# Resisting Tag Spam by Leveraging Implicit User Behaviors

Ennan Zhai
Yale University
ennan.zhai@yale.edu

Zhenhua Li
Tsinghua University
lizhenhua1983@tsinghua.edu.cn

Zhenyu Li
ICT, CAS
zyli@ict.ac.cn

Fan Wu
Shanghai Jiao Tong University
fwu@cs.sjtu.edu.cn

Guihai Chen
Nanjing University
gchen@nju.edu.cn

## ABSTRACT

Tagging systems are vulnerable to tag spam attacks. However, defending against tag spam has been challenging in practice, since adversaries can easily launch spam attacks in various ways and scales. To deeply understand users' tagging behaviors and explore more effective defense, this paper first conducts measurement experiments on public datasets of two representative tagging systems: Del.icio.us and CiteULike. Our key finding is that a significant fraction of correct tag-resource annotations are contributed by a small number of *implicit similarity cliques*, where users annotate common resources with similar tags. Guided by the above finding, we propose a new service, called Spam-Resistance-as-a-Service (or SRaaS), to effectively defend against heterogeneous tag spam attacks even at very large scales. At the heart of SRaaS is a novel reputation assessment protocol, whose design leverages the implicit similarity cliques coupled with the social networks inherent to typical tagging systems. With such a design, SRaaS manages to offer *provable guarantees* on diminishing the influence of tag spam attacks. We build an SRaaS prototype and evaluate it using a large-scale spam-oriented research dataset (which is much more polluted by tag spam than Del.icio.us and CiteULike datasets). Our evaluational results demonstrate that SRaaS outperforms existing tag spam defenses deployed in real-world systems, while introducing low overhead.

## 1. INTRODUCTION

Today's tagging systems – employed by Del.icio.us [4], CiteULike [2], BibSonomy [1], *etc.* – have quickly gained enormous popularity in recent years. They facilitate users' finding the resources of their interests based on the tags posted by other participants. In general, each *resource* in a tagging system (*e.g.*, a web page on Del.icio.us) is annotated with multiple *tags*. When a user issues a tag query (called a *tag search*), the system returns resources associated with that tag. Then, the user may consume some of the returned results, and annotate these consumed resources with some tags.

Many efforts, nevertheless, indicate that tagging systems are vulnerable to *tag spam* [14, 15, 17]. In the launch of a typical tag spam attack, malicious users (or spam attackers), with an intention to mislead normal users, generate and annotate a particular target resource with numerous erroneous and irrelevant tags. The large-scale tag spam attacks in practice would make normal users without sufficient "experience" frequently consume unwanted resources, thus adversely affecting the usability of tagging systems [14].

### 1.1 Motivation

Defending against tag spam has been challenging in practice, since adversaries can easily take advantage of tagging systems' nature – allowing users to create a number of tags with any personal choice of keywords – to launch spam attacks in various ways and scales. A variety of approaches have been used to defend against spam attacks, which can be roughly classified into two categories: *static detection approaches*, and *dynamic assessment approaches*.

The static detection approaches [9, 12, 19, 20, 22, 36] typically use machine learning and data mining techniques to analyze tagging system datasets, thus detecting, identifying and removing tag spam matching certain target characteristics. However, these solutions are limited to specific tag spam patterns and datasets, and usually introduce much computational overhead to the back-ends of tagging systems, due to amount of analysis tasks.

The dynamic assessment approaches [17, 21, 35, 38, 40, 41] leverage users' activities to evaluate the correctness of their posted tags, thus prioritizing the most relevant tag search results and degrading the ranks of misleading results to the end of the search result list. The major advantage of dynamic assessment efforts lies in that their effectiveness is not restricted to particular tag spam patterns or datasets, because they do not require direct identification and removal of the numerous "pattern matching" misleading tags. Instead, they gradually diminish the impact of attackers and the misleading tags through result ranking and updating. In addition, dynamic assessment techniques do not introduce much computational overhead to the tagging systems' back-ends, as they do not involve computationally expensive analysis. As a consequence, the dynamic assessment approaches are generally considered more effective in resisting massive, heterogeneous tag spam attacks [17, 39, 41].

Nevertheless, existing dynamic assessment solutions [17, 21, 35, 38, 40, 41] are still subject to heuristic algorithms whose designs focus on addressing particular attack strategies observed from measurement traces rather than arbitrary forms of (*i.e.*, heterogeneous) attack strategies. Even worse, none of them has ever offered provable or quantifiable guarantees on the defense capability (*e.g.*, a theoretical boundary). In other words, existing efforts fail to provide theoretical proofs on "how well" these solutions are, or "what types" and "what scales" of attacks they can defend against.[1]

The status quo motivates us to raise a challenging question: *Is it possible to design a new dynamic assessment scheme that can*

[1]Provable guarantees are important to tag spam defenses, since any experiment-based evaluations cannot cover heterogeneous attacks.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org.
*Proceedings of the VLDB Endowment,* Vol. 10, No. 3
Copyright 2016 VLDB Endowment 2150-8097/16/11.

*defend against heterogeneous tag spam attacks and hold provable guarantees on the defense capability. Meanwhile, the proposed scheme should not introduce much computational overhead.*

## 1.2 Our Approach and Contribution

To answer the above question, this paper first conducts measurement experiments on datasets of two representative tagging systems: Del.icio.us and CiteULike, in the hopes of deep understanding users' tagging behaviors and exploring more effective dynamic assessment defense. Suppose we use an annotation, $\langle t, r\rangle$, to denote the tagging relation between tag $t$ and resource $r$. Our measurement results reveal: 1) a small number (say, $M$) of groups of users in a typical tagging system contribute a significant fraction (typically 70%) of correct annotations; and 2) the users within each of the groups have a number of overlapping annotations. We call each of these groups an *implicit similarity clique* (detailed definition in §3), because these similarity cliques only exist at the logical level. In other words, no party explicitly maintains such cliques and nobody explicitly knows which users belong to which cliques.

Guided by the above insights, we propose a new dynamic assessment scheme, called Spam-Resistance-as-a-Service (or SRaaS), to effectively defend against tag spam attacks. At the heart of SRaaS is a social network-based personalized reputation assessment protocol. While there have been several efforts that use reputation-based techniques for dynamic assessment defenses [35, 40, 41], the social network-based reputation algorithm of SRaaS further leverages implicit similarity cliques to offer a stronger defense capability – for all types of tag spam attacks to our knowledge even at very large scales, SRaaS can bound each user's *loss* (*i.e.*, the total number of unwanted consumed resources) within $O(M)$. As mentioned above, $M$ is in fact the number of implicit similarity cliques that publish a significant fraction of correct annotations in the tagging system. Given that $M$ is relatively small in practice, the loss of each user should be well acceptable. On the contrary, existing reputation algorithms [33, 37] usually work on specific attack strategies and scales (*e.g.*, the number of attackers should be smaller than the number of honest users), and fail to offer strong or provable guarantees on the defense capability.

We build a prototype tagging system equipped with SRaaS, and evaluate the performance and capability of SRaaS based on a large-scale tag spam oriented research dataset [5]. This research dataset is much more polluted by tag spam than datasets from regular tagging systems such as Del.icio.us and CiteULike datasets. In the performance evaluation, we observe that SRaaS introduces less than one second's delay to each user's search, even in a dataset containing 10 million tag-resource annotations. In the defense capability evaluation, we compare SRaaS with three prevalent dynamic assessment schemes (Boolean [7], Occurrence [6], and Coincidence [17]) and a static detection scheme (SpamDetector [22]) under three representative tag spam attacks. The results indicate that SRaaS greatly outperforms the existing efforts. For example, in an extremely spam polluted environment where each resource is annotated with about 500 misleading tags, a given SRaaS user can always obtain spam-free search results after about 15 tag queries. In comparison, the search results of the other four schemes are still occupied by tag spam even after 50 queries.

In summary, this paper makes three main contributions:

- We (are the first to) discover the existence and notice the value of implicit similarity cliques in real-world tagging systems (§3).
- By leveraging the implicit similarity cliques, we propose SRaaS to effectively diminish the influence of massive, heterogeneous tag spam attacks with provable user-loss guarantees (§4).

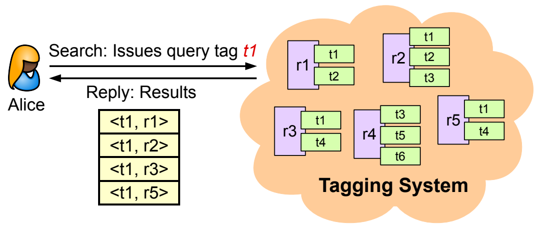


**Figure 1: An example execution for a typical user Alice's search and reply, *i.e.*, the first step described in §2.2. First, Alice issues a tag search with query tag $t_1$. Then, the tagging system responds to her by returning search results which contain matched annotations: $\langle t_1, r_1\rangle$, $\langle t_1, r_2\rangle$, $\langle t_1, r_3\rangle$ and $\langle t_1, r_5\rangle$.**

- Our experiments running an SRaaS prototype on a large-scale dataset demonstrate that SRaaS not only outperforms the four prevalent schemes under various tag spam attacks, but also introduces low system overhead (§5).

## 2. SYSTEM MODEL

SRaaS is a spam-resistant service applicable to typical tagging systems (*e.g.*, Del.icio.us and CiteULike) that possess certain properties. This section describes a tagging system model holding these properties, and key terminologies used throughout this paper.

## 2.1 System Components and Behaviors

**Users.** In a typical tagging system, users' purposes are to find resources of their interest. Users normally have a relatively long lifetime (*e.g.*, more than two weeks). They can be either honest users (*i.e.*, normal users) or spam attackers. We will define detailed behaviors of both types of users in §2.3.

**Tagging behaviors.** Users in tagging systems can annotate resources (*e.g.*, web pages in Del.icio.us) with certain tags. The relation tuple $\langle tag, resource\rangle$ that annotates a *resource* with a *tag* is called an *annotation*. Each user may annotate a resource with various tags and the same tag may only be applied once to each resource; otherwise, the redundant tags will be ignored by the system automatically. We say that a user *publishes* an annotation $\langle T, R\rangle$ if the user annotates the resource $R$ with the tag $T$. $T$ is called the tag of annotation $\langle T, R\rangle$, and $R$ is the resource of the annotation $\langle T, R\rangle$. Moreover, this user is called the *annotator* of this annotation. Note that an annotation may have multiple annotators, since it might be published by many users.

For any user, each annotation is either *correct* or *incorrect*. For example in Del.icio.us, if Alice finds an annotation $\langle T, R\rangle$, where $T$ is the tag "shoes" and $R$ is a web page about a dog, then she may say that this annotation is incorrect. In practice, whether a certain annotation is correct or not is somewhat subjective, since different users would have different opinions. Additionally, we say that two annotations are *the same* if and only if both resources and tags of these two annotations are the same; otherwise, we say that the two annotations are *distinct*.

**Social networks.** Users residing in a typical real-world tagging system can establish their social networks. Namely, each user can create her own friend relationships with other users. To offer a fast way for building social networks, tagging systems allow users to import their friend information from other social networking web sites (*e.g.*, Facebook and Twitter). Even better, many well-known tagging systems (*e.g.*, Flickr and Del.icio.us) allow users to log into the systems with their social network accounts (*i.e.*, the so-called "federated identity technique"), thus making the establishment of their friend information more convenient. In addition, each user in tagging systems can also create new friend relationships with users she is interested in.

## 2.2 Resource Discovery Execution

To discover the resources of interest, a user (say, Alice) needs to execute a *resource discovery process* consisting of the following two steps, as demonstrated in Figure 1.

**Step 1: Tag search & reply.** Alice first issues a tag search (we use $t$ to denote the tag in query) in a tagging system. Then, the system returns her *matched annotations* retrieved from the back-end database of the tagging system. *Matched* implies that the tags of the returned annotations are the same as the query tag $t$. So far, we say that Alice finishes one tag search with respect to the query tag $t$, and receives *search results* which contain all matched annotations. To elaborate clearly, we define $\{A_{t(i)}\}_{i=1}^{n}$ as the set of search results that match the query tag $t$, where $n$ denotes the size of the set, and $A_{t(i)}$ denotes the $i$-th annotation in the search results, which contains a resource annotated with tag $t$. For example in Del.icio.us, $\{A_{t(i)}\}_{i=1}^{n}$ may be all the annotations containing different web pages annotated with the query tag $t$.

**Step 2: Resource consumption.** With the search results in hand, Alice can pick one or more resources out of $\{A_{t(i)}\}_{i=1}^{n}$ to *consume*. Here, we borrow the terminology *consume* from the filed of recommendation systems. A consume could be accessing a web page in a bookmark-related tagging system (*e.g.*, Del.icio.us), watching a video in a video-related tagging system (*e.g.*, YouTube), or something else. In Figure 1, Alice first picks an annotation (*e.g.*, $A_{t(3)}$) out of the search results $\{A_{t(i)}\}_{i=1}^{n}$ and then consumes the selected resource of $A_{t(3)}$. After that, Alice annotates the consumed resources correctly (*i.e.*, publishes correct annotations with respect to the consumed resources) in her opinion. We assume that the fraction of correct annotations in $\{A_{t(i)}\}_{i=1}^{n}$ is at least $\gamma > 0$. For any user, each consumed resource is either *the resource of interest*, *i.e.*, the resource of a correct annotation, or *unwanted*, *i.e.*, the resource of an incorrect annotation.

## 2.3 Threat Model

In a typical tagging system, we assume that the system provider, *e.g.*, Del.icio.us provider, is honest. In contrast, users can be potentially malicious. A user is either an *honest user* (*i.e.*, normal user) or a *spam attacker* (*i.e.*, malicious user).

**Honest users.** We assume that there are $H$ honest users in the system. Some of them never publish incorrect annotations and the others seldom publish incorrect annotations. This assumption on honest users is necessary and reasonable in practice.

**Spam attackers.** To reflect practical and severe threats, we make three-fold assumptions on spam attackers. First, we assume that a spam attacker is intelligent and behaves arbitrarily. Spam attackers can also search, consume and annotate resources, but they typically annotate resources with misleading tags. Besides, we assume that spam attackers are capable of annotating numerous resources with a lot of (*e.g.*, thousands of) misleading tags, and they know which annotation is published by which honest user(s). In addition, spam attackers may collude. For the total number of spam attackers, $S$, we do not set any limitation to $S$, so $S > H$ is allowed.

## 3. INSIGHTS FROM MEASUREMENTS

To deep understand the impact of users' tagging behaviors on tagging systems and explore more effective defense, we conducted measurement experiments based on two public datasets [3, 26], which record traces of users' annotations and tagging behaviors in two representative tagging systems: Del.icio.us and CiteULike.

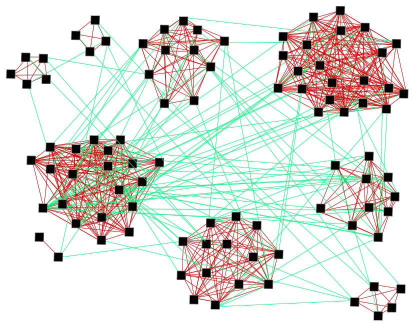


**Figure 2: An example for tagging similarity and similarity cliques. This result is extracted from the Del.icio.us dataset [26]. Black points mean users. Red and green lines denote high and low tagging similarity between two users, respectively.**

## 3.1 Terminologies

Before revealing our measurement results, we first define two important terminologies – *tagging similarity* and *implicit similarity clique* – we use throughout the rest of this paper.

**Tagging similarity.** By following the widely used cosine similarity principle [29], we define *tagging similarity* as the ratio of overlapping annotations (*i.e.*, identical annotations on the same resources) published by two users over the resources annotated by the two users in common. We use $S_{A,B}$ to denote the tagging similarity between two users $A$ and $B$, and use Equation (1) to compute $S_{A,B}$.

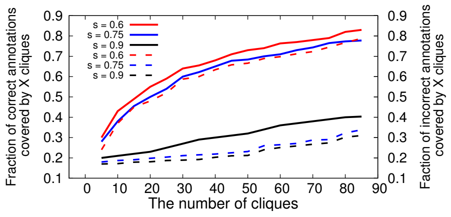
$$S_{A,B} = \frac{\sum_{r_j \in R} \left( \sum_{t_i \in C_{r_j}} |N(t_i, r_j)| \right)^2}{\left( \sqrt{\sum_{r_j \in R} \left( \sum_{t_i \in T_{A(r_j)}} |N(t_i, r_j)| \right)^2} \sqrt{\sum_{r_j \in R} \left( \sum_{t_i \in T_{B(r_j)}} |N(t_i, r_j)| \right)^2} \right)} \tag{1}$$

Here, $R$ is the set of resources annotated by $A$ and $B$ in common, and $r_j$ is the $j$-th resource of the common resource set $R$. $C_{r_j}$ is the set of the tags annotated by $A$ and $B$ in common to the resource $r_j$. $t_i$ denotes the $i$-th tag in a tag set. $T_{x(r_j)}$ means the set of tags annotated by the user $x$ to the resource $r_j$. $N(t_i, r_j)$ denotes the set of annotations that annotate $r_j$ with $t_i$ and $|N(t_i, r_j)|$ is the size of $N(t_i, r_j)$. The range of $S_{A,B}$ is $[0, 1]$, and a higher value indicates that $A$ and $B$ have more overlapping interests.

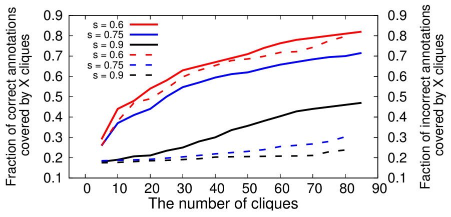
**Implicit similarity clique.** We define a *similarity clique* as a group of users, where the tagging similarities between the users are equal to or higher than a threshold $s$. Typically, the value of $s$ is a number between 0.5 and 1.0, say 0.75. §3.2 discusses how to determine the value of $s$ in practice.

Because similarity cliques only exist at the logical level, we also call similarity cliques as *implicit similarity cliques*. In other words, no party explicitly maintains such cliques, and nobody explicitly knows which users belong to which cliques. Note that similarity cliques are distinct from social network clusters or groups, because users in each similarity clique may not be friends.

**Example.** Figure 2 depicts an example for illustrating the above two terminologies. This result is extracted from a public Del.icio.us dataset [26]. In Figure 2, there are nine similarity cliques. Users (black points in Figure 2) within a similarity clique have high similarity, *e.g.*, $S_{A,B} \geq (s = 0.75)$, to each other. We use red lines between



(a) Del.icio.us [26]. Solid lines and dashed lines correspond to left-hand and right-hand Y-axis, respectively.



(b) CiteULike [3]. Solid lines and dashed lines correspond to left-hand and right-hand Y-axis, respectively.

**Figure 3: The relationship between the number of cliques and the fractions of annotations published by the cliques.**

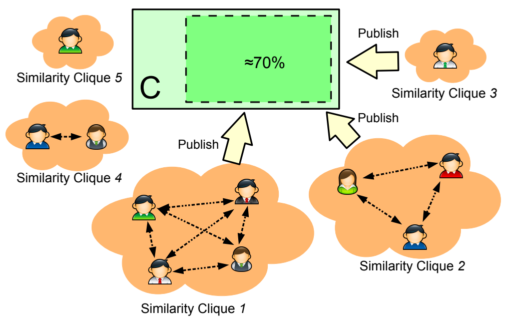


**Figure 4: An example for the important insight driving our design. $C$ (green box) is the set of all the correct annotations ever appeared in a tagging system. There are five similarity cliques. Each black double sided arrow denotes two users have similarity $\geq 0.75$. The minimal number of similarity cliques that can cover $0.7$ of $C$ is three. Thus, $M$ is 3 in this example.**

two black points to denote that the tagging similarity between two users is higher than 0.75. Also, there are many green lines across similarity cliques, meaning that the users of different cliques have tagging similarity less than 0.75. It is worth noting that there are a number of red lines across similarity cliques, but we choose to hide them so as to make the figure tidy.

## 3.2 Measurement Findings

For each of the studied datasets (*i.e.*, Del.icio.us dataset [26] and CiteULike dataset [3]), our measurement experiments include the following three steps:

- First, we compute and record each user's tagging similarity with respect to each of the other users in the dataset, thus determining (implicit) similarity cliques.
- Second, we manually extract both correct and incorrect annotations from the dataset. We use $C$ and $D$ to denote the set of all the correct and incorrect annotations, respectively.
- Third, we measure the relationship between $\xi \cdot C$ and $M$, where $\xi \cdot C$ denotes a fraction $\xi$ of correct annotations $C$, and $M$ is the number of similarity cliques. This measurement aims to indicate that a fraction $\xi$ of $C$ are published by $M$ similarity cliques. Similarly, we also measure the relationship between $\xi' \cdot D$ and $M$, where $\xi' \cdot D$ means a fraction $\xi'$ of incorrect annotations D.

Figure 3 shows our measurement results about the relationship between $M$ and the fraction of correct and incorrect annotations published by the $M$ similarity cliques. In our measurement, we select different values of the threshold $s = 0.6, 0.65, 0.7, 0.75, 0.8, 0.85$ and 0.9, to explore the effects of different $s$. Note that Figure 3 only shows the cases of $s = 0.6$, 0.75 and 0.9 to make the figure tidy. Form the measurement results, we have two major findings:

- When $s = 0.6$, a significant fraction of correct annotations (*e.g.*, 80% of $C$) are contributed by $M$ (*e.g.*, 50) implicit similarity cliques. However, these $M$ similarity cliques also contribute a significant fraction of incorrect annotations (*e.g.*, 70% of $D$).
- When $s = 0.9$, a small fraction of correct annotations (*e.g.*, 30% of $C$) and a small fraction of incorrect annotations (*e.g.*, 20% of $D$) are both posted by $M$ (*e.g.*, 50) similarity cliques.

Obviously, there is a tradeoff between the fractions of correct and incorrect annotations as $s$ varies. By carefully examining the performance of using various $s$, we figure out a "sweet spot": $s = 0.75$. When $s = 0.75$, a significant fraction of correct annotations (*e.g.*, 70% of $C$) and a small fraction of incorrect annotations (*e.g.*, 20% of $D$) are contributed by $M$ similarity cliques, and $M$ is relatively small (*e.g.*, 50). In practice, a tagging system administrator can use the above heuristic method to select a sound threshold for $S_{A,B}$ (that leads to the biggest "distance" between the line representing the fraction of correct annotations and the line representing the fraction of incorrect annotations). For example, in Figure 3a and Figure 3b, when $s = 0.75$, the distance between the solid line and the dashed line is the biggest, so we select $s = 0.75$ as the threshold.

In addition, there may exist different groups of similarity cliques that contribute the same fraction $\xi$ of $C$ in practice. Thus, the value of $M$ in Figure 3 is the minimal number of cliques that can contribute $\xi \cdot C$. For example in Figure 3a, 54 cliques in the Del.icio.us dataset can cover 0.7 of correct annotations. Here 54 is the minimal number of cliques that can cover 0.7 of $C$. In other words, it is impossible to cover 0.7 of $C$ with less than 54 cliques.

Figure 4 plots an example for our key finding. In particular, there are three similarity cliques (*i.e.*, Cliques 1-3 in Figure 4) that together contribute 70% correct annotations (*i.e.*, 0.7 of $C$). Let's assume similarity cliques $1, 2, 4$ and 5 can also cover 0.7 of $C$. However, in this example, $M$ should be 3, because the three is the minimal number of similarity cliques that can cover a significant fraction of correct annotations (*i.e.*, Cliques 1-3).

## 4. DESIGN OF SRaaS

By leveraging the implicit similarity cliques, we propose a new dynamic assessment scheme, SRaaS (Spam-Resistance-as-a-Service), to effectively diminish the influence of massive, heterogeneous tag spam attacks with provable user-loss guarantees. This section details the design of SRaaS.[2]

[2]While SRaaS's design is inspired by theoretical efforts in the recommendation system community [8, 27, 28, 39], recommendation systems are different from tagging systems. In recommendation systems, each user can only cast one vote on the same consumed resource to express their opinions on the resource's qualities (*i.e.*, good or bad). In contrast, tagging system users could annotate the same resources with multiple tags. More importantly, tags do not serve as a metric for evaluating the qualities of resources, but serve as keywords to convenient resource discovery.

**Algorithm 1:** SRaaS ranking algorithm for any typical SRaaS tagging system user, say Alice.

```
Input  : Search results $\mathbb{R}$
Output : $\mathbb{R}'$ which would be returned to Alice
1 begin
2     if there are one or more annotations whose reputation scores $\geq h$ in search results then
3         produce $\mathbb{R}'$ containing all annotations whose reputation scores $\geq h$;
4         order annotations in $\mathbb{R}'$ based on their reputations;
5     else
6         produce $\mathbb{R}'$ containing all annotations;
7         randomly order annotations in $\mathbb{R}'$;
8         remove annotations (from $\mathbb{R}'$) which are published by annotators who have ever received negative feedback from one or more friends of Alice;
```

## 4.1 Overview

SRaaS can be deployed on any tagging system that holds the properties described in §2. We call the tagging systems equipped with SRaaS as *SRaaS tagging systems*. Specifically, an SRaaS tagging system user (say, Alice) executes a resource discovery process through the following steps:

1. **Search:** Alice issues a tag search $t$ to the SRaaS tagging system;
2. **Ranking:** SRaaS extracts corresponding results and ranks them according to the *SRaaS ranking algorithm* (detailed in §4.2);
3. **Reply:** SRaaS replies Alice with the ranked results for her to consume;
4. **Consume:** With the search results in hand, Alice consumes some of them (*e.g.*, annotating consumed resources with appropriate tags in her opinion);
5. **Latent feedback:** SRaaS generates Alice's latent feedback with respect to the consumed resources (detailed in §4.3);
6. **Reputation computation:** SRaaS updates all the relevant users' reputation scores based upon the *SRaaS reputation algorithm* (detailed in §4.4).

Compared with typical tagging systems (defined in §2.2), three operations are specially designed in our SRaaS tagging system: ranking, latent feedback, and reputation computation. In the rest of this section, we first detail the design of these three operations in §4.2-§4.4, respectively. Then, we demonstrate SRaaS's provable guarantees on its defense capability (§4.5). Finally, we describe SRaaS's practical issues and corresponding solutions (§4.6).

In this section, our discussion will be with respect to a typical honest SRaaS tagging system user (called Alice). We assume that Alice has successfully registered herself in the system and has a private friend list which is obtained through importing her social network (*e.g.*, from Facebook). Only Alice herself and the SRaaS tagging system can access her friend list.

## 4.2 Ranking

**Personalized reputation list.** In an SRaaS tagging system, each user has a personalized *reputation list* which stores the personalized reputation scores of all other users in the system. By personalized, we mean that the reputation list is not identical across different users. Namely, user $A$ and user $B$ could have, in their reputation lists, different reputation scores on user $D$. In other words, one user could be associated with a different reputation score in each of the other users' reputation lists, and her score in one list does not affect that in another user's list. For a given SRaaS tagging system user's reputation list, the initial reputation score of any of the other users is 0. The reputation scores of Alice's friends are $h$ (a predefined threshold $\geq 1$). The intuition behind this design is based on the fact that users look their friends as "more trustworthy participants".

**Ranking algorithm.** After Alice issues a query tag $t$ to the SRaaS tagging system, the system conveys this query to the SRaaS ranking algorithm (Algorithm 1). The algorithm then retrieves matched results from the back-end database, and obtains a set of results $\mathbb{R}$.

- In $\mathbb{R}$, if there is one or more annotations whose reputation scores (defined below) are $\geq h$, the ranking algorithm would produce a new set of results $\mathbb{R}'$ which only contains the annotations whose reputations are $\geq h$, and then orders the results in $\mathbb{R}'$ according to their reputation scores. *The reputation score of an annotation is calculated as the summation of the reputation score of each of the annotators of this annotation.* Finally, the system replies Alice with the set $\mathbb{R}'$.
- In $\mathbb{R}$, if all the annotations' reputations are $< h$, the ranking algorithm generates $\mathbb{R}'$ by randomly ordering all the annotations in $\mathbb{R}$. After that, the ranking algorithm removes annotations (from $\mathbb{R}'$), which are published by annotators who have ever received negative feedback from one or more Alice's friends.

The above approach is called the SRaaS ranking algorithm.

**Intuition.** As shown in Algorithm 1, we say that SRaaS leverages social network to overcome an important practical issue called *cold start* problem [32], which exists in many existing dynamic assessment tag spam defenses [17, 38] and reputation systems [33, 39]. In particular, cold start problem in tagging systems means that a newcomer needs quite a long time to achieve *stable maximum effectiveness* status, since the newcomer needs to accumulate more "experience" by consuming both correct and incorrect annotations. Stable maximum effectiveness means users can always pick correct annotations out of search results. In SRaaS ranking algorithm, because a newcomer can build and import her friend information, the newcomer can quickly pass through the cold start period by getting help from her friends.

## 4.3 Latent Feedback

After Alice picks one or more resources out of $\mathbb{R}'$ and consumes them, she needs to annotate the consumed resources with some corresponding tags. We propose a novel *latent feedback* scheme for users to give SRaaS feedback regarding whether, in their opinions, the consumed resources have been correctly annotated with the tags in the query. The basic idea of the latent feedback scheme is to automatically infer Alice's feedback based on her posted tags rather than asking Alice to provide feedback explicitly.

In particular, after Alice annotates some consumed resource, $r$, with one or more tags (*e.g.*, $t_1, ..., t_k$), SRaaS leverages a well-developed tag semantic similarity checking tool [23] to quantify a latent feedback value $f$ in a fine-grained manner:

$$f = \max(s(t, t_1), s(t, t_2), ..., s(t, t_k)),$$

where $t$ is Alice's query tag, and $s(a, b)$ is the tag semantic similarity between tags $a$ and $b$. The range of $f$ is $[0, 1]$, and a higher value indicates that tags $a$ and $b$ are more similar. Inspired by Markines *et al.* [23, 24], if $f \geq 0.5$, SRaaS considers that Alice provides a positive feedback to the annotation $\langle t, r\rangle$ (*i.e.*, the consumed resource has been correctly annotated with the tag in query); otherwise, SRaaS considers Alice's feedback is negative (*i.e.*, the consumed resource has been incorrectly annotated with the queried tag).

Using the abovementioned tag semantic similarity checker enables our latent feedback scheme to capture the tags that are not

**Algorithm 2:** SRaaS reputation algorithm for any typical SRaaS tagging system user, say Alice. Each typical user starts with 0 reputation score. The parameters satisfy $s = 0.75$, $\alpha > 1$, $h \geq 1$, $\omega = h/\alpha$ and $0 \leq \beta < 1$.

```
1  After Alice's latent feedback on the consumed annotation $\mathbb{A}$:
2  begin
3      if $\mathbb{A}$ is correct (i.e., $0.5 \leq f \leq 1$) then
4          if $\mathbb{A}$'s reputation score is $< h$ or one of $\mathbb{A}$'s annotators is Alice's friend then
5              for $x \leftarrow$ each of the annotators of $\mathbb{A}$ do
6                  if $x$ is not Alice's friend then
7                      if $x$'s reputation is 0 then
8                          set $x$'s reputation score to $\omega/(H+S)$;
9                      else
10                         multiply $x$'s reputation score by $\alpha \cdot f$;
11                 for $y \leftarrow$ each user whose similarity with $x \geq s$ do
12                     if $y$'s reputation is 0 then
13                         set $y$'s reputation score to $\omega/(H+S)$;
14                     else
15                         multiply $y$'s reputation score by $\alpha \cdot f$;
16     if $\mathbb{A}$ is incorrect (i.e., $0 \leq f < 0.5$) then
17         multiply the reputation scores of all annotators for $\mathbb{A}$ by $\beta \cdot f$;
```

exactly the same but semantically similar, thus significantly improving the capability and accuracy of the scheme. Furthermore, the quantified feedback value (*i.e.*, $f$) offers a fine-grained way (rather than binary feedback) to affect reputation computation (see §4.4).

Latent feedback is one of the most important designs of SRaaS. The scheme not only enables SRaaS to obtain feedback from users, but also avoids additional feedback operations provided by users.

## 4.4 Reputation Computation

Based on the latent feedback from Alice, SRaaS reputation algorithm updates the reputation scores of all the relevant users in Alice's personalized reputation list. Note that Alice's feedback can only affect her reputation scores on other users. Because each SRaaS user maintains her/his own personalized reputation list, Alice's feedback cannot affect other users' reputation lists.

If the consumed resource is from a correct annotation $\mathbb{A}$ and the reputation score of $\mathbb{A}$ is lower than $h$, the reputation algorithm multiplies the reputation score of each annotator of $\mathbb{A}$ by $\alpha \cdot f$, where $\alpha$ is a constant higher than 1 and $f$ is the quantified latent feedback value (defined in §4.3). Meanwhile, the algorithm multiplies $\alpha \cdot f$ to the reputation score of each user in Alice's reputation list who has *high similarity* (defined in §3.1) with one or more annotators of $\mathbb{A}$. In other words, the algorithm upgrades the reputation scores of the users who are in the same similarity cliques with the annotators of $\mathbb{A}$. Note that in this case, if the annotation $\mathbb{A}$ has any annotators whose reputations are 0, each of such annotators will be given a reputation score of $\omega/(H+S)$, where $\omega$ is a positive constant ($\omega = h/\alpha$). The tagging similarity between two users, $S_{A,B}$, is computed by Equation (1).

If the consumed resource is from an incorrect annotation $\mathbb{A}$, the reputation algorithm multiplies the reputation score of each of the annotators of $\mathbb{A}$ by $\beta \cdot f$, where $\beta$ is a constant ($0 \leq \beta < 1$).

The above scheme is called SRaaS reputation algorithm (shown in Algorithm 2). The design of SRaaS's reputation algorithm obtains strong defense capability due to the following two reasons.

- Honest users in implicit similarity cliques can earn reputations very fast. Because a fraction $\xi$ of correct annotations are contributed by implicit similarity cliques, it is $\xi$ probability for our algorithm to upgrade the reputation scores of users in one or more similarity cliques, when Alice consumes a correct annotation. Thus, after a few rounds of resource discovery processes, users in implicit similarity cliques would obtain reputation scores much higher than other users, thereby significantly diminishing the impact of malicious users.
- Another important design is: if Alice consumes the resource from a correct annotation whose reputation is higher than $h$, no annotator would increase reputation score. This design avoids that adversaries may try to get "free" reputation scores through publishing many "unhelpful" but correct annotations.

**How to determine the values of $\alpha$ and $\beta$?** In §4.5, we will prove that setting $\alpha$ and $\beta$ to different values does not essentially affect the defense capability of SRaaS; nevertheless, these two parameters can affect the *convergence* of SRaaS users. Convergence means for how long an SRaaS user can always pick correct annotations out from search results and consume the resources of her interest. In other words, convergence determines how fast an SRaaS user can achieve a "stably good experience". To this end, we propose an adaptive approach (inspired by empirical studies [11, 37, 39]) that automatically sets the values of $\alpha$ and $\beta$ by estimating the proportions of honest and malicious users in the system[3]:

- When honest users are more than malicious users in the tagging system (*i.e.*, $H > S$), setting $\alpha > 5$ and $\beta = 1/\alpha$ would give SRaaS a better convergence [11, 37]. Our experiments in §5.4 indicate that setting $\alpha = 10$ and $\beta = 0.1$ generates the best convergence for our collected real-world dataset.
- When malicious users are more than honest users in the tagging system (*i.e.*, $S > H$), setting $\alpha < 5$ and $\beta = 1/\alpha$ would give SRaaS a better convergence [39]. Our experiments in §5.4 indicate that setting $\alpha = 2$ and $\beta = 0.5$ generates the best convergence for our collected real-world dataset.
- If users already have a well-maintained friend list, the values of $\alpha$ and $\beta$ almost do not affect the convergence of SRaaS, due to the help from friends (see §5.4).
- When such a proportion is hard to estimate in certain cases, according to our experiences we suggest to set $\alpha = 5$ and $\beta = 0.2$ for a usually moderate convergence (see §5.4).

**How to determine the value of $h$?** In SRaaS, no user can have a reputation score higher than $\alpha \cdot h$ (refer to Algorithm 2), so we could compute $h$ by $h =$ the maximum reputation score$/\alpha$. Because the maximum reputation score of a tagging system is known in practice, it is straightforward to figure out $h$.

## 4.5 Provable Guarantees

We now provide provable guarantees on SRaaS's defense capability. Before that, we first describe a key metric.

**Loss.** We define *loss* as the total number of unwanted resources consumed throughout a given user's lifespan. We can consider loss as the "the opposite of goodness" of both traditional tagging systems and SRaaS tagging systems, and we are only concerned with the expectation on their losses. We think a truly spam-resistant tagging system should achieve a rather small loss per result ($= \frac{\text{loss}}{\text{the \# of consume}}$) that is much smaller than $1 - \gamma$, where $\gamma$ is the fraction of correct annotations in the system. It is very difficult for regular tagging systems, *i.e.*, without SRaaS, to achieve this goal [14, 15, 17]. On

[3]Existing efforts [10, 34] have proposed practical approaches to estimate the proportions of honest and malicious users in social network-based systems.

the contrary, SRaaS is capable of assisting its users to achieve this target based upon their "tagging behaviors".

**Guarantees.** We now prove a series of important guarantees offered by SRaaS, thus demonstrating that an SRaaS tagging system can bound the loss of each of its users within a constant.

LEMMA 1. *In a given SRaaS tagging system, there are $H$ honest users and $S$ spam attackers. For some honest user u, we define $\Delta_u$ to be the number of correct annotations whose reputations are lower than h and that are consumed by this honest user u. For any given $M$, $\alpha$, $\omega$, $f$, and h, and regardless of the strategies of spam attackers, we have:*

$$\Delta_u \leq M\lceil \log_{0.5\cdot\alpha}(f\cdot\alpha\cdot h\cdot(H+S)/\omega)\rceil \tag{2}$$

PROOF. For an honest user $u$, assume the reputation scores of all the resources in her search results are lower than $h$, if she consumes the resources published by one or more members in similarity cliques, SRaaS would increase the reputation scores of users involved in at least one similarity clique. Note that no user can have a reputation score higher than $f\cdot\alpha\cdot h$. Otherwise, this user will have a reputation score higher than $h$ before the last reputation increasement, which violates our definitions. Because every user starts with a reputation $\omega/(H+S)$ and the lowest value of $f$ is 0.5 in a positive feedback (based on Algorithm 2), the reputation score of a similarity clique member can be multiplied by $0.5\cdot\alpha$ for at most $\lceil\log_{0.5\cdot\alpha}(f\cdot\alpha\cdot h\cdot(H+S)/\omega)\rceil$ times before his reputation score reaches $f\cdot\alpha\cdot h$. Therefore, the user $u$ can consume at most $M\lceil\log_{0.5\cdot\alpha}(f\cdot\alpha\cdot h\cdot(H+S)/\omega)\rceil$ resources published by similarity cliques when there is no resource whose reputation score $\geq h$. We get $\Delta_u \leq M\lceil\log_{0.5\cdot\alpha}(f\cdot\alpha\cdot h\cdot(H+S)/\omega)\rceil$. □

**Observation from Lemma 1.** Lemma 1 indicates that with a small $M$, $\Delta_u$ will be small as well no matter what types and what scales of the spam attacks are. In addition, the resource published by similarity cliques and consumed by the honest user is a random resource from at least $\gamma\xi\cdot Y$[4] annotations whose reputations are lower than $h$, since in the $\Delta_u$ case, all the annotations appeared in the current round cannot have reputations higher than $h$ (see §4.2). Given this fact, we can look $\Delta_u$ as the number of successful selections when repeating an experiment of at least $\gamma\cdot\xi$ success probability for $P_s+G_s$ times. Here, $P_s$ denotes the number of incorrect annotations whose reputations are lower than $h$ and that are consumed by an honest user. $G_s$ means the number of correct annotations whose reputation scores are lower than $h$ and that are consumed by the honest user. According to the geometric distribution, we have $E[P_s+G_s] \leq \frac{1}{\gamma\cdot\xi}\Delta_u$, and $E[G_s] \leq \frac{1}{\xi}\Delta_u$.

LEMMA 2. *Consider some honest user u and any given $\alpha$, $\beta$, $\omega$ and h. Let $P_f$ be the number of incorrect annotations whose reputation scores are $\geq$ h and that are consumed by this honest user u. Let $G_s$ be the number of correct annotations whose reputation scores are lower than h and that are consumed by the user u. Then, regardless of the strategies of spam attackers, the relationship between $P_f$ and $G_s$ is:*

$$(h - h\cdot 0.5\cdot\beta)\cdot P_f \leq (\frac{\omega}{H+S} + h\cdot f\cdot\alpha - h)\cdot G_s \tag{3}$$

PROOF. Spam attackers can only increase their reputation scores by publishing correct annotations. Thus, each such annotation enables these spam attackers to obtain less than $h\cdot f\cdot\alpha - h$ additional reputation scores. On the other hand, whenever the user $u$ consumes the resource of an incorrect annotation whose reputation score $\geq h$, the reputation scores of all the annotators for this resource will be multiplied by $f\cdot\beta$. Therefore, SRaaS's algorithm confiscates at least $h - h\cdot 0.5\cdot\beta$ reputations from spam attackers. Because the total confiscated reputations will never be higher than the reputations which the spam attackers can possibly obtain, we have $(h-h\cdot 0.5\cdot\beta)\cdot P_f \leq (\frac{\omega}{H+S} + h\cdot f\cdot\alpha - h)\cdot G_s$. □

THEOREM 1. Let $|\mathbb{L}|$ be the loss of a given honest SRaaS user $u$ and $S$ be the total number of spam attackers. We already have $\Delta_u$ – the number of correct annotations whose reputations are lower than $h$ and that are consumed by the user $u$. Then, regardless of the strategies of spam attackers, $|\mathbb{L}|$ is:

$$|\mathbb{L}| \leq \Delta_u\cdot\frac{1}{\gamma\cdot\xi}\cdot(1-\gamma+\frac{\gamma\cdot(f\cdot\alpha-1+\omega/(H+S))}{1-f\cdot\beta}) \tag{4}$$

PROOF. Lemma 1 shows $\Delta_u \leq M\lceil\log_{0.5\cdot\alpha}(f\cdot\alpha\cdot h\cdot\frac{H+S}{\omega})\rceil$, $E[G_s] = \frac{1}{\xi}\Delta_u$ and $E[P_s+G_s] \leq \frac{1}{\gamma\cdot\xi}\Delta_u$. Through applying Lemma 2, we have $h\cdot(1-0.5\cdot\beta)\cdot P_f \leq (\frac{\omega}{H+S}+h\cdot f\cdot\alpha-h)\cdot G_s$. Because $|\mathbb{L}| = P_f + P_s$, by solving the above equations, we can yield the desired result: $|\mathbb{L}| \leq \Delta_u\cdot\frac{1}{\gamma\cdot\xi}\cdot(1-\gamma+(\gamma\cdot(\alpha\cdot f-1+\frac{\omega}{H+S})/(1-\beta\cdot f)))$. □

We use $\alpha = 5$, $\beta = 0.2$ and $\omega = h/\alpha = 1/5 = 0.2$ to achieve the guarantees of SRaaS. Because of Lemma 1, we know $\Delta_u \leq M\lceil\log_{0.5\cdot\alpha}(f\cdot\alpha\cdot h\cdot(H+S)/\omega)\rceil$, where $M$ is the number of similarity cliques. Thus, if set $\alpha = 5$, $\beta = 0.2$ and $\omega = 0.2$, we have $|\mathbb{L}| \leq \frac{1+2\gamma}{\gamma\cdot\xi}\cdot M$. Because the constants $\gamma$ and $\xi$ are not 0, $|\mathbb{L}|$ becomes $O(M)$. In addition, we conduct experiments to demonstrate the effectiveness of SRaaS under different parameter assignments (see §5).

Based on the above proof, we note that an SRaaS tagging system can bound each user's loss within $O(M)$ no matter how many times (even up to infinite) a user executes tag searches, and regardless of attack forms. For $M$, our measurement results based on large scale public datasets have indicated that $M$, the number of similarity cliques, tends to be relatively small in practice (§3.2). Thus, we conclude that our approach, SRaaS, is capable of strongly bounding any SRaaS tagging system user's loss in practice.

## 4.6 Practical Issues and Solutions

This section discusses a few practical issues of SRaaS, as well as the corresponding solutions.

**Dealing with the systems with high $M$.** While our measurement results have revealed $M$ – the minimal number of similarity cliques that can cover a significant fraction of correct annotations – is small in representative tagging systems, it is possible that some other tagging systems do not hold this property in practice. Such problem would lead to the loss boundary becomes relatively high. We have evaluated this case in §5.5, and our results show SRaaS is not affected too much.

**Malicious friends.** Although SRaaS users benefit a lot from their social networks, one problem is *how to face malicious friends*? While existing solutions, *e.g.*, SumUp [32], and FaceTrust [30], are able to resist malicious friends, they may introduce much additional overhead and implementation complexity in practice. One potential practical solution is to use the algorithm proposed by SocialFilter [31], which can detect malicious friends' behaviors by comparing similarities among friends of a certain user.

[4]$Y$ denotes the number of annotations that appear in the current round.

## 5. EVALUATION

While we have proved that SRaaS can bound each (honest) user's loss within $O(M)$, there are four important questions to answer.

1. How is the additional overhead introduced by SRaaS (§5.2)?
2. How is the defense capability comparison between SRaaS and existing prevalent approaches, such as dynamic assessment and static detection schemes (§5.3)?
3. What is the impact of different values of $\alpha$ and $\beta$ on the convergence of SRaaS (§5.4)?
4. How is the defense capability of SRaaS if tag spam attackers disguise themselves as honest users in $M$ (§5.5)? In other words, we aim to evaluate high-$M$ attacks where malicious users try to achieve the largest loss of honest users.

In this section, we conduct experiments on a well-known dataset (detailed below) to answer the above questions respectively.

### 5.1 Experimental Setup

In order to evaluate SRaaS, we developed a prototype tagging system with all mechanisms of SRaaS in Java. We deployed this SRaaS tagging system on a workstation with Intel Xeon Quad Core HT 3.7 GHz CPU and 16 GB of RAM. Moreover, we ran a MySQL 5.1.54 server on this workstation as the back-end of our prototype.

**Dataset.** We choose to use a well-known public dataset [5] to evaluate our prototype system. This dataset is released by BibSonomy [1] as a part of the ECML PKDD discovery challenge on tag spam in social bookmarking systems [5]. One of the most important goals of this public dataset is to specifically provide resources for tag spam related research. This dataset consists of about 32,000 users who have been manually labeled either as honest users (10,000) or spam attackers (22,000). There are 2,461,957 resources and 14,074,956 annotations in this dataset. In 14,074,956 annotations, there are about 13,000,000 incorrect or misleading annotations.

**Users' behaviors.** Throughout our experiments, all the users' tagging behaviors (both honest and malicious users) follow our user model defined in §2.3. Namely, during the process of our experiments, both honest users and spam attackers participate in the system to search, consume and annotate consumed resources. Even if an honest user consumes an unwanted resource, she annotates this resource with correct tags. On the contrary, a spam attacker annotates consumed resources with misleading tags. Spam attackers may also annotate correct tags based on different attack strategies (defined in §5.3.3). Throughout our experiments, users may leave and rejoin the system randomly.

**Social network setting.** We generate the social network for our experiments according to the small world property of online social networks [25], and establish the friend-relationships for users based on widely adopted Kleinberg model [16]. In particular, we follow the Zipf distribution with its parameter $\alpha = 1$ (specified in small world property [25]) to build friend links between users. The process of building friend links goes from the users who have the most friends to the users who have the fewest friends. Because there are 32,000 users in our system, the users who have the most friends maintain 25,000 friend links according to our specified distribution, and the users who have the fewest friends hold 4 friends. The average node degree in this generated graph is 24 according to the social network measurement results conducted by Alan *et al.* [25].

### 5.2 Evaluating System Overhead

This section answers the first question: how is the additional back-end overhead introduced by SRaaS?

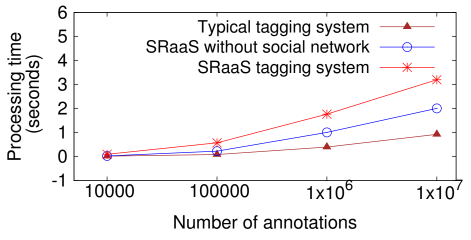


**Figure 5: Overhead on tag search and reply operations**

Our main purpose is to measure SRaaS's query processing time (seconds) between the system (typical or SRaaS tagging system) receives a query tag from a given user and the system returns ranked results to the user. Namely, how long do the tag search and reply operations need. This part of running time is the most important factor for tagging systems, because this would directly affect how long do users need to spend on waiting for tag search results.

We conduct the experiments on different scales of sub-datasets which contain $x$ annotations. We vary $x$ between $10,000$ and $10,000,000$ to cover a wide range of real-world settings. Figure 5 shows the running time of executing a resource discovery process in different scales of environments. We observe that SRaaS and SRaaS without social network do not introduce too much overhead to users in tagging systems. SRaaS without social network means each user does not have any friend and follow SRaaS's algorithms as a pure system newcomer. We observe that the additional overhead introduced by social network is totally acceptable in practice. In addition, SRaaS's space complexity is quadratic, $O(n^2)$, with respect to the number of users (*i.e.*, $n$) in the tagging system.

### 5.3 Evaluating Representative Attacks

To answer the second evaluation question, this section compares SRaaS with three prevalent dynamic assessment approaches and a static detection approach under three different tag spam attacks.

**Experimental execution.** Each of the experiments is composed of 50 *experimental cycles*. The behaviors of honest users follow the features and distributions revealed by Golder *et al.* [13]. In particular, in each experimental cycle, every (honest or malicious) user randomly launches 0-10 tag searches, and then consumes resources according to their rankings in the search results. Thus, if a resource is ranked higher in the search result, it is more probable that this resource will be consumed. We specify that each honest user annotates each of her consumed resources with 1-50 correct tags, and the distribution of the number of tags annotated by honest users follows the classical power-law distribution revealed by Golder *et al.* [13]. This means that a small fraction of honest users annotate each resource with many tags (*e.g.*, 30-50) and, on the contrary, a large fraction of honest users only annotate several tags (*e.g.*, 1-5). In addition, our system always generates latent feedback for every consumed resource.

We specify the number of misleading tags each malicious user annotates per experimental cycle in §5.3.4 and §5.3.5, since this parameter depends on attacking levels of malicious users. In each experimental cycle, there are 1,000 new resources to be added into the environment. These new resources are assigned to a few users randomly, and then tagged by those users. After each cycle, the number of spam search results is recorded.

In the following, we first present a metric used to evaluate the capability of spam-resistant efforts (§5.3.1). Then, we describe the four efforts to be compared (§5.3.2) and three tag spam attacks (§5.3.3). Finally, we show our experimental results (§5.3.4 and §5.3.5).

### 5.3.1 Metric: SpamFactor

In the rest of our evaluation, we use a widely accepted metric, called SpamFactor [14, 17], to quantify the "spam impact" in tag search results. The main reason we intend to use SpamFactor as our evaluation metric is that SpamFactor is affected by not only the number of unwanted resources (*e.g.*, the loss defined in §4.5) but also the positions of misleading annotations in the search results. In other words, from the experimental perspective, SpamFactor is more comprehensive metric than the "loss" metric.

SpamFactor metric is proposed by Koutrika *et al.* [17]. In order to measure the impact of tag spam, a $SpamFactor(t)$ is defined as follows. Given a query tag $t$, the tagging system returns a ranked result $R_K$ containing $K$ items, *i.e.*, $R_K = [r_1, r_2, ..., r_K]$, where $rank(r_{i-1}, t) \geq rank(r_i, t), 2 \leq i \leq K$. Then, $SpamFactor(t)$ for the query tag $t$ is computed by the formula: $SpamFactor(t) = (\sum_{d_i \in D_K} w(d_i) \cdot \frac{1}{i})/H_K$, where $w(d_i) = 1$ if $d_i$ is a misleading annotation or $w(d_i) = 0$ if $d_i$ is a correct annotation. $H_K$ is the $K^{th}$ harmonic number, *i.e.*, it is the sum of the reciprocals of the first $K$ natural numbers, *i.e.*, $H_K = \sum_{i \in [1...K]} 1/i$.

From SpamFactor definition, we learn higher SpamFactor represents greater spam in the results. According to Koutrika *et al.* [18], the SpamFactor $< 0.1$ suggests a spam-free tag search result. In our experiments, the SpamFactor focuses on the top 20 search results.

### 5.3.2 Four Compared Schemes

We compare SRaaS with three dynamic assessment schemes, Boolean [7], Occurrence [6], and Coincidence [17], and a static detection scheme, SpamDetector [22]. They not only have been widely adopted in real-world tagging systems, but also cover features of most of spam-resistant strategies.

**Boolean scheme.** Boolean scheme is a simple dynamic assessment scheme used in some popular tagging systems (*e.g.*, Slideshare [7]). The basic idea of Boolean scheme is that the tagging system randomly ranks annotations that match a given query tag $t$.

**Occurrence scheme.** Occurrence scheme (deployed by Facebook, YouTube, and Rawsugar [6]) ranks search results by counting the number of annotations containing the query tag $t$, and returns the top ranking results.

**Coincidence scheme.** Coincidence scheme is designed as a spam-resistant tag search scheme which has been used by some systems (*e.g.*, SpamClean [41] and Coincidence [17]). The Coincidence scheme is the first effort taking user's reputation into account. In particular, user $u_i$ is considered more reliable than user $u_j$ if $u_i$'s annotations more often coincide with other users' annotations compared to $u_j$'s annotations. Namely, $u_i$ is more often in agreement with her peers. Coincidence scheme assigns each user a global trust degree which is the sum of the same annotations between this user and the other users in the system, and then the system orders each search result based on the average of all the annotators' trust degrees of each annotation.

**Static detection scheme: SpamDetector.** SpamDetector [22] is a static detection approach, which is different from the above three schemes. SpamDetector holds six large datasets containing various tag spam, and the detector analyzes them for training purpose. Then, SpamDetector uses learned rules to check whether there exist tag spam, *i.e.*, the annotations violate the learned rules.

### 5.3.3 Three Types of Tag Spam Attacks

There are mainly three categories of representative tag spam attacks in the current tagging systems: normal tag spam attack, collusion attack, and tricky attack.

**Normal attack.** For the adversaries who launch normal tag spam attacks, they randomly select some of the resources in the system, and then annotate these resources with random misleading tags in order to achieve the purpose of misleading normal users. In practice, the normal tag spam attack acts independently, which means these malicious users are "lousy annotators".

**Collusive attack.** In some cases, malicious users (*i.e.*, spam attackers) may launch attacks collusively. Collusive attackers annotate many same resources with the number of the same misleading and popular tags in order to make these resources easy to be searched by the normal users who are seeking those popular tags.

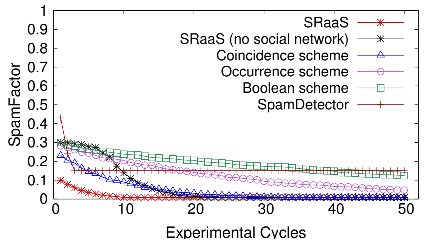
**Tricky attack.** Tricky attackers first pretend themselves by publishing correct annotations in order to earn credits. Then, they annotate specific victim resources with misleading tags. Tricky attacks are powerful strategies that are constructed to compromise existing anti-spam mechanisms (*e.g.*, Coincidence and SpamClean). In particular, because almost all the existing anti-spam efforts update users' credits according to the correctness of their actions, it is easy for tricky attackers to earn many credits from publishing "unhelpful" annotations. In return, these high-credit attackers can "disturb" normal users' search results even if they have already deployed anti-spam techniques.

### 5.3.4 Evaluating Light-Weight Attacks

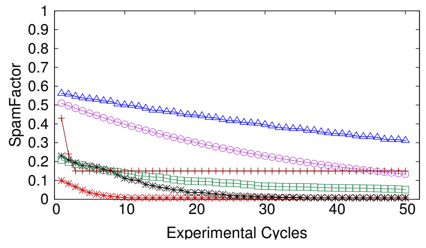
We first evaluate the impact of light-weight attacks on the tagging system. Light-weight attack means each attacker annotates each of her consumed resource with 10-50 misleading tags in each experimental cycle. In the experiments, we compare SRaaS with Boolean, Occurrence, Coincidence and SpamDetector under the three attacks respectively. Figure 6 shows the impact of the three attacks.

**Discussion on normal attack.** As shown in Figure 6a, Boolean and Occurrence schemes are affected significantly by normal attack, because their SpamFactors decrease to below 0.1 after 30 and 50 experimental cycles respectively. On the other hand, Coincidence scheme presents good defense capability to normal attack since it considers not only the annotations that associate a resource to a query tag, but also correlations among the users who have published these annotations. Different from dynamic assessment schemes, we observe SpamDetector offers search results with SpamFactor $\approx 0.15$ throughout the experiment. Because tag spam with a new semantic pattern and constant number (10-50) is added into the evaluated environment in each cycle, SpamDetector, as a static anti-spam technique, handles dynamic tag spam attacks poorly. In addition, we note the SpamFactor of SRaaS (no social network) is high at the beginning of our experiment; however, its SpamFactor declines quickly after the 8th cycle and decreases to below 0.1 in the 12th cycle. The reason is that honest users in the similarity cliques have been "awarded" many reputations after several cycles, and then users can always consume the resources of annotations published by these similarity cliques. As shown in Figure 6a, the SpamFactor of SRaaS is lower than 0.1 at the beginning of the experiment.

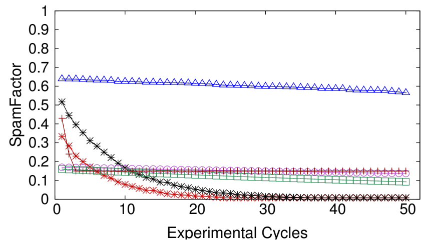
**Discussion on collusive attack.** Figure 6b indicates that the collusive attack can lead to serious influences on both Coincidence and Occurrence schemes. Coincidence scheme works badly because under collusive attacks the reliability degrees of many malicious users become high, which means most incorrect annotations are placed at the top of search results. Similarly, the phenomenon that Occurrence scheme has a high SpamFactor is also based on the same reason. However, why can SRaaS and Boolean work much better than the above two schemes under collusive attack? The reason is SRaaS and Boolean schemes select random resources to consume, so that users choose the annotations attacked by the collusive attackers with



(a) Impact of normal attack.

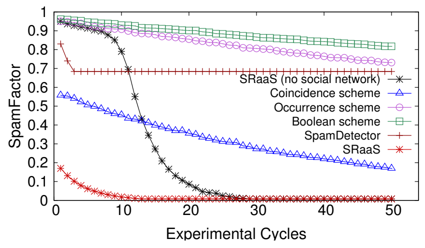


(b) Impact of collusive attack.

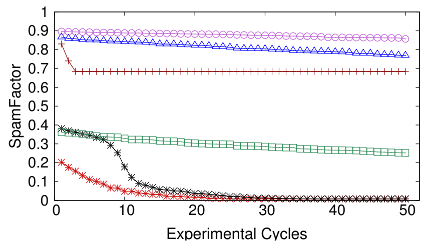


(c) Impact of tricky attack.

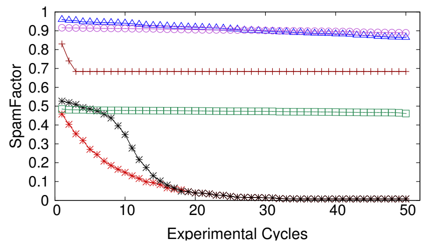
**Figure 6: Impact of three tag spam attacks (light-weight attacks).**



(a) Impact of normal attack.



(b) Impact of collusive attack.



(c) Impact of tricky attack.

**Figure 7: Impact of three tag spam attacks (heavy-weight attacks).**

much lower probability than Coincidence and Occurrence schemes. Moreover, with cycles growing, users establish reputations with honest users in similarity cliques gradually, so that the SpamFactors of SRaaS can keep decreasing all the time. Finally, we observe that SpamDetector performs almost the same as in the normal attack case. As a static detection approach, SpamDetector is not affected by different attacking behaviors, and it only analyzes the semantic patterns of the added tag spam. Because our experiments (normal attack, collusive attack and tricky attack) only change the attacking behaviors rather than the semantic patterns, SpamDetector performs almost the same under the three types of attacks.

**Discussion on tricky attack.** Figure 6c shows the impact of tricky attack. Coincidence scheme performs badly under tricky attack. We find that the normal users of Coincidence scheme assign high reliability degrees to many "like-minded" users who are actually malicious users (tricky attackers); meanwhile, these normal users are "deceived" by the tricky attackers. The reason is that tricky attackers successfully utilize the vulnerability of Coincidence scheme to mount tag spam attacks. Namely, tricky attackers annotate the resources with both correct and misleading tags and then publish them. On the other hand, although Boolean and Occurrence schemes are also impacted by tricky attack, their SpamFactors can be controlled below 0.2 with cycles growing. Influenced by tricky attack, the SpamFactors of SRaaS (with and without social network) are very high (0.5 and 0.3, respectively) at the beginning. However, their SpamFactors converge quickly to below 0.1 in 8 and 15 cycles respectively. This reveals that although tricky attackers can obtain high reputations, our approach can find and punish them as experimental cycles increases.

### 5.3.5 *Evaluating Heavy-Weight Attacks*

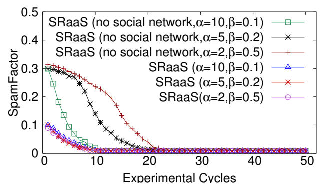
We then evaluate the impact of heavy-weight attacks on the tagging systems. Heavy-weight attack means each attacker annotates each of her consumed resource with 100-500 misleading tags in each experimental cycle. We make similar comparison as what we did for the light-weight attack (in §5.3.4). Figure 7 shows the impact of three attacks at heavy-weight level.

As shown in Figure 7, we find that all of six approaches all deteriorate significantly under heavy-weight attacks. *Why is Coincidence scheme affected so significantly under heavy-weight normal attacks (shown in Figure 7a)?* From the description of Coincidence scheme, we know that Coincidence scheme is an anti-tag spam mechanism and the experiment shown in Figure 6a also demonstrates the scheme is able to defend against normal tag spam attack. However, its bad result under heavy-weight normal attacks reveals that as malicious users and incorrect annotations proliferate, some malicious users may obtain high reliability degrees, so Coincidence scheme deteriorates significantly. As shown in existing efforts [41], Coincidence scheme (*e.g.*, SpamClean) cannot resist too strong attacks (*e.g.*, the # of normal attackers $>$ the # of honest users).

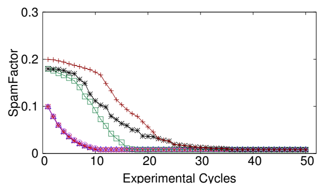
Why are the SpamFactors of Boolean scheme under heavy-weight tricky attacks (shown in Figure 7c) higher than those under light-weight tricky attacks (shown in Figure 6c)? The reason is that as the number of malicious users and incorrect annotations grows, the percentage of incorrect annotations in each search result has been much higher than that of correct ones, so there are many unwanted resources in the search results of Boolean scheme (under heavy-weight tricky attacks).

Under all the three types of heavy-weight attacks, Occurrence performs very badly. Because the key feature of heavy-weight attacks is to create a large amount of misleading tags and Occurrence ranks search results according to the number of annotations, Occurrence suffers from heavy-weight attacks regardless of attack models. On the contrary, although SRaaS suffers from heavy-weight attacks at the beginning of experiments, it quickly assigns and propagates high reputations to honest users in similarity cliques, thus significantly diminishing its SpamFactor.

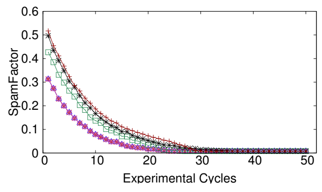
As a static detection approach, SpamDetector bears a relatively high SpamFactor under heavy-weight attacks (but regardless of at-



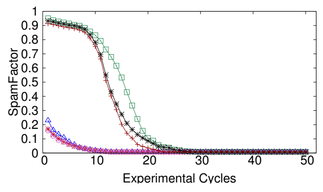
(a) Light-weight normal attack.



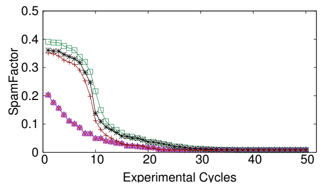
(b) Light-weight collusive attack.



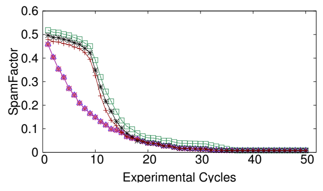
(c) Light-weight tricky attack.



(d) Heavy-weight normal attack.



(e) Heavy-weight collusive attack.



(f) Heavy-weight tricky attack.

**Figure 8: Impact of $\alpha$ and $\beta$ on SRaaS's convergence.**

tacking behaviors), because much more tag spam with new semantic patterns is added into the evaluated environment.

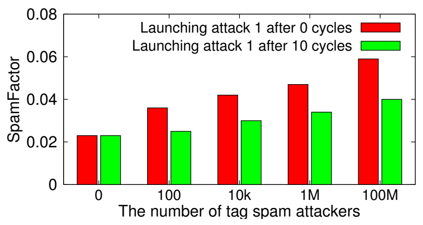
## 5.4 SRaaS's Convergence with Different $\alpha$, $\beta$

We have proved that the values of $\alpha$ and $\beta$ do not affect SRaaS's defense capability in §4.5. However, $\alpha$ and $\beta$ may affect the convergence of SRaaS. Thus, we now evaluate SRaaS's convergence by running it on the previous dataset with different values of $\alpha$ and $\beta$.

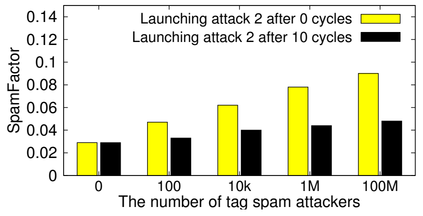
Figure 8 presents SRaaS's convergence under six different types of spam attacks (used in §5.3.4 and §5.3.5). In each type of attack, we execute SRaaS with ($\alpha = 10, \beta = 0.1$), ($\alpha = 5, \beta = 0.2$) and ($\alpha = 2, \beta = 0.5$), respectively, and we have the following findings. First, as shown in Figure 8a, 8b, and 8c, when honest users are more than malicious users in the tagging system (*i.e.*, light-weight attack $H > S$), setting $\alpha = 10$ and $\beta = 0.1$ can give SRaaS a better convergence. Second, as shown in Figure 8d, 8e, and 8f, when malicious users are more than honest users in the tagging system (*i.e.*, heavy-weight attack $S > H$), setting $\alpha = 2$ and $\beta = 0.5$ can give SRaaS a better convergence. Third, if users already have a well-maintained friend list (*i.e.*, social networks), the values of $\alpha$ and $\beta$ almost do not affect the defense capability of SRaaS. Finally, setting $\alpha = 5$ and $\beta = 0.2$ can always obtain a "moderate" (of course sub-optimal) convergence. Such an assignment can be used for the case in which the proportions of honest users and malicious users are hard to estimate.

## 5.5 Evaluating the High-$M$ Attacks

To answer the third question of the experimental goals, *i.e.*, evaluating the defense capability of SRaaS against the case with very high $M$, which means there are many similarity cliques covering a significant fraction of correct annotations in the evaluated environment. In order to construct such "high-$M$" attacks, we need to modify spam attackers' strategies to lead to the largest loss of normal users. While we make no claim that the constructed attacks



(a) Impact of the high-$M$ attack 1.



(b) Impact of the high-$M$ attack 2.

**Figure 9: Impact of the constructed high-$M$ attacks.**

must be the worst-case attacks, we hope to understand how well can SRaaS defend against such strong attacks.

We generate $5,000$ new resources, and partition them into 50 sets with 100 resource per set. We construct one experimental cycle corresponding to each set. Each experimental cycle contains all the 100 resource in the set. We assume that a user $u$ wants to consume 1-20 resources in each experimental cycle.

**Constructing high-$M$ attack 1.** To construct the first worst-case attack, $S$ malicious users enter our system and are split into two equal sets $\mathbb{M}_1$ and $\mathbb{N}_1$. Each user in $\mathbb{M}_1$ publishes correct annotations (the same as honest users in $M$) in that cycle, and each user in $\mathbb{N}_1$ publish the same number of incorrect annotations. In the second cycle, $\mathbb{M}_1$ is split into two equal-size sets $\mathbb{M}_2$ and $\mathbb{N}_2$, and users in $\mathbb{N}_1$ are replaced with $S/2$ fresh attackers. One half of fresh attackers are added to $\mathbb{M}_2$ and the remaining half are added to $\mathbb{N}_2$. Same as before, users in $\mathbb{M}_2$ publish correct annotations, while users in $\mathbb{N}_2$ publish the same incorrect annotations. Such process is repeated for all the remaining experimental cycles.

**Constructing high-$M$ attack 2.** We also construct another type of high-$M$ attack, called high-$M$ attack 2: initially, the adversaries create 100 sets of attackers where each set has $S$ malicious users. The execution process is broken into multiple segments, where each segment consists of $m + n$ experimental cycles. In the first $m$ experimental cycles of each segment, each of the 100 sets publishes a distinct correct annotation in each experimental cycle. In the next $n$ experimental cycles of each segment, each set publishes a distinct incorrect annotation in each experimental cycle. Our experiments show that using $m = 5$ and $n = 1$ incurs the largest loss.

**Discussion.** Figure 9 shows that SRaaS's average SpamFactor during all the cycles under different types of high-$M$ attacks, when the attack starts after experimental cycle 0 and 10, respectively. The number of attackers, *i.e.*, $S$ in the above attack construction descriptions, varies from 0 to 100M (*i.e.*, X-axis in Figure 9).

It is clear that the SpamFactor is below 0.1 throughout the whole experiments, since these malicious attackers – who disguise themselves as honest users in $M$ – fail to obtain any reputation scores (due to the design in Algorithm 2). This demonstrates the defense capability of SRaaS is still good under large-scale high-$M$ attacks. Besides, we note that the robustness of SRaaS becomes stronger when users have used SRaaS for some time (*e.g.*, 10 experimental cycles in our experiment).

## 6. CONCLUSION

This paper presents SRaaS, a novel tag spam-resistant service which is applicable to today's heterogeneous tagging systems. SRaaS is featured by its provable guarantees on the defense capability, thus significantly diminishing the influence of tag spam attacks. While the design of SRaaS is facilitated by techniques from recommendation systems, information retrieval and social networks, SRaaS mainly leverages users' implicit tagging behaviors (*i.e.*, the so-called "implicit similarity cliques") to achieve its strong defense capability. Comprehensive experiments on a realistic dataset confirm the efficacy and efficiency of SRaaS.

## Acknowledgements

We thank the anonymous reviewers for their insightful comments. We also thank Jiewen Huang, Kun Ren, and Avi Silberschatz for their valuable feedback on earlier version of this work. This work was supported in part by the High-Tech Research and Development Program of China ("863 – China Cloud" Major Program) under grant 2015AA01A201, the National Natural Science Foundation of China (NSFC) under grants 61432002 (State Key Program), 61632020 (State Key Program), 61471217, and 61422208, as well as the CCF-Tencent Open Fund under grants IAGR20150101 and RAGR20160105.

## 7. REFERENCES

[1] BibSonomy. http://www.bibsonomy.org.
[2] CiteULike. http://www.citeulike.org/.
[3] CiteULike dataset. http://konect.uni-koblenz.de/networks/citeulike-ut.
[4] Del.icio.us. http://del.icio.us/.
[5] ECML PKDD Discovery Challenge Dataset. http://www.kde.cs.uni-kassel.de/ws/rsdc08.
[6] Rawsugar. http://rawsugar.com/.
[7] Slideshare. http://slideshare.net/.
[8] B. Awerbuch and T. P. Hayes. Online collaborative filtering with nearly optimal dynamic regret. In *SPAA*, 2007.
[9] T. Bogers and A. van den Bosch. Using language models for spam detection in social bookmarking systems. In *ECML PKDD*, 2008.
[10] Q. Cao, X. Yang, and C. Palow. Uncovering large groups of active malicious accounts in online social networks. In *CCS*, 2014.
[11] C. P. Costa and J. M. Almeida. Reputation Systems for Fighting Pollution in Peer-to-Peer File Sharing Systems. In *Peer-to-Peer Computing*, 2007.
[12] A. Gkanogiannis and T. Kalamboukis. A novel supervised learning algorithm and its use for spam detection in social bookmarking systems. In *ECML PKDD*, 2008.
[13] S. A. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *J. Information Science*, 32(2):198–208, 2006.
[14] P. Heymann, G. Koutrika, and H. Garcia-Molina. Fighting spam on social web sites: A survey of approaches and future challenges. *IEEE Internet Computing*, 11(6):36–45, 2007.
[15] I. Ivanov, P. Vajda, J. Lee, and T. Ebrahimi. In tags we trust: Trust modeling in social tagging of multimedia content. *IEEE Signal Process. Mag.*, 29(2):98–107, 2012.
[16] J. M. Kleinberg. The small-world phenomenon: An algorithm perspective. In *STOC*, 2000.
[17] G. Koutrika, F. A. Effendi, Z. Gyöngyi, P. Heymann, and H. Garcia-Molina. Combating spam in tagging systems. In *AIRWeb*, 2007.
[18] G. Koutrika, F. A. Effendi, Z. Gyöngyi, P. Heymann, and H. Garcia-Molina. Combating spam in tagging systems. Technical Report Technical report, available at http://dbpubs.stanford.edu/pub/2007-11, November 2007.
[19] B. Krause, C. Schmitz, A. Hotho, and G. Stumme. The anti-social tagger: Detecting spam in social bookmarking systems. In *AIRWeb*, pages 61–68, 2008.
[20] A. Kyriakopoulou and T. Kalamboukis. Combining clustering with classification for spam detection in social bookmarking systems. In *ECML PKDD*, 2008.
[21] B. Liu, E. Zhai, H. Sun, Y. Chen, and Z. Chen. Filtering spam in social tagging system with dynamic behavior analysis. In *ASONAM*, pages 95–100, 2009.
[22] B. Markines, C. Cattuto, and F. Menczer. Social spam detection. In *AIRWeb*, 2009.
[23] B. Markines, C. Cattuto, F. Menczer, D. Benz, A. Hotho, and G. Stumme. Evaluating similarity measures for emergent semantics of social tagging. In *WWW*, 2009.
[24] B. Markines and F. Menczer. A scalable, collaborative similarity measure for social annotation systems. In *HYPERTEXT*, 2009.
[25] A. Mislove, M. Marcon, P. K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Internet Measurement Comference*, 2007.
[26] A. Narayanan. Del.icio.us dataset. http://randomwalker.info/data/delicious/delicious-rss-1250k.gz.
[27] P. Resnick and R. Sami. The influence limiter: Provably manipulation-resistant recommender systems. In *ACM RecSys*, 2007.
[28] P. Resnick and R. Sami. The information cost of manipulation-resistance in recommender systems. In *ACM RecSys*, 2008.
[29] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.
[30] M. Sirivianos, K. Kim, and X. Yang. FaceTrust: Assessing the credibility of online personas via social networks. In *HotSec*, 2009.
[31] M. Sirivianos, X. Yang, and K. Kim. Socialfilter: Collaborative spam mitigation using social networks. *CoRR*, abs/0908.3930, 2009.
[32] N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-resilient online content voting. In *NSDI*, 2009.
[33] K. Walsh and E. G. Sirer. Experience with an object reputation system for peer-to-peer filesharing. In *NSDI*, 2006.
[34] G. Wang, T. Wang, H. Zheng, and B. Y. Zhao. Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers. In *USENIX Security*, 2014.
[35] Y. Wang, S. Yao, J. Li, Z. Xia, H. Yan, and J. Xu. ReSpam: A novel reputation based mechanism of defending against tag spam in social computing. In *8th IEEE SOSE*, 2014.
[36] C. Wei, Y. Liu, M. Zhang, S. Ma, L. Ru, and K. Zhang. Fighting against Web spam: A novel propagation method based on click-through data. In *SIGIR*, 2012.
[37] L. Xiong and L. Liu. PeerTrust: Supporting reputation-based trust for Peer-to-Peer electronic communities. *IEEE Trans. Knowl. Data Eng.*, 16(7):843–857, 2004.
[38] Z. Xu, Y. Fu, J. Mao, and D. Su. Towards the semantic web: Collaborative tag suggestions. In *Collaborative Web Tagging Workshop in conjunction with WWW*, 2006.
[39] H. Yu, C. Shi, M. Kaminsky, P. B. Gibbons, and F. Xiao. DSybil: Optimal sybil-resistance for recommendation systems. In *IEEE Symposium on Security and Privacy*, pages 283–298, 2009.
[40] E. Zhai, L. Ding, and S. Qing. Towards a reliable spam-proof tagging system. In *SSIRI*, pages 174–181, 2011.
[41] E. Zhai, H. Sun, S. Qing, and Z. Chen. SpamClean: Towards spam-free tagging systems. In *CSE (4)*, 2009.