Vis-à-Vis: Online Social Networking via Virtual Individual Servers

Amre Shakimov † Harold Lim † Landon Cox † Ramón Cáceres ‡

Duke University † AT&T Labs ‡
{shan, harold, lpcox}@cs.duke.edu ramon@research.att.com

Abstract

Online social networks (OSN) are hugely popular, but their centralized control of sensitive user data raises important privacy concerns. This paper presents Vis-à-Vis, a decentralized framework for online social networking based on the privacy-preserving notion of a Virtual Individual Server (VIS), a personal virtual machine running within a cloud-computing utility. In Vis-à-Vis, each person manages personal data such as friend lists, photographs, and messages through his own VIS. In addition, VISs collectively self-organize into larger groups corresponding to shared user attributes such as interests and educational background. Vis-à-Vis uses distributed hash tables to provide a decentralized, efficient, and scalable set of operations for joining, leaving, and searching a wide range of OSN groups. We have evaluated our Vis-à-Vis prototype implementation in several virtual-computing environments using experimental parameters observed in the Facebook OSN. Our results show that our prototype's performance is reasonable and demonstrate that Vis-à-Vis represents an attractive and feasible alternative to today's centralized OSNs.

1 Introduction

Online social networking has become ubiquitous and its popularity continues to grow. A social network is a collection of people tied by common interests such as those resulting from friendship, family, work, hobby, or geography. Online social networks (OSNs) provide the electronic infrastructure on which users can come together to identify members of their social networks, form groups, and share information such as personal profiles, photos, and messages. Popular OSNs include Facebook [10], LinkedIn [18], and MySpace [24]. To demonstrate the size and growth of these networks, the Facebook website had more than 132 million unique visitors in June 2008, compared to 52 million a year earlier [32]. Clearly, OSNs like Facebook provide a great deal of utility to their users.

Unfortunately, the nature of today's OSN services raises important concerns about user privacy and autonomy. In particular, the dominant services concentrate personal data for millions of people under a single administrative domain. This centralization makes them vulnerable to large-scale privacy breaches from intentional and unintentional data disclosures.

Commonly used terms of service that grant OSN service providers full rights to all content posted to their services (e.g., [13], [19]) raise additional concerns. OSN users cede ownership of their data to their service provider, giving providers complete control over how their users' data is distributed. Privacy policies published by service providers offer some protection, but policies are subject to change at any time and at the sole discretion of the service provider (e.g., [12], [17]).

In this paper we introduce *Vis-à-Vis (VaV)*, a decentralized approach to online social networking that avoids the above problems. In Vis-à-Vis, each person stores personal data on his own individual server, allowing each person to retain control of his data along with the associated software and access-control policies. Each server acts as a proxy for its owner in the context of OSN activities. In particular, the servers form overlay networks that represent OSN groups, with one overlay per group. The same server can belong to multiple overlay networks, just as one person can belong to multiple social groups.

Vis-à-Vis relies on the concept of *Virtual Individual Servers (VISs)*, where each person's server is hosted in its own virtual machine running within a utility computing infrastructure such as Amazon Elastic Compute Cloud (EC2) [1]. We believe that individual consumers will adopt virtualized utility computing for many of the same reasons that enterprises are adopting it: VISs unburden users from having to maintain their own high-availability hardware without forcing them to give up control of their data, software, and policies. In contrast to free OSN services, paid utility-computing services such as EC2 allow users to retain full rights to the content and applications that users place on these services [2]. Furthermore, the highly distributed nature of Vis-à-Vis makes a large-scale privacy breach much less likely than in centralized OSN services.

This paper makes the following contributions:

- It presents the design of Vis-à-Vis, a privacy-preserving framework for online social networking based on the novel concept of Virtual Individual Servers.

- It describes a prototype implementation of VaV that has proven to work in a variety of virtualized computing environments, including Emulab, PlanetLab, and EC2.
- It evaluates VaV in those environments using experimental parameters taken from the Facebook online social network.

Experimental results using our prototype implementation indicate that Vis-à-Vis is a viable and desirable alternative to the prevailing OSN service architecture. The measured latency of common OSN operations grows slowly, if at all, with the size of the corresponding OSN group. Similarly, the memory required by a VIS to participate in Vis-à-Vis is manageable and grows with the size and number of OSN groups to which a user belongs.

2 Background

Individuals increasingly require a permanent online presence, as evidenced by their growing use of online services such as blogging and photo-sharing services. We argue that their interests would be better served by using their own VISs to host such services, particularly OSN services.

The drawback of relying on VISs is currently their cost. At the moment, Amazon EC2 charges ten cents per hour for a default virtual machine with 1.7 GB of memory, 1 virtual core, and 160 GB of persistent storage. Data-transfer fees vary depending on the location of the machines the virtual machine is communicating with. Even without network-usage fees, maintaining a virtual machine on EC2 for a month currently costs close to $75.

We recognize that many people may choose to give up privacy and autonomy in exchange for free service. However, we feel that it is valuable to explore the benefits and limitations of alternatives such as Vis-à-Vis, especially as public awareness of privacy issues grows and the price of utility computing drops.

Vis-à-Vis aims to satisfy the following goals: to mimic the privacy expectations and trust relationships in offline social networks; to support both OSN groups that anyone can join and groups with restricted membership; to allow both public groups whose existence is openly advertised, and secret groups whose existence is known only to their members; to scale to both very large and very small groups; to efficiently support common OSN operations such as a user finding groups of interest, joining a group, leaving a group, and searching for information within a group.

3 Design

Online social networks (OSNs) that cede responsibility for user data to a single administrative entity are inherently prone to violations of users' privacy. Sensitive user data creates an attractive target for hackers and can be abused by internal administrators, as some have accused Facebook employees of doing [22]. Even worse, a site can leak inappropriate information directly to users' close social relations. For example, Facebook's Beacon program caused an uproar when it began actively reporting users' online activity such as purchases and visited web pages to their friends [20]. As OSNs become more entwined with users' lives and acquire more detailed personal information (e.g., location histories), these dangers will grow.

To preserve users' privacy, an OSN could also be implemented as a peer-to-peer federation of independent administrative domains, with each member responsible for storing and serving her own data using a home or work PC. Unfortunately, providing reasonable availability under the churn characteristic of desktop machines requires data to be widely replicated [23].

A conventional replication strategy in which any node can host any object is inappropriate in this setting given the sensitive nature of users' data. A socially-informed replication strategy might be possible, but introduces several additional complexities. For example, a set of socially-connected hosts may not fail independently of one another.

A more fundamental difficulty with replicating OSN data on friends' machines stems from friends' non-overlapping social connections. Any pair-wise shared secrets between friends cannot be replicated, requiring a centralized public-key infrastructure (PKI) for authenticating requesters and enforcing access control policies. Furthermore, aggregated data such as the Facebook news feed that collects updates from a user's friends can only be replicated at hosts that are authorized to access each of these data feeds.

In this paper, we present *Vis-à-Vis*, an architecture for OSNs that resolves these tensions through *Virtual Individual Servers (VISs)*. A VIS is a personal virtual machine instance hosted by a cloud computing utility such as Amazon's EC2. The key insight behind the Vis-à-Vis architecture is that VISs enable OSNs in which *data management and service availability are decoupled*. Building an OSN of federated VISs offers an attractive new alternative to existing OSN architectures because it gives users direct control of their sensitive personal data and relies on the compute utility to ensure that machines are highly available.

The goal of the Vis-à-Vis architecture is to give users complete control of their data and to support as many features of existing OSNs as possible. The first step toward this goal was recognition that OSN data can generally be categorized as either *restricted* or *searchable*. Restricted information is used to enable data sharing

among mutually trusting users. Restricted information is only accessible to authenticated parties and is subject to access-control policies specified by the data owner. Searchable information in OSNs is accessible to a much wider subset of the OSN and allows strangers with similar attributes or interests to find each other.

In Vis-à-Vis, restricted information is stored and managed by users' individual VISs. VISs act as reference monitors over this sensitive data, authenticating requesters and enforcing access-control policies. Vis-à-Vis manages searchable information through the *typed group* abstraction. Typed groups represent a potentially large set of users who share an attribute. Vis-à-Vis can support nearly all searching and browsing features common to OSNs through a concise set of primitives for manipulating this abstraction.

The rest of this section describes Vis-à-Vis's data-management schemes in greater detail.

3.1 Restricted Information

Restricted information in Facebook generally consists of 1) complete-profile views including friend lists, personal pictures, walls, news feeds, and other activity notifications and 2) inter-user messaging state. Restricted information can only be viewed by other users with the explicit consent of the data owner.

Support for restricted information in Vis-à-Vis is relatively straightforward. Each user's VIS maintains pointers to its friends' VISs and uses a protocol such as Diffie-Hellman [8] to negotiate cryptographic shared secret keys whenever a new friend link is established. Once these keys are negotiated, if a user wants to access her friend's restricted information, their VISs mutually authenticate and establish a secure communication channel. Using this secure channel, the querying VIS invokes a well-known Vis-à-Vis API to access her friend's data.

In response to these API requests, each user's VIS acts as a reference monitor, consulting access-control policies to determine whether the requesting VIS is authorized to perform a given operation. Of course, once a VIS gives an authorized VIS a piece of restricted information, it cannot stop that other VIS from further divulging that information. However, such a betrayal of trust is not substantially different from those that can happen in offline social networks when one person gives information in confidence to another.

The state that a VIS must maintain to securely serve restricted information grows with the size of a user's list of friends. Both prior studies of OSNs and our own limited study of Facebook show that a typical user will maintain links with hundreds of friends. Given the relatively small size of the pair-wise cryptographic state that would have to be maintained for each friend, VISs should have no trouble managing this data.

3.2 Searchable Information

Searchable information in Facebook consists of 1) users' "networks," which refer to a user's employer, past or present educational institutions, or current geographic region, 2) biographical details such as a user's religious views, birthdate, cultural interests, and hometown, and 3) public groups and "fan pages." Some searchable information is only available to members of the same network. Since searchable state is shared among thousands and sometimes millions of users, managing it in a single VIS is infeasible.

As a result, searchable information in Vis-à-Vis is accessed and manipulated through the *typed group* abstraction. This abstraction is general enough to capture most of the ways in which users are browsed in existing online social networks. Typed groups logically consist of users with shared interests or attributes and are named by a *descriptor* that can be expressed as a $\langle \mathsf{type}, \mathsf{key} \rangle$ pair. For example, a user with name "Jane Doe", who graduated from Duke University, and works for AT&T might wish to join the groups $\langle \mathsf{FULLNAME}, \mathsf{JaneDoe} \rangle$, $\langle \mathsf{FIRSTNAME}, \mathsf{Jane} \rangle$, $\langle \mathsf{LASTNAME}, \mathsf{Doe} \rangle$, $\langle \mathsf{EDU}, \mathsf{DukeUniversity} \rangle$, and $\langle \mathsf{EMPL}, \mathsf{AT\&T} \rangle$.

Vis-à-Vis relies on types to disambiguate groups that might otherwise have similar descriptions. Vis-à-Vis does not restrict descriptors' format or provide a predefined set, as is the case for top-level domains in DNS. Types are only provided for convenience; users are free to assign arbitrary descriptors to the groups they create, although we believe that a set of well-known types would emerge in any real deployment.

Vis-à-Vis supports four operations on typed groups under the following semantics:

- bool create(descriptor) Creates a group and adds the creator as the initial member. If the group already exists, the operation fails.
- bool join(descriptor) Allows a user to insert herself into a group. Success depends on the admission policy for the group.
- bool leave(descriptor) Allows a user to remove herself from a group. This operation should always succeed.
- string query(descriptor, string) Allows a user to send a query to all members of a group. Depending on the group semantics, the query may require a user to authenticate themselves to receive an answer.

Vis-à-Vis implements the typed-group abstraction through a multi-tiered distributed hash table (DHT) structure. We use DHTs because of their scalability and existing support for multicast communication [7]. Though many DHT deployments have sufferred from problems with prohibitive volumes of control traffic [28]

and lack of security guarantees [6], the stability of the compute utilities in which individual VISs execute saves Vis-à-Vis from these problems.

First, since VISs are maintained by a compute utility, they are highly available. This allows nodes in Vis-à-Vis to safely suppress the vast majority of keepalive messages that might be required to maintain the same structure in more volatile environments.

Second, as compute utilities begin to deploy trusted platform modules (TPMs), Vis-à-Vis will be able to avoid the correctness and security concerns that can arise in open DHT platforms due to Byzantine and malicious nodes. TPM has been virtualized and integrated into common utility hypervisors such as Xen [4]. A TPM infrastructure within the cloud will allow compute utilities to prove to their customers what software is executing under their account. For Vis-à-Vis, this will allow nodes to prove to each other that they are executing correct protocol implementations.

However, even with TPM, since Vis-à-Vis is an open framework without a centralized PKI, it cannot prevent all Sybil attacks [9]. A single user can create multiple anonymous Vis-à-Vis presences or use social-engineering techniques to masquerade as another user. All of the OSNs of which we are aware suffer from the same problem. Facebook provides mechanisms for defeating some forms of masquerading by restricting access to certain network attributes. For example, Facebook verifies that a user has access to an email address in the `duke.edu` domain before allowing them to join the Duke University network. As a result, if a user receives a friend request from someone claiming to be their college boyfriend, the likelihood that the requester is actually their former boyfriend is greater if they can demonstrate membership in the college's network. As we will describe shortly, since Vis-à-Vis supports arbitrary group admission policies, Vis-à-Vis users can apply similar anti-Sybil measures when appropriate.

Vis-à-Vis's multi-tier DHT structure is composed of a set of highly-available VIS instances, where each VIS corresponds to a human identity. VISs assign themselves a unique 160-bit identifier and collaboratively form a logical ring using the identifier namespace. The top-level DHT is called the *Meta Group*. It maps keys consisting of the cryptographic hash of a group descriptor to values consisting of a fixed-size list of VISs identifiers and small amount of additional data describing the group such as a an XML specification of the group's query API.

Typed groups are implemented as DHTs on the same identifier space as the Meta Group and are maintained by the VISs of its members. This requires VISs to maintain routing state and store key-value pairs for the Meta Group and every typed group of which they are members. Lists stored in the Meta Group contain a subset of the VISs whose owners are members of the typed group described by the corresponding hash input.

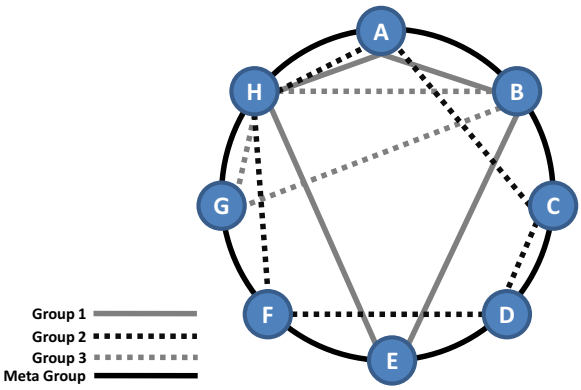

Figure 1: **Example Vis-à-Vis Network with eight Virtual Individual Servers and three groups.**

Figure 1 shows a Vis-à-Vis network of eight VISs and three groups. Group 1 is composed of VISs A, B, E, and H, Group 2 is composed of A, C, D, F, and H, and Group 3 is composed of B, G, and H. All VISs are members of the Meta Group.

Admission, key-value semantics, and access control of typed-group DHT state can be arbitrarily defined by the group members. For example, group-specific shared state such as descriptions, notifications, messages, and pictures can be stored in a group's DHT. The storage burden of this state is only borne by a group's members. Because failures are rare, traffic surges due to reestablishing k-replication guarantees over this shared group state will be infrequent.

An attractive side-effect of Vis-à-Vis's typed-group implementation is that it enables closed, secret groups. For example, a group descriptor could be established out-of-band among a cabal of mutually trusting human users. By controlling admission and establishing group cryptographic state among their VISs, the group could be established within the Vis-à-Vis framework without the knowledge of anyone but the group members themselves.

The create, join, leave, and query operations for accessing typed groups are implemented as follows:

create

To create a typed group, a user first performs a lookup of the appropriate key in the Meta Group. If the lookup succeeds, then an error is returned. If the lookup fails, then the user inserts a new list with her VIS as the sole member into the Meta Group under the appropriate key and instantiates a routing table for the newly created DHT.

join

To join a group, a user performs a lookup of the appropriate key in the Meta Group. If the lookup fails, then an error is returned. If the lookup succeeds, then the user's VIS chooses a node in the returned list and requests permission to join the group.

Vis-à-Vis supports arbitrary admission policies. For example, in a completely open group, the initial node can immediately return DHT routing state for the group to bootstrap the new member as well as any group-specific cryptographic state. More interesting policies are also possible. Mimicking the admission policies of Facebook's educational networks, group members could require evidence of access to an email address within a specific domain. Alternatively, a group could require explicit human consent from a majority, or even all, of of its members before adding a new member.

Regardless of the policy, once an admission process has reached a decision, a user's VIS returns the result.

leave

To leave a group, a user will need to drop the DHT routing state associated with the group and stop responding to keepalive messages for that DHT.

query

To query a group, a VIS can use a multicast service such as Scribe [7] to contact all members of a group. If the group was configured without multicast support, the VIS can also use query the basic DHT, thereby trading space to maintain multicast trees for time to traverse the DHT in a more naïve way. As with admission, groups are free to provide arbitrary query APIs and can store an API description in the Meta Group along with the short list of members.

3.3 Common OSN Features

To demonstrate the generality of the Vis-à-Vis architecture and the typed group abstraction in particular, this section describes how many common OSN features can be implemented using Vis-à-Vis.

3.3.1 Searching and Browsing

Groups can be searched by using the descriptor to retrieve an initial list of members and the query-API description from the Meta Group. The IP addresses of members' VISs can be resolved using the routing state of the Meta Group. Following the format outlined in the API description, a user can then submit a query to each of these addresses. To find a friend within a particular group, a user would perform a lookup of their VIS identifier in the group DHT.

3.3.2 Suggestions

Friend and group suggestions are an extremely useful feature of OSNs. This feature can be implemented in Vis-à-Vis as follows. First a user's VIS queries each of her friends' VISs for a list of their friends. The user's VIS then counts the number of non-mutual friends returned by her friends and sorts this set in descending count order. The first n users in this sorted list can be suggested as people the user might want to be friends with. This process can also be applied to groups and interests.

3.3.3 Third-Party Applications

Facebook applications have quickly become a popular feature of the OSN [15]. Many of these small applications mimic many features of groups by aggregating information about their install base on off-OSN servers. For example, the "Visual Bookshelf" application allows friends to share information about which books they have recently read. These applications can easily be implemented as typed group by searching for their friends' VIS identifiers within the application-specific DHT and submitting queries to those that are found.

Another common type of Facebook application allows users to embed data from non-Facebook sources within their Facebook profile. For example, the "My Flickr" application allows installers to automatically display in Facebook photographs they have posted to the Flickr photo sharing service. These applications can simply run in a user's VIS, grabbing data from disparate Internet accounts and integrating into a single Vis-à-Vis view.

3.4 Limitations

Although, Vis-à-Vis can support many common OSN operations, some are either expensive or infeasible to implement. One such feature is an "Advanced Search" interface which allows users to filter users across a wide range of attributes. An example advanced search query might be to find all users in the Duke network, living in New York, who list basketball and sushi as an interest. Vis-à-Vis struggles with this kind of query because it cannot easily compute the intersection of multiple typed groups. This requires a more centralized data-management infrastructure than Vis-à-Vis can support. This type of limitation is a tradeoff that Vis-à-Vis incurs for the sake of improved privacy.

4 Implementation

We built a Vis-à-Vis prototype following the design described in the previous section and deployed this prototype on a variety of virtualized computing environments. This section describes our implementation.

4.1 Core DHT-Based Software

Our Vis-à-Vis prototype uses Pastry [31] to provide basic distributed hash table (DHT) functionality. We also use Scribe [7] to provide multicast functionality on top of Pastry, but only in groups whose configuration options require multicast. We first give some background on Pastry and Scribe, then describe our additions to this software base.

4.1.1 Pastry

Pastry [31] is an implementation of distributed hash tables. Like similar systems, Pastry provides routing of small messages across the overlay of nodes associated with the same DHT, and exhibits a number of desirable properties such as scalability, fault tolerance, and automatic load rebalancing. Every node in Pastry has a unique ID (a 128-bit unsigned integer) representing a position in a logically circular keyspace. These IDs can be obtained, for example, by hashing the IP address or DNS name of a node. Each node maintains 3 different tables: a routing table, a neighborhood list table, and a leaf-nodes table. Pastry uses these tables to help route messages within the overlay.

The properties of Pastry DHTs guarantee that the number of hops each message takes will not be greater than $log_{2^b} N$, where N is the number of nodes and b is a configuration parameter that is typically set to 6. Additionally, the space required for storing the routing table is not greater than $log_{2^b} N * (2b + 1)$ entries.

There is no fundamental reason why a Vis-à-Vis implementation must be based on Pastry. We could have based ours on any similar DHT system such as Chord [33], CAN [27], or OpenDHT [29]. We chose Pastry because of the availability of a robust, open-source software base [21] upon which a number of higher-level applications and services have been built and deployed.

4.1.2 Scribe

Scribe [7] is such a service, a decentralized multicast and publish/subscribe facility built on top of Pastry. Like Pastry, Scribe uses heartbeat messages to automatically detect nodes joining and leaving, either voluntarily or due to failure. It then re-organizes its routes accordingly. A Scribe group is formed by the union of Pastry routes from each group member to the root of the group. By utilizing Pastry's routing mechanism, the union of Pastry routes is ensured to be a tree. Furthermore, each node in the Scribe group maintains a children table pointing to all of its child nodes in the multicast tree. When a node receives a multicast message, it forwards the message to all of the nodes in its children table. In addition to multicasts, Scribe provides functionality for anycasts and event notifications.

4.1.3 Additions to Pastry and Scribe

Meta Group: We made minimal modifications to Pastry to implement our Meta Group concept. For join and leave operations, the default Pastry functionality is sufficient because this group's join and leave semantics are identical to Pastry's. For information searching and advertising operations, we created two new types of Pastry messages: searchGroupInfo and GroupInfo. The contents of these messages are the identifying information of the group. To search or advertise, the hash of the group information is generated and used as the ID to which the message is routed. Then, the node with the closest ID to the generated ID receives the message. We extended Pastry's default message handler to handle these two new types of message.

When a node receives a GroupInfo message, it first searches its own database for the group information specified in the message. By database here we mean any suitable data store, from a simple XML file to a full relational database server. If the group information does not exist in the database, the node adds a new entry containing the received group information. If the group information already exists, the node only updates the entry if the sender is the same as the owner of the group information, as specified in the database.

Groups: As mentioned earlier, the Vis-à-Vis architecture supports a wide variety of OSN groups by allowing the creator of a group to specify a number of independent configuration parameters. These parameters include:

- Membership approval policy, e.g., open to everyone, requires the approval of the administrator, requires the approval of a minimum number of existing members, subject to a vote by all current members, etc.
- Membership notification policy, e.g., no one is notified of new joins and leaves, only the administrator is notified, all members are notified, etc.
- Group visibility policy, e.g., group is advertised in a Meta Group, group is kept secret, etc.

Our implementation and evaluation work has focused on two example group configurations that represent two extremes in the range of options available in this parameter space. In the remainder of this paper we refer to these two configurations as follows:

- Open Group: Membership requests are immediately granted without consulting anyone, and no one is notified of joins and leaves.
- Closed Group: Membership requests are subject to a vote by all existing members, and all members are reliably notified of joins and leaves.

We now describe the implementation of these two sample group configurations.

Open Group: An Open Group is a modified Pastry group. It is similar to a Meta Group because any node can join the group. Nevertheless, we modified the Pastry join protocol to have an explicit out-of-band handshake between the candidate node and a representative of the group. In the case of an Open Group this handshake is implemented not out of necessity but to have a uniform mechanism across different group configurations. In this way all groups use the same mechanism and the creator-defined policies are configurable per group.

To handle the modified version of the join protocol, we created two new types of Pastry message: JoinGroupRequest and JoinGroupResponse. For Open Groups, a candidate node sends a JoinGroupRequest message to a representative of the group. When the representative node receives the message, it first checks whether it belongs to the group specified in the JoinGroupRequest message. If it is a member of the specified group, it then sends an approval message to the candidate node through a JoinGroupResponse message. When the candidate node receives an approval, it formally joins the Open Group by joining the DHT of the open group through the representative node.

As with a Meta Group, leaving an Open Group is identical to leaving a Pastry group. Therefore, a member leaves an Open Group by calling Pastry's destroy function. This function destroys and removes the calling member's node from the Open Group DHT.

To find and retrieve information in a group, we created two messages: tagMessage and searchTagResult. If a member wants to find a keyword, a tagMessage containing the keyword is created. To determine where to send this message, the hash of the keyword is used as the destination ID. This message gets routed to the node with the closest ID to the destination ID, which in turn sends back a searchTagResult message containing data associated with the keyword. In our implementation, the recipient node searches its XML file for the tag with the same keyword and all related entries are returned.

Closed Group: A Closed Group is a modified Scribe group. A Closed Group uses the same handshake protocol as an Open Group. However, we made significant additional modifications to both Pastry and Scribe to maintain the invariant that every group member keeps an accurate list of all other group members. When a representative node receives a JoinGroupRequest, it forwards the message to the creator of the group. The creator then initiates a voting mechanism by multicasting a VoteRequest message to the group. This message contains the credentials of the candidate node and the deadline for the vote. We added a timer to Pastry that the creator uses every time a vote is required. When the timer goes off, the creator starts counting all of the ballots received and then sends a response message containing the result of the vote to the candidate node, through the representative node. The representative node also saves the result in its own database.

When a member node receives a VoteRequest message, it stores the vote information in its own database for human inspection. The owner of this node can then review its list of pending vote requests and send his ballot at a later time. The user interface is web-based, as explained in the next subsection.

When the candidate node receives a response message, it can then formally join the group only if the result is positive. To formally join the group, the candidate node joins the Closed Group DHT through the representative node. The representative node then checks the result of the vote in its database. If the result is positive, the candidate node is added to the Closed Group DHT and a membership update is multicast to the Closed Group.

We also modified Scribe to maintain the member list invariant during node failures. In Scribe, each node sends a heartbeat message to its parent node. When a parent node fails to respond to the heartbeat message, a node-fault handler automatically reorganizes the multicast tree. We extended the Scribe node-fault handler to also notify the creator of the group about the parent node's failure. To determine whether the parent node actually failed, the creator of the node then sends a liveness-check message to the parent node. If the parent node still fails to respond to this message, then the creator of the group multicasts a message to remove the parent node from the member list.

Leaving a Closed Group is not as simple as leaving an Open Group because each member needs to be notified. A member must first notify all members of the group before leaving the Closed Group by multicasting a RemovalUpdateMemberList message, which tells all recipients to remove the member from their respective member list. Only after multicasting this message can a member call Pastry's destroy function.

In our implementation, searching for information in a Closed Group works the same way as in an Open Group.

4.2 Auxiliary Web-Based Software

For testing purposes we also developed a primitive web-based user interface to Vis-à-Vis. Each VIS runs an Apache Tomcat server in addition to the core DHT-based software described above. We deployed Java Server Pages (JSPs) and Servlets in the Tomcat server to implement the user interface and its underlying logic. The JSPs present simple web forms that a person can use to join and leave a group, search and advertise group information, search and advertise tags, and cast ballots for pending vote requests. The Servlets store objects related to the Meta Group and other groups as session objects.

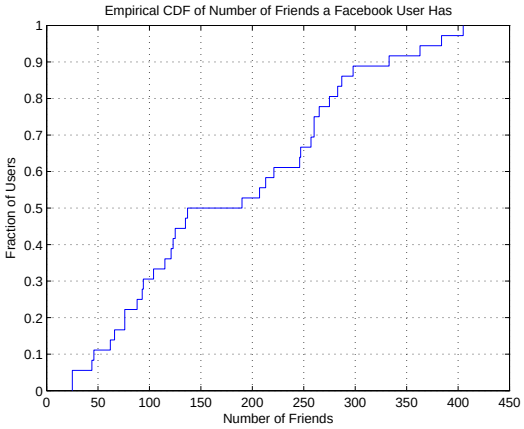

Figure 2: **Empirical CDF of the number of friends a Facebook user has.**

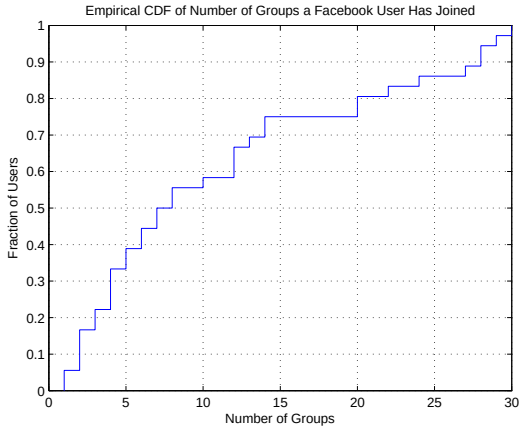

Figure 3: **Empirical CDF of the number of groups a Facebook user has joined.**

5 Evaluation

5.1 OSN Characterization

We set out to characterize real OSNs in order to have up-to-date statistics with which to drive our evaluation of Vis-à-Vis. In particular, we measured the following Facebook characteristics:

- Number of friends a user has.
- Number of groups a user has joined.
- Number of members in a group.

We used these numbers to gain insight into appropriate parameters to use in our experiments, since we want to evaluate the performance of our prototype as it would be used by a typical OSN user. We chose Facebook to be the source of these numbers because:

- Facebook is by some measures the largest OSN service in the English-speaking world [32].
- Facebook has an open API that can be used for developing third-party applications [11].

5.1.1 Facebook Application

In order to obtain the basic parameters of Facebook, we developed a simple Facebook application named "Online Social Network Portrait". This application gathers and stores the aforementioned characteristics of every user that agrees to run it. For privacy reasons we used the MD5 cryptographic hash algorithm [30] to scramble all user and group IDs before saving anything to stable stage.

5.1.2 Facebook Statistics

We have been running the "Online Social Network Portrait" application for approximately two weeks[1] and

[1]Information about this application and the most current results are available on the application group page http://www.facebook.com/group.php?gid=41974516054

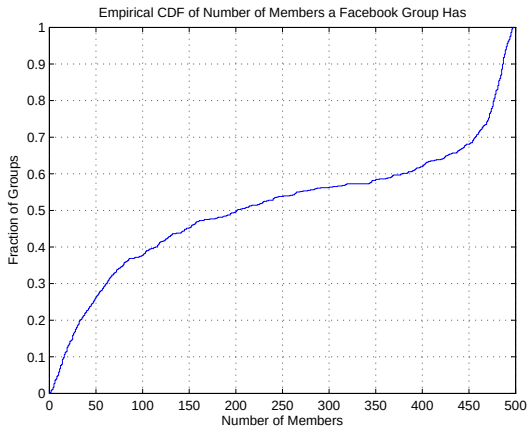

Figure 4: **Empirical CDF of the number of members in a Facebook group.**

sampled 669 distinct groups and 43 users. Figures 2, 3, and 4 show cumulative distributions for the three characteristics of interest, while Table 1 shows summary statistics.

	min	max	avg	std deviation
# of friends per user	1	612	187.06	114.67
# of groups per user	1	51	11.97	10.26
# of members per group	25	497	246.24	196.49

Table 1: **Summary statistics of sampled users and groups on Facebook (excluding one minimum and one maximum value)**

5.2 Experimental Testbeds

We evaluated the performance of our Vis-à-Vis prototype on Emulab and PlanetLab, two experimental testbeds composed of virtualized computing nodes. We also deployed VaV and tested its correctness on several other collections of machines running the Xen virtual machine monitor [3]. We will now describe each testbed.

5.2.1 Initial Testbeds

We first deployed the Vis-à-Vis prototype on three different environments: a cluster of laboratory machines at AT&T Labs, an experimental utility computing cluster at Duke University, and the commercial utility computing infrastructure provided by Amazon EC2. Machines in all three environments ran the Xen virtual machine monitor. These experiments served to show the correctness of our VaV design and implementation, but we were limited in how many virtual machines we could deploy in these environments. For example, EC2 currently limits each customer to 20 virtual machines. As a result, we moved on to conduct larger-scale experiments on Emulab and PlanetLab.

5.2.2 Emulab

Emulab [35] is a network testbed that provides resources for conducting experiments on distributed systems. All of the resources are located on a single site, the University of Utah. Each user can request a set of virtual machines to emulate a network with arbitrary topology. Emulab provides the user with an interface to specify the bandwidth, latency, and losses between nodes, along with other parameters. Furthermore, a user can create a virtual image that can be uploaded to all of her nodes in the experiment.

We provisioned 105 physical Emulab nodes for our experiments. On each node we installed a pre-built VIS image containing our Vis-à-Vis prototype software, Apache Tomcat 6.0.18, and Java JDK 1.5. We used a simple topology where all nodes are connected to a single 100Mbps network switch, without artificial latency and losses. We used the experimental results from this simple topology as a baseline with which to compare our results from PlanetLab, which in contrast capture many complexities of real-world networks.

5.2.3 PlanetLab

PlanetLab [34] is a world-wide research network for the deployment of distributed systems. PlanetLab consists of hundreds of sites around the world. Each of these sites hosts one or more physical nodes. Authorized users can request virtual machines from any site for their experiments. Unlike Emulab, PlanetLab sites are geographically distributed. PlanetLab experiments thus capture characteristics of the live Internet: variable bandwidth, latency, and packet losses. In addition to network variability, PlanetLab experiments capture the effects of load placed on the test nodes themselves by the many experiments running on PlanetLab at any one time. For example, during our experiments the average CPU load on each node was greater than 10, as reported by the Linux *top* command.

For our PlanetLab experiments we provisioned 120 VISs distributed throughout the Earth, as seen in Figure 5. On each VIS we installed the same software that we installed on our Emulab VISs, namely our Vis-à-Vis prototype software, Apache Tomcat 6.0.18, and Java JDK 1.5.

5.3 Vis-à-Vis Performance Characteristics

We evaluated the performance of our Vis-à-Vis prototype using three criteria: latency of basic OSN operations, memory overhead incurred to maintain the necessary data structures, and traffic required to maintain the underlying DHT structure.

5.3.1 Latency of OSN Operations

The time VaV takes to complete OSN operations is critical to the user experience. We measured the latency of the following example operations as we varied the group size: joining an Open Group, joining a Closed Group, and searching for information in a group.

For each of these sample operations, we created a Meta Group consisting of all available VISs. Every operation was conducted 5 times on a group size of 20, 40, 60, 80, and 100 VISs randomly chosen from our 120 provisioned VISs. Figures 6, 7, and 8 report the average of these results along with their standard deviations.

Joining an Open Group: For this experiment we randomly picked one node from the Meta Group and had it request to join the Open Group through a randomly picked member of the Open Group. We measured the time between when the candidate node sent the join request and when it became a member of the group. The results are shown in Figure 6.

This figure shows that join latency for this type of group does not grow appreciably as the group size grows from 20 to 100. The Emulab results are quite stable. The PlanetLab results exhibit more variability due to the varying network characteristics and testbed loads discussed earlier. Overall, these results indicate that the join operation will scale well to larger group sizes, at least to sizes in the hundreds of members, the largest we saw in our Facebook study.

Joining a Closed Group: For this experiment we randomly picked one node from the Meta Group and had it send a join request to a randomly chosen member of the Closed Group. Recall that our Closed Group re-

Figure 5: **Geographic distribution of the 120 Virtual Individual Servers in our PlanetLab experiments.**

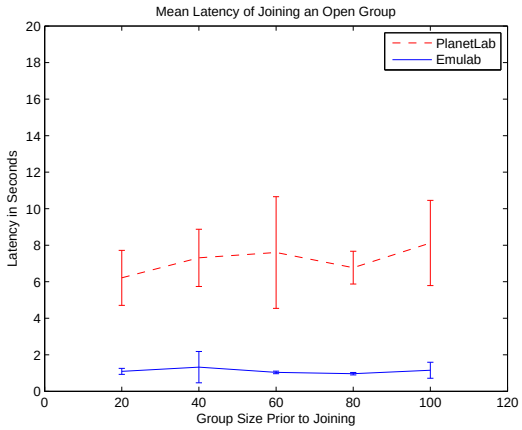

Figure 6: **Mean latency of joining an Open Group in Emulab and PlanetLab. Error bars show the standard deviation.**

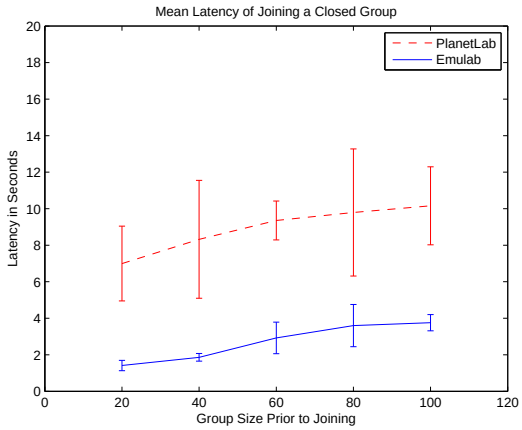

Figure 7: **Mean latency of joining a Closed Group in Emulab and PlanetLab. Error bars show the standard deviation.**

quires that all members vote on whether to admit the candidate node to the group. In addition, it requires that all members be notified as to whether the join request was granted or not. Because both of these configuration choices require that all group members be contacted before the join operation can complete, our Closed Group can be thought of as the worst case for join latency. Figure 7 shows the measured join latency, not counting the human think time that would normally be required to allow group members to enter their votes.

As shown in Figure 7, the latency of joining a Closed Group grows slowly with the size of the group. This growth is due to those configuration choices that require all members to be contacted as part of the join operation. Groups created with less burdensome configurations will perform better, as did the Open Group.

Finding and Retrieving Information in a Group: The goal of this experiment is to measure the latency of searching for a keyword within a previously created group. We measured the time between a randomly chosen member of a group submits a query containing a keyword and when it receives the data associated with this keyword. More concretely, during our experiment this node searches for all data associated with randomly generated and previously advertised keywords.

Figure 8 shows that search latency is not greatly affected by group size, as desired. Again the Emulab results were very stable because they come from a controlled laboratory environment, while the PlanetLab results exhibit greater variability because of external factors.

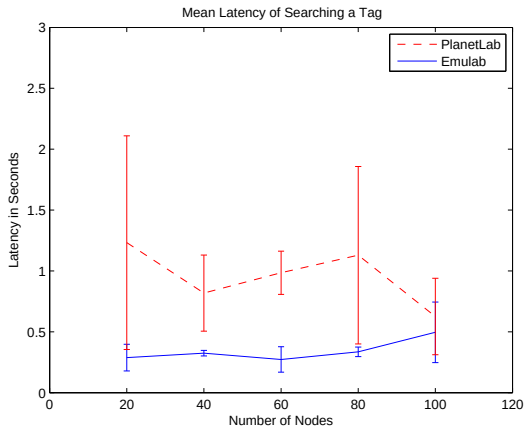

Figure 8: **Mean latency of searching for a data item in Emulab and PlanetLab. Error bars show the standard deviation.**

5.3.2 Memory Overhead

Memory overhead is another important metric because it yields insights into how much memory a VIS needs to dedicate to maintaining Vis-à-Vis functionality. The overall memory consumption for a group consists of the Pastry route table, the Pastry leaf set, other Pastry-related data structures, Scribe-specific datastructures, and VaV-specific data structures. To quantify these values we serialized relevant Java objects and measured their sizes. Other instances of memory consumption were measured directly.

The left and right graph of figure 9 show the measured memory overhead of an Open and Closed Group, respectively. The graph shows that the memory overhead of an Open Group tapers off after a group reaches a certain size. The curve for Open Group increases until the size reaches 24 because in our implementation the Leaf Set size is 24.

In contrast, the memory overhead of a Closed Group grows with the size of the group. The reason for this growth is that the Closed Group was configured to require that each member store the list of all members of the group, whereas this requirement is not there in the Open Group. We can consider the Closed Group configuration to be the worst case for memory overhead, as we did with join latency. A wide range of group configurations will incur less memory overhead that our example Closed Group. Furthermore, we argue that groups such as this Closed Group, where members care to vote on each new membership request and want to know the identity of all other members, have natural size limits that reduce the importance of the scalability limitations that we have just discussed.

5.3.3 Maintenance Traffic

In order to maintain the underlying DHT structure, VISs must exchange regular heartbeat messages with the nodes from their Leaf Sets and Route Tables. In the presence of churn, VISs exchange updated rows and update them locally. The rate of these messages is configurable and determined based on node churn, availability, and failure rate. We see that in a highly available environment maintenance traffic is not an issue due to rare failures. However, it is possible to reduce even this traffic using several performance optimizations, for example, by merging all Leaf Sets and Route Tables of a VIS into a Super Leaf Set and a Super Route Table in order to take advantage of having common members in several groups. Another example may include usage of SuperPastry [5], which is a hybrid of structured and unstructured overlays. SuperPastry significantly reduces churn rate and, therefore, maintenance traffic in the system.

6 Related Work

This section discusses work related to Vis-à-Vis that has not already been mentioned elsewhere in this paper.

Social VPN [14] is a virtual network that exploits social and overlay networks. It uses the infrastructure of social networks, such as Facebook, to exchange credentials of users. Social VPN takes advantage of the relationships formed in social networks to determine which pairs of people already trust each other, so as to automatically configure IPsec tunnels between machines belonging to these pairs. In contrast, Vis-à-Vis keeps track of the social relationships and handles the OSN operations. Moreover, the motivation of Social VPN is fundamentally different from the motivation of Vis-à-Vis. We avoid turning over sensitive information to a central server, since this would undermine the whole idea of Vis-à-Vis.

NOYB [16] takes a completely different approach to the privacy threats of centralized OSNs by encrypting some of the data that users hand to the service. This is appealing because it may allow users of existing OSNs to retain their existing profiles, while Vis-à-Vis requires users to completely divest themselves.

NOYB partitions user data into small cryptographic units called atoms, which correspond to various attributes associated with a user. The technique allows users to cooperatively scramble their profile attributes with other trusted users' via pseudorandom substitution so that friends can locate a user's real attributes, but the OSN cannot. Another key feature of NOYB is that the OSN is oblivious to the presence of the encrypted data, because atoms stored in the OSN look like legitimate profile values.

NOYB represents a very appealing approach to privacy in OSNs, but there are several drawbacks compared

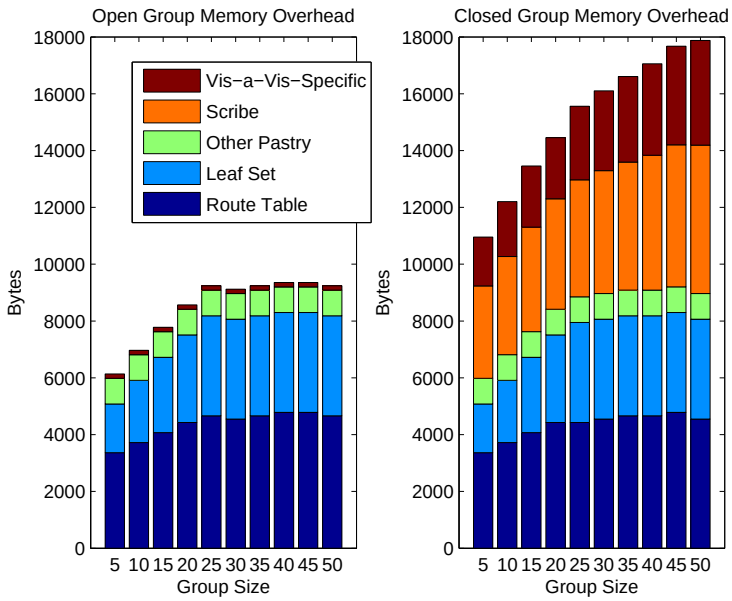

Figure 9: **Memory overhead per Virtual Individual Server to support Open and Closed Groups.**

to Vis-à-Vis. First, retaining any presence in a centralized OSN leaves users open to the "Beacon attack," in which the service directly notifies a user's friends of potentially sensitive activity in other corners of Internet. Vis-à-Vis users are not vulnerable to such attacks because their VISs control all data sent along their social links. Second, NOYB requires additional key-managing software to be installed on any client machine accessing encrypted profile data, including public kiosks and mobile phones where it may not be convenient. Vis-à-Vis only requires clients to have a web browser. Finally, it is unclear how well NOYB generalizes to non-textual information, while Vis-à-Vis can secure arbitrary data types.

Turtle [26] is an anonymous peer-to-peer network designed for private information sharing using pre-existing social relations. Turtle projects trust relationships from the real world into the overlay to eliminate fear of censorship or legal sanctions in open networks. To control sensitive data every node allows to assign a specific attribute set to each shared data item, such that only authorized people are able to get this item. All communications within Turtle are encrypted and search queries are broadcasted to all "friend nodes" using a hop count bit. Vis-à-Vis has a similar motivation as Turtle and has a few similar design decisions. Namely, closed groups in Vis-à-Vis also assume out-of-band mutual trust between nodes and every member of the closed group relies on all members of this group. Thus, betrayal of one member can compromise other members. Besides, sensitive data can be received only through friends, hence, eliminating "man-in-the-middle" attack. However, Turtle does not try to position itself as a social network service, and, therefore, it does not provide any means for forming different groups or advertise points of interests without flooding all members. Moreover, Turtle's underlying architecture is an unstructured overlay, hence, flooding a query may not be efficient when compared to multicast within a DHT. Finally, it is not clear how Turtle behaves in the presence of churn and unavailability of some friends.

Seaweed is a scalable infrastructure for delay-aware querying of large and highly distributed datasets [25]. This infrastructure is used to submit routine queries in WANs and it effectively tolerates unavailability without the need to replicate the actual data. Seaweed trades availability for latency, and one person may be willing to wait longer to get more accurate results, whereas another may be satisfied with less thorough data obtained sooner. This infrastructure is built using a DHT and has all of the attractive properties exhibited by structured overlays: fault-tolerance, scalability and self-organization. While motivation for building Vis-à-Vis and Seaweed are very different, they have a number of similarities. Both of these projects refuse to replicate the actual data, the former because of bandwidth overhead and the latter for privacy issues. Seaweed replicates metadata within the overlay which helps to predict availability of the node even when this particular node is unavailable. Vis-à-Vis uses replication of meta information in the Meta Group for workload balancing as well as for failure-resistance.

7 Conclusion

We have presented Vis-à-Vis, a privacy-preserving framework for online social networking based on the novel concept of Virtual Individual Servers. VISs are personal virtual machines running in a cloud computing facility. Each user owns his own VIS and maintains control over the data, software, and access-control policies on his VIS. Each VIS represents its owner in the context of OSNs. VISs form overlay networks, one overlay for every OSN group that its owner joins. VaV uses distributed hash tables to achieve scalable and efficient OSN operations on this large federation of machines. Most importantly, the decentralized nature of VaV offers great privacy advantages compared to the prevailing OSN architecture based on centralized services. We feel it is valuable to explore the benefits and limitations of alternatives such as Vis-à-Vis, especially as public awareness of privacy issues continues to grow and the price of cloud computing continues to drop.

References

[1] Amazon Web Services LLC. Amazon elastic compute cloud (amazon ec2), 2008. `http://aws.amazon.com/ec2/`.

[2] Amazon Web Services LLC. Amazon web services customer agreement, September 2008. `http://aws.amazon.com/ec2/agreement/`.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM Press.

[4] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vtpm: Virtualizing the trusted platform module. In *USENIX Security*, 2006.

[5] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 85–98, Berkeley, CA, USA, 2005. USENIX Association.

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

[7] M. Castro, P. Druschel, A. marie Kermarrec, and A. Rowstron. Scalable application-level anycast for highly dynamic groups. In *In Proc. NGC 2003*, 2003.

[8] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[9] J. R. Douceur. The sybil attack. In *Workshop on Peer-to-Peer Systems*, 2002.

[10] Facebook. `http://www.facebook.com`.

[11] Facebook. Facebook developers. `http://developers.facebook.com/`.

[12] Facebook. Facebook principles, December 2007. `http://www.new.facebook.com/policy.php`.

[13] Facebook. Facebook terms of use, September 2008. `http://www.new.facebook.com/terms.php`.

[14] R. Figueiredo, O. Boykin, P. St. Juste, and D. Wolinsky. Social VPNs: Integrating overlay and social networks for seamless P2P networking. In *Workshop on Collaborative Peer-to-Peer Systems (COPS), Rome, Italy*, June 2008.

[15] M. Gjoka, M. Sirivianos, A. Markopoulou, and X. Yang. Poking Facebook: Characterization of OSN Applications. In *SIGCOMM Workshop on Social Networks*, 2008.

[16] S. Guha, K. Tang, and P. Francis. NOYB: Privacy in online social networks. In *SIGCOMM Workshop on Social Networks*, 2008.

[17] LinkedIn Corporation. Linkedin privacy policy, July 2008. `http://www.linkedin.com/static?key=privacy_policy`.

[18] LinkedIn Corporation. Linkedin: Relationships matters, 2008. `http://www.linkedin.com`.

[19] LinkedIn Corporation. Linkedin user agreement, July 2008. `http://www.linkedin.com/static?key=user_agreement`.

[20] Louise Story and Brad Stone. Facebook retreats on online tracking, November 2007. The New York Times.

[21] Max Plank Institute for Software Systems. Freepastry. `http://freepastry.org/FreePastry/`.

[22] Megan McCarthy. How Facebook employees break into your profile, November 2007. `http://www.valleywag.com`.

[23] J. Mickens and B. Noble. Exploiting availability prediction in distributed systems. In *In Proceedings of NSDI, 2006*, pages 73–86, San Jose, CA, May 2006.

[24] MySpace. `http://www.myspace.com`.

[25] D. Narayanan, A. Donnelly, R. Mortier, and A. Rowstron. Delay aware querying with seaweed. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 727–738. VLDB Endowment, 2006.

[26] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and private data sharing with turtle: Friends team-up and beat the system. In *In Proc. of the 12th Cambridge Intl. Workshop on Security Protocols*, 2004.

[27] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 31, pages 161–172. ACM Press, October 2001.

[28] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a dht. In *USENIX Techincal Conference*, 2004.

[29] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. Opendht: A public dht service and its users. In *SIGCOMM*, 2005.

[30] R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Internet Activities Board, April 1992.

[31] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[32] E. Schonfeld. Facebook is not only the worlds largest social network, it is also the fastest growing, August 2008. `http://www.techcrunch.com/2008/08/12/facebook-is-not-only-the-worlds-largest-social-network-it-is-also-the-fastest-growing/`.

[33] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[34] The Trustees of Princeton University. Planetlab: an open platform for developing, deploying, and accessing planetary-scale services. `http://www.planet-lab.org/`.

[35] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *In Proc. of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, pages 255–270, Boston, MA, Dec. 2002.