Privacy-aware and highly-available OSN profiles

Rammohan Narendula, Thanasis G. Papaioannou, and Karl Aberer
School of Computer and Communication Sciences, EPFL, Switzerland
Email: firstname.lastname@epfl.ch

***Abstract*—The explosive growth of online social networks (OSNs) and their wide popularity suggest the impact of OSNs on today's Internet. At the same time, concentration of vast amount of personal information within a single administrative domain causes critical privacy concerns. As a result, privacy-conscious users feel dis-empowered with today's OSNs. In this paper, we report on an on-going research work and introduce a privacy-aware decentralized OSN called *porkut*. Our system exploits trust relationships in the social network for decentralized storage of OSN profiles and their content. By taking users' geographical locations and online time statistics into account, it also addresses availability and storage performance issues. We finally advocate indexing of social network content and present an approach for indexing in a privacy-preserving manner.**

***Keywords*-online social network; privacy-preserving index; connected dominating set; trust**

I. Introduction

Online social networks (e.g. Facebook.com, Orkut.com) have recently seen an explosive growth. Facebook received 130 million visitors in a single month in 2008 [1] and currently has more than 200 million users. As a result, these OSNs have become store houses of unprecedented amount of data in the form of messages, photos, links, and personal information. Facebook has grown to be the the world's largest photo sharing service surpassing even dedicated photo-sharing online applications (e.g. Flickr). It is also the largest instant messaging service on the web. Researchers argue that future Internet will be very much influenced by social networks regarding the location of content and knowledge, and the user interactions [2]. However, most of the current social networks operate on infrastructure administered by a single authority (*big-brother*), such as Google, Facebook, etc. These organizations perform mining of personal data hosted inside users profiles and exploit it for targeted advertisements, in order to be compensated for their huge investments in infrastructure. During sign-up time, users consciously or unconsciously permit the organizations to share their personal information with third-parties in whatever form the organizations choose to [3]. In addition, the leakage of personal information from OSNs can be associated with the user activity on non-OSN sites as well [4].

However, the exponential growth of the OSNs suggests that users are ready to trade privacy over utility of the services offered. As a result, there is almost negligible motivation for the OSN operators to address privacy concerns of the users. In order to address privacy concerns of OSN users, research community has resorted to the P2P paradigm for OSN content management. Replacing the big-brother with a community of users, enables OSN users to have complete control on their profile content.

In this paper, we present an initial design of such a system, referred to as *porkut*, where users organize a social network over a P2P overlay with privacy-preserving data access. We briefly outline the system architecture and mainly focus on the distributed storage layer. Specifically, we propose a decentralized mechanism for users to manage their own online social network on top of resources collectively contributed by themselves. Such a design is motivated with several goals in mind: a) It eliminates the requirement for a single big-brother who can exploit the users' profile data for his own interest without users' consent. b) It preserves the privacy of individuals social profile content, as they have complete control on who can access which parts of the content. c) It exploits the trust relationships among users in the social network to improve the content availability and the storage performance. Both issues are non-trivial in a P2P setting. Three approaches with different goals for improving storage performance are introduced, while maintaining high content availability. A user's profile content is hosted only on a set of self-defined trusted nodes that enforce access control on the content. This set of trusted nodes is selected intuitively keeping the availability and performance goals in mind. Other issues such as the structure of the profile content, the format of the access control policies, trusted identity management, and other data integrity issues are beyond our scope.

In addition, the system constructs a privacy-preserving index of the social network content that enables privacy-aware searching. We argue that such an index enables content discovery among friends in OSNs and helps system users discover new friends (based on content, such as common interests etc.) within the OSN application and establish new social connections. This is an add-on feature over existing OSNs like Facebook that do not allow content-based search. Such an index is hosted over the P2P overlay in a distributed hash table (DHT). Users can specify their privacy objectives during content publishing and thus content existence and ownership are only revealed according to their preferences. This index could be used to serve advertisements on searches and distribute the revenues to the users according to their published content. This way, users can benefit from their content without compromising their privacy. In contrast, the current OSN applications exploit users' content for own monetary gains.

The rest of the paper is organized as follows. In Section II, a brief description of the system is provided. The storage layer is discussed in Section III. The privacy preserving indexing is described in Section IV. In Section V, we discuss the related work and, finally in Section VI, we conclude this paper and outline our future work.

II. System Overview

As mentioned earlier, the *porkut* system exploits the trust relationships among friends and social network connections to improve the availability and search performance of the system. We assume that a user of *porkut* runs the client on his office or personal laptop/computer. Hence, for the rest of the paper, we use the terms *user* and *node* interchangeably.

A user u's profile content is hosted only on a set of self-defined trusted nodes, which enforce access control on the content on behalf of the user. This set of trusted nodes for a user is referred to as his *trusted proxy set (TPS)*. The TPS members for a user are properly selected with respect to the availability and performance goals. We observe that every user in an OSN has friends scattered over a limited set of geographical locations (e.g. his home town, working location, home country, location of previous institute etc.). Moreover, we observe that each user's online timings are predictable to a large extent (e.g. his office hours, completely offline on weekends). Exploiting these facts, we populate this set of trusted nodes in such a way that, at any given time, one node in this set is online to satisfy the profile access requests, while at the same time, the content is located at a node falling within a geographical neighborhood away from the user that frequently asks for it. The computation of the set TPS based on a user's social graph is explained in next section. Each user u is identified by a unique identifier denoted by UId_u. Note that a TPS is a set of UIds.

The *porkut* system employs a distributed hash table (DHT) hosted at the resources contributed by the users. This DHT is used for storing the privacy-preserving index of the profile content and other meta information, e.g. the current IP address of a user. A user u and his TPS mapping is stored in the DHT in the form of *(key,value)* pair with *key* being the UId_u and *value* being the members of the TPS. Using cryptographic signatures, it should be trivial to test the authenticity of such an entry in the DHT. This user-to-TPS mapping in the DHT is useful for contacting the nodes where the profile of a particular user is stored.

We assume that, with a reasonable replication factor, one can ensure that the data items stored inside DHT are highly available in spite of node churn. As a trusted storage is not required by the system design, such a DHT could be hosted at a highly available cloud storage or in publicly available OpenDHT-like services [5]. The *porkut* storage architecture is illustrated in Figure 1. Therein, the user u_1 has 5 friends in the OSN, namely u_2 to u_6. The set $TPS = \{u_1, u_2, u_4\}$ is shown in the figure and a mapping between u_1 and $TPS[u_1]$ is inserted into the DHT. The user social graph is represented as *online time graph*, which is explained in the next section.

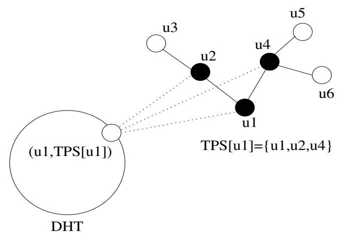

Fig. 1. The *porkut* storage layer

III. Storage Layer

In this section, we discuss the storage mechanism of the *porkut* system, and mainly address the construction of the set $TPS(u)$ for a user u from his social graph.

The social network graph is denoted as $G(U, R)$, where U is the set of users represented by the vertices in the graph and R is the set of friendship relations represented by edges. For example, an edge between two vertices u_1 and u_2 models the fact that users u_1, u_2 are friends. We assume that friendship relationships are symmetric. This is the default assumption in current OSN applications, e.g. Orkut, Facebook. We use the notation $N_G(u)$ to represent the set of neighbors (i.e. friends on the OSN) of user u in the social graph G, and $N_G[u]$ to represent $N_G(u) \cup \{u\}$.

We assume that each user u in the social network is characterized with two parameters: his *geographical location* and *online time period*. For instance, the location can be set to the country/city where the user is currently located. We exploit location information of friends of a user, in order to place data as close as possible to the nodes that most frequently access the data for getting profile updates etc. Therefore, data is stored on nodes falling within a certain geographical proximity from its most-frequent access points.

This is quantified by the metric *access cost* $C_{u_1}^{u_2}$ between two geographical locations/users/nodes u_1 and u_2, which is defined as the cost of the communication link between them (i.e. unit cost for transferring data in between these two nodes). This could be measured, for example, in terms of RTT between these two nodes.

Online time period represents the usual time that the user is online in the social network. This is the time window in which the user contributes his resources (i.e. bandwidth, storage, and processing power) for the social network operation. The node can only reply to the data access requests (for the data it hosts) that are generated during this time window. Beyond this time window frame, the user is offline. We denote the location and online time period parameters for a user u as L_u and OT_u respectively. Given two users u_1 and u_2's locations and online time settings, we argue that they can contact each other and thus exchange data if and only if their online time intervals overlap, which we represent by the condition that $OT_{u_1} \cap OT_{u_2} \neq \emptyset$.

A. Trusted Proxy Set

Each user u selects some of the neighbors in his social network as *trusted* nodes. The user trusts these nodes both for storing his profile content and for enforcing access control on the access requests. We believe that storing content in plain text and leveraging mutual trust relationships for access control enforcement simplifies the system to a great extent. This way of exploiting trust relationships for access control was first introduced by the authors in [6] and employed for the social network case in [1]. We assume that users mutually cooperate for hosting content and delegating access control with some social contracts. The intuition is that users do not breach the delegation responsibilities because of social pressure and monitoring. This is left for future study. Alternative solutions, which employ encryption mechanisms for access control and content storage [7], not only involve complicated key management issues, but also, they are highly inefficient in terms of storage overhead, as the same data item may need to be encrypted multiple times for different users with different access rights.

Let $T(u) \subseteq N_G(u)$ be the set of trusted users/nodes for user u based on his social relationships. $T[u]$ also includes the user u himself in the set of trusted nodes. The user selects a subset of these trusted users for hosting his content. We call this set as *trusted proxy set (TPS)* ($TPS(u) \subseteq T(u)$). The content of user u is stored on the members of the set $TPS(u)$ and itself, which is denoted as

$$TPS[u] = TPS(u) \cup \{UId_u\}$$

We propose the following criteria to select the proper set of members into TPS from the set of all the trusted users of a user: i) *low access and consistency costs* and ii) *high data availability*. To this end, the number of replicas should consider access and update costs and replica placement should consider users online time settings.

Next, we describe three approaches for the computation of the set $TPS[u]$ that satisfy high availability but have different cost minimization objectives. In every approach, once $TPS[u]$ is computed, for each friend/user in the social neighborhood of user u (i.e., $\forall v \in N_G(u)$), a *mount point* is configured (represented by M_v) for accessing u's profile. In other words, for a certain friend of u, u's profile is said to be *mounted* at a certain node. Note that, by definition, the mount point is available at some point in time during the friend's online time frame so that he can access u's profile. However, a single-mount-point-per-user technique allows to access the profile replica only when that mount point is online. To increase the availability, we can use all the nodes in $TPS[u]$ as mount points. In this case, M_v would be the *primary mount point* and the remaining would be the *secondary* ones. In the rest of the discussion, we assume that content accesses are being done from the primary mount point.

Given the above, the purpose of the following algorithms is to compute a *storage configuration* for user u, which is given by:

- the set $TPS[u]$, and
- $\forall v \in N_G(u)$, the mount point M_v, where $M_v \in TPS[u]$.

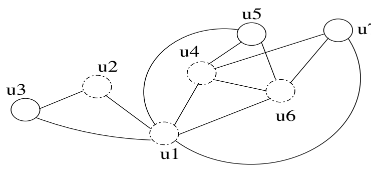

Fig. 2. The graph OG_{u_1}

B. Computing the storage configuration

Computing the storage configuration for a user u involves two steps:

i) Constructing the online time graph.
ii) Storage configuration computation from this graph based on some criterion.

For simplicity, we assume that geographical locations are considered at the granularity of country, assuming an OSN user has friends scattered over several countries. First, we construct the *online time graph* (denoted by OG_u) for user u. This graph will be used to compute $TPS(u)$.

Definition 1: Online time graph: for a user u (denoted by OG_u) is defined as $(N_G[u], E)$ where $N_G[u]$ is the set of vertices and E is the set of edges, such that

$$\forall v_1, v_2 \in N_G[u], \ \exists \text{ an edge}(v_1, v_2) \in E \ \text{ iff}$$
$$(v_1 \in T[u] \vee v_2 \in T[u]) \wedge (OT_{v_1} \cap OT_{v_2} \neq \emptyset)$$

Next, we specify the following two conditions on the graph OG_u, which are necessary and sufficient in order to compute a valid storage configuration.

1) OG_u must be connected. Only then, every user in the set $N_G[u]$ can access u's content.
2) The sub-graph induced by the set $T[u]$ i.e., the graph $OG_u[T[u]]$ must also be connected, in order to allow content synchronization across TPS members pass through only trusted nodes[1].

We suppose that each user constructs OG_u offline locally from the set of friendship relations that he has in the social network and their online time (OT) specifications. The construction of OG_u is explained with the following example. Assume a user u_1 with neighbors in the OSN u_2 to u_7 and their locations set as follows: L_{u_1} is *Switzerland*, L_{u_2} and L_{u_3} are *India*, and finally the rest are *US-West*. Assume OT set to 8am to 5pm local time for all users. Let $T[u_1] = \{u_1, u_2, u_4, u_6\}$. The resulting OG_{u_1} is shown in Figure 2.

Note that $OG_u[T[u]]$ is expected to be connected for a reasonable number of trusted friends with overlapping online times (given 120 friends per user in Facebook and 100 in Orkut on average [2]). Otherwise, another node $v \in OG_u$, yet $v \notin T[u]$, has to be employed in the TPS construction as well. However, profile data stored at v has to be encrypted by a key shared by the $T[u]$ members. This approach would be particularly useful in the bootstrap phase of the social network.

In the next subsections, we describe three algorithms with different cost minimization objectives for TPS generation and

[1]However, as long as the first condition is met, nodes from the set $T[u]$ can be removed one by one until the resulting induced graph becomes connected.

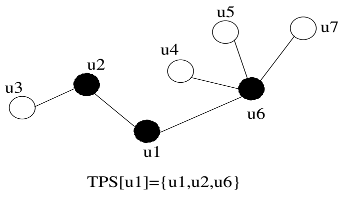

Fig. 3. *MAC* approach

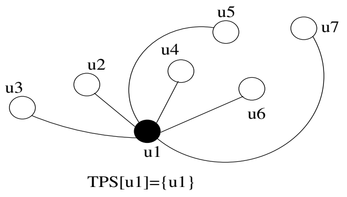

Fig. 4. *MNR* approach

user-mount point mappings. If two TPS members are not directly connected in OG_u, synchronization has to happen through another node $v \in T[u]$. In this case, a profile replica is stored at node v as well; however, still v is not considered as a member of TPS, as it is not a mount point for any neighbor.

1) Minimize the access cost (MAC): The MAC approach prioritizes only the access cost for each friend in a user's social network. Hence, for every user v in OG_u, it assigns the nearest (i.e., with minimum access cost) trusted node connected to v as the mount point, i.e.

$$\forall v \in OG_u, M(v) = v' : C_v^{v'} \leq C_v^i, \quad \forall i \in T[u]$$

Then,

$$TPS(u) = \{v : v \in T(u) \wedge \exists\ v' \in N_G(u) : M(v') = v\}$$

The set $TPS(u)$ contains all members of $T(u)$, which are assigned as *mount points* for friends of u.

In OG_{u_1} (Figure 2), assume that $C_{India}^{Switzerland} = 1$ and $C_{US-West}^{Switzerland} = 2$. The resulting storage configuration for the MAC approach is shown in Figure 3.

2) Minimize the number of replicas (MNR): The MNR approach determines the number of replicas to be maintained for a user, so as to minimize the storage and replica management overhead. In addition, it applies an optimization step in order to minimize the access costs as well.

Our approach exploits the fact that the set TPS can be modeled as the *minimum connected dominating set (MCDS)* on the graph OG_u, with the additional constraint that the members of the MCDS must belong to $T[u]$. Hereby, we modify a greedy algorithm from [8] to solve this variant of the MCDS problem.

Algorithm 1 The MNR algorithm

```
1:  Mark all $v \in OG_u$ as $white$
2:  Mark $u$ as *black*
3:  Mark all neighbors of $u$ in $OG_u$ as *grey*
4:  **while** $\exists$ a $white$ node in $OG_u$ **do**
5:      Select a *grey* $v' \in T(u)$ such that $v'$ has the highest number of $white$ neighbors in $OG_u$
6:      Mark $v'$ as *black* and its neighbors as *grey*
7:  **end while**
8:  $TPS[u]$ is the set of all *black* nodes in $OG_u$
9:  **for all** *grey* nodes $v$ in $OG_u$ **do**
10:     $M_v = v' : C_v^{v'} \leq C_v^i, \ \forall i \in TPS[u]$
11: **end for**
```

3) Minimize storage cost: This approach quantifies the *storage cost* of a given storage configuration ($x = (M, TPS[u])$) and, by exploring the entire solution space, picks the storage configuration with the minimum effective cost. The storage cost is measured in terms of the total cost incurred for accessing and updating the profile content by a user's friends in addition to that of replica synchronization among all TPS members. We do not consider the access cost incurred by non-friend users, even though the system allows such users to access the profile content on case-by-case basis based on the access control settings.

Let n_a^v be the number of times a user v accesses a user u's profile content with each access involving s_a^v units of data access on average. n_u^v and s_u^v represent number of updates and update sizes respectively. Note that this update is performed on M_v, which must be then pushed to the other members of the TPS as well. We assume that these parameters are approximated from the statistics collected over a certain period. To this end, the user u selects the configuration x that minimizes its storage cost, i.e.

$$\arg\min_x \sum_{v \in N_G(u)} \Big[n_a^v \cdot s_a^v \cdot C_{M_v}^v + n_u^v \cdot s_u^v \cdot C_{M_v}^v + \Sigma_{v' \in TPS[u]-\{M_v\}} \left(n_u^v \cdot s_u^v \cdot C_{M_v}^{v'} \right) \Big]$$

We refrain from further discussion of this approach for brevity reasons.

C. Handling updates in social graph and TPS

As social relations evolve, there will be updates in a user's social graph. Moreover, breach of trust or of the social contract to host and enforce access control on behalf of others, may result to updates in the set TPS. Once a node v is removed from a user u's TPS, it is no longer contacted for u's content. All users in $N_G(u)$ for which v is the mount point are informed of this change. Such nodes are mapped to a new temporary mount point (say the node u itself), until one of the three aforementioned algorithms are run to assign them new mount points. We assume the user periodically invokes TPS computation process to accommodate the updates made on OG graph because of updates in the set $T(u)$ or updates in friendship relationships.

Since revocations can happen from the set TPS, users must choose TPS members carefully. Such revocation can happen either because one of the three aforementioned algorithms excludes an existing member from the set TPS, or a breach in the social contract is noticed. However, we believe that mutual social contracts (i.e. reciprocative hosting of data between users) restrict users from maliciously exploiting their hosted data after their removal from the TPS. Handling additions to the set TPS is simple: user u copies the replica of the profile to this new member, which there on, serves access requests.

When a new social relationship is made by user u, we assign as default mount point for the new member the node u itself, or another TPS node that has an overlapping online time interval. Later, the new friend could be assigned a different mount point based on the result of the execution of above algorithms.

When there is a change in the location of some trusted nodes, the graph OG_u may get disconnected. Noticing this, node u should set itself as mount point of the disconnected nodes. We suggest u to adjust its online time frame OT_u in order to make the TPS graph connected in this case.

D. Replica synchronization

We propose that after every update, the concerned mount point *pushes* the update to other TPS members during their online time frame. Note that $OG_u[T[u]]$ is connected. Assume that each TPS member is informed of other members by the user u during TPS creation. Until recent updates reach a mount point, it continues to serve access requests with out-dated content, which is acceptable, as *porkut* aims to eventual consistency among replicas with tolerable temporary inconsistencies.

E. Accessing a user's profile

A user u's profile content is available to his friends in the social network directly through their mount points. New nodes which are not assigned any mount point, can reach the TPS members via the DHT index and access the content after appropriate authorization. However, as already mentioned, the exact organization of the profile content, the request format, and the access control policies are beyond our scope.

IV. Privacy preserving indexing

We advocate privacy-aware indexing of social networking content of users in the system. Such index facilitates content discovery on OSN among friends and allows users with specialized interesting content to reach new potential friends. Furthermore, this index allows for short-lived friendship relations for the exchange of a particular content.

A. Privacy objectives

porkut's indexing service addresses various levels of privacy, which are described below:

- *No privacy*: Content with no privacy requirements is freely accessible by any social network participant.
- *Owner privacy*: The owner of a particular content (i.e. the user in whose profile the content exists) should not be able to be determined with certainty by the index entry for the content.
- *Content and Owner privacy:* In addition to owner privacy, the index entry should not allow someone to determine with certainty whether a particular content item exists in the system or not.

B. Index creation

A conventional DHT-based index has entries in the form *(key,value)* pairs, where a content identifier (i.e., search term on the index) maps to the key and the user profile identifier (UId) maps to the value field. In order to achieve content and owner privacies, *porkut* indexing mechanism uses k-anonymization techniques [9] and (key,value) pairs are replaced by (key[],value[]) pairs i.e., a list of keys are now mapped to a list of values. We call such an index entry as (c, o)- entry, where c is the size of the key list and o is the size of value list. A user inspecting a (c, o)-entry cannot identify which of the content items exist in the system. By analogy, the conventional index entries are referred to as $(1, 1)$- entries. When a user creates an index entry for a content item, he mixes the item identifier with $c - 1$ randomly chosen yet meaningful item identifiers and the owner identifier with $o - 1$ randomly chosen user identifiers, thus creating a (c, o)-entry from a $(1, 1)$-entry. Each user uses a *dictionary* of content items which, for example, can be constructed from all of his accessible content items in the social network. This dictionary is used as input to the content anonymization technique.

Content entries that require *no privacy* use $c = 1, o = 1$. When only *owner privacy* is needed, $c = 1, o > 1$ are employed. Using $c > 1, o > 1$ results in index entries that support both *content and owner* privacies.

Once a user constructs (c, o)-entry, he publishes this entry into the DHT anonymously by employing a Crowds-like source anonymization technique [10], where a crowd is the set of these o users in the index entry. At the end of anonymous routing, a (c, o)-entry is inserted into the DHT as c separate $(1, o)$ entries with each of them having one of the c keys as a pivot. The detailed privacy preserving index construction and its evaluation for a P2P system are described in [11].

A user retrieves from the DHT, the list of $UIds$ associated with his searched key. Then, for each of the $UIds$, he contacts one (again k-anonymized) of its corresponding TPS members for the content item that he looks for. Our index allows strangers (i.e. non-friend users) to contact each other based on interesting content. Authentication and authorization follow this step.

V. Related Work

There is significant related work on privacy issues in social networks. The possibility for involuntary personal information leakage in current social networks is highlighted in [12], e.g. by means of certain OSN features like annotating or tagging user photos, and its effects are demonstrated in [4].

Lockr system [13] improves the privacy of centralized and decentralized content sharing systems. It allows users to control their own social information by decoupling the social networking information from other OSN functionality using social attestations, which act like capabilities. However, these social attestations are used only for authentication and authorization is enforced using separate authorization policies. Persona [14] uses attribute-based encryption to realize privacy-preserving OSNs. The attributes a user has (e.g., friend, family member, colleague) determine what data he can access. The NOYB approach [3] adopts a novel approach for preserving content privacy. They observe that if users address their privacy issues themselves by hosting encrypted content on OSNs, they could be expelled from the OSN by the OSN operator. Hence, they propose to replace users profile content items with "fake" items randomly picked from a dictionary. NOYB encrypts the index of the user's item in this dictionary and uses the ciphered index to pick the substitute. On the other hand, flyByNight [15] encrypts the users' content that hosts on the OSN.

Recently, the issue of using decentralized infrastructures for organizing OSNs in a privacy-preserving manner, was addressed by the research community [1], [7], [16]. PeerSon [16] adopts encryption mechanisms for content storage and access control enforcement. It uses a two-tier architecture in which the first tier is a DHT, which is used as a common storage by all participants. The second tier consists of peers and contains the user data. The DHT stores the meta-data required to find users. Peers connect each other directly, exchange the content, and then disconnect. [7] addresses privacy in OSNs by storing profile content in a P2P storage infrastructure. Each user in the OSN defines his own view ("matryoshka") of the system. In this view, nodes are organized in concentric rings, having nodes at each ring trusted by the nodes in its immediate inner ring, with the user node being the center of all rings. The user's profile data is stored encrypted at the innermost ring, which is accessed by other users through multi-hop anonymous communication across this set of concentric rings. In the DHT, an entry for a user with the list of nodes in the outermost ring is added. Thus, [7] achieves both content privacy (using encryption) and anonymity of searcher and hosting nodes, yet limited content discovery and profile availability, as opposed to our approach.

In [1], a decentralized OSN, Vis-à-Vis is proposed, where a user's profile content is stored at his own machine called as virtual individual server (VIS). VISs self-organize into P2P overlays, one overlay per social group what has access to content stored on a VIS. Three different storage environments are considered: cloud alone, P2P storage on top of desktops, a hybrid storage, and their availability, cost, and privacy trade-offs were studied. In desktop-only storage model, a *socially-informed replication scheme* was proposed, where a user replicates his content to his friend nodes and delegates access control to them. However, normally, a uses trusts only a fraction of his friends to the extent of delegating access control enforcement, as considered in our *porkut* approach along with online time information. Our earlier work [6] considered access control delegation in P2P systems in terms of trust transitivity.

Tribler [17] is a P2P file sharing application which exploits friendship relationships, tastes and preferences of users to increase the performance of file sharing. However, in Tribler, users host their own profile and therefore profile placement for high availability and low access or consistency cost are not considered. Finally, LifeSocial [18] is a P2P-hosted OSN where users employ public-private key pairs to encrypt profile data that is stored in a distributed way and is indexed in a DHT. Friends can read a user's profile based on a symmetric key that is encrypted with their public keys. However, data privacy and profile availability are not considered in [18].

VI. Conclusion and Future Work

In this paper, we presented the initial design of porkut, a privacy-preserving decentralized OSN. We emphasized on satisfying high availability and lookup efficiency of scattered OSN profiles. The users geographical locations and online time statistics were exploited in deciding the user's profile storage points. Three algorithms with different cost minimization objectives were presented for selecting the set of nodes that host OSN profiles, while preserving high availability. As a future work, we plan to deploy the *porkut* system, and study its performance, availability and privacy characteristics in detail.

Acknowledgement

This work was funded by the Swiss Nano-Tera OpenSense project (Nano-Tera ref. 839_401).

References

[1] A. Shakimov, A. Varshavsky, L. P. Cox, and R. Cáceres, "Privacy, cost, and availability tradeoffs in decentralized osns," in *Proc. of the WOSN*, 2009.

[2] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. of the 7th Internet measurements conference*, 2007.

[3] S. Guha, K. Tang, and P. Francis, "Noyb: privacy in online social networks," in *Proc. of the WOSP*, Seattle, WA, USA, 2008.

[4] B. Krishnamurthy and C. E. Wills, "On the leakage of personally identifiable information via online social networks," in *Proc. of the WOSN*, 2009.

[5] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "Opendht: a public dht service and its uses," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 73–84, 2005.

[6] N. Rammohan, Z. Miklos, and K. Aberer, "Towards access control aware p2p data management systems," in *Proc. of the 2nd International workshop on data management in peer-to-peer systems*, 2009.

[7] L. A. Cutillo, R. Molva, and T. Strufe, "Privacy preserving social networking through decentralization," in *Proc. of the WONS*, 2009.

[8] L. Ruan, H. Du, X. Jia, W. Wu, Y. Li, and K.-I. Ko, "A greedy approximation for minimum connected dominating sets," *Theoretical Computer Science*, vol. 329, no. 1-3, pp. 325 – 330, 2004.

[9] L. Sweeney, "k-anonymity: a model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, 2002.

[10] M. K. Reiter and A. D. Rubin, "Crowds: anonymity for web transactions," *ACM Trans. Inf. Syst. Secur.*, vol. 1, no. 1, 1998.

[11] R. Narendula, T. G. Papaioannou, and K. Aberer, "Panacea: Tunable privacy for access controlled data in peer-to-peer systems," 2010, EPFL Technical Report 148337. http://infoscience.epfl.ch/record/148337.

[12] I.-F. Lam, K.-T. Chen, and L.-J. Chen, "Involuntary information leakage in social network services," in *Proc. of the 3rd International Workshop on Security*, 2008.

[13] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman, "Lockr: better privacy for social networks," in *Proc. of the CoNEXT*, 2009.

[14] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *Proc. of the ACM SIGCOMM*, 2009.

[15] M. M. Lucas and N. Borisov, "Flybynight: mitigating the privacy risks of social networking," in *Proc. of the WPES*, 2008.

[16] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "Peerson: P2p social networking: early experiences and insights," in *Proc. of the ACM EuroSys Workshop on Social Network Systems*, 2009.

[17] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips, "Tribler: a social-based peer-to-peer system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 2, pp. 127–138, 2008.

[18] K. Graffi, P. Mukherjee, B. Menges, D. Hartung, A. Kovacevic, and R. Steinmetz, "Practical security in p2p-based social networks," in *Proc. of the IEEE LCN*, October 2009.