

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

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

Duke University †
Durham, NC, USA
{shan, harold, lpcox}@cs.duke.edu

AT&T Labs ‡
Florham Park, NJ, USA
ramon@research.att.com

Abstract

Online social networks (OSNs) are immensely 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 self-organize into overlay networks corresponding to social groups with whom their owners wish to share information. Vis-à-Vis uses distributed hash tables to provide efficient and scalable operations for joining, leaving, and searching a wide variety of OSN groups. We have evaluated our Vis-à-Vis prototype in several virtual-computing environments using experimental parameters observed in the Facebook OSN. Our results demonstrate that Vis-à-Vis represents an attractive 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 [2], LinkedIn [3], and MySpace [4]. As an example of 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 [31]. Clearly, OSNs 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., [2], [3]) raise additional concerns. Users thereby give providers complete control over how their is distributed. Advertising-driven business models create pressure for providers to use data in ways that may violate consumer privacy. 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., [2], [3]).

In this paper we introduce *Vis-à-Vis*, a decentralized approach to online social networking that avoids the above problems. In Vis-à-Vis, each person stores his own 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. Individual servers communicate with each other as peers and are free to reside anywhere in the Internet.

We focus on *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 allow users to retain full rights to the content and applications that users place on these services [1]. Furthermore, the decentralized nature of Vis-à-Vis makes a large-scale privacy breach much less likely than in centralized OSN services.

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.

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 Vis-à-Vis that has proven to work in a variety of virtualized computing environments, including Emulab, PlanetLab, and EC2.
- It evaluates Vis-à-Vis 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 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 servers to host such services and the associated data. An individual server would have many uses as an online proxy for its owner, and its resources would be shared among the services hosted there. In this work we explore the specific but important use of individual servers to host online social networking services.

2.1 Spectrum of alternatives

We considered three approaches to supporting online social networking via a peer-to-peer federation of individual servers:

- Use highly available virtual machines hosted in the cloud, which we term Virtual Individual Servers.
- Use machines hosted in homes or offices, augmented with socially-informed data replication.
- Use a hybrid of the first two approaches.

The main advantage of cloud-hosted virtual machines is their high availability. Virtual machines that reside and migrate within a professionally managed hardware infrastructure can be assumed to fail very rarely without unduly burdening individual users with management tasks. Their main drawback 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 ma-

chines with which the virtual machine is communicating. Even without network-usage fees, running a virtual machine for a month costs close to US\$75.

A cheaper alternative would be to use desktop-class machines that reside in user's homes or offices. A desktop machine consumes a lot less than US\$75 of energy per month and can share a flat-rate Internet connection with other machines. Unfortunately, providing reasonable availability under the churn characteristics of desktop machines requires data to be widely replicated [21]. 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 is possible but introduces 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.

A third possibility is to use a hybrid of the two approaches just described: use a desktop machine as the primary host and fail over to a cloud-hosted service when the desktop machine becomes unavailable. Running the desktop-based server as a virtual machine would allow efficient transfer of state to the cloud using proactive virtual machine migration techniques such as those used by Internet Suspend/Resume [30]. This solution could combine high availability with low cost because the cloud-hosted virtual machine could provide a stable backup while being quiescent most of the time. However, it is not immediately clear how to initiate a failover since failures are in general unexpected.

In the remainder of this paper we focus on the cloud-hosted case and leave for future work a more detailed exploration of the desktop and hybrid cases.

2.2 Goals for Vis-à-Vis

Vis-à-Vis aims to satisfy the following goals:

- To give users fine control over what personal data they share with whom.
- 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 small and very large groups. The size of OSN groups can range from a few family members to several million people who share a metropolitan area.
- 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.

2.3 Trust and Threat Model

Vis-à-Vis's decentralized structure provides users with stronger privacy protections than centralized services such as Facebook and MySpace, but it cannot eliminate breaches completely. Here we outline the classes of attacks that Vis-à-Vis is and is not designed to address.

Vis-à-Vis makes several assumptions about the guarantees provided by a compute utility and the software executing within each user's VIS. First, we assume that any compute utility that hosts a VIS supports a Trusted Platform Module (TPM) capable of attesting to the software stack in use by the VIS. While such capabilities are rare at the moment, we believe that TPMs will be a common feature among compute utilities in the future. Vis-à-Vis may still function in the absence of TPMs, but can lead to a wide range of security problems that are inherent to open systems [9]. TPM has been virtualized and integrated into common utility hypervisors such as Xen [7]. 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 also allow nodes to prove to each other that they are executing correct protocol implementations.

Vis-à-Vis also assumes that users' VISs are well administered and free of malware. Users must properly configure the access-control policies of their software, and keep their software versions up to date, including applying any critical security patches. As with all other networked systems, software misconfiguration and malware are serious threats to Vis-à-Vis, but are orthogonal to the focus of our design. If an adversary gains control of a user's VIS it would not only gain access to the user's own personal data, but could potentially access others as well by masquerading as the victim.

Vis-à-Vis's trust model is built on the stated business interests of compute utilities and inter-personal relationships of users; both the compute utility and a user's social relations are trusted to not leak any sensitive information to which they have access. Because compute utilities control the hardware on which VISs execute, they can easily access all of a user's personal data. Vis-à-Vis trusts utilities to not exercise this power because their business model is based on selling computational resources to users rather than advertising to third parties. The legal language of utilities' user agreements,

such as EC2's, formalize these limitations [1] and reduce the threat of personal data being abused. Vis-à-Vis also cannot prevent a user's trusted social relations from leaking damaging personal information or colluding with adversaries. As with trust relationships in offline social networks, users assume that their friends will not retell sensitive information to others.

Vis-à-Vis assumes an adversary capable of eavesdropping, deleting, and modifying all network communication between VISs. Adversaries can create an arbitrary number of Vis-à-Vis nodes and use social engineering to masquerade as other, legitimate users. Furthermore, Vis-à-Vis cannot prevent attacks by adversaries who collude with the compute utilities or other users.

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 [20]. 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 [17]. As OSNs become more entwined with users' lives and acquire more detailed personal information (e.g., location histories), these dangers will grow.

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 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 Vis-à-Vis architecture aims to give users complete control of their data and to support as many features of existing OSNs as possible. The first step toward these goals 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 [11] 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.

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 $\langle \text{type}, \text{key} \rangle$ 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 $\langle \text{FULLNAME}, \text{JaneDoe} \rangle$, $\langle \text{FIRSTNAME}, \text{Jane} \rangle$, $\langle \text{LASTNAME}, \text{Doe} \rangle$, $\langle \text{EDU}, \text{DukeUniversity} \rangle$, and $\langle \text{EMPL}, \text{AT\&T} \rangle$.

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 pre-defined 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 five 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.

`bool insert(descriptor, string)` Allows a user to publish searchable data to 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 distributed hash tables (DHTs) [29][32]. We use DHTs because their properties mesh well with the goals of Vis-à-Vis. One, DHTs form peer-to-peer connections between network nodes without relying on any centralized components. Two, DHTs scale up to a large number of nodes while incurring low overhead for a small number of nodes. Three, DHTs support multicast communication [10]. Though many DHT deployments have suffered from problems with prohibitive volumes of control traffic [26] and lack of security guarantees [9], 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.

However, even with TPM, since Vis-à-Vis is an open framework without a centralized PKI, it cannot prevent all Sybil attacks [12]. 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 is based on a multi-tier DHT structure composed of a set of highly-available VISs, 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* and is used to advertise and search for public OSN groups. 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 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 their 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.

Because all VISs participate in the Meta Group, the underlying DHT may need to accommodate tens or even hundreds of millions of nodes. The need to scale to such numbers will primarily affect the DHT in two ways: the volume of traffic needed to maintain the structure and the latency of retrieving typed-group lists. As mentioned previously, control traffic can be suppressed by taking

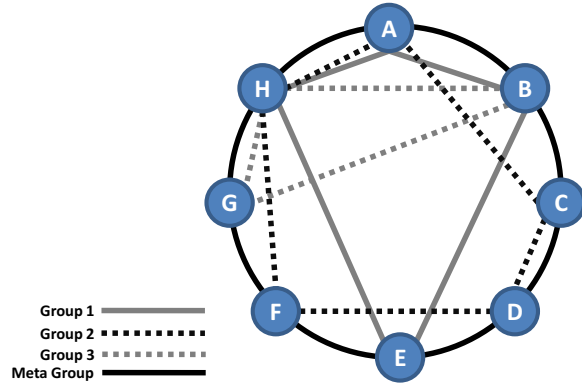


Figure 1: Example Vis-à-Vis Network with eight Virtual Individual Servers and three groups.

advantage of the VISs' stability. Since nodes' availability is ensured by the compute utility, the rate at which keepalive messages are sent can be kept low, and state would be infrequently copied to new hosts in order to maintain sufficient replication factors in the face of failures. DHT latency can be improved through caching techniques such as those proposed by BeeHive [24].

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 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.

The create, join, leave, insert, 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 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.

insert

To publish searchable data to a group, a user inserts the data in the corresponding DHT using the basic DHT insert operation.

query

To query a group, a VIS can use a multicast service such as Scribe [10] to contact all members of a group. If the group was configured without multicast support, the VIS can also use the basic DHT lookup operation, 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 Automatic Suggestions

Automatic suggestions of people and groups of potential interest 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.

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 groups by searching for 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 it 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 [29] to provide basic distributed hash table (DHT) functionality. We also use Scribe [10] 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 [29] 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 160-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 [32], CAN [25], or OpenDHT [27]. We chose Pastry because of the availability of a robust, open-source software base [19] upon which a number of higher-level applications and services have been built and deployed.

4.1.2 Scribe

Scribe [10] 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, all members are reliably notified of joins and leaves, and each member maintains a list of all other members.

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.

Leaving a Closed Group is not as simple as leaving an Open Group because all other members need 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.

Closed Group Node Failures: In a Closed Group, each member needs to maintain a list of other members. To ensure that each node has a consistent member list, 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.

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.

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 [31].
- Facebook has an open API that can be used for developing third-party applications [13].

5.1.1 Facebook Application

We developed a simple Facebook application named “Online Social Network Portrait” to extract basic Facebook parameters. 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 [28] to scramble 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 four months¹ and sampled 817 distinct groups and 60 users. Figures 2, 3, and 4 show cumulative distributions for the three characteristics of interest, while Table 1 shows summary statistics.

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

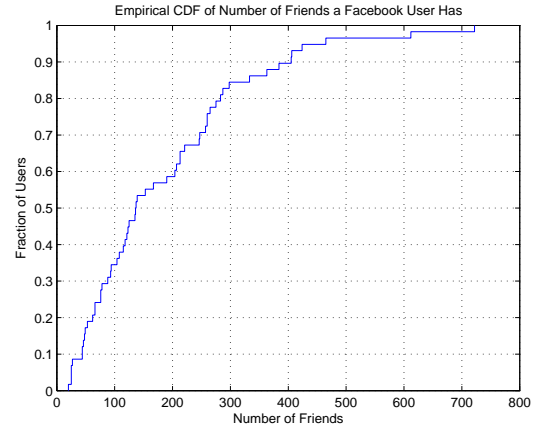


Figure 2: Empirical CDF of the number of friends a Facebook user has.

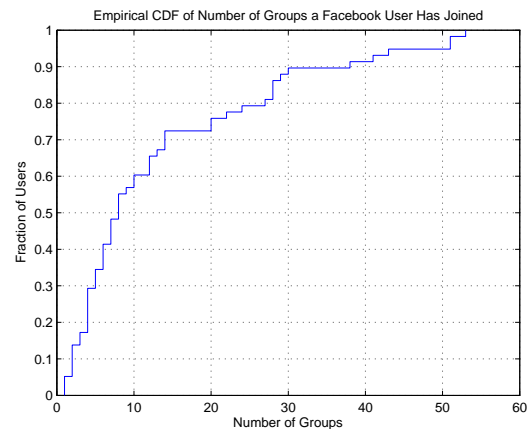


Figure 3: Empirical CDF of the number of groups a Facebook user has joined.

5.2 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 Vis-à-Vis and tested its correctness on several other sets of machines running the Xen virtual

	min	max	avg	std deviation
# of friends				
per user	20	772	186.66	148.87
# of groups				
per user	1	53	13.78	13.86
# of members				
per group	1	497	244.66	195.48

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

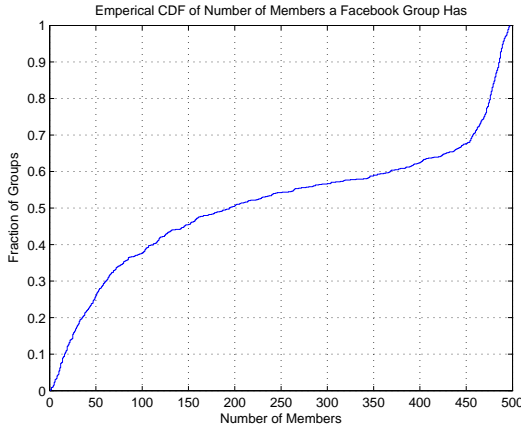


Figure 4: **Empirical CDF of the number of members in a Facebook group.**

machine monitor [6]. 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 Vis-à-Vis 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 [33] 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 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 [5] 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 Vis-à-Vis takes to complete OSN operations is important to the Vis-à-Vis user experience. At the same time, it is important to note that our latency requirements are relatively lax because we are dealing with social interactions. For example, it is acceptable for a group join operation to take many seconds to complete, even hours or days when human approval of a join request is involved. 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. Since in our Facebook measurements we determined that the number of members per group is only in the order of hundreds (see Table 1), 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.



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

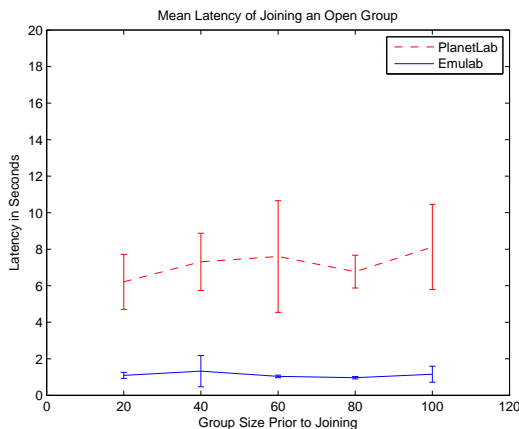


Figure 6: Mean latency of joining an Open Group in Emulab and PlanetLab. Error bars show the standard deviation.

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 and even with 100 nodes the latency is still less than 2 seconds. The PlanetLab results exhibit more variability due to the varying network characteristics and testbed loads discussed earlier. Moreover, the mean latency of joining an Open Group of size 100 is only at most 2 seconds longer than the mean latency of joining an Open Group of size 20. 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 requires 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. In both Emulab and PlanetLab experiments, the mean latency of joining a Closed Group of size 100 is at most 2 seconds longer than joining a Closed Group of size 20. 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.

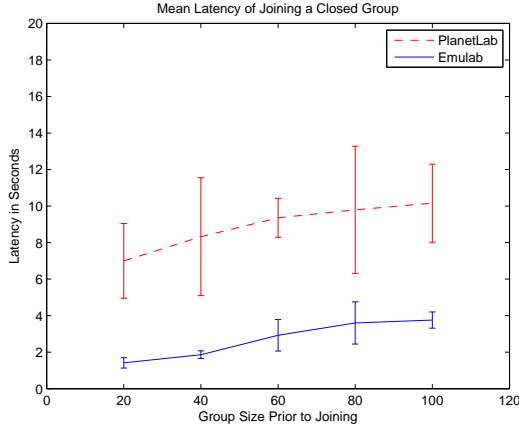


Figure 7: **Mean latency of joining a Closed Group in Emulab and PlanetLab. Error bars show the standard deviation.**

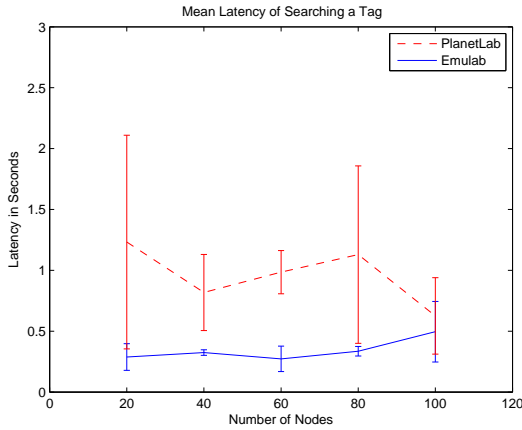


Figure 8: **Mean latency of searching for a data item in Emulab and PlanetLab. Error bars show the standard deviation.**

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. In both testbeds, the mean latency of searching for a data item is less than 1.5 seconds.

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 Vis-à-Vis-specific data structures. To quantify these

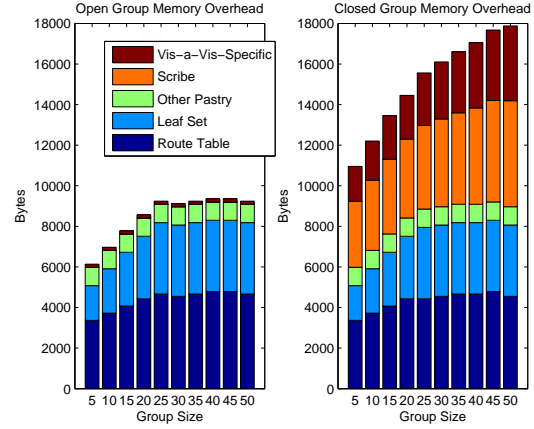


Figure 9: **Memory overhead per Virtual Individual Server to support Open and Closed Groups.**

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 than 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 Overhead

In order to maintain the DHT structures underlying Vis-à-Vis, VISs exchange regular heartbeat messages with the nodes in their Leaf Sets and Route Tables. The rate of these messages is configurable based on node availability, churn, and failure rate. In our prototype, we used the default heartbeat rates used by Pastry: 60 seconds for the Leaf Set and 900 seconds for the Route Table. The size of our Leaf Sets is 24 and the Route Tables contain 16 columns and 40 rows. Each heartbeat message is less

than 100 bytes long.

Based on the above, the maintenance traffic per group per node per day can be quantified using the following equation:

$$traffic = time \times size_{msg} \times \left(\frac{size_l}{r_l} + \frac{size_r}{r_r} \right)$$

$$traffic = 86400 \times 100 \times \left(\frac{24}{60} + \frac{640}{900} \right) = 9600000 \text{ bytes}$$

where $time$ is the total seconds in a day, $size_{msg}$ is the message size, $size_l$ is the size of the Leaf Set, $size_r$ is the size of the Route Table, r_l is the Leaf Set heartbeat rate and r_r is the Route Table heartbeat rate. Plugging into the equation the values mentioned previously yields that each node only generates and consumes 9.6MB of data per group per day. This amount of traffic is inexpensive to support. For example, Amazon EC2 currently charges \$0.10/GB for incoming traffic and \$0.17/GB for outgoing traffic, so the cost of maintenance traffic is only \$0.002592 per group per node per day. Furthermore, this value is the worst case because it assumes that the Leaf Set and Route Table’s entries are full. In a small group, most of the Route Table’s entries will be empty.

We expect to reduce the maintenance traffic overhead significantly through a combination of optimizations. First, we could reduce the rate of heartbeat messages well below the Pastry default values because of the high availability, low churn, and low failure rate of cloud-hosted VISs. Second, we could merge multiple Leaf Sets and Route Tables in a VIS into a Super Leaf Set and a Super Route Table to take advantage of having common members in multiple groups. Third, we could adopt SuperPastry [8], a hybrid of structured and unstructured overlays. SuperPastry significantly reduces churn rate and, therefore, maintenance traffic in the system. We leave these optimizations for future work.

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 and establish secure channels between them. 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 central idea of Vis-à-Vis.

NOYB [16] takes a different approach to the privacy threats of centralized OSNs by encrypting some of the data that users hand to the service. This approach is appealing because it may allow users of existing OSNs

to retain their existing profiles, while Vis-à-Vis requires users to divest themselves from existing OSNs. However, NOYB has several drawbacks compared 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.

Like NOYB, flyByNight [18] uses encryption to ensure that sensitive data is never posted to OSNs in unencrypted form. flyByNight is also appealing because it allows users to continue using existing OSNs and the social state that users have accumulated there. However, flyByNight remains vulnerable to several types of attack from within the OSN, which Vis-à-Vis avoids by doing away with centralized OSNs.

Turtle [23] is an anonymous peer-to-peer network designed for private information sharing among pre-existing social relations. All communication within Turtle is encrypted and search queries are broadcast to all “friend nodes”. Vis-à-Vis and Turtle share some motivations and design decisions. Namely, closed groups in both systems assume out-of-band mutual trust between nodes and every member of the closed group relies on all other members for privacy. Also, sensitive data is received only through friends, hence eliminating the “man-in-the-middle” attack. However, Turtle does not position itself as a social network service or use structured overlays such as DHTs. Therefore, it does not provide any means for forming groups, or for advertising and searching for topics of interest without flooding all members.

Seaweed is a scalable infrastructure for delay-aware querying of large and highly distributed datasets [22]. 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 the 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 for privacy issues and the latter because of bandwidth overhead.

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. Vis-à-Vis uses distributed hash tables to achieve scalable and efficient OSN operations on this large federation of machines. Most importantly, the decentralized nature of Vis-à-Vis 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 Elastic Compute Cloud (EC2). <http://aws.amazon.com/ec2/>.
- [2] Facebook. <http://www.facebook.com>.
- [3] LinkedIn. <http://www.linkedin.com>.
- [4] MySpace. <http://www.myspace.com>.
- [5] PlanetLab. <http://www.planet-lab.org/>.
- [6] 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 *Proc. of SOSP*, 2003.
- [7] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vTPM: Virtualizing the trusted platform module. In *Proc. of USENIX Security*, 2006.
- [8] M. Castro, M. Costa, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *Proc. of NSDI*, 2005.
- [9] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. of OSDI*, 2002.
- [10] M. Castro, P. Druschel, A. Marie Kermarrec, and A. Rowstron. Scalable application-level anycast for highly dynamic groups. In *Proc. of NGC*, 2003.
- [11] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.
- [12] J. R. Douceur. The sybil attack. In *Proc. of IPTPS*, 2002.
- [13] Facebook. Facebook Developers. <http://developers.facebook.com/>.
- [14] R. Figueiredo, O. Boykin, P. St. Juste, and D. Wolinsky. Social VPNs: Integrating overlay and social networks for seamless P2P networking. In *Proc. of COPS*, 2008.
- [15] M. Gjoka, M. Sirivianos, A. Markopoulou, and X. Yang. Poking Facebook: Characterization of OSN applications. In *Proc. of WOSN*, 2008.
- [16] S. Guha, K. Tang, and P. Francis. NOYB: Privacy in online social networks. In *Proc. of WOSN*, 2008.
- [17] Louise Story and Brad Stone. Facebook retreats on online tracking, November 2007. The New York Times.
- [18] M. M. Lucas and N. Borisov. flyByNight: Mitigating the privacy risks of social networking. In *Proc. of WPES*, 2008.
- [19] Max Plank Institute for Software Systems. FreePastry. <http://freepastry.org/FreePastry/>.
- [20] Megan McCarthy. How Facebook employees break into your profile, November 2007. <http://www.valleywag.com>.
- [21] J. Mickens and B. Noble. Exploiting availability prediction in distributed systems. In *Proc. of NSDI*, 2006.
- [22] D. Narayanan, A. Donnelly, R. Mortier, and A. Rowstron. Delay aware querying with seaweed. In *Proc. of VLDB*, 2006.
- [23] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and private data sharing with Turtle: Friends team-up and beat the system. In *Proc. of SPW*, 2004.
- [24] V. Ramasubramanian and E. G. Sirer. Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays. In *In NSDI*, pages 99–112, 2004.
- [25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proc. of SIGCOMM*, 2001.
- [26] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling churn in a DHT. In *Proc. of USENIX Annual Technical Conference*, 2004.
- [27] 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 *Proc. of SIGCOMM*, 2005.
- [28] R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Internet Activities Board, April 1992.
- [29] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. of Middleware*, 2001.
- [30] M. Satyanarayanan and et al. Pervasive personal computing in an Internet Suspend/Resume system. *IEEE Internet Computing*, 6(4), 2007.
- [31] 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/>.
- [32] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of SIGCOMM*, 2001.
- [33] 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 *Proc. of OSDI*, 2002.