Defending against large-scale crawls in online social networks

Mainack Mondal *MPI-SWS* · Bimal Viswanath *MPI-SWS* · Allen Clement *MPI-SWS* · Peter Druschel *MPI-SWS*

Krishna Gummadi *MPI-SWS* · Alan Mislove *Northeastern University* · Ansley Post *Google*

Technical Report MPI-SWS-2011-006

Abstract

Thwarting large-scale crawls of online social networks (OSNs) like Facebook and Renren is in the interest of both the users and the operators of these sites. OSN users wish to maintain control over their personal information, and OSN operators wish to protect their business assets and reputation. Traditional defenses against crawlers involve rate-limiting the browsing activity of individual users. However, these schemes are ineffective against crawlers with many accounts, be they fake accounts (Sybils) created by a crawler or compromised accounts of real users obtained on the black market.

We propose Genie, a system that can be deployed by OSN operators to defend against crawlers. Genie's design is based on the observation that the browsing patterns of normal users and crawlers are very different: most normal users limit their profile views to a small number of other users that are in their close network neighborhood, while even a crawler with many accounts needs to view profiles of users that are relatively more distant from the closest user account he controls. Genie exploits this fact by limiting the rate of profile views based on the connectivity and distance between a profile viewer and viewee in the social network. An experimental evaluation using real-world data gathered from a popular OSN shows that Genie frustrates large-scale crawling. Most browsing by ordinary users is not affected; the few users who are affected can recover easily by adding a few friend links.

1 Introduction

Online social networking sites (OSNs) like Facebook, MySpace, and Orkut have the personal data of hundreds of millions of users. OSNs allow users to browse the (public) profiles of other users in the network, making it easy for users to connect, communicate, and share content. Unfortunately, this functionality can be exploited by third-parties to aggregate and extract data about millions of OSN users. Once collected, the data can be republished [1] and mined in ways that may violate users' privacy. For instance, it has been shown that private user attributes like sexual orientation can be inferred from a user's set of friends and their profile attributes [1, 15]; a third party with access to aggregated user data could easily apply these techniques.

These third-party aggregators, which we refer to as *crawlers*, represent a significant problem for OSN site operators as well. User data provides the OSN operators with a revenue stream, for example, via targeted advertisement. OSN operators wish to protect this asset by stopping crawlers. Moreover, OSN operators cannot ensure that data collected by a third party is used according to the operator's privacy policy. Yet, the OSN operator is likely to be held responsible if the data is used in ways that violate the policy, at least in the court of public opinion. For example, Facebook was widely blamed in the popular press [3] when a single crawler gathered public profiles of over 100 million users [1]. As a result, OSN operators need effective mechanisms to thwart large-scale crawling of OSN sites.

Today, OSN operators use rate-limiting techniques to restrict a crawler's ability to collect data on a large fraction of the network. These techniques limit the rate at which a single user account or IP address can view user profiles [18]. Unfortunately, crawlers can circumvent these schemes by creating a large number of fake user accounts, by employing botnets to gain access to many IP addresses, or by using compromised accounts of a large number of real users.

In this paper, we propose Genie, a system that OSN operators can deploy to limit crawlers. Genie leverages the differences in the browsing patterns of normal users and crawlers to effectively thwart large-scale crawls of the social network. Genie's design is based on the insight that legitimate profile views tend to occur between users who are well connected and close to each other in the social network. A crawler, on the other hand, is limited in his ability to form or control enough links to be close to all users whose profiles he wishes to retrieve. Genie exploits this fact by enforcing rate limits in a way that is sensitive to the distance and degree of connectivity between viewer and viewee.

Using profile view data from Renren, a Facebook-like OSN that is popular in China [17], we observe that the distance between users who legitimately view each other's information tends to be low; the average distance between the viewer and viewee in the social network is 1.62. A crawler, on the other hand, would require a very large number of well-distributed accounts to be able to crawl the entire network with such short path lengths. Moreover, forming a social link requires the approval of both users in most OSNs. Thus, it is difficult for an attacker to acquire a sufficient number of them [12]. In fact, even the purchase of a relatively large number of compromised accounts on the black market would not give the attacker access to a sufficient number of links to quickly crawl the network.

Genie works by deriving a credit network [6, 7, 10] from the social network. In brief, credit is associated with links in the social network, and a viewer must "pay" credits to the viewee, along a path in the social network, when viewing a profile. Compared to conventional per-account or per-IP address rate-limiting, credit networks offers two key advantages. First, in a credit network, the rate limits are associated with social network links rather than user accounts or addresses. As a result, the attacker gains little by creating many (Sybil) accounts or using many IP addresses [11]. Second, the greater the distance between viewer and viewee in the social network, the stricter is the rate limit imposed by the credit network on profile views. Consequently, even attackers with access to a relatively large number of compromised user accounts are unable to crawl the network in a short period of time.

The contributions of this work are as follows:

- We analyze profile viewing data from the Renren social network and show that there exist significant differences between normal user and crawler workloads. Specifically, we show that even for a powerful crawler, the average distance between viewer and viewee in the social network is significantly larger than for legitimate profile views.

- We present the design of Genie, which leverages credit networks derived from the already-existing social network to block large-scale crawling activity, while allowing legitimate browsing unhindered.

- We demonstrate the feasibility of deploying Genie with an evaluation using large but partial social network graphs obtained from Renren, Facebook, YouTube, and Flickr, and a mix of real and synthetically generated profile viewing traces. We demonstrate that Genie effectively blocks crawlers while the impact on legitimate users is minimal.

2 Related work

In this section, we describe relevant prior work on limiting large-scale crawls of OSNs and leveraging social networks to defend against Sybil attacks.

2.1 Limiting large-scale crawls

There exists a limited literature on techniques used for limiting crawls of web services. Two techniques used in practice are *robots.txt* [4], and IP- or account-based rate limiting [2].

Robots.txt is a file stored at the web server which indicates a set of pages on that server that should not be crawled. Compliance with this list is voluntary and robots.txt consequently provides no defense against crawlers intent on scraping the entire network.

Large web sites like Yahoo! often rely on a simple per-IP rate limit to control access to their web services [2]. Each IP is allocated a maximum number of requests which are replenished in 24 hour intervals. This approach limits the work an individual user can impose on the service, but does not defend against botnets that control many IPs.

Online social networks like Facebook [18] often use account-based rate limits on requests to view profile pages. Similar to IP-based rate limits, this approach works well if users are assumed to only possess a single account; in the face of Sybils or compromised accounts, it provides much weaker guarantees.

Wilson et al. proposed SpikeStrip [20], a system designed to discourage OSN crawlers. SpikeStrip uses cryptography to make information aggregation from OSN websites inefficient. SpikeStrip rate limits the number of allowed profile views per browsing session and prevents different browsing sessions from sharing data. Thus, crawlers cannot aggregate or correlate data gathered by different sessions.

Despite its elegant design, SpikeStrip affects the usability of the OSN. For example, SpikeStrip does not allow two OSN users to share website links of a common friend. Moreover, SpikeStrip would require OSNs like Facebook to change the way they use CDN services like Akamai to serve users' content. Unlike SpikeStrip, Genie does not affect the usage of the OSNs. As we will show later, Genie can be deployed with minimal disruption to the browsing activities of normal users.

2.2 Social network-based Sybil defenses

Recently there have been proposals to leverage social networks to defend against Sybil attackers sending spam in communication systems (Ostra [14]) and cheating in online marketplaces (Bazaar [16]). Genie's design borrows some ideas from these Sybil defense works, but it differs from them in two fundamental ways. First, we target the general problem of preventing aggregation of publicly available data in a social network and address challenges associated with both Sybil and stolen accounts, while previous works have focused exclusively on Sybil accounts. Second, unlike Ostra and Bazaar which can rely on interacting users to provide the crucial feedback on whether a communication is spam or whether a transaction is fraudulent, Genie cannot expect users to provide feedback on whether a profile request received is from a crawler. Thus, unlike Ostra and Bazaar, Genie is tasked with the challenge of differentiating profile views from crawlers and normal users. A substantial novelty in Genie's design lies in the way it exploits the differences in the browsing patterns of crawlers and normal users to limit the former while leaving the latter unaffected.

3 System and attack model

System model Online social networking and sharing sites such as Facebook, Renren, Google+, and Orkut share a common system model. Users create personal accounts, establish friend links with other users, and post information (which is often of a personal nature). The graph formed by the entire set of friend links forms a social network. In the OSNs of interest to us, forming a friend link requires the consent of both users.

User can chose to make their data *private* (i.e., visible only to the user and the site operator), *public* (i.e., visible to every user of the social network), or *semi-public* (i.e., visible to subsets of friends or friends of friends). In practice, many users chose to make their profile information public, despite the private nature of some of the information posted. Contributing to this choice may be that sites encourage public sharing, that the default privacy setting is ”public”, that other privacy choices are not always intuitive, and that many users are not fully aware of the privacy risks.

In this paper, we are not concerned with the reasons why users share their information publicly. Instead, we are concerned with users who seek to aggregate and create external copies of the public profile information of as many other users as they can.

Attack model We are concerned with *crawling* attacks where a user (or group of users) illicitly visits many (if not all) of the user profiles in the network in order to gather the publicly available profile information. We assume that the crawler is agnostic to *which* users he crawls, and is trying to crawl as many user profiles as possible.

Today, social networking sites tend to impose a rate limit on the user profile page views a single user can request, in order to slow down crawlers. However, there are two ways in which a crawler can overcome these limits.

An attacker can mount a *Sybil attack* by creating multiple user accounts, thereby overcoming the per-user rate limit. It is important to note that while the attacker can form an arbitrary number of friend links among his Sybil accounts, his ability to form links between his Sybil accounts and real users is limited by his ability to convince real users to accept a friend link, regardless of how many user accounts he controls. The significance of this point will become clear in the following section, where we describe how Genie leverages social links to limit crawling activity.

An attacker can also mount a *compromised account attack* by taking control (e.g. stealing the passwords) of existing accounts in the system. The attacker can gain access to such accounts, for instance, by first using a phishing attack, or by purchasing the credentials of already compromised accounts on the black market [12]. This attack is more powerful than a Sybil attack, because every additional compromised account increases the number of links to real users that the attacker has access to. Again, the significance of having access to such links will become clear in the following section.

We assume that an attacker with access to compromised accounts cannot compromise the accounts of strategically positioned users of his choosing. Instead, compromised accounts are randomly distributed throughout the network. Additionally, we assume that the attacker does not actively form new links involving compromised accounts. The reason is that such activity would likely alert the legitimate owner of the account, who might then alert the site provider and cause the attacker to lose access to the account.

4 Normal vs. crawler workload analysis

In this section, we compare profile viewing workloads of normal users and crawlers. Our analysis reveals substantial differences between the two workloads. In particular, we show that normal users mostly visit the profiles of a small number of 1-hop or 2-hop neighbors in the network, while even the most strategic crawlers have to fetch a large number of profiles outside of their immediate (1-hop or 2-hop) network neighborhood to accomplish their goal. In later sections, we show how Genie can exploit the differences in the browsing behavior of normal users and crawlers to rate-limit crawlers, while leaving normal users unaffected.

4.1 Normal users' profile viewing behavior

We used data gathered from the Renren social network [17], which is a Facebook-like social network popular in China. We concentrate on the largest connected component of the Renren's Peking University's network. We refer to the data set as the Renren Peking University or simply RR-PKU [11] data set. The largest connected component of RR-PKU network graph has 33,294 nodes and 705,262 undirected edges [11]. The data set also includes a trace of profile visits made by users to other users (both friends or non-friends) within the network. We analyze a trace containing 96,844 profile views covering two weeks of browsing by these users in September 2009.[1] We highlight our key findings below.

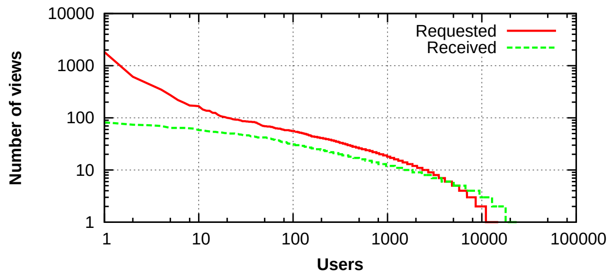


Figure 1: **The number of profile view requests made and received by RR-PKU users over two weeks.**

1. Most users make/receive few profile requests, but a small number of users make/receive a large number of requests: Figure 1 shows the distribution of profile requests made and received by individual users in the RR-PKU network. The plots show a considerable skew in the request distributions. Most ($>$ 90%) of users make or receive fewer than 10 requests, while a handful of users ($<$ 140) make 50 or more requests. Thus, most users in the social network tend to interact with only a small number of other users in the network. In later sections, we leverage this social behavior of users to differentiate them from crawlers, who necessarily have to fetch profiles from a large number of users in the network.

2. The number of profile requests made or received by users is strongly correlated with their number of friends: The Pearson correlation coefficients between the rankings of users ordered based on the number of requests they make (or received) and their node degree

[1]The Renren trace does not provide timestamps or an ordering for individual profile views. Therefore, we generated a time series by assigning each profile view a timestamp chosen uniform randomly within the two-week period covered by the trace. This time series was used in all analyses conducted in the paper.

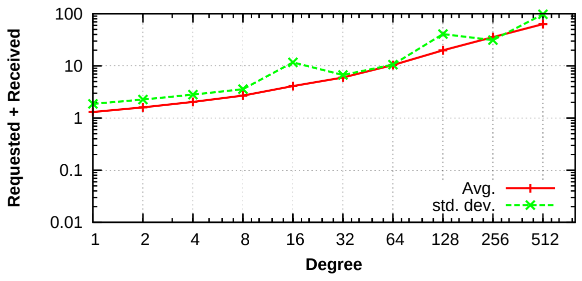


Figure 2: RR-PKU **users are divided into exponential degree bins. We show the average number of activities and standard deviation for each degree bin.**

is 0.67 (0.5). The high correlation coefficient affirms the intuitive hypothesis that the more active nodes in the network would also have more friends in the network and vice-versa. However, nodes degrees and activity are not always in exact proportion. Figure 2 illustrates this by plotting the average and standard deviation of activity (# of requests made or received) of nodes grouped into bins based on their degree. The plots show that while the average activity shows a clear upward trend as node degrees increase, the standard deviation of nodes' activity also increases, suggesting that nodes of a certain degree do not always exhibit the same amount of activity. In later sections, we leverage these observations to generate synthetic workload traces.

Another implication of the above observations is that the imbalance in user activity, measured as the magnitude of the difference between the number of profile requests made and received by individual users, per friend link would be tightly bounded. Intuitively, this can be explained as follows: both the number of profile requests made and received by a user tend to rise or fall with the user's degree. So, the difference in requests made and received divided by the user's degree would be expected to be small. Figure 3 confirms this by plotting the number of *unbalanced views* (i.e., $|Req\_made - Req\_received|$) per friend link for different users. Almost all users have the imbalance in viewing activity per friend link lower than 10. In later sections, we describe how Genie crucially leverages this observation to limit crawlers, while allowing normal workload unaffected.

3. Not all profiles requests are unique; a small but non-trivial fraction of requests are repeated: Users tend to repeatedly visit the profiles of some of their friends to track updates to their profiles. In our two-week trace, we found 17,307 (17.8%) out of the 96,844 profile requests to be such repeat requests. Our estimate of repeat views is likely be conservative, as we are restricted to a two-week trace. One would expect the percent of repeat views to increase with the length of the workload trace as users return to profiles for updates. How-

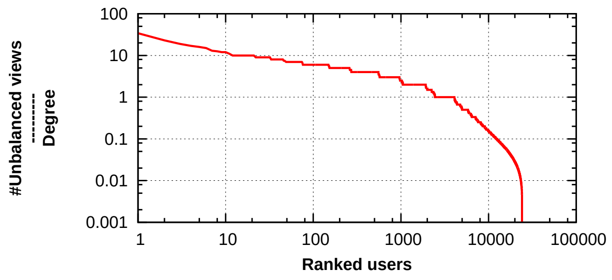


Figure 3: **The number of unbalanced views per link for RR-PKU users over two weeks.**

ever, the take-away here is that repeat requests further decrease the number of distinct profiles fetched by individual users. On the other hand, repeat views represent sub-optimal use of crawler resources, whose goal is to fetch profiles of as many distinct users as possible. Later we present a system design that takes advantage of this difference between the workloads.

4. Users mostly request profiles of others who are within their immediate (1- or 2-hop) network neighborhood: Figure 4(a) shows the distribution of network distance, measured in terms of hops, between interacting RR-PKU users. We observe that over 80% of all profile requests are between users who are separated by two hops or less. Figure 4(b) shows the distribution of network distance for profile requests made by individual users. For most users, only a small fraction of profiles they request, as well as a small fraction of users who request their profiles, lie beyond their 1-hop (direct friends) and 2-hop (friends-of-friends) neighborhood. As we will show later, locality in user interaction is yet another feature that distinguishes normal workloads from crawlers'; Genie leverages this fact to thwart crawlers.

4.2 Crawlers' profile viewing behavior

Crawlers attempt to fetch the profiles of all nodes in the network using nodes that are under their control. So crawler workloads are closely tied to the structure of the social network as well as the embedding of the nodes controlled by the crawler within the network. In this section, we study crawler workloads generated over five different social network graphs with varying numbers of nodes (and links) under the control of the crawler.

Table 1 shows the number of nodes, edges and average degree per node for network graphs gathered from five popular social networking sites namely, Renren, Facebook, YouTube, and Flickr [8, 9, 17, 21]. These networks exhibit a wide range of nodes (33,000 to 1.6 million), links (700,000 to 15.4 million), and average node degrees (5.2 to 19). We simulated crawlers with varying degrees of attack power by allowing crawlers to control

Network	Nodes	Links	Avg. deg.
RR-PKU	33 K	700 K	21.18
Facebook	63 K	816 K	25.7
Youtube	1.1 M	2.9 M	5.2
Flickr	1.6 M	15.4 M	19.0

Table 1: **Number of nodes, links, and average user degree in the large-scale social networks used to evaluate Genie.**

Network	1	10	100	1000
RR-PKU	26	415	3,638	14,938
Facebook	13	237	2,123	15,970
Youtube	6	26	592	4,129
Flickr	5	613	2,242	16,015

Table 2: **Strength of attackers in different social graphs. Each column corresponds to specific number of random compromised accounts, and the corresponding number of attack links in different graphs.**

1, 10, 100, and 1,000 randomly chosen nodes within the network. Table 2 shows the number of links that connect crawlers to the rest of the nodes in the different graphs. Note that while a crawler with 1,000 compromised accounts might not seem as a resourceful attacker, for the sizes of networks that we are considering, the crawler does represent a very powerful attacker. For example, the crawler controlling 1,000 accounts controls around 0.1% of all nodes in the YouTube network. For a point of comparison, this would be equivalent to controlling 800,000 accounts in the real-world Facebook network.

To generate the crawling workload, we divided the task of crawling user profiles across the set of nodes controlled by the attacker as follows: for each user profile, we assigned the task of fetching it to the crawler's node closest to the user. Such assignment generated crawler workloads with best possible network locality in profile fetches. Below we contrast the properties of the resulting crawler workloads with those of normal users.

In contrast to normal workloads, profile fetches by crawlers are highly non-local: Figures 5 (a) and (b) show the locality in profile visits by crawlers with different attacking capacities across the different network graphs. For large network graphs like Flickr or YouTube (not shown, but results are similar to Flickr), we observe that even a powerful crawler with 1,000 accounts has only a small fraction (less than 30%) of requested profiles within the 2-hop neighborhood of the nodes under its control. For small network graphs like Facebook and Renren (not shown, but similar to Facebook), the powerful crawler does have a majority of network nodes within 2-hop neighborhood. However, in these networks, 1,000 compromised accounts represents 3% of all nodes (and 2% of all links), and is equivalent to compromising 15

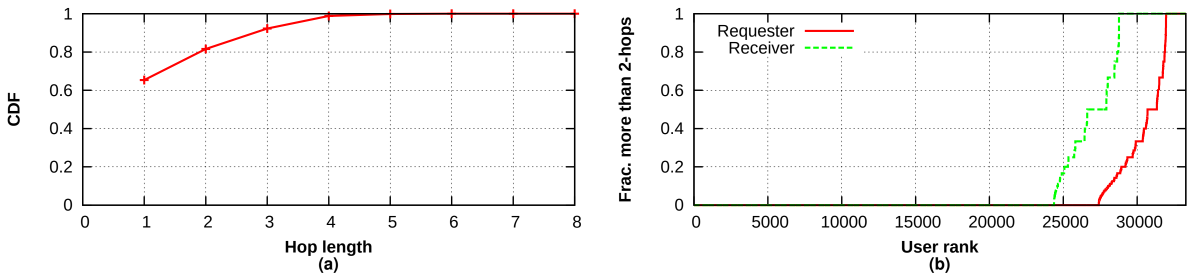
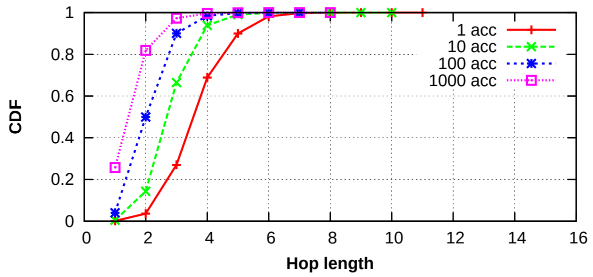


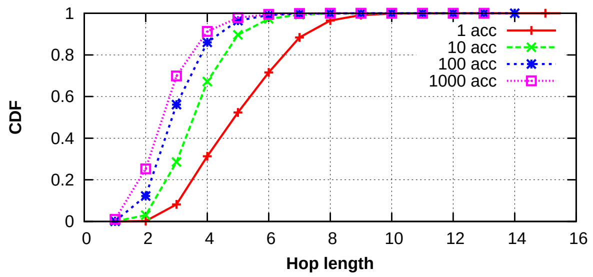
Figure 4: **(a) CDF of hop distance separating viewers and viewees in RR-PKU network. The activities are highly local. (b)Fraction of requested and received views that lie beyond 2 hop-distances for individual users. For most users, only a small fraction of their views requested or received lie beyond 2-hop neighborhood.**

million nodes in the real-world Facebook network.

Overall, the results indicate that to mimic the high locality in profile views for normal users, crawlers would have to control a non-trivial fraction of all nodes and links in the network.



(a) **Facebook**



(b) **Flickr**

Figure 5: **The variation in CDF of the hop distribution generated by attackers in different social graph using different number of compromised accounts. To fully mimic the high locality in normal user views the attackers have to control more than** 1000 **compromised good user accounts.**

In contrast to normal users, nodes under control of crawlers request many more profile requests they receive: Figure 6 illustrates another aspect of the workload in which crawlers differ from normal users. It shows the imbalance in profile views requested and received by nodes controlled by the crawlers for each link that is incident on the nodes. The observed imbalance for each node is a couple of orders of magnitude higher than that observed for normal users (see Figure 3). In the next section, we present a system design that exploits these observed differences in browsing patterns of normal users and crawlers.

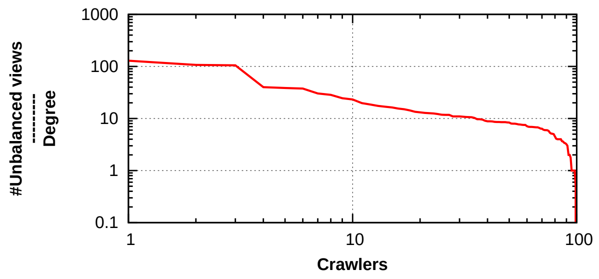


Figure 6: **The number of unbalanced views per link for crawlers in RR-PKU.**

5 Genie Design

In this section, we present the design of Genie, analyze its security properties, and discuss its vulnerabilities.

5.1 Design overview

Genie relies on a variant of a credit network [7, 10]. A credit network is a directed graph $G = (V, E)$, where each edge $(u, v) \in E$ is labeled with a scalar credit value $c_{uv} \geq 0$. Genie's credit network graph is isomorphic with the OSN graph. That is, each node in the credit network corresponds to a OSN user and there is a pair of directed edges $(u, v), (v, u)$ in the credit network iff the users u, v are friends in the OSN.

Genie allows user s to view user t's profile information iff the max-flow between s and t in the credit network is at least $f(d_{st})$, where f is a non-decreasing cost function, and d_{st} is the length of the shortest path between s and t in the credit network. That is, there exists a set of edge-disjoint paths in the credit network from s to t such that (a) every edge along a given path pi has at least c_{p_i} credits available, and (b) $\sum_i c_{p_i} = f(d_{st})$. If the view is allowed, then the credit value on each edge (u, v) on each path p_i is reduced by c_{p_i}, and the credit value on each edge (v, u) in the reverse direction of the same path is increased by c_{p_i}.

The net effect of this transaction on the credit network is that the viewer has $f(d_{st})$ fewer credits available on his adjacent links, while the viewee has $f(d_{st})$ more credits available on his adjacent links. For intermediate nodes on the paths, the total amount of available credit does not change, though the distribution of credit among their adjacent links does. Whether this distribution matters to a node depends on how well the credit network is connected. If a node is well connected and can reach any other node through any of its adjacent links, then the distribution is not likely to matter [6]; if it is not, a legitimate node may be unable to view a profile because of the distribution of credit (liquidity).

It is commonly assumed that social networks have a very high degree of connectivity, thus ensuring good liquidity in the credit network. However, liquidity may be an issue for new users, inactitive users, or small fringe communities who have not (yet) established strong social connections to the rest of the network. We will consider this issue in Section 6.

We now describe how Genie leverages the characteristics of normal profile viewing activity observed in the previous section, in order to thwart large-scale crawlers.

5.1.1 Leveraging unbalanced view ratios

As noted in the previous section, crawlers inherently request many more profile views than they receive, while normal users tend to have a more balanced ratio of profile requests to profile views. Genie leverages this observation by allowing a node to use credits obtained by being viewed to view the profiles of other users. This means that the number of profile views a user can make depends on how often her own profile is viewed. This choice causes crawlers to become more likely to run out of credit, as their viewing is inherently unbalanced.

However, the number of views a normal user receives may not exactly match the number of views they wish to legitimately issue. To allow for such imbalances, the credits on each pair of links between two nodes are rebalanced at a fixed rate. Specifically, the credit values on each pair of links $(u, v), (v, u)$ are initially set to a positive value i. Any difference $c_{uv} - c_{vu}$ is rebalanced at a fixed rate r_b, i.e., at each time step

$c_{uv} \leftarrow c_{uv} - \frac{r_b}{2}(c_{uv} - c_{vu})$, and
$c_{vu} \leftarrow c_{vu} + \frac{r_b}{2}(c_{uv} - c_{vu})$, where $0 < r_b \leq 1$.

5.1.2 Leveraging different path lengths

As the previous section noted, crawlers who aim to crawl a significant fraction of the social network must crawl users who are further away from themselves than a non-crawling user would typically crawl. Genie leverages this to discriminate against crawlers by charging more credits to view users further away.

This approach is implemented though the function f. The cost of a view, in terms of the number of credits charged, increases with the distance between viewer and viewee in the OSN, i.e. $f(d_{st}) = d_{st} - 1$. This creates a bias against crawlers, whose views tend to cross longer distances then those of normal users.

Genie must be configured with the initial credit value i and the rebalancing rate r_b. We will describe how to determine these parameters in Section 6.

5.1.3 Leveraging repeated views

Another way in which crawlers differ from legitimate viewers is that the latter tend to repeatedly view the profiles of users they are interested in, as shown in Section 4. A crawler, on the other hand, would not view the profile of a given user more than once, or at most very infrequently (e.g., when re-crawling the social network to see what has changed). Genie takes advantage of this fact by not charging for repeated profile views within a given time period.

Therefore, the cost function f is refined to evaluate to zero if user s views the profile of a user t within T days of when s was last charged for viewing t, and $d_{st} - 1$ otherwise. A typical value for T would be on the order of months. As a result, legitimate repeat views within T are free, while a crawler would be charged for each profile view.

5.2 Security properties

Genie's credit network effectively rate-limits profile views by users in a way that is sensitive to the connectivity and distance between viewer and viewee in the OSN. This form of rate limiting discriminates against crawlers by leveraging the crawler's inherent connectivity limitations within the OSN.

Let $C \in V$ be the set of user accounts controlled by the attacker and $N = V - C$ be the set of user accounts not controlled by the attacker. The crawler's goal is to view the profiles of all users in N as quickly as possible (he can trivially obtain the profiles of users in C).

We call an edge (c, n) in the social network an *attack edge* if $c \in C$ and $n \in N$. The edge cut separating C and N (i.e., the set of attack edges) is called the *attack cut*.

To determine the rate r_c at which a crawler can view profiles in N, we need only consider the attack cut, because all of the crawler's views have to cross this cut. Profile views within N or within C are irrelevant, because they must cross the attack cut an even number of times, and thus won't change the credit available along the cut.

The rate r_c is determined by the following factors:

- A, the size of the attack cut (number of attack edges): A powerful attacker has a large attack cut. (Because he is able to form many social links with users in N, or because he acquired many compromised accounts and the associated social links.)

- d_c, the average OSN distance between nodes in N and the corresponding closest node in C: Per our threat model, even a powerful attacker faces diminishing returns when trying to add more attack edges to reduce the average distance.

- r_c, the expected rate of profile views received by nodes in C from nodes in N: Per our threat model, the attacker has little control over this rate, and we can conservatively assume that it is the same as the expected rate of views received by a node in N.

- r_b, the rebalancing rate: Determined by the OSN operator.

- f, the view cost function. Determined by the OSN operator.

Using $f(d) = d - 1$, the maximal steady-state crawling rate

$$r_c = A\frac{r_b + r_c(d_n - 1)}{d_c - 1}$$

where d_n is the average distance in the OSN for legitimate profile views. The numerator is the attacker's "income", the rate at which he can acquire credits. The denominator is the attacker's cost, in credits, per profile view. As we can see, the maximal crawling rate increases linearly with the number of attack edges, at a slope defined by the second term. A larger r_b, r_c or d_n increases, a larger d_c decreases the slope.

It is important to note that the only way for the attacker to increase the crawling rate is to obtain more attack edges. Attack edges are relatively hard to obtain in large quantities. Obtaining an attack edge requires forming a social link with a user not already controlled by the attacker, or compromising a user account that has social links with users not already controlled by the attacker. Creating more user accounts by itself is ineffective, because it does not yield new attack edges. As a result, the credit network renders Sybil attacks as such ineffective.

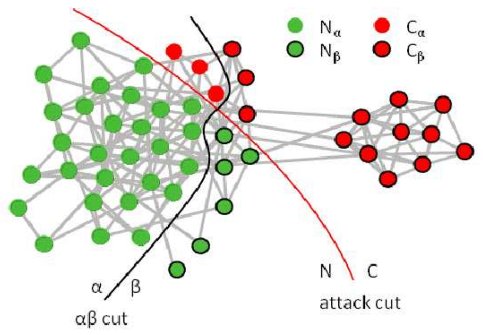


Figure 7: **Illustration of attack and $\alpha\beta$ cuts.**

Additionally, the credit network effectively makes the power of a corrupted account attack proportional to the acquired attack edges.

5.3 Potential for denial-of-service attacks

A key concern with any credit-based network is a credit exhaustion attack, where an attacker seeks to prevent legitimate transactions among innocent users. In the case of Genie, there are two questions to consider. First, can an attacker's attempt to crawl the network prevent good users from viewing each other's profiles? Second, can an attacker target specific users and prevent them from viewing other users' profiles, or from having their own profile viewed by other users?

Due to the properties of the credit network, the severity of the attack (i.e., the rate of failed profile views the attacker can cause) is identical in each case. However, in a targeted attack, the attacker can choose to selectively inflict pain on a subset of users. In the following, we consider both types of attacks simultaneously.

First, we note that the cost function f has a cost of zero credits for a 1-hop profile view. As a result, profile views among friends are never denied, no matter what the state of the credit network. Next, we consider legitimate profile views among non-friends.

Let us consider any cut of the social network other than the attack cut. This $\alpha\beta$ cut partions V into V_α and V_β, C into C_α and C_β, and N into N_α and N_β, respectively. We call the cut between C_α and N_α the α attack cut, and the cut between C_β and N_β the β attack cut, respectively. Likewise, we call the cut between N_α and N_β the N-$\alpha\beta$ cut, and the cut between C_α and C_β the C-$\alpha\beta$ cut, respectively. See Figure 7.

Now, we consider the question to what extent the activity of nodes in C can impair nodes in N in their ability to view the profiles of users on the other side of the $\alpha\beta$ cut.

Observation 0: Without loss of generality, we can ignore views from users in V_α to users in V_α, and from users in V_β to users in V_β. The paths associated with such views must cross the $\alpha\beta$ cut an even number of times, which means that they do not change the total amount of credit available on this cut.

Observation 1: Without loss of generality, we can focus our attention on the case where the nodes in C_α crawl nodes in N_β, but nodes in C_β do not crawl nodes in N_α. This is because the credit imbalance caused by the crawler's activities along the $\alpha\beta$ cut is maximized in this case.

Observation 2: The nodes in N_β cannot be impaired, because the crawler's activity increases the amount of credit available to them along the $\alpha\beta$ cut. However, nodes in N_α may be impaired, because the crawling of nodes in N_β by nodes in C_α reduces the credit available to N_α along the $\alpha\beta$ cut.

Observation 3: The crawler's activities can reduce the maximal viewing rate available to nodes in N_α by the ratio of the sizes of the α attack cut and the N-$\alpha\beta$ cut. If the α attack cut is larger than the N-$\alpha\beta$ cut, then the crawler can render the nodes in N_α unable to view the profiles of non-friends in V_β.

Observation 4: Social networks are known to have a very densely connected core, with smaller communities connected at the edges [13]. This means that cuts through the core of the network are very large. An attacker would have to be very powerful (i.e., have a very large attack cut) to be able to significantly impair such cuts, and thus a large number of users.

Observation 5: If the attacker is well connected to a small fringe community, it can exhaust credit along the cut that separates the small community from the core of the network. However, by definition this would only affect a relatively small number of users in that community. Moreover, these users can rectify the problem by adding more links to the core of the network.

To summarize, an attacker would have to be very powerful to be able to have a noticeable effect on cuts through the core of the network, which would impact larger numbers of users. A modestly strong attacker can impair users in small fringe communities. However, such users can respond by forming additional links to the core of the network. (Recall our assumption that an attacker cannot compromise the accounts of specific users of his choosing. If an attacker were able to do this, he could target weakly connected users or small communities very effectively.)

We will further explore the impact of crawlers on legitimate users empirically in Section 6.

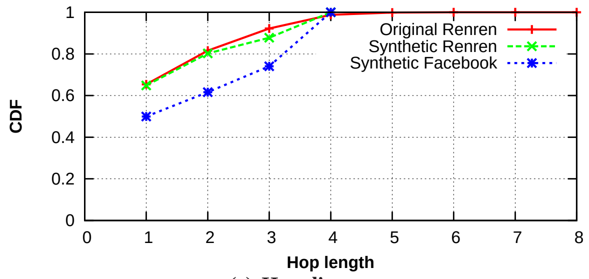
6 Evaluation

In this section we evaluate the performance of Genie over several different social networks. When evaluating Genie's performance, we focus on the two primary metrics of interest: (i) the time required for an attacker to crawl a Genie-protected network and (ii) the amount of legitimate activity blocked by Genie.

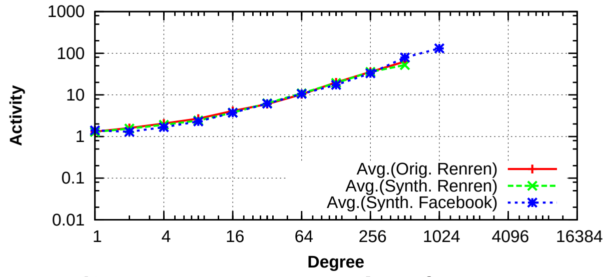
6.1 Datasets used

To evaluate the performance of Genie, we need four datasets: (i) a social network graph, (ii) a time-stamped trace of normal users' profile views in the form of (X,Y,T) where X views Y at time T, (iii) a crawler's attack topology, that is, how the nodes controlled by the attacker are embedded in the network and (iv) the attacker's profile crawling trace.

Social network graphs: We evaluate the performance of Genie on the four different social network graphs (RR-PKU, Facebook, You Tube and Flickr), which we discussed earlier. Table 1 shows their high-level characteristics. While our RR-PKU and Facebook networks are small with tens of thousands of nodes, our YouTube and Flickr graphs are large with millions of nodes.



(a) **Hop distance**



(b) **Degree to #interaction correlation for** RR-PKU

Figure 8: **Comparison of different synthetic workloads generated and the original workload.**

Gathering and generating workload traces: Gathering profile viewing traces for large social networks is very hard because it requires explicit cooperation from social network site operators, who are often very reluctant to show traces out of privacy concerns. It does not help that there have been high profile incidents when site operators have come to regret their decision to share their user data for research purposes [5]. We found it impossibly hard to obtain workload traces for graphs other than RR-PKU [11].

So we decided to design a workload generator that reproduces the key features of the original RR-PKU trace

Network	#Nodes	#Unique Activities
RR-PKU	33k	77,501
Facebook	63k	123,368
Youtube	1.1m	1,326,184
Flickr	1.6m	1,592,052

Table 3: **Statistics of synthetic workload generated for different large networks.**

which we observed in Section 4. These features include the correlation between request/receiver node degree and number of interactions, and locality of interactions.

To this end, we assign requesters and receivers in the original RR-PKU workload trace into variable sized bins based on their node degrees. For each degree bin, we calculate the number of profile requests made and received by users in the bin along with the hop length distribution of the profile requests. These statistics are then used for generating a synthetic workload for a given target graph.

Next, nodes of the target network are also divided into similar sized bins based on their node degree. Based on the RR-PKU requester/receiver degree and number of interactions statistics, we sample a set of nodes in the target graph as possible requesters and receivers. From this set, we pick different requester receiver pairs which fit the RR-PKU interaction hop length distribution. This gives us an activity trace for the target graph.

We generate synthetic workload traces for the 4 networks that we consider: RR-PKU, Facebook, Youtube and Flickr. The high level statistics of the new workloads are shown in Table 3. [2] To test if the newly generated traces preserve the key features we aim to reproduce, we plot the locality of interactions in Figure 8(a) and the correlation between node degree and number of profile views in Figure 8(b) in both the original and the synthetic workload traces. The plots match quite well indicating that the synthetic workload generator retains the key properties of the original RR-PKU workload.

Crawler's attack topology: We model attackers by corrupting random nodes in the graph. We simulate 1,10,100 and 1000 corrupted accounts in the RR-PKU, Facebook, YouTube and Flickr network and took the average of multiple simulations to report our results. As the attacker gets access to more number of corrupted accounts in a network, he also acquires more number of attack links to honest users. The varying strength of attackers on different networks is discussed in Section 4.2 and Table 2.

[2]We note that while the larger graphs have higher number of profile request activities in the trace, the number of activities do not scale linearly with the number of nodes in the trace. This is to be expected because larger graphs have a greater fraction of low degree nodes in their network that generate few profile requests, if any. Our generator faithfully captures the correlation between node degrees and their activity. Other studies of social networks, such as [19], have observed similar trends.

Attacker's profile crawling trace: To generate the attacker crawling workload, we follow the same attacker model discussed in Section 4.2. The primary objective of the attacker is to crawl the entire graph in as few refresh periods as possible. For each user profile, an attacker closest to that user is assigned the task of crawling that profile. This strategy ensures that the attacker achieves maximum network locality in profile crawling.

6.2 Trace-driven simulation methodology

To evaluate the performance of Genie, we built a trace driven simulator. Our simulator takes the four datasets described above as inputs. We use the social graph connecting the users to simulate a credit network. For each profile request in the workload traces, our simulator checks if there exists a set of paths in the credit network that allow p units of credits to flow between the viewer and the viewee, where p denotes the shortest path length separating the viewer and the viewee. To this end, our simulator computes the max-flow paths [16] between the viewer and the viewee. If the max-flow is larger than p, then the profile view is allowed and if it is not, then the view is blocked. If the profile view is allowed, the credits along the links of the max-flow paths are updated as described in Section 5.

A key input to our simulator is the credit refreshment rate, which denotes the rate at which exhausted credits on the links are replenished. We set the credit refreshment rate in our simulator by tuning the following two parameters described in Section 5.1.1: (i) the initial credit value, i, assigned to each link in the network at the beginning of the simulation, and (ii) the credit rebalance rate, r_b, which restores some of the exhausted credits on the links after each time step, say of duration t. We set the parameter r_b to 1, which has the effect of restoring the credit values on all links to i after every refresh time period. So $\frac{i}{t}$ represents the effective credit refreshment rate, which determines the number of profile views accepted both for crawlers and normal users. Larger the value of credit refreshment rate, the more the number of profile views accepted from both crawlers and normal users and vice-versa. Thus, the key evaluation challenge that we address using our simulator is: *does there exist a credit replenishment rate that would offer a good tradeoff between blocking crawlers and allowing legitimate activity, i.e., a credit replenishment rate that forces crawlers to invest a long time to fetch all profiles, while the amount of blocked user activity is kept minimal.*

Problem with scaling simulations to large graphs: While we were able to run our simulator over the smaller RR-PKU network with 33 thousand nodes, we found it computationally hard to scale our simulations to the much larger YouTube and Flickr social networks with millions of nodes, edges, and profile views. The computational complexity arises out of three reasons: (i) even a single max-flow computation over a large graph is expensive ($O(V.E^2)$), (ii) we have to perform millions of such computations, one for each profile request in the trace, and (iii) worse, the computations cannot be parallelized and have to be done online and in sequence, as the max-flow computation for a profile request has to account for credit changes on links in the network due to all prior profile requests in the workload trace.

Precomputing paths to scale simulations to large graphs: To work around the scalability bottleneck for large graph simulations, we *precomputed* shortest paths between every pair of viewer and viewee in the workload trace. To check whether a profile request should be allowed, rather than have the simulator compute max-flow paths online, we force the simulator to use the offline precomputed shortest path between the viewer and the viewee. Precomputing shortest paths offline scales significantly better than online max-flow computation because (i) a single shortest path computation (complexity ($O(V + E)$)) is considerable less expensive than a single max-flow (complexity $O(V \times E^2)$ in worst case), and (ii) shortest path computations for individual profile request can be parallelized on a large cluster of machines.

An upper-bound on blocked user activity: While approximating max-flow paths with shortest paths scales well, it comes at a cost: a profile request in the credit network may be denied by our simulator because the precomputed single shortest path between the viewer and viewee lacks sufficient credit, even though there are multiple potentially longer (max-flow) paths with sufficient credit. More precisely, for a given pair of nodes in a given credit network graph, lack of sufficient credit along a shortest path between them does not necessarily imply lack of sufficient credit along max-flow paths. However, if there is sufficient credit along the shortest path, one can be certain that there exist max-flow paths with sufficient credit between the nodes. Thus, our optimized simulations result in an *over-estimate or upper-bound on the rejected profile views*.

To check how tight the upper-bound estimates using precomputed shortest paths are to actual numbers of rejected profile views, we compared the performance of max-flow paths and shortest paths over the RR-PKU social network. The relatively small size of RR-PKU allows us to simulate max-flow paths as well. Figure 9 shows the percentage of blocked normal user activity for the different path computation strategies in the presence of an attacker with 10 accounts. As the amount of credit available per refresh period increases the percentage of blocked activity also decreases for both types of path computations. However, our approximation technique consistently provides an upper bound on the percentage

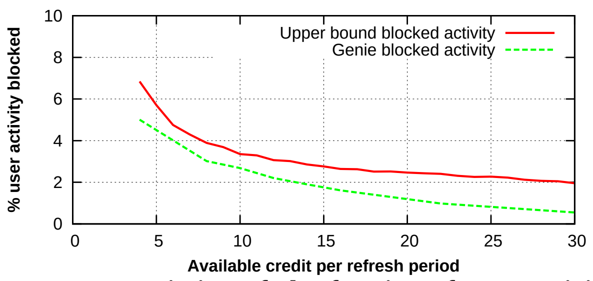


Figure 9: **Variation of the fraction of user activities blocked with different credit values in RR-PKU. We compare the results obtained using shortest-paths (red line) with those obtained using max-flow paths (green line). The red line provides a close upper bound for the green line.**

of blocked user activities across the different values of available credit. The difference between the actual and upper-bound plots is always lower than 2% of all user activities, suggesting a fairly tight upper-bound.

A lower-bound on time taken to finish crawls: A key goal of our simulations is to determine whether there exist credit replenishment rates that offer a good tradeoff between maximizing the crawling time for attackers and minimizing blocked profile requests by legitimate users. A conservative approach to determining the existence of such credit replenishment rates would consider lower-bounds on time taken for an attacker to finish its crawls, while considering upper-bounds on blocked user activity. Having proposed a scalable way to upper-bound the blocked user activity, we now propose a scalable way to lower-bound the time taken for an attacker to finish his crawl of the entire network.

Let C be the set of nodes that are under the control of the attacker, L the set of links that connect these nodes to other nodes in the network, N the set of remaining nodes in the network, and i be the credit value on links that is replenished after every refresh period. We can compute a lower-bound on the time it would take to fetch the profiles as follows: within a single credit refreshment period, the maximum amount of credit available the attacker to fetch user profiles is $i \times |L|$. We can compute T, the minimum number of credits that the attacker would need to fetch all the N user profiles, by summing the credits required to fetch each of the N nodes from the attacker controlled node that is closest to them. Then, the attacker would need at least $\frac{T}{i \times |L|}$ refresh periods to fetch all the profiles, which constitutes a lower-bound on crawl time.

To check how tight the lower-bound obtained using our formula above is, we compared the estimates of the time needed to perform complete crawls of the RR-PKU social network given by our formula with those obtained by running max-flow computations over the RR-PKU so-

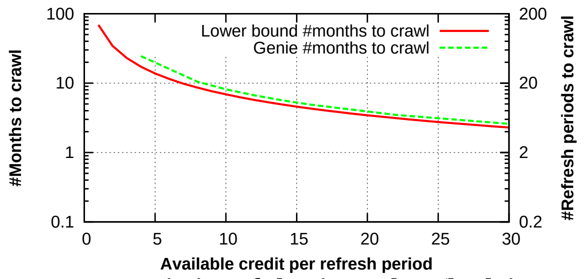


Figure 10: **Variation of the time taken (both in terms of months and number of refresh periods) by an attacker to complete a crawl with different credit values. We compare the lower bounds (red line) calculated with our formula with the experimental results (green line). The lower bounds are tight.**

cial network. Figure 9 shows the number of credit refreshment periods as well as the time needed to crawl the entire network for an attacker controlling 10 nodes. The time needed to crawl is obtained by multiplying the number of credit refreshment periods with duration of the credit refreshment period, which is 2-weeks in our traces. [3] As the available credit per refresh period increases the estimated time to crawl the entire network decreases as well. However, our formula consistently provides a very tight lower bound on the time taken to crawl the network for attackers with varying attack power across the different values of credit replenishment rates.

6.3 Tradeoff between limiting crawlers and blocked activity

We now switch our attention to the core tradeoff that is being made as we select the appropriate credit refreshment rate — the amount of time it takes the crawler to crawl the entire graph and the fraction of legitimate activity that is blocked. We have already observed that to block crawlers effectively we need to replenishing credits at a slow rate. However, a limited rate of credit replenishment opens up the possibility of legitimate users' views getting blocked. In this section, we explore the extent to which legitimate user activity is blocked by Genie as it tries to limit crawlers.

We ran our simulator for various different values of available credit per refresh period. For each value, we compute two metrics: (i) the time it would take for an attacker to finish its crawl and (ii) the percentage of legitimate user activity that is blocked. We compare these two metrics looking for a good tradeoff, where crawlers are effectively slowed down, while good user activities

[3]Our original RR-PKU workload trace covers a 2-week period and hence, the synthetic workload traces we generated are also over the same period.

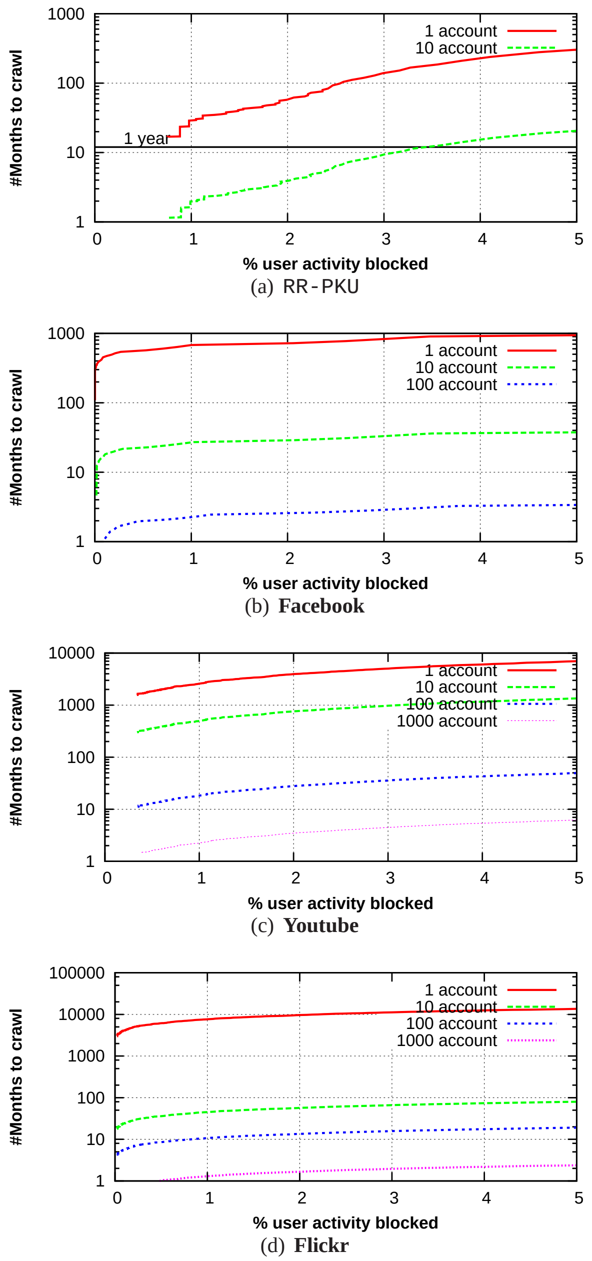


Figure 11: **Trade off between fraction of user activity blocked and time taken to finish a complete crawl with attackers of varying strengths over different social network graphs. We conservatively used the upper-bounds for blocked user activity and the lower-bounds for time to crawl here. Further, we conservatively allowed crawlers to exhaust the credits on links before allowing any legitimate user activity.**

blocked is held to a minimum. We present the basic tradeoff for our different social networks in Figure 11. For each social network, we show the results for attackers of different strengths. On YouTube and Flickr graphs with more than 1 million nodes, we considered an at-

Total activity blocked	Src	Dest	Middle
2,574	1,961	423	190

Table 4: **Classification of total activity blocked for RR-PKU.**

tacker controlling up to 1000 nodes, while for RK-PKU with only 30 thousand nodes, we limited the attacker strength to 10 nodes. While the absolute number of compromised accounts controlled by the attacker might seem small, it is worth noting that the percentage of compromised nodes in these networks is still substantial. For a point of comparison, controlling 1000 nodes in a 1 million node network is equivalent to controlling 800,000 accounts in the current Facebook network with 800 million accounts.

The plots show that it is possible to slow-down crawls sufficiently to force an attacker to spend several months to tens of months to complete a single crawl. At the same time, the percentage of blocked user activity can be held to less than 5%. In many instances, the blocked activity can be held lower than 1%. Thus, there are two important take-away from these results: first, when Genie is employed a certain amount of legitimate activity will unavoidably be blocked. Second, unless the attacker is overwhelmingly powerful, the impact of the attacker on legitimate users is minimal.

6.4 Alternate strategies for blocked users

We observed in the previous section that a certain amount of blockage of legitimate activity is unavoidable. We now pose a simple question: can users do anything to minimize the amount of their blocked activity? To answer this question, we first investigate the blocked views in more detail. We then propose some recourses available to users with blocked activity.

We analyze the set of blocked activities in our extensive RR-PKU simulation, where we compute max-flow paths to verify if a profile view has to be allowed. We intentionally focused on max-flow based simulations because of the certainty that profile views blocked during such simulations are rejected due to lack of credit in the network. In simulations that yield upper-bounds on blocked activities, an activity might be rejected due to using shortest paths to approximate max-flows.

For the analysis in this section, we focus on one simulation experiment, where 2.6%(or $2,574$ activities) of the user activities are blocked and the attacker controlling 10 compromised accounts needs 8 months to complete the crawl.

A profile view request can be blocked due to one of the three reasons: the profile viewer runs out of credit on all links connected to itself (i.e., source blocked request), the credit on links connecting the profile viewee

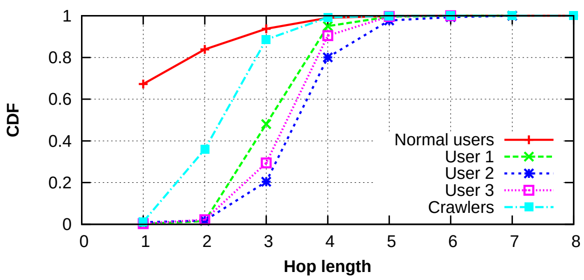


Figure 12: **Comparison of the hop distance distribution of 3 users with too many views in RR-PKU. Their profile view characteristics deviates considerably from the normal user activities.**

is exhausted (i.e., destination blocked request), or the request is blocked due to credit exhaustion somewhere in the middle of the network. We quantify the different number of blocked activities falling in the above three categories in Table 4. Strikingly, only 7% of the blocked requests (i.e., 0.18% of all requests) are blocked due to lack of credit on links in the middle of the network. Most blocked activity is due to credit exhaustion on links connecting the viewers or the viewees. On examining the degrees of these viewers and viewees, we find that 96% of them have node degree 1 and 99% of them have degrees 5 or less! That is, most activities blocked near the source or destination, is due to source or destination nodes having way too few number of friends and lying on the fringes of the network graph. Our results support our observation in Section 5 that social network graphs are sufficiently well connected in their core that most blocked activity (and credit exhaustion) occurs close to the fringes.

Next, we investigated the amount of blocked activities for individual users. We found that a small number users are bearing the brunt of the blocked activities. 1808 of the 2574 (or 70%) of the blocked profile requests are made by 3 users in the network. Interestingly, all three users issue two order of magnitude more requests than an average RR-PKU user and they are all blocked near the source. Further, investigation suggests that these top 3 users exhibit crawler-like characteristics. Figure 12 shows that the three most blocked users issue considerably more long distance views than normal users. In fact, network locality of their profile requests resembles that of a crawler (see Figure 5) than that of a normal user (see Figure 4(a)). Ignoring these three users, whose request trace bears strong resemblance to crawlers, the percentage of total blocked activity falls to less than third of its original value, which is already a low percentage (2.6%) of all activity.

For the remaining users who contribute to only 30% of blocked views, we have already observed that most (99%) of the users have degree less than 5. These are users who got blocked because their low number of friend links are insufficient to support the reasonable number (on average 6 views) of view requests they issued. However, we argue that there is a simple and natural recourse available to them. They can simply make more friends!

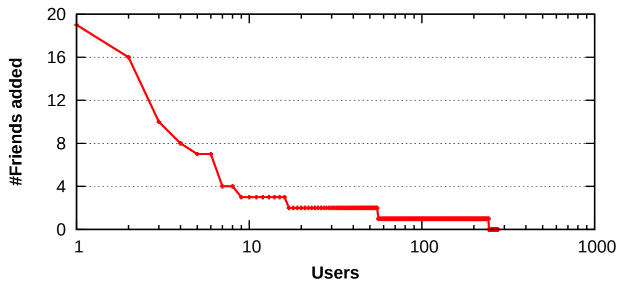


Figure 13: **We observed how many extra links most of the blocked RR-PKU users needed for completing their activities. Evidently they needed just a few more links.**

In order to test this hypothesis, we perform a simple experiment. We re-run the Genie simulation where each blocked user (falling in the low degree category), establishes a friend link to the destination of the blocked request (i.e requester sends a friendship request and the receiver approves it). This immediately leads to the acceptance of that view request. At the end of the simulation, we look at the number of friend links established by each blocked user so that all the earlier blocked requests could now be accepted.

Figure 13 shows the distribution of number of friend links established versus the ranked set of users. A significant majority (97%) of the blocked users can get their requests accepted by only establishing very few links (less than 4). Thus, most of the activity blocked by Genie would be accepted if the users of the blocked requests spent a minimal effort to establish a small number of friends. In fact, if Genie were to be deployed, it would naturally incentivize users to form a few more friend links. Given that many OSN sites already explicitly encourage their users to form more friend links, we believe that the overheads from Genie would be acceptable for a majority of users.

7 Conclusion

In this paper, we address the problem of preventing large-scale crawls in online social networks, and present Genie, a system that can be deployed by OSN operators to thwart crawlers. Based on trace data from the Renren OSN, we show that the browsing patterns of normal users and crawlers are very different. While most normal users view the profiles of a modest number of users who tend to be nearby in the social network, even a strong, strategic crawler must view profiles of users who are further away. Genie exploits this fact by limiting the rate of profile views based on the connectivity and social network distance between a profile viewer and viewee. An experimental evaluation on multiple OSNs shows that Genie frustrates large-scale crawling while rarely impacting browsing by ordinary users; the few honest users who are affected can recover easily by adding a few additional friend links.

References

[1] `http://tcrn.ch/9JvvmU`.

[2] Rate limiting for yahoo! search web services. `http://developer.yahoo.com/search/rate.html`.

[3] `http://on.msnbc.com/qvLkX2`.

[4] A standard for robot exclusion. `http://www.robotstxt.org/orig.html`, 1994.

[5] `http://cnet.co/6JiHr8`, 2011.

[6] P. Dandekar, A. Goel, R. Govindan, and I. Post. Liquidity in credit networks: A little trust goes a long way. In *NetEcon*, 2010.

[7] D. DeFigueiredo and E. T. Barr. Trustdavis: A non-exploitable online reputation system. In *CEC'05*.

[8] Facebook. `http://www.facebook.com`.

[9] Flickr. `http://www.flickr.com`.

[10] A. Ghosh, M. Mahdian, D. Reeves, D. Pennock, and R. Fugger. Mechanism design on trust networks. In *NetEcon*, 2007.

[11] J. Jiang, C. Wilson, X. Wang, P. Huang, W. Sha, Y. Dai, and B. Y. Zhao. Understanding latent interactions in online social networks. In *IMC*, 2010.

[12] S. S. Marti Motoyama Damon McCoy, Kirill Levchenko and G. M. Voelker. Dirty jobs: The role of freelance labor in web service abuse. In *Proceedings of the USENIX Security Symposium*, 2011.

[13] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and Analysis of Online Social Networks. In *IMC*, 2007.

[14] A. Mislove, A. Post, K. P. Gummadi, and P. Druschel. Ostra: Leveraging Trust to Thwart Unwanted Communication. In *NSDI'08*.

[15] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: Inferring user profiles in online social networks. In *WSDM*, 2010.

[16] A. Post, V. Shah, and A. Mislove. Bazaar: Strengthening user reputations in online marketplaces. In *NSDI*, 2011.

[17] Renren. `http://www.renren.com`.

[18] T. Stein, E. Chen, and K. Mangla. Facebook Immune System. In *SNS'11*.

[19] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the Evolution of User Interaction in Facebook. In *Proc. WOSN'09*, Barcelona, Spain, Aug 2009.

[20] C. Wilson, A. Sala, J. Bonneau, R. Zablit, and B. Y. Zhao. Don't tread on me: Moderating access to osn data with spikestrip. In *WOSN*, 2010.

[21] YouTube. `http://www.youtube.com`.