High-Speed Detection of Unsolicited Bulk Emails

Sheng-Ya Lin, Cheng-Chung Tan, Jyh-Charn Liu, Michael Oehler[+]

Computer Science Department, Texas A&M University
{shengya, jonastan, liu}@cs.tamu.edu

[+]National Security Agency
mjo@tycho.ncsc.mil

ABSTRACT

We propose a *Progressive Email Classifier* (PEC) for high-speed classification of message patterns that are commonly associated with unsolicited bulk email (UNBE). PEC is designed to operate at the network access point, the ingress between the Internet Service Provider (ISP) and the enterprise network; so that a surge of UNBE containing fresh patterns can be detected before they spread into the enterprise network. A real-time *scoreboard* keeps track of detected *feature instances* (FI) based on a *scoring* and *aging* engine, until they are considered either from valid or UNBE sources. A FI of a valid email is discarded, but an anomalous one is passed to a *blacklist* to control (e.g., block or defer) subsequent emails containing the FI.

The anomaly detector of PEC can be used at different protocol layers. To gain some insights on the performance of PEC, we implemented PEC and integrated it with the *sendmail* daemon to detect anomalous URL links from email streams. Arbitrarily chosen on-line texts and URL links extracted from a corpus of spamming-phishing emails were used to compose testing emails. Experimental results on a Xeon based server show that PEC can handle 1.2M score/age updates, parse 0.9M URL links (of average size 30 bytes) for hashing and matching, and parsing of 25,000 email bodies of average size 1.5kB per second. The *lossy* detection system can be easily scaled by progressive selection of detection features and detection thresholds. It can be used alone or as an early screening tool for an existing infrastructure to defeat major UNBE flooding.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General – *Security and Protection;* C.4 [**Performance of Systems**]**:** *Design studies.*

General Terms

Security, Performance, Design, Experimentation.

Keywords

Unsolicited Bulk Email Blacklist, Scoreboard, Significance Level, Effectively Consecutive Hit, Confidence Interval.

ANCS'07, December 3–4, 2007, Orlando, Florida, USA.

1. BACKGROUND

Driven by profit making, spammers continue to launch enormous amount of unsolicited bulk email (UNBE) that floods enterprise networks, consumes network resources and poses a severe threat to the credibility of legitimate email communications. Employing a hit and run, highly mobile email delivery, and sophisticated (victim) harvesting strategies, educated spammers routinely defeat spamming filters that rely on trained historical signatures.

Most UNBE is designed for elicitation, phishing or advertisement. Without human interpretation, no existing technique can guarantee the correct classification of the intent of a random email copy. However, as the number of email copies increases, the probability of detecting repeated syntax / structural patterns in email copies also increases. This is particularly true for emails aiming at marketing or phishing, most of which were carefully crafted to deliver deceptive information. We call the email constructs that can (not) be easily altered the *variants* (in*variants*) of an UNBE source. Invariants of UNBE that also commonly occur to normal emails cannot be used for UNBE filtering, e.g., "Hello, help, etc". An UNBE invariant is called an UNBE *feature* when it can be effectively used for design of UNBE detectors. A feature can be a type of key word, e.g., an html statement, or an advertising image, or an email design style, e.g., IP number based URL.

A particular realization or instance of a feature is called a *feature instance* (FI). Even though many UNBE features are well known, spammers can still defeat the filters by morphing the FIs. We argue that a critical need for anti-spamming management is real-time sensing and classification of UNBE signatures, so that countermeasures can be deployed timely in response to the current status (hot, gray, cold, etc) of the spamming activities. A major challenge is how to screen the large volume of traffic at minimal overhead. Consider a general scenario: when a bulk of UNBE messages launched from a source, they travel through the backbone and then enter enterprise networks via their Network Access Points (NAP). Among different locations, the ingress from the ISP to the enterprise network is an excellent location to sample emails and extract the anomalous FIs in real-time, because of the large number of UNBE samples concentrated at this one point. Our goal is to minimize computing overheads of early detection of such patterns without adverse performance impact.

In this paper, we propose a *Progressive Email Classifier* (PEC) for early detection of UNBE at ingress of an enterprise network. PEC can be used as a behavior based filter alone, or if needed, interfaced to existing anti-spamming tools. Instead of relying on statistics or trained patterns for detection, PEC detects a surge of FIs of predefined features without prior training. PEC detects UNBE based on the *self similarity* property of a large number of identical/similar UNBE FIs such as the URL links,

message structures, or the last few SMTP relay hops, etc. that may signal the onset of a bulk of freshly composed UNBE messages.

Message components not defined by the detection feature are considered variants of the UNBE messages that would be ignored by PEC. On the basis of an O(1) *spinning wheel* algorithm to keep track of feature instances, the core detection engine of PEC can also be used in other protocol layer for real-time anomaly detection without prior training.

2. EXSISTING SOLUTIONS

A myriad of solutions have been developed for email filtering. The computationally feasible puzzles technique developed decades ago is being revisited as a rate limiting technique to deter spam [1]. Black lists (SpamCop, Sorbs, etc.) of spamming sources and phishing web sites are being tracked by organizations [2] [3]. Among other measures, global email providers adopt source authentication. Gmail and Yahoo both adopted the DomainKeys technology, which aims to detect spoofing senders by using encrypted tags to be compared with a shared or public database of active spamming Internet addresses [4].

Spamassassin [5] filters spamming emails based on a compound scoring system calculated from Bayesian statistics, DNS and URL black lists, and various spam databases. Implemented in Perl, [6] it is not designed for line-speed inspection. PILFER [7] uses ten features e.g., IP-based URLs, nonmatching URLs, javascript, etc to detect *phishing* frauds. The features were generated using off-line SVM. Spamato [8] is another popular spam filter, which provides an extendable interface to support plug-ins of third-party filters and makes score decisions using the filter outcomes. Currently it supports Vipul's Razor Filter [9] (blacklisted domain name look up), Earl Grey Filter [10], etc. Domainator [11] is also a URL-based filter, which uses "URL+spam" or "URL+blacklist" as the keyword to inquire Google for the likelihood of spamming. SpamGuru [12] supports user voting through Lotus Notes mail clients to handle black/white lists. A pipelined-based architecture allows multiple filers cascaded together to perform analysis of a message until it can be declared a spam or otherwise.

The sophisticated computations in content analysis based filters limit them to non real-time environments. On the other hand, PEC is designed as a line-speed detector to capture surge of new feature instances in emails. It should not come as a surprise to see integration of the two classes of detectors in different ways. For instance, PEC can use existing tools to determine the likelihood of spamming of a newly detected surge of a feature instance, at the ingress of the enterprise network. Or, PEC can publish the score results so that existing spam filters can use the published list to adjust its score weights at severs or desktops.

Behavior profiling is a viable technique to differentiate normal vs. anomalous patterns of email [13] and other general usage [14] [15] [16] based on usage frequency, social cliques, and interactions. Selection of the distance function needs to be done based on the nature of detection features [17].

Spamming emails can also be viewed as a type of *heavy hitters* of network traffic. However, despite the similar nature between PEC and these network heavy hitter detectors, they operate on entirely different premises, because PEC is a point detector but its counterparts are for Internet wide analysis. Hierarchical aggregation and computation of multi-dimensional data (IP address, port number, and protocol fields of IP flows, etc) have been extensively investigated, e.g., [18] [19], where a heavy hitter is essentially traffic flow whose volume, which is computed from the hierarchical data structure, exceeds certain threshold. Once detected, one could even locate the source via a high dimensional *key space*. A deterministic sampling technique called *lossy counting* (LC) [20] was proposed to detect heavy hitters with bounded errors. The sketch-based approach [22] used a small amount of memory to detect anomalous traffic. Based on it, a reverse hash method [23] is used to identify the keys of culprit flows without extra recording memory.

Intrusion detection systems (IDS) and antivirus software systems also rely on string matching and parsing to capture known attack patterns. Popular IDS software, such as L7-filter [23], Snort [24], and Bro [25], parse and match patterns (attack vectors) as a component of their finite state machines (DFA or NFA) to represent patterns. Hardware and software based parsing acceleration [26], [27], [28], [29], [30]is a highly active research area.

Yet, there is no solution to address UNBE at the ingress of an enterprise network. We will focus on fast algorithms tailored for detecting the onset of structural or content similarity (word count and number of URL links, image rendering sources, etc) of buffered emails in a time window, so that the content's similarity can be extracted by a higher-level of analysis and in order to configure UNBE detectors on the fly.

3. PEC SYSTEM ARCHITECTURE

PEC is designed to detect anomalous surges of *feature instances* (FI) of major UNBE flooding. An FI is a particular realization of the *UNBE feature* $\mathcal{F}$, which is any email construct that is likely to be used by spammers. From the viewpoint of system event management, each detected FI represents one *feature clock (FC)* that drives all state updates. To operate in a broadband environment, the detection process in PEC is made dynamic and lossy, i.e., some messages may pass the (overloaded) detector without being checked, to avoid adverse performance effects due to any type of resource contention.

Let $\mathcal{F} = \{ \alpha_1, \alpha_2, \alpha_3 \ldots, \}$ represent a set of binary strings which can be expressed and parsed by a finite automata, and $\alpha_i \in \mathcal{F}$ is an FI of $\mathcal{F}$. An email construct is not a viable UNBE feature if it cannot be effectively used to discriminate regular emails from UNBE, e.g., the greeting words, subject line, etc. As of writing of this paper, key words/phrases ("click here", "getting rich"), deceptive URL links, or remotely rendered images are the most prevalent UNBE features. The detection features can be added/removed ("progressive") based on resource constraint and user requirements.

Let γ denote a newly identified FI by PEC, γ is assigned one of three states: $X_\gamma \leftarrow$ G/B/W, i.e., *Gray* (unchecked), *Black* (UNBE), or *White* (not UNBE), until it is removed from the system. $X_\gamma (v) \leftarrow$ G, where v is the current FC value. γ will be retained for a certain time period before its state changes, i.e., $X_\gamma \leftarrow$W/B. $X_\gamma \leftarrow$ B (from G) if the number of its occurrences, called *score*, R_γ exceeds a *score threshold*, S, but $X_\gamma \leftarrow$ W if its *age* A_γ exceeds an *age threshold, M,* the age of γ is the time elapsed before its score is increased. S and M are two major design parameters that decide the detection sensitivity and false alarm rates of the system.

Referring to Figure 1, we propose a cascaded filter architecture consisting of *blacklist* and *scoreboard* to track FIs. Messages being filtered are parsed for FIs by the feature parser(s)

in the blacklist module. After an FI γ is extracted and hashed to H_γ which is checked against the *hotlist,* which is the hash vector of currently active FIs in some UNBE. If a hit occurs to H_γ on the hotlist, it means that γ is an UNBE instance, and the SMTP server could take countermeasure action, e.g., X-mark the (UNBE) message through the X-mark queue, or simply increasing its hit count. Otherwise, (γ, H_γ) are placed on the *graylist cache,* and H_γ is placed on the *graylist queue* of scoreboard for further tracking.

The hotlist is a single bit array with all or a part of H_γ as its table address. The graylist cache serves as a temporary lookup table between FI and $H_{FI.}$ (γ, H_γ) is moved into the UNBE database once it appears on the *report queue*, i.e., R_γ was found to exceed *S* by the scoreboard. Or, (γ, H_γ) is simply removed from the cache when Aγ exceeds *M.* The average life span of UNBE feature instances is short. As such, a background thread periodically examines the UNBE database so that the FI can be removed when it becomes cold.

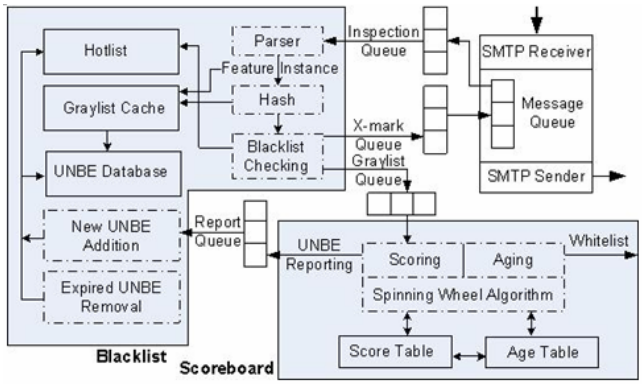

Figure 1. PEC Architecture.

H_γ passed from the blacklist to the scoreboard is tracked by *scoring* and *aging* functions based on a *competitive aging-scoring scheme* (CASS). Each FI that does not hit the blacklist but enters the scoreboard generates a *virtual clock (VC)* which triggers all the state transitions. In the rest of the discussions we will use *v* to denote the current VC value, unless explicitly defined otherwise.

If H_γ is new, it is placed into the *score table* (ST) and its score $R_\gamma \leftarrow 1$, and age $A_\gamma \leftarrow 0$ (in the *age table* (*AT*)). Otherwise, if H_γ is already in ST, R_γ is incremented and $A_\gamma \leftarrow 0$. Three types of operations are interlocked in each VC: increase of R_γ, rest of A_γ, and increase of A_β, $\forall\, \beta \neq \gamma$. An FI that does not have its score increased for a consecutive number of *VCs* is eliminated from the ST, i.e., $X_{FI} \leftarrow W$ and FI is not an UNBE feature. CASS is summarized as follows.

R1. If H_γ (or equivalently γ), is already in the scoreboard, then R_γ increments, and $A_\gamma \leftarrow 0$. Otherwise, insert it into the ST with $R_\gamma \leftarrow 1$, and also into the AT with $A_\gamma \leftarrow 0$. An old entry H_τ in ST and AT may need to be removed from scoreboard, i.e., $X_\tau \leftarrow W$, if $A_\gamma > M$ happens to occur at the same VC.
R2. A_β increments and R_β unchanged, where $\forall\ \beta \neq \gamma$,
R3. Report H_γ to blacklist if $R_\gamma > S$; remove H_γ from scoreboard after reporting.

A critical design goal of CASS is to reduce the O(N) computing cost in age increment of a naïve implementation to O(1). To solve this problem, we propose a spinning wheel algorithm to keep track of ages of entries by modeling AT into a cyclic queue, and using the queue location of an entry to represent its age. This way, the scoring (of γ) and aging operations, i.e., steps R1-R3, of all entries in AT can be collapsed into a few fixed computing cycles, independent of the sizes of AT and ST.

Different hash functions can be used for H. For simplicity and computing speed, we adopt the SDBM hash function [31] for PEC. Hash collision is inevitable, but this problem can be remedied by adding a further step of exact string matching or additional hashing. To keep our current discussion focused on the system architecture, we will address the hash management issue in the future.

3.1.1 Scoring Function

Similar to the hotlist, the score table (ST) could use the hash values of FIs as its address directly when a large number of FIs need to be tracked. Given that FIs would only stay in the scoreboard for a relatively short time before they are classified as UNBE (*B)* or normal (*W).* We propose to only use a fraction of the hash value to implement ST based on a *pointer table* and linked lists, as the data structure illustrated in Figure 2. Here, each linked node consists of *score, AT-index,* and *stamp.* The score field keeps track of R_γ the AT-index field points to the AT location of H_γ in the AT, *L*, and $AT_L \leftarrow H_\gamma$ if H_γ is a new FI so that a reverse look up can be made from any AT location of its associated linked node through the pointer table. The stamp is derived from a fraction of H_γ bits as follows.

When an entry H_γ in the graylist queue is being processed by the scoreboard, H_γ is divided into two parts, H_γ^A and H_γ^C, respectively, where H_γ^A serves as the address of a pointer table entry, which points to a linked node N, and then H_γ^C is checked against the stamp field of N. If a hit in ST occurs to H_γ, its score R_γ is incremented, and if R_γ becomes larger than *S*, H_γ is sent to the report queue (to the blacklist), and both its entries in both AT and ST are nullified. All nodes linked to N may need to be checked until a hit occurs, or a new node will be created for H_γ. By passing *L* to the aging function, we can adjust the ages of all entries in the AT. We note that any of the *k* bits in H_γ can be assigned to H_γ^A, but experiments show that using the lowest order bits tend to have lower collision ratios (or shallower linked lists).

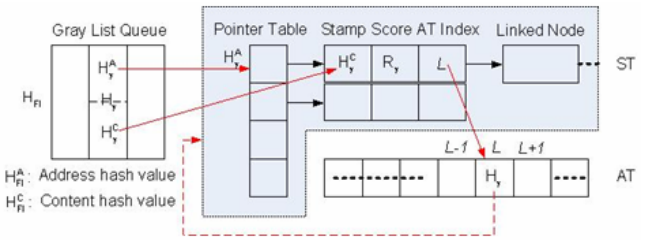

Figure 2. Score Table and Age Table.

3.1.2 Aging Function

The aging function manages the ages of FIs in the scoreboard by modeling AT into a *cyclic queue* (CQ), where the CQ position of H_γ represents its relative age among all entries in the scoreboard: Let the head and tail of CQ be denoted as CQ_D and CQ_T, respectively. The age of H_γ is smaller than that of H_β if it is positioned closer to CQ_D than H_β is. For a CQ entry placed at the

AT_i, its (next) adjacent queue entry is located at $AT_{(i+1 \bmod M)}$, where M is the table size, and AT_0 is the first array location.

Recall that scoring of one FI also implies aging of all other entries. To age all other FIs in the scoreboard at the lowest cost, first we make the age threshold *M* also the size of AT and we move CQ_D from its current location in AT to that of CQ_T at each VC. The notion of "spinning wheel" is attributed to shifting of new CQ_D to that of existing CQ_T, as illustrated in Figure 3.

The entry H_γ retrieved from the graylist queue into the scoreboard is placed at the new CQ_D position (in addition to proper ST updates) because it is made the "youngest" entry in scoreboard by CASS rules. Now that *M* is both the size of AT and the age threshold, a non-null H_β, $H_\beta \neq H_\gamma$, located at the (existing) CQ_T is the only entry whose age is to exceed *M*. As a result, these few actions required to realize R1, R2 and R3 can be computed in a few fixed steps, regardless of the value of *M*. Of course, the ST and AT entries for both H_β and H_γ need to be updated to maintain the overall consistency. It is trivial to prove that the computing complexity of the spinning algorithm is O(1).

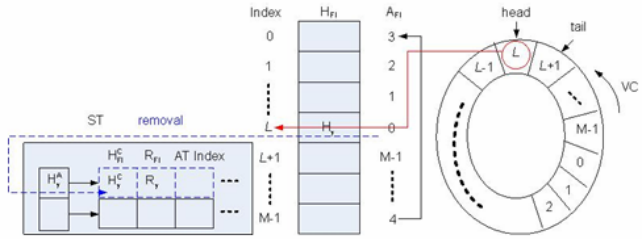

Figure 3. The spinning wheel algorithm in a CQ.

In summary, AT entries need to be adjusted in the current VC after the aging function receives a request from the scoring function to handle H_γ based on three cases: (1) $R_\gamma = 1$, (2) $1 < R_\gamma \leq S$, (3) $R_\gamma = S+1$. In case (1), γ is a brand new FI and therefore shift the AT location of the new CQ_D to that of the existing CQ_T, and then write H_γ into the new CQ_D location. In case (2), γ is already in the scoreboard, but R_γ is not high enough to be considered an UNBE. Four steps are needed for this case: step (a) use AT-index of H_γ in ST to locate the AT table address of H_γ and nullify its content. Step (b) change the AT location of the new CQ_D to that of the existing CQ_T. Step (c) copy H_γ into the new CQ_D location. Step (d) copy the new CQ_D location back to the AT-Index field of the H_γ node in the ST. In case (3), R_γ exceeds the score threshold and needs to be passed to the blacklist. We nullify the AT entry of H_γ,(erase the H_γ node in ST), and shift and move the location of the new CQ_D to that of the existing CQ_T as usual. Of course, any non-null entry H_β in the existing CQ_T needs to be purged from the scoreboard.

The interlocked operations between scoring and aging functions are illustrated through the following example consisting of three FIs: α, β and γ. The states of each FI are tracked by its score and age. Its state (W/G/B) is marked by very light, gray and solid black horizontal line segments. The event that triggers the state of an FI to change is marked by the arrowed curves. The arrival of an FI represents one VC tick. For ease of presentation, we draw the R_{FI} and A_{FI} in positive and negative Y directions, respectively. Note that VCs are not uniformly distributed on the real time line in this example.

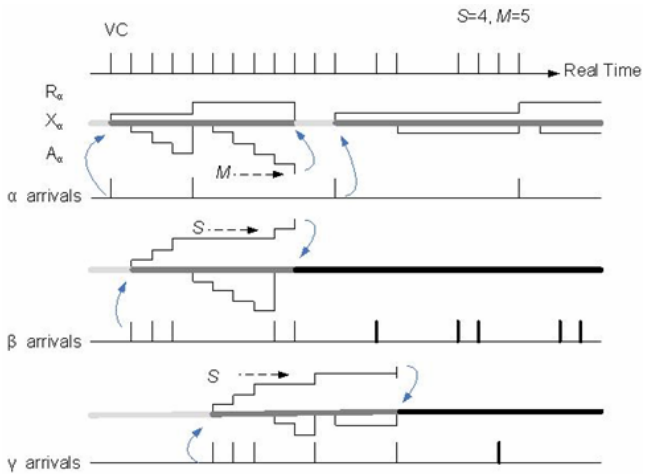

Figure 4. An illustration of aging-scoring processes in CASS.

4. DETECTION LATENCY OF FIs

Any FI will stay in the scoreboard for a finite amount of time before it is purged with its state labeled as B/W. The sojourn time for a normal FI is simply *M* counts of VCs, but it is not as obvious in determining that of an UNBE FI, i.e., the detection latency. Instead of full assessment of detection, sensitivity and selectivity in the classical sense, which would require extensive field experiments, we propose to investigate the relationship between the detection latency (time elapsed before an UNBE is placed in the blacklist) and the two key parameters *M* and *S*. That is, our goal is to answer the following two questions:

Q1: "What is the minimum value of *M* to detect an UNBE attack (of known density) with a success probability of higher than α ?"

Q2: "For a given *M*, what is the maximum value of *S* to guarantee that a probability measure P(detection latency < ς) > α ?"

Scoring and aging of FIs in the CASS constitute a competition process among FIs in the scoreboard. To answer the aforementioned questions, we only consider a single UNBE source condition here. It will become clear from simulations that this represents a worse case scenario than multi-UNBE source cases when input rates are fixed.

Answer to Q1:

Let foreground and background events, E_f and E_b denote UNBE and normal traffic, whose average rates are μ_f and μ_b, respectively. Without loss of generality, we assume that $\mu_b > \mu_f$. Recall that the arrival of each FI represents a VC tick; let λ denote the mean rate measured by instances of E_f /VC, then

$$\lambda = \mu_f / (\mu_b + \mu_f). \qquad (1)$$

Given λ, we are interested in determining the relationship between *M* and α, the probability that the instance of E_f occurs more than once in *M* VCs. Consider the more challenging case of low density UNBE detection, i.e., $\mu_f << \mu_b$, λ can be approximated as the rate of a Poisson process $\{N_f(\tau), \tau \geq 0\}$,

where $N_f(\tau)$ is the number of E_f happening prior to τ VCs. Since an E_f instance will be purged from the scoreboard if and only if $N_f(t+M)-N_f(t)\le 1$, then

$$P(N_f(t+M)-N_f(t)\le 1) \quad (2)$$

is the probability of an E_f instance expiring in M VCs, i.e., $1-\alpha$, where P is the probability density function of the Poisson distribution, expressed by

$$P(N_f(t+\Delta t)-N_f(t)=n)=\frac{e^{-\lambda t}(\lambda\Delta t)^n}{n!}. \quad (3)$$

In addition, the cumulative density function, CDF, of the Poisson distribution is

$$P(N_f(t+\Delta t)-N_f(t)\le n)=\frac{\Gamma(n+1,\lambda\Delta t)}{n!}, \quad (4)$$

where Γ is the incomplete gamma function defined as $\Gamma(x,y)=\int_y^\infty t^{x-1}e^{-t}dt$. Thereby given the survival probability α of an E_f instance and a known λ, M can be computed using

$$\Gamma(2,\lambda M)=1-\alpha. \quad (5)$$

Figure 5 shows a pivoting point located at $\lambda M=5$ of the Γ mapping curve. It means that the survival probability of an E_f instance only have marginal increase after the size of AT exceeds $5/\lambda$.

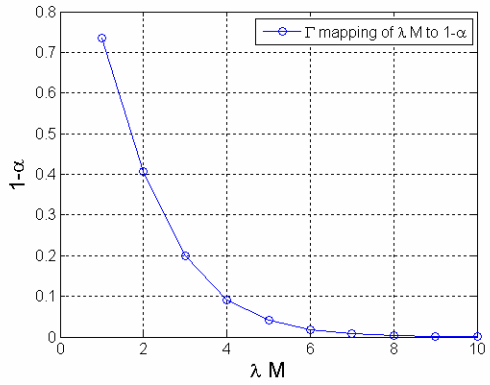

Figure 5. The incomplete Γ mapping between λM and $1-\alpha$.

Answer to Q2:

Next, we discuss setting of S in *Q2*. First, we consider the expected number of E_f instances before it is declared an UNBE for a given probability value α and S. Let H_k denote the number of hits of E_f until k *consecutive effective hits* (*CEH*) occurs, where an *effective hit* of E_f denotes the scenario that $\Delta t \le M$, when a new E_f instance occurs, where Δt represents the time interval between two adjacent E_f instances. That is, the score of E_f does increase for this hit, and it is not kicked out due to aging. If it takes H_{S-1} VCs to obtain S-1 *CEH*, then either the next time is an effective hit and we have S *CEH,* or it is not an effective hit and the scoring procedure of the E_f instance must begin anew. For an E_f instance, its effective hit probability is α, otherwise, $1-\alpha$. The expected value of H_S can be expressed as

$$\begin{aligned} E(H_S \mid H_{S-1}) &= \alpha(H_{S-1}+1)+(1-\alpha)(H_{S-1}+1+E[H_S]) \\ &= H_{S-1}+1+(1-\alpha)E[H_S] \end{aligned}. \quad (6)$$

Taking expectations on both sides of the preceding yields

$$E(H_S)=E(H_{S-1})/\alpha+1/\alpha. \quad (7)$$

Since H_1, the number of E_f until the first *effective hit,* is equal to one, we see that $E(H_1)=1$, and recursively

$$E(H_2)=2/\alpha,$$
$$E(H_3)=1/\alpha+2/\alpha^2,$$
$$E(H_4)=1/\alpha+1/\alpha^2+2/\alpha^3,$$

and in general,

$$\begin{aligned} E(H_{S+1}) &= 1/\alpha+1/\alpha^2+\ldots+1/\alpha^{(S-1)}+2/\alpha^S \\ &= (1-2\alpha^{-S}+\alpha^{-S+1})/(\alpha-1) \end{aligned}, \quad (8)$$

where $E(H_{S+1})$ implies the expected number of E_f instances before it is declared UNBE. With this result, we can determine the detection latency of E_f in terms of physical time.

With the probability α, the average detection latency ς expressed in terms of VC is

$$\varsigma=E[H_{S+1}]/\lambda. \quad (9)$$

Or, when expressed in terms of physical time (second),

$$\varsigma=E[H_{S+1}]/\mu_f. \quad (10)$$

Example:

We use the following simple example to demonstrate the setting of M and S values for a target detection rate and latency. Here, the average rate of E_f is 1.5, and that of E_b 15.0. Our design goal is to detect E_f in 520 VCs given the *Timely detection Rate* (*TR*) $\alpha=0.96$, i.e., the percentage that the actual detection latency is no more than the expected latency.

Since $E_f=1.5$ and $E_b=15.0$, we can thereby get $\lambda=0.091$ using Equation (1). Next, $M=55$ can be derived by plugging $\lambda=0.091$ and $\alpha=0.96$ into Equation (5) such that $\Gamma(2,0.091*M)=0.04$. To detect the E_f instance in 520 VCs, it means that we have $E[H_{S+1}]=47.320$ by using $\varsigma=520$ in Equation (9). To meet the detection latency goal, a larger S value is preferred over a smaller one, so that the false positive alarms would be lower. By plugging S = 23, 24, and 25 respectively into Equation (8), we can obtain $E(H_{24})=44.257$, $E(H_{25})=47.143$, and $E(H_{26})=50.149$. As a result, we choose $S=24$ among the three choices because it is the largest S value which can meet the detection latency requirement.

To evaluate the accuracy of the model, we made three major experiments to examine (i) the actual detection latency using the model-derived parameters, i.e., $S=24$ and $M=55$, (ii) the *TR*, by varying S, while M = 55, and (iii) the *TR* by varying M, but bounding S to 24. For each major experiment, we have 10 samples, numbered from 1 to 10, each sample is based on 100 runs, and the overall experimental results are plotted in Figure 6.

From Figure 6 it shows that the average detection latency for each sample is around 300 VCs, which is much lower than the model based estimation, 520 VCs. This means that in most cases the E_f instance can be detected in 520 VCs, but in some cases the

E_f instance is detected in more than 520 VCs, and it is called the *taRdy detection Rate* (*RR*), *RR*=1-*TR*.

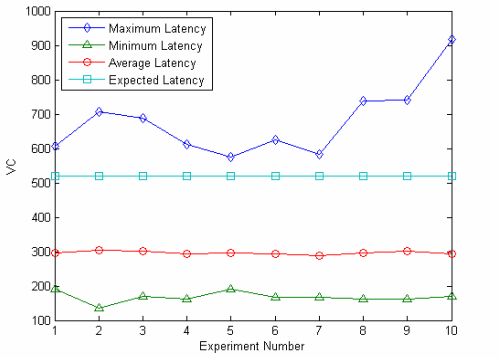

Figure 6. Detection latency when *S*=24, *M*=55.

Next, we measured the effects of *S* on *TR* with the change of *S*, and the results are plotted in Figure 7. Among the different parameter settings, pair (S, M) = (24, 55) gives the best experimental fit to the model based estimation of *TR*, which is 0.96. When S is reduced to 22, *TR* increases to [0.96, 0.99], and when S is increased to 26, the median of its *TR* becomes 0.94, which is lower than 0.96.

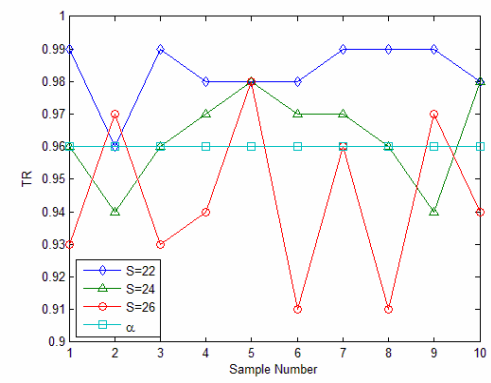

Figure 7. *TR* vs. *S* values when *M*=55.

Finally, we measured the effect of *M* on *TR*, and the results are plotted in Figure 8. Both the sojourn time of the E_f and the detection probability increase with *M*. When *M* is increased to 57, the mean value of *TR* (line marked with circle icons) is increased to 0.97. When *M* is reduced to 53, the mean value of *TR* (line marked with diamond icons) is decreased to 0.95.

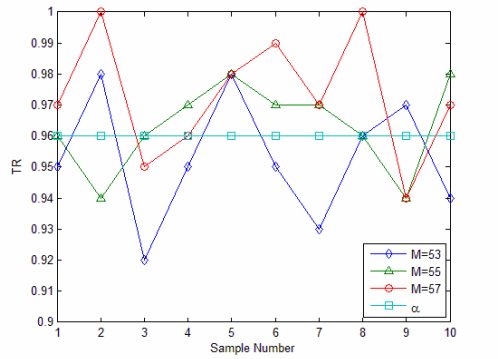

Figure 8. *TR* vs. the *M* values, when *S*=24.

5. PERFORMANCE EVALUATION

A prototype of PEC was developed to evaluate its performance in a 100 Mbps local network. The prototype has four elements, as illustrated in Figure 9. PEC and the sendmail daemon run on a Dell PowerEdge 1420 with Xeon 3.0 GHz CPU and 2GB memory. The control console and email senders run in Windows XP based PCs.

The first prototype element is email generator together with a SMTP client to generate regular and UNBE emails, following the experiment instructions sent from the control console. The control console accepts user commands (in XML) to set up experiment parameters. It also accepts user commands to set the sizes of the blacklist, ST, and AT. The control console also has a graphical interface to display the runtime status of PEC, such as the detected UNBE, normal messages, utilization ratio of the AT, and the full contents of a detected UNBE. The third element is the sendmail 8.14.1 SMTP daemon [32] which runs on a dedicated machine with standard port setup. Sendmail can handle up to 20 emails per second in our experiments.

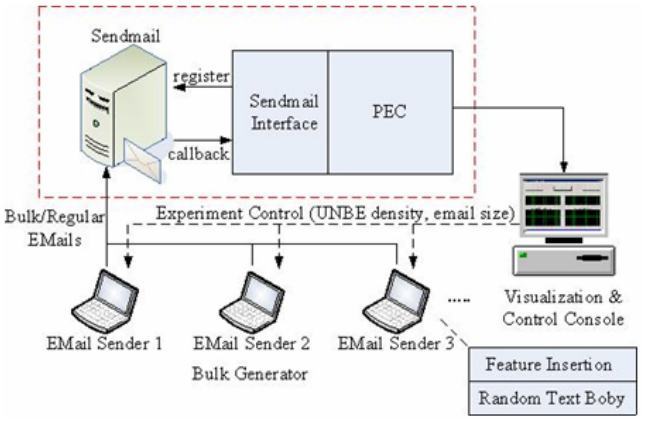

Figure 9. Experimental set up of UNBE Detection.

The fourth element is the PEC, which is interfaced to sendmail daemon through its standard registration process, so that a set of callback functions are exposed to sendmail. Sendmail can also expose basic email structures, i.e., header, email body, end of file, etc. to PEC. PEC is implemented in the blacklist and scoreboard threads, along with another background thread for UNBE retirement of the blacklist. The blacklist is interfaced to the Berkeley DB [33] for high level management purposes. The table size of the (single bit) hotlist is set at 2^{32} bits, or 512 MB. The sizes of ST, AT, graylist queue and report queue were changed based on experimental goals. In all experiments, the sizes of AT and graylist cache are set to be *M,* the age threshold.

In this experiment, only URL links are checked by PEC. An X-mark flag is inserted to the message header of a message if it has an URL on the blacklist of PEC. An UNBE batch is generated using a mixture of random text files downloaded from Internet, and a spamming URL list (SUL) extracted from the spamming corpus [34]. Subject lines and sender names were randomly generated. An UNBE specification for an experiment run includes the total number of messages, a list of URLs that would be selected from SUL to be put into the email body. The user can specify an exact number of times that each URL should be used, and the range of the message size.

The main performance criteria include: (1) detection latency (which measures the number of messages that passes the system since onset of an UNBE wave). (2) The peak throughput of the feature parser, blacklist, and scoreboard. (3) The performance impact of the queue size between the blacklist and scoreboard. (4) Finally, the impacts of the hash function on hotlist collisions, and the tradeoff between detection sensitivity, memory size and computing cycles in the scoreboard.

Figure 10 is a snapshot showing that the runtime status of PEC. The four plots (starting from left upper corner, clockwise) represent an UNBE(red, upper half plot)/regular(green, lower half plot) mixture of a sender source, PEC detection states, average score of all entries in ST, and AT utilization rate. The time series runs from left to right on the screen, i.e., an event at right side of another occurred at an earlier time instance.

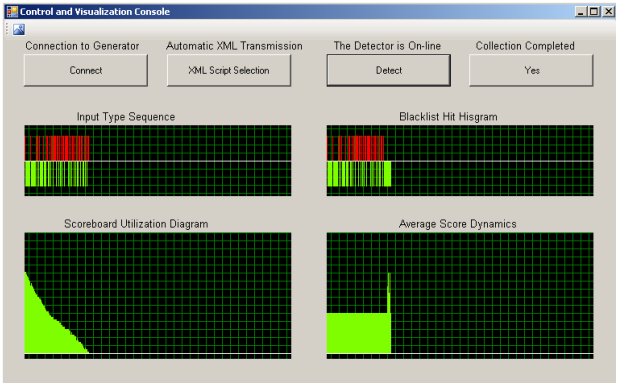

Figure 10. A snapshot of the PEC status on the control console.

Figure 11 shows the content of the AT changed when a new FI arrives at the scoreboard. In the left part of the figure, the header points to the youngest entry, [414738, 3724]. The right part shows that after a new FI, [124489, 176] arrives. It becomes the youngest one, and the oldest FI, [862, 1822] expires and is purged from AT and ST.

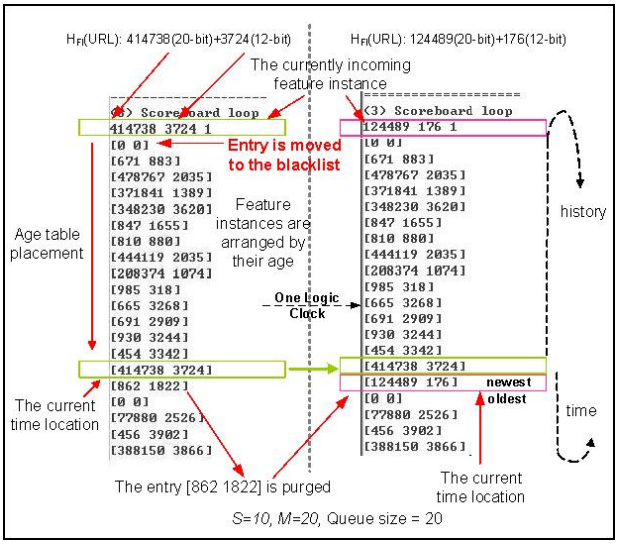

Figure 11. A snapshot of AT updates for a new FI.

Next, we measured the detection latencies under different conditions. Let *d* denote the number of UNBE messages divided by the number of total messages in one experiment observation bin. The expected detection latency is equal to *S/d*, where *detection latency* is the number of VCs elapsed from the first appearance of an FI until it is marked as an UNBE. Given a *d* specified by the user, UNBE and regular messages are randomly placed in the bin. To eliminate potential effects of small age table size, it was always set larger than that of the observation bin, so that the relationship between detection latency and other parameters can be characterized. In all experiments, the observation bin size was set as 2000, and every reported point was made by taking the average of 10 different runs.

The first experimental result is the relationship between detection latency vs. UNBE density. The two curves in Figure 12 depict respectively the expected and experimental values of detection latencies of a single UNBE source at six different densities (50, 100, 150, 200 …, 300 UNBE messages/bin). The score threshold is set at 100. As expected the detection latency decreases with the UNBE density. When only one UNBE source is considered, a linear relationship between the scoreboard threshold and detection latency was observed.

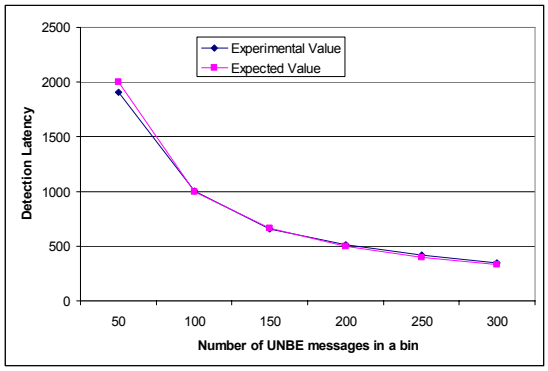

Figure 12. Detection latency of a single UNBE.

The scoreboard is designed to detect one UNBE at a time, and the age of every entry is affected by the densities and number of UNBE sources. In the next experiment, we examined the impact of multiple UNBE sources on the detection latency, where *S=50*. Given an UNBE source *A*, six tests were made where one additional UNBE source is added to the experiment at a time. That is, at test *i*, UNBE *A* is generated with *i* additional UNBE sources. The density of A is fixed at 100 instances per bin, where the bin size = 2000, and the density of every remaining UNBE sources is increased from 50 to 300 instance/bin, and we observe the change of the detection latency of UNBE A in the tests, and the results are plotted in Figure 13. The last curve marked as "other sources" is the average detection latency of other non-A UNBE sources.

The experimental results suggest that in general, the detection latency decreases with the number of concurrent UNBE sources. When A has the same density as other sources, they have the same detection latency. When A has higher or equal density as other sources, i.e., 100 vs. 50 in the first observation point, the detection latency of A is close to its expected value, i.e., 1000. The detection latency of A becomes smaller than its (single source) expected detection latency with increasing densities of other UNBE sources. This is because of the *vacuum* effect of the VCs caused by a detected UNBE. Once an UNBE is removed from the scoreboard; it will be blocked by the blacklist

indefinitely. As a result, densities of all other UNBE remaining in the scoreboard are increased, and therefore reduced detection latencies. When the density of non-A UNBEs is 200 instances/bin, their measured detection latencies are 714 and 496, respectively, close to the expected values 700 and 500.

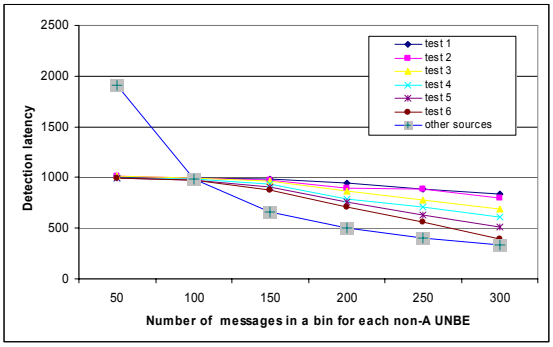

Figure 13. Detection latency for multiple UNBE sources.

In addition to the detection performance, computing cost is another critical issue for PEC. For this purpose, we first evaluated different modules (feature parser, blacklist checker, and scoreboard) in isolation, and then tested the whole system with all modules integrated together.

The feature parser is benchmarked by email bodies embedded with randomly generated URLs. The URL parser is implemented by a light-weight deterministic finite automaton (DFA). The input of the feature parser is the email bodies, whose average size is from 1.5 KB to 7.5 KB containing 2 URLs on average. From Figure 14 it shows that the processing capability decreases as the size of an email body increases. We did not consider MIME parsing because that Sendmail can separate the header and body from an email.

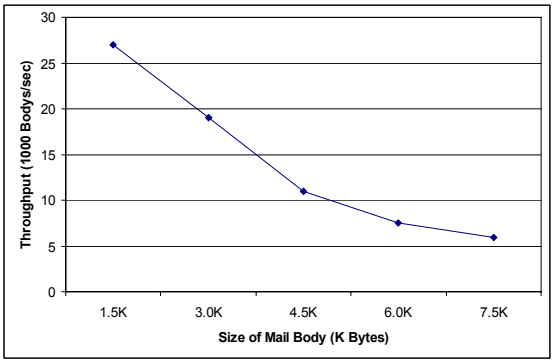

Figure 14. Throughput of feature parser.

Nowadays, URL links can contain much more than a simple domain name followed by some path. Therefore it is necessary to evaluate the sustained throughput when long and complex scripts are included in the URL. The blacklist is benchmarked by randomly generated URLs of a wide range of lengths, and the results are plotted in Figure 15. The main operations include the SDBM hash value generation, hotlist lookup, and the graylist cache update. The UNBE database access is not considered in the benchmark. The results suggest that high speed URL parsing, e.g., 300k to nearly 1M links can be parsed by commodity hardware.

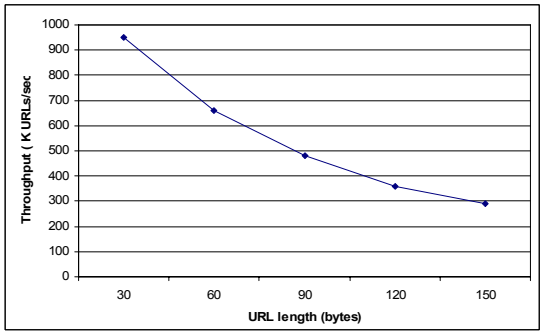

Figure 15. Throughput of the blacklist checking.

The last, but not least throughput measure is that of the scoreboard. The scoreboard is benchmarked by randomly generated 32-bit unsigned integers, and experimental results show that it can process 1.2 M requests per second.

The two pthreads for blacklist and scoreboard communicate through the graylist queue and the report queue. The two threads use the mutex primitive to make mutually exclusive access to the graylist queue. As a stress test, one million hash values are directly generated by a random number generator and then fed to the blacklist and scoreboard to simulate the extreme condition that the STMP, URL parsing and hashing were done in negligible time. Figure 16 depicts the time needed for handling 1 million hash values with change of the queue size. The results suggest that when the queue size is smaller than a threshold, the processing time would grow rapidly because of rapidly increasing overhead for the mutex operations because of inadequate queue sizes. Beyond the threshold, the queue length has minimal impact on the throughput.

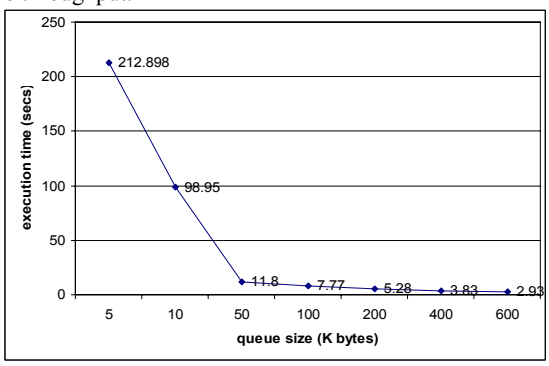

Figure 16. Scoreboard throughput vs. queue sizes.

5.1 Collision Ratio in Hash Functions

Increasing the hash length will reduce collisions but it comes at the cost of increased memory sizes. In this part of experiments, we investigated the relationship between the collisions and table sizes for the hotlist, and the relationship between detection sensitivity, sizes of ST and AT tables and the depths of link lists in the scoreboard.

The hotlist is a single-bit table, so that when the hash length is 32 bits long, 512 MB of memory space suffices. Experimental results show that the collision ratio is very close to zero after 1M randomly generated URLs is hashed. The same cannot be said when we reduce the hash length, and the experimental results are given in Figure 17

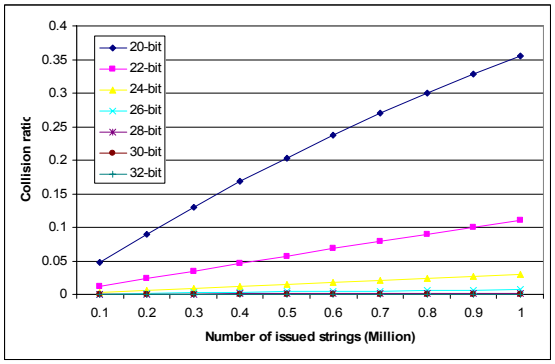

Figure 17. The collision ratio in blacklist.

There exists a more complex relationship between detection sensitivity, the sizes of AT and ST, score threshold S, and the depths of the linked lists in the ST. For simplicity, we consider S a fixed constant in this experiment. In general, we observed that increasing of M would increase UNBE detection sensitivity because of longer staying of an FI in the scoreboard. As a result, the chance of hash collisions with other FIs increases. This will lead to longer linked lists in ST. Similarly, increasing of H_{FI}^{A} (for the pointer table addressing) should reduce the depths of linked lists in ST.

To gain a sense of the relationships between these parameters, we conducted a series of experiments, where S is fixed at 50. The pointer table size is changed in different experimental runs to observe the relationship between M and the depths of link lists. That is, given a pointer table size, the maximum depths of linked lists are measured against different M values, and the results are reported in Table 1. In this table, the most left column is the bit length of H_{FI}^{A}, which also represents the size of the pointer table. For instance, when H_{FI}^{A} = 20 bits, it means the pointer table has 1M entries. The top most rows represents the maximum depth of linked lists in ST, and an table entry indexed by the (table size, depth) represents the smallest M value (measured in thousands, or k) necessary to observe the said depth in ten experimental runs. For instance, if H_{FI}^{A} =22 bits, and M=40k, one can expect the depth of linked lists in ST to be one in most time. But if M = 50k, then likely the depth of some linked lists would be 2 or less.

Table 1. The relationship between H_{FI}^{A}, M, and linked lists.

	(depth) 2	3	4	5	6	7
(bit length) 20	20k	70k	167k	297k	582k	657k
22	45k	165k	337k			
24	145k	515k				
26	375k					
28	520k					
30	650k					

It is observed that spammers often used different URL expressions that point to the same web page, e.g., replacing the last string followed by the last "/" by another one. Tracking domain names is also a good way to differentiate UNBE sources and it can significantly increase hit ratio in blacklist. However, when two users operate within the same domain name, e.g., http://www.geocities.com/user1, http://www.geocities.com/user2, using domain name alone for filtering could lead to significant false positive rates. A white list of domain names is needed to solve this problem. In this test samples, the number of tested Spammer URLs is 66501, and those are all different. Table 2 shows the result of testing how the three ways increase the hit ratio.

Table 2. Hit ratio increased by handling different prefixes.

Method	Hit Ratio Increased	Examples
Remove the last character	11.9%	(1) and (2)
Remove the string after the last "/"	27.3%	(3) and (4)
Use domain name	61.2%	(5) and (6)
(1)mailshere.biz/profile/41991 (2)mailshere.biz/profile/41996 (3)appliancekiosk.geappliances.com/NASApp/ClickThru/ClickThru?q=df-1kkhQLEGhj6FEsKyiWNXKsRR (4)appliancekiosk.geappliances.com/NASApp/ClickThru/ClickThru?q=9b-7-XFQSIs_-MZdLePIxf09dRR (5)www.slammer7piggy.com/?0gQF6VyMLeif0QXXZ (6)www.slammer7piggy.com/?dXKwf0tqXJMLeiZ		

6. CONCLUSION

Using matching of (untrained) URLs as the UNBE feature, this paper is mainly focused on the architectural designs of PEC. Our experiments show that PEC is capable of high speed capturing of repeated patterns (FIs) that are usually associated with UNBE. This technique does not require prior training and can be implemented in both software and hardware. The aging and scoring engines have constant time complexity in keeping track of the gray list; making them highly suitable to be used at NAP. Experimental results show that the vacuum effect in the CASS process makes PEC robust to concurrent UNBE sources.

Meeting performance requirements is only one of the design goals. The other critical issue is selection and refinement of UNBE features. URL is not the only UNBE feature, albeit it will remain one of the most prevalent ones. The preliminary study results from the spamming samples, as shown in Table 2, suggest that small changes in the URL links can have significant performance effects. Classical techniques, such as IP-based URLs, number of domains, and number of dots used in PIFLER can be used to create a weighted scoring and aging system while maintaining the high efficiency of the scoreboard architecture. Finally, another interesting and important issue is creation of a compound scoring/aging system using multiple features simultaneously.

7. ACKNOWLEDGEMENTS

This research is supported in part by the Army Research Office under the contract number C07-00485, and by National Science Foundation under the contract numbers CNS- 0530210 and DUE- 0516825.

8. REFERENCES

[1] C. Dwork, Andrew Goldberg, and Moni Naor, "On Memory-Bound Functions for Fighting Spam," Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO 2003), pp. 426-444, August 2003

[2] "The Spamhaus project," retrieved at Jan. 27, 2007, from http://www.spamhaus.org/index.lasso

[3] "Know Your Enemy: Tracking Botnets, 2005," retrieved at Jan. 28, 2007, from http://www.honeynet.org/papers/bots/.

[4] "Domainkeys: Proving and Protecting Email Sender identity," retrieved at Jan. 8, 2007, from http://antispam.yahoo.com/domainkeys

[5] "The apache SpamAssassin Project," retrieved at Feb. 08, 2007, from http://spamassassin.apache.org/

[6] "SpamAssassin Benchmark," retrieved at Feb. 14, 2007, from http://www.isode.com/whitepapers/spamassassin-benchmark.html

[7] Ian Fette, Norman Sadeh, Anthony Tomasic, "Learning to Detect Phishing Emails," June 2006, CMU-ISRI-06112

[8] "Spamato Spam Filter System," retrieved at Mar. 22, 2007, http://www.spamato.net/

[9] "Vipul's Razor," retrieved at Mar. 22, 2007, http://razor.sourceforge.net/

[10] "Earl Grey Filter," retrieved at Mar. 22, 2007, http://www.spamato.net/

[11] "Domainator," retrieved at Mar. 22, 2007, http://www.spamato.net/

[12] R. Segal, J. Crawford, J. Kephart, and B. Leiba, "SpamGuru: An Enterprise Anti-Spam Filtering System," In Proceedings of the First Conference on E-mail and Anti-Spam, 2004

[13] S. J. Stolfo , et al, "Behavior-based modeling and its application to Email analysis," ACM Transactions on Internet Technology, Volume 6 , Issue 2 (May 2006)

[14] Ke Wang, Janak J. Parekh, S. J. Stolfo "Anagram: A Content Anomaly Detector Resistant To Mimicry Attack," In Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection(RAID 2006)

[15] Shobha Venkataraman et al, "New Streaming Algorithms for Superspreader Detection," In Network and Distributed Systems Security Symposium, Feb 2005.

[16] Fulu Li, Mo-Han Hsieh, "An Empirical Study of Clustering Behavior of Spammers and Group-based Anti-Spam Strategies," Proceedings of Third Conference on Email and Anti-Spam, 2006

[17] D. Gao, M. Reiter and Dawn Song, "Behavioral Distance for Intrusion Detection," Symposium on Recent Advance in Intrusion Detection (RAID), Sep 2005.

[18] Yin Zhang, Sumeet Singh, Subhabrata Sen, Nick Duffield, Carsten Lund, "Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluations, and Applications," In Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, 2004.

[19] Graham Cormode et al., "Finding Hierarchical Heavy Hitters in Data Streams," In Proc. Of the 29th VLDB Conference, Berlin, Germany,2003. SIGCOMM conference on Internet measurement, 2003

[20] Gurmeet Singh Manku, Rajeev Motwami, "Approximate Frequency Counts over Data Stream," In Proceedings of the 28th VLDB conference, Hong Kong, China, 2002.

[21] Krishnamurthy, B. et al, "Sketch-based change detection: Methods, evaluation, and applications," In Proc. of ACM SIGCOMM IMC, 2003.

[22] Robert Schweller, Ashish Gupta, Elliot Parsons, Yan Chen, "Reversible Sketches for Efficient and Accurate Change Detection over Network Data Streams," In Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, 2004.

[23] "Application Layer Packet Classifier for Linux," retrieved at Jan. 18, 2007 from http://l7-filter.sourceforge.net

[24] "Snort," retrieved at Jan. 05, 2007 from http://www.snort.org

[25] "Bro IDS," retrieved at Jan. 09, 2007 from http://bro-ids.org./Overview.html

[26] Fang Yu, Ahifeng Chen, Yanlei Diao, "Fast and memory-efficient regular expression matching for deep packet inspection," ANCS'06

[27] S. Kumar, S, S. Dharmapurikar, F. Yu, P, Crowley, J. Turner. "Algorithms to Accelerate Multiple Regular Expressions for Deep Packet Inspection," SIGCOMM'06

[28] J. Moscola, J. Lockwood, R. P. Loui, and Michael Pachos, "Implementation of a Content-Scanning Module for an Internet Firewall," Proc. FCCM, 2003

[29] B. C. Brodie, R. K. Cytron, and D. E. Taylor, "A Scalable Architecture for High-Throughput Regular Expression Pattern Matching," ISCA'06

[30] S. Dharmapurikar, M. Attig, and J Lockwood, "Deep packet inspection using parallel bloom filters," IEEE Micro, 2004

[31] "SDBM," retrieved at Mar. 03, 2007 from http://search.cpan.org/src/NWCLARK/perl-5.8.8/ext/SDBM_File/sdbm/README

[32] "Sendmail.org," retrieved at Mar. 17, 2007 from http://www.sendmail.org/

[33] "Berkeley DB," retrieved at Feb. 11, 2007 from http://www.oracle.com/technology/products/berkeley-db/db/index.html

[34] "2005 TREC Public Spam Corpus," retrieved at Jan. 16, 2007 from http://plg.uwaterloo.ca/~gvcormac/treccorpus/about.html