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

Amre Shakimov*, Harold Lim*, Ramón Cáceres†, Landon P. Cox*,
Kevin Li†, Dongtao Liu*, and Alexander Varshavsky†

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

†AT&T Labs, Florham Park, NJ, USA
{ramon,kevinli,varshavsky}@research.att.com

***Abstract*—Online social networks (OSNs) are immensely popular, but their centralized control of user data raises important privacy concerns. This paper presents Vis-à-Vis, a decentralized framework for OSNs based on the privacy-preserving notion of a Virtual Individual Server (VIS). A VIS is a personal virtual machine running in a paid compute utility. In Vis-à-Vis, a person stores her data on her own VIS, which arbitrates access to that data by others. VISs self-organize into overlay networks corresponding to social groups. This paper focuses on preserving the privacy of location information. Vis-à-Vis uses distributed location trees to provide efficient and scalable operations for sharing location information within social groups. We have evaluated our Vis-à-Vis prototype using hundreds of virtual machines running in the Amazon EC2 compute utility. Our results demonstrate that Vis-à-Vis represents an attractive complement to today's centralized OSNs.**

I. Introduction

Free online social networks (OSNs) such as Facebook, Twitter, and Foursquare are central to the lives of millions of users and still growing. Facebook alone has over 500 million active users and 250 million unique visitors each day [1]. The volume of data handled by OSNs is staggering: Facebook receives more than 30 billion shared items every month [1], Twitter receives more than 55 million tweets each day [2], and Foursquare handled its 40-millionth check-in only five weeks after handling its 22-millionth [3].

At the same time, many recent incidents suggest that trusting free centralized services to safeguard sensitive OSN data may be unwise [4], [5], [6], [7], [8]. These examples underscore the risks inherent to the prevailing OSN model. First, concentrating the personal data of hundreds of millions of users under a single administrative domain leaves users vulnerable to large-scale privacy violations via inadvertent disclosures [4] and malicious attacks [5]. Second, providers offering free services must generate revenue by other means. OSN terms of service often reflect these incentives by giving the provider the right to reuse users' data in any way the provider sees fit [9]. These rights include sharing data with third-party advertisers without explicit consent from users [10].

Unsurprisingly, many people have grown wary of the OSN providers they depend on to protect their private information. In a survey of 2,253 adult OSN users, 65% had changed their privacy settings to limit what information they share with others, 36% had deleted comments from their profile, and 33% expressed concern over the amount of information about them online [11]. In a survey of young adults, 55% of 1,000 respondents reported being more concerned about privacy issues on the Internet than they were five years ago [12].

Given the importance of OSNs in users' lives and the sensitivity of the data users place in them, it is critical to limit the privacy risks posed by today's OSNs while preserving their features. To address this challenge, we have developed a general framework for managing privacy-sensitive OSN data called *Vis-à-Vis*. Vis-à-Vis can interoperate with existing OSNs and is organized as a federation of independent, personal *Virtual Individual Servers (VISs)*. A VIS is a virtual machine running in a paid cloud-computing utility such as Amazon Elastic Compute Cloud (EC2) or Rackspace Cloud Servers. Utilities provide better availability than desktop PCs and do not claim any rights to the content placed on their infrastructure [13]. Thus, just as cloud utilities are already trusted with many enterprises' intellectual property, utility-based VISs store their owner's sensitive data and arbitrate requests for that data by other parties.

In this paper, we focus on the rapidly growing challenge of preserving the privacy of location information within an OSN. Location-based OSNs utilize privacy-sensitive information about users' physical locations and are increasingly popular [14], [15], [16], [17]. For example, as of June, 2010, Foursquare had more than 1.5 million users and was expected to grow to 2 million users by July [18].

Vis-à-Vis supports location-based OSNs through a group abstraction that gives members control over how they share their location and allows them to query the locations of other members. Groups are administered by the users that created them using a range of admissions policies. For example, groups can be open, as are most of Facebook's "fan pages", restricted by social relationships such as "Alice's friends," or

978-1-4244-8953-4/11/$26.00

restricted by a set of credentials such as "The Duke Alumni Club of New York." Depending on a group's admission policy, members may wish to share their location at finer or coarser granularities. Prior studies have shown that users will typically disclose their full location or no location at all with close friends [19], but will utilize vaguer location descriptions when sharing their location information on public sites [20].

In addition, we aim for Vis-à-Vis groups to scale to thousands of members. While we expect most groups with which people share private location information will be limited to hundreds of members (e.g., the average Facebook user has 130 friends [1]), we want the flexibility to scale to much larger groups if the need arises (e.g., 23% of Facebook's fan pages have more than 1,000 members [21]).

To provide users with flexible control of their location information and to scale to groups with thousands of members, Vis-à-Vis organizes VISs into per-group overlay networks we call *location trees*. Within each tree, higher nodes represent coarser geographic regions such as countries while lower nodes represent finer regions such as city blocks. Interior nodes are chosen from the set of member VISs via a distributed consensus protocol. A user may publish her location to a group at an arbitrary granularity as long as her VIS becomes a leaf node within the subtree covering this location. Queries over a region are sent to the interior nodes covering that region and passed down the tree to lower nodes. Using this hierarchy, Vis-à-Vis guarantees that location queries complete in $O(\log(n))$ hops for groups of size n.

This paper makes the following contributions:

- It presents the design of a privacy-preserving framework for location-based OSNs based on hierarchical overlay networks of Virtual Individual Servers (Sections II and III).
- It describes an implementation of this framework, including a companion social application for mobile phones, that provides efficient and scalable OSN operations on distributed location data (Section IV) .
- It demonstrates the feasibility of our approach through performance experiments involving up to 500 virtual machines running in Amazon EC2, including machines distributed across two continents. We found that the latency of our decentralized system is competitive with that of a centralized implementation of the same OSN operations (Section V).

Despite this paper's focus on the sharing of geographic locations within large social groups, it should be clear that the Vis-à-Vis framework can be generalized to support other data types and social relationships, for example photographs shared among pairs of friends. In addition, Virtual Individual Servers can support many other applications besides online social networking, for example a personal synchronization service for mobile devices and a personal email service [22].

II. Location-based groups

The central abstraction supported by Vis-à-Vis is a *group*, as befits the central role played by social groups in OSNs. As a foundation, each principal in Vis-à-Vis is defined by a public-private key pair. *Users* are defined by a self-signed key pair. The private half of each user's key pair is stored securely by her VIS, allowing a VIS to act on her behalf. Users distribute their public key and the IP address of their VIS out of band (e.g., via email or an existing OSN such as Facebook).

Each group consists of an owner, a set of users defining the group's membership, and a mapping from group members to geographic regions. The group owner is the user who initiates and maintains the group. Each user within the group possesses a shared attribute such as an inter-personal relationship with the group owner or an interest in a particular topic. The geographic region associated with each group member is a geographic area the user wishes to share with other group members. Shared regions can be fine-grained or coarse-grained and can be set statically (e.g., hometown) or updated dynamically (e.g., current GPS coordinates).

Groups are named by a *descriptor* consisting of the group owner's public key and a string used to convey the attribute shared among group members. Descriptors can be expressed as a $\langle K^+_{owner}, string \rangle$ pair, where K^+_{owner} is the public key of the group owner. For example, a user, $Alice$, who is the president of the Duke Alumni Club of New York and works for AT&T might create groups $\langle K^+_{Alice}, DukeClubNYC \rangle$, and $\langle K^+_{Alice}, AT\&Tcoworkers \rangle$. Including the group owner's public key in each group descriptor allows users to manage the descriptor namespace independently and prevents naming conflicts. Descriptors do not contain privacy-sensitive information and, like a user's public key and VIS IP address, are distributed out of band. We imagine that descriptors will be embedded in web sites and users' existing OSN profiles.

Vis-à-Vis supports five operations on location-based groups under the following semantics:

- create(string, policy) : Creates a new, empty group using the public key of the caller and the passed-in string to construct the group descriptor. The policy argument allows group owners to define a call-back for implementing admission-policy plugins.
- join(descriptor, region, credential) : Adds the caller to the group with the specified descriptor, and, if successful, sets the region she shares with other group members. Success of the call depends on whether the credential passed in by the caller satisfies the admission policy for the group. Credentials can be empty for open groups, an email address from a specific domain such as duke.edu, or a signed social attestation [23] specifying a relationship between users.
- remove(descriptor) : Removes the caller from the group.
- update(descriptor, region) : Creates a mapping from the caller to a geographic region within a group. The caller must already be a group member for this operation to succeed.
- search(descriptor, region) : Returns a list of all group members (and their associated geographic regions) whose regions are contained within the passed-in region.

The update and search operations are general enough to support a wide range of options. Depending on how much detail users want to reveal about their location to other group members, they can post different regions to different groups at arbitrary granularities. For example, if a user is usually not

interested in being located by alumni of her alma mater, she could limit the region she shares with them to the city where she lives. However, in the fall, when she gathers with other fans to watch college football games, she may want to share the location of the bar where they usually meet. Such decisions can be made independently of how her location is represented in any other groups to which she might belong.

III. Architecture

To realize the Vis-à-Vis group abstraction, we organize users' VISs into the hierarchy shown in Figure 1. The Vis-à-Vis architecture is similar in many respects to the distributed tree structures utilized by Census [24] and P2PR-Tree [25]. We choose to use a hierarchical structure rather than a distributed hash table (DHT) because DHTs' simple get-set interface does not easily support the range queries required for the search operation. We initially considered using a distributed skip graph [26] to manage location information, but were unhappy with the overhead of resorting distributed lists whenever a user changed her location.

Users access location-based groups through *clients* such as stand-alone mobile applications and web browsers. Vis-à-Vis is designed to interoperate with, rather than replace, established OSNs such as Facebook. A combination of embedded descriptors, web-browser extensions, and OSN APIs such as Facebook Connect allow users to adopt Vis-à-Vis while retaining their significant investments in existing platforms. Existing OSNs are treated as untrusted services that store only opaque pointers to sensitive content stored in VISs.

For example, users can integrate a Vis-à-Vis-managed location-based group into a Facebook group by embedding a group descriptor in the Facebook group's information. When a user loads the group page in their web browser, a Vis-à-Vis browser extension interprets the document received from Facebook, identifies the group descriptor, and rewrites the page to include information downloaded from the appropriate VISs. Rendered pages can include a map displaying members' current locations, or UI components for updating a user's published location. These techniques for integrating externally managed information with existing OSNs are not new [23], [27], [28], [29]. Due to space constraints, we will not discuss the browser-side aspects of our architecture in any further detail. We present an example of a standalone mobile client in Section IV.

Clients interact with groups via their user's VIS. Each group supports a *membership service* responsible for implementing admission policies and for maintaining pointers to the group's *location tree*. The root and interior nodes of the location tree are *coordinator* VISs. In addition, each group member's VIS acts as a *leaf node* in the location tree.

The rest of this section describes Vis-à-Vis's trust-and-threat model, the function of each architectural component, and how these components collectively implement the group operations described in Section II.

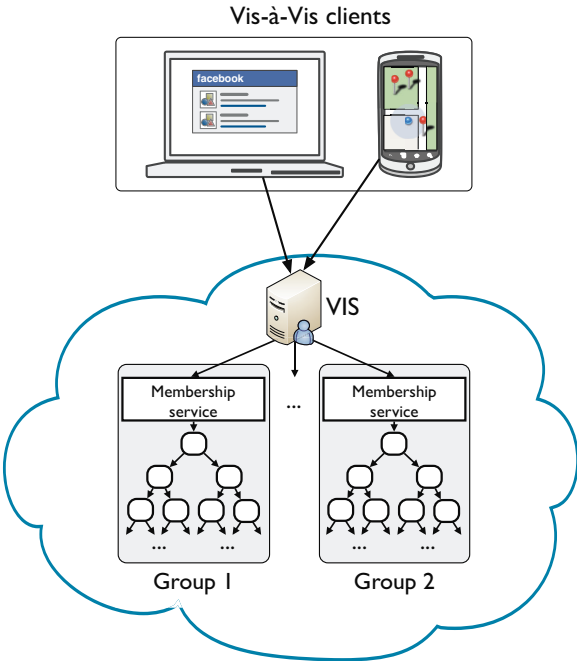

Fig. 1. Vis-à-Vis architectural overview.

A. Trust-and-threat model

Vis-à-Vis's decentralized structure provides users with stronger privacy protections than centralized services such as Facebook and MySpace, because it gives them more direct control over who has access to their personal data. Of course, Vis-à-Vis cannot eliminate breaches completely, and this section describes the classes of attacks that Vis-à-Vis is and is not designed to address.

Vis-à-Vis's trust model is based on the business interests of compute utilities and the 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. Unlike decentralized OSNs such as Persona [27], in which users do not trust their storage services, Vis-à-Vis exposes unencrypted data to the utilities hosting VISs.

There are three key benefits of entrusting compute utilities with access to cleartext data: 1) it simplifies key management, 2) it reduces the risk of large-scale data loss due to lost or corrupted state, and 3) most important for this paper, it allows computations such as range queries to be executed on remote storage servers, which is a key requirement for mobile services such as Loopt [14] and Google Latitude [15].

Since compute utilities control the hardware on which VISs execute, utility administrators can access all of a user's personal data. Despite this, Vis-à-Vis's assumption that compute utilities will not exercise this power is reasonable. Compute utilities' business model is based on selling computational resources to users rather than advertising to third parties. The legal language of utilities' user agreements formalize these limitations [13], providing economic and legal consequences if personal data is abused. In contrast, OSN terms of use usually grant the service provider a "non-exclusive, transferable,

sub-licensable, royalty-free, worldwide license to use any IP content that you post" [9]. We note that many companies already trust compute utilities with their intellectual property by hosting their computing infrastructure there.

In addition, 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 [30]. A TPM infrastructure within the cloud allows compute utilities to prove to their customers what software is executing under their account.

While such capabilities are rarely utilized at the moment, we believe that TPMs will be a commonly supported feature for 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 [31]. For Vis-à-Vis, an attested software stack will 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, or preferably providers of managed VIS services, must properly configure the access-control policies of their software, and install the latest available security patches. As with 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.

With this trust-and-threat model in mind, we now discuss the design of Vis-à-Vis in greater detail.

B. Membership service

A group's membership service is initially implemented by the group founder's VIS, although multiple VISs can participate in the membership service if the group grows. The primary function of the membership service is to provide a gateway to the location tree, in which group members' location information is maintained. New group members attempt to join a group through the membership service executing on the group owner's VIS. The owner's IP address is distributed out of band in the same manner as the owner's public key.

Access to the location tree can either be open to all requests or guarded, depending on the admission policy of the group. For example, mimicking the admission policies of Facebook's university networks, a group's membership service could require evidence of access to an email address within a specific domain. If the membership service receives sufficient evidence, it can issue a group capability in the form of a secret key. Subsequent requests among group members could be authenticated using this capability.

C. Location trees

Vis-à-Vis applies the distributed-systems techniques of *consensus, leases, and hierarchy* [32] to provide efficient, fault-tolerant, and scalable range queries over group members' location information. Group-specific location information in Vis-à-Vis is accessed through each group's *location tree*. A location tree is a hierarchical routing structure designed to efficiently and scalably support range queries over user locations. Unlike other data structures which use an arbitrary location region to partition a map, location trees use hierarchical political divisions. Since the divisions of a map are already known, the levels of the tree can be statically computed. Figure 2 shows an example tree. The top level represents countries, followed by states, counties, cities, and places. Leaf nodes represent users. In this example, user $U1$ is in place $P1$, in the city of Durham, within Durham County, in the state of North Carolina (NC), in the United States (US).

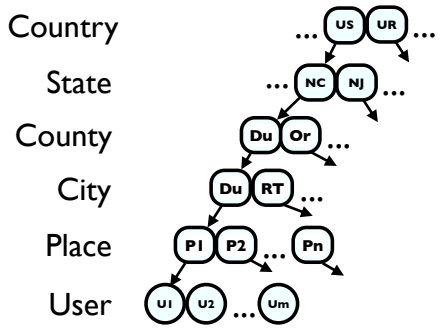

Fig. 2. User $U1$'s view of a group's location tree.

Each member's VIS occupies exactly one leaf node, regardless of the granularity at which she shares her location. The only constraint users face when inserting their VIS is that it must be a leaf node within the subtree covering its shared location. If a user's shared location is coarse, it may become a leaf node randomly below the corresponding interior node.

A potential danger of using static political divisions is that the finest-grained regions could be vulnerable to flash crowds. For example, places defining a sports arena would be unlikely to scale up to venue capacities of tens of thousands of users. To avoid such problems, Vis-à-Vis could easily apply techniques described by Census [24] and P2PR-Tree [25] for dynamically re-partitioning geographic regions into smaller subregions. Because of its hierarchical structure, any dynamic decomposition of geographic space would be hidden from higher levels of the tree.

1) Routing state: Each node maintains a subgraph of the tree, giving it partial knowledge of the group membership and members' locations. We have optimized for the expected common case of queries about nearby locations by having nodes store more information about closer VISs than farther VISs, thereby ensuring that queries complete in fewer hops on closer regions than farther regions. For example, in Figure 2, user $U1$ maintains a list of its siblings in $P1$ (i.e., $U2$ to Um) and their shared locations. Nodes also maintain a list of regional *coordinators* for each level of the tree along their path to the root. A coordinator is a VIS whose shared location is within the corresponding region.

Coordinators are identified through a distributed consensus

protocol such as Paxos [33]. The coordinators are elected by and from the pool of immediate child VISs. For example, in Figure 2, the coordinator for $P1$ is elected from the pool of $U1, U2, \ldots, Um$. Similarly, the coordinator for the city of Durham is elected by and from the coordinators of $P1, P2, \ldots Pn$, the coordinator for Durham county is elected by and from the coordinators of cities in Durham county, and so forth. A top-level coordinator serves at all levels of the tree.

In Figure 2, user $U1$ maintains a list of coordinator VISs for each of the following: places $P1$ to Pn in the city of Durham, all cities in Durham County, all counties in North Carolina, all states in the US, and all countries. To retrieve a list of user locations in Monmouth County, New Jersey (NJ), US, user $U1$ forwards a search request to the NJ coordinator. Because this VIS is sharing a location in NJ, it will have a pointer to the Monmouth County coordinator (if there are any users in this region), and forwards the search request to this VIS.

It is the responsibility of the Monmouth County coordinator to collect the list of users sharing their location within its county. The Monmouth coordinator has pointers to the coordinators for all cities in the county, which in turn have pointers to all places within those cities. Thus, the Monmouth coordinator forwards the search request to the coordinators for all cities in Monmouth, each of which forwards the request to the coordinators of all places in their cities. The place coordinators finally return lists of their leaf VISs and their shared locations to the Monmouth coordinator.

Unless the Monmouth coordinator knows the number of places in the tree that are populated, it cannot know when all results have been returned. One way to address this would be to use a time-out, such that the coordinator waits for a fixed period of time before returning an answer to a query. However, this would require coordinators to wait for that period on every request, even if all answers had been received. Instead, coordinators maintain extra information with their parents about the number of populated places below them in the tree. In our example, this would allow the Monmouth coordinator to know how many query responses to expect from the place coordinators. Once all of the responses are received, the Monmouth coordinator combines the search results and returns them to $U1$.

2) Tree maintenance: Because coordinators' identities are replicated within the routing state of all group members, it is important for members to have a consistent view of which VIS is a coordinator for which region, even as VISs leave the group or change locations. Vis-à-Vis maintains the consistency of this state using leases. VISs obtain a lease for their role as coordinator and periodically multicast lease-renewal messages down the tree as long as they continue to serve.

Coordinator failures are detected through explicit withdrawals (in the case of a user changing locations) or expiring leases (in the case of unplanned disconnections). Thus, in addition to maintaining a list of coordinators along the path to the root, each VIS must also maintain the lease expiry time for any coordinator they would be a candidate to replace. If a coordinator fails to renew their lease, a new election is held and election results are multicast down the tree. For example, in Figure 2, leaf nodes $U1, U2, \ldots, Um$ would maintain leasing state for the coordinator of $P1$, the coordinators for $P1, P2, \ldots, Pn$ would maintain leasing state for the coordinator of the city of Durham, and so forth.

Similarly, it is important for VISs sharing the same place coordinator to have a consistent view of their siblings. Thus, VISs also maintain expiry state for their siblings. Nodes periodically renew their lease with their siblings via multicast. If a sibling's lease expires without renewal, the sibling is assumed to have failed.

D. Operations

Vis-à-Vis groups implement each group operation—create, join, remove, update, and search—using the leasing and routing state maintained by VISs.

- create: To create a new group, a user distributes the group descriptor and the IP address of her VIS as needed. The owner's VIS provides the membership service for the group, which is similar to Facebook group founders being automatically made group administrators.
- join: To join a group, a VIS contacts the group's membership service and asks for the addresses of the top-level coordinators. If admitted into the group, the VIS then uses these lop-level hosts to recursively identify the coordinator of the place where it wishes to become a leaf node.
  If the coordinator for the place exists, the new VIS notifies the coordinator of its presence. The coordinator then multicasts the IP address and shared location of the new VIS to other VISs in the region, who forward their information to the joining node. On the other hand, if the place coordinator or any coordinators along the path to the root do not exist, the joining VIS elects itself the coordinator of these previously unpopulated regions. Notification of these updates are forwarded to the appropriate regions.
- remove: To remove itself from a group, a VIS sends a purge message to its sibling VISs, removing it from their routing state. A VIS can also remove itself by allowing its leases to expire.
- update: If a location update does not change a VIS's place, the VIS simply multicasts its new location to its siblings as part of its lease-renewal messages. If a location update requires moving to a new region, the node purges itself from its siblings' routing state (either explicitly or via an expired lease), looks up the coordinator for the new region, and notifies the new coordinator of its location.
- search: Search is performed in two stages. First, a VIS decomposes the requested bounding-box region into the smallest set of covering subregions. For each subregion in this set, the VIS looks up the coordinator, and if the coordinator exists, sends it a search request. Search requests received by coordinators are satisfied using the recursive procedure described in Section III-C1.

IV. Implementation

We built a Vis-à-Vis prototype based on the design described in Section III and deployed it on Amazon EC2. We also modified Apache ZooKeeper to support namespace partitioning. Finally, we created a mobile phone application called Group Finder to test and evaluate our Vis-à-Vis prototype. This section describes these software implementations.

A. Vis-a-Vis and ZooKeeper

Our prototype Vis-à-Vis implementation is written in Java and consists of over 3,300 semi-colon terminated lines of code and 50 class files.

Software for maintaining location-tree state is a multi-threaded application that maintains its position in the tree by exchanging heartbeats with other nodes and participating in coordinator elections. This software also serves incoming requests from the client, e.g., update the shared location, leave or join the tree. Our implementation supports both IP- and application-level multicast. However, since Amazon EC2 does not support IP multicast, we use application-level multicast for the experiments reported in Section V.

We use ZooKeeper [34] for our coordinator election service. In our Vis-à-Vis prototype, the ZooKeeper server runs on the group founder's VIS. However, ZooKeeper supports clustering and can be run on multiple VISs. ZooKeeper includes a leader election function based on Paxos [33], which we use for our coordinator election. We have modified ZooKeeper to support optional namespace partitioning. This allows us to span the coordinator election service across multiple datacenters, improving performance and reducing the number of expensive roundtrips across long distances. For example, if the tree consists of a number of VISs in Europe and North America, then it is beneficial to use two ZooKeeper instances: one in Europe and another in North America. Each ZooKeeper instance is responsible for a particular namespace, e.g., *EU* or *US*, and serves requests from the same continent. However, if a European VIS wants to join a location in the US, the EU-based ZooKeeper would redirect this VIS to a US-based ZooKeeper instance.

Finally, each VIS runs a Tomcat server. Low-level Java Servlet APIs provide an interface that supports the group operations described in Section II. High-level APIs provide application-specific operations. These APIs only accept requests from the owner of the VIS.

B. Group Finder mobile application

We also developed a companion iPhone application for Vis-à-Vis called Group Finder. The application allows a user to submit her location along with a status message to a group she belongs to, and retrieve the last known locations and status messages of the other group members. We refer to the operation of submitting a location and status message to a server as a "check-in".

A screenshot of Group Finder is shown in Figure 3. The current location of the user is shown as a blue dot and the most recent check-in locations of the group members as pins.

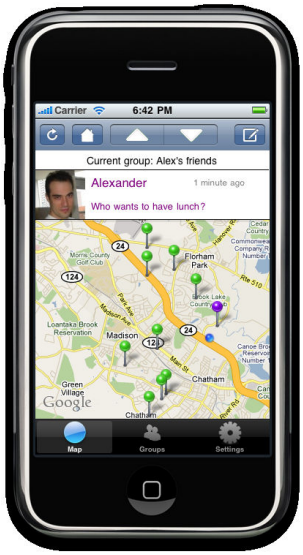

Fig. 3. Screenshot of the Group Finder application.

Selecting a pin shows the corresponding group member's photo, his last status message, and the time since his last update. The buttons at the top of the screen allow the user to check in or retrieve information about the latest check-ins from group members. Just below the buttons, Group Finder displays the name of the currently selected group.

In our implementation, each user's mobile phone communicates only with that user's VIS. Location updates and status messages are shared only within the current group. Retrieving the latest check-in locations of group members is implemented in the VIS as a call to the search operation, with the location bounds equal to the map area shown on the screen. Check-ins invoke the update operation.

We used Group Finder to debug the Vis-à-Vis framework and measure end-to-end latencies over 3G and WiFi networks. We report on these findings in Section V.

V. Evaluation

In our experimental evaluation of Vis-à-Vis, we sought answers to the following three questions:

- How well does our Vis-à-Vis prototype perform the core group operations of join, update, and search?
- How well does our Vis-à-Vis prototype perform when the nodes are geographically spread across continents?
- How well does our Group Finder application perform when communicating with Vis-à-Vis over WiFi and 3G networks?

We wanted to characterize the performance of our decentralized approach to managing information in location-based OSNs. For comparison, we also implemented the Vis-à-Vis group abstraction using a centralized approach. The centralized service is a conventional multi-tiered architecture, consisting of a front-end web-application server (Tomcat server) and a back-end database server (MySQL server). We expected the

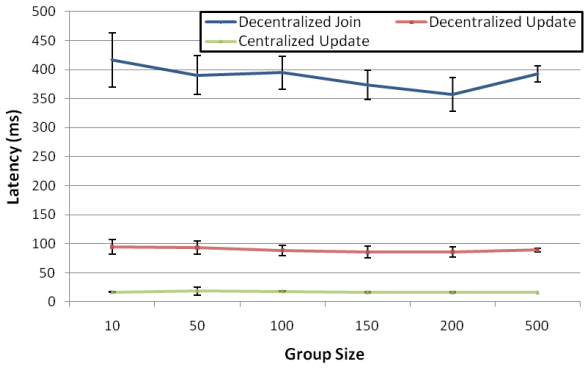

Fig. 4. join and update performance.

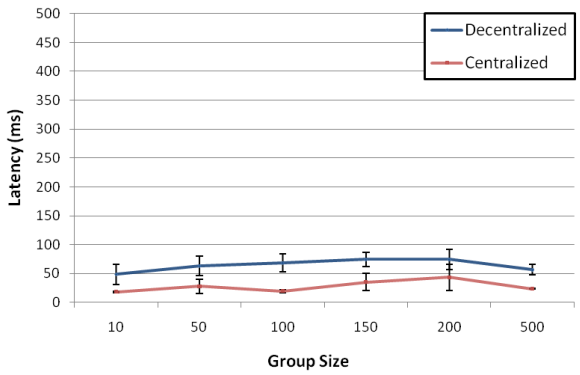

Fig. 5. Local search performance.

centralized server to provide lower latency than our decentralized Vis-à-Vis prototype, but wanted to determine whether our prototype's performance was competitive.

We ran micro-benchmarks, studied the effect of geographic distribution of VISs, and measured end-to-end latencies using our Group Finder application. All experiments used EC2 virtual machines as VISs. Each virtual machine was configured as an EC2 Small Instance, with one virtual core and 1.7 GB of memory. Our centralized service ran on the same type of virtual machine in the same data center.

The location tree for all experiments was based on our Group Finder app, which uses an 8×8 grid for its finest-grained locations. The resulting location tree had four levels (including leaf nodes) and four coordinators per level.

A. *Micro-benchmarks*

For our micro-benchmarks, we wanted to measure the latency from an external host to complete the join, update, and search operations in both Vis-à-Vis and our centralized implementation. We measured latency at a wired host at Duke from the time an operation request was issued to the time it completed. VISs were hosted within the same Amazon data center on the east coast of the US, where the round-trip latency between machines varied between 0.1 and 5ms.

For these experiments, we varied the group size from 10 to 500 members, and assigned each member a separate VIS. VISs inserted themselves into the tree as randomly-placed leaf nodes. For each experiment, we report the mean time over 20 trials to complete each operation, including network, DNS, and server latency. In all figures, error bars denote standard deviations. Due to occasional, transient spikes in the time to complete DNS lookups, we did not include some outliers in our micro-benchmark results. The vast majority of DNS lookups completed within tens of milliseconds, but occasionally lookups took hundreds of milliseconds or timed out altogether. We attribute these spikes to well documented problems specific to Amazon EC2 [35], but since the spikes were rare we did not thoroughly examine them. Thus, when network latency was more than an order of magnitude higher than normal, we removed 1) these high-latency trials, and 2) an equal number of the lowest-latency trials.

Figure 4 shows the average latency to complete join and update operations for our Vis-à-Vis prototype and our centralized implementation. In the centralized case, join and update are identical: both essentially insert a new value into the MySQL database. For Vis-à-Vis, join is more expensive than update because the VIS must initialize its routing state before registering its own location with a coordinator.

As expected, the centralized implementation had lower latencies than Vis-à-Vis, with update operations completing in approximately 20ms for all group sizes. Nonetheless, our decentralized Vis-à-Vis implementation performs reasonably well, with join operations completing in approximately 400ms and update operations completing in under 100ms for all group sizes. These results demonstrate two important properties of Vis-à-Vis: 1) that join and update provide reasonable efficiency, even though the centralized case is faster, and 2) that join and update scale well to large group sizes.

To understand search performance, we investigated two cases. In the first case, we performed local search operations within a single fine-grained region. These searches only required communication with one coordinator VIS. In the second case, we performed search operations across the entire group so that the locations of all group members were retrieved. These searches required contacting all coordinators in the tree and represent a worst case for Vis-à-Vis.

Figure 5 shows the average latency to perform a local search for the centralized implementation and Vis-à-Vis. As expected, both perform well, returning results in under 100ms and easily scaling to 500-member groups. Recall that VISs inserted themselves randomly into the tree, so that in the case of a 500-member group, low-level coordinators returned an average of 8 locations per query.

Figure 6 shows the average latency to perform a group-wide search for the centralized implementation and Vis-à-Vis. Again as expected, Vis-à-Vis's decentralized approach has higher latency than the centralized case for group-wide searches since queries require communicating with multiple coordinators. However, Vis-à-Vis's latency plateaus at around 200ms for all groups larger than 100, while the centralized approach experiences increased latency for 500-member groups.

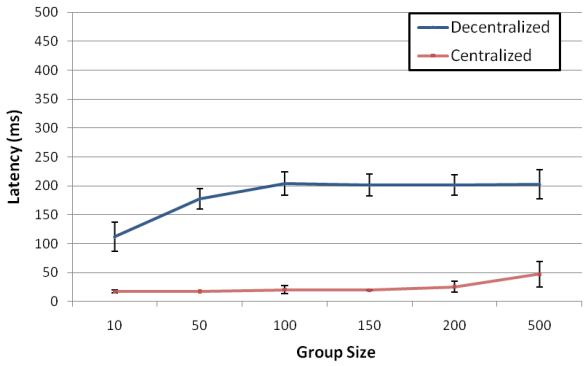

Fig. 6. Group-wide search performance.

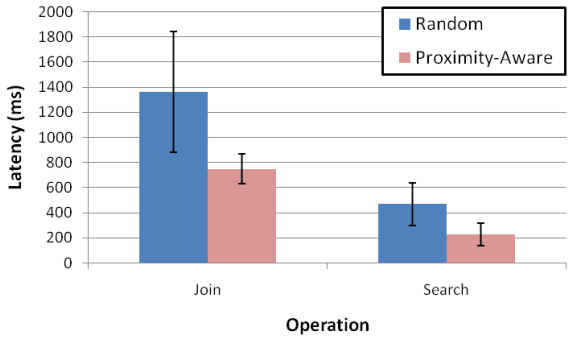

Fig. 7. Effect of geographic distribution of VISs.

Like the join and update micro-benchmarks, these results demonstrate 1) that Vis-à-Vis's performance is reasonable, even when compared to a fast centralized implementation, and 2) that Vis-à-Vis scales well, even in the worst case when all coordinators must be contacted.

Note that our maximum group size of 500 corresponds to the majority of Facebook groups, even though many are much larger [21]. We would have liked to generate results for groups of 1,000 or more VISs, but could not due to EC2 limitations. Nonetheless, given the results of our micro-benchmarks, we see no reason why even our unoptimized prototype would not scale to tens of thousands of nodes.

B. Effect of geographic distribution of VISs

To study the effect of geographic distribution of VISs on Vis-à-Vis, we built a location tree with nodes hosted at two Amazon EC2 data centers located in distant geographic locations: 50 nodes in the US and 50 nodes in the European Union (EU), specifically Ireland. We left the group's membership service in the US zone, and ran one ZooKeeper instance in each geographic zone with a partitioned namespace. We used the same client machine at Duke as in Section V-A. The round-trip latency between the US- and EU-based nodes varied between 85 and 95ms.

We compared two different methods of constructing the tree. The first is a *random assignment*, where a VIS joins a random location. This method is expected to have poor performance since an EU-based VIS has to contact a US-based ZooKeeper instance when joining a US-based location, and possibly a number of US-based coordinators. The second method is *proximity-aware assignment*. This scenario assumes that US- and EU-based VISs are more likely to join a ZooKeeper instance near their respective published locations. This assignment should have better performance since a EU-based VIS will mostly interact with EU-based servers. However, both of these methods incur unavoidable overhead from the network latencies between the US-based client machine and the EU-based VISs, and between the EU-based VISs and the US-based group's membership service.

Figure 7 shows the latencies of the *join* and *search* operations on a cross-continental 100-node tree with 50 nodes in Europe and 50 nodes in North America. In the case of random assignment, we measured the latencies of an EU-based VIS joining a random (EU or US) location of the tree. For the proximity-aware assignment, we measured the latency of a EU-based VIS joining an EU location of the tree. For both scenarios we measured the local search operation performed by an EU-based VIS.

As expected, latencies using the random assignment method are longer than those using the proximity-aware method. However, even the shorter latencies are longer than those reported in Section V-A due to the unavoidable overhead described above.

C. End-to-end latency

The goal of our final set of experiments was to measure the end-to-end latency of a mobile application using Vis-à-Vis. These experiments were performed at Duke using our Group Finder iPhone application. We instrumented the Group Finder and server code to measure the latencies of checking in and retrieving group members' locations. We measured the network and server latency to complete these tasks while varying the following parameters: network type (WiFi or 3G cellular), group size (10 or 100 members), and architecture (Vis-à-Vis or a centralized server).

During a check-in, the user's client uploaded a location via an update call to a server: her VIS in the Vis-à-Vis case and the centralized server in the other. For Vis-à-Vis experiments, the user's VIS synchronously translated the user's location to the correct coordinator, notified the coordinator of the user's new location, then returned control to the mobile device. A user retrieved group members' locations through a search call. In Vis-à-Vis this call propagated queries down the location tree, and in the centralized case it led to a MySQL query.

As with the micro-benchmarks, we report the mean and standard deviation of latencies across 20 trials. However, unlike with the micro-benchmarks, we do not remove outliers for our end-to-end results. This is because spikes in 3G network latency are common and removing outliers from the wireless experiments would have given an inaccurate view of Vis-à-Vis's performance.

Group Size	Latency Component	Decentralized			Centralized	
		search	update (far away)	update (nearby)	search	update
10	Network	169 (98)	365 (97)	295 (225)	211 (57)	263 (177)
	Server	36 (14)	297 (37)	14 (15)	10 (3)	8 (1)
	Total	205	662	309	221	271
100	Network	376 (166)	322 (57)	362 (89)	311 (87)	287 (37)
	Server	160 (28)	295 (40)	10 (9)	68 (50)	9 (3)
	Total	536	617	372	379	296

TABLE I
GROUP FINDER MEAN LATENCY OVER WIFI IN MILLISECONDS, WITH STANDARD DEVIATIONS IN PARENTHESES.

Group Size	Latency Component	Decentralized			Centralized	
		search	update (far away)	update (nearby)	search	update
10	Network	870 (918)	3004 (3678)	1673 (1104)	1103 (905)	1280 (1845)
	Server	44 (12)	438 (184)	27 (15)	7 (1)	7 (1)
	Total	914	3442	1700	1110	1287
100	Network	2812 (3428)	3873 (6003)	2294 (3098)	1940 (1263)	1363 (1160)
	Server	155 (41)	385 (71)	28 (17)	8 (6)	31 (33)
	Total	2967	4258	2322	1948	1394

TABLE II
GROUP FINDER MEAN LATENCY OVER 3G CELLULAR IN MILLISECONDS, WITH STANDARD DEVIATIONS IN PARENTHESES.

Table I shows the time for Group Finder to retrieve the locations of a group's members (i.e., to complete a search operation) over WiFi, broken down by network and server latency. The increased server latency for 100 group members in the decentralized case reflects the need to communicate with more coordinators. The server latency is comparable to our search micro-benchmarks for groups with 100 members.

Table I also shows the time for a Group Finder user to check-in (i.e., to complete an update operation) over WiFi. For the decentralized setup, we measured the time to check-in to a region that is far away, which required contacting several coordinators, and the time to check-in to a nearby region, which required contacting a single coordinator. As with our micro-benchmarks, group size had little effect on search latency. Also as expected, checking in to a nearby location was faster than checking in to a far-away location.

These results demonstrate that the performance of common tasks in Group Finder, such as retrieving members' locations and checking in to a nearby location, are dominated by network latency rather than server latency. Only in the uncommon case when a user checks in to a far away region would server latency approach network latency under WiFi.

Table II shows the latency of the same Group Finder tasks using the iPhone's 3G cellular connection. These results show that 3G network latency was often an order of magnitude greater than that of WiFi. In addition, standard deviations for 3G latency were often greater than the means themselves. As a result, server latency was small relative to network latency in all our 3G experiments.

Overall, our end-to-end experiments demonstrate that for mobile applications such as Group Finder, performance is often dominated by the latency of wireless networks rather than that of Vis-à-Vis's back-end services. This effect reduces the perceived performance differences between Vis-à-Vis and centralized OSNs.

VI. RELATED WORK

The importance of protecting privacy in OSNs has attracted significant attention from the research community. This section compares Vis-à-Vis to the most relevant related work.

We have published two short workshop papers on Virtual Individual Servers and Vis-à-Vis [22], [36]. That early work led us to redesign our core data structures and algorithms to improve performance. More specifically, we replaced distributed hash tables and skip graphs with the location trees presented here. We then reimplemented our system according to the new design and built a new location-based social application on top of Vis-à-Vis. Finally, we did a larger and more thorough evaluation of our system using EC2 instead of PlanetLab and Emulab.

We have also developed Confidant [29], a decentralized OSN based on personal computers residing in homes or offices. Confidant focuses on a socially-informed replication scheme to improve on the limited availability of such machines. In contrast, Vis-à-Vis is based on highly available VISs that do not require replication.

Most of the proposed decentralized OSNs, such as Persona [27], NOYB [28], flyByNight [37], PeerSoN [38], and others [39], assume that the underlying storage service responsible for holding users' personal data is not trustworthy. Puttaswamy protected users' location information under this assumption as well [40]. Vis-à-Vis represents a philosophical departure by entrusting compute utilities such as EC2 with access to unencrypted versions of this data. As explained in Section III-A, we feel that trusting compute utilities is warranted given their business models and terms of use. As TPM-enabled services are more widely embraced by utilities [30], the trustworthiness of users' VISs will increase. Furthermore, Vis-à-Vis leverages the trust it places in compute utilities and the VISs they host to provide services such as range queries

over location data that are not possible when servers hold encrypted data.

Cutillo et al. [41] proposed a peer-to-peer OSN scheme that leverages trust based on the social relationships among friends and acquaintances to replicate profile information and anonymize traffic. In contrast, Vis-à-Vis is designed to support flexible degrees of location sharing among large groups of users, possibly in the absence of strong social ties.

Finally, the hierarchical organization of Vis-à-Vis shares many traits with both P2P-RTree [25] and Census [24], although neither is focused on OSN privacy. P2P-RTree is a spatial index designed for peer-to-peer networks. The subset of location-tree information maintained by VISs in Vis-à-Vis is very similar to the information stored by peers in P2P-RTree, but does not provide any fault-tolerance mechanisms or consistency guarantees.

Census [24] is a platform for building large-scale distributed applications that provides a consistent group membership abstraction for geographically dispersed nodes. Census allows the entire membership to be replicated at all nodes, and tightly integrates a redundant multicast layer for delivering membership updates efficiently in the presence of failures. Vis-à-Vis also uses leases and multicast to maintain consistent views of the membership among participating VISs, but does not require the entire tree to be replicated at each node because users are likely to be involved with many groups.

VII. Conclusion

We have presented the design, implementation, and evaluation of Vis-à-Vis, a decentralized framework for online social networks. Vis-à-Vis is based on a federation of independent Virtual Individual Servers, machines owned by individuals and preferably running in a paid, virtualized cloud-computing infrastructure. Vis-à-Vis preserves privacy by avoiding the pitfalls of the prevailing OSN model based on centralized free services. We focused on Vis-à-Vis's use of distributed hierarchies to provide efficient and scalable location-based operations on social groups. We deployed a Vis-à-Vis prototype in Amazon EC2 and measured its performance against a centralized implementation of the same OSN operations. Our results show that the latency of our decentralized system is competitive with that of its centralized counterpart.

Acknowledgements

The work by the co-authors from Duke University was supported by the National Science Foundation under award CNS-0916649, as well as by AT&T Labs and Amazon.

References

[1] "Facebook statistics," http://www.facebook.com/press/.
[2] Business Insider, "Twitter finally reveals all its secret stats," April 2010.
[3] Mashable, "Foursquare exceeds 40 million checkins," May 2010.
[4] ArsTechnica, "EPIC fail: Google faces FTC complaint over Buzz privacy," February 2010.
[5] ArsTechnica, "Twitter gets government warning over 2009 security breaches," June 2010.
[6] Associated Press / MSNBC, "Facebook shuts down beacon marketing tool," September 2009.
[7] ArsTechnica, "Are "deleted" photos really gone from Facebook? Not always," July 2009.
[8] ArsTechnica, "Creepy insurance company pulls coverage due to Facebook pics," November 2009.
[9] Facebook, "Statement of rights and responsibilities," http://www.facebook.com/terms.php.
[10] TechCrunch, "Senators Call Out Facebook On Instant Personalization, Other Privacy Issues," April 2010.
[11] M. Madden and A. Smith, "Reputation management and social media," May 2010, http://www.pewinternet.org/Reports/2010/.
[12] C. Hoofnagle, J. King, S. Li, and J. Turow, "How different are young adults from older adults when it comes to information privacy attitudes & policies," April 2010, http://ssrn.com/abstract=1589864.
[13] "Amazon Elastic Compute Cloud (EC2)," http://aws.amazon.com/ec2/.
[14] "Loopt," http://www.loopt.com.
[15] "Google latitude," http://www.google.com/latitude.
[16] "Gowalla," http://www.gowalla.com.
[17] TechCrunch, "Twitter Turns On Location. Not For Twitter.com Just Yet." November 2009.
[18] TechCrunch, "Foursquare now adding nearly 100,000 users a week," June 2010.
[19] S. Consolvo and *et al*, "Location disclosure to social relations: why, when, & what people want to share," in *CHI '05*, 2005.
[20] S. Ahern and *et al*, "Over-exposed?: privacy patterns and considerations in online and mobile photo sharing," in *CHI '07*, 2007.
[21] TechCrunch, "It's Not Easy Being Popular. 77 Percent Of Facebook Fan Pages Have Under 1,000 Fans," November 2009.
[22] R. Cáceres and *et al*, "Virtual individual servers as privacy-preserving proxies for mobile devices," in *MobiHeld '09*, 2009.
[23] A. Tootoonchian, S. Saroiu, Y. Ganjali, and A. Wolman, "Lockr: better privacy for social networks," in *CoNEXT '09*, 2009.
[24] J. Cowling, D. R. K. Ports, B. Liskov, R. A. Popa, and A. Gaikwad, "Census: Location-aware membership management for large-scale distributed systems," in *USENIX '09*, 2009.
[25] A. Mondal, Y. Lifu, and M. Kitsuregawa, "P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments," in *EDBT Workshop on P2P and Databases*, 2004.
[26] N. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "Skipnet: A scalable overlay network with practical locality properties," in *USITS '03*, 2003.
[27] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona: an online social network with user-defined privacy," in *SIGCOMM '09*, 2009.
[28] S. Guha, K. Tang, and P. Francis, "NOYB: Privacy in online social networks," in *WOSN '08*, 2008.
[29] D. Liu and *et al*, "Confidant: Protecting OSN Data without Locking It Up," May 2010, Duke University Technical Report TR-2010-04, submitted for publication.
[30] N. Santos, K. P. Gummadi, and R. Ridrigues, "Towards trusted cloud computing," in *HotCloud '09*, 2009.
[31] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," in *OSDI '02*, 2002.
[32] B. W. Lampson, "How to build a highly available system using consensus," in *WDAG 96*, 1996.
[33] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, vol. 16, no. 2, pp. 133–169, 1998.
[34] "ZooKeeper," http://hadoop.apache.org/zookeeper.
[35] G. Wang and T. S. E. Ng, "The Impact of Virtualization on Network Performance of Amazon EC2 Data Center," in *IEEE INFOCOM*, 2010.
[36] A. Shakimov and *et al*, "Privacy, cost, and availability tradeoffs in decentralized OSNs," in *WOSN '09*, 2009.
[37] M. Lucas and N. Borisov, "flybynight: mitigating the privacy risks of social networking," in *SOUPS '09*, 2009.
[38] S. Buchegger, D. Schiöberg, L. H. Vu, and A. Datta, "PeerSon: P2P social networking - early experiences and insights," in *SocialNets '09*, 2009.
[39] J. Anderson, C. Diaz, J. Bonneau, and F. Stajano, "Privacy Preserving Social Networking Over Untrusted Networks," in *WOSN'09*, 2009.
[40] K. P. N. Puttaswamy and B. Y. Zhao, "Preserving privacy in location-based mobile social applications," in *Hotmobile'10*, 2010.
[41] L. A. Cutillo, R. Molva, and T. Strufe, "Privacy preserving social networking through decentralization," in *WONS '09*, 2009.