LotusNet: tunable privacy for distributed online social network services

Luca Maria Aiello, Giancarlo Ruffo
Computer Science Department - Università degli Studi di Torino
Corso Svizzera 185, 10149 Turin, Italy
{aiello,ruffo}@di.unito.it

* * *

PREPRINT

Abstract

The evolution of the role of online social networks in the Web has led to a collision between private, public and commercial spheres that have been inevitably connected together in social networking services since their beginning. The growing awareness on the opaque data management operated by many providers reveals that a privacy-aware service that protects user information from privacy leaks would be very attractive for a consistent portion of users. In order to meet this need we propose LotusNet, a framework for the development of social network services relying on a peer-to-peer paradigm which supports strong user authentication. We tackle the trade-off problem between security, privacy and services in distributed social networks by providing the users the possibility to tune their privacy settings through a very flexible and fine-grained access control system. Moreover, our architecture is provided with a powerful suite of high-level services that greatly facilitates custom application development and mash up.

Keywords: social networks, information leakage, privacy, peer-to-peer, DHT, access control, reputation, folksonomies

1. Introduction

The Social Web is probably the most pervasive and ubiquitous effect of the Web 2.0 phenomenon. Today, *Online Social Networks* (OSNs) have become one of the main means of interaction and information dissemination for Web users [1, 2]. Most of the modern *Social Network Services* (SNSs) are based on a collaborative paradigm, where content that is published and shared between participants is produced by the users themselves. Such aggregation of personal and possibly sensitive information belonging to several million of users is the real wealth owned by today's major OSN providers (e.g., Facebook, MySpace), that offer free registration to their social platform in exchange for a free access to user data.

Entrusting such a high volume of information to an external provider involves many hazards that can seriously undermine the privacy of OSNs users. Of course, the possibility of an undesired information mining performed by the provider is one of the risks, but the most alarming threat to privacy concerns the *leakage* of the personal data from the context in which they were originally defined [3]. In fact, very often the content management policies adopted by SNS providers allow third parties to exploit the OSN user information for many different purposes (e.g., faceted advertising) without any awareness of the user. Another dangerous implication of leakage is the information *linkage* [4], i.e., the possibility for an unauthorized third party to aggregate data from different social data centers in order to infer non-expressed information about the identity or the behavior of a user. In Facebook, the brand new instant-personalization service [5] and the access policies to personal content for third-party applications [6] are evident examples of such possibility.

Therefore, the actual risk is that users lose completely the control over the diffusion of their own information. Even if most of existent SNSs offer the possibility to change some privacy settings or profile visibility options, it is often impossible to fully customize settings to obtain appropriate privacy policies. Furthermore, the general trend followed by providers is to relax more and more the privacy constraints of the default user settings (e.g., [7]).

Although, until the recent past, users could be misinformed about who actually has the grant to access their online information [8], the awareness on SNSs privacy leaks is rapidly growing and spreading among the OSN users, thanks also to the relevance attributed by the media to this topic. Recently, also many voluntary protest initiatives are blooming on the Web [9, 10, 11], reflecting the fact that the user demand for privacy is becoming stronger.

For these reasons, to restore the control of the users on their own information has become a major challenge for the research community today [12]. Replacing centralized SNS providers with peer-to-peer (p2p) services [12, 13], where no central authority can claim the right to exploit the data, is a proposal that is achieving widespread success among the scientific community. However, implementing complex SNSs on purely decentralized layers does not guarantee privacy per se and introduces several practical problems that do not occur in centralized implementations.

First, widespread p2p systems like structured overlay networks suffer from many *security* issues [14] that can considerably undermine the stability of the network, and of course of the applications layered above. This point is even more important for social applications, that usually have high robustness requirements motivated by the high-quality user experience they must offer. Second, in a p2p open environment data can be potentially accessed by everyone. For this reason, proper access control policies should be implemented in order to offer a full customizability of the level of data *privacy*. Last, p2p overlays offer a very restricted and low-level API. Since social applications usually require high level primitives like inter-application notifications or identity management functions, the distributed social layer should expose a suite of higher-level *services* that prevent an excessive overhead for the application development task.

To satisfy simultaneously security, privacy, and quality of service requirements is the crucial challenge in the research on distributed OSNs, especially because, if taken to the extremes, such three requirements tend to conflict with each other. Our contribution to this field is the definition of *LotusNet*, a framework for the implementation of p2p SNSs based on a Distributed Hash Table (DHT) system. LotusNet uses strong authentication of peers at overlay level to provide security and stability to social applications, high level services, and fine-grained discretionary access control to private resources. Furthermore, LotusNet does not impose a very high, fixed privacy level to other requirements' detriment, but allows the users to *tune* the *trade-off* between privacy and services which is most favorable to them. Finally, our approach allows an easy application mash up and relieves the applications from many service implementation details, thus making their development very quick and easy.

Previous proposals (see Section 2) focus mainly on the problem of access control or on single social services like contacts discovery. Compared to other p2p OSNs, our framework is wider because it considers security, privacy and services as a whole and the trade-off between them, providing a complete and coherent solution without the imposition of any constraint or assumption on the nature of applications or on the structure of the social network.

The remainder of the paper is structured as follows. Section 2 presents an overview on previous works aimed at privacy preservation in OSNs. Section 3 proposes a general-purpose OSN model and lists the requirements we want to assure in such system. In Section 4 the main features of our identity-aware DHT are exposed. Sections 5 and 6 are the core of the work and defines the architectural modules that compose the LotusNet architecture. Section 7 inspect the effects of a crawling attack on LotusNet and proposes some additional countermeasures to preserve user information privacy. Conclusions are given in Section 8.

2. Related work

The high practical relevance of the privacy issues in OSNs has recently strongly attracted the interest of the research community. Broadly, the works published in these last few years can be classified into two main currents. The first focuses on the design of fully decentralized privacy-aware OSNs. Here, besides dealing with information confidentiality and access control, authors try to propose effective solutions to implement the main services provided by classic server-based architectures (e.g., data availability) on the distributed layer.

Contrarily to the p2p-based proposals, the other current addressed the privacy goal assuming the existence of a centralized content provider, but proposing workarounds to avoid undesirable information mining or defining guidelines to enhance the privacy services offered by existent providers. This choice is motivated by the advantage in preserving the quality of service offered by a classic client-server model.

2.1. Peer-to-peer OSNs

Some research papers about the implementation of social application on p2p layers were made even before the viral spreading of OSNs (e.g., [15]), but of course the idea of using a p2p framework to solve ONSs privacy issues is very recent [13, 12].

The *PeerSon*[1] system [16] is one of the first p2p design for a OSN. The goal of user information privacy is achieved through symmetric encryption of content stored in a DHT. The p2p network serves mainly as a lookup service: once the two endpoints' contacts have been retrieved from the DHT, direct connections are established. When a friend is offline, update notifications are managed asynchronously through the DHT using a pull approach. Full decentralization and encryption prevent, respectively, the "Big Brother" effect and network crawling activities aimed to data collection. However, advanced access control features like highly dynamic group membership are not taken into account.

The *Safebook*[2] p2p OSN [17, 18, 19] focuses on resource availability as well as content privacy and end-to-end communications confidentiality. It combines a DHT with another p2p network called *Matryoshka*. Peers in the Matryoshka are connected by trust bonds: user items are stored at highly trusted neighbors. The DHT is used to store the contacts of the Matryoshka members and encryption is used to preserve content privacy. The Safebook design provides also the presence of a trusted, offline identification service to avoid Sybil attacks.

A similar approach is adopted in [20]. Here, the problem of content availability on fully decentralized social networks is faced with a p2p storage model based on the concept of trust. Items replicas are stored at a set of nodes trusted by the content owner, called Trusted Proxy Set (TPS). The initial selection of TPS nodes and their churn dynamics are handled accordingly to the geographical location of nodes, in order to place the data as close as possible to nodes that often access the content. References to TPS nodes are published on a DHT layer. Data on the DHT are indexed with a k-anonymization technique to grant both owner and content privacy.

In [21] authors propose a DHT-based storage with access control capabilities for social shared resources. Encrypted items are published together with several copies of the secret key; each copy is encrypted with the key of a user who have access permission for that resource. A drawback of this solution is that when the access control list of a specific content must be changed, a new updated item has to be built and stored again. User registration phase and secure p2p communications are inspected as well.

Participants discovery in fully decentralized OSN is discussed in [22]. Authors define a gossip protocol, implemented on the Tribler network [23], for user discovery. When the target friend is unavailable and the search initiator goes offline, a set of online *helper* peers is delegated to perform a periodic probing activity aimed at the retrieval of the searched peer's contact.

[1] http://www.peerson.net
[2] http://lnx.0131mmp.com/Safebook_site

Vis-a-Vis [24] is a scheme for decentralized OSN that targets high content availability. Each user stores her personal data in a Virtual Individual Server (VIS) that is kept on the local user's desktop and replicated on a cloud infrastructure. When the desktop is offline, the cloud service is activated and the availability is not interrupted. Further replicas of the content are hosted among the VISes of the social contacts, that are connected together through a p2p overlay network. However, the rental cost of the cloud service makes this proposal not very practicable.

2.2. Privacy solutions for centralized OSNs

NOYB[3] [25] is an encryption tool based on a substitution cipher, useful to hide the real user's profile information from the eyes of the data center. In fact, the provider is prevented from telling fake encrypted data from valid information. A secret key shared between NOYB groups, together with a public dictionary, stored by a Trusted Third Party, are used for encryption and decryption. A proof-of-concept sketched on the Facebook case is presented. This approach applies only to textual fields and it is not suitable for any other kind of sensitive information, like social connections, group affiliations, photos, etc.

Lockr[4], a discretionary access control system adaptable to centralized OSNs as well as to the BitTorrent streaming system, is presented in [26]. The idea is to decouple the social information (i.e., the contact list) from the content the users share with others. In this approach, the list of friends is not explicitly stored anywhere; instead, every user distributes a signed social attestation to each of her social contacts; only users that have a proper social attestation are allowed to access the resources. In order to avoid replay attacks, the attestation is verified through a zero-knowledge protocol. In a centralized setting, a dedicated server is responsible for the attestation validity check. In BitTorrent, thanks to the introduction of a signed social torrent, attestation verification can be managed in a p2p manner.

FlyByNight [27] is another attempt to mitigate privacy risks in OSNs by introducing a privacy-aware architecture that can be integrated with existing centralized providers. Also here, cryptography is used to conceal content to the provider; an external server is used as key repository.

Persona [28] is a privacy-aware OSN where user data are kept into a centralized untrusted storage. Information privacy is accomplished using encryption. In particular, Attribute Based Encryption is used to flexibly manage group membership and revocation of privileges. A possible integration with existing OSNs is drawn.

A similar approach is described in [29], where authors propose a client-server model with untrusted server as a privacy-aware OSN architecture. In this work, the server is used as a public storage without any access control policy (it offers only simple PUT and GET primitives). The task of preserving the privacy of

[3] http://adresearch.mpi-sws.org/noyb.html
[4] http://www.lockr.org

user information is delegated to clients that upload on the server encrypted and equally-sized data blocks in order to hide both resources' content and size. Only users with proper keys can access to the items. Identity verification and key distribution are managed in a p2p fashion.

In [6], authors address the privacy problems related to the disclosure of personal information through APIs provided by online social services like Facebook and OpenSocial. They propose an alternative social platform design that prevents third party applications from obtaining real user data. They introduce a *privacy-by-proxy* anonymization technique that do not reveal any non-public information to external applications or to the application users that are not granted with the proper privileges. This goal is achieved by customizing the interaction protocol between the social service provider and the applications.

In [30], authors propose a machine learning methodology to infer resources access control policies to be applied to the set of social contacts, based on a small learning set of representative friends for which access control settings are actively specified by the user. The position of a friend on a specific network cluster of the friendship graph is used together with her profile information as training feature for the classifier.

2.3. Our direction

In this work, we recur to the p2p paradigm to arrange a privacy-aware solution for OSNs. This choice is motivated by several reasons. First, we believe that applying a blurring mask to the user data layer in order to hide sensitive information stored in a centralized environment is a solution that would be not accepted by the service providers themselves, whose terms of service often impose very strict conditions on information inserted by users (e.g., [31]). Furthermore, the implementation of privacy-aware solutions that imply to build up a centralized infrastructure for data management would be too costly if a very large audience must be supported. Conversely, a p2p solution can be realized with negligible infrastructural cost and offers a more transparent privacy service due to its intrinsic distributed nature.

In a previous work, we sketched the fundamental traits of a DHT-based OSN [32]. In the present paper we extend it going deeper into the architectural details and adding new significant features to the architectural modules. In particular, we define an additional layer of *core services* layered on the DHT node that include a new notification service. As a result, we present an extended API that can meet the most of privacy and functional requirements that social applications have.

3. OSN requirements

A SNS can be defined, in its most general meaning, as a *customizable suite of inter-operable, identity-based applications*. In this context, every user composes its own combination of applicative modules, or *widgets*, into a customized

application suite, where every widget can share data with other possibly heterogeneous, widgets running locally or remotely. Note that this is a very general model because no assumption is made about the nature of widgets or the structure of exchanged data, and both synchronous and asynchronous communications are allowed. Given this general definition, we inspect a set of desired privacy, security, and service requirements that are common to a very wide range of social widgets. Such requirements are at the basis of the architectural design of our distributed social framework.

3.1. Privacy requirements

Confidentiality. Any kind of content created by any widget should be accessible only to those authorized to have access. This property has a wide meaning. First, it implies the definition of access control policies that allow a flexible and fine-grained specification of grants. Furthermore, in a OSN context, *fully-customizable* confidentiality is the most effective patch to stop information leakage.

Ownership privacy. The creator or the owner of a content should be enabled to not disclose the ownership information to other users. The relational tie between a user and a content of a certain type could be in fact a very valuable information, potentially detrimental to the user privacy.

Social interactions privacy. Social ties and the data flowing on them can tell much information on the behavior of individuals and on the dynamics of groups in a networked environment [33]. For this reason, a user should be able to arbitrarily hide the interaction between local and remote widgets.

Activity privacy. The type and number of widgets that compose a user application suite must be considered sensitive user information, because they partly reveal the nature of user activity on the network. Flexible privacy settings should be provided to tune the exposure of the application suite composition to the public.

3.2. Security requirements

Channel authentication. Communication channels should be two-way authenticated, so that both the initiator and the recipient can check the identity of the partner. Unauthenticated channels are potentially vulnerable to social engineering attacks. *Phishing* aimed to sensitive data collection [34], for example, is mainly based on persuading the target user that the attacker is a known and trusted entity. Strong authentication, combined with an educated behavior of users, greatly reduces also the risk of identity theft, that is one of the main issues in today's Web-based OSNs [35].

Data integrity and authenticity. Information that is published by widgets and exchanged with remote entities must not be modifiable by any non-allowed user. The genuineness of content should also be assured through ownership verifiability, to avoid making wrong associations between a content and a user that is not its owner.

Non-repudiation. Users are fully responsible of their actions on the network. In many distributed services, the agreement on social or micro-economical transactions passes through messages that are exchanged by the parties involved. The property of non-repudiation of the content's ownership lays the foundations for traceability of user actions and is therefore a deterrent to frauds, to the spreading of false information on the network, and to *spamming* activities.

3.3. Service requirements

Content availability. Access to data should not be conditioned by the connection status of the owner. Even when the owner is offline, users with a proper permission should be able to access the data.

Flexible communications. End-to-end communication can be synchronous or asynchronous. A widget should be able to listen for live notifications and to recover messages that were sent during its offline time.

Easy integration. The *mash up*, namely the composition of existing building-block services into more complex applications, is at the basis of the Web 2.0 paradigm. For this reason, the interoperability between social applications should be facilitated as much as possible.

Search facilities. Users are interested in acquiring new contacts and in exploring the resources published in the OSN. Proper search engines should allow to find the desired items, but in compliance with privacy requirements stated above, if possible.

Reputation management. Collaborative environment like OSNs can often rely on reputation and trust notions to balance social interactions or negotiations. For this reason, widgets should be provided with common tools to quantitatively express their perceived reputation of other participants and to convey their reputation beliefs to remote widgets.

3.4. The wall, the fence, the garden

Privacy, security, and service requirements (summarized in Table 1) lay on three orthogonal axes; an ideal OSN platform should preferably meet at the same time all the requirements that are listed in the previous sections, thus maximizing the level of fulfillment for each of the three properties. However, it is clear that such peak cannot be reached due to the partial conflict between some requirements that lays on different axes. For instance, security properties like traceability and authentication may often conflict with ownership and social interaction privacy, which can in turn affect the effectiveness of search facilities or the significance of a reputation system.

To effectively depict this trade-off we can use a *walled garden* metaphor. Users in OSNs are like gardeners who work for their plants and flowers to thrive. As long as the garden is open to the external environment it receives the benefit of sun and rain and new seeds can root on its ground, carried by the wind or by other gardeners visiting. But, without any shelter, the garden also suffers from the bad weather and can fall prey to vandals and thieves. Gardeners can rise fences to turn away malicious visitors and walls to protect the garden from the

Privacy *(Wall)*	**Security** *(Fence)*	**Services** *(Garden)*
Confidentiality	Channel authentication	Content availability[1]
Ownership privacy	Integrity and authenticity	Flexible communications[2,3]
Interaction privacy	Non-repudiation	Easy integration[2]
Activity privacy		Search facilities[4]
		Reputation management [5]
Sections 5.2, 5.3	Section 4.1	Sections [1]6.4, [2]5.1, [3]6.1, [4]6.2, [5]6.3

Table 1: Summary of the social network service requirements, partitioned in the three main categories. In the bottom row we report the Sections of the paper in which we discuss the fulfillment of such requirements in LotusNet.

weather, but high **fences** (*security policies*) prevent good visitors to easily access to the **garden** (*services* and user *resources*) and thick **walls** (*privacy settings*) can deprive the flowers of light and water; this is the *gardener's dilemma*.

LotusNet is designed to reach a good trade-off between the three aspects. LotusNet is based on secure interaction protocols and enables a set of powerful services, but its architecture is not bound to a single system-defined privacy setting. Instead, LoutsNet provides the users with the ability to tune their own privacy configuration with fine granularity over a range of possible privacy policies, thus giving wide freedom to the user to open doors and windows in their own privacy wall.

4. DHT layer

We base the architecture of our social framework on a single p2p network and in particular on a Distributed Hash Table. This choice is motivated mainly by the good properties that DHTs offer [36]. Extreme scalability, maintenance and dissemination of stored content, resistance to high churn, efficiency in locating rare objects and simplicity of interface are very important qualities that these platforms offer in a transparent way. Furthermore, given that the main task of the social widgets is to share and collect data, a distributed storage is one of the simplest means to implement this service.

The DHT we use is called *Likir* [37], a customized version of Kademlia [38]. Likir enhances the simple and efficient Kademlia protocol with strong identity management at overlay level: as well as a numeric identifier, Likir nodes are marked with a user identity and interactions between peers are two-way authenticated. In the following, we report a quick overview on its features and on the interface it exposes to applications.

4.1. Secure design: raise the fences

Likir can be used basically like any other DHT except for the fact that users must fulfill a preliminary user registration procedure in order to receive a certified identifier for their DHT node. Likir architecture provides a Certification Service (CS) for this purpose.

We assume that each user has a pair of RSA keys and an *OpenId* account. Before the very first bootstrap, a user contacts the CS through a web service and sends her public key and OpenId using a simple submission form. Once the OpenId is validated, the CS produces a signed certificate containing the user's OpenId and public key, an expiration time and a random 160bit string that will represent the Kademlia identifier of the DHT node. We call this certificate a $LikirId$.

Once a user has obtained her $LikirId$, it does not need to contact the CS until its expiration. If the CS goes offline, the user registration service becomes unavailable but the network activities are not affected at all, because the users that previously obtained their $LikirId$ can join the overlay without querying the CS anymore. Since the expiration time can be chosen to last even many years, we can state that the CS is *not* a single point of failure of the system. In our design, the CS is implemented with a centralized server.

Once the $LikirId$ is obtained, the node can join the network by performing the bootstrap procedure. The Likir overlay protocol is a customization of the Kademlia Remote Procedure Calls with an additional secure exchange of $LikirId$s for authentication purposes. We refer to [37] for the details of the protocol and for considerations on its overhead sustainability, while here we focus only on its main properties.

First, overlay communications are two-way authenticated. Authentication plus the binding of the user identity with a fixed and random Kademlia ID has a very high impact on the security of the p2p network. It has long been known [39] that most of the security attacks to structured p2p systems leverage the fact that overlay nodes are marked with very loose identities, i.e., numeric identifiers that are chosen arbitrarily by peers at each bootstrap. Loose identities allow attackers to position Sybil nodes [40] into specific portions of the DHT. Controlling strategically positioned nodes allows to eavesdrop p2p traffic directed to entire region of the DHT keyspace, for censorship or denial of services purposes [41]. Also unauthenticated channels, that are unavoidable if weak identities are assigned to overlay participants, make the routing tables vulnerable to the injection of fake entries [41]. In Likir, authentication plus the binding of the user identity with a fixed and random Kademlia ID effectively counteracts this kind of threats, thus making the p2p layer more robust.

The second Likir's main property is the verifiable ownership of content. This is achieved attaching certificates, signed by the owner, to every content published on the DHT; since the owner identity and the hash of the resource are included in the certificate, resource integrity and ownership verifiability are assured. This feature is the main building block for identity-centered services on the DHT because it allows a secure identity-based resource retrieval. Please note that the authenticated interaction protocol and the certificates allow to satisfy at overlay level all the security properties defined in Section 3.2.

As final remark, it is very important to notice that, even if a p2p implementation of the CS would be feasible [42], the presence of a centralized service for certification is not in contrast to our decentralized vision of the OSN, basically because the CS has the same capability to observe the network of a simple Likir

node. The CS does not know any information about the subscribing users but the fact that they participate to the Likir service; as explained is Section 5.2, even simple Likir nodes can easily determine whether a user is on the Likir network. This is a very minimal information which cannot tell anything about the user's activity, or the resources she owns, or the social contacts she has. Knowing the binding between the user identity (the OpenId) and the corresponding DHT address (the Kademlia ID) does not help in any way to monitor the incoming/outgoing network traffic of that user. If the CS falls in untrusted hands, the attacker cannot disclose any private user information and not even the list of nodes in the network because, in principle, the CS is not even required to store the *LikirId*s that have been issued in the past. Of course, the CS is crucial for the *security* of the Likir network: for instance, an attacker who steals the CS private key could generate Sybil nodes marked with arbitrary Kademlia IDs, and for this reason the CS should be properly protected, as well as a certification authority in a PKI.

4.2. API

Likir offers an essential but very powerful set of primitives in its interface to applications. In particular, here we present an extended version of the three main primitives defined in [37].

1. PUT*(key, obj, type, public, ttl).* It is the basic insertion primitive. It lookups the DHT nodes responsible for storing objects marked with *key* and puts the binding between *key* and *obj* in their storages. *type* is a string denoting the application-specific type of the object and *ttl* determines the expiration time of the content. The *public* parameter is a flag used to determine the visibility of the published resource. If set to *true*, the object will be downloadable by any peer in the network, otherwise proper access control policies must be applied (see Section 5).
2. GET*(key, type, userId, recent, grant).* Queries the DHT for content marked with *key*. Only objects marked with a certain *type*, or belonging to a user identified by *userId* could be retrieved. Please note that even if an identity-based resource filtering could be implemented also in classical DHT by attaching proper identity tags to published items, in Likir the ownership of content is verifiable in a *secure* way thanks to certificates. The *recent* boolean parameter allows to retrieve only the last published version of the objects (e.g., a common strategy to release an update of a content on a DHT is to publish a new version of the object with same key and type). Finally, the user can specify a *grant* for the access to a non-public resource; access control techniques are discussed in Section 5.
3. BLACKLIST*(userId).* Adds the specified user to a local blacklist. Every new incoming message from *userId* will be discarded; this is possible because overlay interactions are authenticated in Likir. This operation is useful to expunge misbehaving users from the network (see Section 5).

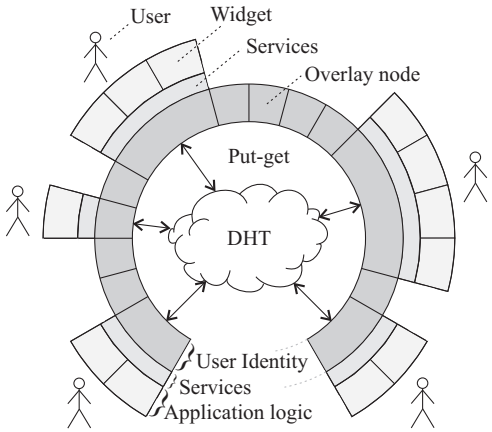

Figure 1: Conceptual scheme of the LotusNet social network service

5. Architecture

Starting from the Likir platform, we define *LotusNet*, a framework for a privacy-aware implementation of SNSs. In LotusNet, widgets are layered on the p2p network and can interact by exchanging objects through the DHT. The goal of LotusNet is to build higher-level functions upon Likir to realize the requirement-driven OSN model defined in Section 3.

The API offered by the Likir middleware represents the first step toward the goal, however we need to build higher-level functions upon it. The resulting idea is depicted in Figure 1: a custom suite of widgets relies on a layer of social network services, i.e., an extension of the DHT primitives offering higher-level functionalities useful for OSNs. Such services are directly based on the API of the overlay node, which encapsulates the identity management and authentication features. The DHT cloud is the primary medium for p2p communications.

In the following, we inspect the three main aspects that realize this idea. First we present a general framework for widgets interaction and integration, then we discuss in detail the access control facilities of our architecture and finally we define a set of service modules that compose, together with the Likir interface, the complete API for widgets. During the dissertation, we highlight the architectural aspects that satisfy the requirements we defined in Section 3. The detailed architecture of a Likir client is depicted in Figure 2; every element of this scheme will be expounded in this Section and in the next one.

5.1. Basic interaction and integration

In classical DHT-based applications, like file-sharing (e.g., eMule), despite the preliminary index-side filtering that storage nodes perform based on the keywords specified in the search query, the content retrieval process usually returns a big quantity of results that often contains several almost-equivalent

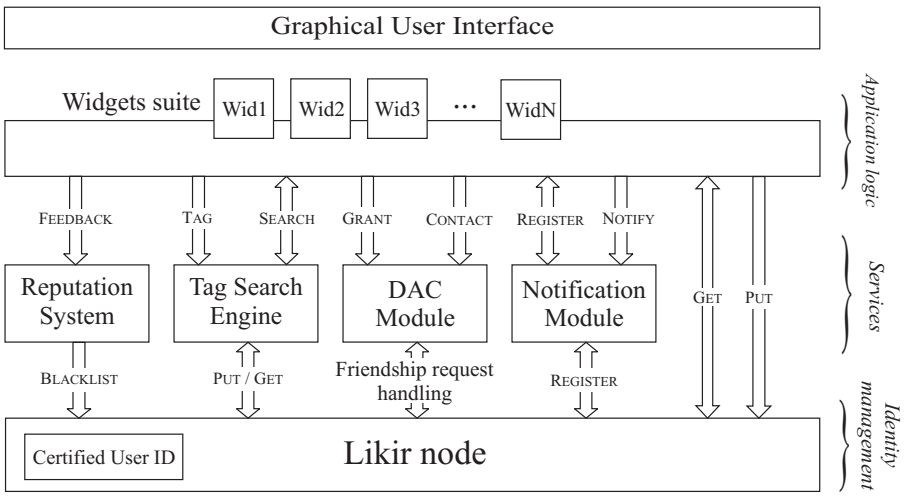

Figure 2: Architectural scheme of the LotusNet platform

versions of the same resource. This is perceived as an acceptable output by the user, who simply selects the resource that best fits her needs among the returned set in order to start the download.

The same situation leads to a completely different scenario in OSNs. In most of the cases, OSN users search for *specific* resources that belong to *people they know yet* (their "friends" or "contacts"). For this reason, they expect to receive a very precise response to their queries; for example, a search for the last wall post on my friend's blog is not supposed to return the whole set of posts of that person or the last wall updates of the whole list of the friends of mine.

Instead, a very sharp resource retrieval can be made with Likir thanks to its filtering facilities. If all the filtering parameters are set in GET, at most *one* resource is returned, i.e., the most recent item published by the selected owner for the specified key and type. This possibility allows to manage the access to frequently updated information (e.g., user status modification). Two relevant, unique aspects characterize this kind of filtering. First, the identity-based filtering is performed in a fully verifiable way thanks to certificates paired to the resources. Second, the whole filtering is made by index nodes, thus relieving the local application from *any* filtering responsibility.

It is worth noticing that widgets are not forced to communicate exclusively through the DHT and can establish direct connections if needed. In this case, the distributed storage can be used for preliminary Diffie-Hellman exchange in order to establish a secure out-of-band connection. Since the key agreement protocol is performed on a fully authenticated layer, the new secure channel will be also authenticated as well as encrypted.

Identity-based resource retrieval has a very good implication also on integration between different widgets. Since identity is managed at overlay level, all the data published by the same node are marked with the same user identity,

regardless of the nature of the widget that generated the content. Furthermore, suppose that widgets publish their API, describing the internal structure of the items they manage together with the lookup keys and types associated to them; this is a realistic assumption given the fact that publishing an application API is a practice that is already adopted by the vast majority of Web 2.0 services. Given such premises, integration becomes easy, because every widget can gather and aggregate content from other different widgets owned by the known social contacts just invoking a simple method.

As an illustrative example, suppose that users A and B use a social calendar widget (W_{SC}) and an instant messaging application (W_{IM}). A very simple kind of integration could be displaying A's daily commitments in the B's chat panel, and vice versa. Supposing that the social calendar API is known, and that the information is not protected by any kind of access control policy, you have to call just a single operation in the chat code in order to retrieve the correct data to display on the panel. In A's client, for example, it will be the following:

$$obj = \text{GET}(K(W_{SC}), current\_date, B, true, none) \tag{1}$$

where $K(W_{SC})$ is the lookup key chosen by the calendar application and supposing that the string representing the current date is the content type for the commitments of the present day. Clearly, this mechanism can be extended to any other kind of cross-application integration.

The extreme openness of this scheme, where every application can potentially cooperate with any other module, rescues the OSN from the so called *walled garden problem*. Paradoxically, in fact, even if the user information leaks from centralized OSNs, trampling on the user privacy rights, it is often difficult to share data between different OSNs or to reuse social applications in a profitable way, due to the heterogeneity of the platforms or to narrow content management policies [43]. Some centralized solutions, like the OpenId-compliant Global Social Platform [44], have been proposed to overcome this problem, but without receiving sufficient consensus so far. Instead, full decentralization and OSN modularity allow to compose any kind of OSN as an arbitrary combination of cooperating widgets and even many different OSNs can be linked together in the same way.

5.2. Building the social graph: access control and contact discovery

We outlined a scenario of maximum interoperability, where every user can potentially interact with any other and access the whole information stored in the DHT. Even if this flexibility grants a very high level of customizability of the OSN structure, it totally lacks privacy. In order to preserve privacy, the shared information, which is potentially available to everyone must be channeled into the communication pipes that participants create by establishing social ties. In other words, we must model the social acquaintance graph that links together OSN users and then limit the resource sharing only to the pairs of linked users. Furthermore, users should be able to define with a fine granularity the portion

of the personal information that is shared with arbitrarily-defined groups of neighbors.

Cryptographic access control techniques suitable for groups with dynamic membership has been extensively studied in the literature, e.g., [45, 46]. In particular, solutions based on *key regression* schemes manage the eviction of a group member by redistributing to remaining members a new key that is used to encrypt new items but that can be used as well to get the previous group key. Under the assumption that a former member can have access to all the data published before her exclusion from the group, a *lazy revocation* [47] policy can be adopted: an old item is re-encrypted with the new key only when an authorized member modifies it, thus avoiding the re-encryption of all the items at every membership change.

Even if this strategy may be suitable in UNIX-like shared file systems, it is too less flexible in a OSN context, for several reasons. First, since the vast majority of items (e.g., posts, photos) which are shared in social network are written once by their owner and never modified, lazy revocation is often not applicable. Furthermore, even if, in principle, users formerly granted to access the content could have saved it locally, we argue that preventing a user whose grant is revoked to download the old data would be even a better guarantee for privacy that would be appreciated by many users. Finally, the most important point is that the mentioned key management techniques are not suitable for *overlapping* groups, commonly used in OSNs: if an item is accessible by several different groups it must be encrypted several times, thus greatly increasing the complexity of keys and groups management.

To provide a more flexible access control we recur to signed *grants* to specify permissions. Grants are associated with social contacts and not with shared resources, so their number does not grow with the quantity of resources owned or with the number of rules in the privacy policy. A grant certificate, produced by a user A for a user B is composed as follows:

$$Grant_A(B) = \{A||B||regExp||expireTime\}_{Sig_A} \tag{2}$$

It contains the identities of the owner and of the granted user, an expiration time and a regular expression that is a compressed list of all the allowed content types. The token is signed by the issuer.

Of course, grant certificates can be used if the entities that have the duty to store the user data are able to verify the validity of the grant. The key idea that allow grants in our setting is the use of authenticated channels for the overlay communications. In fact, when an overlay index-node receives a request for a protected resource, it can ask for a valid grant before returning it, being able to securely verify if the identity of the querying peer is the same identity specified inside the grant. Note that the index-node can check the signature validity because, during the content insertion phase, the publisher's $LikirId$, which contains her public key, is sent along with the object. Finally, the regular expression allows to specify permissions for an arbitrary subset of resource types.

In practice, the use of grants is made in the DHT API. As mentioned in Section 4.2, Likir's PUT primitive allows to specify if the stored resource requires

a permission to be accessed (parameter $public = false$), and the GET function takes a grant as a parameter.

Revocation of grants is implemented through expiration. The problem of choosing a proper life span to balance best the grant renewal cost with the maximum period during which a user with a revoked permission can still access to a protected resource has not a single optimal solution. In a fully-customizable framework, such duration should be chosen by the applications or even by the user itself, depending on the trust he places on the granted friends. Anyway, since the number of grant is limited by the number of contacts (that is quite limited in the vast majority of cases), we suggest that short durations (e.g. one week) can be used.

Grants are very flexible and powerful, but they do not hide published content from the eyes of the index-nodes, that can mine their local storages as they wish, even without a proper certificate. To shelter user information from this potential privacy breach, content are encrypted. Please note that in this case encryption does not imply a complex key management system. In fact, each participant can use a *single* encryption key to protect the full set of its data; the key is shared with the known contacts and it does not need to be replaced when the access control policies change.

An access control mechanism based on grants associated to users, instead on permissions attached to resources, draws implicitly the edges of the social network. With grants, the social network topology should not be explicitly mapped anywhere: the existence of a $Grant_A B$ means that a social tie has been established between A and B. Of course, the nature of ties can differ depending on the set of capabilities that the corresponding grant specifies. Besides, the asymmetrical structure of grants allows to build both *directed* and *undirected* social networks, depending on whether grants are reciprocated or not.

Implementation

In LotusNet, the *Discretionary Access Control Module* (DACM) is responsible for the management of the individual social connections and to set privacy policies by assigning grants. The DACM is layered directly on the Likir node and has a very simple behavior. At its startup, it creates a daemon listening on a TCP port and puts its address on the DHT, using a lookup key extracted from the user identifier. Then it enters in a passive state, waiting for incoming TCP connections or for calls performed by local widgets. In particular, DACM's API to applications is the following:

$$\text{CONTACT}(userId) \tag{3}$$

$$\text{GRANT}(userId, regExp) \tag{4}$$

$$\text{GETGRANT}(userId) \tag{5}$$

$$\text{GETSELFGRANT}() \tag{6}$$

When a widget W, in the user A's application suite, wants to find a new friend B, it calls CONTACT(B)[5]. This method triggers a DHT lookup for B's DACM contact. If the contact is found, it means that B is on LotusNet. However, at this stage, nothing else is known about B, not even the widgets he has installed on its client. In order to determine if B is using widget W, the GRANT(B,^{W\.}) function is called[6]. Doing so, a direct message containing a grant certificate for B is sent to B's DACM; this is interpreted by B as a new incoming friendship request. If the request is accepted, B reciprocates it by sending a proper grant for A. An analogous interaction occurs between widgets for the periodical renewal of grants.

For the sake of brevity, here we omit many less relevant implementation details. For example, here A directly sends a grant to ask for B's friendship, but more complex interactions could be implemented as well. For example, A may want to release the grant only if it will be reciprocated, or even more complex transactions can occur. A very general framework for resources negotiations which can be possibly applied to this case is described in [48].

The call of GRANT(userId,regExp) is used also to update the regular expression for a formerly granted user, in order to extend or reduce its permissions; in this case, a new signed grant is produced to replace the old one. The DACM stores in a local database both received and released grants. Methods GETGRANT(*userId*) and GETSELFGRANT() are used, respectively, to get the grant received by a social contact from a local database and to obtain a self-signed grant to access to the information stored by the local widget on the DHT.

Just for completeness, we note that this API can be profitably extended to manage grant distribution also to local widgets. If every widget is supplied with a grant that contains only the minimal permissions that allow its correct activity, *trojan horse* widgets are prevented to fetch private information stored on the DHT by other widgets in the same application suite and spread it publicly on the network without any permission.

5.3. Tuning the privacy level: lift up the walls

To show to what extent the privacy requirements listed in Section 3.1 are satisfied by the LotusNet design, we discuss two attack scenarios whose main actors are LotusNet users with two different roles.

In the first scenario, the attacker is a generic node that aims to disclose the private information of a target user, from which she has not received any grant. The attacker can query the index nodes that are supposed to store the target's data, trying to find a breach in the access control mechanism. Supposing

[5] We suppose that B's identifier is known. A specific indexing application for user searching could be built, however this implies the disclosure of some personal information that are revealed for indexing purposes (e.g. hometown, schools attended, etc.)

[6] We suppose that content types begin with the main widget's name, that we assume to be unique, followed by a dot. The regular expression is written in POSIX notation, and it means "any string beginning with *W*."

that index nodes behave correctly and properly follow access control protocol, confidentiality trivially holds because peers without a proper grant cannot access protected resources. Furthermore, supposing that index nodes return a generic "content unavailable" message in response to unauthorized requests for protected resources, an attacker cannot infer the type nor the index key of the content stored by the target user; so, owner and activity privacy are satisfied. Last, social interaction privacy is not breakable because the attacker cannot learn what is the set of users that are granted to access the private content.

The attacker can also try to infer some information on the activity of the target user by analyzing the incoming overlay network traffic. However, the messages that the attacker can possibly receive from the target or from peers interested in retrieving the victim's data are just Likir lookup requests. Lookup messages are used to locate the index nodes for a given item and their payload contains only the DHT lookup key. This little information is not enough to learn if the lookup requests are aimed to store or to retrieve a content, what kind of content is looked up and who is the owner of that content.

In the latter scenario, the attacker is an index node and aims to disclose private information from the content she stores and the incoming requests she receives. Since stored information is encrypted, confidentiality holds, but the other privacy requirements cannot be fully satisfied because they are in conflict with the authenticated p2p communication that the Likir layer provides. In fact, the index node knows the identity of the owner of every stored item because of signed certificates (owner privacy), she is aware of the item types because they are specified by the publisher in the PUT primitive for indexing purposes (activity privacy), and she can log all the identities of the peers who make a GET request for a certain item, thus inferring social relations between the owner of the requested content and every querier of that content (social interactions privacy).

The second scenario confirms the intuition of conflicting OSN requirements presented in Section 3.4: full privacy is not reachable if all the security and service requirements are maintained. However, even in this case, LotusNet grants a good level of privacy. In fact, since the location on the overlay is determined by the Kademlia identifier inside the LikirId, a node cannot arbitrarily position itself on the keyspace because the identifier is generated randomly by the trusted CS. Being placed in a random overlay position, a malicious index node cannot intercept the data belonging to a specific target user. Moreover, since different widgets presumably use several different lookup keys to remotely store their data, user data are randomly scattered on the DHT; therefore, recovering the full information about a user is practically unfeasible for an attacker. In a nutshell, the information held by a index node is little and fragmentary, therefore the risk of privacy infringement is very low and we believe that it can be considered acceptable in most of the cases.

However, to reduce further the risk of privacy violation for information that is particularly sensitive, LotusNet allows the *tuning* of the privacy level required by the widgets. This can be done simply storing the sensitive data among a set of trusted contacts specified by the user [17]. This approach is easy to

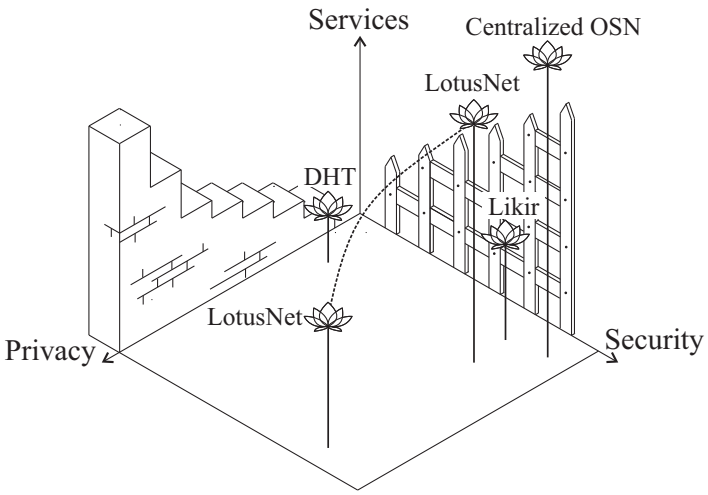

Figure 3: Trade off between security, privacy and services in OSNs depicted as a walled lotus garden. The Position of lotus flowers respect to the wall and the fence represent respectively the privacy and security levels, while the height of the flowers represent the potential number (and quality) of service that the user can benefit by sharing her data with others.

follow because in Likir it is easy to get the overlay ID from a known user ID because they are permanently tied together by the signed $LikirId$. Practically, a trivial customization of the Likir PUT and GET primitives (PUT^+ and GET^+ in the following) which allow to explicitly specify the list of index nodes that are required to store/return the data is enough to implement this strategy. Of course, in order to not disclose the sensitive information to untrusted peers, index nodes must not apply the usual content dissemination policies for data published with this customized PUT operation.

Exploiting this opportunity increases the privacy level while preserving all the security properties of the Likir network, but at the cost of decreasing the quality of service. First, trusted nodes may not be enough in number to grant a good redundancy of the data and if uptime distributions are not very heterogeneous (which is likely if their geographical locations are very close), the availability of data is partly compromised. Second, since trusted nodes will be located in different regions of the overlay keyspace, the PUT^+ operation should perform a different lookup for every selected index node, thus slowing down the data save. Symmetrically, the GET^+ operation executed by data consumers should probably perform several lookups to locate at least one online trusted index node, thus slowing down the search procedure. However, it is worth notice that these drawbacks can be reasonably limited if the set of trusted peers is selected among the users who are potential consumers of the data and if this maximum privacy policy is applied only for resources with critical privacy requirements.

Figure 3 depicts the trade-off between security, privacy and service require-

ments recurring to the metaphor of the garden recalled in Section 3.4; we symbolically map different frameworks for OSN development on the three-dimensional space, in order to compare them. Specifically, OSNs based on classic DHTs can satisfy few privacy and security requirements and offer a minimal set of services; the security level is considerably higher if the Likir DHT is used. LoutsNet benefits from the Likir security property and offers the opportunity to arbitrarily increase the privacy level from a minimum threshold (all published items are public) to a level where everything is protected by access control policies and content is stored at trusted peers; as stated before, as privacy level is increased, the quality of service (in terms of both efficiency of basic insertion/retrieval operation and of remote widgets that could benefit from that hidden information) decreases. In any case, privacy in LotusNet is always higher if compared to centralized social networks, since centralized SNS providers hold the complete information about every user in the network.

6. Services: grow the garden

A OSN does not live of privacy alone. The success of a SNS platform is mainly determined by the services it offers to users and applications. From our point of view, ongoing projects on p2p OSNs have not yet focused enough on the realization of complex social tools. In this Section we give a contribution in this direction by defining three core services of the LotusNet architecture: notifications, reputation management and high-level content indexing. The defined services extend the basic Likir API with operations 3-11, thus offering a high-level set of social primitives at the basis of the construction of complex SNSs. The set of methods offered by the LotusNet interface is summarized in Table 2.

6.1. Notifications

Social widgets can adopt synchronous and/or asynchronous protocols to communicate, depending on the application logic. If synchronous interaction between peers is needed (e.g., instant messaging application), it is reasonable to think that the two endpoints establish a direct connection to manage the data stream flowing in both directions. Conversely, asynchronous communications (i.e., *notifications*) fit better applications that are quiescent most of the time (e.g. wall posting service in OSNs); in this case, usually an always-online entity stores the message and delivers it as soon as the target user is online.

When using an overlay for message exchange, notifications can be stored on the DHT, but the *pull-based* approach at the basis of any DHT would force applications to continuously probe the network in search of new incoming messages. Several publish-subscribe schemes for DHTs aimed at equipping the overlay network layer with a *push-based* notification service has been proposed in the past [49]. Solutions proposed in literature (e.g. [50, 51]) are very articulated and include features like multicast communication, dynamic groups management, continuous complex queries, and so on.

Here we use a simpler and lightweight unicast notification service that is suitable for end-to-end notifications between social widgets. The idea is rooted in the extreme precision that DHT routing algorithms have in locating the overlay node whose identifier is the nearest to the lookup key, even in presence of a high churn rate. In particular, it has been shown that the Kademlia lookup procedure is characterized by a very high precision [52]: regardless of the distance between the querier identifier and the target key, the set of replica nodes located by the lookup contains the peer whose identifier is the nearest to the target key with probability near to 1.

Since in Likir every peer has a *fixed* position on the keyspace, every user can determine the exact overlay position of any of its friends. Notifications can thus be easily implemented by sending PUT massages with a lookup key which is equal to the Kademlia identifier of the user to be notified. Notification messages can be easily distinguished from all the other overlay message received because they are marked with a key which is *exactly* equal to the local node ID; since the Kademlia keyspace has a cardinality of 2^{160}, it is quite unlikely that random messages are mistaken for proper notifications.

We extend the Likir API with the operation REGISTER(HANDLE), that simply specifies an application handle to which all the notification messages are passed. Furthermore, we introduce a *Notification Module* (NM) as a middleware between DHT notifications and applications. The NM registers on the Likir node as notification handler and offers two main methods to the widgets above:

$$\text{REGISTER}(applicationName) \tag{7}$$

$$\text{NOTIFY}(userId, applicationName, obj) \tag{8}$$

Using REGISTER, widgets are notified with all the incoming messages marked by the specified application type, while the NOTIFY operation sends an arbitrary notification object, marked with the application type of the notifying widget, to the target $userId$. The NM manages transparently the binding between the $userId$ and its corresponding Kademlia ID and probes the DHT at every startup to search for missed notifications.

6.2. Folksonomic content search

A resource search engine is a central component in modern SNSs. Search tools are used by OSN participants to expand their knowledge to thematic contexts that reside beyond their local social cluster. For this reason, we believe that an effective instrument for content search cannot be left out of consideration in p2p designs of OSNs if they are intended to be competitive with their centralized counterparts.

In our framework, like in many other decentralized OSN designs, content encryption is one of the means to obtain content confidentiality. Obfuscating data enhances privacy but has also a negative side-effect on searching procedures. In fact, any content-based resource indexing is not applicable if the structure of shared objects is completely obscured by ciphers.

Certainly, encrypted objects can be found specifying their type and DHT key in classic DHT lookups. This approach is effective as long as the entity that initiates a search procedures is an automated application, that knows by protocol the set of keys and types associated to the objects it has to insert or retrieve from the overlay. But this system becomes too rigid when the search query is submitted directly by a human user. In fact, DHT lookups are limited to the well-known *exact-match problem* [53]: the knowledge of the *exact* lookup key is required in order to retrieve the content from the distributed storage.

The straightforward solution adopted in most of the DHT applications, like file sharing, is to compute the lookup key hashing the words of a representative content name. Nevertheless, the name assigned to a resource by its publisher may be insignificant to people that are interested in its retrieval and, in practice, this approach hides many objects from their potential consumers.

A more suitable alternative comes from the collaborative tagging paradigm brought into vogue by the Web 2.0 philosophy. In tagging systems, resources are marked with arbitrary labels assigned directly by the users; the result is a corpus of classifications called *folksonomy* in contrast with the taxonomical paradigm of content categorization. Recurring to folksonomies is often necessary in very populated collaborative systems because the extremely high and continuously growing number of objects would make unfeasible for a team of experts to perform a classification. Furthermore, it has also been shown that the overall quality of the folksonomic indexing structure is comparable with the classic taxonomies, in terms of accuracy, completeness and consistency [54].

For this reasons, we define a folksonomy-based search engine for our social architecture. Given the distributed setting in which we are positioned, the bindings between tags and resources are directly stored by the users in the DHT. The idea is that a OSN user can mark a resource with a tag just publishing a reference pointer to that resource (i.e., its DHT key and its type) using the hash code of the assigned tag as lookup key. Subsequent searches for that tag will return the set of resources labeled with it. It is worth noting that if indexed resources are not public, a proper grant is anyway needed to get the content referenced by the pointer.

In addition to the basic indexing feature, folksonomies can be exploited to realize also a *navigational* paradigm of content search, where a user can drift from a selected category to another based on the semantic correlation between them. Examples of such paradigm can be found in classic query-based search engines like Google Wonder Wheel [55] as well as in tag-based search engines like Yahoo! Tag Explorer [56].

In folksonomic navigation, the most similar tags to the selected one are returned to the querier at each step. Anytime, the user can choose to continue the navigation or to retrieve the resources marked with the current (and with all the previously selected) tag. Implementing this possibility in our architecture implies to explicitly map the similarity relations between tags on the DHT. In particular, in order to consistently maintain the information on the tag-tag similarity graph over time and to efficiently manage the frequent addition of both tags and indexed resources, we rely on the $DHARMA$ approximated algorithm

we defined in [57]. In DHARMA, tag-tag similarity is based on the notion of co-occurrence [58] (i.e., the similarity score between two tags is the number of resources that the two tags label in common) and the update of similarity arcs is managed with an approximated strategy that limit the cost of the both tagging and navigation operations to a constant number of lookups.

As a result of these considerations, we can define a folksonomy-based search engine layered on Likir together with its high-level API:

$$\text{TAG}(label, obj) \tag{9}$$

$$\text{SEARCH}(tag) \tag{10}$$

The semantic of the operations is self explanatory. The SEARCH method returns both the set of tags similar to the specified label and, of course, the set of tagged resources.

6.3. Reputation management

Reputation is at the basis of many social dynamics, in real world as well as in OSNs. Online market places, question and answer bulletin boards and fora are just some examples in which reputation and trust play a key role in social transactions and in the selection of reliable partners. Very often, reputation is strongly *context-dependent*. For instance, a good photographer in a picture sharing system could be a bad movie reviewer and vice versa. For this reason, reputation systems are designed to operate within the boundaries of single applications and, typically, users are not represented with a single "karma" but with many, possibly conflicting, reputation*s* depending on the application they are plunged in.

However, introducing a *context-independent* notion of reputation could play a very important role in improving the service of collaborative p2p systems and of social networks in particular. Spamming, phishing, frauds and trolling are among the today's most serious threats for the security and usability of SNSs. Although these attacks are often made through the channels provided by single applications, their particular gravity has repercussions not only on the context of the application itself, but also on the quality of service of the underlying OSN: a social platform which does not effectively filters spam or does not provide any tool to detect fraudulent participants is indeed unreliable and, ultimately, not attractive to the public.

In centralized OSNs, malicious users like spammers could be traced and banned by the provider, which is able to detect anomalous behaviors since it can monitor all the activity happening on the social network. Conversely, decentralized OSNs should be complemented with a handle to spread the information about misbehaving users that are detected by honest participants. In LotusNet, even if independent services can be accessed through different widgets installed in the application suite, the actions performed by a user in different social contexts are always referable to its *single* identity, determined uniquely by the $LikirId$. This feature allows to build a cross-application reputation system, where a reputation is associated directly to the misbehaving peer and not simply to a widget user.

The *blacklist* operation, offered by the Likir API, is meant to realize this idea. Interactions with blacklisted users will be avoided *at overlay level*. This means that any message or content, coming from any widgets belonging to the blacklisted user, will be automatically discarded by Likir. When a appreciable consensus is reached among the network, the misbehaving node will be not able to interact with the majority of honest peers, thus being confined to a dead network partition.

Relying on such low-level primitive, we can define a Reputation Module (RM) in our architecture's service layer. Its interface offers a single operation:

$$\text{FEEDBACK}(userId, score, proof) \tag{11}$$

The idea is that an application can specify an evaluation on the behavior of a known user with a numeric score. Together with that, a proof of the good/bad behavior of the evaluated user can be stored by the RM. The proof is simply the content for which its behavior is being evaluated; for example, in a generic question and answers system, a post with inappropriate language can be an evidence of user misbehavior. Note that, since the ownership of content published on the DHT is verifiable, malicious users cannot forge fake proofs against honest peers, thus avoiding *social mobbing* phenomena. When the reputation score falls below a certain threshold, the bad user is locally banned using the BLACKLIST primitive.

Here we are not interested into define the details of a specific reputation system (i.e., how scores are managed), also because different schemes can fit better different types of OSNs (remind that several complex OSNs can coexist in LotusNet). Several suitable reputation schemes that potentially suits our context can be found in literature (e.g. [59]). Instead, the crucial point we want to underline is that the reputation information can be securely spread across the network thanks to proofs associated to users. Even a simple gossiping protocol could be effective: when the RM blacklists a user, it sends the proofs of its bad behavior to neighbors in the social networks that, in turn, can blacklist the same identity too and forward the information to their neighbors. Since, due to the *small-world effect* [60], the typical diameter of social networks is quite small, the information is diffused in few hops across a great portion of the network, thus rapidly excluding the malicious user from social activities.

The adopted approach has several advantages. First, our scheme strongly contrasts the *whitewashing* phenomenon, thus limiting the risk that a user whose reputation is stained in a social context can rejoin with a different identity or can damage other systems where its identity is still unknown. Second, global reputation makes very high the cost of any bad behavior, thus being a powerful deterrent against malicious peers. Last, since in pure p2p OSNs single peers take upon themselves the critical responsibility of storing data from other participants, it is reasonable that malicious peers should be not trusted anymore to store user data; information spreading techniques that are implemented at overlay level are able to transparently replicate the data stored by the blacklisted user to another more reliable index node.

6.4. On storage service

Assuring content availability is one of the most important challenges in the implementation of distributed SNSs. In our system, the DHT is intended to be the main storage layer for items generated by LotusNet widgets. Availability, lookup accuracy and efficient content dissemination are *intrinsic* properties of DHT storage platforms. Well-known incentives mechanisms for DHT peers to stay online [61, 62] can be applied to Likir as well. Moreover, note that Likir is a general-purpose DHT which can simultaneously support many other services beside LotusNet (e.g., classical file sharing); so, LotusNet clients could profitably advantage also of the storage provided by DHT nodes that are not OSN participants.

Many DHT-based utilities that complement the basic storage services with more complex functionalities like durability or unlimited content size have been presented in literature (e.g., [63]). We do not want to stuck on a specific storage scheme, because different applications could have very different needs regarding the storage. Instead, our aim is to define a flexible framework that offers a basic, privacy-aware storage facility and that can be used as fundamental building block to create more complex storage services without losing the property of privacy-awareness.

This is accomplished with the use of grants. If the items published on the DHT are used as pointers to external services, the same privacy mechanism that combines grant-based access control with content encryption can be used. In a nutshell, grants are self-contained entities that can be reused also in services that are external to the DHT. The possibility of reusing grants preserves the content confidentiality across different storage systems.

Of course, if the DHT is used as main mean of storage, non-stop availability of data comes at the price of relying on untrusted storage servers, i.e., the index nodes. Usually, this is not considered to be a good policy [64] unless basic security/privacy requirements like confidentiality, data integrity and authenticity are satisfied [65]. We have shown that the LotusNet architecture transparently satisfies such requirements by design and, additionally, DHT redundancy property is a good shelter against denials of service like the *smashing attack* [64]. Moreover, like pointed out in Section 5.3, thanks to the strong binding between overlay ID and user identity, LotusNet is flexible enough to allow widgets to store their data at trusted index nodes, like the known friends, at cost of potentially losing full-time availability.

7. Crawling attack

Previously in Section 5.3 we showed that a malicious entity cannot crawl the DHT searching for private information and cannot even position probe nodes on the overlay to intercept sensitive data. However, a weakness of this scheme reside exactly in the delicate phase of creation of new contacts. Indeed, the attacker can leverage the incautious behavior of some users inducing them to

Module	Operation	Description
Likir	PUT($key, obj, type, public, ttl$)	Publish an object on the DHT
	GET($key, type, userId, recent, grant$)	Retrieve an object from the DHT
	BLACKLIST($userId$)	Put $userId$ in a local blacklist
	REGISTER()	Register a handle for incoming notifications
DAC	CONTACT($userId$)	Search for $userId$ in LotusNet
	GRANT($userId, regExp$)	Issues a grant for $userId$ for types specified by $regExp$
	GETGRANT($userId$)	Gets the grant received from $userId$
	GETSELFGRANT($userId$)	Gets a self-signed grant
Notification	REGISTER($applicationName$)	Register an application-specific handle for incoming notifications
	NOTIFY($userId, applicationName, obj$)	Send a notification to a specific user and application
Tag Search	TAG($label, obj$)	Tag the specified object
	SEARCH(tag)	Tag-based search. Return a set of tags and a set of resources
Reputation	FEEDBACK($userId, score, proof$)	Rate a user's behavior

Table 2: Summary of the LotusNet services to applications

accept the attacker's contact request and, consequently, to issue grants for it. In this Section we want to show to what extent malicious peers can extract user information from the network using this technique. Additionally, we propose further security expedients that could be adopted to improve the privacy preservation also in this scenario.

We suppose that the attacker is a common peer that runs a *crawling* application whose behavior is sketched by the pseudo-code in Algorithm 1. Basically, the crawler starts from a seed user (lines 1-2), asking for its friendship and offering in exchange a grant for the full set of its resources (lines 5-6). If the request is accepted (line 7), the set of widgets specified in the received grant is extracted (line 8). For each widget, all the accessible information is retrieved from the DHT and possibly added to a local database (lines 9-10). The crawler can also try to extract informations about target user's friends directly from the widgets' data (line 11). The whole procedure is then repeated for the new discovered contacts in a breadth-first fashion (lines 12,3,4).

Of course, extensive crawls must be executed by *automatic* procedures like the crawling application described. For this reason, a first counterattack to limit the effectiveness of crawls is recurring to a contact establishment procedure that is resistant to robots. Requiring that both parties solve a puzzle which implies human interaction (e.g., captchas [66]) before they can connect to each other with a social tie is a very lightweight and inexpensive solution that can severely limit this kind of attacks.

Apart from this consideration, if the target user is somehow tricked to issue a grant for the attacker, a portion of its private information is inevitably disclosed. The extent to which this happens is proportional to the level of trust that the

Algorithm 1: Crawler widget

```
   Input  : seed, a user identifier
 1 List contacts = new List ()
 2 contacts.add (seed)
 3 while contacts not empty do
 4     String userId = contacts.pop ()
 5     contact (userId)
 6     grant (userId,".*")
 7     if grant received from userId then
 8         Set widgets = getWidgets (userId)
 9         for w in widgets do
10             retrieveInformation (w)
11             List newContacts = retrieveContacts (w)
12             contacts.add (newContacts)
```

target user places in the new friend. Here, two cases may occur.

In the first case, the attacker introduces herself with a user identifier that recalls to the target node a person she knows already in real life (or in other online social contexts). The belief of being interacting with a highly trusted peer can lead the victim to disclose a high quantity of its personal information. This problem can be mitigated by adopting some basic *web-of-trust* features borrowed from the PGP setting. Users can produce special tokens for their social contacts in order to certify the binding between the user and its identity. When contacting a peer, a user exhibits its own identity certifications; since link formation process in social networks often involves *triadic closure* [67] (i.e., people becoming friends have often at least one friend in common), it is likely that the recipient of the friendship request directly knows some of the certifiers. The absence of known certifiers should warn the user about a potential privacy risk.

In the latter case, the attacker's identity is completely new to the target user. In this situation, the grants should be released gradually. To this end, a profitable collaboration between the Reputation Module and the DAC Module could be exploited, posing an upper bound to the permissions that can be issued for a social contact according to its the reputation level. Afterward, the growth of reputation in time determined by positive feedbacks received from other known peers or from the local user may cause the expansion of the grant.

If all these countermeasures fail, can the attacker seize also the identifiers of the victim's social neighbors (line 11 in Algorithm 1)? Or, worse, can the attacker retrieve also data *belonging* to its neighbors? We previously noticed that the information about the user's personal contact network is not explicitly mapped anywhere, so the attacker may learn it only analyzing the resources published by the widgets it has been granted the access to. Moreover, widgets may often not require to store user identifiers explicitly. For example, in publish-subscribe applications, where a user publishes updates for a group of *followers*,

there is no need to store the list of recipients together with the published data, simply because the access to the content is managed using grants only.

However, even if the attacker succeeds in learning a part of the social network topology, the most important thing is that the personal information of that contacts cannot be unveiled, since the grants gained by the attacker are useful to retrieve only the victim's protected content. In order to access the information of discovered friends, the attacker must reiterate the whole crawling procedure.

In this setting, considering all the countermeasures we presented, a crawl of the p2p network aimed to data collection would be very costly and limited to very inattentive users. Of course, the greatest defense ever against privacy leakages is a continual user awareness.

8. Conclusions

An increasing awareness about issues on privacy of personal information in centrally-managed Online Social Networks (OSNs) is strengthening among Social Network Services (SNSs) users. In order to tackle this problem, we follow the path traced by a recently-formed research community that proposed to migrate SNSs on peer-to-peer (p2p) layers in order to escape the opaque management of sensitive information performed by service providers.

We presented *LotusNet*, a new architectural scheme of a DHT-based OSN that focuses on three strictly-interconnected aspects related to OSNs: security, privacy and services. First, we showed that binding a user identity to both overlay nodes and published resources results in a enhancement of overlay network robustness and allows to define a secure identity-based resource retrieval. Then, using this identity-aware API, we shaped the structure of the SNS in terms of services that our framework offers to custom social applications.

We provide confidentiality by assigning the access control responsibility to overlay index-nodes. A content stored in the DHT is returned to the querier only if a proper grant, signed by the its owner, is shown to the item keeper. Flexibility of grants and the possibility to store data at trusted index nodes allow the users to tune the desired privacy level for the activity around any particular widget or resource. Furthermore, grants exchanged between peers implicitly model the social network's topology, facilitating the establishment of new social contacts.

Besides access control, we define a set of core services useful for a wide range of social applications. We propose an asynchronous notification service entirely supported by the overlay layer, a tag-based search engine service to overwhelm the exact-match lookup problem of structured p2p network, and a handle for a reputation system intended to expel misbehaving users in a cross-application fashion.

LoutusNet is the product of design choices that are agnostic with respect to the layers above. We do not make any assumption on the structure of applications or messages that compose the SNS. As a result, we presented a social framework that can host several, possibly interconnected SNSs thus creating

an environment without closed *information silos*. Also, the range of facilities available is much wider respect to previous related works.

Likir, the overlay network we used, together with some of the core services we defined, have been implemented in the form of a Java library [68]. The LotusNet project activity will continue for both implementation and large-scale experiments in a real-world setting.

References

[1] A. Ostrow, Social networking more popular than email, `http://mashable.com/2009/03/09/social-networking-more-popular-than-email` (March, 2009).

[2] A. Kazeniac, Facebook takes over top spot, Twitter climbs, `http://blog.compete.com/2009/02/09/facebook-myspace-twitter-social-network` (February, 2009).

[3] B. Krishnamurthy, C. E. Wills, On the leakage of personally identifiable information via online social networks, in: WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks, ACM, New York, NY, USA, 2009, pp. 7–12. doi:http://doi.acm.org/10.1145/1592665.1592668.

[4] B. Krishnamurthy, C. E. Wills, Characterizing privacy in online social networks, in: WOSP '08: Proceedings of the first workshop on Online social networks, ACM, New York, NY, USA, 2008, pp. 37–42. doi:http://doi.acm.org/10.1145/1397735.1397744.

[5] Facebook Inc., Social plugins and instant personalization, `http://www.facebook.com/help/?page=1068` (April, 2010).

[6] A. Felt, D. Evans, Privacy protection for social networking platforms, in: Web 2.0 Security and Privacy 2008 (in conjunction with 2008 IEEE Symposium on Security and Privacy), 2008.

[7] K. Opsahl, Facebook's eroding privacy policy: a timeline, `http://www.eff.org/deeplinks/2010/04/facebook-timeline` (April, 2010).

[8] R. Gross, A. Acquisti, Information revelation and privacy in online social networks, in: WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society, ACM, New York, NY, USA, 2005, pp. 71–80. doi:http://doi.acm.org/10.1145/1102199.1102214.

[9] We're quitting Facebook, `http://www.quitfacebookday.com` (May, 2010).

[10] Web 2.0 suicide machine, `http://suicidemachine.org` (2010).

[11] D. Grippi, M. Salzberg, R. Sofaer, I. Zhitomirskiy, The diaspora project, `http://www.joindiaspora.com` (2010).

[12] P. Antoniadis, B. L. Grand, Self-organised virtual communities; bridging the gap between web-based communities and P2P systems, International Journal of Web Based Communities 5 (2) (2009) 179–194. doi:http://dx.doi.org/10.1504/IJWBC.2009.023964.

[13] S. Buchegger, A. Datta, A Case for P2P Infrastructure for Social Networks - Opportunities and Challenges, in: WONS'09: 6th International Conference on Wireless On-demand Network Systems and Services, Snowbird, Utah, USA, 2009.

[14] G. Urdaneta, G. Pierre, M. van Steen, A survey of DHT security techniques, ACM Computing Surveys`http://www.globule.org/publi/SDST\_acmcs2009.html`, to appear.

[15] H. Lundgren, R. Gold, E. Nordström, M. Wiggberg, A distributed instant messaging architecture based on the Pastry peer-to-peer routing substrate, in: SNCNW 2003, Swedish National Computer Networking Workshop, Stockholm, 2003.

[16] S. Buchegger, D. Schiöberg, L. H. Vu, A. Datta, PeerSoN: P2P Social Networking - Early Experiences and Insights, in: SNS'09: 2nd ACM Workshop on Social Network Systems, Nürnberg, Germany, 2009.

[17] L. A. Cutillo, R. Molva, T. Strufe, Leveraging social links for trust and privacy in networks, in: INet Sec 2009. Open Research Problems in Network Security. Zurich, Switzerland, 2009.

[18] L. A. Cutillo, R. Molva, T. Strufe, Privacy preserving social networking through decentralization, in: WONS 2009, 6th International Conference on Wireless On-demand Network Systems and Services, Snowbird, Utah, USA, February 2-4, 2009.

[19] L. A. Cutillo, R. Molva, T. Strufe, Safebook: feasibility of transitive cooperation for privacy on a decentralized social network, in: AOC 2009, 3rd IEEE International WoWMoM Workshop on Autonomic and Opportunistic Communications, Kos, Greece, June 15, 2009.

[20] R. Narendula, T. G. Papaioannou, K. Aberer, Privacy-aware and highly-available OSN profiles, in: COPS '10: the 6th International Workshop on Collaborative Peer-to-Peer Systems. In proceedings of WETICE 2010 : 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, to appear.

[21] K. Graffi, P. Mukherjee, B. Menges, D. Hartung, A. Kovacevic, R. Steinmetz, Practical security in p2p-based social networks, in: LCN'09: 34th IEEE Conference on Local Computer Networks, 2009.

[22] S. Abbas, J. Pouwelse, D. Epema, H. Sips, A gossip-based distributed social networking system, in: WETICE'09: 18th IEEE International Workshops

on Enabling Technologies. Groningen, Netherlands, IEEE Computer Society, June 29 - July 1, 2009, pp. 93–98.

[23] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, H. J. Sips, TRIBLER: a social-based peer-to-peer system, Concurrency and Computation 20 (2) (2008) 127–138. doi:http://dx.doi.org/10.1002/cpe.v20:2.

[24] A. Shakimov, A. Varshavsky, L. P. Cox, R. Cáceres, Privacy, cost, and availability tradeoffs in decentralized osns, in: WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks, ACM, New York, NY, USA, 2009, pp. 13–18. doi:http://doi.acm.org/10.1145/1592665.1592669.

[25] S. Guha, K. Tang, P. Francis, Noyb: privacy in online social networks, in: WOSP '08: Proceedings of the first workshop on Online social networks, ACM, New York, NY, USA, 2008, pp. 49–54. doi:http://doi.acm.org/10.1145/1397735.1397747.

[26] A. Tootoonchian, K. K. Gollu, S. Saroiu, Y. Ganjali, A. Wolman, Lockr: social access control for web 2.0, in: WOSP '08: Proceedings of the first workshop on Online social networks, ACM, New York, NY, USA, 2008, pp. 43–48. doi:http://doi.acm.org/10.1145/1397735.1397746.

[27] M. M. Lucas, N. Borisov, Flybynight: mitigating the privacy risks of social networking, in: WPES '08: Proceedings of the 7th ACM workshop on Privacy in the electronic society, ACM, New York, NY, USA, 2008, pp. 1–8. doi:http://doi.acm.org/10.1145/1456403.1456405.

[28] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, D. Starin, Persona: an online social network with user-defined privacy, in: SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication, ACM, New York, NY, USA, 2009, pp. 135–146. doi:http://doi.acm.org/10.1145/1592568.1592585.

[29] J. Anderson, C. Diaz, J. Bonneau, F. Stajano, Privacy-enabling social networking over untrusted networks, in: WOSN '09: Proceedings of the 2nd ACM workshop on Online social networks, ACM, New York, NY, USA, 2009, pp. 1–6. doi:http://doi.acm.org/10.1145/1592665.1592667.

[30] L. Fang, K. LeFevre, Privacy wizards for social networking sites, in: WWW '10: Proceedings of the 19th international conference on World wide web, ACM, New York, NY, USA, 2010, pp. 351–360. doi:http://doi.acm.org/10.1145/1772690.1772727.

[31] Facebook Inc., Terms of use, `http://www.facebook.com/terms.php` (April, 2010).

[32] L. M. Aiello, G. Ruffo, Secure and flexible framework for decentralized social network services, in: Proceedings of SESOC '10: Security and Social Networking Workshop, IEEE Computer Society, 2010.

[33] L. Getoor, C. P. Diehl, Link mining: a survey, ACM SIGKDD Explorations Newsletter 7 (2) (2005) 3–12. doi:http://doi.acm.org/10.1145/1117454.1117456.

[34] T. N. Jagatic, N. A. Johnson, M. Jakobsson, F. Menczer, Social phishing, Communications of the ACM 50 (10) (2007) 94–100. doi:http://doi.acm.org/10.1145/1290958.1290968.

[35] L. Bilge, T. Strufe, D. Balzarotti, E. Kirda, All your contacts are belong to us: automated identity theft attacks on social networks, in: WWW '09: Proceedings of the 18th international conference on World wide web, ACM, New York, NY, USA, 2009, pp. 551–560. doi:http://doi.acm.org/10.1145/1526709.1526784.

[36] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim, A Survey and Comparison of Peer-to-Peer Overlay Network Schemes, IEEE Communications Surveys and Tutorials 7 (2005) 72–93.

[37] L. M. Aiello, M. Milanesio, G. Ruffo, R. Schifanella, Tempering Kademlia with a robust identity based system, in: P2P'08: 8th International Conference on Peer-to-Peer Computing, IEEE Computer Society, 2008, pp. 30–39. doi:http://dx.doi.org/10.1109/P2P.2008.40.

[38] P. Maymounkov, D. Mazières, Kademlia: A peer-to-peer information system based on the XOR metric, in: IPTPS '02: 1st International Workshop on Peer-to-Peer Systems, 2002, pp. 53–65.

[39] E. Sit, R. Morris, Security considerations for peer-to-peer distributed hash tables, in: IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Springer-Verlag, London, UK, 2002, pp. 261–269.

[40] J. Douceur, The sybil attack, in: IPTPS '02: 1st International Workshop on Peer-to-Peer Systems, 2002.

[41] A. Singh, T. W. Ngan, P. Druschel, D. Wallach, Eclipse attacks on overlays: Threats and defenses, in: InfoCom '06: 25th Conference on Computer Communications, IEEE Computer Society, 2006.

[42] F. Lesueur, L. Mè, V. Viet Triem Tong, An Efficient Distributed PKI for Structured P2P Networks, in: P2P'09: 9th International Conference on Peer-to-Peer Computing, IEEE Computer Society, 2009.

[43] C. M. A. Yeung, I. Liccardi, K. Lu, O. Seneviratne, T. Berners-Lee, Decentralization: The Future of Online Social Networking, in: W3C Workshop on the Future of Social Networking, 2009.

[44] M. Mostarda, D. Palmisano, F. Zani, S. Tripodi, Towards an OpenID-based solution to the Social Network Interoperability problem, in: W3C Workshop on the Future of Social Networking, 2009.

[45] K. Fu, S. Kamara, T. Kohno, Key regression: Enabling efficient key distribution for secure distributed storage, in: NDSS'06: Proceedings of Network and Distributed Systems Security Symposium, 2006.

[46] K. E. Fu, R. L. Rivest, Group sharing and random access in cryptographic storage file systems, Tech. rep., Masters thesis, MIT (1999).

[47] M. Backes, C. Cachin, A. Oprea, Lazy revocation in cryptographic file systems, in: SISW'05: Proceedings of the Third IEEE International Security in Storage Workshop, 2005.

[48] A. C. Squicciarini, F. Paci, E. Bertino, A. Trombetta, S. Braghin, Group-based negotiations in p2p systems, IEEE Transactions on Parallel and Distributed Systems 99 (PrePrints). doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2010.25.

[49] M. Bender, S. Michel, S. Parkitny, G. Weikum, A comparative study of pub/sub methods in structured p2p networks, in: Databases, Information Systems, and Peer-to-Peer Computing, Springer, 2006, pp. 385–396.

[50] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, P. Druschel, Scribe: The design of a large-scale event notification infrastructure, in: Networked Group Communication, Springer, 2001, pp. 30–43.

[51] J. Kannan, B. Yang, S. Shenker, P. Sharma, S. Banerjee, S. Basu, S.-J. Lee, SmartSeer: Using a DHT to process continuous queries over peer-to-peer networks, in: INFOCOM '06 : 25th IEEE International Conference on Computer Communications, IEEE Communications Society, 2006.

[52] T. Cholez, I. Chrisment, O. Festor, Efficient DHT attack mitigation through peers' ID distribution, in: HOT-P2P '10 : 7th International Workshop on Hot Topics in Peer-to-Peer Systems, IEEE press, 2010.

[53] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, I. Stoica, Complex queries in dht-based peer-to-peer networks, in: IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Springer-Verlag, London, UK, 2002, pp. 242–259.

[54] P. Heymann, A. Paepcke, H. Garcia-Molina, Tagging human knowledge, in: WSDM '10: Proceedings of the third ACM international conference on Web search and data mining, ACM, New York, NY, USA, 2010, pp. 51–60. doi:http://doi.acm.org/10.1145/1718487.1718495.

[55] Google Wonder Wheel explained, `http://www.googlewonderwheel.com` (2009).

[56] Yahoo! Tag Explorer, `http://tagexplorer.sandbox.yahoo.com` (2008).

[57] L. M. Aiello, M. Milanesio, G. Ruffo, R. Schifanella, Tagging with DHARMA, a DHT-based Approach for Resource Mapping through Approximation, in: HOT-P2P '10 : 7th International Workshop on Hot Topics in Peer-to-Peer Systems, IEEE press, 2010.

[58] B. Markines, C. Cattuto, F. Menczer, D. Benz, A. Hotho, G. Stumme, Evaluating similarity measures for emergent semantics of social tagging, in: WWW '09: 18th World Wide Web conference, 2009.

[59] M. Srivatsa, L. Xiong, L. Liu, TrustGuard: countering vulnerabilities in reputation management for decentralized overlay networks, in: WWW '05: 14th international conference on World Wide Web, 2005, pp. 422–431. doi:http://doi.acm.org/10.1145/1060745.1060808.

[60] D. J. Watts, S. H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (6684) (1998) 440–442.
URL `http://dx.doi.org/10.1038/30918`

[61] B. Yu, P. M. Singh, Incentive mechanisms for peer-to-peer systems, in: Second International Workshop on Agents and Peer-to-Peer Computing, 2003, pp. 77–88.

[62] K. G. Anagnostakis, F. C. Harmantzis, S. Ioannidis, M. Zghaibeh, On the impact of practical p2p incentive mechanisms on user behavior, Tech. rep., NET Institute (2006).

[63] A. Rowstron, P. Druschel, Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility, ACM SIGOPS Operating Systems Review 35 (5) (2001) 188–201. doi:http://doi.acm.org/10.1145/502059.502053.

[64] D. Mazieres, Don't trust your file server, in: HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, IEEE Computer Society, Washington, DC, USA, 2001, p. 113.

[65] E.-J. Goh, H. Shacham, N. Modadugu, D. Boneh, Sirius: Securing remote untrusted storage, in: NDSS'03: Internet Society (ISOC) Network and Distributed Systems Security Symposium, 2003, pp. 131–145.

[66] L. von Ahn, B. Maurer, C. Mcmillen, D. Abraham, M. Blum, reCAPTCHA: Human-Based Character Recognition via Web Security Measures, Science 321 (5895) (2008) 1465–1468.

[67] D. Romero, J. Kleinberg, The directed closure process in hybrid social-information networks, with an analysis of link formation on twitter, in: AIII '10 : Proceedings of 4th International AAAI Conference on Weblogs and Social Media, 2010.

[68] Likir official website, `http://likir.di.unito.it` (2010).