SOUP: An Online Social Network By The People, For The People

David Koll
University of Goettingen
Goettingen, Germany
koll@cs.uni-goettingen.de

Jun Li
University of Oregon
Eugene, OR, USA
lijun@cs.uoregon.edu

Xiaoming Fu
University of Goettingen
Goettingen, Germany
fu@cs.uni-goettingen.de

ABSTRACT

Concomitant with the tremendous growth of online social networking (OSN) platforms 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 user data at individual end-user machines across the OSN.

However, such an approach must address new challenges. In particular, it must achieve high availability of the data of *every* user with minimal replication overhead and without assuming *any* permanent online storage. At the same time, it needs to provide mechanisms for encrypting user data, controlling access to the data, and synchronizing the replicas. Moreover, it has to scale with large social networks and be resilient and adaptive in handling both high churn of regular participants and attacks from malicious users.

While recent works in this direction only show limited success, we introduce a new, decentralized OSN called the Self-Organized Universe of People (SOUP). SOUP employs a scalable, robust and secure mirror selection design and can effectively distribute and manage encrypted user data replicas throughout the OSN. An extensive evaluation by simulation and a real-world deployment show that SOUP addresses all aforementioned challenges.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer Communication Networks—*Distributed Systems*

Keywords

Online social networks; OSN; decentralized OSN; privacy

1. INTRODUCTION

As online social network (OSN) providers deal with tremendous amounts of user information, they can obtain a deep insight into their users' personal interests, opinions, and social relationships, which raises severe privacy and security concerns. Facebook, for example, has long reached the one billion user mark and already controls the personal data of approximately one sixth of the world's population. Yet, the hunt for user data is not over and users are often helpless when faced with changes, as demonstrated by Facebook's recent acquisitions of the Instagram and WhatsApp user bases with multi-billion-dollar agreements. Moreover, centralized control often results in the misuse of user data [1]. For example, LinkedIn leaked millions of its users' passwords [2], and Facebook passed sensitive shopping information of users to the public without their consent [3].

While it remains to be seen whether OSN providers would give up a major source of income and grant comprehensive security and privacy means to their users (e.g., encryption of data), decentralized OSNs (DOSNs) are becoming more promising for better user data security and privacy. Instead of relying on a central data repository, a DOSN can allow users to regain control over their data.

Recently, researchers have proposed a wide range of DOSN approaches, which considerably differ in how the central data repository is substituted and how the user controls her data [4–13]. However, each of these systems suffers from multiple shortcomings, including (i) limited success in providing high data availability [7, 9, 11–13]; (ii) discrimination of users based on their dependency on other nodes [6, 7, 11–13]; (iii) dependency on powerful nodes [4–7]; (iv) high overhead [10–13]; (v) low adaptivity to OSN dynamics [4, 6–13]; (vi) susceptibility to adversaries [4, 6, 7, 9–13]; (vii) a lack of data encryption [4, 7, 9]; (viii) non-consideration of mobile users [6, 7, 9–13]; and (ix) technical feasibility and economical deployability issues [5, 6, 8, 13].

Motivated by the absence of a full-fledged DOSN, we introduce a different approach called the Self-Organized Universe of People (SOUP). In addressing the drawbacks of existing works, we make the following contributions:

- To achieve high data availability, we propose a new, generic approach to storing user data in a DOSN. While every user can store her data at her own machine, she can rely on a scalable, robust and secure mirror selection design to select other OSN participants as mirrors for her data and make the data highly available, even if she herself may not be always online. SOUP is able to synchronize the replicas stored at the mirrors and keep them up-to-date on possibly multiple devices of a user. It does not rely on permanently available or altruistically provided storage, although it can make an opportunistic use of such resources as they become available.
- To provide a robust OSN, SOUP ensures that regardless of participants' social relations or online probabilities, data for *all* participants is highly available. Such a property is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Middleware'14, December 08–12, 2014, Bordeaux, France.
Copyright 2014 ACM 978-1-4503-2785-5/14/12...$15.00.
http://dx.doi.org/10.1145/2663165.2663324.

essential not only to not discriminate any user, but also to enable access to all data of interest at any time.

- To limit the overhead, SOUP ensures that there exist only as many replicas as required, and stabilizes the mirror set so that replicas are not frequently moved.
- To achieve reliability and resiliency, SOUP is designed to be adaptive to the dynamics often seen in a DOSN and it can quickly respond to changes in the system and continue to provide high performance. Moreover, its operation is not significantly affected by malicious OSN users, as it can tolerate up to half of the identities in the OSN being controlled by an adversary.
- To grant data privacy, effective encryption mechanisms ensure that only eligible users can access encrypted data.
- To support mobile users, SOUP is designed mobile-friendly as it minimizes data transfer and resource consumption on mobile nodes. Moreover, even with a high churning rate of mobile nodes, SOUP is stable and performs well.
- To demonstrate its feasibility, we run extensive simulation experiments with three different real-world datasets to show that compared with related work, SOUP does provide superior performance in all aforementioned features.
- Finally, to show its deployability we implement SOUP on both desktop and mobile platforms and investigate its performance in a real-world deployment.

The remainder of this paper is structured as follows. After conducting a comprehensive review of related work in Sec. 2, we describe the design of SOUP in Sec. 3, and devote the entire Sec. 4 to a critical component of SOUP, the mirror selection. We present an extensive, simulation-based evaluation of SOUP in Sec. 5. We then describe our implementation in Sec. 6, and evaluate SOUP based on an experimental deployment in Sec. 7. Finally, we discuss open issues in Sec. 8 and conclude our paper in Sec. 9.

2. RELATED WORK

In the course of decentralizing OSNs, classical P2P or distributed data storage approaches (e.g., [14–17]) might offer a solution. However, these approaches are generally designed for supporting traditional decentralized applications such as file sharing, which are often characterized by long durations of user online time, typically spanning from multiple hours up to days. Orthogonal to such applications, users' online patterns in social networks show high activity peaks with large gaps of offline time [18, 19]. Additionally, content in social networking platforms is often uploaded and accessed from mobile devices that may be disconnected most of the time. Further, there exist inherent relations between the OSN users, which can imply storage incentives among them and discourage freeriding. Intuitively, a user will prefer to store the data of a friend to that of a stranger. Furthermore, in contrast to traditional P2P systems, tit-for-tat strategies are not as desirable for OSNs. Users rather need the OSN *as a whole* to be robust, with each user's data accessible at any given time. Otherwise, even highly contributing users may find data of interest unavailable.

With these reasons in mind, researchers have suggested a wide range of solutions that are specifically tailored to decentralize OSNs: The first approach is to distribute data control and storage to a limited number of permanently online storage locations [4–7]. The storage might either be altruistically provided (as e.g., in Diaspora [4] or SuperNova [7]) or based on economic incentives such as user payments (as e.g., in Vis-a-Vis [5] or Confidant [6]). 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, 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 altruistic and paid servers is also a concern, as data loss can occur when such servers become disengaged abruptly. Further, in Diaspora and SuperNova, since users are not able to encrypt their data, full privacy of data is not achieved; the danger of misusing user data is shifted from one central provider to several quasi-central providers.

The second approach is to ask each user to provide a permanently available storage space for their own encrypted profile [8]. 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 incentivizing storage and configuration providers, which however results in monetary costs for the user.

The third approach is to let nodes cooperate and provide temporarily available storage to each other [9–13]. SOUP is designed to follow this rationale as well. With the mutual cooperation of nodes and flexible data storage locations, users can be independent of dedicated 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 [9] 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 time of less than eight hours a day achieve less than 90% availability for their data. Since online time in OSNs is power-law distributed [18, 19], the majority of users are unable to make their data highly available, and even highly contributing users may not be able to find data they want.

Cachet [10] replicates the data of users within a distributed hash table (DHT). While this approach ensures availability, it also increases the communication overhead. As OSNs usually experience high churn rates [19], data often has to be transferred from departing nodes to other DHT members. This is particularly the case for mobile nodes. Also, Cachet does not minimize the number of replicas, which increases the overhead to keep all replicas of a user's data up-to-date.

Safebook [11], MyZone [12] and ProofBook [13] mirror each user's data at a subset of their direct friends. Unfortunately, a user thus depends on her social contacts for data storage, as she needs enough suitable friends that qualify as a mirror. This is difficult for many users in OSNs who maintain few social links [20]. As a result, such systems typically achieve low data availability rates (e.g., 90% in [11, 12]).

Finally, none of the above schemes *explicitly* consider mobile (i.e., smartphone) devices, which have become one major way of using OSNs. Approaches that do not require every node to contribute resources (e.g., Diaspora) can tolerate such devices. However, the majority of solutions require mobile nodes to perform as regular nodes, which can be inefficient due to the limited capabilities of mobile devices.

We summarize the features offered by existing DOSNs in Table 1. As discussed before, none of SOUP's competitors

Approach	High Data Availability	Robustness / No Discrimination	Independency of Powerful Nodes	Low Overhead	Adaptivity	Resiliency	User Data Privacy	Explicit Mobile Awareness	No Cost for Users
Diaspora [4]	✓	✓	✗	✓	✗	O	✗	-	✓
Vis-a-Vis [5]	✓	✓	✗	✓	O	O	O	-	✗
Confidant [6]	✓	✗	✗	✓	✗	✗	O	✗	✗
SuperNova [7]	✗	✓	✗	✓	✗	✗	✗	✗	✓
Persona [8]	✓	✓	✗	✓	✗	O	✓	-	✗
PeerSoN [9]	✗	✗	✓	✓	✗	✗	✗	✗	✓
Cachet [10]	✓	✗	✓	✗	✗	✗	✓	✗	✓
Safebook [11]	✗	✗	✓	✗	✗	✗	✓	✗	✓
MyZone [12]	✗	✗	✓	✗	✗	✗	✓	✗	✓
Proofbook [13]	✗	✗	✓	✗	✗	✗	✓	✗	✗
SOUP	✓	✓	✓	✓	✓	✓	✓	✓	✓

Legend: ✓ feature supported; ✗ feature not supported; O feature partially supported; - feature not applicable

Table 1: DOSN Approaches Summarized.

can provide all the features required for the operation of a full-fledged DOSN. In fact, each solution has deficiencies in multiple categories, whereas SOUP supports all features.

3. SOUP

We now present the design of the Self-Organized Universe of People (SOUP), and show how the system provides the critical features necessary to operate a competitive DOSN.

3.1 SOUP Overview

In SOUP, every participating node (or user) maintains its own data, and selects a small set of other nodes as **mirrors** to store a replica of its data, in order to keep its data available even when it is offline. Data replication is necessary, because SOUP does not rely on a central repository and users are not permanently online.

Every SOUP node comprises the *SOUP middleware* and the *SOUP applications*. The SOUP middleware resides between the network stack and SOUP applications. Its main functions include (i) to organize SOUP nodes into a structured overlay; (ii) to handle mobile nodes; (iii) to ensure user data privacy; (iv) to maintain and synchronize user data; and (v) to establish communication channels with other SOUP nodes. Multiple SOUP applications can run concurrently on top of the middleware, each of which manages the node in a different social network for the same user. The SOUP middleware provides a generic API to SOUP applications. For instance, once the data of the user is changed, the middleware can notify all running applications, so they all have the most recent version of the user's data.

Our focus in this paper is on the SOUP middleware, and we revisit SOUP applications in Sec. 6 and 7, where we show the implementation of a SOUP node and a SOUP-based distributed OSN, respectively. We present the main middleware functions in this section, and describe in detail how SOUP selects mirrors in Sec. 4. Recall a key contribution of SOUP is selecting nodes as mirrors while addressing critical challenges (Sec. 1).

3.2 SOUP Overlay

Nodes in SOUP form a structured overlay, as shown in Fig. 1. The overlay acts as a globally searchable *information directory* and is based on a distributed hash table (DHT). The DHT enables efficient publish and lookup operations in a decentralized fashion, making a centralized information repository unnecessary. Every SOUP user can publish her

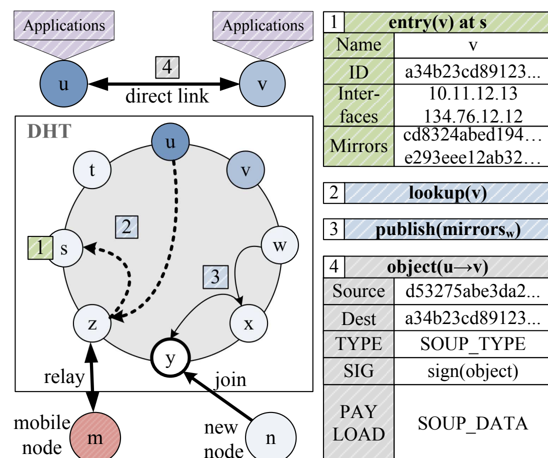


Figure 1: SOUP Overlay

directory entry at the node that is responsible for her ID in the DHT key space (e.g., v's entry is published at s—Step 1 in Fig. 1), and any other node can locate the node to retrieve the entry (e.g., u can look up v's ID—Step 2). An entry typically contains a user's name, her SOUP ID, the interfaces (i.e., IP addresses) via which she can currently be contacted, and the SOUP IDs of all the mirrors of her data. Here, the **SOUP ID** is a 64-bit SHA-256 hash over the user's 1024-bit public key and uniquely identifies the user.

It is important to note that in contrast to some related work [10], a user only publishes pointers to mirror nodes (i.e., SOUP IDs) in the DHT (e.g., w publishes her mirrors at y—Step 3), whereas the data themselves are stored among nodes themselves. Directly storing data in the DHT would have undesirable consequences: First, every user would have no control over which other nodes will be her mirrors to host her data, whereas the mirrors would have no option to reject unwanted data. Second, it would increase the overhead of the system; whenever a node departs—which can be often since SOUP nodes may have a high churning rate—it has to transfer all its DHT data to another node.

SOUP incorporates a list of publicly known *bootstrapping nodes* to help new nodes join SOUP. A bootstrapping node is simply a regular node enhanced with a function to bootstrap others: a new node can use such a bootstrapping node as its entry point to the DHT, thus adding itself to the DHT; for example, in Fig. 1, node n joins the DHT via a bootstrapping node y. It can then prepare its entry (including looking up

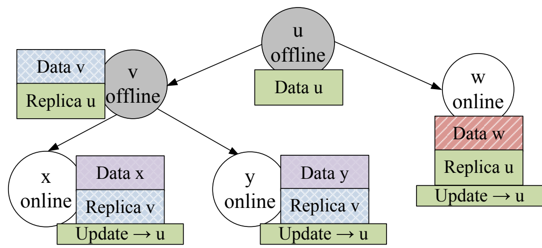


Figure 2: SOUP's Replica Management

its own SOUP ID in the DHT to make sure there is no collision with another user's SOUP ID), and publish it to the DHT, which enables other nodes to look up the entry.

3.3 Mobile Nodes

SOUP is designed to be friendly to mobile nodes. As these devices often experience high churn (e.g., because of connectivity changes) and long response times (e.g., due to limited bandwidth), they can decrease the performance and stability of the DHT overlay. SOUP addresses this challenge by exempting mobile nodes from the DHT. Instead, a mobile node will relay its DHT publish and lookup operations through a *gateway node* that is on the DHT (e.g., node m will relay through node z in Fig. 1). As doing so frees mobile nodes from directly executing DHT operations (e.g., shifting entries), it also saves resources on the mobile devices.

A mobile node initially uses a bootstrapping node as its gateway (the same node it contacted to join SOUP). However, every time it encounters another node, it checks that node's ability to relay DHT requests (every regular node can set a limit to mobile connections) and switches to that node as a gateway if possible to reduce the load on bootstrapping nodes (e.g., node m has switched from node y to z in Fig. 1). Note that since data itself is not stored in the DHT, the relayed requests do not consume a lot of bandwidth.

3.4 Data Privacy

To ensure the confidentiality of all privacy-relevant user information, every SOUP user encrypts all her data using Attribute Based Encryption (ABE) [21], then distributes one replica to each mirror. The encryption routine only introduces a limited overhead, even for mobile devices [8]. In ABE, the symmetric key for encrypted content is protected by an Access Structure, which is defined by a combination of attributes, so that only requesters holding the correct attribute key can decrypt it. This allows a user to grant fine-grained access to her confidential data, as it cannot be accessed by other entities except those holding the corresponding attribute keys. For instance, the user can limit access to one item to users holding two specific attributes, while three different attributes are required for another item. The attributes themselves can be arbitrary (e.g., such as *colleague* or *lives in my city*). In particular, the mirrors themselves cannot access the data stored at their premises without holding the correct attributes. Note that requests to modify any data must be encapsulated in an appropriately (i.e., with the owner's asymmetric private key) signed SOUP object, and will otherwise be discarded.

3.5 Data Synchronization

A user may receive updates from other users (e.g., messages to the user). Depending on the content, an update might require the user to alter her data. If the user is online herself, she can directly receive updates, order them based on the timestamps included in the received SOUP objects, and alter her data accordingly. If the user is offline (e.g., u in Fig. 2), she then needs assistance from her mirrors (e.g., v and w in Fig. 2). The mirrors act as a surrogate by receiving and storing updates to the data, which can then be collected and ordered by the returning online user later. Note that the mirrors themselves are not eligible to modify the user's data. As u is offline, updates for u have to be stored at u's mirrors, v and w. Mirror v itself is also offline, so that updates for u (not the whole replica of u) have to be further passed on to v's mirrors x and y. SOUP is designed such that at least one mirror of each user is online at any time (Sec. 4) and can retrieve these updates. Hence, v can retrieve any updates to u's data upon returning online from its own mirrors. Hereby, all mirrors always present the most recent user data if they are online, which also enables the data owner to synchronize different personal devices.

3.6 SOUP Communication

To request data, a user establishes a connection with another user in two steps: First, she looks up the entry of her communication partner in the DHT (e.g., u looks up v's entry in Fig. 1; m would do so via its gateway z). Afterwards, she extracts the partner's addressable interfaces from the entry and creates a direct communication channel. This channel can be based on any networking protocol, ranging from standard TCP/IP to Bluetooth, if available.

Once a communication channel is established, the communication partners (u and v in Fig. 1) exchange signed *SOUP objects*, which can contain arbitrary information (Step 4 in Fig. 1). Applications running on top of SOUP can encapsulate payload (such as user data or friend requests) into SOUP objects, and thereby exchange content transparently via the middleware. The transparency allows the development of any kind of OSN application on top of the middleware.

4. MIRROR SELECTION

A major component of SOUP is its scalable, robust and secure approach to selecting mirrors for storing user data among heterogeneous nodes in a DOSN with very high availability. We first outline the challenges we face, and then describe our decentralized mechanisms that address them.

4.1 Challenges

To fulfill its promise to eliminate the drawbacks of existing DOSNs, SOUP must address a variety of key challenges, that have not been solved in their entirety before. The primary challenge is to achieve high data availability that is very close to that of centralized OSNs while decentralizing the control and storage of the data.

In leveraging resources of participating nodes, SOUP must address the heterogeneity of OSN users. Besides a variety of 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 [18, 19, 22]. Moreover, SOUP must recognize that the users' machines are not servers and usually only offer 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 addi-

tion, our selection scheme should be open to altruistically provided resources and exploit them if available.

Also, our approach should exploit the potential of social relations within the OSN. In particular, users can provide each other feedback whether they succeeded or failed in obtaining data from a participating node. If utilized properly, such cooperation can help every user to distribute their data within the OSN more efficiently.

Finally, SOUP's selection mechanism must have several key properties: (i) 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 many 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 to each joining node. (ii) 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, those nodes who need to access its data may find it unavailable. (iii) It must cope with every possible adverse situation. For example, an OSN can have a high churn rate due to short-lived sessions [18, 23]; a large fraction of nodes may depart the system at the same time due to a network failure; worse, malicious users in an OSN can launch all kinds of attacks. SOUP must continue to offer high performance in all such unfavorable scenarios.

4.2 Mirror Selection Overview

SOUP has 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 a *replica*—is available at another node—called a *mirror*. The core task for SOUP is thus mirror selection: every OSN node needs to select the most eligible nodes as mirrors before it places its data replicas there.

Every node employs two modes to select its mirrors, a *bootstrapping* mode and a *regular* mode. When a node joins the OSN and has no knowledge about it, it runs in the bootstrapping mode, which allows it to gain a foothold in the OSN; it obtains recommendations from each node it encounters and ranks mirror candidates based on this information (Sec. 4.3). Once a node befriends others, it begins to learn from them about their experience in accessing its data at its mirrors, and transitions to the regular mode; it will now rely on friend experience to rank mirror candidates (Sec. 4.4).

The two modes differ in their way of ranking mirror candidates, but follow the same routine for selecting mirrors (Sec. 4.5). Here, a node will primarily consider that the higher a candidate is ranked, the more likely it will make the node's data available. Note it is more so with the regular mode when direct user experience is used for ranking, as opposed to looking at strangers' recommendations in the bootstrapping mode. SOUP further allows every node to dynamically select as many mirrors as needed. As a result, no matter whether a node itself is online a lot or not, and no matter whether it has many friends or just a few, as long as it has enough quality mirrors via SOUP's algorithms, its data will be highly available through those mirrors.

SOUP leverages social relationships in the mirror selection process primarily through experience exchanges: node u's experience in accessing node w's data via w's mirror v helps w decide if v is a good mirror. But social relationships can be useful in other contexts as well. Since friends have more incentives and higher trust to store data for each other, a node assigns a higher weight to friend candidates when selecting mirrors, and protects profiles of friends when dropping data from its storage.

Dropping data may be necessary if a node is chosen as a mirror by many nodes, and its resources are exhausted. A dropping strategy is critical, especially when an adversary is flooding the OSN and many nodes receive numerous malicious requests to store data. For this task SOUP employs a *protective dropping* mechanism (Sec. 4.6).

4.3 Mirror Candidate Ranking in the Bootstrapping Mode

SOUP allows new nodes to quickly achieve high data availability. At the time a user joins the OSN, she does not possess any information about well-suited mirrors. However, as she contacts other nodes, these nodes can suggest such mirrors to the new node. Here, we exploit that OSN users are most active when they have just joined, and they contact many other nodes [22]. In particular, every time a new node u contacts a node v, v suggests the set of mirrors that works well for itself to u. If u cannot obtain any recommendations, she will randomly select mirrors from her contacts.

However, a user should not use the bootstrapping mode for too long. A mirror w suggested by v might not be a good choice for u for various reasons. Node behaviour in OSNs is heterogenic (e.g., w.r.t. online time [18]) and w is probably not the best fitting node for u. Moreover, w might not be willing to store data for u in the first place, or an attacker could fake recommendations to lure others into storing their data at her site.

4.4 Mirror Candidate Ranking in the Regular Mode

SOUP's regular mode makes use of knowledge that a node does not have while bootstrapping, but can obtain after it has established social relations with other users. It will then leverage their observations to rank mirror candidates.

As illustrated in Fig. 3, 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 will record an *experience value* (exp_v) based on u's friends' experience regarding v in the KB. A node w is *friends* with u if there is an edge (u, w) in the OSN's social graph G, which represents a social connection between both nodes. The experience value is the basis for ranking mirrors. In addition, the entry for v will record whether or not v is friends with u and a TTL (time-to-live) value that decreases every time u does not choose v as a mirror (TTL not shown in Fig. 3). Also, for every node w that is a friend of u, u records an experience set $ES_u(w)$ as shown in Fig. 4. This set records u's observations of w's mirrors; that is, when requesting w's data (Step 1 in Fig. 4), u records whether or not the data is available at w's mirrors (see Fig. 3b). It will then periodically transmit its experiences to w (Step 2). Besides confining overhead, we limit the experience set exchange to friends for two further reasons: First, users request the their friends' profiles more often than those of strangers. This way, they can record experience sets on the fly when requesting the data anyway. Second, this limitation raises the bar for malicious nodes

KBu	Node v	sr(u,v)	expv
	w	1	-

ESu(w)	Friend f	Mirror(f)	# of req	succ

(a) Initial State

Node v	sr(u,v)	expv
w	1	-
x	0	-
y	0	-

Friend f	Mirror(f)	# of req	succ
w	v1	12	8
w	v2	10	1
w	v3	10	9

(b) First Observation Recordings

Node v	sr(u,v)	expv
w	1	0.67
x	0	0.22
y	0	0.90

Friend f	Mirror(f)	# of req	succ
w	v1	14	10
w	v4	7	6
w	v3	8	8

(c) After Exchange of Experience Sets

Figure 3: Maintenance of knowledge base KB (top table) and experience sets ES (bottom table) at node u. Initially, u only knows one node (node w in (a)), which is also friends with u (i.e., $sr(u, w) = 1$). As u learns about new nodes, it adds them to KB_u (e.g., x, y in (b)). For each friend, node u further observes the performance of the friend's mirrors and records its experiences in $ES_u(friend)$ (e.g., w in (b)). u also receives $ES_j(u)$ from each friend j, allowing u to calculate the experience ranking for each node in KB_u (c). As u continues to record its own experiences for friend nodes (c), node w has replaced node v_2—which u had rated low—with node v_4.

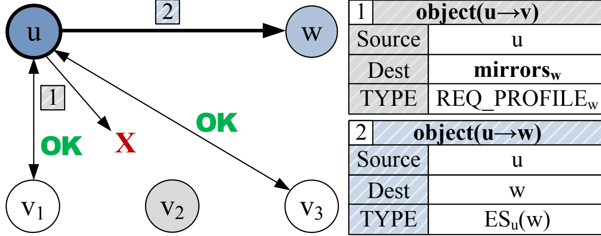


Figure 4: Recording of Experience Sets

trying to perform slander, as they have to establish social connections to their victims first.

Thus, for every node j that u is friends with, u will receive an experience set $ES_j(u)$ from j, which 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:

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

where n is the number of experience sets that u's friends have reported, $o_{(j,v)}$ is the number of observations regarding v that a friend j is reporting since the last experience set exchange, o_{max} is the maximum number of observations that j can report, and $av_{(j,v)} \in [0,1]$ is the availability of v during j's requests of u's data. Eq. (1) 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 as exchanging observations is limited to friend nodes). Instead, it cares for the number of attempts and successes of each reporting node in receiving its data.
- **Security.** However, a single malicious node could report huge numbers of manipulated observations, outweighing a lot of regularly observing nodes. We limit the maximum number of observations, o_{max}, that a node can report to confine the influence of a few (potentially malicious) nodes.

Hence, a significant portion of the recommending nodes, i.e., the friends of 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 than an older one (exp_v^{old}). Otherwise, a malicious node could perform a traitor attack, where it obtains an excellent reputation just to exploit it afterwards. In particular, such a node could offer exceptional storage capacities and online time to get selected as a mirror by many users, just to disappear later. Or, the quality of a mirror could suddenly deteriorate because of accidental reasons like connectivity problems. Applying the aging factor supports quick adaption to such situations. However, α should also not be over-valued, since a performance degradation can be temporary as well. When evaluating α, we found that observing only the most recent observations might in fact lead to unstable mirror sets. Setting $\alpha = 0.75$ provided us with the best trade-off between adaptation and stability in our experiments.

Algorithm 1 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 > ε 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 friends with 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 overlooking better nodes
add to M_u a random node v''
return M_u
```

4.5 Choosing Mirrors from the Ranking

Once a node obtains the ranking of mirror candidates from either mode, it selects its mirrors from the candidates, as depicted in Algorithm 1. It has a *target error rate*, ϵ, such that the probability of her data being unavailable is less than ϵ. First, it adds the top-ranked candidate nodes to its mirror set one by one, until the estimated likelihood of the data not being available is less than a target error rate ϵ:

$$p_{err} = \prod_{i=1}^{n} (1 - r_i) < \epsilon \quad (2)$$

where r_i is the experience value of the i-th node in the candidate list. 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 its friends:

$$r_v = max\left(\beta \cdot r_v, 1\right), \quad where \ \ \beta > 1. \qquad (3)$$

Friend 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 β shows that a friend has to provide at least 80% performance of unrelated mirrors ($\beta \approx 1.25$)in order to offer the best availability and overhead, i.e., it cannot be significantly inferior to the unrelated nodes. Note that, in contrast to related works, nodes with few or low-ranked friends are not discriminated by the social filter and can still achieve high availability. The filter is rather an option for those nodes with highly ranked befriended mirror candidates. Finally, u adds to its mirror set a random node for which it has not yet determined a ranking. This way, u prevents a possible overlooking of even better suited nodes.

4.6 Protective Dropping

A mirror node v may not always have enough space to store the data for another node, say u (e.g., if v is a popular mirror that wants to prevent getting overloaded with storage requests). While v can simply neglect u's storage request, alternatively, v can also drop another node's data to make more space for u. This will not only provide more flexibility, it will also enable v to choose what data to store.

If a miscreant orchestrates a sybil attack and floods the OSN with a large amount of storage requests, v may quickly fill up its storage space. Therefore, in SOUP, each mirror node v implements a dropping policy that favors friends. As malicious identities usually have difficulties establishing social connections to regular nodes [24], v can drop the profiles of the malicious node, leaving space for the data of friends.

On the downside, this practice would discriminate honest nodes without or with few friends, since these nodes need to rely on non-friend nodes. Therefore, for each node w that stores its data at node v, v calculates a dropping score, d_w, for w's data as follows (with notations given in Table 2):

- As v exchanges experience sets with each friend, say u, it also learns which nodes store their data at u.
- If w also stores its data at u, we increase d_w by 1. To protect the data of friends, their score is decreased by $1/\beta$ (recall that we use $\beta \approx 1.25$). 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.
- If d_w reaches a threshold θ, node v then blacklists w from storing its data on v.

The threshold can vary depending on the willingness to avoid false positives, which can occur due to network errors, e.g., an error when publishing a new set of mirrors. Our experiments provided the best results with a three-strike principle, in which $\theta = 300$ and $c = 100$. Thus, a node w will be blacklisted at v after v observed three mismatched mirror sets. In our evaluation (Sec. 5), we will show that this mechanism effectively protects SOUP against attackers who try to flood the system.

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 2: Protective Dropping Notations.

OSN	Nodes	Edges	Avg. Degree
Facebook	90,269	3,646,662	40.40
Epinions	75,879	508,837	6.71
Slashdot	82,169	948,464	11.54

Table 3: Datasets for SOUP Evaluation [25, 26]

5. SIMULATION AND ANALYSIS

We start our evaluation with a large-scale simulation of our data replication scheme, with regards to the challenges listed in Sec. 4.1. Our experiments with three real-world datasets show that SOUP provides high data availability with low overhead, and does so for *all* nodes in the OSN. SOUP performs even better when altruistic nodes exist, and successfully copes with node churn and malicious attacks.

5.1 Metrics, Datasets, and Methodology

We first define two basic performance metrics:

- **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 the robustness, openness, and resiliency of SOUP:

- **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 increase performance when altruistically provided resources are available.
- **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 3. These datasets cover a variety of real-world social graph features and can help evaluate SOUP's performance in different contexts. For instance, users should not depend on their number of friends, which is why we chose the less-connected Epinions dataset with an average node degree of only 17% of that in the Facebook dataset. We run simulations of SOUP using these datasets, and measure the metrics defined above. We further handle the following parameters associated with each user:

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 4.5).

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

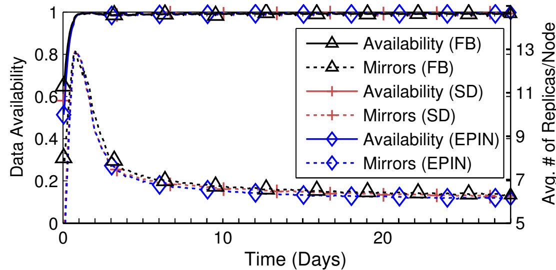


Figure 5: SOUP achieves high availability with low overhead.

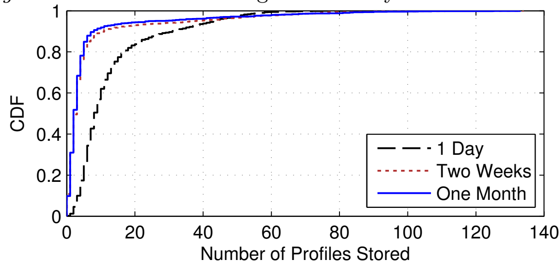


Figure 6: SOUP proves to be stable, and 90% of the users do not have to store more than 7 replicas.

on bursty interaction patterns of users and typically follows a power-law distribution [18, 22, 27]. 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 [28], we consider three time zones (US, Europe and Africa, and Asia and Oceania), where a node's probability of belonging to these zones is 0.4, 0.3, and 0.3, respectively. To bootstrap, nodes join our experiments asynchronously according to their online probability.

User activity pattern. Different evolutions of OSN user activity have been observed [18,22]. We model user activity to be exponentially decreasing [22]. 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 (i.e., get in contact with them) in order to find the best-suited mirrors, in all literature we are aware of, this model represents the worst observed case.

Available storage space per node. Each node must have a specific storage space value in order to evaluate the storage overhead and dropping strategy of SOUP. The storage space available at each node follows 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. 7.

5.2 Results and Analysis

5.2.1 Data Availability and Replica Overhead

Fig. 5 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 upon joining. As SOUP reaches equilibrium, the high level of availability is maintained for the entire remaining period.

After joining, due to the lack of knowledge about good mirrors, nodes do not select well-suited mirrors yet, and the number of replicas increases. However, as soon as nodes obtain more precise rankings, the quality of mirrors improves

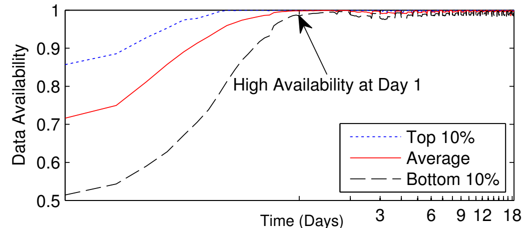


Figure 7: SOUP is robust and does not discriminate any node based on social relations or online time.

and the replica overhead is reduced by about 50%. On average each node needs to store less than seven replicas.

5.2.2 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. We show SOUP's profile distribution in Fig. 6 (we report results from the Facebook dataset for the rest of this section; we observed the same behaviour of SOUP with both other datasets). After day one, around half the nodes need to store 10 or more replicas so that the system achieves high availability. However, as user experiences are more accurately measured, 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, we find that as mirror rankings become more accurate, the drop rate of data converges from 0.07% to a very low 0.045%. 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 mobile nodes, are rarely chosen as mirrors, saving storage, bandwidth and battery on these devices.

5.2.3 Robustness

Regardless of their own online probability or quantity of friends, every user should achieve a very high level of data availability. We pick the top and bottom 10% of users (first with regards to their own online probability and second with regards to their number of friends) and compare their performance in Fig. 7. After just one day, even the bottom 10% of users obtain data availability of above 99%. Hence, in contrast to related works (e.g., [9–13]), users are discriminated neither based on their own online time nor based on the quantity and quality of their social relations. Instead, SOUP offers a robust OSN.

5.2.4 Openness

One of SOUP's challenges is to exploit altruistically provided resources efficiently. Fig. 8 shows the impact of the presence of small percentages of altruistic nodes that are steadily online. We can observe that 5% (a=0.05) altruistic nodes can cause a slight increase and stabilization of availability, but the improvement in terms of replica overhead is more prominent; as altruistic nodes become known to the OSN, nodes can select fewer mirrors than before to achieve the same level of availability. Hence, while SOUP does not rely on any kind of altruistic nodes, it can exploit such nodes' resources if available.

5.2.5 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

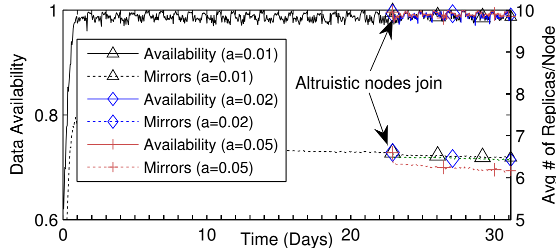


Figure 8: SOUP can exploit altruistic resources.

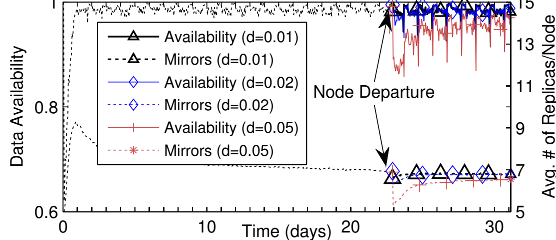


Figure 9: SOUP is resilient against node dynamics.

to the rest of the OSN (Fig. 9). If we assume the top 5% of nodes in terms of online time leave the OSN simultaneously (d=0.05), there is 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 independent from the top 1-2% of nodes, as data availability does not significantly drop as these nodes leave the OSN. Still, 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, these mirrors will receive a lower ranking, and SOUP will distribute the load among additional mirrors. If a mirror is completely taken down, SOUP will choose a different one, as shown above. Compared to the static mirror choices of related work, SOUP is the only approach capable of such adaptation towards both increasing and decreasing resources.

5.2.6 Resiliency Against Malicious Nodes

None of the existing DOSN solutions consider attacks on their system. We measure SOUP's performance under attack of up to half the nodes in the OSN. In our experiment, SOUP not only needs to tolerate the attackers after having stabilized, but also has to bootstrap in their presence.

First, we study the impact of the *slander attack*, in which attackers manipulate experience sets (or recommendations to bootstrapping users). We assume that the malicious users have infiltrated the OSN successfully and they send out recommendations at the maximum rate. Fig. 10 shows that even when 50% of social relations—and thereby experience sets—are subject to slander, the data availability at most drops to around 95% ($m = 0.5$). Second, we investigate a *flooding* attack, in which an attacker creates multiple identities (*Sybils*) and floods benign mirrors with data. We show results for different percentages of Sybils in Fig. 11. Even with as many Sybils as regular identities in the OSN, the data availability does not drop below 90% for the benign users in the long run. The replica overhead, although increased, does not exceed 13 copies per node. In this case, protective dropping prevents data of socially connected nodes from being dropped for a Sybil's data, and avoids the full utilization of resources at benign nodes.

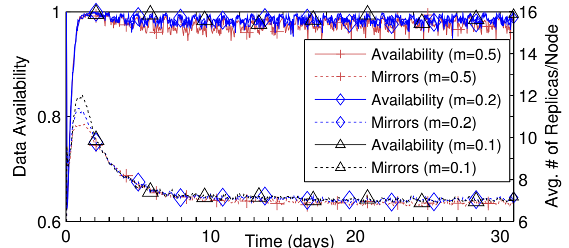


Figure 10: SOUP is resilient against a slander attack.

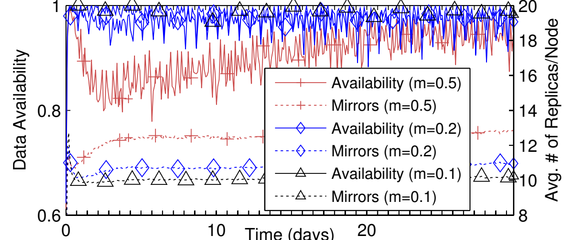


Figure 11: SOUP can recover from a flooding attack.

Approach	Online time assumption	Availability	# of replicas
SOUP	Power-law	~99.5%	~6.5
Safebook	Uniform, all nodes p=0.3	~90%	13-24
SOUP		~98.5%	~14
PeerSoN	10% of nodes p=0.9 25% of nodes p=0.87 30% of nodes p=0.75 30% of nodes p=0.3	<90%-100%; Depends on p	6
SOUP		~100%	4

Table 4: SOUP vs. related work (p = online probability).

5.2.7 SOUP vs Related Work

SOUP's superiority over state-of-the-art solutions mainly stems from its *qualitative* properties, which we extensively evaluated above. Compared against those, SOUP is robust, adaptive to node dynamics, and resilient against attacks.

To further compare the performance of SOUP and related work *quantitatively*, we run simulations of SOUP under the node online time distributions assumed in related works, in those cases where the distributions were available to us. As shown in Table 4, 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. 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 create 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.

6. IMPLEMENTATION

Our implementation of a SOUP node comprises two components: the *SOUP middleware* and the *SOUP applications*. Taken together, both constitute a SOUP node as depicted in

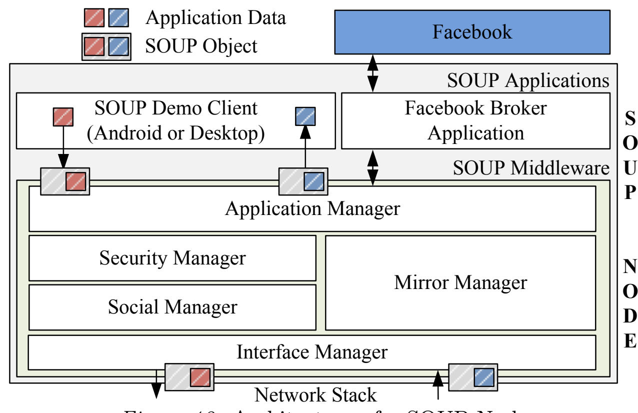


Figure 12: Architecture of a SOUP Node

Fig. 12. Note however that it is also possible to exclusively run the middleware component, for instance to provide an altruistic node, which only acts as a mirror or DHT relay for mobile nodes.

For SOUP applications, we have implemented a SOUP Demo Client that can run on either a PC or Android device to organize SOUP nodes into a social network (see Fig. 13). It supports essential OSN functionalities: Users can search for each other in the OSN, establish friendships, share photos or exchange messages. We also implemented a broker application that can suggest friends to a SOUP node when it is fed with data through the Facebook API.

For the SOUP middleware, we have implemented it also for both desktop and Android systems. It consists of several modules, each responsible for a pre-defined task and easily exchanged for an improved or different approach at any time. All modules communicate by passing SOUP objects to each other. One particular module is the *Application Manager* that has a simple interface with SOUP applications. It has two functionalities: (i) it allows arbitrary social applications to run on top of the SOUP middleware; and (ii) it enables communication between applications transparent to the middleware itself. On one hand, it encapsulates content from a SOUP application into SOUP objects. On the other hand, it decapsulates content destined for an application from SOUP objects received from other modules. Furthermore, the *Social Manager* module is responsible for processing requests when an object indicates a change to the social data, and the *Security Manager* module deals with all encryption-related tasks using our own optimized ABE implementation.[1] The *Mirror Manager* module is responsible for the selection of mirrors. A node needs to push any change of its data to its mirrors, and it also needs to manage the data that it mirrors for others. Last, if any of these modules need to communicate with other nodes, they do so by passing an object to the *Interface Manager*, which can then initiate communication via a suitable network interface. Consider a friend request as an example: After an application initiates the request, the Application Manager converts the request to an appropriate SOUP object, and passes it to the Social Manager. The Social Manager manipulates the user's friend list and forwards the object to the Security Manager. The Security Manager encrypts and signs the object and hands it to the Interface Manager, which sends the object to the request target over an appropriate link. If the Interface Manager later receives an encrypted request confirmation object from the target, it forwards it to the Security Manager, which unlocks the object and issues a confirmation to the application via the Application Manager.

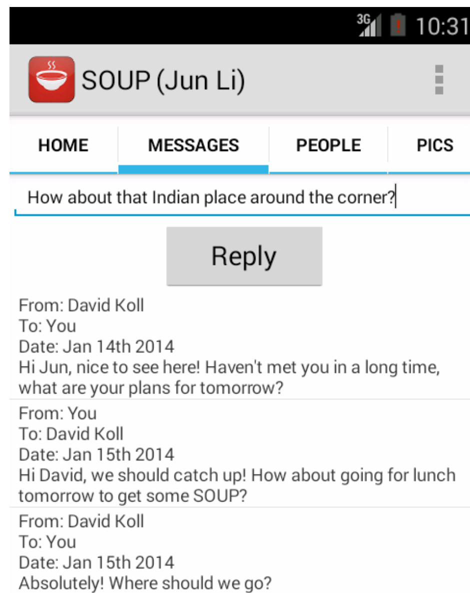


Figure 13: SOUP on a Nexus 4 Android Phone

For the underlying DHT, we use FreePastry 2.1[2], an open source implementation of the Pastry DHT [29]. Most of our code and executables are available online.[3]

7. DEPLOYMENT

We have deployed SOUP on our own real-world DOSN of 31 users. Four of those were using different Android mobile phones. All phones were relaying via the same gateway node, and that node also acted as the bootstrapping node for users running the regular SOUP client. We collected several days of data, during which our users established 282 friendships, shared 204 photos, and exchanged 1189 messages.

Our deployment outperformed our simulation results with regards to availability (we did not observe a single loss) and replica overhead. However, note that our large-scale simulations should be more accurate, since we observed much longer online times in our experiment than typical for OSNs. The lessons learned from the deployment are the following:

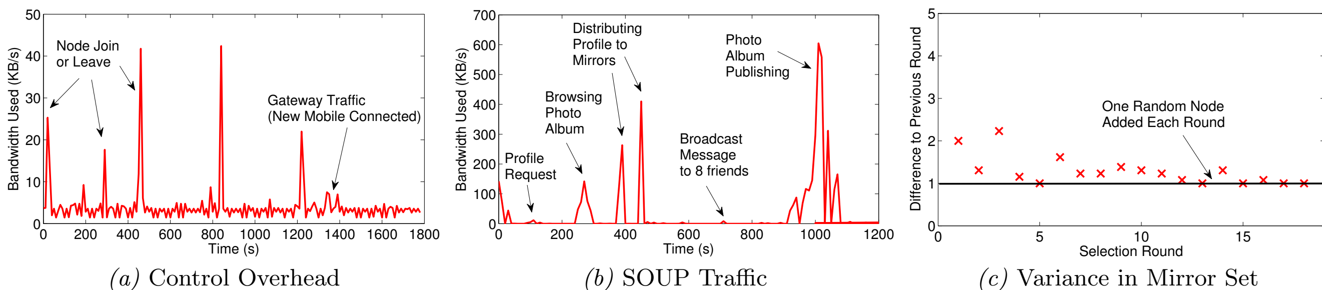
The bandwidth consumption of SOUP is not a concern. We show the bandwidth consumption of the DHT at our bootstrapping node in Fig. 14a. Only upon join and leave operations (i.e., shifting some entries in the DHT) we observe utilization of the network interface at around 20-40 KB/s. At the same time, lookups do not have a visual impact. Thus, the cost of relaying for a mobile node is confined to its join procedure, which requires several DHT operations.

The traffic introduced by SOUP itself is also manageable. Fig. 14b shows the most bandwidth intense period of 20 minutes we observed for any user during the time of data collection. Messaging or simple profile requests do not consume a lot of bandwidth and are hardly distinguishable from an idle link. More intense activities like skipping through a photo album does not consume a regular user's bandwidth as well, as she takes her time to view the pictures. Note that the data traffic a user generates by *consuming* content is approximately the same as in centralized OSNs, as the user needs to download the data in those systems as well. Only when

[1]Based on https://github.com/wakemecn/cpabe

[2]http://www.freepastry.org

[3]http://soup.informatik.uni-goettingen.de/



(a) Control Overhead *(b)* SOUP Traffic *(c)* Variance in Mirror Set

Figure 14: SOUP introduces little control overhead (a), can manage the traffic (b) and the mirror sets are stable (c).

producing or mirroring content, SOUP generates additional traffic. Produced data has to be distributed to the mirrors as well and when acting as a mirror, the uplink of a user is used. As a consequence, in Fig. 14b, the link is most utilized is at the creation of a photo album. As in centralized OSNs, mobile users on a data plan should delay such uploads until they can access a WiFi link. Motivated by these measurements our Android implementation disables mirroring on the Android device, while not cancelling concurrent mirroring on her stationary device(s). Users can however opt in to contributing their Android device as a mirror (e.g., consider a tablet that is connected to WiFi most of the time.)

The mirror sets in SOUP are stable. Overall, we find that the mirror sets remain stable and do not differ much between selection rounds as shown in Fig. 14c. After the initial rounds, most mirror changes are additions of a random node as described in Sec. 3. Each round, only few nodes change additional mirrors. As a consequence, the whole data of a user does not have to be transmitted often, and the communication overhead remains modest.

The cryptographic overhead is low. We found that the processing times for our ABE operations are $\approx$ 10% faster than existing JAVA implementations. When encrypting a key with four attributes (the processing time grows linearly with the number of attributes, independent of the data size [21,30]), the 90th percentile encryption time was 262ms, whereas decryption was four times faster and took 61ms per data item. As improved libraries for encryption (e.g., DOSN-specific approaches [30]) become available, SOUP can implement these as a new version of the Security Manager.

SOUP can manage much more. We further tried to push SOUP to its limits by using data from other real-world OSNs. In our deployment, we obtained access to the Facebook data of 20 users. Their profiles offer details beyond a crawler's results, as those often do not include major parts of a user's data [8] (e.g., photos on Facebook are not publicly available by default). The average profile size was ≈10 MB, with the largest profile containing hundreds of photos in 27 photo albums and one video. This profile consumed 60 MB of disk space in total. Overall, the data disclosed 2035 unique data items to us. 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, and large items rarely exist. These findings largely coincide with those in [8]. The whole data sums up to 206 MB. We selected one mirror as a host for all data. Recall that storing these 20 profiles is three times as much as 90% of SOUP nodes will have to store. We then asked for text, photo and video data from the mirror according to the request probabilities for each data type as described in [23]. As shown in Fig. 15, the average consumption is well below 600 KB/s, even if the mirror has to handle 20 requests per second. With an increasing request frequency large items are hit more often, which causes the spikes in our measurements. As a result, a request might time out once a mirror becomes overloaded, which may happen especially to nodes mirroring popular data, or limits its bandwidth. Note that unlike other approaches where users are stuck with a static or heteronomous set of mirrors [9–13], SOUP will adapt to this situation.

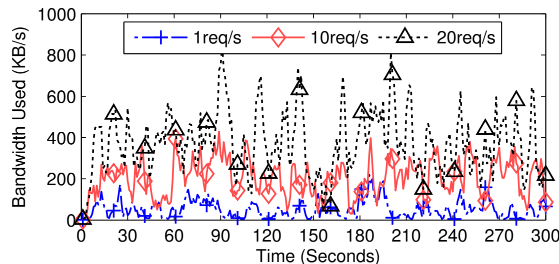


Figure 15: Bandwidth consumption at high request rates.

8. DISCUSSION AND FUTURE WORK

Use of social relations. 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 trustworthiness of social relations within an OSN is questionable. In particular, it remains unclear to which extent the binary model of social relations reflects the real world [31]. Recently, there have been approaches to breaking this model and studying the effect of more expressive social relations [32, 33]. In fact, friend relations in OSNs are multi-faceted and the existence of the relation itself only contributes very little to its tie strength [33]. Even though SOUP is performing well if a large fraction of the social relations in the OSN is compromised, exploiting more precise relation models may provide an opportunity to further improve its performance, stability and resiliency. For instance, during mirror selection, SOUP could prefer closely related users represented by a strong tie. The selecting node could value their experience sets more than those of mere acquaintances, which could further reduce the impact of manipulated experience sets. Or, the value of the social filter β could be adjusted to the strength of the relation with each particular friend. Also, data of closer friends could be more secured from being dropped.

Large profiles. Although the storage and communication overhead is currently unproblematic when deploying SOUP to the real world, there might be difficulties if users share much larger data items or generate extremely large user profiles in the future. One option to overcome such

difficulties could be the use of network coding to distribute a large profile among mirror nodes. Network coding originally was proposed to improve the throughput utilization of a given network topology [34], but can also be used in the context of decentralized data storage [35]. Here, a file f can be split into k equally sized (f/k) *pieces*, which are in turn encoded into n fragments using an (n, k) maximum distance separable code. After distributing the fragments to n nodes, it is possible to obtain the complete information from k encoded fragments. Thus, instead of storing full replicas among mirrors, SOUP could distribute encoded parts of the profile (pieces), and then allow for reconstruction from those parts' fragments. Doing so can (i) prevent one node from being overloaded with a large profile and (ii) increase the data availability of SOUP as only k fragments need to be available to reconstruct the data of interest.

Extended recommendations. The recommendations in SOUP currently measure whether or not a user's friends were able to retrieve the user's data from her mirrors (i.e., the availability of data). SOUP can be extended in a way that a user's friend also reports the bandwidth available at the mirrors, which is then considered during mirror selection. Ultimately, this could lead to a better quality of service for users requesting data from mirrors.

9. CONCLUSION

In this paper we presented SOUP, a decentralized online social network (DOSN). SOUP addresses the severe privacy concern from centralized OSNs by returning the control over who can access which data to the users, and can securely encrypt user data according to a fine-grained access policy. As our key contribution, we advocate a new approach to storing user data in a large-scale DOSN. It successfully obtains the best mirror nodes for each user to achieve high data availability with little overhead, while keeping all selected mirrors synchronized. It is robust in that data of *all* users are highly available without node discrimination, and can converge to a stable state quickly. It further copes with adverse situations effectively, and can opportunistically leverage social relations and altruistically provided resources. Enabling the aforementioned features distinguishes SOUP as a unique, full-fledged DOSN from existing DOSNs and their deficiencies. Finally, our comprehensive implementation and real-world deployment of SOUP validate its practicability, including its solid support toward mobile users.

Acknowledgements: We would like to thank the anonymous reviewers for their valuable feedback, as well as Joshua Stein, Skyler Berg and Nicole Marsaglia for their proof-reading efforts. This work has been partially supported by the EU FP7 CleanSky ITN (grant no. 607584), the National Science Foundation (grant no. CNS-0644434 and CNS-1118101), the Lindemann Foundation and the Simulation Science Center, sponsored by the Volkswagen Foundation and the State of Lower Saxony, Germany. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the supporters.

10. REFERENCES

[1] B. Debatin, J. P. Lovejoy *et al.*, "Facebook and Online Privacy: Attitudes, Behaviors, and Unintended Consequences," *Journal of Computer-Mediated Communication*, vol. 15, pp. 83–108, 2009.
[2] http://linkd.in/1jaSeJK (all URLs checked 31/08/2014).
[3] C. Dwyer, "Privacy in the Age of Google and Facebook," *Technology and Society Magazine, IEEE*, vol. 30, no. 3, pp. 58 –63, 2011.
[4] "The Diaspora Project," https://joindiaspora.com/.
[5] A. Shakimov, H. Lim *et al.*, "Vis-a-Vis: Privacy-preserving online social networking via Virtual Individual Servers," in *COMSNETS'11*.
[6] D. Liu, A. Shakimov *et al.*, "Confidant: Protecting OSN Data without Locking it Up," in *Middleware '11*.
[7] R. Sharma and A. Datta, "SuperNova: Super-peers based Architecture for Decentralized Online Social Networks," in *COMSNETS'12*.
[8] R. Baden, A. Bender *et al.*, "Persona: An Online Social Network with User-defined Privacy," in *SIGCOMM '09*.
[9] K. Rzadca, A. Datta *et al.*, "Replica Placement in P2P Storage: Complexity and Game Theoretic Analyses," in *ICDCS '10*.
[10] S. Nilizadeh, S. Jahid *et al.*, "Cachet: A Decentralized Architecture for Privacy Preserving Social Networking with Caching," in *CoNEXT'12*.
[11] L. Cutillo, R. Molva *et al.*, "Safebook: A Privacy-Preserving Online Social Network Leveraging on Real-life Trust," *Com. Mag., IEEE*, vol. 47, no. 12, pp. 94–101, 2009.
[12] A. Mahdian, R. Han *et al.*, "Results from a Practical Deployment of the MyZone Decentralized P2P Social Network," *CoRR*, 2013.
[13] S. Biedermann, N. P. Karvelas *et al.*, "ProofBook: An Online Social Network Based on Proof-of-Work and Friend-Propagation," in *SOFSEM'2014: Theory and Practice of Computer Science*. Springer, 2014, pp. 114–125.
[14] T. Isdal, M. Piatek *et al.*, "Privacy-preserving P2P Data Sharing with OneSwarm," in *SIGCOMM'10*.
[15] S. Androutsellis-Theotokis and D. Spinellis, "A Survey of Peer-to-Peer Content Distribution Technologies," *ACM Comput. Surv.*, vol. 36, pp. 335–371, 2004.
[16] A. Pace, V. Quema *et al.*, "Exploiting Node Connection Regularity for DHT Replication," in *SRDS'11*.
[17] S. L. Blond, F. L. Fessant *et al.*, "Choosing Partners Based on Availability in P2P Networks," *ACM TAAS*, vol. 7, no. 2, p. 25, 2012.
[18] L. Gyarmati and T. Trinh, "Measuring User Behavior in Online Social Networks," *IEEE Network*, vol. 24, no. 5, pp. 26 –31, 2010.
[19] F. Benevenuto, T. Rodrigues *et al.*, "Characterizing User Behavior in Online Social Networks," in *IMC '09*.
[20] A. Mislove, M. Marcon *et al.*, "Measurement and Analysis of Online Social Networks," in *IMC '07*.
[21] J. Bethencourt, A. Sahai *et al.*, "Ciphertext-Policy Attribute-Based Encryption," in *IEEE SP '07*.
[22] C. Wilson, B. Boe *et al.*, "User Interactions in Social Networks and their Implications," in *EuroSys'09*.
[23] F. Schneider, A. Feldmann *et al.*, "Understanding Online Social Network Usage from a Network Perspective," in *IMC'09*.
[24] B. Viswanath, A. Post *et al.*, "An Analysis of Social Network-based Sybil Defenses," in *SIGCOMM'10*.
[25] B. Viswanath, A. Mislove *et al.*, "On the Evolution of User Interaction in Facebook," in *WOSN'09*.
[26] "Stanford Large Network Dataset Collection," http://snap.stanford.edu/data/.
[27] J. Jiang, C. Wilson *et al.*, "Understanding Latent Interactions in Online Social Networks," in *IMC'10*.
[28] A. Verde, "Facebook Demographics," http://www.slideshare.net/amover/facebook-demographics-2011.
[29] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Middleware '01*.
[30] O. Bodriagov and S. Buchegger, "Encryption for Peer-to-Peer Social Networks," in *Security and Privacy in Social Networks*. Springer, 2013, pp. 47–65.
[31] D. Koll, J. Li *et al.*, "On the State of OSN-based Sybil Defenses," in *NETWORKING'14*.
[32] S. Tang, J. Yuan *et al.*, "Relationship Classification in Large Scale Online Social Networks and its Impact on Information Propagation," in *INFOCOM'11*.
[33] E. Gilbert and K. Karahalios, "Predicting Tie Strength with Social Media," in *CHI'09*.
[34] S.-Y. Li, R. Yeung *et al.*, "Linear Network Coding," *Inform. Theory, IEEE Trans. on*, vol. 49, no. 2, pp. 371 –381, 2003.
[35] A. Dimakis, P. Godfrey *et al.*, "Network Coding for Distributed Storage Systems," *Information Theory, IEEE Trans. on*, vol. 56, no. 9, pp. 4539 –4551, 2010.