Georg-August-Universität
Göttingen
Institute of Computer Science

ISSN 1611-1044
No. IFI-TB-2013-01

Technical Report

With a Little Help From my Friends: Replica Placement in Decentralized Online Social Networks

David Koll, Jun Li, Xiaoming Fu

Technical Reports
of the Institute of Computer Science
at the Georg-August-Universität Göttingen

January 2013

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstr. 7
37077 Göttingen
Germany

Phone: +49 (551) 39 - 172000
Fax: +49 (551) 39 - 14403
E-Mail: office@cs.uni-goettingen.de
WWW: www.ifi.informatik.uni-goettingen.de

With a Little Help From my Friends: Replica Placement in Decentralized Online Social Networks

David Koll, Jun Li, Xiaoming Fu

Abstract

Online social networking (OSN) platforms have seen a tremendous increase in user population and user-provided content. Concomitant with OSNs' growing popularity, however, are increasing concerns from users about their privacy and the protection of their data. As user data management is usually centralized, OSN providers nowadays have the unprecedented privilege to access every user's private data, which makes large-scale privacy leakage at a single site possible.

One way to address this issue is to decentralize user data management and replicate encrypted user data at individual end-user machines across the OSN. However, such an approach must address new challenges. In particular, it must achieve high data availability with minimal replication overhead for all users, without relying on permanent online storage. Moreover, it has to scale with large social networks and be resilient and adaptive in multiple dimensions, including high churn of participants and attacks from malicious users.

Whereas recent works show limited success in this direction, we introduce a different approach that exchanges recommendations among socially related nodes in order to effectively distribute user data replicas throughout the OSN. We conduct extensive evaluation based on both simulation and the deployment of a prototype and show that our approach addresses all aforementioned challenges.

I. Introduction

The evolution of Online Social Networks (OSNs) into ubiquitous platforms of communication raises severe privacy and security concerns. As OSN providers such as Facebook deal with tremendous amounts of user information, they also obtain a deep insight into their users' personal opinions, social relationships, and economical or political references. With currently over 900 million users, Facebook is the most popular OSN platform and already controls the personal data of more than 10% of the world's population. and of a far larger portion of people with internet access Yet, the hunt for user data is not over, and users are often helpless when faced with patronizingly imposed changes of any kind, as recently demonstrated by Facebook and its acquisition of the Instagram user base with a billion dollar agreement. Moreover, the centralized control often results in the misuse of user data [13]. The Facebook Beacon application, for example, leaked sensitive shopping information of users to Facebook without their consent, resulting in a class action lawsuit that cost Facebook $9.5 million to settle in 2009 [16]. As another example, the leakage of millions of passwords of LinkedIn users also proves that large-scale security breaches at centralized repositories are a valid threat [1].

While it remains to be seen whether OSN providers give up a major source of income and grant comprehensive security and privacy means to their users (e.g., encryption of data), decentralized OSNs are becoming more promising for better user data security and privacy. Instead of relying on a central data repository, a decentralized OSN can allow users to regain control over their data. For instance, users can encrypt their data according to their own need, enforce access control of their data at their discretion, or distribute their data at their designated storage facilities or user devices across the OSN.

In this report, we address the key challenge of making all data of all users available at all times in the absence of a central provider. Whereas one might consider classical peer-to-peer (P2P) data storage approaches [6], the OSN scenario requires a specifically tailored solution for a number of reasons. First, the online patterns of OSN users are much more bursty than in traditional decentralized applications [18]. Second, social networking platforms are increasingly employed by mobile devices, which usually come with meager storage capacity. Third, an OSN needs to be robust *as a whole* in the sense that it needs to provide high data availability for all data of interest, even if it does not originate from an active, contributing user. Otherwise, the remaining users might suffer from the data unavailability.

Recently, researchers have proposed a wide range of alternative systems that consider the characteristics of an OSN in their design [7], [12], [21], [26]. In particular, the inherent social relation among OSN users offers potential incentives for users to host data for each other that go beyond the motivation of tit-for-tat in classical P2P networks. In these approaches, the degree of storage decentralization ranges from employing a limited number of high capacity devices to using end-user machines only. While some of these solutions provide means to realize data storage in decentralized OSNs, each introduces new shortcomings, ranging from technical and economical deployability issues [7] to limited robustness [26] and discrimination of users due to a dependency on socially related nodes [12], [21].

In an effort to overcome these drawbacks, this work offers the following contributions:

- We propose a new, generic approach to placing replicas of user data upon a decentralized OSN. Dubbed **SOUP** (Self-Organized Universe of People), our system can be integrated within any such OSN. In addition to enabling every user to store their data at their own machine, SOUP efficiently distributes replicas of a user's data to

carefully selected OSN participants. Despite conservative assumptions on the availability of resources, SOUP's performance in terms of availability of user data is close to that of a centralized solution. By introducing little replica overhead and low communication overhead, SOUP scales well with heavily populated OSNs and is also easily deployable, supporting a fast transition from a centralized OSN to a decentralized one.

- SOUP further provides salient key features: It is adaptive such that it can quickly recover from node dynamics (e.g., large-scale node departures) and is resilient to high churn rates, a typical product of OSNs with high levels of mobile device access. SOUP is also able to tolerate a high ratio of malicious OSN users without a significant performance degradation. We show that even if half of the identities in an OSN are controlled by the adversary and attack SOUP, the system performs steadily. Furthermore, our approach offers a robust OSN *as a whole*, regardless of the participants' social relations and online probability. Such a property is essential not only to not discriminate any user, but also to enable access to all data of interest at any time.
  Finally, SOUP does not rely on permanently available or altruistically provided storage, but is able to make an opportunistic use of such resources as they become available. In the presence of unselfishly donated resources —such as a permanently online server with high data capacity— our system converges quickly to better performance.
- We incorporate SOUP into a decentralized OSN prototype. Our evaluation, based on our deployment experience, shows that decentralized OSNs with SOUP are feasible in terms of bandwidth and control overhead. As a side effect, we also gain a first glimpse into OSN profile data that is not based on crawling. Whereas a crawler might miss a lot of data that is simply not publicly available, we were able to collect a small but meaningful set of data that contains *complete* profiles of Facebook and Google+ users. Analyzing this set of data reveals the practicality of SOUP, since data items and profile sizes in OSNs are of surprisingly small size.

The remainder of this report is structured as follows: Section II reviews related work. Section III illustrates the design of SOUP in detail, followed by a discussion of possible attacks in Section IV. Sections V and VI present an extensive evaluation of SOUP based on simulation and deployment, respectively. Open issues are discussed in Section VII, and Section VIII concludes this report.

II. Related Work

Distributed storage systems have been studied intensively, especially in the past decade [6]. They are generally designed for supporting traditional decentralized applications such as file sharing, which are often characterized by long durations of online time typically spanning multiple hours up to days. However, users' online patterns in social networking sites show high activity peaks with larger gaps of offline time [9], [18]. Additionally, content in social networking platforms is often uploaded and accessed from mobile devices that may be disconnected most of the time. For instance, users may turn off their data connections to save battery from time to time. Another OSN characteristic is the inherent relation between the participants, which can imply storage incentives among the users of the OSN. Intuitively, a user will prefer to store the data of their friends to that of a stranger, and in case of storage exhaustion, a user would prefer to drop data of the latter. Furthermore, in contrast to traditional P2P systems, the tit-for-tat strategy is not as desirable for OSNs. Users rather need the social network *as a whole* to be robust, with each user's data accessible at any given time. Otherwise, even highly contributing users would suffer if their data of interest is unavailable.

With the aforementioned reasons in mind, researchers have suggested three different approaches to the decentralized storage in the context of OSNs:

The first approach is to distribute data control and storage to a limited number of permanently online storage locations [4], [21]. The storage might either be altruistically provided or based on economic incentives such as user payments. However, altruistic provisioning, usually from a limited set of volunteers, is unlikely to meet the demand of a large-scale social network with as many as several hundred million users. At the same time, a requirement for user payments will most likely prevent a large-scale transition from current centralized OSNs, which do not impose fees on their users. The dependency on both altruistically provided and paid servers is also a concern, as data loss can occur when such servers become disengaged abruptly.

The second approach is to ask each user to provide a permanently available storage space for their own profile [7]. This approach does provide high data availability and low overhead, but it requires *all* users to be technically able to provide and configure their own permanently available data storage, which is impractical. The issue might be mitigated by incentivating storage and configuration providers, which however results in monetary costs for the user again.

The third approach, as represented by PeerSoN [26] and Safebook [12], is to let nodes cooperate and provide temporarily available storage to each other. With the mutual cooperation of nodes and flexible data storage locations, this approach allows users to be independent of altruistic servers and their drawbacks. Additionally, as every participant is contributing resources, the OSN can operate without additional costs for every user. The major challenge of this approach, however, is to provide high data availability to the users.

PeerSoN introduces an optimized node selection algorithm, and nodes with mutual agreements store data for each other. The main issue of this approach is its inability to construct a robust OSN. Users with an online probability of less than 30% achieve less than 90% data availability, meaning one out of ten requests for their data is unsuccessful. Given the power-law distribution of online time in OSNs [9], [18], the majority of users is unable to achieve high levels of data availability. The outcome is a frail OSN, in which even highly contributing users may find data of interest unavailable.

Safebook mirrors each user's data at a subset of their direct friends, and the data is only accessible through a path of so called shells. The mirrors form the innermost shell, friends of mirrors form the second shell, and so on. Retrieving a node's data requires traversing the shells along the path of nodes that befriend each other. Unfortunately, a user depends on her social contacts in two ways: Initially, she needs enough suitable friends to build the innermost shell, which is difficult for many users in OSNs as they only maintain very few social relations [22]. Moreover, *all* nodes on the same path towards the innermost shell need to be online simultaneously in order to provide access to the data, which is unlikely considering the typical online time patterns of OSN nodes [33].

III. SOUP

We now present SOUP, a scalable, robust and secure system for storing user data among heterogeneous nodes in a decentralized OSN with very high availability. We first outline the challenges SOUP faces, and then describe decentralized mechanisms that address these challenges and achieve data availability that approximately matches that of a traditional centralized OSN.

A. Design Challenges

The primary challenge that SOUP needs to overcome is to achieve high data availability that is very close to that of centralized social networks while decentralizing the control and storage of the data. Our approach will not rely on a central entity (e.g., a global information repository); instead, it will leverage resources provided by participating nodes without introducing significant replica and thereby—as replicas need to be distributed—communication overhead.

In leveraging resources of participating nodes, SOUP must address the significantly differing characteristics of OSN users. Besides diverse hardware configurations (e.g., mobile versus desktop devices), the online time patterns of OSN nodes can differ substantially from each other. Given the power-law distribution of online times in OSNs, the majority of users are seldom online, and sessions are usually short and bursty [9], [18], [33]. Moreover, SOUP must recognize that the users' machines are not servers and usually only offer very limited storage capacities. SOUP must thus use the resources each node supplies efficiently. As a node exhausts its capacity, it must be able to decide which data to keep and which to drop. In addition, SOUP should be open to the potentialities of altruistically provided resources and exploit them if available.

While the most preferred resource selfless users may provide is probably highly available storage with high capacity (i.e., a server provided by a user), smaller contributions to the resource pool (e.g., a desktop machine that is on for a relatively long time) should also be exploited.

Also, SOUP should exploit the potential of social relations within the OSN. In fact, multiple applications have leveraged these relations to defend against attackers or improve their performance (e.g., [31]). Also, some works suggest that social relations can be the usually required incentive for self-interested nodes to cooperate with other nodes in an OSN [**?**], [17]. In particular, users can provide each other feedback whether they succeeded or failed in obtaining data from a participating node. If utilized efficiently, such cooperation and exchange can help every user to distribute their data within the OSN more efficiently.

Finally, SOUP must have several key properties: (1) It must scale to the dimensions of OSNs and be easily deployable in large-scale scenarios. The latter requires a quick convergence to a stable system state; even when a large number of nodes join the system at the same time and each node has little information to begin with, SOUP must provide effective means to quickly reach high data availability. (2) It must be robust. Even if a node's capabilities or social connections are weak, its data should *not* be less available than data of others. Otherwise, not only the weak node will suffer, but also those who need to access its data. (3) Additionally, SOUP must be resilient in multiple dimensions. Because OSN sessions are often short-lived [18], [27], SOUP needs to handle high churn rates. It further needs to be adaptive to departures of large fractions of nodes at the same time that are caused by reasons such as network failures. Finally, it must be resilient under the attack of malicious users, performing, e.g., a sybil attack [15].

B. System Overview

The main goal of SOUP is to ensure that at any given time for every OSN node, either the node's data is available at the node itself (the node is online), or a copy of the data—called *replica*—is available at another node—called *mirror*. The core task for SOUP is thus mirror selection: every OSN node needs to be able to select the most eligible nodes as mirrors before it places its data replicas there.

Every node employs two different modes to select its mirrors: a *bootstrapping* mode and a *regular* mode. When a node joins the OSN and does not know anything about the network, it runs in the bootstrapping mode; it gathers online time statistics from nodes that it discovers while exploring the OSN (i.e., how often these nodes are online), ranks them based on the statistics (Sec. III-C), and selects some of them as mirrors. Once a node establishes its first social relation, it begins to learn from its friends about their experience in accessing its data at those mirrors, and transitions to the regular mode; it will rely on friend experience to rank mirror candidates (Sec. III-D), and select mirrors.

In SOUP, nodes will not simply select the top ranked nodes. SOUP in fact gives a higher weight to nodes with social relations (i.e., friends). Furthermore, we include a random component to ensure that a node does not get stuck with a certain set of mirrors, while better candidates may exist.

A node will thus select its mirrors with all these considerations, and store its data at the mirrors (Sec. III-E).

As a node may be selected as a mirror by many nodes, it sometimes has to drop data of some nodes. The dropping strategy is critical, especially when an adversary is flooding an OSN and many nodes receive numerous requests to store data for the attacker (Sec. III-F).

Out of scope issues. In SOUP every replica must be encrypted and only authorized users can decrypt it. In particular,

Algorithm 1 Ranking mirror candidates in the bootstrapping mode at node u

M_u: set of u's mirrors, initially empty
V_u: set of mirror candidates for u
A^u,A^v: online-time matrices of nodes u, v
ot_v: online-time ranking of a candidate v
for all $v \in V_u$ **do**
  exchange A^u, A^v
  calculate ot_v with Equation (1)
end for
$r_{ot} \leftarrow$ sort V_u by ot

a mirror node must not be able to access the personal information it stores unless it is authorized to do so. The method for data encryption, however, is not the focus of this paper. SOUP can exploit any of the promising works in this direction; for example, SOUP can use Attribute Based Encryption (ABE) where each user can control the access to their data with a very fine granularity by distributing appropriate attribute keys to her authorized users [10]. Furthermore, every node will publish its set of mirrors to the OSN so that any node can look up the mirrors to access the node's data. A wide range of solutions already exist for the publish and lookup operations, including those using Distributed Hash Tables (DHT) [8].

C. Mirror Candidate Ranking in the Bootstrapping Mode

SOUP provides a mechanism to allow new nodes to quickly achieve high data availability, even with limited initial information. At the time a user joins the OSN, it does not possess any information about well-suited mirrors. However, as it makes contacts to other nodes, it can treat every contacted node as a mirror candidate and rank its eligibility as a mirror. In fact, based on findings of [33], OSN users are most active when they have just joined, and they contact many other nodes (e.g., with many initial friend requests).

More specifically, as shown in Algorithm 1, every time a node u contacts a node v, nodes u and v can exchange their online time matrices A^u and A^v, where A^u and A^v are $m \times n$ matrices with $m = 7$ and $n = 24$ that respectively represent u and v's probability of being online at every hour of a week. Node u then computes v's ranking as follows:

$$ot_v = \frac{1}{mn} \cdot \sum_{i=1}^{m} \sum_{j=1}^{n} A^v_{i,j} \cdot (1 - A^u_{i,j}) \qquad (1)$$

Rather than assuming that online probabilities of OSN nodes follow a uniform distribution [12] and mirroring on nodes with similar online probability [26], SOUP exploits the heterogeneity of online times in OSNs [18]. Usually, the higher the probability that v is online at a time, the more preferable it is for u to use v as a mirror. However, v's online probability has to be further viewed from u's perspective, as u is looking for nodes to supplement A^u. Based on these grounds, v obtains a better ranking by Eq. (1) if v has a high probability online when u itself is not online.

However, the online-time ranking of a candidate is only a theoretical indicator of its quality, and should only be used when bootstrapping a node. A candidate, once chosen by a node as a mirror, may appear to be unwilling or unable to serve during the node's absence. It may have a limited storage capacity and has to drop some of the data it previously accepted (a node that is online most of the time can receive a lot of mirror requests and quickly fill its storage space). A malicious node could also lie about its online probability and lure others to store their data at its site (possible attacks will be discussed in Section IV).

D. Mirror Candidate Ranking in the Regular Mode

SOUP's regular mode makes use of knowledge that a node does not have during bootstrapping, but can obtain after it has gained a foothold in the OSN. As soon as a node has established social relations with some users of the OSN, it leverages the observations of these users to rank its mirror candidates.

Algorithm 2 Ranking Mirror Candidates in the Regular Mode

$M_u \leftarrow$ set of u's mirrors
$f \leftarrow$ node socially related to u
$ES_u(f) \leftarrow$ experience set of u regarding f's mirrors
$v \leftarrow$ candidate node
$KB_u \leftarrow$ knowledge base of u
for all $f \in KB_u$ **do**
  exchange $ES_u(f)$, $ES_f(u)$
end for
for all $v \in KB_u$ **do**
  calculate exp_v (Equation (2))
end for
$r_{exp} \leftarrow$ sort KB_u by exp

As illustrated in Algorithm **??** and exemplarily in Figure 1, a node u in regular mode maintains two data structures: **a knowledge base (KB)** and **experience sets (ES)**. In the knowledge base, every entry is about a node that u knows. With regard to an entry for node v, if v is a mirror of u, u would record an *experience value* of u's socially-related nodes regarding v. The experience value is the basis for ranking mirrors. In addition, the entry for v will record whether or not v is socially related to u and a TTL (time-to-live) value that decreases every time u does not choose v as a mirror (TTL not shown in Figure 1). Also, for every node w that is socially related to u, u records an experience set $ES_u(w)$. This set records u's observations of w's mirrors; that is, when requesting w's data, u records whether or not the data is available at w's mirrors.

We limit the experience set to only socially-related nodes for a number of reasons: First, users request the profiles of their friends more often than those of strangers. This way, nodes can record their experience set on the fly when requesting the data anyway. Second, tracking an arbitrary number of nodes would introduce unnecessary overhead to the system. Finally, this limitation raises the bar for malicious nodes trying to perform slander, as they have to establish social connections to their victims first.

	Node v	*sr(u,v)*	exp_v
KB_u:	4	1	-

	Friend f	*Mirror(f)*	*# of Req*	*Succ*
$ES_u(4)$				

(a) Initial State

Node v	*sr(u,v)*	exp_v
4	1	-
2	0	
18	0	
932	0	

Friend f	*Mirror(f)*	*# of Req*	*Succ*
4	18	12	8
4	**437**	**10**	**1**
4	1	10	9
4	55	11	11

(b) First Observation Recordings

Node v	*sr(u,v)*	exp_v
4	1	0.67
2	0	0.22
18	0	0.9
932	0	0.13

Friend f	*Mirror(f)*	*# of Req*	*Succ*
4	18	12	10
4	**213**	**7**	**6**
4	1	12	7
4	55	16	16

(c) After Exchange of Experience Sets

Fig. 1: Maintenance of knowledge base KB (top table) and experience sets ES (bottom table) at node u. Initially, u only knows one node (i.e., node 4 in (a)), which is also socially related to u (i.e., $sr(u,v) = 1$). As u learns about every new node, u adds it to KB_u (i.e., nodes 2, 18, 932 in (b)). For each socially related node v, node u further observes the performance of v's mirrors and records its experiences in $ES_u(v)$ (node 4 in (b)). u also receives $ES_w(u)$ from each socially related node w, allowing u to calculate the experience ranking for each node in KB_u (c). As u continues to record its own experiences for socially related nodes (c), node 4 has replaced node 437—for which u observed a bad performance—with node 213.

For every node j that is socially related to node u, u will periodically receive an experience set $ES_j(u)$ from j that includes j's observations about u's mirrors. For any mirror v, node u can then calculate v's experience value—which also serves as v's ranking—as:

$$exp_v = (1-\alpha) \cdot exp_v^{old} + \alpha \cdot \frac{1}{n} \sum_{j=1}^{n} \cdot \frac{\sqrt{o_{(j,v)}} \cdot av_{(j,v)}}{\sqrt{o_{max}}} \quad (2)$$

where $o_{(j,v)}$ is the number of observations regarding v that j is reporting since the last experience set exchange, o_{max} is the maximum number of observations that j can report, and $av_{(j,v)} \in$ [0,1] denotes the availability of v during j's requests of u's data. Eq. (2) was designed with respect to a trade-off between accuracy and security:

- *Accuracy.* u usually does not care *who* tried to access its data (especially since the exchange of observations is limited to socially related nodes). Instead, it cares for the number of attempts and successes of each reporting node in receiving its data. For instance, in retrieving u's data from v, if one node reports 100% success (9 successes out of 9 attempts) while another reports 0% (0 out of 1), v should not receive a mediocre ranking since totally 9 out of 10 attempts succeeded.
- *Security.* However, a single maliciously motivated node could falsely report huge numbers of observations, outweighing a lot of regularly observing nodes. We introduce two measures to limit the influence of a few (potentially malicious) nodes in this calculation. First, we limit the maximum number of observations, o_{max}, that j can report. Second, we extract the square root of the number of transactions to further limit the impact a single node has on the calculation of exp_v.

With exp_v computed as above, a significant portion of the recommending nodes, i.e., the socially related users to u, need to be malicious to have an impact on the selection scheme, while the experience from nodes with more observations still carries more weight.

Finally, α is the aging factor of observations, and a more recent observation carries more weight. Otherwise, a malicious node could perform a traitor attack where it obtains an excellent reputation just to exploit it afterwards [17]. In particular, such a node could offer exceptional storage capacities and online time, and thereby likely getting selected as a mirror by many users, just to disappear later. Or, the quality of a mirror could suddenly deteriorate simply because of accidental reasons such as hardware or connectivity problems. Applying the aging factor supports quick adaption to such situations.

Algorithm 3 Mirror Selection at Node u

```
M_u: set of u's mirrors, initially empty
C_u: a ranked list of mirror candidates
r_v: a candidate v's ranking value
# Select nodes from C_u
p_err ← 1
while p_err > 0.01 do
    add next top ranked element v from C_u to M_u
    p_err = p_err · (1 − r_v)
end while
# Apply social filter to nodes in M_u
# (sr(u,v)=1 if u is socially related to v; 0 otherwise.)
for all v ∈ M_u that sr(u,v) = 0 do
    if ∃v' ∈ (KB_u − M_u) such that
    sr(u,v') = 1 and r_v' · β > r_v then
        replace v with v' in M_u
    end if
end for
# Prevent starvation
add to M_u a random node v''
return M_u
```

E. Mirror Selection

Once a node obtains the rankings of mirror candidates from either mode, it selects its mirrors from the candidates, as depicted in Algorithm 3. First, it adds the top-ranked candidate nodes to its mirror set one by one, until the estimated probability of the data not being available is less than 1%:

$$p_{err} = \prod_{i=1}^{n} (1 - r_i) < 0.01, \quad (3)$$

where r_i is either the online probability ranking or the experience value of the i-th node in the candidate list, depending on

which mode the node is in. Second, SOUP further exploits the inherent OSN incentives where nodes would rather prefer to store data of a friend than a stranger. The node applies a social filter to raise the ranking values of those socially related:

$$r_v = max\left(\beta \cdot r_v, 1\right), \quad where \;\; \beta > 1. \tag{4}$$

Socially related nodes will thus move up in the ranking and can even replace some unrelated nodes as mirrors. The usage of this social incentive, however, must not be over-stretched: Evaluating different values of β shows that a friend has to provide at least 80% performance of unrelated mirrors in order to offer the best availability and overhead. Basically, socially related nodes can replace unrelated nodes only if the latter are not significantly inferior. Finally, u adds to its mirror set a random node for which it has not yet determined a ranking (e.g., a new entry in its knowledge base). This way, u could be prevented from possible overlooking of even better suited nodes.

Finally, u publishes the selected nodes to the directory service (e.g., DHT), so that users requesting u's data know where to look for it.

F. Protective Dropping

A mirror node v may not always have enough space to store the data for another node, say u. While v can simply neglect u's storage request, alternatively, v can also drop another node's data to make more space for u. Dropping will not only provide more flexibility and adapt to node dynamics, it will also enable v to choose what data to store.

In SOUP, each mirror node v implements a dropping policy that favors socially related nodes. If a miscreant orchestrates a sybil attack and floods the OSN with a large amount of storage request under many different identities, v may quickly fill up its storage space. However, as those malicious identities usually have difficulties establishing social connections to regular nodes [31], v can drop the profiles of the malicious node, leaving space for socially related nodes.

On the downside, this practice alone would discriminate honest nodes without or with few social relations, since these nodes need to rely on socially unrelated nodes. Moreover, malicious nodes can obtain social relations with some victim nodes in any case, thus able to flood the victims. Also, an attacker might still succeed when aiming at increasing the replica overhead in the system to the point where every node's storage resources are completely utilized.

Therefore, for every node w that stores its data at node v, node v calculates a dropping score, d_w, regarding w's data as depicted in Algorithm 4 (with notations given in Table I):

Symbol	*Meaning*
v	Node at which storage is exhausted
w	Node that has stored a replica at v
d_w	Dropping score for replica of node w
β	Social filter
θ	Blacklisting threshold
c	Mismatch increase (constant)

TABLE I: Protective Dropping Notations

Algorithm 4 Protective Dropping at a Node v (invoked when storage space is full)

c_v: storage capacity of v
R_v, R_u: set of replicas stored at v and u
G_v: set of nodes with social links to v
d_w: dropping score for replica of node w
M_w: set of mirrors of w
θ: ban threshold
β: social filter
c: increase of d_w in case of a mismatch
B_v: ban list at v
$d_w \leftarrow 0$:
if $data_u \rightarrow v$ and $|R_v| == c_v$ **then**
  # Compare replica sets and compute $p_d(w)$
  for all $w \in R_v$ **do**
    for all $u \in G_v$ **do**
      retrieve R_u
      if $w \in R_u$ **then**
        $d_w \leftarrow d_w + 1 \cdot \beta$
        if $u \notin M_w$ **then**
          $d_w \leftarrow d_w + c$
        end if
      end if
      # If threshold is reached, add node to ban list
      if $d_w > \theta$ **then**
        $w \rightarrow B_v$
      end if
    end for
  end for
  drop $data_w$ with highest d_w
end if
return updated R_v, B_v

- When v exchanges experience sets with each socially related node, say u, node v also learns which nodes store their data at u.
- If w also stores its data at u, we increase d_w by 1. In case w is socially related to v, however, we increase d_w by $1/\beta$ to protect the data of socially related users. If w is a flooder and tries to store its data on as many nodes as possible, d_w will then be high, and w's data will incur a high dropping probability. (Also, consider two benign nodes w, w' where w has a larger mirror set. Since v generally contributes less to the overall availability of w than to that of w', dropping the data of w has less impact than dropping the data of w'.)
- If v observes a copy of w's data in itself, but v is not listed in w's published mirror set, it increases d_w by a large constant c, as such a mismatch between the announced (e.g., published in the DHT) and the real mirror set may indicate a flooding attempt. We assume the attacker w to be sophisticated enough to announce a regularly sized set of mirrors, avoiding easy classification as an attacker when announcing hundreds or thousands of mirrors to the OSN.
- If d_w reaches a threshold θ, node v then blacklists w from

storing its data on v.

In our evaluation (Sec. V), we will show that this mechanism effectively protects SOUP against attackers who try to flood the system, and reliably identifies the attackers.

IV. Security Analysis

While in Section III we have described how SOUP balances between accuracy and security when processing experience exchange and employs protective dropping when handling malicious requests for storing data, we now further discuss how SOUP deals with malicious nodes publishing false information or compromising user privacy.

A. Countering False Information

Whether in the bootstrapping mode or the regular mode, OSN users may receive false information from an attacker. Such false information, however, can hardly cause serious damage to SOUP.

False testimony. For OSN users still in the bootstrapping mode, an attacker can distribute manipulated online time matrices to lure credulous users into storing their data at the attacker's site, leaving benign users' data availability at the mercy of the attacker. However, since the bootstrapping mode is only used for a short time upon a node's join, the impact of this attack is limited. Moreover, to be successful even to this limited extent, an attacker has to identify newly joining nodes in the first place in a time-critical fashion, which is hard in a decentralized system.

Slander. Similarly, in the regular mode, an adversary can also manipulate her experience sets. They can contradict reality by inverting the experience set score of the nodes, for example. A single forged set, however, will be outvoted by benign nodes. At the same time, collusion attacks are difficult to execute. Because SOUP makes use of the social connections between users but colluding attackers cannot establish social connections to benign users easily [31], it is difficult for them to become a majority in the recommendation system.

We evaluate the impact from both attacks further in Sec. V.

B. Protecting User Privacy

Another possibility for a miscreant is to serve as a mirror and perform tracking of accesses to the mirrored data. Leaking the resulting access patterns could compromise the accessing user's privacy. However, the gain for the attacker and the damage to the user remain questionable. First, the attacker has to operate as a benign mirror constantly to obtain and maintain a high ranking, which is costly in terms of providing resources. Second, unless the attacker is able to control *all* replicas of a user, his view of the overall data access is still limited, as he can only put his eyes on a portion of the data accesses. The attacker can thus not be sure whether or not this is representative for a user's overall access behavior. Achieving control of all replicas of a user is not easy. Initially, the attacker would need to provide a wide set of exceptionally well performing mirrors. Afterwards, it would have to get the target user to store its replicas on exactly these mirrors. This is difficult if the user already has established a stable set of mirrors, as the attacker has to rely on being randomly added to that set multiple times. This puts SOUP ahead of those related works, which rely on a single replica for which access can be easily tracked by the replicating entity [4], [7], and also of current centralized OSNs, in which the *provider itself* can track access for *all* members of the network.

We recognize that leakage of access information might still be embarrassing for a user, if the leaked information only needs to entail a binary notion of *whether or not* the user accessed the data. This can be treated as an instance of Private Information Retrieval (PIR) [11], which deals with the problem of access pattern tracking at an untrusted server and there is a variety of established approaches to countering the adversary. We will not propose our own solution here, but rather rely on promising, already existing work [23], [32], of which some (e.g., [14]) can be already set up by the user herself, if desired.

V. Simulation and Analysis

We now evaluate SOUP on a large scale with regards to the challenges listed in Sec. III-A. Based on experiments with three different real-world datasets, we show that SOUP provides high data availability with low overhead, and does so for *all* nodes in the OSN. SOUP can perform even better when altruistic nodes exist, and successfully cope with node churns and different types of malicious attacks.

A. Metrics, Data Sets, and Methodology

We first define two basic performance metrics of SOUP:

- *Data availability* at time t: The ratio of the number of users whose data is available at time t to the total number of users in the OSN.
- *Replica overhead* at time t: The average number of replicas each OSN node has at time t.

We then use these two performance metrics to measure SOUP's robustness, openness, and resiliency:

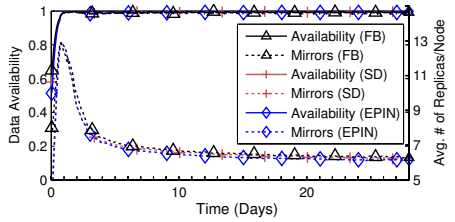
- *Robustness:* SOUP's ability to provide high performance to all nodes in an OSN, regardless of a node's online time, social relations and device capability;
- *Openness:* SOUP's ability to provide higher performance when altruistically provided resources are available; and
- *Resiliency:* SOUP's ability to maintain its performance when facing adverse scenarios.

We use three different large-scale datasets to evaluate SOUP as listed in Table II. These datasets cover a wide range of real-world social graph characteristics and can help evaluate SOUP's performance in different OSN contexts. For instance, users of SOUP should not be dependent on their number of friends, which is why we choose the Epinions dataset, which offers an average degree that is only 17% of the average degree in the Facebook dataset.

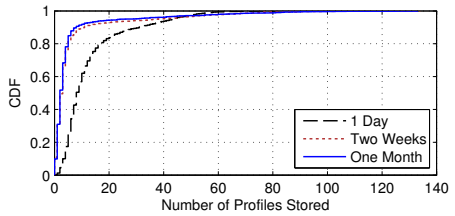
We run simulations of SOUP using these datasets, and measure the metrics defined above. In doing so, we further handle the following parameters associated with each user:

OSN	*Nodes*	*Edges*	*Avg. Degree*
Facebook [30]	90,269	3,646,662	40.40
Epinions [25]	75,879	508,837	6.71
Slashdot [20]	82,168	948,464	11.54

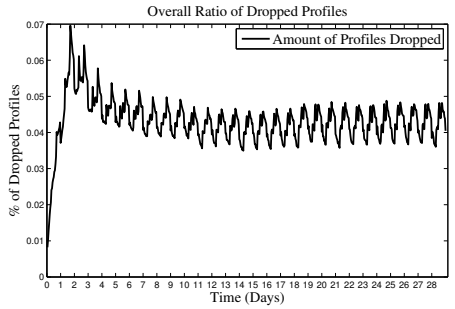
TABLE II: Datasets for SOUP Evaluation



(a) Performance



(b) Storage Distribution



(c) Dropped Profiles

Fig. 2: SOUP achieves high performance in all three data sets (a), and is stable (b,c).

- *Target error rate* ϵ. We assume every user defines her target error rate as 0.01; i.e., every user aims at a 99% likelihood of her data being available (Sec III-E).
- *Node online probability.* For every node, we must know if it is online at any given time to determine if a user's data is available at this node. Node online time is based on bursty interaction patterns of users and typically follows a power-law distribution [18], [19], [33]. We therefore assume that around 60% of the nodes are available less than 20% of the time, and there are only very few highly available nodes. Note that this power-law model incorporates the high churn rates typical for an OSN. We further apply diurnal patterns to populate the online time matrix of each node. According to [29], we consider three time zones (US, Europe & Africa, and Asia & Oceania), where a node's probability of belonging to these zones is 0.4, 0.3, and 0.3, respectively. Finally, to bootstrap the system, nodes join our experiments asynchronously according to their online probability.
- *User activity pattern.* Another parameter is how often a user accesses the data of other users. Different evolutions of OSN user activity have been observed [18], [33]. We follow [33] and model user activity to be exponentially decreasing. After an initial phase of high interaction once joining an OSN, a user's activity decreases exponentially to become less than one interaction per day. As nodes in SOUP must gain knowledge about other participants in order to find best-suited mirrors, in all the literature that we are aware of, this model represents the worst observed case for SOUP.
- *Available storage space per node.* We must assign a specific storage space value to every node in order to evaluate SOUP's storage overhead and its dropping strategy. We model the storage space available at each node as a Gaussian distribution, with a median of space for mirroring data of 50 users, which requires no more than half a gigabyte of disk space as shown in Sec. VI.

B. Results and Analysis

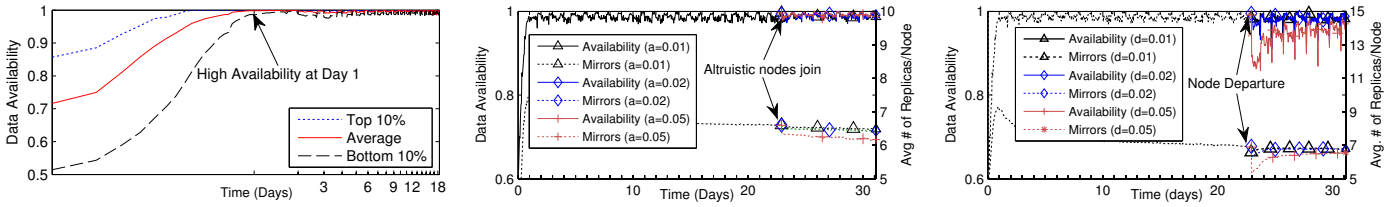
We now present the results obtained by simulating SOUP using the methodology above.

1) Data Availability and Replica Overhead: Figure 2a shows SOUP's data availability and replica overhead for each dataset. In all the three datasets, SOUP achieves the targeted availability of above 99% after only one day, even though no node has any knowledge as the simulation begins. As SOUP reaches equilibrium, the high level of availability is maintained for the entire remaining period.

Two distinct phases exist regarding the replica overhead. In the early stages, due to imprecise user experience and the lack of knowledge of good mirrors, nodes do not select well-suited mirrors yet. However, as soon as nodes obtain more precise rankings, the quality of mirrors improves, and the replica overhead is reduced by about 50%. On average each node needs to store less than 6.5 replicas.

2) Stability and Communication Overhead: SOUP needs to reach a stable state quickly in order to keep communication overhead low. In particular, if a user frequently changes her set of mirrors, all her data has to be transmitted to new mirrors often. In Figure 2a, SOUP has shown to equilibrate in its overall performance. We further show SOUP's profile distribution in Figure 2b. Here we take three snapshots throughout the simulation: after the first day, after two weeks, and after one month. After day one, many nodes need to store a medium amount of profiles so that the system achieves high availability. However, after nodes accurately measure user experiences, SOUP improves to a point where 90% of the nodes need to store less than 7 replicas after two weeks. We observe the same distribution at the end of our simulation, indicating that SOUP has reached a stable state.

Further, as mirror rankings become more accurate, the drop rate of data converges from 0.07% to a very low 0.045% as shown in Figure 2c. Hence, of 1000 profiles that are stored at mirrors, only about five are dropped. Finally, the upper half of our nodes with regards to online time provides more than 90% of all replicas. This indicates that weak nodes, in particular



(a) Availability for Different Subsets of Nodes *(b)* Join of Altruistic Nodes *(c)* Node Dynamics

Fig. 3: SOUP enables high availability for all nodes (a), is open to altruistically provided resources (b), and resilient to node dynamics (c).

mobile nodes, are rarely chosen as mirrors, saving storage, bandwidth and battery on these devices.

3) Robustness: Figure 3a shows that regardless of their own online probability, every user achieves a very high level of data availability. Also vital is the fast convergence even for seldom online nodes: After just one day, even the 10% least online users obtain data availability of above 99%. Hence, no user is discriminated based on their own online time and SOUP offers a robust OSN. We further observe that, while SOUP achieves high availability even for nodes that are almost never online, inactive, or maintain no social relations, most of the few total losses also occur for such nodes (i.e., for users not participating in the OSN at all).

4) Openness: One of SOUP's challenges is to exploit altruistically provided resources efficiently. Figure 3b shows the impact of the presence of small percentages of altruistic nodes that are steadily online. As each node is already aiming for a 99% data availability, we can observe that 5% (a=0.05) altruistic nodes can cause a slight increase and stabilization of availability, while the improvement in terms of replica overhead is more prominent. As altruistic nodes become known to the OSN, nodes do not need to select as many nodes as before to achieve the same level of availability. Thus, even with only small numbers of altruistic users—and without assuming any server nodes with high capacity—the replica overhead can already be reduced.

5) Resiliency Against Node Dynamics: In addition to high churn rates, we now consider the case in which a fraction of the users abruptly becomes unavailable to the rest of the OSN. The reasons for this can be manifold e.g., accidental network and node failures, overloaded mirrors (e.g., if hosting a very popular profile), or Distributed Denial of Service (DDoS) attacks). If we assume the top 5% of nodes in terms of online time leave the OSN simultaneously (d=0.05), we can see a noticeable drop in both data availability and replica overhead directly after the departure caused by the concomitant loss of mirrors within the OSN. However, the remaining nodes adapt quickly by choosing new mirrors, and SOUP's performance improves without introducing any additional replica overhead. Interestingly, SOUP is highly independent from the top 1-2% of nodes, as there is no significant drop in data availability when these nodes leave the OSN. Nonetheless, a specific profile might be unavailable, either when an adversary attacks its mirrors or when mirrors of popular data deny service due to overloading. In such a case, SOUP will first increase the amount of mirrors due to a reduced ranking of the current ones, and the load will be distributed among additional mirrors. If a mirror is completely taken down, SOUP will choose a different one, as shown above. Such adaptation puts SOUP ahead of the static mirror choices of related work.

6) Resiliency Against Malicious Nodes: Adversaries are not limited to DDoS attacks, but may also employ different attacks on SOUP's selection scheme as discussed in Sec. III and IV. To evaluate the resiliency of SOUP in this dimension, we assume different fractions of the OSN to be compromised. Due to increased potential to control a large number of nodes (e.g., by using cloud services), we measure SOUP's performance under attack of up to half the nodes in the OSN. To challenge our system further, our experiment is designed in the way that SOUP not only needs to tolerate the malicious nodes after reaching equilibrium, but also has to bootstrap and stabilize in their presence.

First, we examine a *false testimony* attack as specified in Sec. IV. We assume that the attacker has identified nodes which use the bootstrapping mode. Figure 4a shows that only when *all* nodes rely on the bootstrapping mode (upon joining the system), the attack has a slight impact on the replica overhead in the system. This is caused by luring regular nodes into storing data on malicious devices by manipulating online time matrices. In that case the benign nodes estimate less mirrors are needed to reach the desired target error rate, thereby reducing the replica overhead in the initial phase. However, this becomes irrelevant as soon as nodes enter the regular mode, at which point the attack is warded off.

Second, we study the impact of the *slander attack*, in which the attackers issue manipulated experience sets. We assume that the malicious users have already infiltrated the regular OSN successfully and they send out recommendations at the maximum rate. Figure 4b shows that even when 50% of social relations—and thereby experience sets—are subject to slander, the data availability drops to around 95% in the worst case. If the fraction of malicious nodes is smaller, there is only little impact on the data availability, since these nodes are outvoted by benign nodes in the selection process. Another observation is that the attack has an impact on the replica overhead of SOUP in the early stages, which is for a similar reason as within the false testimony attack: Typically, nodes do not have their most suited mirrors in their knowledge base right at the beginning of their lifetime in the OSN. Thus, the experience sets obtained from malicious nodes will obtain praising of nodes that perform badly in reality. Hence, nodes again estimate a lower required number of mirrors in this

Online Time Distr.	*Approach*	*Availability*	*Replicas*
Power-law	SOUP	99.5%	6.5
uniform, all nodes: p=0.3	Safebook	90%	13-23
	SOUP	98.5%	14
10% of nodes: p=0.95; 25% of nodes: p=0.87; 30% of nodes: p=0.75; 30% of nodes: p=0.33	PeerSoN	$<90\%$ to $\approx 100\%$ depending on node availability (frail OSN)	6
	SOUP	$\approx 100\%$	4

TABLE III: SOUP compared to related work (p = online probability).

phase.

Finally, we investigate a *flooding* attack, in which an attacker creates multiple identities (i.e., sybils) and tries to flood the benign users with data. We show the results for different percentages of sybils in Figure 4c. Even in our worst measured case with as many sybil identities as regular identities in the OSN, the data availability does not drop below 90% for the benign users in the long run, and the replica overhead, although increased, is still reasonable at around 13 copies per node. This is due to protective dropping. First, it prevents data of socially connected nodes from being dropped for a sybil's data. Second, it prevents the full utilization of resources at regular nodes. In addition, SOUP can detect attackers with time passing. With an increasing number of attackers being identified, SOUP is able to recover from the attack. Hence, although storing some copies of malicious nodes, even a flooding attack with half of the identities in the OSN being malicious can be tolerated quite well.

7) SOUP vs Related Work: SOUP's improvements over state-of-the-art solutions such as PeerSoN and Safebook [12], [26] mainly lie within its qualitative properties, which we extensively evaluated above. Compared against PeerSoN and Safebook, SOUP is robust, adaptive to node dynamics (e.g., it can dynamically select mirrors), and resilient against attacks.

To further quantitatively compare the performance of SOUP and related work, we run simulations of SOUP under the node online time distributions assumed in related works. As shown in Table III, SOUP outperforms both PeerSoN and Safebook, providing higher data availability and lower replica overhead. In particular, when compared with Safebook, SOUP achieved 8.5% higher availability while keeping the replica overhead near the lower bound of Safebook. We observe that in this scenario SOUP performs slightly worse than in our original experiments. This is caused by the uniform online time distribution, due to which SOUP cannot exploit the heterogeneity of node characteristics to select well-suited mirrors. In the PeerSoN scenario, the online times of nodes are drastically improved over our power-law assumption. Still, PeerSoN is not able to provide a robust OSN and the data availability ranges between less than 90% and close to 100%, as nodes depend on their own online times. SOUP however provides close to 100% data availability for *all* nodes and further reduces the replica overhead by one third.

VI. Deployment and Experiments

We now provide experimental results of SOUP based on a small-scale deployment of a prototype. We measure SOUP's bandwidth and storage overhead at highly frequented mirrors. As a side result, we are able to catch a first glimpse of the ground truth of real-world profiles in popular OSN services.

A. Prototype

The SOUP algorithms are generic and can be applied to decentralized OSNs of any kind. As a proof of concept, we have incorporated SOUP into our own decentralized OSN prototype [5]. The prototype acts as a P2P social networking overlay that resides as a middleware between arbitrary social applications and the networking stack, [1]. and includes functionalities based on the FreePastry 2.1 DHT [3] for operating the OSN such as replica synchronization and lookup functionality.

B. Dataset and Setup

Because a user may only request certain *items* (e.g., the wall of the user or a picture) in a profile, rather than the entire profile, we first obtained information about the size of items within a profile. One could use crawling to do so, but a crawler's results often do not include major parts of a user's data [7]. For instance, photo uploads on Facebook are not publicly available by default. To bypass this limitation, we asked Facebook and Google+ users for permission to access their data, and were able to obtain 20 complete profiles.

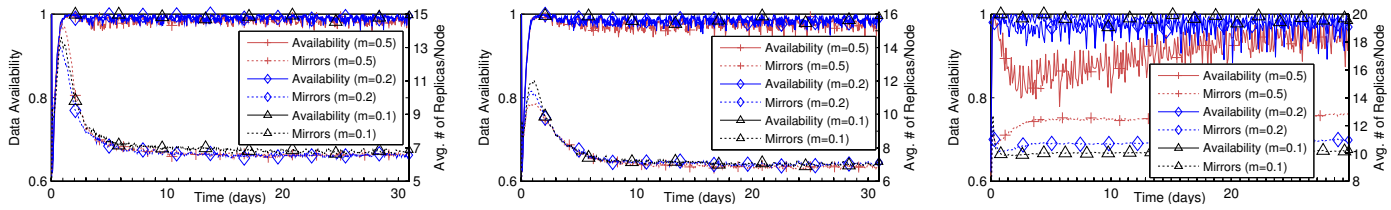
The average profile size was $\approx$10 MB, with the largest profile containing 27 photo albums and 3 videos and using totally 60 MB of disk space. Moreover, the data disclosed 2035 unique items to us. As shown in Figure 5a, most items are small and large items rarely exist. More than 35% of all items are less than 10 KB in size, and 93%—including most images—are less than 100 KB in size. Coupled with the findings of [27], where photos (17%) and videos (1%) account for only a small fraction of traffic on Facebook, this result is encouraging to SOUP in terms of storage or bandwidth consumption.

We measure SOUP's storage and bandwidth consumption by deploying our prototype to regular workstations and Amazon EC2 [2] virtual machines. Throughout our experiments, one of our workstations is permanently online and attracts all other profiles quickly.

C. Results

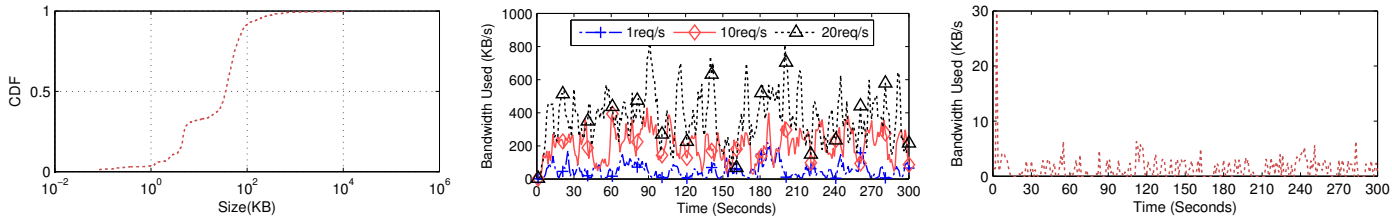
1) Storage Overhead: Our first investigated metric is the storage overhead at the mirrors. We observe the permanently online node, which mirrors all 20 profiles—three times of what 90% of SOUP nodes will have to handle. Still, the data sums up to 206 MB of consumed space, which is both surprisingly low and easily manageable, even for nodes with little resources. In fact, for 90% of all nodes, SOUP will only use as little as 70 MB of disk space.

[1]Source code and further details available at http://gemstone.informatik.uni-goettingen.de.



(a) False Testimony Attack *(b)* Slander Attack *(c)* Flooding Attack

Fig. 4: System performance under different attacks, where 10%, 20% and 50% of the nodes act maliciously (m=0.1,0.2,0.5).



(a) Item Distribution *(b)* Data Traffic *(c)* Control Traffic

Fig. 5: Evaluation of our item set (a), and both bandwidth consumption (b) and control overhead (c) introduced by our prototype.

2) Bandwidth Consumption: We model the data requests of different types of items according to [27], in which photos make up for 13% of requests and videos for 0.5% of requests. Interestingly, except for a few spikes, the average consumption is well below 600 KB/s, even if the mirror has to handle 20 requests per second, as shown in Figure 5b. Unsurprisingly, as the request frequency increases, the likelihood to hit a large item (i.e., a video) increases, which causes the spikes in our measurements. As a result, a request might time out once a mirror becomes overloaded or limits its bandwidth for sharing, which may happen especially to nodes mirroring popular data. Recall that unlike other approaches where users are stuck with a static set of mirrors [12], [26], SOUP will adapt to this situation when selecting mirrors. Figure 5b also shows that lower request rates do not congest typical broadband connections.

3) Control Overhead: The control overhead that is required to maintain the DHT overlay turns out to be easily manageable. As shown in Figure 5c, it consumes—with the exception of the initial node join—below 5 KB/s. This result is consistent with existing studies regarding the control overhead of large-scale DHT traffic even under high churn [24].

4) Latency and Processing Overhead: We also measure the *application level* latency that our clients experience when requesting the data from the mirror, which is under full load (12 requests per second). Regardless of the location, the request latency is $2 \cdot RTT + \epsilon$, where 1 RTT is due to TCP connection establishment and $\epsilon < 2ms$ is the actual processing overhead introduced by SOUP.

VII. Discussions

Recap: Does SOUP Meet All the Design Challenges?

In SOUP any OSN node can check as many mirror candidates as needed and select some or even all of them as mirrors. As every node does this regardless of its capability or social relations, SOUP is able to make data of *any* node highly available (i.e. *robustness*). As every node usually only interacts with a small number of nodes (e.g., friends or mirrors), the overhead is kept low and SOUP is also *scalable* in this sense.

Furthermore, SOUP exploits node heterogeneity in its mirror selection. A node can quickly select some good enough nodes as mirrors initially, then leverage the experience from socially related nodes to identify best mirrors. It can also receive updates of such experience, thus staying *resilient* to node dynamics as it can reselect its mirrors if needed. Altruistic nodes are thus likely to be chosen, but mobile nodes are not. A node is also more likely to choose a socially related node as its mirror, which in turn is less likely to drop data of its friends than data of strangers. Finally, SOUP's design supports security, as we elaborated in Section IV.

Social relations in SOUP

In SOUP, social relations not only provide incentives to store data, but also play a role when selecting mirrors, deciding which data should be dropped, or filtering out malicious users and limiting their impact. However, the expressiveness of social relations within an OSN is questionable. Whereas research has shown that these relations can be exploited in various ways (e.g., to defend against sybil attacks [31]), it remains unclear to which extent the binary model of social relations reflects the real world. Recently, there have been approaches to breaking this model and studying the effect of more expressive social relations [28]. Leveraging a multitude of social relations (e.g., of different relation nature or sensibility) between users may provide an opportunity to further improve the performance, stability and resiliency of SOUP. For instance, when selecting mirrors, SOUP could assign higher weights to the experience sets of more closely related users, and assign low weights to that of the loosely related acquaintances. Same with protective dropping, where data of closer friends could be more secured from being dropped. Another option could be allowing users to mark some participants as foes (as in the Slashdot OSN),

thereby declaring they are not willing to store data for these users.

VIII. Conclusion

This paper presented SOUP, a generic approach to storing user data in a decentralized OSN. SOUP comprises multiple mechanisms, which, based on exchanging experiences among socially related nodes, effectively distribute data replicas to mirror nodes throughout the OSN. While it effectively addresses the severe privacy concern from centralized OSNs, SOUP's performance is close to such OSNs. Albeit only relying on the resources provided by the OSN users themselves, SOUP provides high data availability and low replica overhead. Thanks to its stability, SOUP introduces only little additional communication cost. Furthermore, it meets a number of challenges that distinguish the system from existing works: SOUP offers a robust OSN as a whole and has proven to be adaptive to altruistically provided resources and node dynamics, and resilient against malicious activities. At the same time, SOUP is deployable in the scenario of large scaled OSN, as it scales with large numbers of users and converges to a stable state quickly. Finally, we investigated profile sizes of real-world OSNs and showed that SOUP, deployed in a real-world prototype, is feasible.

References

[1] http://blog.linkedin.com/2012/06/06/linkedin-member-passwords-compromised/.

[2] http://aws.amazon.com/ec2/.

[3] FreePastry 2.1. http://www.freepastry.org, 2009.

[4] The Diaspora Project. https://joindiaspora.com/, 2010.

[5] The GEMSTONE Project. https://gemstone.informatik.uni-goettingen.de/, 2011.

[6] Androutsellis-Theotokis, S., and Spinellis, D. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Comput. Surv. 36* (December 2004), 335–371.

[7] Baden, R., Bender, A., Spring, N., Bhattacharjee, B., and Starin, D. Persona: An Online Social Network with User-defined Privacy. In *SIGCOMM '09*, ACM.

[8] Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Looking up Data in P2P Systems. *Commun. ACM 46* (February 2003), 43–48.

[9] Benevenuto, F., Rodrigues, T., Cha, M., and Almeida, V. Characterizing User Behavior in Online Social Networks. In *IMC '09*.

[10] Bethencourt, J., Sahai, A., and Waters, B. Ciphertext-Policy Attribute-Based Encryption. In *IEEE SP '07*.

[11] Chor, B., Goldreich, O., Kushilevitz, E., and Sudan, M. Private Information Retrieval. In *Foundations of Computer Science'95*, pp. 41 –50.

[12] Cutillo, L., Molva, R., and Strufe, T. Safebook: A Privacy-Preserving Online Social Network Leveraging on Real-life Trust. *Com. Mag., IEEE 47*, 12 (2009), 94–101.

[13] Debatin, B., Lovejoy, J. P., Horn, A.-K., and Hughes, B. N. Facebook and Online Privacy: Attitudes, Behaviors, and Unintended Consequences. *Journal of Computer-Mediated Communication 15*, 1 (2009), 83–108.

[14] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The Second-Generation Onion Router. In *USENIX Security'04*.

[15] Douceur, J. R. The Sybil Attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (2002).

[16] Dwyer, C. Privacy in the Age of Google and Facebook. *Technology and Society Magazine, IEEE 30*, 3 (2011), 58 –63.

[17] Feldman, M., Lai, K., Stoica, I., and Chuang, J. Robust Incentive Techniques for Peer-to-Peer Networks. In *EC'04*, ACM.

[18] Gyarmati, L., and Trinh, T. Measuring User Behavior in Online Social Networks. *IEEE Network 24*, 5 (2010), 26 –31.

[19] Jiang, J., Wilson, C., Wang, X., Huang, P., Sha, W., Dai, Y., and Zhao, B. Y. Understanding Latent Interactions in Online Social Networks. In *IMC'10*.

[20] Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. *CoRR* (2008).

[21] Liu, D., Shakimov, A., Caceres, R., Varshavsky, A., and Cox, L. P. Confidant: Protecting OSN Data without Locking it Up. In *Middleware '11*.

[22] Mislove, A., Marcon, M., Gummadi, K. P., Druschel, P., and Bhattacharjee, B. Measurement and Analysis of Online Social Networks. In *IMC '07*.

[23] Mittal, P., Olumofin, F., Troncoso, C., Borisov, N., and Goldberg, I. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *USENIX Security'11*.

[24] Rhea, S., Geels, D., Roscoe, T., and Kubiatowicz, J. Handling Churn in a DHT. In *USENIX ATC'04*.

[25] Richardson, M., Agrawal, R., and Domingos, P. Trust Management for the Semantic Web. In *ISWC'03*.

[26] Rzadca, K., Datta, A., and Buchegger, S. Replica Placement in P2P Storage: Complexity and Game Theoretic Analyses. In *ICDCS '10*.

[27] Schneider, F., Feldmann, A., Krishnamurthy, B., and Willinger, W. Understanding Online Social Network Usage from a Network Perspective. In *IMC'09*.

[28] Tang, S., Yuan, J., Mao, X., Li, X.-Y., Chen, W., and Dai, G. Relationship Classification in Large Scale Online Social Networks and its Impact on Information Propagation. In *INFOCOM'11*.

[29] Verde, A. Facebook Demographics. http://www.slideshare.net/amover/facebook-demographics-2011.

[30] Viswanath, B., Mislove, A., Cha, M., and Gummadi, K. P. On the Evolution of User Interaction in Facebook. In *WOSN'09*.

[31] Viswanath, B., Post, A., Gummadi, K. P., and Mislove, A. An Analysis of Social Network-based Sybil Defenses. In *SIGCOMM'10*.

[32] Williams, P., Sion, R., and Carbunar, B. Building Castles out of Mud: Practical Access Pattern Privacy and Correctness on Untrusted Storage. In *CCS'08*.

[33] Wilson, C., Boe, B., Sala, A., Puttaswamy, K. P. N., and Zhao, B. Y. User Interactions in Social Networks and their Implications. In *EuroSys'09*, ACM.