Cachet: A Decentralized Architecture for Privacy Preserving Social Networking with Caching

Shirin Nilizadeh
Indiana University Bloomington
shirnili@indiana.edu

Sonia Jahid
University of Illinois at Urbana-Champaign
sjahid2@illinois.edu

Prateek Mittal
University of California, Berkeley
pmittal@eecs.berkeley.edu

Nikita Borisov
University of Illinois at Urbana-Champaign
nikita@illinois.edu

Apu Kapadia
Indiana University Bloomington
kapadia@indiana.edu

ABSTRACT

Online social networks (OSNs) such as Facebook and Google+ have transformed the way our society communicates. However, this success has come at the cost of user privacy; in today's OSNs, users are not in control of their own data, and depend on OSN operators to enforce access control policies. A multitude of privacy breaches has spurred research into privacy-preserving alternatives for social networking, exploring a number of techniques for storing, disseminating, and controlling access to data in a decentralized fashion. In this paper, we argue that a combination of techniques is necessary to efficiently support the complex functionality requirements of OSNs.

We propose Cachet, an architecture that provides strong security and privacy guarantees while preserving the main functionality of online social networks. In particular, Cachet protects the confidentiality, integrity and availability of user content, as well as the privacy of user relationships. Cachet uses a distributed pool of nodes to store user data and ensure availability. Storage nodes in Cachet are untrusted; we leverage cryptographic techniques such as attribute-based encryption to protect the confidentiality of data. For efficient dissemination and retrieval of data, Cachet uses a hybrid structured-unstructured overlay paradigm in which a conventional distributed hash table is augmented with social links between users. Social contacts in our system act as caches to store recent updates in the social network, and help reduce the cryptographic as well as the communication overhead in the network.

We built a prototype implementation of Cachet in the FreePastry simulator. To demonstrate the functionality of existing OSNs we implemented the 'newsfeed' application. Our evaluation demonstrates that (a) decentralized architectures for privacy preserving social networking are feasible, and (b) use of social contacts for object caching results in significant performance improvements.

CoNEXT'12, December 10–13, 2012, Nice, France.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—Distributed Applications; K.6.m [**Management of Computing and Information Systems**]: Miscellaneous—Security

General Terms

Algorithms, Security

Keywords

privacy, peer-to-peer systems, social networking, caching

1. INTRODUCTION

In the last decade, online social networks (OSNs) such as Facebook, Google+, and Twitter have revolutionized the way our society communicates and have become the de facto mechanism for information sharing between users. Their user bases exceed hundreds of millions of users and their adoption is still growing at a rapid pace.[1]

However, the success of OSNs has come at the cost of user privacy. Users are not in control of their data and depend on the OSN operator to protect their sensitive information. Users' expectations of privacy are often at odds with the operator's business incentives, and, in fact, several providers have been caught selling user data [53]. Moreover, the privacy policies of OSNs are often hard to understand, and constant changes therein further magnify this problem [48]. Additionally, existing centralized architectures present a single point of failure in the system. Any vulnerability in these systems (or even accidental leaks) can be exploited by a malicious adversary to obtain unencrypted sensitive user data.

This lack of user privacy in deployed OSNs has spurred research into the design of mechanisms for privacy-preserving social networking. Some work has focused on using cryptography to protect the contents stored by a centralized OSN provider [7, 17, 30]; our view, however, is that this does not sufficiently protect users' privacy as it allows the provider to learn user relationships and patterns of interactions by

[1] `http://newsroom.fb.com/`

means of traffic analysis. Decentralized architectures can address this issue, yet present a new series of challenges, as in addition to confidentiality and integrity protection that cryptography can provide, it is necessary to ensure the availability of and efficient access to data that is necessary to support common OSN functionality, such as a 'newsfeed'.

Previous work on decentralized OSNs [3, 7, 13, 18, 22, 40] has explored several design decisions: how nodes are organized (in a structured distributed hash table (DHT) or with links between social contacts), where content is stored (by the owner, social contacts, or in a DHT), how it is disseminated (push or pull), and how access control is enforced (cryptographically or with online authentication). We argue that to efficiently support the complex functionality of an OSN, a combination of methods must be used. For example, replicating data at random DHT nodes ensures its availability even when users are offline; however, assembling a newsfeed requires thousands of DHT lookups and, as we saw in our preliminary work [32], can take hundreds of seconds to generate. Likewise, encrypting stored content can provide strong confidentiality guarantees, yet we found that we need online authentication of updates and annotations to ensure the availability of data. Furthermore, attribute-based encryption schemes that provide highly flexible policies are computationally expensive and contribute significantly to the above performance overhead.

In our preliminary design [32], we showed how the confidentiality and integrity of data can be protected by a cryptographic mechanism so that they can be stored in untrusted nodes of a DHT. However, the design suffers from performance issues that arise due to the fetching and decryption of hundreds of small objects belonging to friends, which is required for viewing their walls or for viewing one's own newsfeed. We therefore propose Cachet, a decentralized architecture for social networks that provides strong security and privacy guarantees while efficiently supporting the central functionality of OSNs. Central to Cachet is a hybrid structured-unstructured overlay in which a conventional distributed hash table is augmented with social links between users. We use the distributed hash table as a base storage layer, but add a gossip-based social caching algorithm that dramatically increases performance. New updates are immediately propagated to online social contacts. When an offline user comes back online a presence protocol is used to locate online contacts and query them directly for updates. Additionally, these contacts are used to retrieve cached updates from mutual contacts who are offline as well as speed the discovery of other online contacts. The DHT is then used to retrieve updates that may not be cached, ensuring high availability of data. As mentioned earlier, while several works have been proposed for privacy-enhanced OSNs, Cachet is the first that offers a comprehensive design for OSNs that combines decentralization, attribute-based encryption, and the use of caches to provide high availability, low latencies, and flexible policies for protecting data.

Data in Cachet are stored in container objects that include content, such as status updates and photos, as well as references to other containers; authorized contacts can add comments or other annotations to containers. A container is protected by a cryptographic structure that ensures confidentiality and integrity while supporting multi-principal interactions without revealing policies or user relationships to the storage nodes. The structure includes two components: cryptographic capabilities used by the storage nodes to authenticate update requests and attribute-based encryption used to provide flexible and fine-grained access policies. To reduce computational overhead cached containers are stored in decrypted form and are shared with other contacts upon verifying that they satisfy the corresponding access policy; as such, containers must be decrypted only when fetched directly from the DHT. Storage nodes are trusted only to provide availability of the data with replication used to defend against node failures or intentional misbehavior.

We develop a prototype implementation of Cachet in the FreePastry simulator [51]. To demonstrate the functionality of existing OSNs, we also build and evaluate the newsfeed application. Our results show the importance of using social caching, which reduces the latency of displaying a newsfeed from hundreds of seconds in the base architecture to less than 10. Our architecture thus demonstrates how a careful combination of several distributed systems and cryptographic techniques can be used to provide a compelling privacy-preserving alternative to centralized OSNs.

2. REQUIREMENTS AND PROPERTIES

Functional Model. At a high level OSN functionality consists primarily of users sharing some form of content with their contacts, who then view, comment on, and annotate it. To support this generic functionality, we define a *container object* that consists of a main content object and a list of annotations. The main content can take many types, such as a status, a web URL, a photo, or a collection of container objects (e.g., a photo album, or a 'wall'). Annotations take the form of references to other container objects. A key application common in OSNs is a 'newsfeed', which aggregates and displays recent updates from a user's social contacts; implementing such a newsfeed efficiently is a key challenge in a decentralized OSN.

The container also becomes the unit of access control, with a potentially different set of permissions associated with a container, internally referenced objects, and each individual annotation. Access policies can be defined over social contacts and their attributes, as well as social network distance (e.g., 'friend-of-friend'). Usable access controls in OSNs remain an area of active research [19, 26, 38], so a highly flexible and fine-grained permissions architecture is necessary to support future developments in this field.

Security Requirements. The primary security requirements are confidentiality and integrity of user data, stored in distributed and untrusted storage nodes, and availability of the correct and latest version of the data. Users should be able to have complete control over the permissions to content they create and no user should be able to access content unless explicitly authorized by the owner. Finally, user relationships should remain hidden from third parties, such as the storage nodes.

Threat Model. We assume that the participants in the decentralized OSN may be malicious (or compromised), Byzantine, and capable of launching both active and passive attacks. Distributed systems are vulnerable to the problem of Sybil attacks [24]. However, existing mechanisms are available to defend against them [15, 37]. We consider that up to 25% of the nodes in the system can be malicious, since, beyond that, existing mechanisms [15] are not able to securely route in distributed hash tables, which is a necessary prerequisite to provide both integrity and availability

guarantees. We also assume the existence of mechanisms to defend against denial-of-service (DoS) attacks [25, 49].

3. BASE ARCHITECTURE

Our efficient newsfeed algorithm builds on our preliminary work, which describes a basic storage architecture for decentralized social networks [32].

In our base architecture, privacy is provided through a combination of design features including a DHT for decentralization, cryptography to enforce attribute-based policies, and data representation in terms of objects. Users can define relationships of various types asymmetrically. The basic prototype supports user profile and wall features including status updates, wall posts from social contacts, commenting on posts, and a basic newsfeed algorithm. Existing OSNs such as Facebook, Google+, and Twitter feature such functionality as a major use case.

3.1 Policies

Policies are described through user identities or attributes, as required. Identity-based policies define user-specific access, whereas attribute-based (AB) policies define access for a group of social contacts sharing some common features. AB policies represent formulas over attributes, using operators such as $\wedge$, $\vee$, and k-of-n. Examples of AB policy are: $(friend \wedge coworker) \vee family$, and 2 *of* $\{friend, family, coworker\}$.

Each object is protected with three policies: 1) *Read Policy*, defined through user attributes, describes who can view an object; 2) *Write Policy*, generally set to the object owner's identity, describes who can delete or overwrite the object; and 3) *Append Policy*, an attribute-based policy, defines who can append to an object—in other words, who can comment on an object.

These policies are defined by the owner at the time of object creation and stored in the object metadata. The Read Policy is enforced through the use of cryptography. The Write and Append Policies are enforced through a combination of cryptography and authorization by DHT nodes. The authorization does not reveal a user's identity, hence the storage node is not aware of the identities of users storing or retrieving data from it, and therefore a user's social graph is hidden from the storage nodes. DHT nodes also implement a special append operation that adds a new annotation to the object while leaving existing content unmodified.

3.2 Cryptography

Access policies are enforced cryptographically through a hybrid scheme of traditional public key and attribute-based encryption (ABE) [11, 31]. In ABE an object is encrypted with an AB policy; for example, $P = friend \wedge family$. Each of the intended parties is issued a unique secret key by a key authority defining what attributes apply to that person. For example, a person *Alice* may receive a key with the attributes *"friend", "colleague"*. A person can decrypt an object if and only if her secret key satisfies the policy used to encrypt it (the object). In the hybrid mode, the message is encrypted with a randomly chosen symmetric encryption key, which is in turn encrypted with ABE. In the previous example, Alice cannot decrypt a ciphertext that was encrypted with P since her key does not satisfy the policy P.

We place the Read Policy in the object reference rather than the object itself to protect policy privacy from the storage nodes. The main motivation for this choice is that the version of ABE we are using lacks policy privacy and this approach keeps the policy hidden from untrusted storage nodes. The reference is a part of the parent object and is encrypted with its symmetric key. As a result, the reference cannot reveal the policy. Therefore, confidentiality is ensured through a hybrid approach where the object is encrypted with a symmetric key, and the symmetric key, placed as a part of the reference, is encrypted with ABE. The ABE scheme that we use is an extended version of EASiER, which supports efficient revocation for Ciphertext Policy Attribute-based Encryption [11] with the help of a minimally trusted proxy. Please refer to this extended scheme [31] for further details on the cryptographic scheme.

Integrity of objects is guaranteed through digital signatures. Object owners sign the content of the object. The Write and Append policies are enforced by controlling access to the corresponding signature keys. The Write Policy key is generally encrypted to the object owner, and the Append Policy key is encrypted with an attribute-based policy. The public part of the Write Policy key is also made available as a part of the object reference to ensure its authenticity and prevent vandalism from the storage node. Comment references are signed by commenters using Append Policy keys, thus ensuring the integrity of the comment. Note that when someone comments on an object, status, for example, both the commenter and the status owner's policies are enforced since each of the objects is encrypted using the policies defined by its owner.

An object reference is constructed as follows:

$$objRef \stackrel{\text{def}}{=} (objID, \text{ABE}(K, P), WPK)$$

where *objID* is a random object identifier used to locate it in the DHT, K is the symmetric key used to encrypt the referenced object, P is the read policy, and WPK is the Write-Policy signature public key. $ABE(K, P)$ represents ABEncryption of K with the policy P.

3.3 Data Storage

In the base architecture, users form a DHT, such as Pastry [51] or Kademlia [41]. Data is stored as an object in the DHT using *objID* as the DHT key. In addition to the standard *get* and *put* operations, the DHT also supports an *append* operation. Additionally, the storage nodes verify the Write Policy on objects.

Several security and privacy issues are taken care of through existing mechanisms: lookups can be secured against attacks [5, 15, 33, 46]; availability is ensured through replication; and malicious data overwrites are prevented through write-policy verification. The write policy prevents malicious users from creating modifications that will be accepted by the readers, as they cannot produce a correct signature, but they may overwrite and destroy legitimate data. To address this, the write-policy public key is stored unencrypted as part of the object metadata. The storage node will refuse to overwrite the stored object unless the new data is properly signed by the write-policy key; deletions must likewise be authenticated with a signature. The write policy public key is random and unique for each object to prevent linking an object to its owner.

3.4 Newsfeed

A user's newsfeed is a collection of the latest *status update objects* from each of her social contacts. To provide users with their newsfeed, we designed a basic newsfeed algorithm in our base architecture. However, the algorithm in the base architecture is inefficient — the latest status update objects are fetched sequentially, and each of them is decrypted individually, which makes viewing the newsfeed non-practical. This means that if a user has hundreds of social contacts, then she has to wait until all of her contacts' latest status update objects are fetched and decrypted. In addition, users have to decrypt the wall objects of all of their social contacts and decrypt the most recent reference on the wall. In Cachet though, the update object contains a link to the most recent update and the user does not need to decrypt the potentially large wall object. Additionally, since the ABE format contains the policy necessary for decryption, users can infer whether they will be able to decrypt the object or not and do not need to spend time decrypting the object if they are not authorized to read it.

Decryption is time-consuming. Besides, we do not perform any type of caching or utilize social links to expedite the loading of a user's newsfeed in the base architecture. Furthermore, in practice a user may not be interested in viewing all of her contacts' latest statuses, but subscribe to a few selected ones instead. In Section 4 we will present our enhanced design in which social contacts are employed to cache update objects and a social caching algorithm is used to provide faster access to the objects.

3.5 Example

User Alice joins Cachet by generating several keys, creating profile information and a wall, and saving this information as root and wall objects (respectively) in the DHT. To establish the relationship *friend, co-worker* with Bob, Alice generates an ABE secret key for Bob with the attributes *friend, co-worker*. Bob may establish a different type of relationship with Alice. The necessary keys are exchanged out-of-band.

Figure 1 shows an example object structure. To post a status update to her wall, Alice creates a status object, complete with version number, contents, and public and secret keys for the Write and Append policies ($WPK_1, WSK_1, APK_1, ASK_1$). She generates a signature over the Write-policy signature key (WSK_1). She then picks a random symmetric encryption key K_1 and encrypts the object (except for WPK_1 and APK_1 and the signature). She also chooses a random ID ID_1 and uses this to insert the object into the DHT. Finally, she creates a reference to the status update, including ID_1, K_1, and her Write-Policy public key (WPK_1) and adds it to her wall.

When Bob wants to read Alice's update, he finds the reference on Alice's wall and decrypts K_1 with his attribute-based secret key that he got from Alice. He then retrieves the object from the DHT with the key ID_1 and decrypts the encrypted fields using K_1. Finally, he verifies the signature to ensure the integrity of the object. To comment, Bob first creates a comment object following a process similar to Alice's creation of her update. He then uses the *append* operation to insert a reference to the new object into Alice's update. Assuming he satisfies the A-policy AP_1, Bob decrypts ASK_1 and uses it to generate a signature on the reference.

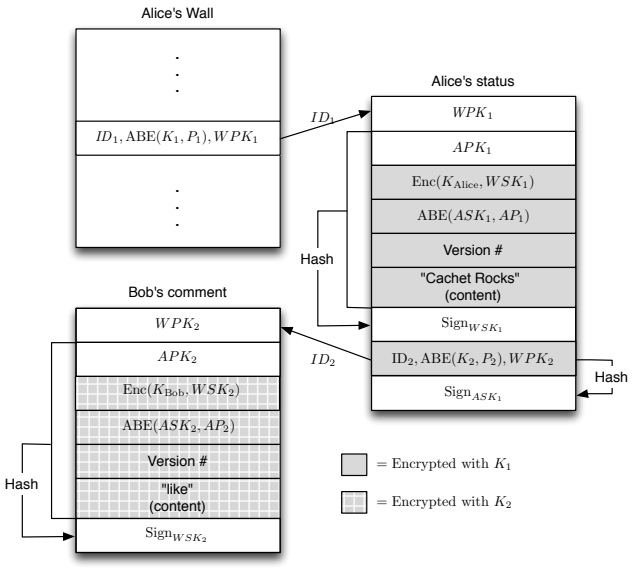


Figure 1: Example objects

Please refer to our prior workshop paper for further details [32].

4. SOCIAL CACHING

Given the use of decentralization and cryptography in our base architecture, downloading and reconstructing a social contact's wall or an aggregated newsfeed is a lengthy process requiring the following steps: 1) decrypting update objects, which are ABEncrypted, to yield metadata such as an update's DHT key and symmetric decryption key; 2) accessing multiple small objects located in different storage nodes using DHT keys provided in the previous step; and 3) decrypting the retrieved update objects with their corresponding symmetric keys. Our preliminary analysis [32] indicates that these operations can take hundreds of seconds, and thus the design needs to be improved for practical deployments.

We propose a gossip-based social caching algorithm that, in combination with an underlying DHT, leverages social trust relationships for dramatically increased performance and reliability. Nodes maintain continuous secure (SSL) connections with online contacts to receive updates directly as soon as they are produced. We describe a presence protocol, which itself uses social caching for finding online contacts. Since ABDecryption of objects is a time-consuming bottleneck, online social contacts who satisfy the ABE policy are leveraged to provide cached, decrypted objects to other contacts who also satisfy the policy for objects related to offline contacts. We emphasize that a data object is not cached by a social contact unless he/she satisfies the ABE policy. Thus, the original ABE policies provided by our base architecture are also preserved by Cachet. The basic object structure in Cachet is extended to include a list of users' IDs that are authorized to decrypt and read the object. Therefore, an object can be forwarded to/cached by the intersection set of one's social contacts and the users in the attached list, and the ABE policies are honored as before.

Our algorithm also seeks to minimize the number of such decryptions (corresponding to DHT lookups for the objects) by dynamically learning which peers yield the most cached

objects. Yet, this approach provides reliability by treating the DHT as a persistent store for objects that may not be cached. Moreover, social caching improves data locality because the social contacts of a user are usually geographically co-located, which minimizes the needed bandwidth for downloading an object.

Gossip-based protocols are reliable and robust tools for data dissemination especially when used in P2P and wireless networks [9, 10, 20, 29, 42]. However, relying purely on gossip protocols for disseminating updates through the social network has some drawbacks: 1) redundant information is passed around and stored in the network, even at nodes that do not desire this information; and 2) social circles have correlated patterns of online presence, making it challenging to ensure availability when large parts of a circle are offline.

4.1 Presence Protocol

Usually a centralized server keeps track of users' presence information (e.g., their current IP address) in P2P networks [3, 52]. In Cachet a distributed approach is applied where every peer stores a presence object in the DHT so that social contacts can obtain a peer's presence information at any time. The presence object has the same structure as other objects, and is ABEncrypted so that the storage node cannot learn the contents of the presence object.[2] It contains the peer's current IP address, and port. With this IP address, peers can connect to their social contacts directly and maintain live connections. The object is signed so that the storage node only allows the owner to update or rewrite it. Whenever a peer joins or leaves the Cachet network, it updates its presence information.

Note that retrieving and decrypting presence objects for all of one's social contacts will have overhead similar to constructing a newsfeed directly from the DHT. To speed up this process presence objects are cached using gossip-based social caching along with content updates. As such, once a few social contacts have been located, discovery of other contacts can proceed at an accelerated pace. Once links to online contacts have been established, subsequent updates are pushed to online contacts directly using the caching protocol described next.

4.2 Gossip-based Social Caching

When a node comes online and joins the Cachet network, it does not have the presence or newsfeed information for any of its social contacts. We now describe the caching algorithm used to progressively retrieve cached, unencrypted versions of these objects to greatly speed up the process of loading the newsfeed. The basic idea of the caching algorithm is for the user to perform a few DHT lookups to get presence objects of some social contacts. Then, she identifies those who are online and contacts them to 1) inform them that she is online, and 2) pull both the cached presence objects and the cached recent updates of their mutual social contacts. The user then uses the new unencrypted presence objects to recursively repeat the two steps above until no new contacts are obtained. At this point another DHT lookup is made for a social contact whose status is unknown and the process is repeated until the presence and status objects of all contacts have been obtained. If a user is contacted by a social contact Q who was offline before, she pushes her all cached objects that Q is authorized to read.This algorithm is thus a pull-push based gossiping algorithm because it involves both operations; a joining node pulls information from online nodes, and a node generating an update pushes them to other online nodes.

Algorithms 1 and 2 are employed by a user P when she joins the social network and the algorithm includes the following steps:

1. *Creating the Presence Table*: User P maintains a presence table that lists all social contacts along with their presence statuses. The social contacts are listed in descending order based on the number of mutual social contacts in common with P — a social contact who has the most mutual social contacts in common with P appears at top of the list. Listing social contacts in descending order of mutual contacts is a greedy approach that attempts to minimize the number of DHT lookups and communications that are needed to retrieve data objects. Thus, by contacting the social contacts on top of the table, more data objects can be potentially obtained. Initially, the presence status for all contacts is *undefined*.

2. *Selecting a Contact*: P chooses an unvisited contact Q from the presence list as follows. P chooses the first contact in the presence table whose status is known to be online. If none exist, then it chooses the top contact with an undefined status. If all contacts are visited or known to be offline, P proceeds to step 7;

3. *DHT Lookup and Connection*: If the presence status of Q is undefined, then P retrieves Q's presence object from the DHT and decrypts it. If Q is offline, then it returns to step 2 to select another contact;

4. *Pulling Information*: Since Q is online, P marks Q as visited and creates a secure connection to Q. P uses this connection to pull presence and update objects for mutual social contacts that P and Q have in common;

5. *Caching Information*: P caches the pulled objects (in unencrypted form). We assume that the object cache is unlimited and can store all social contacts' updates during a session. As argued by Mega et al. [42] most of the objects such as status, posts, comments, links, and pictures are small enough; large objects such as videos can be retrieved from the DHT or online services (e.g., YouTube) on demand only;

6. *Updating Presence Table*: P updates the presence table with the online status of social contacts based on information learned from Q. Then it returns to Step 2 to locate the next social contact to connect to;

7. *Performing DHT Lookups for Offline social contacts with No Mutual Social Contacts*: If the recent updates of some social contacts are missing, then this shows that they do not have any online mutual social contacts with P, and P must obtain these objects from the DHT. Thus, to retrieve the newsfeed, P needs to 1) derive the key for their updates by ABDecryption of their reference embedded in the parent/containing object; 2) perform DHT lookups for them; and 3) decrypt the updates by their corresponding symmetric key.

[2]It could be a separate object or simply embedded in the profile or root object.

By exchanging presence status and recent updates between online social contacts, the presence table and the cache are always up-to-date. Thus, for viewing the newsfeed, peer P just retrieves recent updates from the cache.

4.3 Identifying Mutual Contacts and Authorized Users

Many of the benefits of social caching come from being able to identify mutual social contacts. Although relationships between users are privacy sensitive, in practice many users are comfortable sharing this information with at least their immediate social circle. For further privacy protection it is possible to use a social contact discovery protocol that reveals only mutual social contact relationships and nothing else [21].

Since cached content is stored unencrypted, it is also important to verify that a contact satisfies the access policy associated with the object. It should be possible to extend the private contact discovery protocol to learn the attributes shared by P and Q and thus make an authorization decision based on that.[3] For simplicity, however, in our current implementation we instead include an explicit list of authorized users in each container that can be used to mediate sharing.

We note that users who wish to conceal their social relationships, or reveal only a selected subset, may do so, trading off privacy for the efficiency of social caching.

Algorithm 1: User P joins the network

```
1
2  //User P joins the network
3  generatePresenceTable(table);
4  socialCachingAlg(table, cache);
5  for(social contact Q : table.keySet()){
6    if(!cache.contains(Q.update)){
7      getDHTKeyFor(Q.update);
8      encUpdate = dhtLookUp(Q, Q.updateObj);
9      update = decrypt(encUpdate);
10     cache.put(Q, update);
11   }
12 }
```

Algorithm 2: Social caching algorithm

```
1
2  void socialCachingAlg(presenceTable table,
3   Cache cache){
4    for(SocialContact Q : table.keySet()){
5      Q.visited = TRUE;
6      dhtLookUp(Q, Q.presenceObj);
7      if(Q.presence.status){
8        sendTo(Q, Q.presenceObj);
9        receiveMessageFrom(Q, bufr);
10       if(bufr.contains(presenceObj))
11           updateTable(table, bufr);
12       if(bufr.contains(UpdateObj))
13           selectUpdatesToKeep(cache, bufr);
14     }
15     SocialContact R = selectSocialContact(&table);
16     socialCachingAlg(R, table);
17 }
```

4.4 Deletion and Revocation

When a user deletes an object or modifies the access policy to an object (including changes to a social contact's attributes) these changes are reflected immediately for data that are ABEncrypted and fetched from the DHT. Affected data in the caches must be updated or invalidated. While we do not specify the protocol here, in short: users can send *object invalidation requests* to remove deleted objects from caches, and *revocation requests* to update the access policies for cached objects, i.e., the list of names to be removed from the access lists for various objects. We leave the evaluation of such deletion and revocation to future work.

5. EVALUATION

In this section, we evaluate the performance of our presence and social-caching algorithms. We do not compare Cachet's performance with other caching mechanisms [12, 36, 42, 56, 59] since they are not specifically designed for providing security and privacy as in our setting.

5.1 Implementation and Simulation Setup

We built a simulator for Cachet based on the FreePastry simulator [51], which implements the underlying DHT. We simulate the cryptographic operations for EASiER [31] with 1 attribute policy and 100 revocations run on a standard machine with 2.40GHz Intel Core 2 Duo, 4GB memory, and running Ubuntu 10.04. With this setting, the ABDecryption takes 422ms. The symmetric key decryption (openssl aes-128-enc) takes 0.04ms on a file of size 2500 bytes, the average size of a status update object. We simulated the communication overhead between peers by setting the average communication latency to be 180 ms.[4] To simulate the social graph in Cachet, we used the Facebook friendship graph from the New Orleans regional network [54]. This data set contains a list of all of the user-to-user links from the Facebook New Orleans network and consists of 63,732 nodes and 1.54 million edges. This data set has been used for simulating social graphs in other published work [43, 47, 55].

We evaluated the performance of Cachet by averaging results over the following unit experiment: we used FreePastry to setup a DHT amongst all nodes in the social network, except a particular random user P. We then generate updates for the entire social circle of node P, and simulate Cachet's algorithms. Although our system could be used to cache comments on objects as well, for evaluation we considered a model where newsfeeds include status updates only; we assume a usage model where users click on a particular item to fetch specific comments for that item.

Next we introduce churn in the network, and consider different percentages of nodes amongst P's social contacts and FoFs that remain online — 10%, 30% and 50%. We focus on an online/offline model where different percentages of online friends will affect the caching performance. We note that we are not attempting to evaluate the impact of churn at the DHT layer. In this work we are assessing the effect of nodes joining/leaving and impacting the performance at the caching layer based on how many social, trusted contacts are available. Due to the lack of pertinent data about online/offline patterns in OSNs, we picked various percentages. The 10–30% range is perhaps more pertinent because, for example, Skype has about 45M concurrent users online and 200M active users per month.[5]

[3] Briefly, instead of a contact certificate as in [21] one would use a (contact,attribute) certificate for each attribute.

[4] `http://pdos.csail.mit.edu/p2psim/kingdata/`

[5] Skype Reaches A 45M Concurrent User Peak, And What

We then simulate the node join process for node P, and measure the performance of the newsfeed application.

Performance metrics. We measure performance using the following metrics.

- *Hit Rate*: the percentage of the newsfeed or the presence objects that has been provided by social contacts. To measure worst-case performance, we assume that the number of updates on a user's newsfeed is equal to the number of her social contacts.

  Let e_m be a single unit experiment for a user u with m social contacts. Let d be the number of DHT lookups (involving ABDecryptions) that have been performed for obtaining either u's newsfeed or u's social contacts' presence objects, then:

  $$hitRate(e_m) = \frac{m-d}{m}$$

  This metric measures what fraction of objects were found in the cache.

- *Progressive Hit Rate*: the percentage of the newsfeed or the presence status objects that have been obtained after d DHT lookups and pulling social contacts' cached objects. Let e_m be a single unit experiment for a user, u, who should retrieved presence information of m social contacts and let $\sigma(d)$ be the number of obtained social contacts' presence or status objects after d DHT lookups, then:

  $$hitRate(e_m) = \frac{\sigma(d)}{m}$$

  This metric gives an indication of what percentage of the total objects have been obtained after some number of lookups.

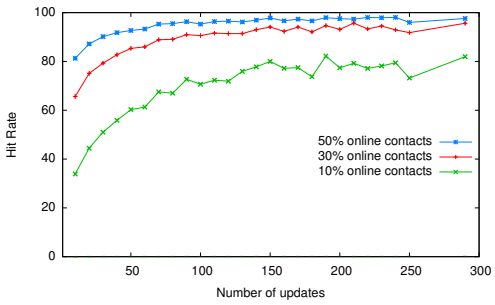
5.2 Results

- *Social caching provides most of the update objects for viewing the newsfeed*

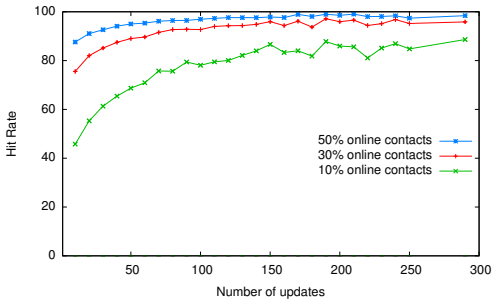
  Figure 2(a) depicts the average *Newsfeed Hit Rate* as a function of number of updates (equal to the number of social contacts of a person) and the fraction of online social contacts. As it is expected, the Newsfeed Hit Rate increases with a larger percentage of online social contacts. However, interestingly, retrieving a larger newsfeed where each of its objects corresponds to a social contact does not decrease the Newsfeed Hit Rate, because more online social contacts are available to push the cached data to the user.

  We repeated a slightly different experiment where online FoFs are also be contacted by a user to get cached objects belong to their mutual offline social contacts by adding them to the presence table. Although Figure 2(b) shows that leveraging FoFs increases the hit rate slightly, the difference is not very high. Thus, we decided to not include FoFs in our algorithm given the additional overhead of caching FoF objects.

  These figures are dependent on the social network graph, and they show what fraction of updates can be provided by social contacts using the social caching algorithm. The results indicate that by using social caching one can rely on her social network to provide most of her newsfeed objects. However, social caching alone is insufficient to ensure availability of the complete newsfeed, necessitating the DHT storage layer.



(a) Only social contacts are contacted



(b) Social Contacts and if needed FoFs are contacted

Figure 2: These figures depict the average *Newsfeed Hit Rate* as a function of the number of updates and the fraction of online social contacts. It can be seen that social caching provides most of the update objects needed for viewing the newsfeed. By comparing the two figures it can be seen that leveraging FoFs increases the hit rate slightly, but the difference is not great.

- *Social caching decreases the latency for retrieving the newsfeed*

  The results illustrated in Figure 2 imply that with social caching, the latency for viewing the newsfeed would be much lower than loading all objects from the DHT and decrypting them. To investigate, we examined the latency for retrieving the newsfeed in Cachet both with and without social caching enabled.

  To calculate the latency, in each single experiment, we considered the simulation time for 1) the communication latency between peers, 2) ABDecrypting the references to both presence and update objects that are not provided by social contacts, 3) performing DHT lookups for retrieving these objects, and 4) decrypting the objects.

  Figure 3 shows that using just the base architecture, the latency for obtaining newsfeed is very high and is also highly dependent on the number of updates; it is also dominated by the ABDecryption latency time. In contrast, applying social caching, the latency decreases by up to an order of magnitude. The time needed to view a newsfeed decreases as the number of online social contacts grows, but even with only 10% of social contacts online,

Looks Like A New Stage Of Momentum, TechCrunch, Oct. 14, 2012.

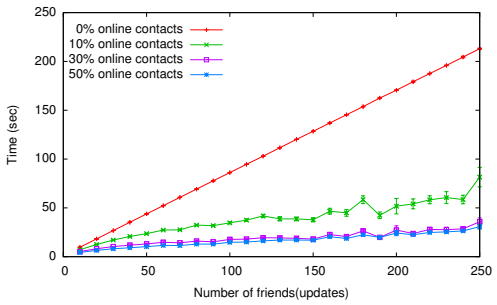


Figure 3: This Figure shows the latency for retrieving the newsfeed in Cachet both with and without social caching enabled. It can be seen that even with only 10% of social contacts online, social caching provides a dramatic performance improvement.

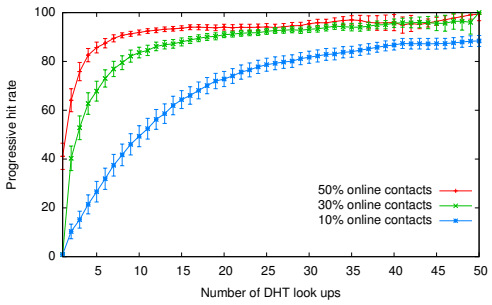


Figure 4: This Figure shows the *Average Progressive Hit Rate* for users who have 100 to 200 social contacts. It can be seen that after a few DHT lookups, users learn about online/offline status of most of thier social contacts. Similarly, most of newsfeed updates can be retrieved after performing a few DHT lookups and contacting the identified online social contacts.

social caching provides a dramatic performance improvement.

- *Most of the presence objects would be available after a few DHT lookups and decryptions*

  We measured the number of presence (or newsfeed update) objects that is provided by online social contacts after contacting them. Figure 4 plots the *Average Progressive Hit Rate* after d DHT lookups and ABDecryptions. For this experiment, we plotted the *Average Progressive Hit Rate* for users with $100 \leq m \leq 200$ where m is one's number of social contacts. As it can be seen, having 50%, 30%, and 10% online social contacts, one can receive about 70% of all presence objects by performing about 2, 5, and 18 DHT lookups (and ABDecryptions) respectively.

  This figure shows that users do not need to wait until all presence (or update) objects are retrieved. Furthermore, they can identify most online social contacts (and their updates) by looking up only a few presence (or update) objects and contacting the social contacts who are identified to be online. Thus, user perceived latency can be reduced by rendering the newsfeed when a threshold number of objects are retrieved, and then the feed is updated as more objects are fetched. For example, with 30% online social contacts, 70% of the newsfeed can be loaded after only a tenth of the total lookups, corresponding to load times of the majority of the newsfeed in under a second.

6. RELATED WORK

6.1 Access Control in Decentralized OSNs

Researchers have designed and proposed several decentralized OSNs such as Diaspora [22], PeerSon [13], Safebook [18], LotusNet [3], SCOPE [40], and Persona [7]. These works do not focus on caching or leveraging social links for fast and efficient data retrieval, but address privacy either through cryptography, architectural modifications, or decentralization of the provider. We have shown that in the absence of social caching, the performance overhead due to cryptography and decentralization is high.

Diaspora is a social network that users install on their own personal web servers without support for encryption. We note that Diaspora is a deployed system with several hundreds of thousands of users [58] and demonstrates the feasibility of large scale decentralized approaches to social networking. Backes et al. [6] present a core API for social networking, which can also constitute a plug-in for distributed OSNs. Their primary focus is on an API that supports anonymous data access in a distributed OSN. However, they assume that the server is trusted with the data for access control. PeerSon, LotusNet, Safebook, and SCOPE benefit from DHTs in their architecture. PeerSon and Safebook suggest access control through encryption, but they fall short in providing fine-grained policies compared to ABE-based access control in Cachet. Moreover, in all of these schemes, the overhead of key revocation affects performance whereas revocation in Cachet is efficient through the use of a semi-trusted proxy (please refer to the base architecture [32]).

In LotusNet, which is based on Likir [4], the authors consider the distributed storage to be trusted and do not perform encryption. Likir uses signed grants to specify permissions and provide access control. Safebook is based on a peer-to-peer overlay network named "Matryoshka". The end-to-end privacy in Matryoshka is provided by leveraging existing hop-by-hop trust of the links. In contrast to using hop-by-hop trust for data lookup and privacy, we leverage trust relationships to improve performance and ensure privacy using cryptographic techniques.

SCOPE is a distributed data management system for specialized P2P social networks. Clients connect to and store data on a group of super-nodes with higher computation and storage capacity. However, clients do not participate in the DHT; only the super-nodes run the DHT code. Clients connect to super-nodes and rely on them for sharing and access control on their data.

Persona [7] combines ABE with a decentralized OSN architecture to ensure data confidentiality. However, Persona does not support fine-grained policies and lacks suitable revocation mechanisms [11]. Persona is not built upon a DHT; users and applications use a storage service hosted on a dedicated storage server or a user's own storage server. The storage service authenticates write operations through the requester's public key and hence can learn the user's social contacts.

We note that some techniques leverage a centralized

provider for maintaining the overall functionality of an OSN but encrypt messages to keep them confidential from the provider [8, 27, 39]. These approaches, however, allow the OSN provider to monitor the interactions of users, censor or remove content by users, and even regulate who can be part of the network. Cachet and the other decentralized solutions attempt to democratize such systems by getting rid of such a powerful centralized provider.

6.2 Information Dissemination in OSNs

One approach for disseminating information in a network is based on gossiping techniques. Mostly, this approach has been applied for disseminating information through wireless networks [9, 28, 29, 44] and P2P networks [34, 35, 60]. However, very few have done research on dissemination of information through decentralized social networks.

Datta and Sharma propose GoDisco [20], a gossip-based decentralized mechanism in which information can be disseminated by using social links and exploiting semantic context. This mechanism is targeted at probabilistic publish/-subscribe systems where a vector of interest categories is attached to each message, and information is broadcast to receivers who may be interested in the message. This mechanism is thus orthogonal to our work — while GoDisco disseminates *public* information to *interested* parties, Cachet focuses on disseminating *private* information to *authorized* parties.

Abbas et al. [1] propose a basic gossip-based protocol for establishing friendship links in a distributed social network considering network dynamics. However, many requirements of social networks, such as dissemination of updates, availability of data, and privacy were not in the scope of their work.

Mega et al. [42] show that applying gossiping algorithms for disseminating users' updates through a P2P social network is feasible. They focus their analysis on the coverage of disseminated updates to the social network, average latency for an update to reach a destination, and the average load in terms of messages sent and received. They do not consider privacy or access control in their design and updates are pushed to all friends and FoFs. In addition, their system relies purely on gossiping protocols for disseminating updates through the social network, which has several drawbacks; for example, there is no guarantee that all updates will be available over time, and a large amount of redundant information is passed around and stored in the network. In Cachet though, updates are stored in the DHT so available over time and friends cache updates for a short time.

Carrasco et al. [14] address the problem of loading newsfeeds efficiently in *centralized* social networks, where data is stored in distributed databases (e.g., at a data center). They propose partitioning the social network based on users' activities over time so that data belonging to users in a partition can be stored in a way that improves locality. The proposed solution implicitly assumes that a centralized management system keeps track of all users over time to facilitate on data partitioning, but such information is not available in decentralized social networks. Nevertheless, distributed algorithms to improve locality of information in the context of decentralized social networks could improve newsfeed performance, and we leave such an exploration to future work.

7. DISCUSSION

Searching social contacts. To enable users to search for their social contacts, we propose leveraging a centralized directory service that maintains the mapping between user names and their root objects (profile page). To prevent the directory service from inferring user relationships, users can either (a) use anonymous communication channels such as Tor [23] to query the server, or (b) leverage private information retrieval protocols [16] to hide the user name mapping that is being retrieved from the server.

Privacy issues. While we believe that Cachet's privacy guarantees surpass existing systems, there is still room for further improvement. First, users that do not satisfy the access control policy of a particular object will be aware that they are being excluded from accessing the object, as opposed to being oblivious to its existence (as in current OSNs). Second, our social caching algorithm leaks information about the identities of users who satisfy a particular policy to all of those identities. Finally, our newsfeed algorithm also reveals information about when a user comes online or offline. However, we emphasize that "online" does not necessarily mean that users are logged in and available. It could be that the person's laptop/desktop is connected to the Internet and participating in the DHT and caching protocol, although the person is not at the computer. In future work we will investigate techniques to limit such sources of information leakage.

Deployment challenges. In contrast to the deployment model of today's popular OSNs, users in Cachet face the burden of (a) spending additional computational resources, and (b) volunteering data storage and bandwidth. However, we note that our social gossiping and caching algorithms make the computational overhead of decrypting ABEncrypted objects practical. Furthermore, online social networks are mostly used to store small objects such as status posts and comments, minimizing the burden for users to volunteer excessive storage and bandwidth.

The decentralized architecture of Cachet brings with it several challenges from a networking viewpoint. First, resilience against node churn becomes an important consideration. The underlying DHT should have enough replication to handle temporary instabilities. We note that data objects are also available through our caching mechanism to alleviate instability issues due to churn. In future work we will study the use of less structured overlay topologies, e.g., the use of more stable 'super peers' [2, 40, 50, 57] to further improve stability. Second, users behind NAT make it difficult to realize peer-to-peer connections with other users. Our architecture requires NAT hole-punching mechanisms, such as that of Evans et al. [45].

Social and economic challenges. Existing OSNs such as Facebook and Google+ already have several hundred million users. Thus a significant challenge for new social network architectures is to be able to attract enough users to achieve a critical mass. We believe that enhanced privacy properties of decentralized architectures such as Cachet would give an incentive to users to switch. Furthermore, government regulations or standards encouraging inter-operable OSN architectures can also help to offset the economic challenges for deployment.

Scalability issues. DHTs are designed to be scalable, but as the network becomes very large, e.g., with one billion nodes, scalability concerns are valid — nodes will be involved

in more overhead for maintaining the DHT structure, and the amount of cached objects may be larger. We leave such evaluations to future work but comment that our caching algorithm scales with the number of friends, and thus we do not expect the large network size to overly affect the performance of our caching algorithm.

8. CONCLUSION

We have presented Cachet, a decentralized architecture for social networks that provides strong security and privacy guarantees while efficiently supporting the central functionality of OSNs. Cachet uses an object-oriented design for flexible data management, attribute-based cryptography for access control, and a hybrid combination of distributed hash table and social contacts for information retrieval. The use of social contacts is the key to making the architecture practical; social contacts in Cachet not only provide information about their own updates, but also about updates from other mutual contacts. Our experimental evaluation using the FreePastry simulator shows that the average time to reconstruct an aggregate newsfeed is less than 10 seconds, as compared with hundreds of seconds without the use of social caching. Our architecture thus demonstrates that a decentralized approach to privacy-preserving social networking is practical.

9. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Awards CNS–0953655 and CNS–1115693, by the Boeing Trusted Software Center at the University of Illinois and by the National Security Agency. We thank John McCurley for his editorial help, and also anonymous reviewers for their useful comments. We also thank our shepherd Cristina Nita-Rotaru.

10. REFERENCES

[1] S. M. A. Abbas, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips. A gossip-based distributed social networking system. In *Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, WETICE '09, pages 93–98, Washington, DC, USA, 2009.

[2] D. Adami, C. Callegari, S. Giordano, M. Pagano, and T. Pepe. A real-time algorithm for skype traffic detection and classification. In S. Balandin, D. Moltchanov, and Y. Koucheryavy, editors, *Smart Spaces and Next Generation Wired/Wireless Networking*, volume 5764 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009.

[3] L. Aiello and G. Ruffo. LotusNet: tunable privacy for distributed online social network services. *Computer Communications*, 35(1):75–88, 2012.

[4] L. M. Aiello, M. Milanesio, G. Ruffo, and R. Schifanella. Tempering Kademlia with a robust identity based system. In *P2P*, 2008.

[5] M. S. Artigas, P. G. Lopez, J. P. Ahullo, and A. F. G. Skarmeta. Cyclone: A novel design schema for hierarchical DHTs. In *P2P*, pages 49–56, Washington, DC, USA, 2005. IEEE Computer Society.

[6] M. Backes, M. Maffei, and K. Pecina. A security API for distributed social networks. In *NDSS*, 2011.

[7] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. Persona: an online social network with user-defined privacy. In *ACM SIGCOMM*, 2009.

[8] F. Beato, M. Kohlweiss, and K. Wouters. Scramble! your social network data. In *Proceedings of the 11th international conference on Privacy enhancing technologies*, PETS'11, pages 211–225, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] S. Ben Mokhtar, A. Pace, and V. Quema. FireSpam: Spam Resilient Gossiping in the BAR Model. In *29th IEEE Symposium on Reliable Distributed Systems (SRDS 2010)*, Nov. 2010.

[10] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. The GOSSPLE anonymous social network. In *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*, Middleware '10, pages 191–211. Springer-Verlag, 2010.

[11] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Security & Privacy*, 2007.

[12] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10. IEEE Press, 2010.

[13] S. Buchegger, D. Schiöberg, L. H. Vu, and A. Datta. PeerSoN: P2P social networking — early experiences and insights. In *SNS*, 2009.

[14] B. Carrasco, Y. Lu, and J. M. F. da Trindade. Partitioning social networks for time-dependent queries. In *Proceedings of the 4th Workshop on Social Network Systems*, SNS '11. ACM, 2011.

[15] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *OSDI*, 2002.

[16] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6), 1998.

[17] E. D. Cristofaro, C. Soriente, G. Tsudik, and A. Williams. Hummingbird: Privacy at the time of Twitter. *IACR Cryptology ePrint Archive*, 2011:640, 2011.

[18] L. A. Cutillo, R. Molva, and T. Strufe. Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network. In *WOWMOM*, 2009.

[19] G. Danezis. Inferring privacy policies for social networking services. In *Proceedings of the 2nd ACM workshop on Security and artificial intelligence*, AISec '09, pages 5–10, New York, NY, USA, 2009. ACM.

[20] A. Datta and R. Sharma. GoDisco: Selective gossip based dissemination of information in social community based overlays. In *ICDCN'11*, pages 227–238, 2011.

[21] E. De Cristofaro, M. Manulis, and B. Poettering. Private discovery of common social contacts. In *9th International Conference on Applied Cryptography and Network Security (ACNS)*, volume 6715 of *LNCS*, pages 147–165. Springer, 2011.

[22] Diaspora*. `https://joindiaspora.com/`.

[23] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, 2004.

[24] J. Douceur. The Sybil Attack. In P. Druschel, F. Kaashoek, and A. Rowstron, editors, *International Workshop on Peer-to-Peer Systems (IPTPS)*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, Mar. 2002.

[25] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel. Denial-of-service resilience in peer-to-peer file sharing systems. In *ACM SIGMETRICS*, 2005.

[26] L. Fang and K. LeFevre. Privacy wizards for social networking sites. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 351–360, New York, NY, USA, 2010. ACM.

[27] A. J. Feldman, A. Blankstein, M. J. Freedman, and E. W. Felten. Social networking with Frientegrity: privacy and integrity with an untrusted provider. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, Berkeley, CA, USA, 2012. USENIX Association.

[28] D. Gavidia. A gossip-based distributed news service for wireless mesh networks. In *In Proc. 3rd IEEE Conf. on Wireless On demand Network Syst. and Services (WONS '06*, pages 59–67. IEEE Computer Society, 2006.

[29] D. Gavidia, G. P. Jesi, C. Gamage, and M. van Steen. Canning spam in gossip wireless networks. In *Procedings of the 4^{th} IEEE Conference on Wireless On demand Network Systems and Services (WONS)*, Jan. 2007.

[30] S. Guha, K. Tang, and P. Francis. NOYB: privacy in online social networks. In *Proceedings of the first workshop on Online social networks*, WOSN '08, pages 49–54. ACM, 2008.

[31] S. Jahid, P. Mittal, and N. Borisov. EASiER: Encryption-based access control in social networks with efficient revocation. In *ASIACCS*, 2011.

[32] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia. DECENT: A decentralized architecture for enforcing privacy in online social networks. In *SESOC*, 2012.

[33] A. Kapadia and N. Triandopoulos. Halo: High-Assurance Locate for Distributed Hash Tables. In *NDSS*, 2008.

[34] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03. IEEE Computer Society, 2003.

[35] A.-M. Kermarrec and M. van Steen. Gossiping in distributed systems. *SIGOPS Oper. Syst. Rev.*, 41(5):2–7, Oct. 2007.

[36] B. Klein and H. Hlavacs. A socially aware caching mechanism for encounter networks. *Telecommunication Systems*, 2011.

[37] C. Lesniewski-Laas and M. F. Kaashoek. Whanau: a Sybil-proof distributed hash table. In *NSDI*, 2010.

[38] Y. Liu, K. P. Gummadi, B. Krishnamurthy, and A. Mislove. Analyzing Facebook privacy settings: user expectations vs. reality. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 61–70, New York, NY, USA, 2011. ACM.

[39] M. M. Lucas and N. Borisov. FlyByNight: mitigating the privacy risks of social networking. In *Proceedings of the 7th ACM workshop on Privacy in the electronic society*, WPES '08, New York, NY, USA, 2008. ACM.

[40] M. Mani, A.-M. Nguyen, and N. Crespi. SCOPE: A prototype for spontaneous P2P social networking. In *PerCom Workshops*, 2010.

[41] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. In *IPTPS*, 2002.

[42] G. Mega, A. Montresor, and G. P. Picco. Efficient dissemination in decentralized social networks. In *Peer-to-Peer Computing*, pages 338–347, 2011.

[43] P. Mittal, M. Caesar, and N. Borisov. X-Vine: Secure and pseudonymous routing using social networks. In *NDSS*, 2012.

[44] E. Modiano, D. Shah, and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '06/Performance '06. ACM, 2006.

[45] A. Müller, N. Evans, C. Grothoff, and S. Kamkar. Autonomous nat traversal. In *10th IEEE International Conference on Peer-to-Peer Computing (IEEE P2P 2010)*, pages 61–64. IEEE, 2010.

[46] A. Nambiar and M. Wright. Salsa: a structured approach to large-scale anonymity. In *ACM CCS*, 2006.

[47] S. Nilizadeh, N. Alam, N. Husted, and A. Kapadia. Pythia: a privacy aware, peer-to-peer network for social search. In *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*, WPES '11, pages 43–48, New York, NY, USA, 2011. ACM.

[48] K. Opsahl. Facebook's eroding privacy policy: A timeline. `https://www.eff.org/deeplinks/2010/04/facebook-timeline`.

[49] I. Osipkov, P. Wang, and N. Hopper. Robust accounting in decentralized P2P storage systems. In *ICDCS*, 2006.

[50] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips. TRIBLER: a social-based peer-to-peer system: Research articles. *Concurr. Comput. : Pract. Exper.*, 20(2):127–138, 2008.

[51] A. Rowstron and P. Druschel. Pastry: Scalable distributed object location and routing for large-scale peer-to-peer systems. In *Middleware*, 2001.

[52] Skype. `http://www.skype.com/`.

[53] E. Steel and J. E. Vascellaro. Facebook, MySpace confront privacy loophole. The Wall Street Journal, May 2010.

[54] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in Facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009.

[55] B. Viswanath, A. Post, K. P. Gummadi, and A. Mislove. An analysis of social network-based sybil defenses. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 363–374, New York, NY, USA, 2010. ACM.

[56] C. Wang, L. Xiao, Y. Liu, and P. Zheng. Distributed

caching and adaptive search in multilayer P2P networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, ICDCS '04. IEEE Computer Society, 2004.

[57] Z. Xu and Y. Hu. SBARC: A supernode based peer-to-peer file sharing system. In *Computers and Communication, 2003.(ISCC 2003). Proceedings. Eighth IEEE International Symposium on*, pages 1053–1058. IEEE, 2003.

[58] H. Zhang and S. Vasudevan. Design and analysis of a choking strategy for coalitions in data swarming systems. In *INFOCOM*, 2012.

[59] J. Zhao, P. Zhang, and G. Cao. On cooperative caching in wireless P2P networks. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, ICDCS '08. IEEE Computer Society, 2008.

[60] R. Zhou and K. Hwang. Gossip-based reputation aggregation for unstructured peer-to-peer networks. *Parallel and Distributed Processing Symposium, International*, 0, 2007.