Traffic classification-based spam filter

Ni Zhang¹,², Yu Jiang³, Binxing Fang¹, Xueqi Cheng¹, Li Guo¹

¹Software Division, Institute of Computing Technology, Chinese Academy of Sciences, 100080, Beijing, China
²Graduate School of Chinese Academy of Sciences, 100039, Beijing, China
³Computing Center, Bureau of Statistics of Heilongjiang Provincial Government, Harbin 150001, China

Abstract

We propose an unsupervised spam filter called Bulk Mail Traffic Classification (BMTC) for filtering junk mails from the perspective of ISPs. Our insight is that spammers generally sent mass unsolicited emails with few alterations to a common message content, which can be found at an extensive traffic environment. In our approach, we classify email delivery traffic into different categories by the similarity of message contents. Then we can decide whether or not a particular email category is spam by the number of similar mails of this category and take measures to filter it. We also design a simulator, two sketch data structure, and a series of algorithms to support our method. We have applied BMTC to email traffic data captured at one of the largest commercial Internet service providers in China, and the experimental result indicates that a 70.4% reduction of junk mail traffic can be achieved with our method. The results also show that BMTC is practical. We can implement it in a high-volume traffic environment handling over millions of mails every day with small memory consumption.

1 Introduction

The rapid increase in the volume of unsolicited bulk commercial emails, also known as junk mails (or spam), has become a serious threat not only to the Internet but also to our society. A recent study by Brightmail¹ found that more than 65% of global email traffic is spam. Furthermore, AOL and MSN, two large ISPs, report that they block a total of 2.4 billion junk mails from reaching their customers per day. This traffic corresponds to about 80% of daily incoming emails at AOL [1]. A number of approaches have been proposed to alleviate the impact of spam, the majority of which is designed to identify spam after it has already been delivered to the intended recipient’s email box. However, even if these techniques are successful so that spam never reaches the target recipient’s eyeballs, there is nothing to prevent the junk mail senders from wasting a significant amount of bandwidth and causing delays to the delivery of good mails. Therefore, a new point of view is now appearing to protect the network resources from abuse by spam, not just to protect the end users.

In this paper we are interested in the feasibility and effectiveness of stopping or reducing spam traffic from the perspective of ISPs (Internet Service Providers). As a prerequisite to stop or reduce spam traffic at an ISP, this paper proposes a novel technique to filter junk mails from bulk email traffic in a high-volume traffic environment.

The main contributions of our study are: (1) to classify email delivery traffic into different categories, making it possible to handle bulk e-mails. (2) The design and implementation of a simulator with two sketch data structures and a series of methods for detecting junk mails effectively.

Our approach utilizes the fact that junk mail senders generally sent mass unsolicited emails with few alterations to common message contents. Thus, we classify email traffic into different categories by the similarity of message contents. If the number of similar emails in any category exceeds the spam threshold which is a tunable parameter, we will mark this entry as spam one. If a new email is classified into a spam category, we can then take measures to filter it.

We apply our method to email traffic data captured at one of the largest commercial ISPs in China, and the experimental result indicates that a 70.4% reduction of junk mail traffic can be achieved.

The rest of the paper is organized as follows. Section II presents related work in spam filtering field and discusses their limitations in order to clarify the motivation of this research. We describe key techniques of our Bulk Mail Traffic Classification (BMTC) in section 3. Section 4 proposes the design and implementation of our mechanism. Section 5 reports the experimental results and compares BMTC with some other spam filtering methods. Finally, Section 6 concludes the paper.

2 Related work

Flooding of spam has become a headache problem to both Internet and society, and a number of methods have been proposed for filtering spam. In general, spam filter can be grouped into three types: access filtering, economic filtering, and content-based filtering.

Access filtering, which merely verifies and authenticates the header information of an email and without disclosing users’ privacy, can be divided further into three categories: blocking, delaying and temporary failure. Blocking that is generally accepted and employed now, means refusing to accept mails from a particular sender. Unfortunately, according to [2], blocking methods are only partially effective, mainly because it is easy for junk mail senders to conform to the heuristics and change their IP address frequently.

Delaying is a method for decreasing throughput of junk mail sending by injecting delay to the Simple Mail Transfer

¹ http://www.brightmail.com
Protocol (SMTP) connection between junk mail senders and email servers. Most mechanisms [3] assume that spam is sent at a high rate, and that slowing it down will reduce the amount of spams received. However, these mechanisms are unlikely to be effective, since the junk mail sender have already sent at low rate [2].

The third kind of access-filtering method is temporary failure. Since SMTP is considered as an unreliable transport protocol, the mechanism of tackling temporary failures has been documented into the corresponding request for comments (RFC) file. Any well-behaved email servers should retry if encountering a temporary failure for a delivery attempt. However, most spamming software today does not retry in order to obtain high throughput. This is the premise of Greylisting2. A server running Greylisting keeps a list of triples consisting of the sender’s IP address, the sender’s email address and the recipient’s email address. If the server has never seen a triple before, it will refuse this delivery and any others that may come within a certain period of time with a temporary failure. Since spamming software do not retry, this would reduce vastly the amount of spam accepted by mail server, while a good mail is subject to some delay. The weakness of this approach is that if (and when) junk mail senders implement retries, the technique will become less effective [2].

Since the main attractiveness of spamming is that sending large amounts of small email messages is relative cheap compared to other marketing techniques, the idea behind economic-filtering is to make sending high volumes of email traffic to be more expensive. The two main categories of economic solutions are computing-time-based systems [4] and money-based systems [5]. The former forces the junk email sender to spend considerable computing resources to send a single spam message, while the latter charges a small amount of money from every email sent.

Content-based filtering happens after a message is fully received. In this case, filter can be implemented by a variety of means, such as rule-based filtering, Naive Bayesian classification [6], support vector machines (SVM) [7], memory-based approach, and checksums methods in collaborative circumstance [8]. Although these filtering mechanisms can effectively reduce the impact of spam on an individual user, they do nothing to protect network resources wasted by spam. In addition, all filters stated above suffer from the problem of incorrectly classifying email and must be continually maintained and updated, so long as junk mail senders develop new means to evade them.

The work of Kenichi et al [9], which describes a density-based spam detector, is much similar to ours. They use document space density and design an unsupervised learning engine with a direct-mapped cache to identify spam. However, when we have implemented their algorithm, we found that although it is an effective mechanism, it has bad performance in vector presentation and update mechanism in detecting bulk emails. In contrast, we develop an effective algorithm that is based on the analysis of email traffic, more effective fingerprint technique, fast similarity check, and flexible update mechanism, and we design two sketch data structures to support our method and a simulator to test our algorithm. Our methods does not require any parsing, aggregation, or tokenizing of the input traffic, never blocks the good emails, nor decreases the delay to good mails.

3 The key techniques of BMTC

There are many design choices in developing a spam filter in a high-volume traffic environment. The primary design criteria and operating objectives of such anti-spam system include:

1) Automatic hand-free deployment and an online update mechanism requiring little or no human interaction.
2) Accuracy in detecting truly spam with very low false positive rate.
3) Efficiency in operation in a high-volume traffic environment with little or no impact on networks’ throughput and latency.

These objectives are difficult to meet concurrently, yet they do suggest an approach that may balance these competing criteria for a spam filter.

A. Fingerprint technique

Our goal is, for each input email, to quickly decide whether it is similar to some earlier emails. To do this, one can often design an approximate data structure that maintains a small sketch of the large object rather than an exact representation. In our mechanism, we adapt a fingerprint technique developed by Manber [10] for finding similar files in a large file system and applied by Broder [11] to detect similar Web documents.

Fingerprints are integers generated by a one-way function applied to a set of bytes. A good fingerprint algorithm generates well-distributed fingerprints, which have the property that if two fingerprints are different the corresponding objects are certainly different, and there is low probability that two different objects have the same fingerprint (The latter event is called a collision.)

We view each email \( M \) as a sequence of bytes \( b_1b_2...b_i \). Contiguous bytes \( b_{i-1}...b_{i-\ell} \) contained in \( M \) is called a window with a length \( \ell \). One method of generating the representative fingerprints for an email is to compute a Rabin fingerprint [12] for every \( \ell \) length window by the following expressions, where \( p \) and \( \delta \) are constant integers.

\[
F(M, i, l) = (b_pb_{i-1}b_{i-2}...b_{i-l} + b_{i-l+1} + b_{i}) \mod \delta \tag{1}
\]

\[
F(M, i + 1, l) = [(F(M, i, l) - b_{i}p') \times p + b_{i+1}] \mod \delta \tag{2}
\]

Therefore, we can compare the representative fingerprints of different emails to estimate their similarity. Computing the Rabin fingerprints is fairly fast since \( p \) and \( \delta \) are constant, and advancing the fingerprints only requires a subtraction, a multiplication, an addition, as shown in (2), rather than generate a new one from scratch.

We also find that it is impractical to keep every computed fingerprint, which would make us fall into memory crisis. We

2 http://projects.puremagic.com/greylisting/
can simply select first $m$ fingerprints of an email. To our surprise, this technique has worked very well in our experiments as we will describe in following sections. In this case, we can associate every email $M$ to a set of fingerprints $P^*_m(M)$, which contains $m$ fingerprints.

$$P^*_m(M) = \{F(M,i,i), F(M,i+1,i), \ldots, F(M,i+m-1,i)\} \quad (3)$$

B. The similarity of emails and the email category

If two sets of representative fingerprints share at least $k$ elements, we say that the emails represented by these sets are similar.

$$\text{sizeof}(P^*_m(M_1) \cap P^*_m(M_2)) \geq k \Rightarrow M_1 \propto M_2 \quad (4)$$

If two emails are similar, we say that they belong to the same mail category $C$.

$$M_1 \propto M_2, M_1 \in C \Rightarrow M_2 \in C \quad (5)$$

We also find that ordinary users seldom send more than 50 similar emails, while junk mail senders often send the same spam with a number far more than that value. So if the number of similar emails in any category exceeds a spam threshold which is a tunable parameter, like 50, we can mark this category as spam one.

C. Two sketch structures and our algorithm

When handling over hundreds of emails per second, a million previous emails may be checked in order to handle the current single e-mail. So we need an efficient lookup mechanism and similarity check method. Moreover, a hot category is more important than the one seldom visited. So we need a flexible update mechanism to drop unimportant entries and leave important ones in order to make space for new entries.

To solve these problems, we have developed a new type of unsupervised learning engine which uses two sketch data structures shown in Figure 1 to meet our needs:

1. The fingerprint database (FD) is a hash bucket which stores all distributed fingerprints. $h_i$ is determined by performing the mod $\theta$ (a constant) operation to fingerprint $f_i$.

$$h_i = f_i \mod \theta \quad (6)$$

If two fingerprints have the same $h_i$, we simply add an entry and never overwritten the former. FD also stores the pointer to the related mail category entry in mail database.

2. The mail database (MD) stores all mail categories’ information, which includes the numbers of similar mail, the first mail ID and the last mail ID of this category, and the pointers to all fingerprints belonged to this category.

When our algorithm for detecting spam is running, two sketches, MD and FD, are used to store the most recent information. For every new coming email, the algorithm first generates the representative set of fingerprints. Each fingerprint in this set is checked against the FD. If it is matched, then updates the related mail category in MD, for example, increasing the number of similar emails and updating the last mail ID. Otherwise, we create a new category in MD and insert the set of fingerprints into FD. If the number of similar emails in any categories exceeds a given spam threshold, we will mark this categories as spam entry.

![Fig. 1. Two sketch data structures](image)

D. Memory management for sketch data structures

In mail database (MD), we should keep the most remarkable mail categories and delete unimportant ones in order to make space for new entries. To do this, our algorithm first assigns a unique $ID$ (an integer) for every input email by the order of its arrival. According to the characteristics of traffic-based spam filtering, the spam category is much important than good category in MD because they will be visited frequently in a very long period. Hence, a definition termed average distance of category is introduced and by which we can drop the outdate category.

We will use the follow notations: Let $C_i$ be the $i^{\text{th}}$ category in MD, $I_i$ be the maximum mail ID, $I_1$ and $I_e$ be the first mail ID and last mail ID of $C_i$, $\text{sizeof}(C_i)$ be the number of similar mails of $C_i$, and $D(C_j)$ be the average distance of $C_i$. Then:

$$d_i = I_i - I_e \quad (7)$$

$$D(C_i) = \frac{I_i - I_1}{\text{sizeof}(C_i)} \text{ if } \text{sizeof}(C_i) > 1 \quad (8)$$

$$D(C_i) = \infty \text{ if } \text{sizeof}(C_i) = 1$$

Then the algorithm drops entries according to two rules as following.

1. For spam category and suspect category (which is described in Sec. 4): $d_i > 10C_i$
2. For good category: $d_i > 10000$.

4 System configuration and implementation

We describe the implementation of our mechanism from the aspects of trace data, its design, and selection of algorithm parameter. Since it is not allowable to do experiments in real networks, we have performed an offline simulation. The simulator must generate original email traffic as in real. Furthermore, the simulator should be easy to use and provide platform of comparison with different filters.

A. Trace data

<table>
<thead>
<tr>
<th>Trace</th>
<th>Date</th>
<th>Mails in trace</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2005.6.15</td>
<td>14673</td>
<td>training</td>
</tr>
<tr>
<td>2</td>
<td>2005.6.30</td>
<td>195727</td>
<td>testing</td>
</tr>
</tbody>
</table>
We collected two traces in June 2005, which is captured at one of the largest commercial ISPs in China, and kept in tcpdump format. The traces only contain SMTP traffic which are composed of all clients to server directional packets for this analysis. We use trace 1 to tune algorithm parameters and use trace 2 to test our algorithm. The summary information of trace data is shown in Table I.

### B. Design of simulator

Figure 2 shows the system structure of our simulator we have designed. The structure is chosen to satisfy the above requirements and consists of four parts. The first part is the traffic generator, which gets packets from trace data and calls packet sending function provided by libnet library to generating email traffic. The second part is the wrapper, which is a platform that builds a bridge between the generator and the spam filter. Since our trace includes unanswered SYN packets, which are most likely generated by port scanners or resetting SMTP connections, so the wrapper should choose valid connections and then pre-process these SMTP traffic into email text. The other work of wrapper is responsible for receiving temporary results from filter and keeping them in a log file. The third part is the spam filter, which classifies email texts provided by wrapper into different categories by looking up FD and then updates MD and FD. The filter also sends the classification results of new coming email to the wrapper. The last part is the statistical unit, which can access log file and MD to gather information and output the statistics.

![Figure 2. The system structure of simulator](image)

### C. Algorithm parameters

Our algorithm includes various control parameters, such as the length of window \(l\), the size of fingerprints set \(m\), and the spam threshold, which will be determined in training phrase. The training data includes 7583 good mails, distributed in 2352 categories and 7090 spam mails, distributed in 23 categories. The effects of altering these parameters are shown in Figure 3 to Figure 6.

1) **Spam threshold and suspect threshold**

Threshold value is an indicator of spamminess, which can be used for marking a category as a good or spam one. If we keep the threshold value very low, then the chances of false positives increases. If we choose a large value, we may miss some spams.

The distribution of the number of similar emails of good and spam category is shown in Figure 3 and Figure 4. From these figures we can see that 81.6% spam categories have more than 50 similar emails, while 98% good categories have less than 50 similar emails, except that some categories were composed of “bounced” mail messages sent by email servers. Luckily, most email servers use a sender address of the null sender “<>” or “postmaster” to operate “bounced” mail, which makes it fairly simple to workaround, by integrating a simple rule-based method to our algorithm.

We also find some categories whose number of similar emails between 30 and 50 appear in both distributions, which will decrease the performance of BMTC in traffic classification. Therefore, we introduce another parameter, suspect threshold, to solve the problem. When facing category whose number of similar emails between the given spam threshold and the given suspect threshold, we need to adopt another method to detect them, which is left for future work.

We also use the total number of emails in suspected categories as a criterion to evaluate the performance in traffic classification. We define the suspected ratio as followed:

\[
\frac{N_r}{N} = \frac{\text{total number of emails in suspected categories}}{\text{total number of emails}}
\]

where \(N_r\) is the total number of suspected emails and \(N\) is the total number of emails. For our test, a spam threshold value of 50 and a suspect threshold value of 30 are employed.

2) **Parameter \(l\) and \(m\)**

These two parameters are determined by the capability of detecting spam and memory consumption. Usually a junk mail sender sends multiple copies of an email by making few alterations. If \(l\) is too large, only large regions are matched, which would increase the average quality of matches but decrease the number of potential spams that are detected. If \(l\) is too small, the average quality of matches is sacrificed, since the probability of two fingerprints colliding increases.

There is also a trade off with \(m\) in term of how well each email is sampled. Large values of \(m\) can increase the likelihood of finding a match for a given email but need more memory for storing more fingerprints and vice versa. Figure 5 shows the accuracy of detecting spam from trace 1 and Figure 6 shows memory consumption for different values of \(l\) and \(m\). We can see that the middle value of \(m\) and the middle value of \(l\) are most effective. We choose \(l=70\) and \(m=60\) for our test.

### 5 Experimental results

This section summarizes some important statistics. The testing is done using an entirely different trace of 195727 emails.

#### A. Results on spam through trace 2

We analyzed trace data through a simulator as described in Sec. IV in test phrase. A server with dual processor Pentium IV, 2.4GHz, 4Gbytes of memory, running Linux9.0 is used as the platform of spam filter in the experiment.

In trace 2, 73% (143,563) of the mails are spams and they distributed only in 115 spam categories, while 36,039 good categories just include 27% (48,971) good mails.
These results show that the similarity is a good index to distinguish spam from good mails.

Once BMTC deals with ten thousand mails, we observe the number of mails classified into three categories, good, spam, and suspected. As is shown in Figure 7, emails classified into suspected categories decrease remarkably with the number of mails processed increasing. When processing more than 150,000 emails, total suspected emails increase slowly and nearly close to a constant (3193 emails). In our test, \( r_s = 0.02 \). That is, almost 98% emails can be classified by BMTC.

Any category in which the number of similar mails is more than 50 is marked as spam one and next mails classified into this category are then viewed as spam. As a result, our method cannot identify the first fifties emails in spam category in real time. The total number of identified junk mails is 137813, which is 70.4% of the total number of emails. This means that a 70.4% reduction of email traffic can be achieved after a short online learning time.

The category with less than 30 similar mails is a good category and the one with number of mails between 30 and 50 is considered as a suspected category that cannot be classified by BMTC.

As is shown in Figure 7, the suspected ratio (as described in formula 9) decrease greatly, which means BMTC has better performance in traffic classification. As a result, we confirm that BMTC can do much better in a high-volume traffic environment.

Figure 8 shows the memory consumption of our method. The non-optimized algorithm just consumes 21MByte within a reasonable CPU time when processing 195,727 emails, which encourages us greatly.

B. Performance Comparison and Discussion

As is reported above, our method can identify 115 spam categories and 137813 junk mails (=143,563-115*50). None of the known content-based spam filter seems to be able to handle bulk mails by limited memory and reasonable CPU time while without human maintenance.

As far as we know, the work of Yoshida (DMC) et al [9] which describes a density-based spam detector that is similar to our work. Table II shows some comparisons between the two algorithms. It shows also the performance of DMC on our email data. The traffic of our trace 2 with first 60,000 emails is used for this experiment. In comparison, both algorithms have the same online learning threshold value (50), which means that they cannot identify junk mails until they find 50 similar emails.

In [9], the implementation employed a hash function provided in linux C library to represent vectors, a direct-mapped cache for similarity checks, and an overwrite...
mechanism to control entries in the hash database. When all the hash values in the cache are overwritten by later emails, DMC deletes the entry in the hash database. Therefore, it is possible to delete hot categories. As shown in Table 2, DMC deleted 65 categories for hash value overlapped, and detected 89 spam categories and 25,432 spams. 7 categories are “bounced mail”, which are wrongly identified as junk mails by DMC.

<table>
<thead>
<tr>
<th>TABLE 2 Comparison with DMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMC</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>Spam threshold (# mails)</td>
</tr>
<tr>
<td>Suspect threshold (# mails)</td>
</tr>
<tr>
<td>i</td>
</tr>
<tr>
<td>m</td>
</tr>
<tr>
<td>Number of spam category</td>
</tr>
<tr>
<td>“bounce” mail category</td>
</tr>
<tr>
<td>Number of spam</td>
</tr>
<tr>
<td>Accuracy</td>
</tr>
<tr>
<td>Recall rate</td>
</tr>
<tr>
<td>Overwritten spam category</td>
</tr>
<tr>
<td>Total CPU time (second)</td>
</tr>
</tbody>
</table>

With a good fingerprint algorithm and two sketch data structures, our BMTC never overwrites old fingerprints, nor drops hot categories. Therefore, BMTC has a desirable result, identifying 82 spam categories and 32245 spams within a reasonable amount of CPU time, by only a few memory consumption. We have checked the samples from all 82 spam categories and find that they are all junk emails.

We check other 27755 (60000-32245) mails, which shows that there are still 11555 junk mails missed by both methods. The reason is that BMTC and DMC cannot identify spam categories which include similar mails less than the spam threshold. As shown in TABLE 2, BMTC achieves 74% recall rate which is much better than that of DMC. Our method also adopts a simple rule-based policy to detect “bounced mail” message sent by email servers (as described in section IV), which can effectively distinguish between real spam categories and non-spam categories whose capacity over 50. As a result, our method marks 82 spam categories and achieves a better accuracy and recall.

An apparent advantage of DMC is CPU time required. DMC could handle 2,727 (=60000/22) emails per seconds, while the required CPU time of BMTC is five times more than that of DMC since never overwrite old fingerprints. However, BMTC can achieve a better result at the cost of processing speed. The test shows that both policies are practical and reasonable. In fact, the capability of handling 512 (=60000/117) emails per seconds can also do well in live environment. In addition, an optimized BMTC version may be focused on this issue.

When using supervised learning methods like Naïve Bayes and SVM, such filters require maintenance tasks. On the contrary, except a short online learning time, BMTC needs no supervisor for learning or decoding message content. This implies that no one is required to trace message content manually and user’s privacy is inherently protected.

6 Conclusion and Future Work

In this paper, we have presented a new technique BMTC for detecting spam from bulk mail traffic. Our technique classifies mail delivery traffic into different categories by the similarity of message contents. If the number of similar mails in any category exceeds a spam threshold, we will mark this category as spam one. We also design two sketch data structures and a series of methods to support our method.

BMTC has three distinct advantages.

1) Automatic hand-free deployment and an online update mechanism.

2) Identifying spam from bulk email traffic with a high accuracy.

3) Handling over large amounts of mails by small memory consumption within a reasonable CPU time.

In addition, a distinguishing feature of our method is that it not only protects the end-users from excessive volumes of unsolicited mails, but also it can cut off spam, and thus effectively utilizes the network bandwidth and reduces the delays to good mails.

The experimental results indicate that the BMTC is effective and practical, and a reduction of 70.4% junk mails may be achieved by our method. We can sketch an implementation in a high-volume traffic environment that requires no modification to the existed codes.

As the work is in progress and the results described in this paper are preliminary, for future work we will evaluate our method according to its sensitivities of parameters and the dependencies of the results on the used data set. Further, we plan to build an online system to filter spam and evaluate our method in live environments. In addition, evaluation of misclassification, e.g., false alarm rate, and the countermeasures to spam attack also remains for future work.

REFERENCES


