A Practical Spam-Resistant Service for Tagging Systems

Ennan Zhai
Yale University

Abstract

Tagging systems are particularly vulnerable to tag spam. However, defending against tag spam has been known to be extremely challenging in practice, since adversaries can easily launch spam attacks in various ways and scales. Driven by existing measurement results on the similarities of tagging behaviors among honest users, this paper proposes a practical approach, Spam-Resistant-as-a-Service (or SRaaS), to defend against spam attacks. SRaaS employs a rank-based tag search scheme whose design combines reputation and social network techniques. SRaaS not only is adoptive to any typical tagging systems, but also can significantly diminish the impact of tag spam attacks even with arbitrary attack strategies. We provide provable guarantees on the defense capability of SRaaS. Through running our SRaaS prototype on realistic tagging system datasets (Del.icio.us), we demonstrate that SRaaS is practical and outperforms the existing tag spam defense techniques.

1 Introduction

Today's tagging systems—employed by, for example, Flickr, Del.icio.us, YouTube, and CiteULike—have become popular services that enable users to conveniently find the resources of their interest based on the tags posted by other participants. In general, each specific *resource* in a tagging system (e.g., a video on YouTube) is annotated with multiple *tags*. When a user issues a tag query (or called 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, unfortunately, indicate that tagging systems in practice are particularly vulnerable to *tag spam*. In the launch of a typical tag spam attack, malicious participants (called spam attackers), with an intention to mislead normal users, generate and annotate a particular target resource with numerous erroneous and irrelevant tags. The huge amount of tag spam attacks in existence would make normal users without sufficient "experience" consume unwanted resources very frequently, thus adversely affecting the availability and usability of tagging systems [12, 18, 41].

1.1 Motivating Our Work

Defending against tag spam has been known to be extremely challenging in practice, since adversaries can easily take advantage of tagging systems' nature—allowing users to create any number of tags with any personal choice of keywords or terms—to launch spam attacks in various ways and in any scale. Heymann et al. [12] have grouped existing defense mechanisms into three categories: *detection-based*, *prevention-based*, and *rank-based* approaches.

The detection-based approaches [7,9,16,20,21,23,27] typically leverage statistical analysis and machine learning techniques to detect, identify and remove tag spam. These efforts are usually *ad hoc* and limited to particular system environments and attack strategies, however. Furthermore, because most of the detection-based techniques are relatively time-consuming and have significant overheads, it is difficult to deploy them in practice [20].

The prevention-based approaches attempt to restrict the spam attacker's action of posting a large number of misleading tags via the use of challenge-response puzzles (e.g., CAPTCHA [34]). Nevertheless, these approaches cannot tackle the spam problem fundamentally, since attackers can use puzzle recognition techniques such as DeCaptcher [1] to solve these restrictions. Thus, this type of mechanisms is usually used as a complement to either detection-based or rank-based approaches.

The rank-based approaches [18,25,37,41] aim at prioritizing the most relevant tag search results and degrading the rank of unmatched or misleading resources to the end of result list according to the accumulation of user experience. The purpose of the rank-based solutions is to diminish the influence of tag spam on the search results, thus rendering the spam attack ineffective. The rank-based defense has two main advantages over the former two types of solutions. First, it is applicable to more general scenarios as the mechanism does not rely on specific datasets or environments. Second, it has higher potential in the defense against attacks in various ways and scales [38], since it does not require direct identification and removal of the numerous individual misleading tags, but instead gradually reducing its impact through ongoing result ranking and upgrading. As a result, rank-based solutions are considered more practical and effective in resisting massive tag spam [18, 41]. Nevertheless, exist-

ing rank-based efforts only explore heuristics whose defense capabilities are evaluated based on the attack strategies observable from measurement trace rather than any provable quantifiable guarantees. In other words, it is never clear "how well" these solutions are, and whether they can defend against arbitrary forms of attacks.

Thus, our goal is to propose a rank-based scheme with provable guarantees on its defense capability; meanwhile, we should ensure the proposed scheme is practical in typical tagging system scenarios.

An existing sybil-resistant recommendation algorithm, called DSybil [38], and a spam-proof search model, called SpamClean [41], have inspired a potential solution in achieving our goal. DSybil is a reputation-based theoretical effort which provides strong provable guarantees against sybil attacks in recommendation systems. Its design is mainly based upon the heavy-tail distribution of voting behavior of honest identities in recommendation systems, which does not hold in tagging system scenarios, unfortunately. Moreover, DSybil algorithm has a cold start problem [33]—users benefit from the algorithm after quite a long time (e.g., higher than 30 recommendation rounds)—which makes the approach impractical. SpamClean is the first effort in utilizing social network to improve the capabilities of rank-based tag spam defenses. However, as discussed above, SpamClean can only defend against known attack strategies. In other words, no evidence indicates that SpamClean can work well under arbitrary forms of attacks. In addition, SpamClean has been demonstrated to be ineffective against large scale attacks (e.g., the percentage of attackers higher than 30%) [41]. While the above two efforts cannot be directly used to address our target problem, their intuitions (reputation and social network) have inspired our design.

1.2 Our Approach: SRaaS

This paper presents an approach called Spam-Resistant-as-a-Service (or SRaaS). To the best of our knowledge, SRaaS is the first practical effort that can sufficiently diminish the influence of large-scale tag spam attacks with provable guarantees. At the heart of SRaaS is a socially enhanced personalized reputation mechanism whose design is mainly driven by existing measurement results: *the power law distribution of similarity of normal users' tagging behaviors* [10, 11, 13, 14, 24, 37]. Suppose we use an annotation, $\langle t, r \rangle$, to denote the tagging relation between tag t and resource r, this measurement result reveals that M groups of users in a typical tagging system can publish a significant fraction of correct annotations, and the users in each of the groups have a number of overlapping annotations.

In order to demonstrate the defense capability of SRaaS, we prove that SRaaS can bound each user's loss (i.e., the total number of unwanted consumed resources) within $O(M)$ no matter what types of spam attack strategies are.

In order to evaluate the usability and defense capability of SRaaS in practice, we implement a tagging system prototype which is equipped with SRaaS, and we generate experimental environment based real-world datasets. First, we measure the overhead of our prototype. Second, we compare SRaaS with three prevalent rank-based tag search schemes: Boolean [4], Occurrence [3] and Coincidence scheme [18] under various tag spam attacks. Finally, we explore the defense capability of SRaaS by constructing a "worst-case" attack, which tries to achieve maximum loss of SRaaS users. The experimental results indicate that SRaaS is practical, outperforms the other schemes, and offers strong resistance against any form of spam attacks.

2 System Model

SRaaS is a spam-resistant service which is adoptable to any typical tagging system (e.g., Flickr and Del.icio.us), which satisfies certain properties. This section describes a tagging system model holding these properties, and explains important terminologies used throughout this paper.

2.1 System Components & Behaviors

Users. In a typical tagging system, users' purposes are to find resources of their interest rather than exhaustively seeking all the related resources. Users normally have relatively long lifetime (e.g., more than two weeks). Users can be either honest users (i.e., normal users) or spam attackers. Section 2.3 defines detailed behaviors of both types of users.

Tagging behaviors. Users in tagging systems can annotate resources (e.g., web pages in Del.icio.us and photos in Flickr) 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 any 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 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 Flickr, if Alice finds an annotation

$\langle T, R \rangle$, where T is the tag "shoes" and R is a photo about a dog, then she may say this annotation is incorrect. In practice, whether a certain annotation is correct or not is subjective since various users have different opinions. We say two annotations are *the same* if and only if both resources and tags of these two annotations are the same; otherwise, we say two annotations are *distinct*.

Social network. Users in tagging systems are allowed to establish their social networks. Namely, each user can create her own friend relationships with other users. In order to offer a fast and reliable way for building social networks, typical tagging systems allow users to import their friend information from other social networking web sites (e.g., Facebook and Twitter). In addition, each of them in tagging systems can also create new friend relationships with users she is interested in.

2.2 Resource Discovery Execution

In order to discover resources of interest, a user (say, Alice) needs to execute a *resource discovery process* containing the following two steps.

Step 1: Tag search. 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 annotation means that the tags of the returned annotations are the same as the query tag t. So far, we say the user 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 ith annotation in the search results, which contains a resource annotated with tag t. For example in Flickr, $\{A_{t(i)}\}_{i=1}^{n}$ may be all of the annotations containing different photos annotated with the query tag t. Figure 1 illustrates a search and reply example.

Step 2: Resource consumption. With the search results in hand, the user Alice can pick one or more resources out of $\{A_{t(i)}\}_{i=1}^{n}$ (i.e., the search results) to *consume*.[1] A consume, for example, could be watching a video in a video-related tagging system (e.g., YouTube) or accessing a web page in a bookmark-related tagging system (e.g., Del.icio.us). In particular, 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 the fraction of correct annotations in $\{A_{t(i)}\}_{i=1}^{n}$ is at least

[1]Here, we borrow the terminology, consume, from recommender system field.

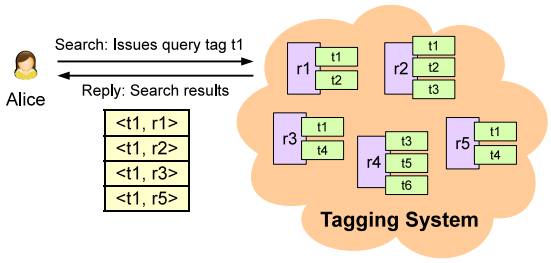


Figure 1: An example execution for a typical user Alice's search and reply, i.e., the first step described in Section 2.2. First, Alice issues a tag search with query tag t_1. Then, the tagging system responds to her by returning search results which contain matched annotations: $\langle t_1, r_1 \rangle$,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

$\gamma > 0$. 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 typical tagging systems, we assume the system provider, e.g., Flickr provider, is honest. In contrast, users can be potential malicious. Namely, a user is either an *honest user* (also called *normal user*) or a *spam attacker* (also called *spammer*).

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. The above assumption on the existence of honest users is necessary and reasonable in practice.

Spam attackers. We assume a spam attacker is intelligent, and any spam attacker may collude with all other ones. We assume spam attackers are able to know which annotation is correct or incorrect, and also which annotation is published by which honest user. For the total number of spam attackers, S, we do not make any assumption, so $S \gg H$ is allowed.

2.4 Important Insight Driving Our Design

We now describe an important measurement result in typical tagging systems [10,11,13,14,24,37]. This result will be used to drive our approach design.

Suppose C is the set of all the correct annotations ever appeared in a tagging system. A fraction ξ of C (e.g., ≥ 0.7) are published by users belonging to a few *similarity groups*. By definition, users within one similarity group have very high *tagging similarities* to each other. Tagging similarity is measured as the ratio of overlapping annotations published by different users (detailed

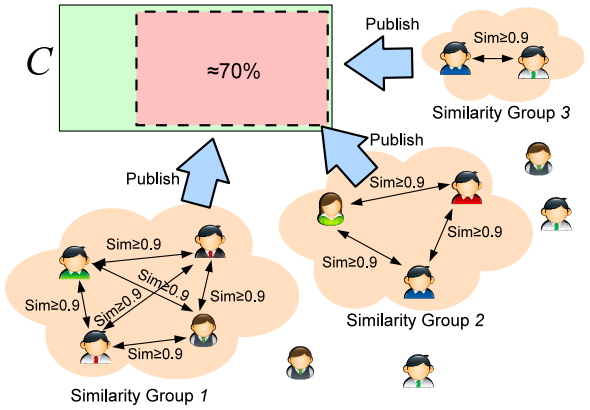


Figure 2: An example for the important insight driving our design. C (green box) is the set of all the correct annotations. There are three similarity groups.

in Equation 1). For example, a tagging similarity ratio of 0.9 or higher could be used as a criteria for assigning users to a similarity group. Note that similarity groups only exist at the logical level. In other words, no party explicitly maintains such groups and nobody knows which users belong to which groups. We define M as the number of similarity groups. Existing measurement efforts [10, 11, 13, 14, 24, 37] reveal that M is relatively small (e.g., ≤ 50) in practice, while the annotations published by the users in all M groups can cover at least 0.7 fraction of C. Figure 2 presents an example for this important insight. There are three similarity groups, and the correct annotations published by these three groups cover about 70% of C, which is the set of all the correct annotations.

3 Basic SRaaS Design

SRaaS can be deployed on any tagging system that holds the properties described in Section 2. We call the tagging systems equipped with SRaaS as *SRaaS tagging systems*. Different from users of existing tagging systems, an SRaaS tagging system user (say, Alice) executes a resource discovery process by the following steps: 1) Alice issues a tag search t to the SRaaS tagging system; 2) The SRaaS tagging system extracts corresponding results and ranks them according to the *SRaaS ranking algorithm* (detailed in Section 3.1); 3) The SRaaS tagging system replies Alice with the ranked results for her to consume; 4) With the search results in hand, Alice consumes some of them; 5) Alice submits feedback to the SRaaS tagging system with respect to the consumed resources (detailed in Section 3.2); 6) Finally, SRaaS updates all the relevant users' reputation scores based upon Alice's feedback (detailed in Section 3.3). Different from

Algorithm 1: Ranking algorithm for an SRaaS user Alice.

```
1 begin
2     if there are one or more annotations whose
      reputation scores ≥ h in search results then
3         produce ℝ′ containing all annotations whose
          reputation scores ≥ h;
4     else
5         produce ℝ′ containing all annotations;
6     randomly order annotations in ℝ′;
```

that a typical tagging systems (defined in Section 2), the resource discovery execution of an SRaaS tagging system has three new operations: ranking, feedback, and reputation computation.

In the following, we first detail the three operations in Sections 3.1-3.3, respectively. Then, we demonstrate SRaaS's provable guarantees on its defense capability (Section 3.4), and analyze the defense capability of SRaaS system in practice (in Section 3.5).

3.1 Ranking

In an SRaaS tagging system, each user has a personalized *reputation list* which stores the personalized reputation scores of all other participants in the system. By personalized, we mean that the reputation list is not identical across different users; i.e., user A and user B could have, in their reputation lists, different reputation scores assigned to participant C. In other words, one user could be associated with a different reputation score in each of the other users' reputation list, and his score in one list does not affect that in another user's list. For a given SRaaS tagging system user, say Alice, in her personalized reputation list, the initial reputation score of any of the other users is 0. After she issues a query tag t to the SRaaS tagging system, the system conveys this query to the ranking algorithm. The algorithm then retrieves matched results from the back-end, and obtains a set of results $\mathbb{R}$. For $\mathbb{R}$, if there is one or more annotations whose reputation scores (defined below) are at least h (a pre-defined threshold), the ranking algorithm would produce a new set of results $\mathbb{R}'$ which only contains the annotations whose reputations are $\geq h$, and then randomly orders the results in $\mathbb{R}'$. The reputation score of an annotation is calculated as the summation of the reputation score of each of the annotators of this annotation. After that, the system replies Alice with the set $\mathbb{R}'$. On the other hand, if there is no annotation has a reputation score equal to or higher than h, the ranking algorithm generates $\mathbb{R}'$ by randomly ordering all the an-

notations in $\mathbb{R}$. The above scheme is called the SRaaS ranking algorithm (shown in Algorithm 1).

Existing theoretical efforts [6, 38] have demonstrated that such random ordering design can effectively prevent the scenario where numerous attackers collude to "overwhelm" the top search results.

3.2 Feedback

After Alice picks one or more resources out of $\mathbb{R}'$ and consumes them, she not only needs to annotates the consumed resources with some corresponding tags, but also needs to give the system *feedback* regarding whether, in her opinion, the consumed resources have been correctly annotated with the tag in query and are the resources she is looking for

In the SRaaS design, feedback is binary, i.e., either [+1 : correct], which means the consumed resource has been correctly annotated with the tag in query or [−1 : incorrect], which means the consumed resource has been incorrectly annotated with this tag. For example, after Alice consumes the resource of $A_{t(3)}$, she thinks that the resource should not be annotated with her query tag t, so she first annotates the resource of $A_{t(3)}$ with the correct tag in her opinion, and then provides the feedback [−1 : incorrect] on $A_{t(3)}$ to the SRaaS tagging system.

3.3 Reputation Computation

Based on the feedback from Alice, SRaaS's reputation algorithm updates the reputation scores of all the relevant users in Alice's personalized reputation list.

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 a constant $\alpha > 1$. The algorithm, in the meantime, multiplies α to the reputation score of each user in Alice's reputation list who has *high similarity* (defined later) with one or more annotators of $\mathbb{A}$ by α. Note that in this case, if the annotation has annotators with 0 reputation, each of such annotators will be given a reputation score of $\omega = h/\alpha$, where ω is a positive constant. The similarity between two users, $Sim_{A,B}$, is computed by Equation (1).

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

Where, R is the set of resources owned by A and B in common and r_j is the jth 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 ith tag of 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 $Sim_{A,B}$ is $[0, 1]$, and a higher value indicates that A and B have more overlapping interests. We define high similarity as $Sim_{A,B} \geq 0.9$. Note that if two users have a similarity score of ≥ 0.9, it means they are in the same similarity group at the logical level (defined in Section 2.4).

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 a constant β ($0 \leq \beta < 1$). Meanwhile, the algorithm multiplies β to the reputation score of each user in Alice's reputation list who has similarity ≥ 0.9 with one or more annotators of $\mathbb{A}$. The above scheme is called SRaaS's reputation algorithm (shown in Algorithm 2).

Algorithm 2: SRaaS's reputation algorithm. Each user starts with 0 reputation score. The parameters satisfy $\omega > 0$, $\alpha > 1$, and $0 \leq \beta < 1$.

```
After Alice's feedback on the given annotation 𝔸:
begin
    if 𝔸 is correct and if 𝔸's reputation score is lower than h then
        for each of the x annotators for 𝔸 with zero reputation (if any) and any user whose similarity with one of the annotators for 𝔸 is higher than 0.9 do
            set his reputation score to ω = h/α;
        for each annotator for 𝔸 with nonzero reputations and any user whose similarity with one of the annotators for 𝔸 is higher than 0.9 do
            multiply his reputation score by α;
    if 𝔸 is incorrect then
        multiply the reputation scores of all annotators for 𝔸 by β;
        multiply the reputation score of each user whose similarity with one of the annotators for 𝔸 is higher than 0.9 by β;
```

3.4 Provable Guarantees

In this section, we will provide provable guarantees on SRaaS tagging systems. Before that, we first describe a metric we want to achieve.

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 γ is the fraction of correct annotations in the system. It is obvious that it is very difficult for existing tagging systems, i.e., without SRaaS, to achieve this goal. In contrast, SRaaS is capable of assisting its users to achieve this based upon their "tagging behaviors".

The following demonstrates the most important provable guarantee offered by SRaaS tagging systems, i.e., Theorem 1. It proves that an SRaaS tagging system is able to bound the loss of each of its users within a constant. Note that SRaaS offers a number of provable guarantees, and we show the complete definitions and the corresponding proofs in Appendix 7.

Theorem 1 Let $|\mathbb{L}|$ be the loss of a given SRaaS user u and S be the total number of spam attackers. We define Δ to be the number of correct annotations that are published by one or more similarity groups and are consumed by the user u. Then, regardless of the strategies of spam attackers, $|\mathbb{L}|$ is:

$$|\mathbb{L}| \leq \Delta \cdot \frac{1}{\gamma \cdot \xi} \cdot (1-\gamma+\frac{\gamma \cdot (\alpha-1+\omega)}{1-\beta}) \qquad (2)$$

Proof sketch: See Appendix 7. □

Corollary 1. We use $\alpha = 2$, $\beta = 0.5$ and $\omega = h/\alpha = 1/2 = 0.5$ to achieve the guarantees of SRaaS. Because of Lemma 1 [40], we know $\Delta \leq M\lceil \log_{\alpha}(\alpha \cdot h \cdot (H+S)/\omega)\rceil$, where M is the number of similarity groups. Thus, if set $\alpha = 2$, $\beta = 0.5$ and $\omega = 0.5$, we have $|\mathbb{L}| \leq \frac{1+2\gamma}{\gamma \cdot \xi} \cdot M$. Because the constants γ and ξ are not 0, $|\mathbb{L}|$ becomes $O(M)$.

We note from the above theorem and corollary 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, the existing studies (e.g., [10, 11, 13, 14, 24]) of large-scale datasets from real-world tagging systems (e.g., Del.icio.us) have indicated that M, the number of similarity groups, tends to be small in practice. Thus, we conclude that our approach can strongly bound any SRaaS user's loss in practice.

3.5 Security Analysis

Although $O(M)$ is an upper bound on each user's loss, it is difficult for adversaries to achieve this "line" in practice.

For a given honest user, Alice, in order to cause the maximum loss, adversaries need to make each user in the same similarity groups receive no higher than ω reputation score; otherwise, most of incorrect annotations will not appear in Alice's search results. Suppose if some of Alice's search results do not include any incorrect annotation, then reputation scores of users who are in some similarity groups would significantly increase by multiplying several α. In other words, once Alice "meets" a few such spam-free search results, it is difficult for adversaries to make Alice consume unwanted resources, since SRaaS can effectively block incorrect annotations. Therefore, the defense capability of SRaaS would become more effective if more honest users can be assigned reputations in Alice's earlier experience. In the next section, we will show how we use social network to achieve this goal.

In order to cause the maximum loss, adversaries may also want to annotate a correct tag to the resource of every correct annotation. In other words, they try to gain reputation scores for free from "unhelpful" tagging efforts and then maximize the influence of their attacks. Unfortunately, if Alice has already consumed some resources of correct annotations before the attack starts, the spam attackers will lose these "opportunities" to gain reputations.

4 Socially-Enhanced SRaaS

Although we have provided provable guarantees on the defense capability of SRaaS, it is still not clear "how fast" can SRaaS achieve *stable maximum effectiveness*. Stable maximum effectiveness means the user can always pick correct annotations out of search results and consume the resources of her interest almost every time. From the basic design of SRaaS, we can know 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. This issue is called cold start problem [33]. Most of the existing reputation systems (e.g., Credence [35] and DSybil [38]) have the cold start problem. Namely, newcomers in those systems are very vulnerable to attacks. Thus, there still leaves one question: *Is it possible to design an approach to enhance the convergence of SRaaS?*

To answer this question, we aim to explore the solution by investigating the characteristics of normal tagging systems, i.e., tagging systems without SRaaS. Previous studies [8, 10, 15, 22, 28, 39] revealed the phenomena that social network enables not only efficient, reliable resource discovery and recommendation with a low additional overhead, but also a significant improvement on the capability of the tagging services. Therefore, based on the consideration about the social nature of tagging systems, we utilize social networks of tagging systems to enhance the convergence of SRaaS. From now on, we

Algorithm 3: Ranking algorithm for a socially-enhanced SRaaS user Alice.

```
1 begin
2     if there are one or more annotations whose reputation scores ≥ h in search results and none of these annotations is published by Alice's friends then
3         produce ℝ′ containing all annotations whose reputation scores ≥ h;
4     else
5         produce ℝ′ containing all annotations;
6         remove annotations published by annotators who have ever received negative feedback from one or more friends of Alice;
7     randomly order annotations in ℝ′;
```

call SRaaS aided by social network as *socially-enhanced SRaaS*.

Each user of a socially-enhanced SRaaS tagging system can establish her own friend relationships with other users by the same methods as existing social networking sites (e.g., Facebook [2] and Technorati [5]). In our design, these friend lists are not public. In other words, only Alice herself and tagging system provider are able to know Alice's friend list. We would show a public friend list may introduce problem later. The intuition of introducing social network to enhance the convergence of the basic SRaaS based on fundamental fact that friends are more reliable than the strange users in the tagging system; therefore, in socially-enhanced SRaaS, a user looks her friends as "more reliable users". Any user Alice can directly set her friends' reputation scores to h after they become Alice's friends. Alice also computes these friends' reputation scores according to socially-enhanced SRaaS's reputation algorithm (i.e., Algorithm 4).

Algorithm 3 and Algorithm 4 present ranking and reputation algorithms of socially-enhanced SRaaS, respectively. In particular, socially-enhanced SRaaS's ranking algorithm has only one difference than the basic one: removing annotations which are published by annotators who have ever received negative feedback from one or more Alice's friends when producing $\mathbb{R}'$. The intuition behind this design is to leverage "experience" of Alice's friends to reduce the number of potentially incorrect annotations in search results.

An important design in the basic SRaaS's reputation algorithm (i.e., Algorithm 2) is: if Alice consumes a correct annotation whose reputation is higher than h, no annotators would increase reputation score. This design avoids that adversaries may try to get "free"

Algorithm 4: Socially-enhanced SRaaS's reputation algorithm. Each user starts with 0 reputation score. The parameters satisfy $\omega > 0$, $\alpha > 1$, and $0 \leq \beta < 1$.

```
1  After Alice's feedback on the consumed annotation 𝔸:
2  begin
3      if 𝔸 is correct then
4          if 𝔸's reputation score is lower than h or one of annotators is Alice's friend then
5              for each of the x annotators for 𝔸 with zero reputation (if any) and any user whose similarity with one of the annotators for 𝔸 is higher than 0.9 do
6                  set his reputation score to ω = h/α;
7              for each annotator for 𝔸 with nonzero reputations and any user whose similarity with one of the annotators for 𝔸 is higher than 0.9 do
8                  multiply his reputation score by α;
9      if 𝔸 is incorrect then
10         multiply the reputation scores of all annotators for 𝔸 by β;
11         multiply the reputation score of each user whose similarity with one of the annotators for 𝔸 is higher than 0.9 by β;
```

reputation scores through publishing many "unhelpful" but correct annotations (discussed in Section 3.5). For socially-enhanced SRaaS's reputation algorithm (i.e., Algorithm 4), if one of annotators of a consumed correct annotation is Alice's friend, each annotators still need to increase their reputation scores. This design aims to avoid honest annotators from being "blocked" by Alice's friends. Because friend lists are private, adversaries cannot obtain reputation scores by following tagging behaviors of Alice's friends.

How to face malicious friends? After introducing social network based enhancement into SRaaS, one obvious problem is *how to face malicious friends*. Unfortunately, the solutions on addressing this "obvious" problem (malicious friends) are not so obvious: 1) because the restriction of SRaaS's system environment, the filtering algorithm proposed in [32] and other similarity-based algorithms (e.g., in the area of recommendation systems) are not applicable in SRaaS; 2) based on the common insight into the utilization of social network, whether we can use SumUp [33], FaceTrust [31] or StopIt [26] to resist malicious friends? These mechanisms can help us, but their additional overhead and implementation complexity is difficult to afford. In fact, SRaaS's reputation

algorithm has provided the good enough penalty mechanism for malicious users through reducing their reputations over time. Thus, our socially-enhanced SRaaS is robust enough to face malicious friends.

5 Experiments

Goals of experiments. We have proved that SRaaS can bound each normal user's loss within $O(M)$. However, we still have the following three important questions to answer: 1) How is the additional overhead introduced by SRaaS? 2) How is the defense capability comparison between SRaaS (and socially-enhanced SRaaS) and the prevalent rank-based tag search schemes from the experimental perspective? 3) How is the defense capability of SRaaS under "worst-case" tag spam attack, which tries to achieve the largest loss of each user?

5.1 Experimental Setup

We developed a prototype tagging system with all mechanisms of SRaaS and socially-enhanced SRaaS in Java. We deployed this SRaaS tagging system on a machine with Intel Xeon Quad Core HT 3.7 GHz CPU and 16 GB of RAM; moreover, we run a MySQL 5.1.54 server on this machine as the SRaaS tagging system's back-end. We used a public and large-scale (1,250,000 annotations) Del.icio.us dataset [30] to generate a realistic environment for our experiments.

Resource setting. In order to construct a "full of spam" environment, we additionally generate 100 misleading tags for each of the resources in the dataset, although these resources have already been annotated with many misleading tags in the dataset.

Honest users' behaviors. We generate $10,000$ honest users according to the trace in our Del.icio.us dataset. These users' tagging behavior distributions [24] are realistic and could be looked as the key basis of designing our approach (discussed in Section 2.4). During the whole process of our experiments, the honest users participates in the system to search, consume and annotate consumed resources. They annotate resources with correct tags. Even if an honest user consumes an unwanted resource, she also annotates this resource with correct tags. Throughout our experiments, honest users may leave and rejoin the system randomly.

Social network setting. We generate the social network of experiments according to the small world property of online social networks [29], and establish the friend-relationships for users based on widely adopted Kleinberg model [17]. We use Watts-Strogatz model [36] to instantiate a social networking graph: a 10000-node graph (i.e., $10,000$ honest users mentioned earlier) with average degree of 24.[2]

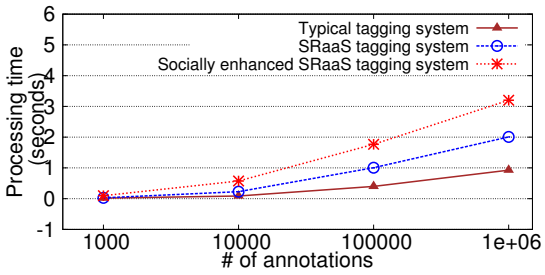


Figure 3: System Overhead.

5.2 System Overhead

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

In this evaluation, we mainly measure processing time (seconds) which means the running time of executing a resource discovery process. We run the experiments on different scales of datasets which contain x annotations. We vary x between $1,000$ and $1,000,000$ to cover a wide range of real-world settings.

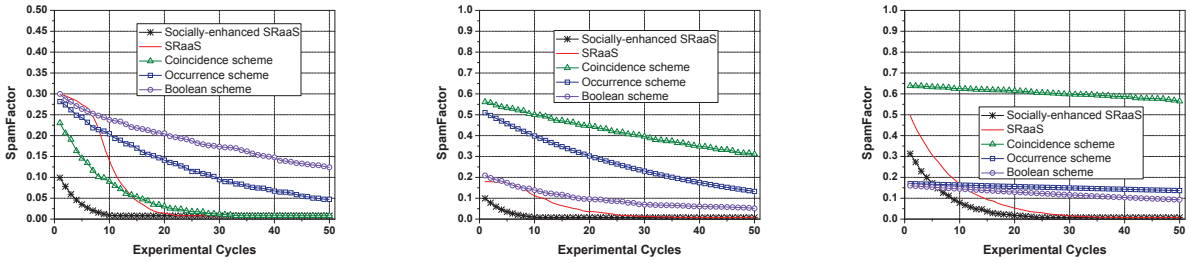
Figure 3 shows the running time of executing a resource discovery process on different scales of environments. From this result, we observe that SRaaS and socially-enhanced SRaaS do not introduce too much overhead to tagging systems. In addition, the additional overhead introduced by social network (than basic SRaaS) is totally acceptable.

5.3 Experiments under Representative Tag Spam Attacks

To answer the second question of our evaluation goals, this section compares our approach (both SRaaS and socially-enhanced SRaaS) with three prevalent tag search schemes under three different tag spam attacks.

Experimental execution. Each of the experiments is composed of 50 *experimental cycles*. In each experimental cycle, each honest user searches $0-10$ specific tags, and then the user selects and consumes these resources according to the order of search results. In each experimental cycle, there are $1,000$ new resources to be added into the environment. These new resources are assigned to honest users randomly, and tagged by those users. After each cycle, the number of spam search results is calculated. Each experiment is run 5 times and the results of all runs are averaged.

[2] The definition of the term node's degree of social networking graph can be found in [29].



(a) Impact of Normal Attack. (b) Impact of Collusive Attack. (c) Impact of Tricky Attack.

Figure 4: Impact of Three Tag Spam Attacks (Lightweight Attacks).

In the following, we first present a metric used to evaluate the capability of tag spam resistances. Then, we describe three prevalent tagging schemes to be compared and three tag spam attacks. Finally, we show our experimental results.

5.3.1 Defense Metric

Although we have always been using loss (the total number of unwanted consumed resources) to evaluate the capability of SRaaS and other tagging systems in the former discussion, we would use *SpamFactor* [12, 18], a metric accepted widely to quantify the "spam impact" in the tag search results, as our experimental metric. The major reason is that SpamFactor is affected by not only the number of unwanted consumed resources (i.e., our defined loss) but also the unwanted resources' positions in the search results. Thus, we consider SpamFactor to be a more comprehensive metric for evaluating SRaaS's defense capability.

For SpamFactor, the higher the position of an incorrect annotation in the search result, the higher SpamFactor. Furthermore, the study in [19] indicates that "SpamFactor less than 0.1 is thought as robust enough for tag search results". Note that, in our experiments, the SpamFactor focuses on the top 10 search results.

5.3.2 Three Representative Tagging Services

There are three types of representative tag search schemes (or models): Boolean [4], Occurrence [3], and Coincidence [18]. They not only have been widely adopted in real-world tagging systems, but also cover features of most of the existing rank-based search schemes. We construct the above three types of schemes and compare them with SRaaS in the following experiments. Given a query tag t, the system returns a ranked list of annotations containing the query tag t, the above three search schemes work as follows.

Boolean scheme. Boolean scheme is an easy tag search scheme which is used in some of current tagging systems (e.g., Slideshare [4]). The key strategy of Boolean scheme is that the tagging system randomly ranks annotations that match the query tag t.

Occurrence scheme. Occurrence scheme (e.g., Rawsugar [3]) 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 an anti-spam tag search model which has been used by some spam-resistant approaches (e.g., SpamClean [41] and Coincidence [18]). It considers users' reliability degrees according to the following basis: 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 its peers. Specifically, Coincidence scheme assigns each user a global reliability 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' reliability degrees of each annotation.

5.3.3 Three Types of Tag Spam Attacks

There are 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 attack, they select some of the resources in the system, and annotate these resources with some misleading tags randomly to achieve the purpose of misleading normal users. Normally, the normal tag spam attack acts independently, that is, these malicious users are "lousy annotators".

Collusive attack. In some cases, malicious users 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 annotate the resources with both correct and incorrect tags, and then publish these annotations to the system. The existing anti-spam mechanisms (e.g., Coincidence scheme) will be a victim when encountering this tricky attack.

Execution of malicious users. We construct two sets for malicious users, $\mathbb{L}$ and $\mathbb{H}$, and use $|\mathbb{L}|$ and $|\mathbb{H}|$ to denote the size (i.e., the number of attackers) of two sets, respectively. At the startup of experiments, each attacker in $\mathbb{L}$ can publish $10-50$ incorrect annotations for any of his resources and then leave system; each attacker in $\mathbb{H}$ can publish $100-500$ incorrect annotations for any of his resources and then leave system. Note that the concrete behaviors of attackers in $\mathbb{L}$ and $\mathbb{H}$ are according to specific attack strategy.

5.3.4 Evaluating Lightweight Attacks

We set $|\mathbb{L}|$ to $2,000$ and $|\mathbb{H}|$ to 0, and compare our approach with three search schemes (described above) under the three attacks respectively. Note that we call the condition of $|\mathbb{H}| = 0$ as lightweight tag spam attacks. Figure 4 shows the impact of three attacks.

Discussion on normal attack. Observing from Figure 4a, we conclude that Boolean and Occurrence schemes are impacted significantly by normal attack, because their SpamFactors decrease to below 0.1 after 30 and 50 experimental cycles respectively. This cannot be tolerant in practice. On the other hand, Coincidence scheme presents good defense capability for 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. In addition, we note the SpamFactor of SRaaS is high at the startup; however, its SpamFactor begins to decline quickly after the 8th experimental cycle and decreases to below 0.1 in the 12th experimental cycle. The reason is that compasses and some honest users who have published many correct annotations receive high reputation scores after several cycles, and then users can always consume the resources of annotations published by similarity groups. We can see that socially-enhanced SRaaS has very good capability throughout the whole experiment. As shown in Figure 4a, the SpamFactor of socially-enhanced SRaaS is lower than 0.1 at the beginning of the experiment.

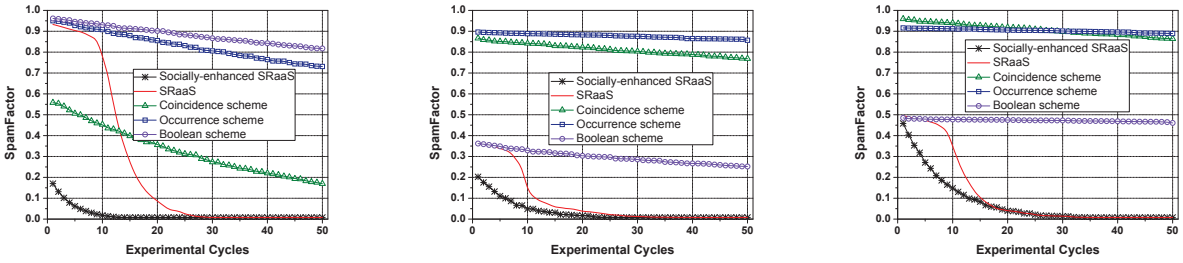
Discussion on collusive attack. Figure 4b indicates that the collusive attack can pose serious influence on both Coincidence and Occurrence schemes. Coincidence scheme works badly because under collusive attacks the reliability degrees of many malicious users become high, which means most incorrect annotations 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 answer to this question is SRaaS and Boolean schemes select random resources to consume, so that users choose the annotations attacked by the collusive attackers with much lower probability than Coincidence and Occurrence schemes. Moreover, with experimental cycles growing, users establish reputation with some honest users (including compasses) gradually, so that the SpamFactors of SRaaS and socially-enhanced SRaaS can keep decreasing all the time.

Discussion on tricky attack. Figure 4c shows the impact of tricky attack. Coincidence scheme cannot provide any qualified tag search results under tricky attack. We note 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 experimental cycles growing. Influenced by tricky attack, the SpamFactors of SRaaS and socially-enhanced SRaaS are very high (0.5 and 0.3, respectively) at the startup. However, their SpamFactors converge quickly to below 0.1 in 8 and 15 experimental cycles respectively. This reveals that although some tricky attackers can obtain high reputations, our approach can find and punish them as experimental cycles increases.

5.3.5 Evaluating Heavyweight Attacks

We set $|\mathbb{L}|$ to 0 and $|\mathbb{H}|$ to $20,000$, and compare our approach with three search schemes (described above) under the three attacks respectively. Note that we call the condition of $|\mathbb{L}| = 0$ as heavyweight tag spam attacks. Figure 5 shows the impact of three attacks.

Discussion. As shown in Figure 5, we note that all of five approaches all deteriorate significantly under heavyweight attacks. *Why is Coincidence scheme impacted so significantly under heavyweight normal attacks (shown in Figure 5a)?* From the description of Coincidence scheme, we know that Coincidence scheme is an anti-tag spam mechanism and the experiment shown in Figure 4a also proves the scheme is able to defend against normal tag spam attack. However, its bad result under heavyweight normal attacks reveals that as malicious users and incorrect annotations proliferate, some malicious



(a) Impact of Normal Attack. (b) Impact of Collusive Attack. (c) Impact of Tricky Attack.

Figure 5: Impact of Three Tag Spam Attacks (Heavyweight Attacks).

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 heavyweight tricky attacks (shown in Figure 5c) higher than those under lightweight tricky attacks (shown in Figure 4c)? 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 heavyweight tricky attacks).

5.4 Evaluating the Worst-Case Attack

To answer the third question of the experimental goals, we generate experimental setting again and construct a worst-case attack, which aims to cause the largest loss. This attack strategy is also called *optimal attacks* in recommendation area [38]. We aim to use this attack to demonstrate the defense capability of SRaaS at experimental aspect.

We generate $5,000$ 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.

To construct the worst-case attack, on the first experimental cycle, 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 in that experimental cycle, and each user in $\mathbb{N}_1$ publish the same incorrect annotations. In the second experimental 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.

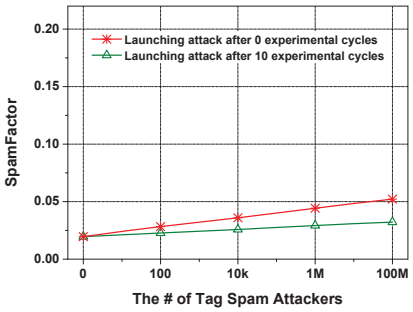


Figure 6: Impact of the worst-case attack.

Figure 6 shows that SRaaS's per-experimental cycle SpamFactor under the worst-case attack, when the attack starts after experimental cycle 0 and 10. It is clear that the SpamFactor is below 0.1 throughout the whole experiment. This proves the robustness of SRaaS is still good under large-scale worst-case 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 rank-based spam-resistant approach which is adoptive to any typical tagging systems. SRaaS provides provable guarantees on its defense capability, which enables SRaaS to significantly diminish the influence of spam attacks in any various and scales. Through experiments, we demonstrate our approach outperforms the three existing tag search schemes under various tag spam attacks and can effectively defend against our constructed worst-case attacks.

References

[1] Decaptcher. http://de-captcher.com/.

[2] Facebook. http://www.facebook.com/.

[3] Rawsugar. http://rawsugar.com/.

[4] Slideshare. http://slideshare.net/.

[5] Technorati. http://www.technorati.com/.

[6] Baruch Awerbuch and Thomas P Hayes. Online collaborative filtering with nearly optimal dynamic regret. In *SPAA*, 2007.

[7] Toine Bogers and Antal van den Bosch. Using language models for spam detection in social bookmarking systems. In *ECML PKDD*, 2008.

[8] Christopher H. Brooks and Nancy Montanez. Improved annotation of the blogosphere via autotagging and hierarchical clustering. In *WWW*, pages 625–632, 2006.

[9] Anestis Gkanogiannis and Theodore Kalamboukis. A novel supervised learning algorithm and its use for spam detection in social bookmarking systems. In *ECML PKDD*, 2008.

[10] Scott A. Golder and Bernardo A. Huberman. Usage patterns of collaborative tagging systems. *J. Information Science*, 32(2):198–208, 2006.

[11] Harry Halpin, Valentin Robu, and Hana Shepherd. The complex dynamics of collaborative tagging. In *WWW*, pages 211–220, 2007.

[12] Paul Heymann, Georgia Koutrika, and Hector Garcia-Molina. Fighting spam on social web sites: A survey of approaches and future challenges. *IEEE Internet Computing*, 11(6):36–45, 2007.

[13] Paul Heymann, Georgia Koutrika, and Hector Garcia-Molina. Can social bookmarking improve web search? In *WSDM*, pages 195–206, 2008.

[14] Andreas Hotho, Robert Jäschke, Christoph Schmitz, and Gerd Stumme. Information retrieval in folksonomies: Search and ranking. In *ESWC*, pages 411–426, 2006.

[15] Ajita John and Dorée Seligmann. Collaborated tagging and expertise in the enterprise. In *Collaborative Web Tagging Workshop in conjunction with WWW*, 2006.

[16] Chanju Kim and Kyu-Baek Hwang. Naive Bayes Classifier Learning with Feature Selection for Spam Detection in Social Bookmarking. In *ECML PKDD*, 2008.

[17] Jon M. Kleinberg. The small-world phenomenon: an algorithm perspective. In *STOC*, 2000.

[18] Georgia Koutrika, Frans Adjie Effendi, Zoltán Gyöngyi, Paul Heymann, and Hector Garcia-Molina. Combating spam in tagging systems. In *AIRWeb*, 2007.

[19] Georgia Koutrika, Frans Adjie Effendi, Zoltán Gyöngyi, Paul Heymann, and Hector Garcia-Molina. Combating spam in tagging systems. Technical Report Technical report, available at http://dbpubs.stanford.edu/pub/2007-11, November 2007.

[20] Beate Krause, Christoph Schmitz, Andreas Hotho, and Gerd Stumme. The anti-social tagger: detecting spam in social bookmarking systems. In *AIRWeb*, pages 61–68, 2008.

[21] Ralf Krestel and Ling Chen. Using Co-occurence of Tags and Resources to Identify Spammers. In *ECML PKDD*, 2008.

[22] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *KDD*, pages 611–617, 2006.

[23] Antonia Kyriakopoulou and Theodore Kalamboukis. Combining clustering with classification for spam detection in social bookmarking systems. In *ECML PKDD*, 2008.

[24] Rui Li, Shenghua Bao, Yong Yu, Ben Fei, and Zhong Su. Towards effective browsing of large scale social annotations. In *WWW*, pages 943–952, 2007.

[25] Bo Liu, Ennan Zhai, Huiping Sun, Yelu Chen, and Zhong Chen. Filtering spam in social tagging system with dynamic behavior analysis. In *ASONAM*, pages 95–100, 2009.

[26] Xin Liu, Xiaowei Yang, and Yanbin Lu. To filter or to authorize: network-layer dos defense against multimillion-node botnets. In *SIGCOMM*, pages 195–206, 2008.

[27] Benjamin Markines, Ciro Cattuto, and Filippo Menczer. Social spam detection. In *AIRWeb*, pages 41–48, 2009.

[28] Cameron Marlow, Mor Naaman, Danah Boyd, and Marc Davis. Position paper, tagging, taxonomy, flickr, article, toread. In *Hypertext*, pages 31–40, 2006.

[29] Alan Mislove, Massimiliano Marcon, P. Krishna Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Internet Measurement Comference*, 2007.

[30] Arvind Narayanan. Del.icio.us dataset. http://randomwalker.info/data/delicious/delicious-rss-1250k.gz, accessed on Sep 9, 2014.

[31] Michael Sirivianos, Kyungbaek Kim, and Xiaowei Yang. FaceTrust: Assessing the Credibility of Online Personas via Social Networks. In *HotSec*, 2009.

[32] Michael Sirivianos, Xiaowei Yang, and Kyungbaek Kim. Socialfilter: Collaborative spam mitigation using social networks. *CoRR*, abs/0908.3930, 2009.

[33] Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-Resilient Online Content Voting. In *NSDI*, 2009.

[34] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. Captcha: Using hard ai problems for security. In *EUROCRYPT*, pages 294–311, 2003.

[35] Kevin Walsh and Emin Gün Sirer. Experience with an object reputation system for Peer-to-Peer filesharing. In *NSDI*, May 2006.

[36] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world networks'. *Nature*, 393(6684):440–442, 1998.

[37] Zhichen Xu, Yun Fu, Jianchang Mao, and Difu Su. Towards the semantic web: Collaborative tag suggestions. In *Collaborative Web Tagging Workshop in conjunction with WWW*, 2006.

[38] Haifeng Yu, Chenwei Shi, Michael Kaminsky, Phillip B. Gibbons, and Feng Xiao. Dsybil: Optimal sybil-resistance for recommendation systems. In *IEEE Symposium on Security and Privacy*, pages 283–298, 2009.

[39] Valentina Zanardi and Licia Capra. Social ranking: uncovering relevant content using tag-based recommender systems. In *RecSys*, pages 51–58, 2008.

[40] Ennan Zhai. Proof for sraas. Technical report, Department of Computer Science, Yale University, 2015. Available at http://www.cs.yale.edu/homes/zhai-ennan/proof.pdf.

[41] Ennan Zhai, Huiping Sun, Sihan Qing, and Zhong Chen. Spamclean: Towards spam-free tagging systems. In *CSE (4)*, pages 429–435, 2009.

7 Appendix

Lemma 1. *Considering any given M, α, ω and h, there are H honest users and S spam attackers in the systems. We define Δ to be the number of correct annotations that are published by one or more similarity groups and are consumed by an honest user. Regardless of the strategies of spam attackers, we have:* $\Delta \leq M\lceil \log_\alpha(\alpha \cdot h \cdot (H+S)/\omega)\rceil$.

Proof sketch. For a honest user, Alice, 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 groups, SRaaS would increase the reputation score of at least one similarity group. Note that no user can have a reputation score higher than $\alpha \cdot h$. Otherwise, this user will have a reputation score higher than h before the last reputation increasement, which is impossible. Because every user starts with a reputation score $\omega/(H+S)$, the reputation score of a similarity group member can be multiplied by α for at most $\lceil \log_\alpha(\alpha \cdot h \cdot (H+S)/\omega)\rceil$ times before his reputation score reaches $\alpha \cdot h$. Therefore, Alice can consume at most $M\lceil \log_\alpha(\alpha \cdot h \cdot (H+S)/\omega)\rceil$ resources published by similarity groups when there is no resource whose reputation score $\geq h$. We get $\Delta \leq M\lceil \log_\alpha(\alpha \cdot h \cdot (H+S)/\omega)\rceil$. □

Observation from Lemma 1. Lemma 1 shows us that with a small M, Δ will be small as well. In addition, because the resource published by similarity groups consumed by Alice is a random resource from at least $\gamma\xi \cdot Y$[3] annotations whose reputations are lower than h, we can view Δ 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 the resources of incorrect annotations whose reputations are lower than h consumed by an honest user, and G_s means the number of the resources of correct annotations whose reputation scores are lower than h consumed by the user. According to the geometric distribution, we can get $E[P_s + G_s] \leq \frac{1}{\gamma \cdot \xi}\Delta$, and $E[G_s] \leq \frac{1}{\xi}\Delta$.

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

Lemma 2. *Considering any given α, β, ω and h, let P_f be the number of the resources of incorrect annotations whose reputation scores $\geq h$ consumed by an honest user and G_s be the number of the resources of correct annotations whose reputation scores are lower than h consumed by the user. Then, regardless of the strategies of spam attackers, the relationship between P_f and G_s is:* $h \cdot (1-\beta) \cdot P_f \leq (\omega + h \cdot \alpha - h) \cdot G_s$.

Proof sketch. Spam attackers can only increase their reputation scores by publishing correct annotations. Thus, each such annotation enables these spam attackers to obtain less than $h \cdot \alpha - h$ additional reputation scores. On the other hand, whenever Alice 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 β. Therefore, SRaaS's algorithm confiscates at least $h - h \cdot \beta$ 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 $h \cdot (1-\beta) \cdot P_f \leq (\omega + h \cdot \alpha - h) \cdot G_s$. □

Theorem 1. *Let $|\mathbb{L}|$ be the loss of a given SRaaS user u and S be the total number of spam attackers. We define Δ to be the number of correct annotations that are published by one or more similarity groups and are consumed by the user u. Then, regardless of the strategies of spam attackers, $|\mathbb{L}|$ is:*

$$|\mathbb{L}| \leq \Delta \cdot \frac{1}{\gamma \cdot \xi} \cdot (1 - \gamma + \frac{\gamma \cdot (\alpha - 1 + \omega)}{1-\beta}) \qquad (3)$$

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

Corollary 1. We use $\alpha = 2$, $\beta = 0.5$ and $\omega = h/\alpha = 1/2 = 0.5$ to achieve the guarantees of SRaaS. Because of Lemma 1 [40], we know $\Delta \leq M\lceil \log_\alpha(\alpha \cdot h \cdot (H+S)/\omega)\rceil$, where M is the number of similarity groups. Thus, if set $\alpha = 2$, $\beta = 0.5$ and

$\omega = 0.5$, we have $|\mathbb{L}| \leq \frac{1+2\gamma}{\gamma \cdot \xi} \cdot M$. Because the constants γ and ξ are not 0, $|\mathbb{L}|$ becomes $O(M)$.