SocialFilter: Introducing Social Trust to Collaborative Spam Mitigation

Michael Sirivianos
Telefonica Research
msirivi@tid.es

Kyungbaek Kim
UC Irvine
kyungbak@uci.edu

Xiaowei Yang
Duke University
xwy@cs.duke.edu

***Abstract*—We propose SocialFilter, a trust-aware collaborative spam mitigation system. Our proposal enables nodes with no email classification functionality to query the network on whether a host is a spammer. It employs Sybil-resilient trust inference to weigh the reports concerning spamming hosts that collaborating spam-detecting nodes (reporters) submit to the system. It weighs the spam reports according to the trustworthiness of their reporters to derive a measure of the system's belief that a host is a spammer. SocialFilter is the first collaborative unwanted traffic mitigation system that assesses the trustworthiness of spam reporters by both auditing their reports and by leveraging the social network of the reporters' administrators.**

The design and evaluation of our proposal offers us the following lessons: a) it is plausible to introduce Sybil-resilient Online-Social-Network-based trust inference mechanisms to improve the reliability and the attack-resistance of collaborative spam mitigation; b) using social links to obtain the trustworthiness of reports concerning spammers can result in comparable spam-blocking effectiveness with approaches that use social links to rate-limit spam (e.g., Ostra [27]); c) unlike Ostra, in the absence of reports that incriminate benign email senders, SocialFilter yields no false positives.

I. Introduction

Centralized email reputation services that rely on a small number of trusted nodes to detect and report spammers, e.g., [1, 5, 8], are being challenged by the increasing scale and sophistication of botnets. In particular, spammers employ multiple malicious hosts, each for a short period of time. In turn, those hosts spam multiple domains for short periods [7, 23, 30]. The above strategies reduce the effectiveness of spam detection from a small number of vantage points. Moreover, several of these services require paid subscription (e.g., CloudMark [1] and TrustedSource [5]).

Motivated by the shortcomings in terms of effectiveness and cost of the above email reputation services, researchers have proposed open and collaborative peer-to-peer spam filtering platforms, e.g., [44, 45]. These collaborative systems assume compliant behavior from all participating spam reporting nodes, i.e., that nodes submit truthful reports regarding spammers. However, this is often an unrealistic assumption given that these nodes may belong to distinct trust domains.

A recent collaborative spam email sender detection system [33] employs trust inference [21, 26, 31] to weigh spammer reports according to the trustworthiness of their reporters. However it is still susceptible to Sybil [15] attacks.

To address the above challenges, we propose SocialFilter: a collaborative spam filtering system that uses social trust embedded in Online Social Networks (OSN) to assess the trustworthiness of spam reporters. Our proposal aims at aggregating the experiences of multiple spam detectors, thus democratizing spam mitigation. In particular, each SocialFilter node submits *spammer reports* (§II-B) to a centralized repository. These reports concern spamming hosts identified by their IP addresses. SocialFilter is a trust layer that exports a measure of the system's belief that a host is spamming. Thus, it enables nodes without spam detection capability to collect the experiences of nodes with such capability and use them to classify email connection requests from unknown hosts.

SocialFilter nodes are managed by human administrators (admins). Our insight is that nodes maintained by competent and trusted admins are likely to generate trustworthy spammer reports, while nodes maintained by admins known to be less competent are likely to generate unreliable reports. The repository utilizes a trust inference method to assign to each node a *reporter trust* (§III-A) value. This value reflects the system's belief that the spammer reports of a node are reliable.

The trust inference method exploits trust transitivity and operates on a trust graph that is formed at the repository as follows (§III-A). Each vertex in the graph is a SocialFilter node, which is administered by a human admin. The edges in the graph are *direct trust* values between nodes administered by admins that are socially acquainted. The social relationships between admins are obtained from massive OSN providers, First, each admin explicitly assigns a direct trust value to nodes that are administered by his friends, based on his assessment of their competency. Second, the direct trust values between nodes are updated to reflect the similarity between their spammer reports. This is based on the observation that trustworthy nodes are likely to report similarly on commonly encountered hosts. Third, the repository computes the maximum trust path from pre-trusted nodes to all other nodes in the trust graph.

However, such transitive trust schemes are known to be vulnerable to the Sybil attack [12, 13, 15]. To mitigate this attack, we again use the social network to assess the belief that a node is a Sybil attacker, which we refer to as *identity uniqueness* (§III-B).

The originator of a spammer report also assigns a confidence level to its report. The reporter trust of a node, its identity uniqueness and its confidence level determine how much weight the repository should place on its report. Subsequently, the repository combines the reports to compute a *spammer belief* value for each host IP reported to the system. This value has a partially Bayesian interpretation, reflecting the belief that a host is spamming (§III-A2,§III-B). This value is exported by the repository to other online systems for diverse purposes. For example, email servers can use them to filter out email messages that originate from IPs designated as spammers.

A recent proposal, Ostra [27], combats unwanted traffic by forcing it to traverse social links annotated by credit balances. The per-link credit balances rate-limit unwanted communication. Unlike Ostra, our proposal does not use social

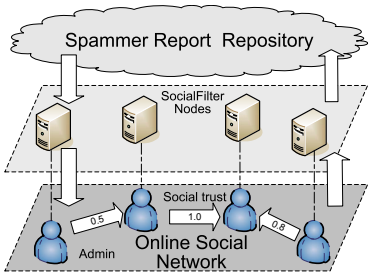


Figure 1. SocialFilter architecture.

links to rate-limit unwanted traffic. Instead it utilizes social links to bootstrap trust between reporters, and to suppress Sybil attacks. We compare Ostra's and SocialFilter's approach in leveraging social trust.

We evaluate our design (§IV) using a $50K$-node sample of the Facebook social network. We demonstrate through simulation that collaborating SocialFilter nodes are able to suppress spam email traffic in a reliable and responsive manner. Our comparison with Ostra shows that our approach is less effective in suppressing spam when the portion of spammers in the network and the number of Sybils they employ exceeds a threshold. However, Ostra can result in a non-negligible percentage of legitimate emails being blocked (false positives), which is highly undesirable. This holds even when receivers do not falsely classify legitimate email as spam. In contrast, in this case SocialFilter yields no false positives. Given the severity of false positives, these results suggest that our system can be a better alternative under a multitude of deployment scenarios.

To the best of our knowledge, SocialFilter is the first OSN-based collaborative spam filtering system to use Sybil-resilient trust inference to assess the overall trustworthiness of a node's spammer reports. Our contributions are:

- We have combined existing trust inference mechanisms to derive an attack-resistant mechanism that assigns trust values (spammer belief) to hosts detected by a collaborative spam mitigation system.
- We have evaluated our design using simulations and compared it to Ostra [27] to illustrate our design's advantages and shortcomings.
- We have demonstrated the plausibility of using social trust to improve the reliability and attack-resilience of collaborative spam mitigation systems.

An early version of this work appeared as a workshop short paper [34]. In this paper, we provide a more detailed description of the identity uniqueness mechanism. We also present more evaluation results with respect to our proposal's effectiveness in the absence of false reporters and colluders.

II. System Overview

A. SocialFilter Components

Figure 1 depicts SocialFilter's architecture. At a high-level, the system comprises the following components: 1) human users that administer networked devices/networks (*admins*) and join a social network and maintain a unique account; 2) *SocialFilter nodes* (or reporters) that are administered by specific admins and participate in monitoring and reporting the behavior of email senders; 3) *spammer reports* submitted by SocialFilter nodes concerning email senders they observe; and 4) a centralized repository that receives and stores spammer reports, and computes trust values.

The same admin that administers a SocialFilter node also administers a group of online systems that interface with the node to report spamming behavior. Interfacing systems can be SMTP servers or IDS systems [29] that register with the repository. The interfacing system can also be one driven by a human user who reports an email (and consequently its originating email server) as spam.

B. Spammer Reports

An email characterization system uses the `ReportSpammer(`h`, confidence)` call of the SocialFilter node RPC API to feedback its observed behavior for an email sender h to the node. The first argument identifies the email sender, i.e., an IP address. The second argument is the confidence with which the system is reporting that the specified host is a spammer. The confidence takes values from 0% to 100% and reflects the fact that in many occasions traffic classification has a level of uncertainty. For example, an email server that sends both spam and legitimate email may or may not be a spammer. For instance, the confidence may be equal to the portion of emails received by host h that are spam [32].

Then, the node submits a corresponding spammer report to the repository to share its experience. For example, if a node i's spam analysis indicates that half of the emails received from host with IP h are spam, i reports:

$$[\text{spammer report}]\ h,\ 50\%$$

To prevent forgery of reports and maintain accountability, nodes authenticate with both the repository and the OSN provider using standard single-sign-on authentication techniques, e.g., [16, 37] or Facebook Connect [4].

C. Determining whether a Host is Spamming

Our system relies on the fact that nodes comprising Internet systems such as email servers, honeypots, IDS, and etc, are administered by human admins. These users maintain accounts in online social networks (OSN). The SocialFilter centralized repository utilizes two dimensions of trust embedded in OSNs to determine the trustworthiness of the reports submitted by SocialFilter nodes:

Reporter trust. The repository computes *reporter trust* values for all nodes by employing a transitive trust inference mechanism. This mechanism relies on comparing the reports of nodes that are socially acquainted to derive pairwise *direct trust* values (§III-A). If two friend nodes i and j have submitted reports concerning the same hosts, the repository can compare their reports to determine the direct trust value d_{ij}. The repository initializes the direct trust d_{ij} to a trust value explicitly submitted by the admin of i. This value is i's assessment on his friend j's ability to correctly maintain its node.

Identity uniqueness. The repository defends against Sybil attacks [15] by exploiting the fact that OSNs can be used for resource testing [27, 38, 42]. The test in question is a Sybil attacker's ability to create and sustain acquaintances. Using a SybilLimit-based [42] technique (§III-B), the OSN provider assigns an *identity uniqueness* value to each node. This value quantifies the system's belief in that node not being a Sybil.

A characterization system can use the `IsSpammer(h)` call of the SocialFilter node RPC API to obtain a *spammer belief* value, which quantifies how certain the system is that host h is spamming. The node obtains this value by querying the repository. The repository derives this value by aggregating spammer reports concerning h, and these reports are weighted by the reporter trust and identity uniqueness of their submitters.

D. Assumptions

We make the following assumptions in our design:

Trustworthy SocialFilter reporters send mostly correct reports and report similarly: We assume that competent and trustworthy SocialFilter admins have correctly configured their spam detection systems, so that their node sends mostly correct reports. We also assume that when they report the same spamming host, their reports mostly match, since a host is expected to send spam to most of the nodes it connects to [41]. In the rest of this paper, we call correctly configured and trustworthy SocialFilter reporters *honest*.

Trusted repository: We assume that the OSN provider and the SocialFilter repository reliably maintain the social graph, and the spammer reports. We trust the repository to correctly compute the spammer belief values.

E. Threat Model

SocialFilter is a collaborative platform aiming at suppressing malicious traffic. In addition, it is an open system, meaning that any admin with a social network account and a device can join. As such, it is reasonable to assume that our system itself will be targeted in order to disrupt its operation. Our system faces the following security challenges:

False spammer reports. Malicious nodes may issue false reports aiming at reducing the system's ability to detect spam or at disrupting legitimate email traffic. We treat incorrectly configured nodes as *malicious*.

Direct trust manipulation. The transitive trust scheme we use to determine a node's reporter trust is vulnerable to manipulation as follows. First, a malicious SocialFilter node may purposely send false spammer reports in order to increase the direct trust between himself and malicious friends that also send false reports. This can result in himself or his malicious friends to have increased reporter trust. Second, he may purposely send true spammer reports in order to increase its direct trust with honest nodes and consequently his reporter trust. Third, he may purposely send false spammer reports to reduce his direct trust with honest nodes, aiming at the honest nodes obtaining decreased reporter trust. Fourth, a malicious host may send legitimate email to an honest node x and spam email to x's honest friend node y, aiming at decreasing the direct trust between x and y. This manipulation can decrease the reporter trust of x or y.

Sybil attack. An adversary may attempt to create multiple SocialFilter identities aiming at increasing its ability to subvert the system using false spammer reports. Defending against Sybil attacks without a trusted central authority that issues verified identities is hard. Many decentralized systems try to cope with Sybil attacks by binding an identity to an IP address. However, malicious users can readily harvest IP addresses through BGP hijacking [30] or by commanding a large botnet.

III. Design

A. Reporter Trust

Malicious nodes may issue false spammer reports to manipulate the perceived belief that a host is a spammer. In addition, misconfigured nodes may also issue erroneous spammer reports. SocialFilter can mitigate the negative impact of malicious or incorrect reports by assigning higher weights to reports obtained from more nodes with higher reporter trust.

The repository maintains a reporter trust value $0 \leq rt_i \leq 1$ for each node i managed by an admin in the social graph. This trust value corresponds to the repository's estimation of the belief that node j's reports are accurate. It is obtained from three sources: a) manual trust assignments between friends in the social networks; b) spammer report comparison; and c) transitive trust.

To derive trust values, the repository needs to maintain the social graph $\mathcal{S}(\mathcal{V}, \mathcal{E})$ of the admins in the system. $\mathcal{V}$ denotes the set of the admins and $\mathcal{E}$ denotes the set of the friend connections between socially acquainted admins. The repository also maintains a reporter trust graph $T(\mathcal{V}, E)$. The vertices of this graph is the set of all admins as is the case for graph $\mathcal{S}(\mathcal{V}, \mathcal{E})$. The edges E are the edges in $\mathcal{E}$ annotated with *direct trust* values between acquainted SocialFilter nodes. Next, we describe how the direct trust values are derived and how the reporter trust values are computed using $T(\mathcal{V}, E)$.

User-defined trust. First, to initialize the direct trust values, the repository relies on the fact that nodes are administered by human users. Admins that are socially acquainted can assess each other's competence. An admin i tags his acquaintance admin j with a *user-defined trust* value $0 \leq ut_{ij} \leq 1$ based on his belief on j's ability to correctly configure his node. The repository uses this value to initialize the direct trust value between friend nodes i and j: $d_{ij} = ut_{ij}$. Users frequently use the OSN to add friends and to communicate with each other, thus the requirement for administrators to rate each other should not induce a substantial usability burden.

Spammer reports comparison. Second, the repository dynamically updates the direct trust d_{ij} by comparing spammer reports submitted by two nodes i and j. The spammer reports of two friend nodes i and j can be compared if both nodes have reported on the same host h. Intuitively, if i and j share similar opinions on h, i should place high trust in j's reports. Let $0 \leq v_{ij}^k \leq 1$ be a measure of similarity between i and j's k_{th} report on a common host. The repository updates i's direct trust to j using an exponential moving average:

$$d_{ij}^{k+1} = \alpha * d_{ij}^k + (1 - \alpha) * v_{ij}^{k+1} \quad (1)$$

As i and j submit more common reports, the direct trust d_{ij}^k gradually converges to the similarity of reports from i and j. α is a system parameter that affects the influence of history on direct trust assessment.

Transitive trust. Third, the repository incorporates direct trust and transitive trust [18, 19] to obtain the reporter trust value for i: rt_i. It does so by analyzing the reporter trust graph $T(\mathcal{V}, E)$ from the point of view of a small set of pre-trusted nodes in $\mathcal{V}$. These pre-trusted nodes are administered by competent admins that are fully trusted by the SocialFilter repository.

We use transitive trust for the following reasons: a) due to

the large number of nodes, the admin of a pre-trusted node i cannot assign a user-defined trust ut_{ij} to every admin of a node j, as he may not know him; b) due to the large number of email hosts, a pre-trusted node i may not have encountered the same hosts with another node j, thus the repository may be unable to directly verify j's reports; and c) even if a pre-trusted node i has a direct trust value for another node j, the repository can improve the correctness of rt_j by learning the opinions of other nodes about j.

The overall reporter trust rt_j can be obtained as the maximum trust path between a pre-trusted node i and the node j in the trust graph $T(\mathcal{V}, E)$. That is, for each path $p \in P$, where P is the set of all paths between the pre-trusted node and j:

$$rt_j = max_{p \in P}(\Pi_{u \to v \in p} d_{uv}) \quad (2)$$

The above trust value is computed from the point of view of a single pre-trusted node. We repeat this process for every pre-trusted node. We then average the reporter trust values for all pre-trusted nodes to derive a final rt_j value. We use multiple pre-trusted nodes to ensure that there is a trust path from a pre-trusted node to most honest nodes j. Also, use many pre-trusted nodes limit the influence of attackers that manage to establish a high trust path with one of them.

Similar to Credence [39], we use the maximum trust path because it can be efficiently computed with Dijkstra's shortest path algorithm in $O(|E| \log |\mathcal{V}|)$ time for a sparse $T(\mathcal{V}, E)$. In addition, it yields larger trust values than the minimum or average trust path, resulting in faster convergence to high confidence on whether a host is spamming. Finally, it mitigates the effect of malicious nodes that have low direct trust value towards honest nodes.

We compute the reporter trust from the point of view of a few pre-trusted nodes, instead of the point of view of each individual node for two reasons: a) we would need to compute the maximum trust path from each of the nodes in the social graph, which would result in a significant computation overhead ; b) the system aims at assessing the ground truth fact of whether a host is a spammer, and not a subjective fact, therefore it is appropriate to incorporate the transitive trust from multiple points of view.

We compute the maximum trust path from the pre-trusted nodes to all other nodes periodically to reflect changes in direct trust values.

1) Direct Trust Manipulation Attack: Our design is inherently resilient to the direct trust manipulation attack mentioned in §II-E. We discuss each manifestation of this attack in turn using the enumeration used in §II-E.

The reporter trust mechanism by itself does not defend against the first attack. To tackle this attack when the malicious friends are Sybils, we incorporate an additional trust mechanism as described in §III-B. By performing the second attack a malicious node may increase its reporter trust, but in doing so, it will have to submit truthful spammer reports. The third attack is effective only if the maximum trust path to the targeted honest node passes through the malicious node. If there is at least one alternative trust path that yields a higher trust value, then the direct trust value between the malicious and the honest node is ignored. The fourth attack can be effective only if the malicious host has a legitimate reason to send email to the targeted honest nodes.

2) Bayesian Interpretation of Reporter Trust: We note that the max trust path metric described above has a partially Bayesian interpretation. The direct trust edge d_{ij} corresponds to the probability that i assigns to j to correctly assign direct trust to other users or to correctly classify spamming hosts. The maximum trust path from the pre-trusted nodes to a node i, rt_i is derived by multiplying the correct classification probabilities that each intermediate node along the maximum trust path assigns to the next node. Thus, it quantifies the belief that user i will correctly classify spam.

B. Identity Uniqueness

Each node that participates in SocialFilter is administered by human users that have accounts with OSN providers. The system needs to ensure that each user's social network identity is bound to its SocialFilter node. To this end, SocialFilter employs single sign-on authentication mechanisms, such as Facebook Connect [4], to associate the OSN account with the spammer repository account.

However, when malicious users create numerous fake OSN accounts, the spammer belief measure can be subverted. Specifically, a malicious user a with high reporter trust may create Sybils and assign high direct trust to them. As a result, all the Sybils of the attacker would gain high reporter trust. The Sybils can then submit reports that greatly affect the spammer belief values.

We leverage existing OSN repositories for Sybil user detection. Using a SybilLimit-like [42] technique, OSNs can approximate the belief that a node's identity is not a Sybil. We refer to this belief as *identity uniqueness*.

Social-network-based Sybil detection takes advantage of the fact that most social network users have a one-to-one correspondence between their social network identities and their real-world identities. Malicious users can create many identities or connect to many other malicious users, but they can establish only a limited number of trust relationships with real users. Thus, clusters of Sybil attackers are likely to connect to the rest of the social network with a disproportionately small number of edges, forming small quotient cuts.

SocialFilter adapts the SybilLimit algorithm to determine an identity-uniqueness value $0 \leq id_i \leq 1$ for each node i. This value indicates the belief that the administrator of node i corresponds to a unique user in real life and thus is not part of a network of Sybils. To be Sybil-resistant, SocialFilter multiplies the identity-uniqueness value id_i by the reporter trust to obtain the trustworthiness of node i's spammer reports. We now describe in detail how we compute id_i.

First, we provide a brief background on the theoretical justification of SybilLimit. It is known that randomly-grown topologies such as social networks and the web are fast mixing small-world topologies [40]. Thus, in the social graph $\mathcal{S}(\mathcal{V}, \mathcal{E})$, the last edge (also referred to as the tail) traversed by a random walk of $\Theta(\log |\mathcal{V}|)$ steps is an independent sample edge approximately drawn from the stationary distribution of the graph. If we draw $\Theta(\sqrt{|\mathcal{E}|})$ $\Theta(\log |\mathcal{V}|)$-long random walks from a legitimate verifier node v and a legitimate suspect node s, it follows from the generalized Birthday Paradox that the sample tails intersect with high probability. The opposite holds if the suspect resides in a region of Sybil attackers. This is

because the Sybil region is connected via a disproportionally small number of edges to the region of legitimate nodes. Consequently, the tails of random walks from the Sybil suspect are not samples from the same distribution as the tails of random walks from the verifier.

SybilLimit [42] replaces random walks with "random routes" and a verifier node v accepts the suspect s if routes originating from both nodes intersect at the tail. For random routes, each node uses a pre-computed random permutation as a one-to-one mapping from incoming edges to outgoing edges. Each random permutation generates a unique routing table at each node. As a result, two random routes entering an honest node along the same edge always exit along the same edge. This property guarantees that random routes from a Sybil region that is connected to the honest region through a single edge will traverse only one distinct path, further reducing the probability that a Sybil's random routes will intersect with a verifier's random routes.

With SocialFilter's SybilLimit-based technique, the OSN provider computes an identity uniqueness value for each node s in the social graph $\mathcal{S}(\mathcal{V}, \mathcal{E})$. At initialization time, the OSN provider selects l pre-trusted verifier nodes. It also creates $2r$ independent instances of pre-computed random permutation as a one-to-one mapping from incoming edges to outgoing edges (routing table). The first $r = \Theta(\sqrt{|\mathcal{E}|})$ routing tables are used to draw random routes from suspect nodes s and the rest r routing tables are used to draw random routes from the verifier nodes v. For each s, the OSN provider runs the SybilLimit-like algorithm as follows:

1) For each of the l verifiers v, it picks a random neighbor of v. It draws along the random neighbors r random routes of length $w = \Theta(\log|\mathcal{V}|)$, for each instance of the $r = \Theta(\sqrt{|\mathcal{E}|})$ routing tables. It stores the last edge (tail) of each verifier random route.
2) It picks a random neighbor of s and draws along it r random routes of length $w = \Theta(\log|\mathcal{V}|)$, for each instance of the nodes' routing tables. It stores the tail of each suspect random route. We refer to steps (1) and (2) of the algorithm as *random routing.*
3) For each verifier v, if one tail from s intersects one tail from v, that verifier v is considered to "accept" s. We refer to this step as *verification.*
4) It computes the ratio of the number of verifiers that accept s over the total number of verifiers l. That ratio is the computed identity uniqueness value id_s.

Nodes query the OSN provider for the identity uniqueness of their peers. The OSN provider performs the above computations periodically to accommodate for topology changes.

Similar to SybilLimit, SybilInfer [14] takes advantage of the fact that clusters of Sybils are connected to the honest regions of social networks with a disproportionally small number of edges. SybilInfer's [14] Bayesian Sybil detection method derives the probability of a suspect node being a Sybil, which is an explicitly actionable measure of trustworthiness. We considered SybilInfer as an alternative of SybilLimit in SocialFilter. Although SybilInfer provides a more formal probabilistic interpretation of identity uniqueness, it is more expensive ($O(|\mathcal{V}|^2 \log|\mathcal{V}|)$ than SybilLimit, which costs ($O(\sqrt{|E|}|\mathcal{V}|\log|\mathcal{V}|)$ in our sparse social graph setting.

C. Spammer Belief

We now describe how we combine reporter trust, identity uniqueness and spammer reports to derive a measure of the belief that a host is spamming. We define *spammer belief* as a score in 0% to 100% that can be interpreted as the belief that a host is spamming: a host with 0% spammer belief is very unlikely to be a spammer, whereas a host with 100% spammer belief is very likely to be one.

1) Spammer Reports: A node i may have email classification functionality through online systems that interface with it using the i's `ReportSpammer()` API. In this case, i considers only the reports of those systems in calculating the belief that a host is spamming. When i receives reports by more than one systems for the same h, i's confidence $c_i(h)$ that h is a spammer is the average (possibly weighted) of these applications' reports. The repository uses this reported average confidence to compute the similarity of i's reports with the reports of its friends, which is used to derive direct trust values.

At initialization time, SocialFilter nodes consider all hosts to be legitimate. As nodes receive emails from hosts, they update their confidence (§ II-B). For efficiency, nodes send spammer report to the repository only when the difference between the previous confidence in the node being a spammer and the new confidence exceeds a predetermined threshold δ.

When the repository receives a new spammer report for h, this new report preempts an older report from the same node, which is thereafter ignored. Consequently, SocialFilter nodes are able to revoke spammer reports by updating them. Each spammer report carries a timestamp. The time interval during which a spammer report is valid is a tunable system parameter. Reports that have expired are not considered in the calculation of the belief that a host is spamming.

2) Spammer Belief Equation: The repository may receive multiple spammer reports originating from multiple nodes $j \in \mathcal{V}$ and concerning the same host h. Subsequently, the repository needs to aggregate the spammer reports to determine an overall belief `IsSpammer(h)` that h is a spammer. It derives the spammer belief by weighing the spammer reports' confidence with the reporter trust and identity uniqueness of their reporters:

$$IsSpammer(h) = \frac{\Sigma_{j \in \mathcal{V}^h}\ rt_j\ id_j\ c_j(h)}{S} Logistic(S) \quad (3)$$

In the above equation, $\mathcal{V}^h \subseteq \mathcal{V}$ is the set of nodes that have posted a spammer report for h. In addition, $S = \Sigma_{j \in \mathcal{V}^h}\ rt_j\ id_j$.

The factor $0 \leq Logistic(S) \leq 1$ discounts the belief in a host h being spammer in case the reporter trust and identity uniqueness of the nodes that sent a spammer report for h is low. It is used to differentiate between the cases in which there are only a few reports from non-highly trustworthy nodes and the cases there are sufficiently many and trustworthy reports. When S is sufficiently large, we should consider the weighted average of the confidence in the reports to better approximate the belief that a host is spammer. But when S is small we cannot use the spammer reports to derive a reliable spammer belief value. Based on these observations, we define the function $Logistic$ as the logistic (S-shaped) function of S:

$$Logistic(S) = \frac{1}{1 + e^{(b-aS)}} \quad (4)$$

a and b are small constants set to 5 in our design. For $S \leq 0.4$, $Logistic(S)$ is very small. However, when S exceeds 0.6, $Logistic(S)$ increases drastically until it becomes 0.5 for $S = 1$. For $S = 2$, $Logistic(S)$ approximates 1.

D. Repository

Practice has shown that centralized infrastructures such as web mail, OSN providers, and email reputation services can scale to millions of clients. Thus, to simplify the design and provide better consistency and availability assurances we use a centralized repository. This repository can in fact consist of a well-provisioned cluster of machines or even a datacenter.

When a node queries the repository for the spammer belief of a host, the repository is interested on the reports for a single host. These reports are sent by multiple nodes, thus for efficiency it is reasonable to index(key) the reports based on the hash of the host's IP.

IV. Evaluation

We evaluate SocialFilter's ability to block spam traffic and compare it to Ostra [27]. The goal of our evaluation is two-fold: a) to illustrate the importance of our design choices, i.e., incorporating identity uniqueness and initializing direct trust with user-defined trust; and b) to shed light on the benefits and drawbacks of our and Ostra's approach in using social links to mitigate spam. In [35], we also demonstrate the efficacy of performing trust computations at the centralized repository.

A. Ostra Primer

Before we proceed with the comparative evaluation, we briefly describe Ostra to provide insights on its operation. Ostra bounds the total amount of unwanted communication a user can send based on the number of social trust relationships the user has and the amount of communication that has been flagged as wanted by its receivers. Similar to SocialFilter, in Ostra an OSN repository maintains the social network. When a sender wishes to establish an email connection to a receiver, it first has to obtain a cryptographic token from the OSN repository. The OSN repository uses the social links connecting the admins of the sender and the receiver nodes to determine whether a token can be issued.

In particular, a node is assigned a credit balance, B, for each social link it's administrator is adjacent to. B has an initial value of 0. Ostra also maintains a per-link balance range $[L, U]$, with $L \leq 0 \leq U$, which limits the range of the users credit balance (i.e., always $L \leq B \leq U$). The balance and balance range for a user is denoted as B_L^U. For instance, the link's adjacent user's state 2_{-4}^{+5} denotes that the user's current credit balance is 2, and it can range between -4 and 5.

When a communication token is issued, Ostra requires that there is a path between the sender and the receiver in the social network. Subsequently, for each link along the social path the first adjacent nodes credit limit L is increased by one, and the second adjacent nodes credit limit U is decreased by one. This process propagates recursively from the sender to the receiver along the social links. If this process results in any of the links in the path to have adjacent nodes of which the credit balances exceed the balance range, Ostra refuses to issue the token. When the email connection is classified by the receiver, the credit limits L and U are restored to their previous state. If the connection is marked as unwanted, one credit is transferred from the balance of the first node of the link to the balance of the second one.

As a consequence of this design, the social links that connect spammers to their receivers eventually have balance beyond the allowed range, and a spammer is prevented from establishing further email connections. In addition, Ostra is Sybil-resilient because the credit available to a sender is not dependent on the number of Sybils it has. It is only dependent on the sender's social connectivity and on whether the sender's emails are classified as wanted.

In summary, Ostra uses the social network as a rate-limiting communication conduit. SocialFilter on the other hand uses the social network as a trust layer from which the trustworthiness of the spam detectors can be extracted.

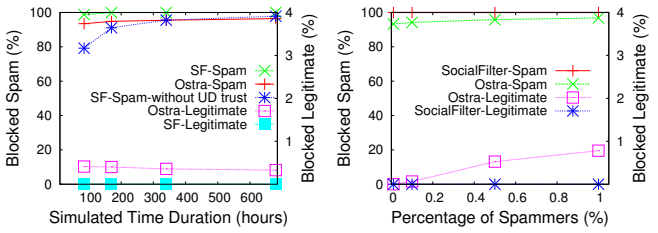
B. Simulation Settings

For a more realistic evaluation, we use a $50K$-user crawled sample of the Facebook social graph [17]. The sample is a connected component obtained from a $50M$-user sample via the "forest fire" sampling method [25], and contains $442,772$ symmetric links. The average number of friends of each user in the graph is approximately 18. The diameter of this graph is 11. The clustering coefficient is 0.178. Each user in the social network is the admin of an email relay server, which we also refer to as a SocialFilter or Ostra node. Nodes can send and receive email connections. We use the SimPy 1.9.1 [28] discrete-event simulation Python framework to simulate the operation of SocialFilter and Ostra. The source code and the OSN data set are available at [6].

We have two types of nodes: *honest* and *spammers*. Honest nodes send 3 legitimate emails per day. 80% and 13% of the legitimate emails are sent to sender's friends and sender's friends of friends respectively, and the destination of the rest 7% emails is randomly chosen by the sender. Spammers send 500 spam emails per 24h, each to random honest nodes in the network. We set Ostra's credit bounds as $L = -5$ and $U = 5$. The above settings are obtained from Ostra's evaluation [27]. Honest and spammer nodes correspond to users uniformly randomly distributed over the social network.

Several nodes can instantly classify spam connections. These instant classifiers correspond to systems that detect spam by subscribing to commercial blacklists or by employing content-based filters. On the other hand, normal nodes can classify connections only after their users read the email. That is, the normal classification can be delayed based on the behavior of the users (how frequently they check their email). 10% of honest SocialFilter nodes have the ability of instant classification and the average delay of the normal classification is 2 hours [27].

In SocialFilter, when a node classifies an SMPTP connection as spam, it issues a spammer report. The issued spammer reports are gathered and aggregated in the repository. When normal users with no capability of instant classification receive SMTP connection requests from previously unencountered hosts they query the repository. Subsequently, the repository returns to them a value that corresponds to the belief that a



(a) (b)

Figure 2. (a) percentage of blocked spam and legitimate emails connections for SocialFilter (SF) and Ostra as a function of simulated time length. The percentage of spammer nodes is 0.5%; (b) Percentage of blocked spam and legitimate email connections for SocialFilter (SF) and Ostra as a function of the percentage of spammer nodes. The simulated time duration is 340h.

host is a spammer (§III-C2). In summary, classifier nodes share their experiences by issuing spammer reports, and normal nodes use the reports to block spam from senders they had not previously encountered.

The reporter trust assigned to nodes is computed using Dijkstra's algorithm based on the pairwise direct trust value between users that are connected in the 50K-node social graph (§III-A). The pairwise direct trust values are derived using Equation 1. The direct trust between users that are friends is initialized to a random value in $[0,\ 1]$. The number of pre-trusted nodes used is 100 and we recompute the reporter trust every 24 simulated hours.

In this evaluation, we compute the similarity between reports using Equation 1 ($\alpha = 0.8$). Assume that the repository receives the k_{th} spammer report from both nodes i and node j that involves a host h to which i and j have assigned confidence $c_i(h)$ and $c_j(h)$. The repository computes the similarity v_{ij}^k as follows:

$$v_{ij}^k = \frac{min(c_i(h), c_j(h))}{max(c_i(h), c_j(h))} \tag{5}$$

The identity uniqueness of each node is computed as described in §III-B by processing the 50K-node social graph. The parameters of the computation are set as follows: $w = 15$, $r = 2000$ and $l = 100$. If the overall spammer belief computed by Equation 3 is over 0.5, a node blocks the SMTP connection.

C. Resilience to Spammers

In Figure 2(a), we depict SocialFilter's and Ostra's spam mitigation effectiveness with varying simulated time duration. We observe that SocialFilter manages to block 99% to 100% of spam after $179h$. Once the repository has obtained sufficiently trustworthy spammer reports from nodes, it can inform all other nodes about the detected spammers. In Ostra, after the percentage of blocked spam reaches only 95% at $340h$, it does not improve with the passage of time. We attribute this difference on the fact that in Ostra spam detection affects only a region of the social network: the one that is affected by the change in the credit balances of the links adjacent to the detector node. On the other hand in SocialFilter, once a sufficiently trustworthy node detects spam, its report can be used by all other nodes in the network to classify the spamming host. Importantly, we also observe that Ostra incurs a non-negligible false positive rate (blocked legitimate emails), which is equal to $\sim 0.4\%$. In contrast, SocialFilter yields no false positives.

Figure 2(b) presents the spam mitigation effectiveness of SocialFilter and Ostra under a varying number of spammers. We make two observations. The first is that as in Figure 2(a), Ostra suffers from a substantial false positive rate when the percentage of spammers is greater than 0.1%. When the percentage of spammers is 1% (500 spammers), around 0.8% of legitimate emails are blocked. We can attribute Ostra's high false positive rate to the following. In SocialFilter, a node blocks an email sender only if it has been explicitly reported as spammer. On the other hand, Ostra blocks links (the credit balance goes out of bounds) in the socials path used by a spammer, and some honest nodes cannot send email because the blocked links are included in all the social paths used by those honest nodes.

The second observation is that our proposal always blocks $\sim 99\%$ of spam as the portion of spammers varies in 0.1% to 1%. Ostra always blocks 93% to 97% of spam connections. We observe that the spam detection rate increases substantially for Ostra and slightly for SocialFilter as the number of spammers increases. This is because the increased spam induces nodes to share more information. As a result, the reporter trust graph becomes more connected, allowing the repository to consider reports from more nodes as trustworthy. In the case of Ostra, it reduces the balance on social links adjacent to spammers resulting in less spam passing through.

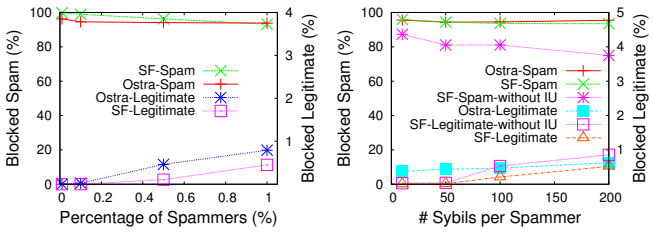
D. Importance of User-defined Trust

In this portion of our evaluation, we demonstrate the importance of using the user-defined trust value (§III-A) to initialize the direct trust between nodes.

Figure 2(a) also shows the spam mitigation capability of SocialFilter when the initial user-defined (UD) trust assigned by friends in the admin social network is 0 ("SF-Spam-without UD trust"). As can be seen, SocialFilter with direct trust initialized with user-defined trust is effective in blocking 99% of spam emails after 85h. On the other hand, when the user-defined trust is 0, it takes a lot more time (up to 340h) for SocialFilter to start effectively blocking spam.

When we do not initialize direct trust with user-defined trust, after $85h$, a SocialFilter node has on average only 0.22 reporter trust. This is because early in the simulation, nodes have encountered a small number of common spamming hosts, thus the repository cannot derive meaningful direct trust values. Consequently the reporter trust graph $T(\mathcal{V}, E)$ is disconnected, resulting in low report trust values. Consequently, the repository is unable to consider many valid spammer reports from the nodes. As time progresses, the repository can derive more meaningful direct trust values by comparing the reports of friend nodes. Therefore, we observe that after 170h our system is able to block almost 100% of spam.

This result validates our choice to tap into the user-defined trust between acquainted SocialFilter admins. This source of trust is important because it enables the initial trust graph to be sufficiently connected, prior to performing spammer report comparisons. User-defined trust also contributes in trust values converging to correct ones faster (given that the admins have assigned appropriate values), even in case common spammer reports are infrequent.



(a) (b)

Figure 3. (a) percentage of blocked spam and legitimate email connections for SocialFilter (SF) and Ostra as a function of the portion of colluding spammers; (b) percentage of blocked spam and legitimate email connections for SocialFilter (SF) and Ostra as a function of the number of Sybils created per spammer. The percentage of spammer nodes is 0.5%. Results for SocialFilter that does not employ identity uniqueness (IU) are also included. The simulated time is 340h.

E. Resilience to Colluders and Sybils

We also consider attack scenarios under which spammers collude to evade SocialFilter and Ostra, and to disrupt email communication from legitimate hosts. We assume that spammers are aware of each other, which is reasonable if the spammers belong to the same botnet. To attack SocialFilter, a spammer submits a report $\{[\text{spammer report}]\ s,\ 0\%\}$ for each of the other spammers s in the network. Also, when a spammer receives a connection from a legitimate host l, it submits $\{[\text{spammer report}]\ l,\ 100\%\}$ to induce SocialFilter to block l's emails. To attack Ostra, each spammer classifies a legitimate email and a spam email connection as unwanted and legitimate, respectively.

Figure 3(a) shows the percentage of blocked spam and legitimate email connections in SocialFilter and Ostra as a function of the portion of colluding spammers in the network. Ostra achieves almost the same effectiveness in blocking spam connections as in the absence of colluding spammers. However, the false positive rate (percentage of blocked legitimate email) increases substantially with the percentage of colluders. Ostra is more affected by false classification because it does not have any method to recognize it.

As can be seen in Figure 3(a), SocialFilter is less effective in blocking spam in the presence of false classification. In fact, when the percentage of colluding spammers reaches 1%, SocialFilter becomes less effective than Ostra. Moreover, the existence of false reporters that incriminate legitimate senders results in a non-zero false positive rate for SocialFilter, which however is substantially less than for Ostra. This is because colluding spammers have very low direct trust to other honest users as their reports are different to those honest nodes. As a result, the reporter trust for spammers is lower, resulting in their reports to be mostly ignored by honest nodes.

1) Sybil Attack: We also consider the case in which colluding spammers create Sybils. These Sybils form a cluster that is directly connected to their creator spammer node. The purpose of the Sybils is to decrease the belief of the repository in the spammer node being malicious and to increase the belief in an honest node being spammer. In addition, Sybils allow the spammer to send messages from many different sources, enabling him to further evade defenses.

At the start of the SocialFilter simulation, Sybils send positive spam reports for all other spammer nodes (including the Sybils). Honest nodes may send legitimate email to spammer nodes but not to their Sybils. When a spammer node receives legitimate email from an honest node, the spammer reports the honest user as a spammer and so do all the Sybils of the spammer. 10% of all Sybils act as spammers sending spam messages at the same rate as their creator. In the simulation for Ostra, Sybil nodes classify a legitimate email and a spam email connection as unwanted and legitimate, respectively.

Figure 3(b) shows the percentage of blocked spam and legitimate email connections as a function of the number of Sybils per spammer in the network. In SocialFilter, Sybil users gets very low identity uniqueness, which becomes even lower as the number of Sybil users per spammer increases. We can thus see that SocialFilter is resilient to the Sybil attack. In Ostra, Sybil spammers cannot send spam because the few social links that connect the creator of the Sybils with the rest of the network become blocked. We observe that when each spammer creates more than 100 Sybils, Ostra is able to block more spam than our proposal. However, Ostra still suffers from higher false positive rate.

2) Importance of Identity Uniqueness: Figure 3(b) also shows the case in which SocialFilter does not employ identity uniqueness ("SF-Spam/Legitimate-without IU"). As can be seen, attackers are very effective in manipulating the system in this case. SocialFilter without identity uniqueness cannot block a substantial percentage of spam, while it blocks a high percentage of legitimate email. This result profoundly illustrates the importance of integrating identity uniqueness in the spammer belief computation (Equation 3).

V. Related Work

Reputation Systems: SocialFilter is inspired by prior work on reputation and trust management systems [10, 21, 26]. Well-known trust and reputation management systems include the rating scheme used by the eBay on-line auction site, object reputation systems for P2P file sharing networks [22, 39] and PageRank [11]. In contrary to the above systems, our system incorporates social trust to mitigate false reporting and Sybil attacks. EigenTrust [22], PageRank [11] and TrustRank [20], provide trust values that enable a system to rank users based on their trustworthiness. However, this value cannot be explicitly interpreted as the belief in a node being honest.

IP Blacklisting: SocialFilter is similar to IP blacklisting services such as SpamHaus [8], DShield [2], CloudMark [1], and TrustedSource [5] in that it employs a centralized repository. Currently, IP blacklisting relies on a relatively small number (in the order of a few hundreds or thousands) of reporters. Reporters submit their attack logs to the centralized repositories, and the repository synthesizes blacklists based on the attack logs. SocialFilter differs in that it automates the process of evaluating the trustworthiness of the reports. Thus it does not incur the management overhead of traditional IP blacklisting services, and can scale to millions of reporters.

Predictive blacklisting [36, 43] improves upon IP blacklisting by creating a customized blacklist for each reporter that is shorter and more likely to be relevant. However, it does not address adversarial behavior by reporters, i.e., false reporting and Sybil attacks. In addition, because it does not employ user-defined social trust, a node is able to obtain a customized ranking only if the node itself has classification functionality.

Collaborative Email Reputation Systems: Prior work also includes proposals for collaborative spam filtering [3, 9, 44, 45]. Kong et al. [24] also consider untrustworthy reporters, using Eigentrust to derive their reputation. These solutions only classify the contents of emails and not the source of spam. This requires email servers to waste resources on email reception and filtering. SocialFilter can assign trust metrics to sources, thereby rejecting unwanted email traffic on the outset.

Similar to SocialFilter, RepuScore [33] is a collaborative reputation management system that allows participating organizations to establish email sender accountability on the basis of past actions. It provides a global reputation value for IP addresses and email domains. Repuscore assigns the same global reputation value for both email senders and reporters of spam. To calculate a global reputation for a sender or reporter, it weighs the local reputation values submitted by the reporters and concerning the sender or the reporter, by the global reputation of the reporters. Unlike SocialFilter, it does not employ sybil-resilient and transitive trust inference, which results in the trust values being susceptible to manipulation. In particular, an email server's reported local reputations are considered reliable unless the email server itself sends spam and becomes detected by other reporters.

VI. Conclusion

We introduced the first collaborative spam mitigation system that assesses the trustworthiness of spam reporters by both auditing their reports and by leveraging the social network of the reporters' administrators. SocialFilter weighs the spam reports according to the trustworthiness of their submitters to derive the a measure of the belief that a host is spamming.

The design and evaluation of SocialFilter illustrates that: a) we can improve the reliability and the attack-resilience of collaborative spam mitigation by introducing Sybil-resilient OSN-based trust inference mechanisms; b) using social links to obtain the trustworthiness of spammer reports can result in comparable spam-blocking effectiveness with approaches that use social links to rate-limit spam (e.g., Ostra [27]); c) unlike Ostra, SocialFilter yields no false positives in the absence of reports that incriminate benign email senders.

VII. Acknowledgements

This work was funded in part by the NSF awards CNS-0845858, CNS-1017858 and NS-0925472. We thank Jeffrey Chase, Landon Cox, Michael Reiter, and the anonymous reviewers, for their helpful feedback and suggestions.

References

[1] Cloudmark. www.cloudmark.com/en/home.html.
[2] Cooperative Network Security Community. http://www.dshield.org/.
[3] Distributed Checksum Clearinghouses Reputations. www.rhyolite.com/dcc/reputations.html.
[4] Facebook connect. developers.facebook.com/connect.php.
[5] Secure Computing, TrustedSource. www.securecomputing.com/index.cfm?skey=1671.
[6] SocialFilter Source Code. www.cs.duke.edu/nds/wiki/socialfilter.
[7] Spammers Moving to Disposable Domains. http://threatpost.com/en_us/blogs/spammers-moving-disposable-domains-071410.
[8] The SpamHaus Project. www.spamhaus.org/.
[9] J. Attenberg, K. Weinberger, A. Dasgupta, A. Smola, and M. Zinkevich. Collaborative Email-Spam Filtering with the Hashing Trick. In *CEAS*, 2009.
[10] M. Blaze, J. Feigenbaum, and J. Jack Lacy. Decentralized Trust Management. In *IEEE S&P*, 1996.
[11] S. Brin and L. Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. In *Computer Networks and ISDN Systems*, 1998.
[12] A. Cheng and E. Friedman. Sybilproof Reputation Mechanisms. In *P2PEcon*, 2005.
[13] A. Cheng and E. Friedman. Manipulability of PageRank under Sybil Strategies. In *NetEcon*, 2006.
[14] G. Danezis and P. Mittal. SybilInfer: Detecting Sybil Nodes using Social Networks. In *NDSS*, 2009.
[15] J. R. Douceur. The Sybil Attack. In *IPTPS*, March 2002.
[16] M. Erdos and S. Cantor. Shibboleth Architecture DRAFT v05. *Internet2/MACE, May*, 2, 2002.
[17] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. A Walk in Facebook: Uniform Sampling of Users in Online Social Networks. In *IEEE INFOCOM*, 2010.
[18] E. Gray, J.-M. Seigneur, Y. Chen, and C. Jensen. Trust Propagation in Small Worlds. In *LNCS*, pages 239–254. Springer, 2003.
[19] R. K. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of Trust and Distrust. In *WWW*, 2004.
[20] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating Web Spam with TrustRank. In *VLDB*, 2004.
[21] K. Hoffman, D. Zage, and C. Nita-Rotaru. A Survey of Attack and Defense Techniques for Reputation Systems. In *ACM Computing Surveys*, 2008.
[22] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *WWW*, 2003.
[23] S. Katti, B. Krishnamurthy, and D. Katabi. Collaborating Against Common Enemies. In *ACM IMC*, 2005.
[24] J. S. Kong, B. A. Rezaei, N. Sarshar, V. P. Roychowdhury, and P. O. Boykin. Collaborative Spam Filtering Using e-mail Networks. In *IEEE Computer*, 2006.
[25] J. Leskovec and C. Faloutsos. Sampling from Large Graphs. In *SIGKDD*, 2006.
[26] S. Marti and H. Garcia-Molina. Taxonomy of Trust: Categorizing P2P Reputation Systems. In *Computer Networks*, 2006.
[27] A. Mislove, A. Post, P. Druschel, and K. P. Gummadi. Ostra: Leveraging Social Networks to Thwart Unwanted Traffic. In *NSDI*, 2008.
[28] K. Mller and T. Vignaux. SimPy (Simulation in Python). simpy.sourceforge.net/.
[29] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Computer Networks*, 1999.
[30] A. Ramachandran and N. Feamster. Understanding the Network-level Behavior of Spammers. In *ACM SIGCOMM*, 2006.
[31] P. Resnick, , R. Zeckhauserand, E. Friedman, and K. Kuwabara. Reputation Systems. In *Communications of the ACM*, 2000.
[32] G. Singaraju and B. B. Kang. RepuScore: Collaborative Reputation Management Framework for Email Infrastructure. In *USENIX LISA*, 2007.
[33] G. Singaraju, J. Moss, and B. B. Kang. Tracking Email Reputation for Authenticated Sender Identities. In *CEAS*, 2008.
[34] M. Sirivianos, K. Kim, and X. Yang. SocialFilter: Introducing Social Trust to Collaborative Spam Mitigation. In *USENIX CollSec*, 2010.
[35] M. Sirivianos, X. Yang, and K. Kim. SocialFilter: Introducing Social Trust to Collaborative Spam Mitigation. www.cs.duke.edu/~msirivia/publications/socialfilter-tech-report.pdf, 2010.
[36] F. Soldo, A. Le, and A. Markopoulou. Predictive Blacklisting as an Implicit Recommendation System. In *IEEE INFOCOM*, 2010.
[37] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *USENIX Winter*, 1988.
[38] D. N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-Resilient Online Content Rating. In *NSDI*, 2009.
[39] K. Walsh and E. G. Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. In *NSDI*, 2006.
[40] D. J. Watts and S. H. Strogatz. Collective Dynamics of Small-World Networks. In *Nature*, 1998.
[41] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipko. Spamming Botnets: Signatures and Characteristics. In *ACM SIGCOMM*, 2008.
[42] H. Yu, P. Gibbons, M. Kaminsky, and F. Xiao. A Near-Optimal Social Network Defense Against Sybil Attacks. In *IEEE S&P*, 2008.
[43] J. Zhang, P. Porras, and J. Ullrich. Highly Predictive Blacklisting. In *USENIX Security*, 2008.
[44] Z. Zhong, L. Ramaswamy, and K. Li. ALPACAS: A Large-scale Privacy-Aware Collaborative Anti-spam System. In *IEEE INFOCOM*, 2008.
[45] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A. D. Joseph, and J. Kubiatowicz. Approximate Object Location and Spam Filtering on Peer-to-Peer Systems. In *ACM/IFIP/USENIX Middleware*, 2003.