Spam Detection in Voice-over-IP Calls through Semi-Supervised Clustering

Yu-Sung Wu, Saurabh Bagchi[1] Navjot Singh[2] Ratsameetip Wita[3]

[1] *School of Electrical & Computer Eng., Purdue University, West Lafayette, IN 47907*

[2] *Avaya Labs, 233 Mt. Airy Rd., Basking Ridge, NJ 07920*

[3] *Chulalongkorn University, Thailand*

{yswu,sbagchi}@purdue.edu, singh@avaya.com, Ratsameetip.W@Student.chula.ac.th

Abstract

In this paper, we present an approach for detection of spam calls over IP telephony called SPIT in VoIP systems. SPIT detection is different from spam detection in email in that the process has to be soft real-time, fewer features are available for examination due to the difficulty of mining voice traffic at runtime, and similarity in signaling traffic between legitimate and malicious callers. Our approach differs from existing work in its adaptability to new environments without the need for laborious and error-prone manual parameter configuration. We use clustering based on the call parameters, using optional user feedback for some calls, which they mark as SPIT or non-SPIT. We improve on a popular algorithm for semi-supervised learning, called MPCK-Means, to make it scalable to a large number of calls and operate at runtime. Our evaluation on captured call traces shows a fifteen fold reduction in computation time, with improvement in detection accuracy.

Keywords

Voice-over-IP systems, spam detection, spit detection, semi-supervised learning, clustering

1. Introduction

As the popularity of VoIP systems increases, they are being subjected to different kinds of security threats [1]. A large class of the threats such as call rerouting, toll fraud, and conversation hijacking incur deviations in the protocol state machines and can be detected through monitoring the protocol state transitions [2],[3]. Additionally, cryptographically secure versions of the common VoIP protocols, such as Secure SIP and Secure RTP, address many of the attacks presented in the literature. However, spam calls in VoIP [4], commonly called SPIT, are becoming an increasing nuisance. The ease with which automated SPIT calls can be launched can derail the adoption of VoIP as a critical infrastructure element. Existing monitoring and cryptographic solutions are not immediately applicable to SPIT detection. In this paper, we address the problem of detection of SPIT calls.

Detection of spam emails is a mature field and there are some similarities to our problem. In both domains, users can provide feedback about individual email or call, for the latter, through a built-in button in some commercially available VoIP phones. However, there exist significant differences—VoIP traffic is real-time and the detection should ideally be real-time as well; some features are expensive to extract in real-time, especially those in voice traffic; the signaling patterns are likely similar in legitimate and malicious calls rendering content-based filtering on signaling traffic ineffective; and features from multiple protocols used in VoIP may be relevant.

In this paper, we present the design of a system that uses *semi-supervised machine learning* for detection of SPIT calls. It builds on the notion of clustering whereby calls with similar features are placed in a cluster for SPIT or legitimate calls. Call features include those extracted directly from signaling traffic, those extracted from media traffic, such as proportion of silence in the call, and those derived from calls. However, previous approaches that use thresholds [5] on the call features are difficult to use in practice since the nature of SPIT calls varies widely. Therefore, we learn the features to use and their relative importance in clustering through runtime observations, which include user feedback.

The popular semi-supervised clustering algorithm called MPCK-Means [6] scales as $O(N^3)$ where N is the number of calls. This would generally be too expensive for real-time operation. We modify this to create our algorithm called eMPCK-Means, using VoIP specific features to reduce it to $O(N)$. Such specialization includes the early use of user feedback and prior knowledge of the number of clusters. Additionally, we create an incremental protocol called pMPCK-Means, that can perform the detection as soon as the call is established.

We evaluate the protocols using four call traces with different characteristics of SPIT and non-SPIT calls, over different proportions of user feedback and accuracy of the user feedback. With a batch of 400 calls, eMPCK-Means is 15 times faster than MPCK-Means, while achieving better detection coverage in terms of true and false positives. Since pMPCK-Means can examine a limited set of call features, it works well only with a large fraction of calls with accurate user feedback.

2. Related work

Rosenberg [4] details the problem of VoIP SPIT and gives various high-level conceptual solutions. The solutions can be placed in three categories [7]: (1) Non-intrusive methods based on the exchange and analysis of signaling messages; (2) Interaction methods that create inconveniences for the caller by requesting them to pass a checking procedure before the call is established; (3) Callee interaction methods that exchange information with the callee on each call. An example work in category 1 is [8] where the authors look at the SIP signaling traffic pattern to detect SPIT. However, they do not provide quantitative data on the detection accuracy. Our experimental results indicate solely relying on SIP message patterns will give low detection coverage. The work by Quittek [7] generates a greeting sound or faked ring tone to the caller right after the call is established and monitors the response voice patterns from the caller to differentiate between human caller and a SPIT generator. This falls in category 2. In comparison, our work encompasses categories 1 and 3.

Kolan [9] presents an approach which maintains the trust information for each caller. The information can be automatically built up through user feedback, or through a propagation of reputation via social networks. The approach can be used in our system where we can embed the caller's trust as one of the call features. However, the reputation database may grow large and a reputation system can be gamed by false praise or false blame.

Clustering is a way to learn a classification from the data [10], especially with unlabeled data. Clustering techniques have been used for detecting e-mail spam in [11],[12]. On the other hand, classification techniques such as SVM [13] are popular for data classification. However, they typically require labeled data and do not take unlabeled data into consideration. Recent developments in semi-supervised classification techniques [14], such as semi-supervised SVM [15], incorporate both labeled and unlabeled data.

3. Design

3.1 Structure of VoIP calls

There are typically three phases involved in a VoIP phone call [16]. The first phase is call establishment through a three-way handshake, which involves (i) the caller sending a SIP INVITE message to the proxy server and the server forwarding the INVITE message to the callee, (ii) the callee replying with a SIP OK message, and (iii) the caller sending SIP ACK message to complete the call establishment phase. The second phase is the conversation, which contains the media stream (voice) transmitted between the caller and the callee typically using RTP/RTCP [17]. The last phase is the call tear down phase, which can be initiated by either the caller or the callee sending a SIP BYE message followed by SIP OK and SIP ACK messages.

3.2 Characteristics of VoIP SPIT calls

A blacklist-based approach can be used at the call establishment phase based on source IP or From URI to drop calls from known SPIT sources. In the media stream phase, a typical pattern one can imagine for SPIT calls is that the caller speaks more than the callee. Another pattern is that the length of the media stream phase, i.e., the call duration, is shorter in the case of calls answered by a live person since SPIT calls are generally undesirable. Also, one can assume that it is more likely that for a SPIT call, a call termination will be initiated by the callee, i.e., the callee sends the SIP BYE message.

Since SPIT calls are usually large volume calls made by some spitter within a period of time, we found that it is also useful to look for patterns in a batch of calls. Certain features are available when looking at the collective set of calls, such as the inter-arrival time between calls. Also statistical learning can only occur with a batch of calls.

3.3 Detection scheme

A VoIP environment typically consists of multiple domains with each domain composed of a few proxy servers and phones belonging to end users. Figure 1 shows an example VoIP environment consisting of two domains. In a VoIP environment, a proxy server's main function is to route the signaling messages. For the specific example we show, here Proxy #1 is used to route the signaling

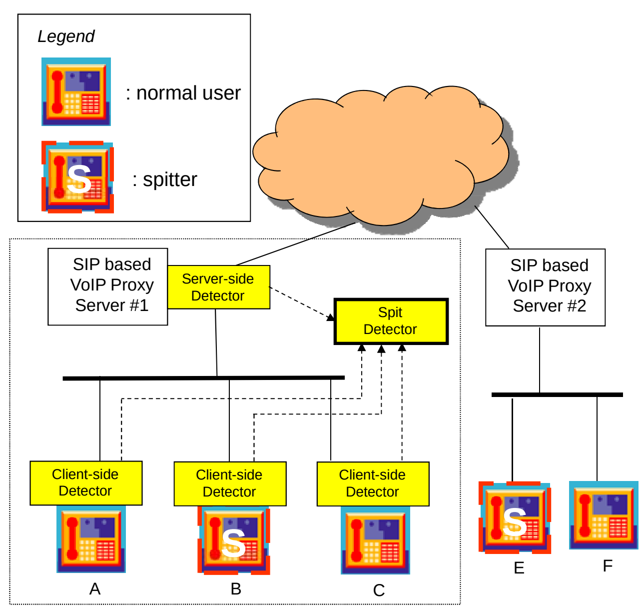

Figure 1. Detecting Spit Calls in a VoIP Environment

messages among phones {A,B,C}. And similarly, Proxy #2 is used to route the signaling messages among phones {E,F}. Cross domain phone calls {A,B,C} $\rightleftarrows$ {E,F} are collaboratively handled by Proxy #1 and Proxy #2. Once a phone call is established, subsequent messages (signaling and voice) can travel directly between phones without involving the proxies. However, an ISP can mandate all traffic pass through the proxies, which is often the case for billing and security purposes.

Our approach in detecting SPIT calls involves placing local detectors at the SIP proxies and the phones in the managed domain. The domains that have our detection mechanism are called *managed domains* and others are called *unmanaged domains*. Essentially, the detectors require observability of the signaling and the media streams within the managed domain. A spitter can exist as any phone in a VoIP environment, whether within a managed (phone B) or an unmanaged domain (phone E).

The embedded detectors collect the information of the phone calls and send them to the SPITDetector, where the logic for differentiating SPIT calls from non-SPIT calls executes. The decoding of the traffic and calculation of the call features are handled by the respective server-side/client-side detectors and only a digest of the necessary information is forwarded up to the detector, thus minimizing network traffic.

SPITDetector supports two modes of detection:

Mode A: Look at each phone call with early detection: In this mode, the SPITDetector has to determine whether a call is a SPIT or not before the media stream of the call is established. This means that the detection has to be completed before the callee picks up the phone. This mode is useful from an end-user's point of view since SPIT calls can be potentially blocked without further annoyance.

Mode B: Look at the whole batch of phone calls: With Mode B, we assume received calls are kept in a collection which are then presented in a batch to our semi-supervised clustering algorithm. This mode provides higher detection accuracy than Mode A due to the availability of complete call feature information. Mode B is attractive to a service provider, rather than to an end user.

4. SPIT Detection using Semi-Supervised Clustering

4.1 Background

In our problem context, each VoIP call is regarded as one data point. We are interested in clustering call data points into two clusters, one containing the SPIT calls, and the other containing the non-SPIT calls. In general, there may be multiple sub-clusters within each cluster corresponding to radically different kinds of SPIT or non-SPIT calls. We explore this approach of multiple sub-clusters further in Sec. 4.7.

Semi-supervised clustering [18], [19], [6] is a recent development in the data clustering research community that aims to address the issue of selecting the proper criteria for clustering. Semi-supervised clustering allows the use of optional labeled data for a subset of the runtime observations to progressively modify the clustering criteria. This means that one does not need to determine *a priori* which features of the data points should be used for clustering. The clustering criteria will be trained into generating clusters that obey the user-labeled data as faithfully as possible [6]. The implicit assumption is that user feedback is perfectly accurate. In our work here, we evaluate the impact of noise in the user feedback.

4.2 VoIP call features for clustering

We construct a data point from each VoIP call based on 17 features: 1-2. From/To URI, 3. Start time, 4.Duration, 5. # of SIP INVITE messages, 6. # of ACK messages, 7-8. # of BYE messages from caller/callee, 9. Time since the last call from the originator of the current call, 10-15. # of 1xx, 2xx, 3xx, 4xx, 5xx, and 6xx SIP Response messages, 16. Call frequency of the originator of the current call, 17. Ratio of non-silence duration of the callee to the caller media streams.

For Mode A early detection, only features 1, 2, 3, and 9 are available. Feature 17 is derived from the RTP media stream by client-side detectors if the media streams are configured to flow directly between clients [20] or it can be provided by the server-side detector if the media streams are configured to flow through the SIP Proxy. We select the universe of features using our domain knowledge, to cover different facets of a VoIP call and to limit the number of features so that online clustering is feasible.

4.3 Labeled data via user feedback

Phone calls received in the managed domain can have optional user feedback information indicating whether a call is a SPIT call or a non-SPIT call. The corresponding data point will be labeled with a SPIT or a non-SPIT tag and fed into the semi-supervised clustering process. Such a data point will be used for adjusting the clustering criteria.

4.4 Extended K-Means for semi-supervised clustering: MPCK-Means

For this work, we select the semi-supervised clustering algorithm called MPCK-Means [6].

$$\begin{aligned}\tau_{\text{mpckm}} = &\sum_{x_i \in \chi}\left(\left\| x_i - \mu_{l_i}\right\|^2_{A_{l_i}} - \log\left(\det\left(A_{l_i}\right)\right)\right) \\ &+ \sum_{(x_i,x_j) x_i \in M} w_{ij} f_M\left(x_i,x_j\right) 1\left[l_i \neq l_j\right] \\ &+ \sum_{(x_i,x_j) x_i \in C} \overline{w_{ij}} f_C\left(x_i,x_j\right) 1\left[l_i = l_j\right]\end{aligned} \tag{1}$$

$$\|x_i - \mu_i\|^2_{A_{l_i}} = (x_i - \mu_i)^T A_{l_i} (x_i - \mu_i) \quad (2)$$

$$f_M(x_i, x_j) = \frac{1}{2}\|x_i - x_j\|^2_{A_{l_i}} + \frac{1}{2}\|x_i - x_j\|^2_{A_{l_j}} \quad (3)$$

$$f_C(x_i, x_j) = \|x'_{l_i} - x''_{l_i}\|^2_{A_{l_i}} - \|x_i - x_j\|^2_{A_{l_j}} \quad (4)$$

$$\begin{aligned} A_h = |X_h| \Bigg(& \sum_{x_i \in X_h} (x_i - \mu_h)(x_i - \mu_h)^T \\ & + \sum_{(x_i, x_j) \in M_h} \frac{1}{2} w_{ij} (x_i - x_j)(x_i - x_j)^T 1[l_i \neq l_j] \\ & + \sum_{(x_i, x_j) \in C_h} \Big(\overline{w_{ij}} (x'_h - x''_h)(x'_h - x''_h)^T \\ & - (x_i - x_j)(x_i - x_j)^T 1[l_i = l_j] \Big) \Bigg)^{-1} \end{aligned} \quad (5)$$

Eq. (1) is the objective function that MPCK-Means minimizes. l_i is the cluster that point x_i is associated with. The main idea is the same as K-Means where intra-cluster distance is being minimized. However the Euclidean distance metric in MPCK-Means is weighted by a cluster-specific matrix A_{li} (one can also use the same A matrix across all clusters)[6]. A_{li} is modified based on user feedback and points in cluster l_i following Eq.(5).

The user labeled data in MPCK-Means is supplied in the form of clustering constraints M (must link sets) and C (cannot link set). Here the M set specifies pairs of data points that should be put in the same cluster while the C set specifies those pairs of data points that should not be put in the same cluster. In Eq. (1), the last two terms are used to add penalty to the objective function from the violation of these constraints. The function f_M returns a value proportional to the distance between the two points that are in different clusters. The function f_C returns a value that is inversely proportional to the distance between two points that are in the same cluster. The points x_{li}' and x_{li}'' represent the two farthest data points in X_{li} with respect to their distance computed using A_{li}. The pseudo code for MPCK-Means is listed as Algorithm 1 below.

Input: Set of data points $X = \{x_i\}_{i=1}^N$, Set of must-link constraints $M = \{(x_i, x_j)\}$, Set of cannot-link constraints $C = \{(x_i, x_j)\}$, # of clusters K, Sets of constraints costs W and $\overline{W}$, $t \leftarrow 0$.

Output: Disjoint K-partitioning $\{X_h\}_{h=1}^K$ of X such that objective function τ_{mpckm} is locally minimized.

Method:

1. Initialize clusters:

1.1. Create the λ neighborhoods $\{N_P\}_{P=1}^{\lambda}$ from M and C.

if $\lambda \geq K$

Initialize $\{\mu_h^{(0)}\}_{h=1}^K$ using weightiest farthest-first traversal starting from the largest N_P.

Else

Initialize $\{\mu_h^{(0)}\}_{h=1}^{\lambda}$ with centroids of $\{N_P\}_{P=1}^{\lambda}$

Initialize remaining clusters at random

2. Repeat until convergence

2.1. For each data point $x_i \in X$

$$h^* = \arg\min_h \Big(\|x_i - \mu_h^{(t)}\|^2_{A_h} - \log(\det(A_h)) + \sum_{(x_i,x_j) \in M} w_{ij} f_M(x_i, x_j) 1[h \neq l_j] + \sum_{(x_i,x_j) \in C} \overline{w_{ij}} f_C(x_i, x_j) 1[h = l_j] \Big)$$

Assign x_i to $X_{h^*}^{t+1}$

2.2. For each cluster X_h, { $\mu_h^{(t+1)} \leftarrow \left(\sum_{x \in X_h^{t+1}} x\right) / |X_h^{t+1}|$ }

2.3. Update_metrics A_h for all clusters $\{X_h\}_{h=1}^K$ (Eq. (5))

2.4. $t \leftarrow t + 1$

Algorithm 1. MPCK-Means (Adapted from [6])

4.4.1 Mapping user feedback to pair-wise constraints in MPCK-Means

The system keeps two sets: F_S (data points of SPIT calls from feedback) and F_N (data points of non-SPIT calls from feedback). For a data point x_i, which has user feedback, the user indicates $x_i \in F_S$ or $x_i \in F_N$. With respect to the MPCK-Means algorithm, must-link constraints M are derived online from pairs of points $(x_i, x_j) \in F_S$ or $(x_i, x_j) \in F_N$. Similarly, cannot-link constraints C are created online from (x_i, x_j), where $x_i \in F_S$ and $x_j \in F_N$.

For ease of exposition, we initially discuss the case with 2 clusters—one each for SPIT and non-SPIT calls. We discuss the extension to multiple clusters in Sec. 4.7.

4.4.2 Building detection predicate

Given a cluster X_h from the clustering algorithm, we use the number of data points with different user feedback in the cluster to determine the association of the cluster. If $|X_h \cap F_S| > |X_h \cap F_N|$, the calls in X_h will be considered SPIT calls; else, they will be considered non-SPIT calls.

4.5 Efficient MPCK-Means

In the cluster assignment step of MPCK-Means (Step 2.1) the time complexity on iterating through the must-link/cannot-link peers of point x_i is a O(N) operation. X is the whole set of data points supplied to the clustering algorithm. N=|X| is the number of data points. The determination of the maximally separated points x'_h and x''_h used in f_c(.) (Step 2.1 of Algorithm 1) and update_metrics (Step 2.3) has time complexity O(N^2). This implies MPCK-Means is O(N^3) since the operation has to be done for each data point (actually O(cN^3) where c is a small fixed number of iterations till convergence). Thus, MPCK-Means does not scale well with large data sets. For our application, where N can be hundreds for a small-sized domain or thousands for a mid-sized domain, it turns out to be prohibitive time-wise to apply the original MPCK-Means directly.

Therefore, we adapt MPCK-Means into the eMPCK-Means (efficient MPCK-Means) algorithm (Algorithm 2). In it, the maximally separated points are estimated through an O(1) approximation algorithm. We use an O(N)

implementation for the neighborhood creation process in the cluster initialization step of MPCK-Means. Additionally, the general practical experience with a K-Means based algorithm is that it converges within a small number of iterations for the main loop (Step 2 in MPCK-Means). Combined these make eMPCK-Means O(N) and the constant is small for a range of VoIP call traces.

4.5.1 eMPCK-Means : Initialize clusters

The eMPCK-Means algorithm creates the initial neighborhoods directly from the user feedback F_S and F_N sets. Specifically, it creates w neighborhoods $\{F_S, F_N, x_{n3}, x_{n4}, \ldots, x_{nw}\}$, where $\{x_{n3}, x_{n4}, \ldots, x_{nw}\} = X\text{-}F_S\text{-}F_N$ is the set of data points not covered by the user feedback. The complexity of this step is O(N). We use the same weighted-farthest-first traversal as in MPCK-Means, which is O(N) when the number of clusters is a constant. Overall, the initialize clusters in eMPCK-Means has O(N) complexity.

4.5.2 eMPCK-Means : efficient estimation of maximally separated points $\left(x_h^{'}, x_h^{''}\right)$

In MPCK-Means, to find the exact maximally separated points $\left(x_h^{'}, x_h^{''}\right)$ used in Eq. (4) and A_h matrix updating[6], it requires evaluating the distance $\left\|x_i - x_j\right\|_{A_h}^2$ for every pair of points $(x_i, x_j) \in X$, which is an O(N^2) operation. Since the matrix A_h is updated in each iteration of the loop of step 2 in Algorithm 1, this evaluation has to be repeated as well.

In eMPCK-Means, we estimate the maximally separated points by first putting data points from X into an array R[1..N] in a random ordering. We then iterate through consecutive elements R[i] and R[$i+1$] in the array. We set $\left(x_h^{'}, x_h^{''}\right)$ to (R[i'], R[$i'+1$]) that gives the maximal value of $\left\|R[i'] - R[i'+1]\right\|_{A_h}^2$. This operation (Step 2 in Algorithm 2) is performed once right after the cluster initialization step and is done K times, once for each cluster h. The time complexity of this step is O(N).

However, since the A_h matrix is updated in each iteration of MPCK-Means (Step 2.3, Algorithm 1), the estimate $\left(x_h^{'}, x_h^{''}\right)$ has to be updated accordingly as well. We embed the updating process into the calculation of the parameterized Euclidean distance $\left\|x_i - x_j\right\|_{A_h}^2$ (Eq. (2)). The parameterized Euclidean distance is calculated in Eq. (3) and Eq. (4) as well. The idea here is that when a pair of points (x_i, x_j) is found to have a greater distance than the current estimate $\left(x_h^{'}, x_h^{''}\right)$ at the time of evaluating the parameterized Euclidean distance, we will set the maximally separated points estimate to (x_i, x_j). The advantage of this approach is that it is an O(1) operation and does not increase the order of complexity of eMPCK-Means. However, this is an approximation because suppose, in the loop to iterate through all the points, we are at point x_A and are calculating $||x_A\text{-}x_B||^2$. The point x_C is to be considered in a later iteration and (x_A, x_C) happens to be the farthest pair of points. Then, the computation for point x_A will not have the accurate distance for the farthest pair of points. Hereafter, when we refer to Euclidean distance computation, we mean that it has maximally separated point estimation embedded within it.

To insure that f_C(.) function (Eq. (4)) does not evaluate to negative values with our approximated estimation of $\left(x_h^{'}, x_h^{''}\right)$, we enforce that the second term is always evaluated before the first term so that there is an opportunity to update $\left(x_h^{'}, x_h^{''}\right)$.

4.5.3 Use only a fixed number of constraints in cluster assignment step

In the cluster assignment step of MPCK-Means (Step 2.1, Algorithm 1), rather than iterating through the complete must-link/cannot-link peers of x_i, which makes Step 2.1 O(N^2), we choose a fixed-sized subset of them. This corresponds to Step 3.1 in eMPCK-Means. This optimization is hinted at by the fact that the must-link/cannot-link information in our domain has significant redundancy. A set of k_1 and k_2 calls placed, through user feedback, in the SPIT and non-SPIT categories generates $k_1^2+k_2^2$ must-link and k_1k_2 cannot-link constraints. On the other hand, we see from experimental results in [6] that MPCK-Means can work reasonably well even with a limited numbers of constraints. The cluster assignment step thus becomes O(N). In general, this can negatively affect the clustering quality. However, we believe it is a trade-off that is necessary in an effort to make the detection scheme scalable.

4.5.4 Pre metrics update on the starting cluster(s)

In MPCK-Means, the first update metrics step (Step 2.3) occurs only after the first iteration of the cluster assignment step (Step 2.1). In the first iteration of the cluster assignment, a default identity matrix is assigned to A_h, which directly affects the quality of the generated clusters from the first iteration and has a long-term effect on the quality of the eventual clusters as we see empirically. Therefore, in eMPCK-Means we conduct a metrics update (Step 1.2, eMPCK-Means, Algorithm 2) early on, right after the initial clusters are generated from the cluster initialization step. Intuitively, the user feedback is available at the outset and this optimization allows the A_h matrix to immediately adapt to the user feedback, which results in more accurate clustering. Additionally, it improves the convergence speed as we see later (Table 1).

Input: Set of data points $X = \{x_i\}_{i=1}^N$, Set of must-link constraints $M = \{(x_i, x_j)\}$, Set of cannot-link constraints $C = \{(x_i, x_j)\}$, Number of clusters K, Sets of constraints costs W and $\overline{W}$, Optional initial cluster centroids $\{\mu_h^{(0)}\}_{h=1}^{K}$, $t \leftarrow 0$

Output: Disjoint K-partitioning $\{X_h\}_{h=1}^{K}$ of X such that objective function τ_{mpckm} is locally minimized.

Method:

1. If initial cluster centroids $\{\mu_h^{(0)}\}_{h=1}^{K}$ is not given in the input

1.1. Create the λ neighborhoods $\{N_P\}_{P=1}^{\lambda}$ with steps from Sec. 4.5.1.

if $\lambda \geq K$

Use weightiest farthest-first traversal to select K neighborhoods $\{N_{P(h)}\}_{h=1}^{K}$.

Assign the data points $\{X_h^{(0)} \leftarrow N_{P(h)}\}_{h=1}^{K}$

Initialize $\{\mu_h^{(0)}\}_{h=1}^{K}$

Else

$\{X_h^{(0)} \leftarrow N_h\}_{h=1}^{\lambda}$

Initialize remaining clusters at random

Initialize $\{\mu_h^{(0)}\}_{h=1}^{K}$

1.2. Update metrics A_h for all clusters $\{X_h\}_{h=1}^{K}$ ([6]).

2. Initialization of maximally separated points (x'_h, x''_h) with respect to each A_h.

3. Repeat until convergence

3.1. For each $x_i \in X$

Randomly select $\begin{array}{l} \overline{M} \in \{(x_i, x_j) \in M\},\ |\overline{M}| = cts_{size} \\ \overline{C} \in \{(x_i, x_j) \in C\},\ |\overline{C}| = cts_{size} \end{array}$.

$$h^* = \arg\min_h \Big(\|x_i - \mu_h^{(t)}\|^2_{A_h} - \log(\det(A_h))$$
$$+ \sum_{(x_i,x_j)\in\overline{M}} w_{ij} f_M(x_i,x_j) 1[h \neq l_j] + \sum_{(x_i,x_j)\in\overline{C}} \overline{w_{ij}} f_C(x_i,x_j) 1[h = l_j] \Big)$$

Assign x_i to $X_{h^*}^{t+1}$

3.2. For each cluster X_h, { $\mu_h^{(t+1)} \leftarrow \left(\sum_{x\in X_h^{t+1}} x\right) \Big/ \left|X_h^{t+1}\right|$ }

3.3. Update_metrics A_h for all clusters $\{X_h\}_{h=1}^{K}$ ([6])

3.4. $t \leftarrow t+1$

Algorithm 2. eMPCK-Means

Algorithm 2 shows the proposed eMPCK-Means with the above modifications to MPCK-Means. Step 1 decides the starting K centroids (means) for the clusters through the use of initial user feedback. For the specific case of the user flagging calls as SPIT or non-SPIT, K=2.

Step 2 initializes the maximally separated points estimation. Step 3.1 performs the cluster assignment. Step 3.2 updates the mean. Note that the mean can be updated in constant time by keeping the sum of the data points and performing an addition/subtraction when a data point is associated with/unassociated from a cluster. Step 3.3 updates the matrix A_h for each cluster h. The goal of this process is to pick A_h's such that the objective function (Eq. (1)) is minimized for the cluster assignment done in the current iteration of Step 3. Conceptually, this process will result in A_h's that puts higher weights on those features which are consistent among data points in the same cluster and lower weights on those that are less consistent.

4.6 Progressive MPCK-Means

The eMPCK-Means algorithm assumes that the data points are available in a batch, and is thus suited for Mode B (batch mode) detection (Sec. 3.3). To support Mode A per-call early detection, we create a variant called progressive MPCK-Means (pMPCK-Means). The pseudo code is given as Algorithm 3. The idea here is that when a new call comes in, pMPCK-Means performs only the cluster assignment step and only for the new data point. The features "From URI", "To URI", "Start time", and "Time from the last call by the same caller" are available at the beginning of the phone call and are used in pMPCK-Means. For the features that are not available, pMPCK-Means fills the data point x_i with the mean values from the cluster to which this point's distance is being computed. This is implicitly carried out in Step 4 of Algorithm 3.

In pMPCK-Means, the update metrics operation only occurs occasionally when the cluster means have changed significantly (exceeding a given threshold $d_{\text{threshold}}$). Estimating the mean is an O(1) operation for each new data point. This amortizes over many calls the cost of A_h computation and the cost of re-clustering all existing data points. However, a cost has to be paid in advance, which is that we require reasonably sized cluster(s) to be grown on the initial data points ($|X| > t_{\text{threshold}}$) through eMPCK-Means. The reason is that we want the initial A_h matrix to be as accurate as possible.

Algorithm: pMPCK-Means

Input: A new data point x_t. , Disjoint K-partitioning $\{X_h^{(t-1)}\}_{h=1}^{K}$ of $X^{(t-1)} = \{x_1, x_2, .., x_{t-1}\}$.

Output: The cluster association l_t for the point x_t.

Disjoint K-partitioning $\{X_h^{(t)}\}_{h=1}^{K}$ of $X^{(t)} = \{x_1, x_2, .., x_{t-1}, x_t\}$.

Internal Variables: $\{\widehat{\mu_h}\}_{h=1}^{K}$

Method:

1. If t < $t_{\text{threshold}}$

$X^{(t)} \leftarrow X^{(t-1)} \cup \{x_t\}$; $\{X_h^{(t)} \leftarrow \varnothing\}_{h=1}^{K}$; Return

2. If $\{X_h^{(t)} = \varnothing\}_{h=1}^{K}$ (all clusters are empty)

$X^{(t)} \leftarrow X^{(t-1)} \cup \{x_t\}$.

Call eMPCK-Means to generate $\{X_h^{(t)}\}_{h=1}^{K}$ from $X^{(t)}$.

$\{\widehat{\mu_h} \leftarrow \mu_h^{(t)}\}_{h=1}^{K}$; Return

3. Randomly select $\begin{array}{l} \overline{M} \in \{(x_i, x_j) \in M\},\ |\overline{M}| = cts_{size} \\ \overline{C} \in \{(x_i, x_j) \in C\},\ |\overline{C}| = cts_{size} \end{array}$.

4. $$h^* = \arg\min_h \Big(\|x_i - \mu_h^{(t)}\|^2_{A_h} - \log(\det(A_h))$$
$$+ \sum_{(x_i,x_j)\in\overline{M}} w_{ij} f_M(x_i,x_j) 1[h \neq l_j] + \sum_{(x_i,x_j)\in\overline{C}} \overline{w_{ij}} f_C(x_i,x_j) 1[h = l_j] \Big)$$

5. $\{X_h^{(t)} \leftarrow X_h^{(t-1)}\}_{h=1}^{K}$; $X_{h^*}^{(t)} \leftarrow X_{h^*}^{(t)} \cup \{x_t\}$

6. If $\left\|\widehat{\mu_{h^*}} - \mu_{h^*}\right\|^2_{A_{h^*}} / \left\|x'_{h^*} - x''_{h^*}\right\|^2_{A_{h^*}} > d_{\text{threshold}}$

/* x'_{h^*}, x''_{h^*} are the maximally separated points wrt A_{h^*} */

Call eMPCK-Means with initial centroids $\left\{\mu_h^{(t)}\right\}_{h=1}^{K}$ to generate $\left\{X_h^{(t)}\right\}_{h=1}^{K}$ on $X^{(t)}$; $\left\{\widehat{\mu_h} \leftarrow \mu_h^{(t)}\right\}_{h=1}^{K}$.

Algorithm 3. pMPCK-Means

4.7 Multi-Class eMPCK clustering

We create a variant of eMPCK in which the initial clusters are split into sub-clusters based on the call types "calls going to voice mail", "calls terminated immediately after the call is established", and "the remaining calls". These three types exhibit different patterns in the non-silence call duration ratio (feature 17, Sec. 4.2). The sub-clusters are formed for both SPIT and non-SPIT calls. This is an attempt to guide the clustering process through expert knowledge. The user feedback however is only able to differentiate between SPIT and non-SPIT calls, and not place a call into a sub-cluster.

5. Experiments and Results

5.1 Testbed

We set up a two-domain testbed with a topology similar to Figure 1, one of the domains being protected by our detection technique. We use Asterisk as the VoIP proxy servers and MjSip for the phone clients. Each domain has 90 phones acting as non-spitters and 6 phones acting as spitters. We use the Poisson distribution to model call arrival times and the Exponential distribution to model call durations.

The generation of call traces was done by only one of the co-authors without providing any information about the nature of non-SPIT and SPIT calls to the rest of the team. This was done by design so that the team working on the detection system does not have any prior knowledge of the call mix. Ideally we would have liked to perform the evaluation on third-party call traces. However, at the time of writing, no such call trace is publicly available.

5.2 Summary of call trace dataset

We collected four call traces from our testbed with varying call characteristics as follows (*call trace name*, *Non-SPIT Call length average*, *Non-SPIT Call inter-arrival time average*, *SPIT Call length average*, *SPIT call inter-arrival time average*, *Number of SPIT calls in trace*, *Number of non-SPIT calls in trace*)*: (v4, 5, 30, 1, 2, 212, 171), (v5, 5, 10, 1, 10, 45, 338), (v6, 5, 30, 1, 10, 94, 289), (v7, 5, 30, 5, 10, 81, 302).* The time unit is minute. In terms of similarity between SPIT and non-SPIT calls, in decreasing order, the call traces are v5, v7, v6, and v4.

There are other characteristics which are shared by the four call traces. Examples include a 60% chance of a call being hung up by the caller for a non-SPIT call and a 10% chance of being hung up by the caller (spitter) for a SPIT call. The media streams for a SPIT call are dominated by the spitter while for a non-SPIT call, the non-silence duration on the caller and the callee media streams are about the same on average.

Other experimental parameter settings are: at most 15 must-link and 15 cannot-link constraints are used. The pMPCK-Means algorithm uses 100 data points initially with eMPCK-Means before commencing incremental operation. Each data point in the experiment is based on the average from 50 runs with the same parameter settings.

5.3 Effect of proportion of user feedback

We evaluate the effect of the proportion of calls that come with user feedback. We assume the same ratio for both SPIT and non-SPIT calls. We assume the feedback is perfectly accurate.

Figure 2 shows the clustering quality with respect to four different algorithms proposed on call trace 4 in terms of the F-Measure [6]. A larger F-Measure value means better quality clustering. From Sec. 5.2, we know that call trace 4 exhibits a very clear distinction between SPIT and non-SPIT calls in terms of call duration and call inter-arrival time. This makes eMPCK perform well with user feedback ratio as low as 0.1. The original MPCK-Means achieves the same level but with a higher user feedback ratio of 0.2. The improved result of eMPCK is due to the pre-metrics update (Sec. 4.5.4), which creates a more accurate weight matrix *A* based on user feedback, prior to iterating over the data points. The F-Measure from eMPCK Multi Class drops with increasing user feedback ratio because we break the cluster into sub-clusters based on the call types. As a result, eMPCK Multi Class will put different types of SPIT and non-SPIT calls into different sub-clusters. Both will hurt the F-Measure since by definition of F-Measure, these calls should be clustered into the same cluster. This negative effect grows stronger as the user feedback ratio increases.

Figure 3 and Figure 4 show the true positive (TP) and false positive (FP) rates of SPIT detection on call trace v4. What we can see here is that eMPCK Multi Class actually performs well despite the poor F-Measure. eMPCK Multi Class performs worse than eMPCK at low user feedback ratio because breaking the initial cluster into sub-clusters reduces the number of call data points with feedback in each sub-cluster. This results in poor clustering and hence low detection accuracy. Compared to eMPCK, MPCK's detection accuracy lags behind due to the lack of pre-metrics updating. pMPCK performs rather poorly even with call trace v4. However, it is still in the usable range (e.g. 0.63 True Positive with a user feedback ratio of 0.2). pMPCK's poor performance is due to the limited features available before the media stream is established.

Due to space constraints, we show only the True Positive curves for call traces v5, v6, and v7 in Figure 5, Figure 6, and Figure 7 respectively. All the algorithms perform worse with call trace v5 due to same inter-arrival time of SPIT and non-SPIT calls. This makes the time since last call from the same caller and call frequency (features 9 and 16 in Sec. 4.2) much less useful. Another factor is the number of SPIT calls in the call trace is decreased to 45 (compared to 212 in v4) which further lowers the clustering quality and detection accuracy. Figure 8 summarizes the True Positive rates from eMPCK across the four call traces. This basically corresponds to how salient the differences between SPIT calls and non-SPIT calls in the call traces are. In order, the easiest one is v4, followed closely by v6, and then v7. The hardest is v5. In v5, SPIT calls are almost indistinguishable from short-duration non-SPIT calls.

We show error-bar ($\pm$1 s.t.d.) for eMPCK in Figure 2. They are omitted in the rest of the figures for presentation clarity. The general trend is that the errors diminish with increasing ratio of user feedback. We observe less than $\pm$5% error across the experiments on call traces 4, 6, and 7 when user ratio is set beyond 0.1. For call trace 5, the error is higher (up to $\pm$25% at 0.1 ratio).

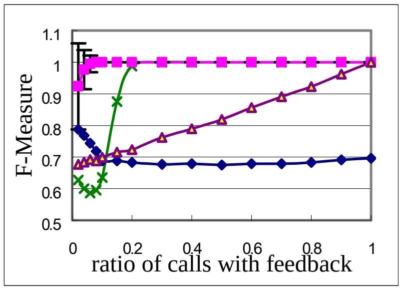

Figure 2. Call trace v4 / F-Measure

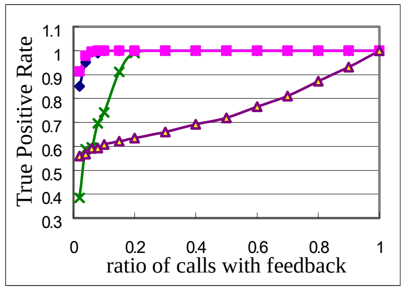

Figure 3. Call trace v4 / TP

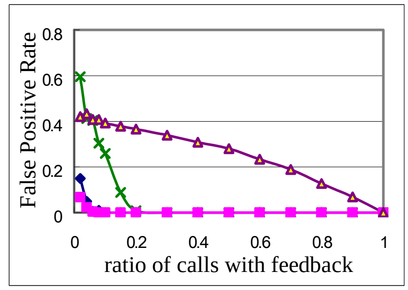

Figure 4. Call trace 4 / FP

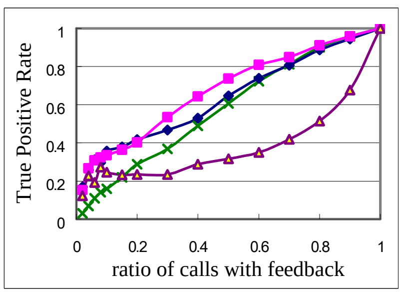

Figure 5. Call trace 5 / TP

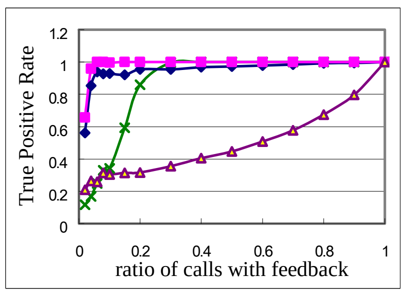

Figure 6. Call trace 6 / TP

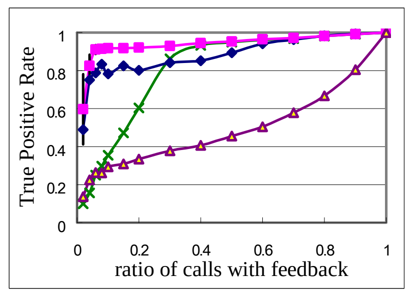

Figure 7. Call trace 7 / TP

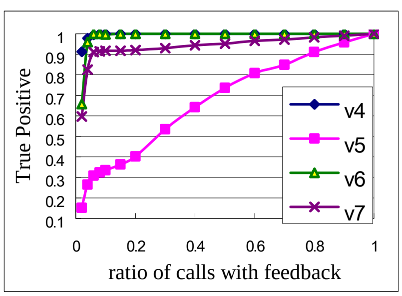

Figure 8. Compare eMPCK True Positive Rate across call traces

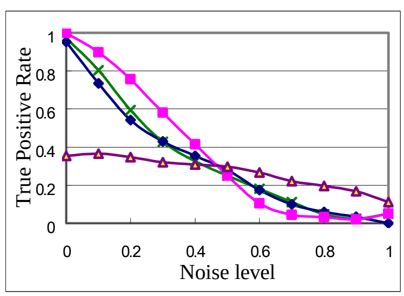

Figure 9. TP vs. Noise in User Feedback

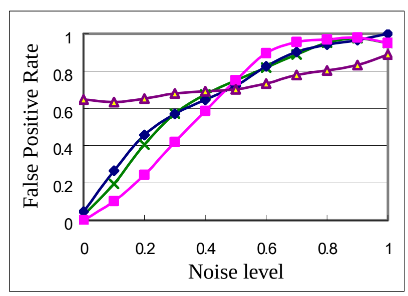

Figure 10. FP vs. Noise in User Feedback

5.4 Scalability of execution time

In this experiment we compare the running times of MPCK and eMPCK by varying the number of call data points. Call trace v7 is used for this experiment. For MPCK, we apply exact optimizations which do not cause loss of accuracy. For example, the maximally separated points evaluation is re-executed only when the A matrix gets changed. The results are based on code compiled with

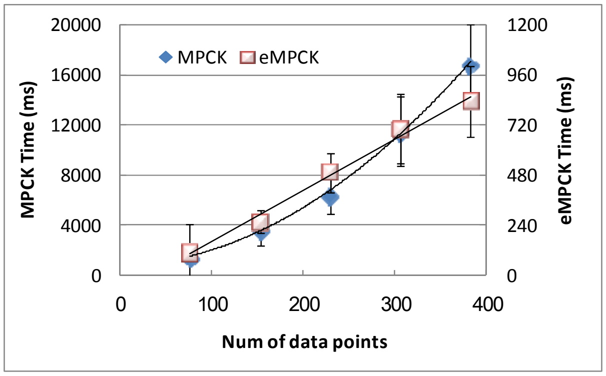

Figure 11. Running Time

MS VC++ 8.0 with default optimization level running on Windows XP, Intel E6400 2.13 GHz CPU.

As Figure 11 shows, MPCK exhibits non-linear growth in the running time as the number of call data points increases (error bars are ±1 std.). eMPCK, on the other hand, exhibits a linear growth in the running time. Also, MPCK takes significantly longer to run compared with eMPCK—15 times longer for a batch of 400 calls. Looking at the number of iterations that each algorithm takes to converge (Table 1), eMPCK fares better. The running time advantage of eMPCK comes from the lower number of iterations as well as the lower running time of each iteration. The lower number of iterations is explained by eMPCK's update of A_h's on the initialized clusters. For call trace v5, the similarity in SPIT and non-SPIT calls renders the A_h initialization ineffective and the number of iterations is roughly equal for both algorithms.

5.5 Effect of noise in user feedback

	MPCK	eMPCK
v4	6.94	3.98
v5	7.80	7.83
v6	7.81	5.38
v7	6.94	4.7
Average	7.37	5.47

Table 1. Number of iterations to convergence

We evaluated different algorithms with various noise levels in the user feedback. When we say the noise level is *c*, it means that a fraction *c* of the user feedback is false, i.e., a SPIT call is reported as non-SPIT and vice-versa. We show the result with call trace 6 for this experiment. The user feedback ratio is fixed at 0.3. Figure 9 shows the true positive rate decreases as the noise level increases. Observing the false positive rates in Figure 10, we conclude that pMPCK is completely unusable through the whole noise level range while the other algorithms are usable at low noise levels. We conclude that pMPCK is usable only for a high proportion of accurate user feedback. Beyond noise level 0.5 eMPCK performance drops below that of MPCK due to our design of the detection predicate (Sec. 4.4.2), namely, considering the cluster that contains more calls marked by the user as SPIT than non-SPIT, to be the SPIT cluster. With noise level above 0.5, the user feedback is wrong more often than right and the negative effect is more pronounced in eMPCK than MPCK, since it did a "better job" of clustering on the user feedback than MPCK. As an example of a usable operating point, consider that at noise levels 0.2 or below, eMPCK has both true positive and true negative above 0.8.

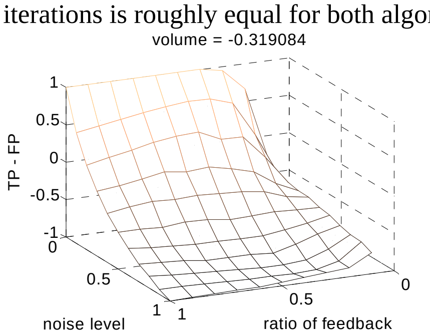

Figure 12. MPCK (TP – FP) for call trace v6

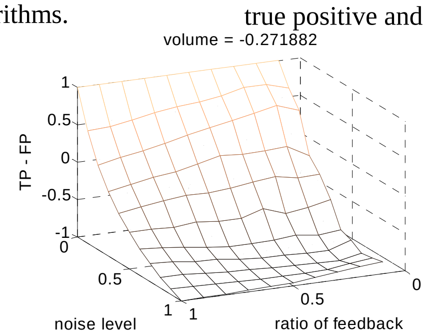

Figure 13. eMPCK (TP – FP) for call trace v6

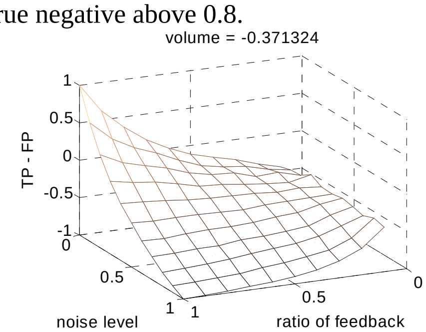

Figure 14. pMPCK (TP – FP) for call trace v6

5.6 Evaluation with noise and feedback ratio

Here we perform an evaluation of all four proposed algorithms with respect to the four call traces. Our evaluation methodology considers the combined effect of proportion of user feedback and the noise level and the results are shown in Figure 12, Figure 13, and Figure 14. In the 3D plot, the Z-axis corresponds to TP-FP, the difference between True Positive rate and False Positive rate, with respect to each pair of feedback ratio and noise level. Intuitively, if TP-FP is greater than zero, it means the detection gives more correct results than incorrect results and can be regarded as a valid operating point where the detection is useful. Due to page length limitation, we show the 3D plots only for call trace 6. A general trend we can see in the 3D plots is that when fixing the noise level, the TP-FP value climbs to a peak and then goes down when varying the feedback ratio from 0 to 1. There is no sharp breakdown of performance for any of the algorithms. If the user feedback is accurate, then even with low ratio of user feedback, the performance is good for MPCK and eMPCK. The performance of pMPCK on the other hand is acceptable only close to the extreme region of almost perfect user feedback for almost all calls. To give an overall quantification of the detection quality, we define the volume metric based on the integral (Eq. (6)). In the ideal case where TP-FP is maintained at 1 through the entire range of noise levels and feedback ratio values, the volume will be 0.9. Table 2 shows the volume for each combination of algorithm and call trace. Call trace v5 gives the lowest volume corresponding to the worst performance for all algorithms. Averaged over the entire range, we see that eMPCK performs best followed by eMPCK (Multi Class), MPCK, and pMPCK.

$$Volume = \int_{n=0}^{1} \int_{f=0.1}^{1} (TP - FP) \bullet df \bullet dn \qquad (6)$$

n: noise level, f: feedback ratio

TP-FP Volume	v4	v5	v6	v7	avg.
MPCK	0.048	-0.595	-0.319	-0.388	-0.314
eMPCK (Multi Class)	0.068	-0.590	-0.330	-0.402	-0.314
eMPCK	0.042	-0.577	-0.272	-0.340	-0.287
pMPCK	0.015	-0.596	-0.371	-0.411	-0.341

Table 2. Summary of TP-FP volume comparisons

6. CONCLUSION

In this paper, we proposed a new approach to detect SPIT calls in a VoIP environment. We map each phone call into a data point based on an extendable set of call features, derived from the signaling as well as the media protocols. This converts the problem of SPIT detection into a data classification problem, where a classic solution is the use of clustering. We apply semi-supervised clustering, which allows for the optional use of user feedback for more accurate classification. This corresponds to users' flagging some calls as SPIT and others as legitimate. We create a new algorithm called eMPCK-Means, based on a previous algorithm called MPCK-Means, which provides linear time performance with the number of calls. eMPCK-Means includes a pre-metrics-update step, which contributes to high (> 90%) detection true positive rates with less than 10% user feedback data points for three of the four call traces used here. We found that it is difficult to attain high detection accuracy based only on features available in the call establishment phase, which would enable a SPIT call to be dropped without the user needing to answer the call. This algorithm pMPCK performs well only with accurate user feedback for a majority of calls.

7. REFERENCES

[1] VOIPSA, "VoIP Threat Taxonomy," 2008.

[2] Y. S. Wu, S. Bagchi, S. Garg, and N. Singh, "SCIDIVE: a stateful and cross protocol intrusion detection architecture for voice-over-IP environments," in *DSN*, 2004, pp. 433-442.

[3] H. Sengar, D. Wijesekera, H. Wang, and S. Jajodia, "VoIP Intrusion Detection Through Interacting Protocol State Machines," in *DSN*, 2006, pp. 393-402.

[4] C. J. J. Rosenberg, "RFC 5039 : The Session Initiation Protocol (SIP) and Spam," 2008.

[5] D. Shin, J. Ahn, and C. Shim, "Progressive Multi Gray-Leveling: A Voice Spam Protection Algorithm," *IEEE Network*, vol. 20, pp. 18-24, 2006.

[6] M. Bilenko, S. Basu, and R. J. Mooney, "Integrating constraints and metric learning in semi-supervised clustering," in *ICML*, 2004, pp. 81-88.

[7] J. Quittek, S. Niccolini, S. Tartarelli, M. Stiemerling, M. Brunner, and T. Ewald, "Detecting SPIT Calls by Checking Human Communication Patterns," in *ICC*, 2007, pp. 1979-1984.

[8] R. MacIntosh and D. Vinokurov, "Detection and mitigation of spam in IP telephony networks using signaling protocol analysis," in *IEEE/Sarnoff Symposium on Advances in Wired and Wireless Communication*, 2005, pp. 49-52.

[9] P. Kolan and R. Dantu, "Socio-technical defense against voice spamming," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 2, 2007.

[10] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967, p. 14.

[11] P. Haider, U. Brefeld, and T. Scheffer, "Supervised clustering of streaming data for email batch detection," in *ICML*, 2007, pp. 345-352.

[12] M. Sasaki and H. Shinnou, "Spam Detection Using Text Clustering," in *International Conference on Cyberworlds*, 2005.

[13] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, pp. 121-167, 1998.

[14] G. Druck, C. Pal, A. McCallum, and X. Zhu, "Semi-supervised classification with hybrid generative/discriminative methods," in *KDD*, 2007, pp. 280-289.

[15] K. Bennett and A. Demiriz, "Semi-supervised support vector machines," *Advances in Neural Information processing systems*, pp. 368-374, 1999.

[16] J. Rosenberg, "RFC 3261 - SIP: Session Initiation Protocol," 2002.

[17] H. Schulzrinne, "RFC 1889 - RTP: A Transport Protocol for Real-Time Applications," 1996.

[18] N. Grira, M. Crucianu, and N. Boujemaa, "Unsupervised and Semi-supervised Clustering: a Brief Survey," *A Review of Machine Learning Techniques for Processing Multimedia Content', Report of the MUSCLE European Network of Excellence (FP6)*, 2004.

[19] T. Finley and T. Joachims, "Supervised clustering with support vector machines," in *ICML*, 2005, pp. 217-224.

[20] voip-info.org, "Asterisk SIP Media Path."