International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 5, July - 2012

A FEATURE SUBSET SELECTION IN E-MAIL SPAM DETECTION

Sivannarayana Nerella[1], K. Rajani Devi[2]
1 Final year M.Tech student, N.I.E.T

2 Assoc. Professor, H.O.D., Department of IT, N.I.E.T.
Sattenapalli (M), Guntur (Dist.), A.P.

ABSTRACT

In the recent in formation industry huge amounts of data is being collected and store continuously. For frequent data updating data collecting agents may communicating with various sources from different places. *Likely electronic mail communication is indispensable nowadays, but the electronic mail spam problem continues growing drastically. In present trends' the notion of collaborative spam filtering with near-duplicate similarity matching scheme has been widely discussed. The basic idea of the similarity matching scheme for spam detection is to maintain a known spam database, information passed by the user, to block subsequent near-duplicate spams. on purpose of achieving effective similarity matching and reducing storage utilization, prior works mainly represent each electronic mail by a succinct abstraction derived from mail content text. However, these abstractions of mails cannot fully catch the evolving nature of spams, and are thus not effective enough in near-duplicate detection. in this paper, we propose a new electronic mail abstraction scheme, which considers e-mail layout structure to represent electronic mails. we represent a procedure to generate the electronic mail abstraction using html content in electronic mail, and this newly devised abstraction can more effectively capture the near-duplicate phenomenon of spams. Moreover, we design a complete spam detection system cosdes (standing for collaborative spam detection system), which possesses an efficient near-duplicate matching scheme and a progressive update scheme. the progressive update scheme enables system cosdes to keep in most up-to-date information for near-duplicate finding. we evaluate cosdes on a live data set gathered from a real electronic mail server and show that our system outperforms the prior approaches in finding results and is applicable to the present world.*

Index terms

Spam detection, electronic –mail abstraction, near-duplicate matching.

I. INTRODUCTION

Electronic communication is prevalent and indispensable although the techniques used by spammers vary nowadays. However, the threat of unsolicited junk e- mails, also known as spams, becomes more and more serious. According to a survey by the website top ten- reviews. 40 percent of e-mails were considered as spams in 2006. The statistics collected by MessageLabs1 show that recently the spam rate is over 70 percent and persistently remains high. The primary challenge of spam detection problem lies in the fact that spammers will always find new ways to attack spam filters owing to the economic benefits of sending spams. Note that existing filters generally perform well when dealing with clumsy spams, which have duplicate content with suspicious keywords or are sent from an identical notorious server. Therefore, the next stage of spam detection research should focus on coping with cunning spams which evolve naturally and continuously.

Although the techniques used by spammers vary constantly, there is still one enduring feature: spams with identical or similar content are sent in large quantities and successively. Since only a small amount of e-mail users will order products or visit websites advertised in spams, spammers have no choice but to send a great quantity of spams to make profits. It means that even with developing and employing unexpected new tricks, spammers still have to send out large quantities of identical or similar spams simultaneously and in succession. This specific feature of spams can be designated as the near-duplicate phenomenon, which is a significant key in the spam detection problem.

In view of above facts, the notion of collaborative spam filtering with near-duplicate similarity matching scheme has recently received much attention. The primary idea of the near-duplicate matching scheme for spam detection is to maintain a known spam database, given by user feedback, to block subsequent spams with similar content. Collaborative filtering indicates that user knowledge of what spam may subsequently appear is collected to detect following spams. . Overall, there are 3 key points of this type of spam detection approach we have to be concerned about.

First, an effective representation of e-mail (i.e., e-mail abstraction) is essential. Since a large set of reported spams has to be stored in the known spam database, the storage size of e-mail abstraction should be small. Moreover, the e- mail abstraction should capture the near-duplicate phenomenon of spams, and should avoid accidental deletion of non spam e-mails (also known as hams). Second, every incoming e-mail has to be matched with the large database, meaning that the near-duplicate matching process should be substantially efficient. Finally, the latest spam's have to be included instantly and successively into the database so as to effectively block subsequent near-duplicate spams. To achieve the small storage size and efficient matching, prior works mainly represent each electronic mail by a succinct abstraction derived from electronic mail content text. Moreover, hash-based text representation is applied extensively. One major problem of these abstractions may be too succinct and thus may not be robust enough to withstand intentional attacks. A common attack to this type of representation is to insert a random normal paragraph without any suspicious key- words into unobvious position of an e-mail.

In such a text, if the whole electronic mail content is utilized for hash- based representation, the near-duplicate part of spams cannot be captured. In addition, the fail positive rate (i.e., the rate of classifying hams as spams) may increase because the random part of e-mail content is also involved in e-mail abstraction. On the other hand, hash-based text representation also suffers from the problem is not suit for all languages.

Finally, images and hyperlinks are important clues to spam detection, but both are helpless to be included in hash based text representation. In this paper, we explode to devise a more sophisticated electronic abstraction. Which can more effectively capture the near- duplicate phenomenon of spams. Promoted by the fact that electronic mail users are capable of easily recognizing similar spams by observing the layouts of e-mails, we attempt to represent each e-mail based on the e-mail layout structure. Fortunately, almost all e-mails nowadays are in Multipurpose Internet Mail Extensions (MIME) format with the text/html content- type.

That is, HTML content is available in an e-mail and provides sufficient information about e-mail layout structure. Almost all e-mails nowadays are in Multipurpose Internet Mail Extensions (MIME) format with the text/html content-type. That is, HTML content is available in an e-mail and provides sufficient information about e-mail layout structure. In view of this observation, we propose the specific procedure Structure Abstraction Generation (SAG), which generates an HTML tag sequence to represent each e-mail. Different from previous works, SAG focuses on the e-mail layout structure instead of detailed content text. In this regard, each paragraph of text without any HTML tag embedded will be transformed to a newly defined tag <my text=>.

Definition 1 (<my text=>). <my text=> is a newly defined tag that represents a paragraph of text without any HTML tag embedded.

Since we ignore the semantics of the text, the proposed abstraction scheme is inherently applicable to e-mails in all languages. This significant feature is superior to most existing methods. Once e-mails are represented by our newly devised e-mail abstractions, two e-mails are viewed as near-duplicate if their HTML tag sequences are exactly identical to each other.

Note that even when spammers insert random tags into e-mails, the proposed e-mail abstraction scheme will still retain efficacy since arbitrary tag insertion is prone to syntax errors or tag mismatching, meaning that the appearance of the e-mail content will be greatly altered. Moreover, the proposed procedure SAG also adopts some heuristics to better guarantee the robustness of our approach.

While a more sophisticated e-mail abstraction is intro- duced, one challenging issue arises: how to efficiently match each incoming e-mail with an existing huge spam database. To resolve this issue, we devise an innovative tree structure, SpTrees, to store large amounts of the e-mail abstractions of reported spams, and SpTrees contribute to substantially promoting the efficiency of matching. In the design of the near-duplicate matching scheme based on SpTrees, we aim at reducing the number of spams and tags which are required to be compared.

By integrating above techniques, in this paper, we designa complete spam detection system Collaborative Spam Detection System (Cosdes). Cosdes possesses an efficient near-duplicate matching scheme and a progressive update scheme. The progressive update scheme not only adds in new reported spams, but also removes obsolete ones in the database. With Cosdes maintaining an up-to-date spam database, the detection result of each incoming e-mail can be determined by the near-duplicate similarity matching process. In addition, to withstand intentional attacks, a reputation mechanism is also provided in Cosdes to ensure the truthfulness of user feedback.

To the best of our knowledge, there is no prior research in considering e-mail layout structure to represent e-mails in the field of near-duplicate spam detection.

In summary, the contributions of this paper are as follows:

i. We propose the specific procedure SAG to generate the e-mail abstraction using HTML content in e-mail, an d t h is newly devised abstraction can m or e effectively capture the near-duplicate phenomenon of spams.

ii. We devise an innovative tree structure, SpTrees, to store large amounts of the e-mail abstractions of reported spams. SpTrees contribute to the accomplishment of the efficient near-duplicate matching with a more sophisticated e-mail abstraction.

iii. We design a complete spam detection systemCosdes with an efficient near-duplicate matching scheme and a progressive update scheme. The progressive update scheme enables system Cosdes to keep the most up-to-date information for near- duplicate detection.

The rest of this paper is outlined as follows: In Section II, preliminaries including the definition of near-duplicate and the related works are given. In Section III, we introduce the novel e-mail abstraction scheme. In Section IV, the complete system model of Cosdes is depicted. The experimental results are shown in Section V, and finally, this paper is concluded with Section VI.

II. PRELIMINARIES

In this section, the definition of near-duplicate, in this paper, is presented in Section II.A We then review the related works on spam detection in Section II.B.

II.A) Definition of NearDuplicate

The central idea of near-duplicate spam detection is to exploit reported known spams to block subsequent ones which have similar content. For different forms of e-mail representation, the definitions of similarity between two e-mails are diverse. Unlike most prior works representing e-mails based mainly on content text, we investigate representing each e-mail using an HTML tag sequence, which depicts the layout structure of e-mail, and look forward to more effectively capturing the near-duplicate phenomenon of spams.

Initially, the definition of <anchor> tag is given as follows:

Definition 2 (<anchor>). The tag <anchor> is one type of newly defined tag that records the domain name or the e-mail address in an anchor tag.

For example, the anchor tag <a href="http://arbor.ee.ntu.edu.tw/index.htm"> is transformed to <arbor.ee.ntu.edu.tw>. The anchor tag <a href="mailto:cytseng@arbor.ee.ntu.edu.tw"> is transformed to <cytseng@arbor.ee.ntu.edu.tw>. The purpose of creating the <anchor> tag is to minimize the false positive rate when the number of tags in an e-mail abstraction is short. The less the number of tags in an e-mail abstraction, the more possible that a ham may be matched with known spams and be misclassified as a spam. Therefore, when the number of tags in an e-mail abstraction is smaller than a predefined threshold, for each anchor tag <a>, we specifically record the targeted domain name or e-mail address, which is a significant clue for identifying spams.

On the other hand, in this paper, the detailed definition of near-duplicate is given as follows:

Definition 3 (Near-Duplicate). Let I ¼ ft$_1$; t$_2$; . . . ; t$_i$; . . . ; t$_n$; <my text=>; <anchor>g be the set of all valid HTML tags with two types of newly created tags, <my text=> and <anchor>, included. An e-mail abstraction derived from procedure SAG is denoted as <e$_1$; e$_2$; . . . ; e$_i$; . . . ; e$_m$>, which is an ordered list of tags, where e$_i$ 2 I. The definition of near-duplicate is: "Two e-mail abstractions ff ¼ <a$_1$; a$_2$; . . . ; a$_i$; . . . ; a$_n$> and fi ¼ <b$_1$; b$_2$; . . . ; b$_i$; . . . ; b$_m$> are viewed as near-duplicate if 8a$_i$ ¼ b$_i$ and n ¼ m."

Definition 4 (Tag Length). The tag length of an e-mail abstraction is defined as the number of tags in an e-mail abstraction. Note that we strictly define that two e-mail abstractions are near-duplicate only if they are exactly identical to each other. The major reason is that there are numerous HTML tag patterns appearing commonly and frequently. Partial matching of HTML tag sequences will cause much higher rate of false positive error, and the complexity will be too high to achieve efficient matching. In addition, for further speed-up, while the tag length of an e-mail abstraction is longer, we even apply a looser matching criterion, which does not degrade detection results.

II .B) RelatedWorks

Since the e-mail spam problem is increasingly serious **nowadays, various techniques have been explored to relieve the problem. Based** on what features of e-mails are being used, previous works on spam detection can be generally classified into three categories: 1) content-based methods, 2) noncontent-based methods, and 3) others. Initially, researchers analyze e-mail content text and model this problem as a binary text classification task. Representatives of this category are Naive Bayes [14], [20] and Support Vector Machines (SVMs) [1], [10], [15], [27] methods. In general, Naive Bayes methods train a probability model using classified e-mails, and each word in e-mails will be given a probability of being a suspicious spam keyword. As for SVMs, it is a supervised learning method, which possesses outstanding performance on text classification tasks. Traditional SVMs [10] and improved SVMs [1], [15], [27] have been investigated. While above conventional machine learning techniques have reported excellent results with static data sets, one major disadvantage is that it is cost-prohibitive for large-scale applications to constantly retrain these methods with the latest information to adapt to the rapid evolving nature of spams. The spam detection of these methods on the e-mail corpus with various language has been less studied yet. In addition, other classification techniques, including markov random field model [3], neural network [6] and logic regression [2], and certain specific features, such as URLs [26] and images [19], [29] have also been taken into account for spam detection.

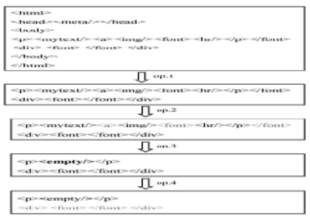
The other group attempts to exploit non content information such as e-mail header, e-mail social network [4], [28], and e-mail traffic [5], [9] to filter spams. Collecting notorious and innocent sender addresses (or IP addresses) from e-mail header to create black list and white list is a commonly applied method initially. MailRank [4] examines the feasibility of rating sender addresses with algorithm PageRank in the e-mail social network, and in [28], a modified version with update scheme is introduced. Since e-mail header can be altered by spammers to conceal the identity, the main drawback of these methods is the hardness of correctly identifying each user. In [5], [9], the authors intend to analyze e-mail traffic flows to detect suspicious machines and abnormal e-mail communication. It is noted that these approaches have to operate in coordination with other complementary methods to gain better results. Moreover, some researchers consider com- bining the merits of several techniques [2], [13], [18]. Even though the performance of classifier integration seems prominent, there is still no conclusion on what is the best combination. In addition, how to efficiently update the whole included classifiers is another unsolved issue.

On the other hand, collaborative spam filtering with near-duplicate similarity matching scheme has been stu- died extensively in recent years. Regarding collaborative mechanism, P2P-based architecture [8], [12], [31], centra- lized server-based system [16], [21], [22], [30], and others [17], [23] are generally employed. Note that no matter which mechanism is applied, the most critical factor is how to represent each e-mail for near-duplicate matching. The e-mail abstraction not only should capture the near-duplicate phenomenon of spams, but should avoid accidental deletion of hams. In [30], the first N hash values of each length L substring are used as vector representation of the e-mail. In [7], [8], [17], a 32-byte code derived from a variation of Nilsimsa digest technique is utilized to represent the distribution of word trigrams in e-mail. In [23], [24], [25], the authors improve the open digest technique [7] by representing each e-mail with multiple digests produced from the strings of fixed length sampled at randomized positions within e-mail. In [12], [31], a feature vector of a block text fingerprint generated from the set of checksums of each length L substring is exploited.

```
Procedure SAG
Input:  the email with text/html content-type,
        the tag length threshold (Lth_short) of the short email
Output: the email abstraction (EA) of the input email
    // Tag Extraction Phase
    Transform each tag to <tag.name>;
    Transform each paragraph of text to <mytext/>;
    AnchorSet = the union of all <anchor>;
    EA = the concatenation of <tag.name>;
    Preprocess the tag sequence of EA;
    // Tag Reordering Phase
    for (each tag of EA) // pn: position number
        tag.new_pn = ASSIGN_PN (EA.tag_length, tag.pn);
        Put the tag to the position tag.new_pn;
    EA = the concatenation of <tag.name> with new_pn;
    // <anchor> Appending Phase
    if (EA.tag_length < Lth_short)
        Append AnchorSet in front of EA;
    return EA;
End
```

Fig. 1 . Algorithmic form of procedure SAG

In [22], the authors make use of spam-vocabulary patterns produced by Teiresias pattern discovery algorithm. In [16], the I-Match signature determined by a set of unique terms shared by spams and the I-Match lexicon is put to use.In [21], the content similarity of e-mails computed using extracted words is measured. It is noted that most existing methods generate e-mail abstractions based mainly on content text. However, randomized and normal paragraphs are commonly inserted in spams nowadays, and thus if an e-mail abstraction is generated by the whole content text, the near-duplicate part of spams cannot be captured. Moreover, generating e-mail abstraction with the content text also suffers from the problem of not being applicable to all languages.


```
<html>
<head><meta/></head>
<body>
<p><mytext/><a><img/><font><hr/></p></font>
<div> <font> </font> </div>
</body>
</html>
```

```
<p><mytext/><a><img/><font><hr/></p></font>
<div><font></font></div>
```

```
<p><mytext/><a><img/><font><hr/></p></font>
<div><font></font></div>
```

```
<p><empty/></p>
<div><font></font></div>
```

```
<p><empty/></p>
<div> <font> </font> </div>
```

Fig. 2. An example of the preprocessing step in Tag Extraction Phase of procedure SAG.

In light of the above problems, it deserves further studies to design a better e-mail abstraction approach that is more robust to withstand intentional attacks.

III .E-MAIL ABSTRACTION SCHEME

In this section, a novel e-mail abstraction scheme is introduced. In Section III.A, procedure SAG is presented to depict the generation process of an e-mail abstraction. The devised data structures SpTable and SpTrees are illustrated in Section III.B. Finally, the robustness issue is discussed in Section III .c.

III.A Structure Abstraction Generation

We propose the specific procedure SAG to generate the e-mail abstraction using HTML content in e-mail. SAG is elaborated with the example of Fig. 3, and the algorithmic form of SAG is outlined in Fig. 1. Procedure SAG is composed of three major phases, Tag Extraction Phase, Tag Reordering Phase, and <anchor> Appending Phase. In Tag Extraction Phase, the name of each HTML tag is extracted, and tag attributes and attribute values are eliminated. In addition, each paragraph of text without any tag embedded is transformed to <mytext=>. In lines 4-5, <anchor> tags are then inserted into AnchorSet, and the first 1,023 valid tags are concatenated to form the tentative e-mail abstraction. Note that we retain only the first 1,023 tags as the tag sequence. The main reason is that the rear part of long e-mails can be ignored without affecting the effectiveness of near-duplicate matching. Sub- sequently, in line 6 of Fig. 1, we preprocess the tag sequence of the tentative e-mail abstraction. One objective of this preprocessing step is to remove tags that are common but not discriminative between e-mails. The other objective is to prevent malicious tag insertion attack, and thus the robust- ness of the proposed abstraction scheme can be further enhanced.

The following sequence of operations is performed in the preprocessing step.

1. Front and rear tags (as shown in the gray area of the example e-mail in the top of Fig. 3) are excluded.

2.Nonempty tags[2] that have no corresponding start tags or end tags are deleted. Besides, mismatched nonempty tags are also deleted.

3. All empty tags[2] are regarded as the same and are replaced by the newly created <empty=> tag.Moreover, successive <empty=> tags are pruned and only one <empty=> tag is retained

4. The pairs of nonempty tags enclosing nothing are removed.

Example 1. Consider the example e-mail abstraction in Fig. 2 that has been produced through the execution of lines 1-5 in procedure SAG. The first operation of the reprocessing step is to remove tags which are in front of the <body> tag and which are in rear of the <=body> tag. With regard to operation 2, since there is no end tag of <a>, this tag is deleted. Besides, the tags <font> and < =font > are also deleted because the position of <=font> is incorrect. Note that we can utilize the stack data structure to determine whether nonempty tags are mismatched. After that, empty tags are transformed to < empty=> in operation 3. More- over, since <mytext=><img=><hr=> appear consecu- tively, only one <empty=> tag is retained. Finally, <div><font><=font><=div> are removed due to the lack of content.

The middle part of Fig. 3 shows an example of a tentative e-mail abstraction and AnchorSet (i.e., <spam:com>) derived from Tag Extraction Process. On purpose of accelerating the near-duplicate matching process, we reorder the tag sequence of an e-mail abstrac- tion in Tag Reordering Phase. Note that since the arrange- ment of HTML tags is regular and in pairs, various sequential patterns of tags are contained in e-mails. In the worst case, if we consider two e-mail abstractions which have the same tag length and differ only in their last tags, the difference cannot be detected until the last tags are compared. To handle this problem, we destroy the regularity by rearranging the order of tag sequence to lower the number of tag comparisons. Note that this process ensures that the newly assigned position numbers of e-mail abstractions with the same number of tags are completely identical. As such, the matching process can be accelerated without violating the definition of near-duplicate in this paper. In lines 8-11 of Fig. 1, each tag is assigned a new position number by function ASSIGN_PN (PN denotes for

Position Number) with following expressions,

pffff b ¼ d Le; r ¼ ðP N$_{orig}$ À 1Þ%b; q ¼ bðPN$_{orig}$À 1Þ=bc þ 1;

P N$_{new}$ ¼ ðb Â rÞ þ ðb À q þ 1Þ;

where L is the tag length of an e-mail abstraction, and P N$_{orig}$ is the original position number. Variable b is the number of buckets. Variable r indicates which bucket should be placed, and variable q is the number of shift counts from the end of this bucket. Fig. 3 demonstrates the assignment of the first six tags. The final e-mail abstraction is the concatenation of all tags with new position numbers (the vacant positions, e.g., positions 9 and 13 in Fig. 3, are ignored). Additionally, if the tag length of an e-mail abstraction is smaller than a predefined tag length threshold (set as 16 in the experi- ments) of the short e-mail, the tags in AnchorSet will be appended in front of the e-mail abstraction. The main objective of appending <anchor> tags is to reduce the probability that a ham is successfully matched with reported spams when the tag length of an e-mail abstraction is short.

An example e-mail abstraction produced by procedure SAG is shown in the bottom of Fig. 3

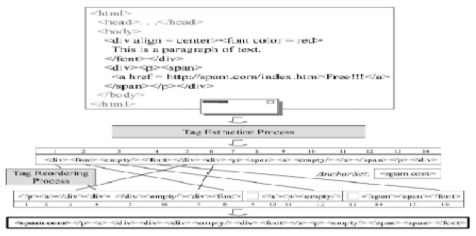

<html>
<head>. . .</head>
<body>
<div align = center><font color = red>
This is a paragraph of text.
</font></div>
<div><p><span>
<a href = http://spam.com/index.htm>Free!!!</a>
</span></p></div>
</body>
</html>
Tag Extraction Process
<div><font><empty/></font></div><div><p><span><a><empty/></a></span></p></div>
Tag Reordering Process
AnchorSet:
<spam.com>
</p><a></div><div></div><empty/><div><font></a><p><empty/></span><span></font>
<spam.com></p><a></div><div></div><empty/><div><font></a><p><empty/></span><span></font>

Fig. 3. An example procedure flow of SAG

III.B. Design of SpTrees

One major focus of this work is to design the innovative data structure to facilitate the process of near-duplicate matching. SpTable and SpTrees (sp stands for spam) are proposed to store large amounts of the e-mail abstractions of reported spams. As shown in Fig. 4, several SpTrees are the kernel of the database, and the e-mail abstractions of collected spams are maintained in the corresponding SpTrees. According to Definition 3, two e-mail abstractions are possible to be near-duplicate only when the numbers of their tags are identical. Thus, if we distribute e-mail abstractions with different tag lengths into diverse SpTrees, the quantity of spams required to be matched will decrease. However, if each SpTree is only mapped to one single tag length, it is too much of a burden for a server to maintain such thousands of SpTrees. In view of this concern, each SpTree is designed to take charge of e-mail abstractions within a range of tag lengths. As can be seen in Fig. 4, SpTable is created to record overall information of SpTrees.

The i_{th} column of SpTable links to the root of SpTree_i by a pointer, and e-mail abstractions with tag lengths ranging from 2^{i} to 2^{i}þ^{1} À 1 belong to SpTree_i. Regarding how an e-mail abstraction is stored in SpTree, Fig. 5 gives an example with the same e-mail abstraction derived from Fig. 3. An e-mail abstraction is segmented into several subsequences, and these subsequences are consecutively put into the corresponding nodes from low levels to high levels. As such, an e-mail abstraction is stored in one path from the root node to a leaf node of SpTree, and hence the matching between a testing e-mail and known spams is processed from root to leaf. As shown in Fig. 5, the example e-mail abstraction is stored in a path from the root node a to the leaf node d. The primary goal of applying the tree data structure for storage is to reduce the number of tags required to be matched when processing from root to leaf. Since only subsequences along the matching path from root to leaf should be compared, the matching efficiency is substantially increased. Note that if each type of HTML tag determines a branch direction (i.e., the tree degree will be the number of HTML tag types) and each level of SpTree contains merely one HTML tag (i.e., the tree height will be the tag length of the longest e-mail abstraction), the number of tag comparisons will be minimum.

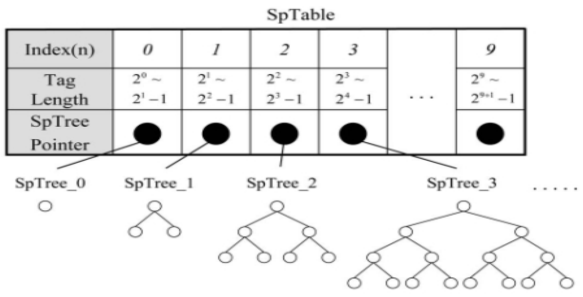


Fig. 4. The data structures of SpTable and SpTrees.

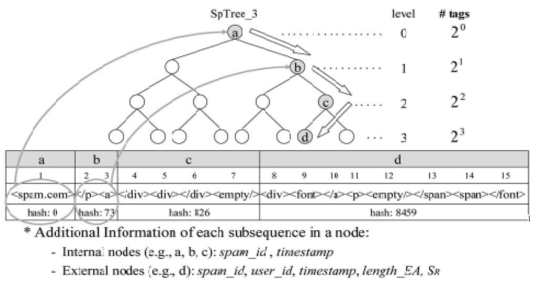

SpTree_3
level
# tags
a
b
c
d
0
1
2
3
2^0
2^1
2^2
2^3
<spam.com>
</p><a>
</div><div></div><empty/>
<div><font></a><p><empty/></span><span></font>
hash: 0
hash: 73
hash: 826
hash: 8459
* Additional Information of each subsequence in a node:
- Internal nodes (e.g., a, b, c): spam_id , timestamp
- External nodes (e.g., d): spam_id, user_id, timestamp, length_EA, Sʀ

Fig. 5. The illustration of SpTree_3 with an example e-mail abstraction.

However, it is infeasible because the degrees and the heights of SpTrees will be too large, and SpTrees will be extremely unbalanced To achieve efficient matching with balanced tree struc- ture, SpTrees are designed to be binary trees. The branch direction of each SpTree is determined by a binary hash function. If the first tag of a subsequence is a start tag (e.g., <div>), this subsequence will be placed into the left child node. A subsequence whose first tag is an end tag (e.g., <=div>) will be placed into the right child node. Since most HTML tags are in pairs and the proposed e-mail abstraction is reordered in procedure SAG, subsequences are expected to be uniformly distributed. Moreover, on level i of each SpTree (with the root on level 0), each node stores subsequences whose tag lengths are equal to 2^{i}. For instance, as shown in Fig. 3, the subsequence <spam:com> (whose tag length is 2^{0}) is placed into level 0, the subsequence <=p><a> (whose tag length is 2^{1}) is placed into level 1, and so forth. Note that since SpTree_i takes charge of e-mail abstractions with tag lengths ranging from 2^{i} to 2^{i}þ^{1} À 1, based on the above-mentioned arrangement, the last subsequence of each e-mail abstraction in an SpTree will be stored in the leaf nodes on the same level. Also note that the tag lengths of subsequences stored in leaf nodes of level j range from 1 to 2^{j}. As described in Section 3.1, we design that the longest length of an e-mail abstraction is 1,023, meaning that there are totally 10 SpTrees (from SpTree_0 to

SpTree_9) in our database while the proposed arrangement is applied. In addition, to further accelerate the process of matching, we employ a hash function to map each subsequence to an integer. The key idea is that only subsequences which look like the testing subsequence should be exactly matched. The hash function is defined as follows:

hashðseqÞ ¼
fðseq½0ŠÞ Ã
2mÀ1 þ
fðseq½1ŠÞ Ã
2mÀ2

þ Á Á Á þ fðseq½m À 1ŠÞ Ã 20;

where m is the number of tags in this subsequence and seq½nŠ denotes the tag type of the n_{th} tag. The function f converts each type of tag to a unique integer. Moreover, for the subsequence which contains more than eight tags, we just use the first eight tags to generate the hash value (i.e., m 8).

With the hash function, most subsequence matching is transformed to the integer matching, and hence the complexity of matching process can be substantially reduced. Overall, the advantageous features of this innovative arrangement are as follows: 1) The height of an SpTree is equal to blg Lc, where L is the tag length of the longest e- mail abstraction in this SpTree. 2) Owing to the fact that parent nodes store less number of subsequences than children nodes, we design that longer subsequences are put into higher levels (the tag length of the subsequence on level i is 2^i). Thus, the number of tags matched from root to leaf is markedly decreased. Moreover, with the hash function, the matching efficiency is substantially increased. 3) The numbers of tags stored in the nodes of an SpTree are expected to be similar, and hence SpTrees are balanced binary trees. Assume that there are N e-mail abstractions in SpTree_i and subsequences on the same level are uniformly distributed. For each node on level j, there are $\underline{N}j$ subsequences (the number of nodes on level j 2is 2^j) whose tag lengths are equal to 2^j.

Then The number of tags stored in each node is $\underline{N}j$ Â 2^j ¼ N, which is not 2correlated with j. It is noted that the number of tags ineach leaf node is less than N because not all subse- quences in leaf nodes are with the longest allowed tag length.

On the other hand, as shown in the bottom of Fig. 5, certain additional information is required and kept in each subsequence of a node. For subsequences in internal nodes, spam id and timestamp are included. For subsequences in external nodes, spam id, user id, timestamp, length EA (the length of the e-mail abstraction), and S_R (the reputation score) are involved.

III.C Robustness Issue

The main difficulty of near-duplicate spam detection is to withstand alicious attack by spammers. Prior approaches generate e-mail abstractions based mainly on hash-based content text. These methods primarily differ in what granularity is used as the input of the hash function. For example, the authors in [16], [21], [22] extract words or terms to generate the e-mail abstraction. Besides, substrings extracted by various techniques are widely employed in [7], [8], [12], [17], [23], [25], [30], [31]. However, this type of e-mail representation inherently has following disadvantages. First, the insertion of a randomized and normal paragraph can easily defeat this type of spam filters. Moreover, since the structures and features of different languages are diverse, word and substring extraction may not be applic-able to e-mails in all languages. Concretely speaking, for instance, trigrams of substrings used in [7], [8], [17] are not suitable for nonalphabetic languages, such as Chinese.

In this paper, we devise a novel e-mail abstraction scheme that considers e-mail layout structure to represent e-mails. To assess the robustness of the proposed scheme, we model possible spammer attacks and organize these attacks as following three categories. Examples and the outputs of preprocessing of procedure SAG.

III.C.1 Random Paragraph Insertion

This type of spammer attack is commonly used nowa- days. The normal contents without any *advertisement keywords are inserted to confuse text based spam filtering techniques. It is noted* that our scheme transforms each paragraph into a newly created tag <mytext=>, and consecutive empty tags will then be transformed to <empty=>. As such, the representation of each random inserted paragraph is identical, and thus our scheme is resistant to this type of attack.

III.C.2 Random HTML Tag Insertion

If spammers know that the proposed scheme is based on HTML tag sequences, random HTML tags will be inserted rather than random paragraphs. On the one hand, arbitrary tag insertion will cause syntax errors due to tag mismatch-ing. This may lead to abnormal display of spam content that spammers do not wish this to happen. On the other hand, procedure SAG also adopts some heuristics (as depicted in Section III.A) to deal with the random insertion of empty tags and the tag mismatching of nonempty tags. Two example outputs and the details of each step can be found in Fig. 2. With the proposed method, most random inserted tags will be removed, and thus the effectiveness of the attack of random tag insertion is limited. We shall verify this inference in Section V.4.

III.C.3 Sophisticated HTML Tag Insertion

Suppose that spammers are more sophisticated, they may insert legal HTML tag patterns. The tag patterns that do conform to syntax rules are inserted, they will not be eliminated. However, although some crafty tricks may be conceivable, it is not intuitive for spammers to generate a large number of spams with completely distinct e-mail layout structure.

Note that due to space limitation, we are not able to discuss all possible situations. Nevertheless, representing e- mails with layout structure is more robust to most existing attacks than text-based approaches. Even though new attack has been designed, we can react against it by adjusting the preprocessing step of procedure SAG. On the other hand, our approach extracts only HTML tag sequences and transforms each paragraph with no tag embedded to <mytext=>, meaning that the proposed abstraction scheme can be applied to e-mails in all languages without modifying any components.

IV. Collaborative Spam Detection System

A complete collaborative spam detection system Cosdes isintroduced in this section. The system model of Cosdes is given in Section IV.1. We then elaborate the processing handlers of Cosdes in Section IV.2. Finally, we describe the reputation mechanism of Cosdes in Section IV.3.

IV.1 System Model of Cosdes

The system model of Cosdes is illustrated in Fig. 6, and the a lg o r i th m ic fo r m is o ut l in e d i n F ig . 7. Ini ia l ly , th r e e parameters, T_m (the maximum time span for reported spams being retained in the system), T_d (the time span for triggering Deletion Handler), and S_{th} (the score threshold for determining spams) should be given for Cosdes. Before starting to do the spam detection, Cosdes collects feedback spams for time T_m in advance to construct an initial database. Three major modules, Abstraction Generation Module, Database Maintenance Module, and Spam Detection Module, are

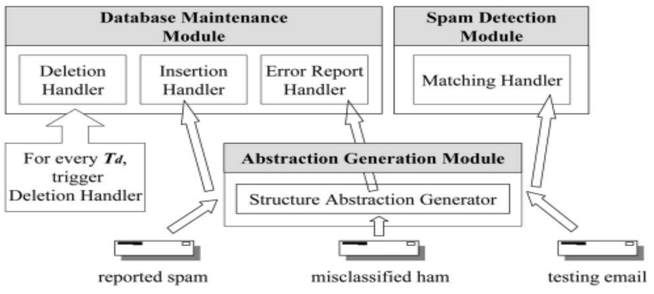


Fig. 6. System model of Cosdes.

V. Performance

To assess the feasibility of system Cosdes, we conduct several experiments to explore its efficiency and detection results. The real spam data sets used in the experiments are from the e-mail servers of Computer Center in National Taiwan University, which has over 30,000 students. Since the ground tr u t h o f r e a l e - m a i l s tr e a m s i s un a v a i l a b l e, s p a m s a r e extracted from the well-known existing system, SpamAssas- sin.[3] Concerning hams, we not only include public data sets (around 4,000 e-mails) provided by SpamAssassin,[4] but also obtain from volunteers. There are about 60,000 spams per day and a set of 7,000 or so hams in the data set. Note that numerous related works have evaluated the proposed methods with static databases. However, to access the performance of spam detection system with near-duplicate matching scheme, real e-mail streams are more appropriate than static data sets. Therefore, in this paper, we use university-scale e-mail streams as the experimental data sets to better simulate the e-mail environment. On the other hand, three representative approaches [7], [24], [30] of near- duplicate spam detection are employed for comparison. The authors of [8], [17] also adopt the same e-mail representation approach as in [7] but with different sharing mechanisms. For ease of presentation, Damiani's work is abbreviated as Digest. Sarafijanovic's work is abbreviated as MultiDigest, and Yoshida's work is abbreviated as Density. It is worth mentioning that Sarafijanovic's work [24] improves Damiani's one [7] by representing each e-mail with multiple digests produced from the strings of fixed length sampled at randomized positions within e-mail. The processes of generating each digest in Digest and MultiDigest are identical. Although Sarafijanovic's work claims that using multiple digests can enhance the robustness of near-duplicate spam detection system, these two works have not been

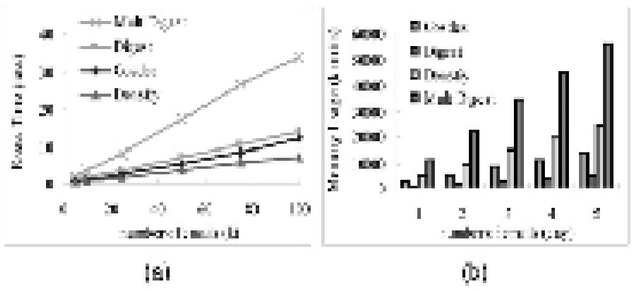


Fig7: The execution time and the memory usage of generating e-mail abstractions with the number of e-mails varied. (a) Execution time of e- mail representation. (b) Memory usage of e-mail representation.

Besides, Yoshida's work considers maintaining a direct-mapped cache to facilitate the process of matching. In the experiments of his work [30], there are 10 million spams in the database and 10 percent of hash values are copied in the cache. However, to fairly compare the detection performance, the cache mechanism of Yoshida's work is not included in our experiments, meaning that all spams in the database are used for detection. We implement Cosdes and comparative techniques with C++ language, and the programs are executed in Windows XP professional platform with Pentium 4—3GHz CPU and 1GB RAM. The programs of Digest and MultiDigest are implemented with source codes shared by original authors of [7] and [24]. Initially, the efficiency and the space usage of generating e-mail representation are investigated in Section V.1. We compare and analyze the detection results of four approaches in SectionV.2. The detailed efficiency analysis is presented in Section V.3. Finally, Section V.4 simulates the spammer attack of random HTML tag insertion.

V.1 E-mail Representation

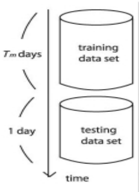
The processing time of the generation of e-mail abstractions with the number of e-mails varied is shown in Fig. 7a. As mentioned in Section II.B, most prior works on near-duplicate spam detection represent e-mails based mainly on content text. Cosdes is the first work to attempt to utilize HTML content, which depicts the layout structure of an e-mail, for representation. Regarding Digest, word trigrams are ex- tracted consecutively along the whole e-mail content. As for Density, the authors acquire the first N (in [30], N is set as 100) hash values of each length L substring with a fixed-size window sliding through an e-mail. As can be seen in Fig. 10a, Density takes the least time since it computes only the first N hash values. As for Digest, hash values of word trigrams in the whole e-mail are required to be computed. Regarding MultiDigest, each e-mail is represented by a set of digests, meaning that each e-mail is separated into multiple strings with fixed length (in [24], the length is set as 60 characters) sampled at randomized positions. Although the length of each string is much shorter, the overall complexity still increases since each e-mail has multiple strings needed to be processed. Fig. 10a shows that the processing time of MultiDigest is about four times longer than that of Digest, and thus MultiDigest takes the longest time among four approaches. In procedure SAG of Cosdes, HTML tags are extracted and each paragraph of text is transformed to the newly created tag <mytext=>. If only these operations are performed, Cosdes will be the most efficient. However, sequence preprocessing and tag reordering are also executed in Cosdes, and therefore, the execution time slightly increases.

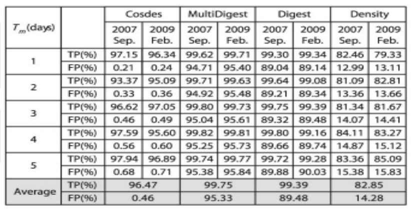
Overall, the cost of generating e-mail abstractions of four approaches is very low. With regard to the space issue, Fig. 7 b shows the memory usage of four approaches with the number of e-mails varied in the database. Note that we estimate a hash value or an integer number as one unit of memory usage, which approximates to 2 bytes.

Digest represents each e-mail with a 32-byte code, which is equal to 16 units. As mentioned above, MultiDigest utilizes multiple 32-byte codes for the representation of each e-mail. In the experi- mental data set, the average number of digests in each e-mail is approximately 12, and thus the memory usage of MultiDigest is larger than that of Digest by around 12 times. Regarding Density, N hash values, that is, 100 units, are the representation of each e-mail. However, since some e-mails are too short to be extracted N hash values, the average memory usage of each e-mail in Density is smaller that 100 units. On the other hand, a sequence of HTML tags is the representation of each e-mail in Cosdes . We can replace each type of tag with a unique integer, and thus each tag can be viewed as 1 unit. It is calculated that the average length of the HTML tag sequence produced by procedure SAG of Cosdes is approximately 35. In addition, as stated in Section III.B, additional information is required in SpTable and SpTrees of Cosdes. We include this memory overhead as well. It can be observed in Fig. 10b that the memory usage of Digest is the least, and MultiDigest uses the largest memory space among four approaches. As for Cosdes, the memory usage is larger than that of Digest by two to four times. Although a fairly succinct e-mail abstraction can greatly reduce the overhead of near-duplicate matching, we can find in the following section that the effectiveness of Digest cannot be validated.

V.2 Accuracy Evaluation

In this section, we evaluate the detection performance of Cosdes and three competitive approaches. The most important requirement for a spam detection system is the capability to resist malicious attack that evolves continu- ously. To examine this capability, two recent streams of spams (collected from National Taiwan University in September 2007 and February 2009) are utilized as the experimental data sets. Regarding the language of content text, 80 percent of all e-mails are in Chinese, and 15 percent of them are in English. The minority of e-mails are in Japanese, French, and so forth. Since Chinese is a nonalphabetic language and English is an alphabetic one, the data set used in the experiments can verify the effectiveness of spam detection system with different kinds of languages to a certain extent. Before a system starts to do the near-duplicate spam detection, a set of known spams is inserted into the system. We consider situations with the parameter T_m varied from one to five days. That is, as shown in the left side of Fig. 8,





T_m(days)		Cosdes		MultiDigest		Digest		Density	
		2007 Sep.	2009 Feb.	2007 Sep.	2009 Feb.	2007 Sep.	2009 Feb.	2007 Sep.	2009 Feb.
1	TP(%)	97.15	96.34	99.62	99.71	99.30	99.34	82.46	79.33
	FP(%)	0.21	0.24	94.71	95.40	89.04	89.14	12.99	13.11
2	TP(%)	93.37	95.09	99.71	99.63	99.64	99.08	81.09	82.81
	FP(%)	0.33	0.36	94.92	95.48	89.21	89.34	13.36	13.66
3	TP(%)	96.62	97.05	99.80	99.73	99.75	99.39	81.34	81.67
	FP(%)	0.46	0.49	95.04	95.61	89.32	89.48	14.07	14.41
4	TP(%)	97.59	95.60	99.82	99.81	99.80	99.16	84.11	83.27
	FP(%)	0.56	0.60	95.25	95.73	89.66	89.74	14.87	15.12
5	TP(%)	97.94	96.89	99.74	99.77	99.72	99.28	83.36	85.09
	FP(%)	0.68	0.71	95.38	95.84	89.88	90.03	15.38	15.83
Average	TP(%)	96.47		99.75		99.39		82.85	
	FP(%)	0.46		95.33		89.48		14.28	

Fig. 8. Performance of detection result

the detection results are produced by inserting spams within T_m days first, and thenthe following one-day spams are tested. Note that eachspam is inserted into the database after the process of matOn the other hand, the entire set of hams is tested.

V.3 Efficiency Analysis

In the succeeding experiments, we initially examine theefficiency of near-duplicate matching. Owing to the fact that each incoming e-mail has to be matched with a huge spam database, the efficiency of near-duplicate matching is crucial to a collaborative spam detection system. On evaluation of matching performance, we consider the situation of matching with the number of e-mails varied while there are identical e-mails in the database. As shown in Fig. 13a, the execution time of Digest is minimal since only two 32-byte codes of each pair of e-mails have to be compared. However, as shown in Fig. 12a, the detection results of Digest are not satisfied even if their matching processes are very fast.

As for MultiDigest, it is defined in [24] that the similarity measure between two e-mails is the maximum number of equal bits over all pairs of digests. According to this definition, processing all pairs of digests between two e-mails requires n^2 comparisons, where n is the average number of digests in an e-mail. It is calculated that n is close to 12 in our data set, meaning that the matching time of MultiDigest is larger than that of Digest by over 100 times. This indicates that the matching process of MultiDigest is the least efficient among four approaches. Regarding Density, the sequences of the first 100 hash values, which are longer than Digest and Cosdes, are matched, and therefore Density takes more time than Digest and Cosdes. It is noted that the authors in [30] propose a cache mechanism to avoid matching each e-mail with the huge database. The cache mechanism enables Density to markedly enhance the efficiency of matching. However, this will degrade the detection performance while the spam database is not as huge as in [30].

To fairly compare the performance, we ignore the cache mechanism of Density in this experiment. On the other hand, Cosdes has to match a longer sequence than Digest, and in essence Cosdes requires more time for matchingwe conduct the efficiency investigation of Cosdes on inserting e-mail abstractions into the database and deleting outdated spams from the database. Owing to the fact that competitive approaches, Digest and Density, did not isolate insertion parts from the systems and did not take account of deletion, we only study the performance of Insertion Handler and Deletion Handler in Cosdes. the execution time of Insertion Handler of Cosdes with the number of e-mails varied. The execution time grows linearly and costs merely 3.5 seconds for inserting 100,000 spams into the database. Moreover, as can be seen in Fig. 10a, the process of generating 100,000 e-mail abstractions costs about 10 seconds. On the other hand, the performance of Deletion Handler.

We evaluate the execution time of deleting spams in one day while the number of e-mails in the database varied. The main purpose of this experiment is to examine whether the efficiency of deletion will be influenced by the amount of e- mails stored in SpTrees. It is shown that the deletion process costs only 2 to 3 seconds in each situation, and the execution time slightly increases with the amount of e-mails. Therefore, we can observe that both the processes of insertion and deletion in Cosdes are efficient and incur very little overhead.

To further evaluate the proposed e-mail abstraction scheme, we consider the sequence preprocessing step and the reordering step of procedure SAG. The primary objective of the sequence preprocessing is to prevent malicious tag insertion attack, and thus the robustness of Cosdes can be enhanced. Fig. 15a shows that generating e-mail abstractions with the sequence preprocessing step leads to little increment of execution time. Although the detection results in Fig. 12b can be inferred that spammers still do not intend to obfuscate HTML content, this protection process enables Cosdes to perform more robustly in the future. On the other hand, the main purpose of the reordering step is to differentiate e-mails with similar tag sequences in the earlier stage of matching in each situation. True positive rate (i.e., TP, a real spam is classified as a spam) and false positive rate (i.e., FP, a real ham is misclassified as a spam) are listed.

As can be seen in Fig. 12a, Cosdes reports 96.47 percent TP rate and 0.46 percent FP rate on average, which has the most outstanding performance. The TP rate of Digest is extremely high but the FP rate is unacceptable. In order to accelerate the process of near-duplicate matching, only a 32-byte code is used in Digest to represent each e-mail. Moreover, as defined in [7], two e-mails are determined as near-duplicate if more than 182 bits of their 32-byte (i.e., 256 bits) codes have the same value. It that as the size of spam database is large, the 32-byte code is not discriminative to clearly distinguish each e-mail, and thus hams are easily mismatched with known spams. As for MultiDigest, although the authors claim in [24] that using multiple digests to represent each e-mail can be more robust against increased obfuscation effort by spammers, the FP rate of MultiDigest is even worse than that of Digest as the size of spam database is large. This is owing to the reason that MultiDigest separates each e-mail into a set of short strings. As long as one digest in the huge spam database is similar to one of digests in the testing e-mail, this e-mail will be classified as a spam. In [7] and [24], there are only 2,500 spams and 2,500 hams in the data set, which might not suffice to reflect the real situation. In addition, the effective-ness of Digest and MultiDigest has not been validated by real e-mail streams. On the other hand, the effectiveness of Density has been evaluated in [30] with 10 million spams in the database. One problem of Density is that a huge number of known spams are required to make the proposed cache mechanism of Density work well. However, in our experi- ments, even though we do not consider the cache mechan- ism of Density and all reported spams are used for near- duplicate spam detection, the effectiveness of Density on more recent e-mail streams cannot be validated. Moreover, several parameters should be given for Density and be adjusted according to different environments. Since the authors in [30] do not provide the parameter tuning method, in this experiment, we follow the same setting as in [30] and obtain the results .Note that various tricks targeting at nullifying the approaches of hash-based text representation have been increasingly employed recently. Besides, most spams in our data set are in Chinese, which is a nonalphabetic language. However, Digest and Multi Digest generate hash values with trigrams of substrings.

VI. CONCLUSION

In the field of collaborative spam filtering by near-duplicate detection, a superior e-mail abstraction scheme is required to more certainly catch the evolving nature of spams. Compared to the existing methods in prior research, in this paper, we explore a more sophisticated and robust e-mail abstraction scheme, which considers e-mail layout structure to represent e-mails. The specific procedure SAG is proposed to generate the e-mail abstraction using HTML content in e-mail, and this newly-devised abstraction can more effectively capture the near-duplicate phenomenon of spams. Moreover, a complete spam detection system Cosdes has been designed to efficiently process the near-duplicate matching and to progressively update the known spam database. Consequently, the most up-to-date information can be invariably kept to block subsequent near-duplicate spams. In the experimental results, we show that Cosdes significantly outperforms competitive approaches, which indicates the feasibility of Cosdes in real-world applications.

VII.REFERENCES

1. Anti-Spam (CEAS), 2007.
2. A.C. Cosoi, "A False Positive Safe Neural Network; The Followers of the Anatrim Waves," Proc. MIT Spam Conf., 2008.
3. E. Damiani, S.D.C. di Vimercati, S. Paraboschi, and P. Samarati, "P2P-Based Collaborative Spam Detection and Filtering," Proc. Fourth IEEE Int'l Conf. Peer-to-Peer Computing, pp. 176-183, 2004.
4. T.R. Lynam and G.V. Cormack, "On-Line Spam Filter Fusion," Proc. 29th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR), pp. 123-130, 2006.
5. K.M. Schneider, "Brightmail URL Filtering," Proc. MIT Spam Conf., 2004.
6. Z. Wang, W. Josephson, Q. Lv, and K.L.M. Charikar, "Filtering Image Spam with Near-Duplicate Detection," Proc. Fourth Conf. Email and Anti-Spam (CEAS), 2007.

AUTHOR-1 BIOGRAPHI

SIVANNARAYANA NERELLA, M.Tech (IT),
Final year student,
NALANDA INSTITUTE OF ENGINEERING AND TECHNOLOGY(NIET),SATTENAPALLI(M.D.), GUNTUR(D.T.),

AUTHOR-2 BIOGRAPHI

K.RAJANI DEVI Assoc. Professor,
H.O.D., Department of IT, N.I.E.T.
Sattenapalli (M), Guntur (Dist.), A.P.