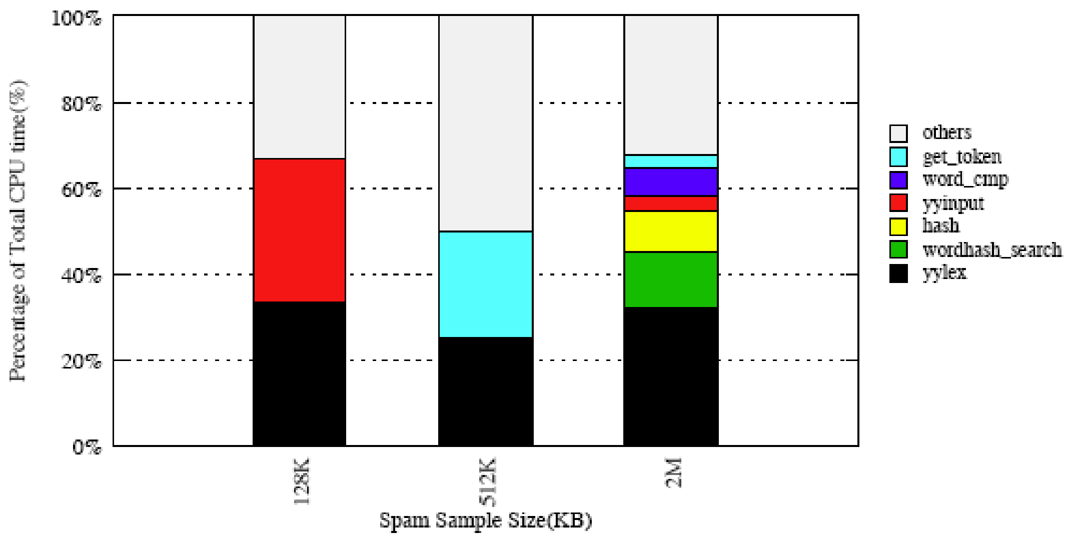


Figure 12. Profiling results of bogofilter with Spam samples

Filtering
The profiling results of bogofilter are shown in Fig. 11 and 12 depicting time-consuming situation of the functions on "Hard Ham" and "Spam" email samples, respectively. Results on the "Hard Ham" (Fig. 11) show that bogofilter spends considerable time (46.3%) on regular express matching function (yylex()) on average. The wordhash searching function (wordhash search()) takes average time of (13.3%). Analysis of the bogofilter source code reveals that function hash() takes care of the reasonable address for hash which should be a multiple of hash length takes amount of (9.9%) on average.

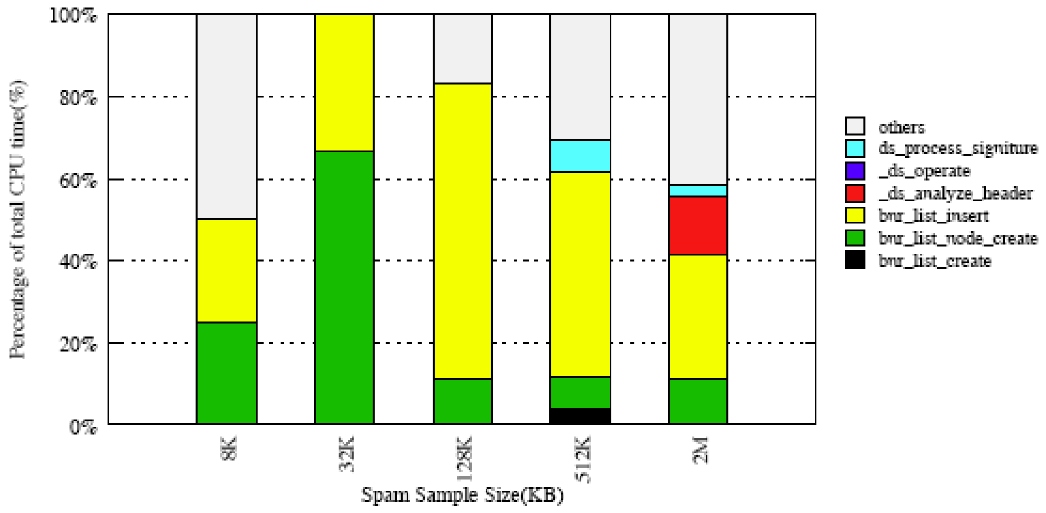


Figure 13. Profiling results of DSPAM using Spam samples

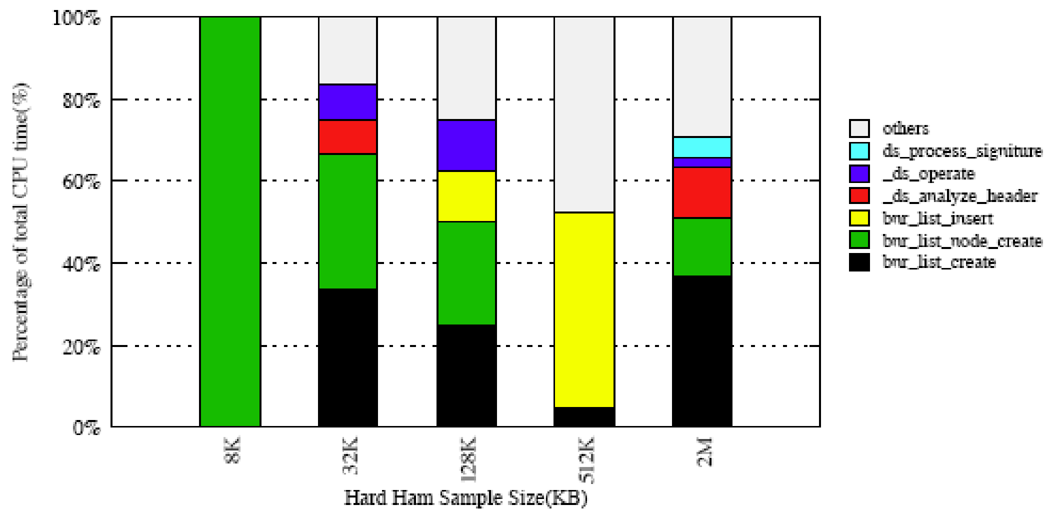


Figure 14. Profiling results of DSPAM using Hard Ham samples

5.3. DSPAM

Fig. 13 and 14 present the profiling results of DSPAM on spam and ham emails. In both figures, three functions (bnr_list_create(), bnr_list_node_create(), bnr_list_insert()) related to Bayesian Noise Reduction are the most time-consuming functions. Bayesian Noise Reduction is a statistical approach to evaluating coherence by instantiating a series of machine-generated contexts to serve as a means of contrast [34]. The implementation of BNR relies on a linked list data structure, and the insertion and creation of nodes take a significant amount of CPU cycles.

6. SIMULATION RESULTS

We investigate how the architecture parameters affect the performance of email filters. We conduct the following experiments with SimpleScalar simulator. We port one of the filters, CRM114, into SimpleScalar. We fed Hard Ham and Spam email samples to CRM114's filtering procedure. Table 3 summarizes the default parameters of Simplescalar simulations. The performance metrics interested are cycle numbers and cache miss rate. We vary parameters including the cache associativity and size, number of ALUs, issue width etc.

Table 4. Default Parameters of Processor Architecture

L1 Instruction cache size	16KB 512:32:1:1
L1 Data cache size	4KB 128:32:4:1
Number of Integer ALU	4
Number of FP ALU	4
Instruction Issue Width	4
Instruction Fetch Queue Size	4

Fig. 15 shows how the L1 data cache associativity affects the miss rate. The performance improvement of two input email samples is similar. The miss rate drops significantly from direct mapped cache to 8-way. 16-way cache configuration brings no considerable benefits. Fig. 16 shows how the L1 data cache size affects the miss rate. Two experiments show very similar results. There is large improvement on the miss rate when the cache size increases from 2KB to 64KB. After that, the reduction of miss rate gets small. Fig. 17 shows how the L1 data cache size affects the cycle time. Although the absolute cycle times are different in two experiments, larger cache shows better performance. However, the benefit is not significant when the cache size is over 128KB. Fig. 18 shows how the number of integer ALU affects the cycle time. More integer ALUs can improve the overall performance until the number reaches four.

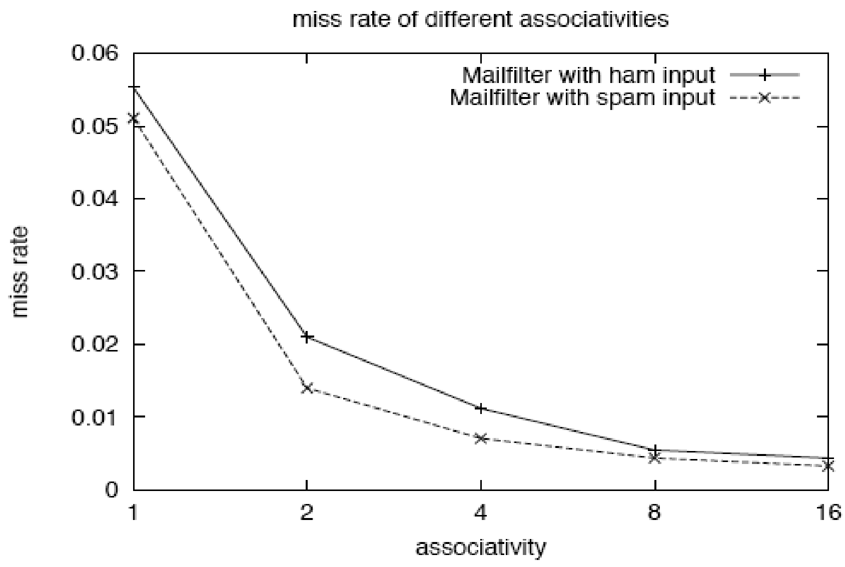


Figure 15. Miss Rate vs Cache Associativity

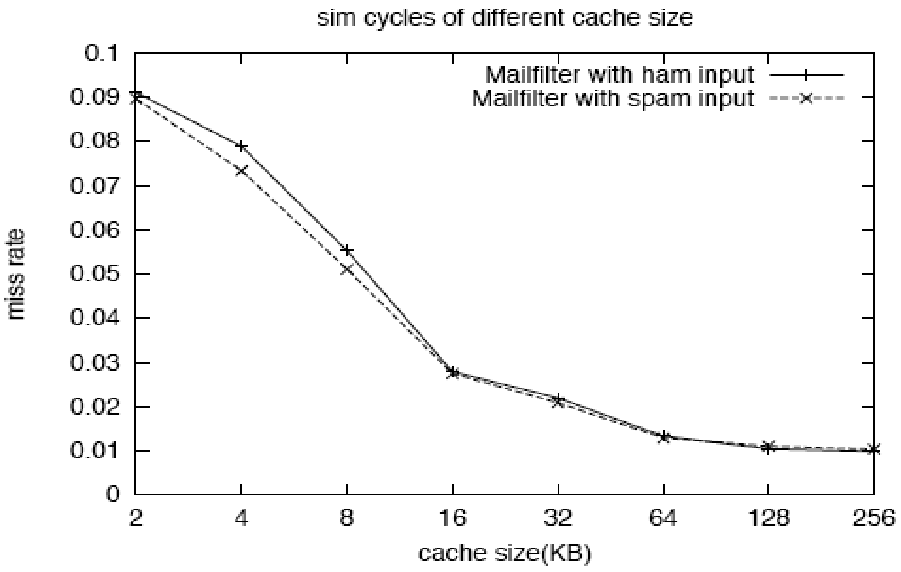


Figure 16. Miss rate vs cache size

We also vary the number of floating point ALUs and multipliers, however, find out that the performance is not sensitive to them (figures not shown). Two FP ALUs or two Multipliers does not outperform one FP ALU or multiplier cases. This is because the FP ALU/MULT instructions are only a tiny fraction of the Bayes classification function. Other functions such as regular expression matching and hashing consume the majority of the CPU cycles and use integer ALUs.

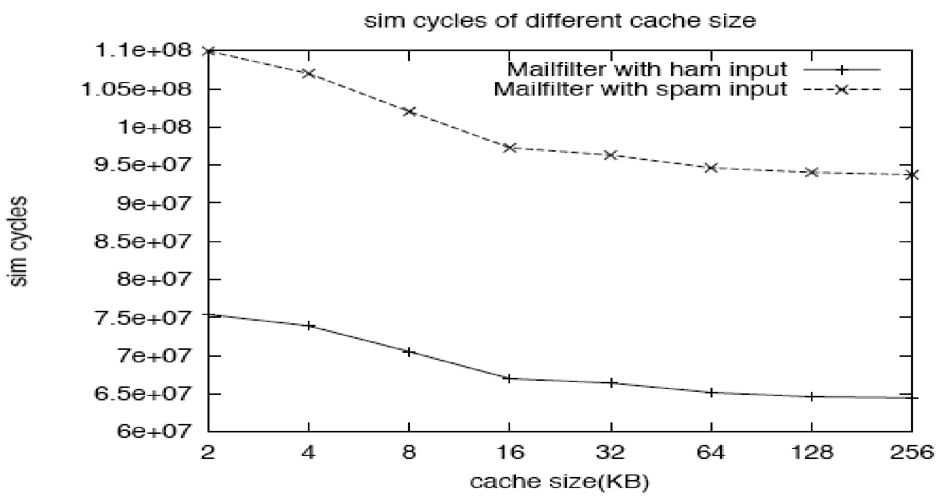


Figure 17. Cycle Time vs Cache Size

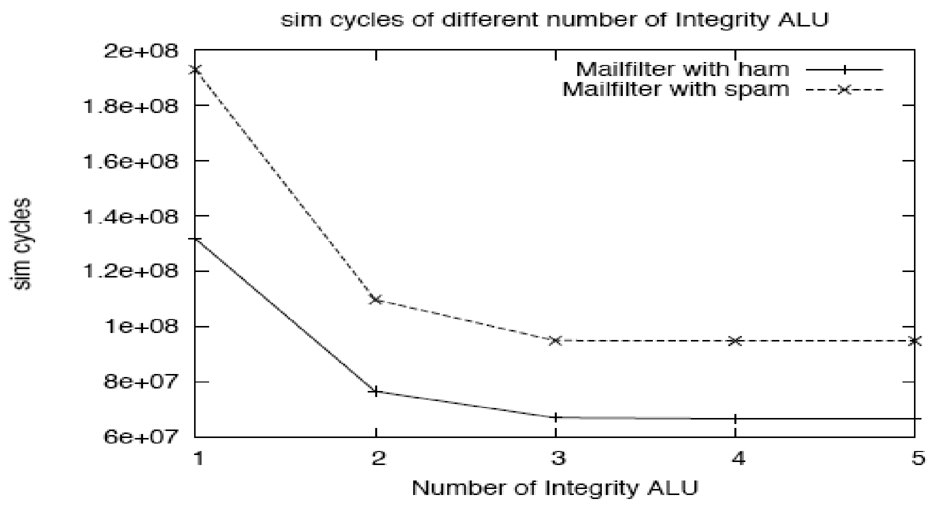


Figure 18. Cycle Time vs the Number of integer ALUs

7. Conclusions

In this paper, as the first step towards novel architecture designs, we present extensive data collected from measurement, profiling and simulation experiments using representative email filtering systems including CRM114, DSPAM, SpamAssassin and TREC Bogofilter. From the obtained data, we make the following observations:

- The data show that the types of emails (spam or ham emails) do not necessarily determine the processing time of the email filters. That is to say there is no direct correlation between the type of an email to the time that a filter needs to spend to classify it.
- The filtering time generally increases as the size of an email increases for easy ham and spam emails. For hard ham emails, the filtering time is not directly related to the size.

- C language based filtering systems such as CRM114, DSPAM and Bogofilter spend less time on the same workload than Perl based ones such as SpamAssassin, although the latter one may provide better APIs to email systems. However, it is worthy noting that the experiments do not focus on the accuracy of the email filtering systems under study.

- Profiling results reveal that regular expression matching, hashing and statistical algorithm computation take the majority of the CPU cycles. Among them, regular expression matching is the top cycle killer function. From simulation experiments on CRM114 with Simplescalar, we find out several guidelines that can help processor design for filtering emails.

- Larger cache associativity improves the cache miss rate, but the improvement is not significant after 16-ways.

- Larger data cache helps, but the benefit is not significant after the cache size increases to more than 128KB.

- Four integer ALUs seems enough for the default processor configuration. Floating point functional units are not sparse resources, one FP ALU and one FP Multiplier are sufficient.

The above findings help us understand the workload of email filtering systems. We will investigate hardware acceleration for the identified time consuming functions in the future.

ACKNOWLEDGEMENTS

The author would thank Jie Fan for her earlier work on this topic. The author thanks Dr. Li Zhao for her constructive comments on this work.

REFERENCES

[1] CRM114 - the Controllable Regex Mutilator. http://crm114.sourceforge.net/.

[2] DSPAM. http://dspam.nuclearelephant.com/.

[3] POPFile: An automatic mail classification tool. http://popfile.sourceforge.net/.

[4] The Apache SpamAssassin Project. http://spamassassin.apache.org/.

[5] TRE: POSIX Compliant Regular Expression Matching Library. http://laurikari.net/tre/.

[6] AOL Service. America Online Releases 'Top 10 Spam' List of 2003, December 2003. http://media.aoltimewarner.com/media/newmedia/cb press view.cfm?release num=55253692.

[7] BBC. Man gets nine years for spamming, April 2005. http://news.bbc.co.uk/2/hi/americas/4426949.stm.

[8] D. Burger, T. M. Austin, and S. Bennett. Evaluating Future Microprocessors: The SimpleScalar Tool Set. Technical Report CS-TR-1996-1308, University of Wisconsin-Madison, Computer Science Department, 1996.

[9] E. Burns. Instant messenger a target for malicious attacks, January 2006. http://www.clickz.com/showPage.html?page=3581866.

[10] Shalendra Chhabra. Fighting spam, phishing and email fraud. Master's thesis, University of California Riverside, December 2005.

[11] G. Cormack and T. Lynam. TREC Spam Filter Evaluation Tool Kit. http://plg.uwaterloo.ca/˜trlynam/spamjig/.

[12] J. Fenton and M. Thomas. Identified Internet Mail. draft-fenton-identified-mail-02, May 2005. http://www.identifiedmail.com/draft-fentonidentified-mail.html.

[13] D. Fetterly, M. Manasse, and M. Najork. Spam, Damn Spam, and Statistics: Using statistical analysis to locate spam web pages. In Seventh International Workshop on the Web and Databases, Paris, France, June 2004.

[14] A. Foong, T. Huff, H. Hum, J. Patwardhan, and G. Regnier. TCP Performance Re-visited. In IEEE ISPASS, Austin, TX, March 2003.

[15] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions. RFC 2045, November 1996. http://www.ietf.org/rfc/rfc2045.txt.

[16] P. Graham. A Plan for Spam, August 2002. http://www.paulgraham.com/spam.html.

[17] J. Graham-Cumming. Spammers Compendium, 2006. http://www.jgc.org/tsc/.

[18] T. Hansen, D. Crocker, and P. Hallam-Baker. DomainKeys Identified Mail. draft-ietf-dkim-overview-03.txt, October 2006. http://ietf.org/internet-drafts/draft-ietf-dkim-overview-03.txt.

[19] S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648 (Proposed Standard), November 2006.

[20] J. Linn. Privacy Enhancement for Internet Electronic Mail. RFC 1421 (Historic), February 1993.

[21] J. Lyon and M. Wang. Sender ID: Authenticating E-Mail. RFC 4406, April 2006. http://www.ietf.org/rfc/rfc4406.txt?number=4406.

[22] R. McMillan. Gartner: Consumers to lose $2.8 billion to phishers in 2006, November 2006. http://www.networkworld.com/news/2006/110906-gartner-consumers-to-lose-28b.html.

[23] Microsoft. IT Value Card: Microsoft Office Live Communications Server 2005 and Communicator 2005, December 2005.

[24] P. Pantel and D. Lin. SpamCop: A Spam Classification and Organization Program. In Proceedings of AAAI-98 Workshop on Learning for Text Categorization, 1998.

[25] Postnit Inc. Press release: Postini announces top five 2007 messaging security predictions as email spam becomes front burner issue again in the new year, December 2006.

[26] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE, 77(2):257286, February 1989.

[27] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In Proceedings of AAAI-98 Workshop on Learning for Text Categorization, 1998.

[28] Sensory Networks. NodalCore Scanner. http://www.sensorynetworks.com/.

[29] P. Smoot and B. Fried. The challenges of managing a megaservice. ACM Queue, 3(10), December 2005. http://www.acmqueue.com/modules.php?name=Content&pa=showpage&pid=353&page=1.

[30] The Internet Patrol. Mariam Abacha Convicted! Nigerian 419 Scam Fraudstress Jailed, 2003. http://www.theinternetpatrol.com/mariamabacha-convicted-nigerian-419-scam-fraudstress-jailed.

[31] URLblacklist.com. URL Blacklist, 2006. http://www.urlblacklist.com/.

[32] Vanquish Inc. Vanquish anti-spam appliance, 2003.

[33] M. Wang and W. Schlitt. Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. RFC 4408, April 2006. http://rfc4408.x42.com/.

[34] J. A. Zdziarski. Bayesian Noise Reduction: Contextual Symmetry Logic Utilizing Pattern Consistency Analysis. January 2004.

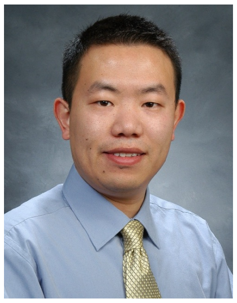
[35] J. A. Zdziarski. Concept Identification using Chained Tokens. In MIT Spam Conference, February 2004.

[36] J. A. Zdziarski. Ending Spam. No Starch Press, 2005.

[37] Luis von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. Communications of the ACM, 47(2), February 2004.

[38] Battista Biggio, Giorgio Fumera, Ignazio Pillai, and Fabio Roli. Image Spam Filtering by Content Obscuring Detection. In CEAS 2007 Fourth Conference on Email and Anti-Spam, Mountain View, CA, August 2007.

Authors



Yan Luo: Yan Luo earned his PhD in Computer Science from the University of California Riverside in 2005. Since then he has been an assistant professor in the Department of Electrical and Computer Engineering at the University of Massachusetts Lowell. Dr. Luo's research spans computer architecture and network systems with recent focus on programmable architectures for high performance networking. He has served on the program committee of many IEEE/ACM conferences and co-chaired the 2009 workshop on performance evaluation of next generation networks. He is a member of IEEE and ACM.