Privacy, Availability and Economics in the Polaris Mobile Social Network

Christo Wilson, Troy Steinbauer, Gang Wang, Alessandra Sala, Haitao Zheng and Ben Y. Zhao
Department of Computer Science, U. C. Santa Barbara, Santa Barbara, CA USA
{bowlin, troysteinbauer, gangw, alessandra, htzheng, ravenben}@cs.ucsb.edu

ABSTRACT

While highly successful, today's online social networks (OSNs) have made a conscious decision to sacrifice privacy for availability and centralized control. Unfortunately, tradeoffs in this "walled garden" architecture naturally pit the economic interests of OSN providers against the privacy goals of OSN users, a battle that users cannot win. While some alternative OSN designs preserve user control over data, they do so by de-prioritizing issues of economic incentives and sustainability. In contrast, we believe any practical alternative to today's centralized architecture must consider incentives for providers as a key goal. In this paper, we propose a distributed OSN architecture that significantly improves user privacy while preserving economic incentives for OSN providers. We do so by using a standardized API to create a competitive provider marketplace for different components of the OSN, thus allowing users to perform their own tradeoffs between cost, performance, and privacy. We describe *Polaris*, a system where users leverage smartphones as a highly available identity provider and access control manager, and use application prototypes to show how it allows data monetization while limiting the visibility of any single party to users' private data.

1. INTRODUCTION

Online social networks (OSNs) such as Facebook and LinkedIn have rapidly evolved from messaging systems for teenagers, to indispensable tools for communication and collaboration for businesses and personal users of all types. Today, account membership in one or more of the major OSNs is no longer optional.

Moving forward, the long-term implications on user privacy are profound. Current OSNs found success in a "walled garden" architecture, where all user data is managed by a single, trusted OSN provider. This centralized model makes an explicit tradeoff that guarantees data availability and viable monetary incentives for OSN providers by sacrificing user control over their personal data. More importantly, tradeoffs in this architecture naturally pit the economic interests of OSN providers against the privacy concerns of their users, *i.e.* providers generate revenue by mining user data for advertisement placement, which is impossible if users intentionally hide their data from providers via encryption [2, 14]. With providers holding leverage over users' data and relationships, this is a battle that users cannot win.

But is this tension necessary? Researchers are studying alternatives to the "walled garden" model in the form of distributed social networks, where each user manages her own data either locally on her machine, or on cloud-based storage. These systems all strive for total privacy by leveraging end-to-end encryption of user data. These designs effectively improve privacy and maintain availability by replacing the economic role of OSN providers with delegated monetary and management costs to the user.

While potentially viable from a technical standpoint, both approaches face serious questions of economic viability and sustainability. First, some designs assume that users can host their own profile data on home servers, and try their best to maintain availability via replication [4]. The same approach was proposed for numerous peer-to-peer storage systems, and has met with little success [3]. For example, most casual Facebook users will not have a highly available home server for their OSN content. A second approach calls for the user to pay for her own virtual machine stored and managed by a cloud provider [12]. While this simplifies the availability problem, significant evidence shows that most OSN users are unwilling to pay for what many consider a free service [5]. A "We Will Not Pay To Use Facebook" group on Facebook grew to more than 2 million members before disappearing.

In this paper, we explore the question: is there an architecture that occupies the middle ground between these two extremes? One that balances the monetary needs of OSN providers with users' need for control over their data, all while providing high data availability? We describe *Polaris*, an architecture for OSNs that preserves monetary incentives for OSN providers to store and manage user data, while also mitigating the systemic privacy concerns associated with monolithic OSNs. To accomplish these goals, Polaris leverages two observations:

- Mobile clients are becoming the dominant platform for web browsing and accessing OSNs. "Smartphones" equipped with significant storage (8+GB), computation (1Ghz CPU) resources, and 3G Internet connectivity will soon dominate the cellphone market [10]. Since they are always-on, and constantly within arm's reach, these devices represent the ideal interface for hosting and managing a user's social identity.
- Recent proposals to architect and standardize open social platforms demonstrate that third-party services can and will compete for sustainable revenue in the OSN market. Using open standards and APIs such as OpenID [8], OAuth [6] and OStatus [11], users can replace centralized, monolithic OSN providers with a collection of *commoditized OSN services* in the cloud, all competing to provide standardized OSN services to the user.

HotMobile'11, March 1–2, 2011, Phoenix, Arizona.

Polaris uses a distributed OSN model, where functionality is divided into different application domains, *e.g.* status updates and photo sharing. For each domain, a user can choose from competing OSN service providers, *e.g.* Picasa, Flickr, and PhotoBucket. Polaris stores sensitive data, like personal profile, on the mobile device [1], but allows Polaris-enabled service providers to host data specific to their application. Providers implement a common API that enables users to connect with friends and share social data irrespective of each individual user's choice of providers. Through an application on the mobile device, users can update their data and specify policies that define how providers should share their data with other users. Finally, the mobile device can also act as secure storage for data deemed highly sensitive by the user.

Key Benefits. Polaris leverages this architecture to provide three key benefits. First, Polaris provides improved privacy compared to current centralized OSNs. Highly sensitive data is stored only on the mobile device, and access is protected by the Polaris users' own privacy policies. Less sensitive social data is hosted and managed by application-specific providers chosen by the user. Since typically collocated OSN functions are partitioned across multiple distinct providers, the user only exposes a small portion of her data to any one provider. Each user retains control over providers, and can revoke access to any that fail to respect her privacy policies or provide poor service. This is made possible by standardized APIs that foster a competitive market for commoditized OSN services. Second, Polaris preserves the economic incentives for OSN providers to offer free storage and managed applications to OSN users. Application providers have access to the limited data they need for their functionality, and can generate contextual ad revenue, *i.e.* ads based on status or photo content. Finally, Polaris offers each user a flexible choice between cost and privacy. Privacy conscious users can choose to pay providers that offer strong security, *e.g.* via fully encrypted data stores, while cost-conscious users can opt for free providers that generate revenue from data mining and targeted advertising.

Challenges. Running an OSN on mobile devices has several associated technical challenges stemming from their energy, bandwidth, and connectivity constraints. Hosting a user's entire set of social data from their mobile device would afford excellent privacy, but is not feasible. While these devices have adequate computational and storage capacity, serving data over the cellular network will be slow, and will quickly exhaust the device's battery.

Instead, the Polaris software on each mobile device acts as the user's identity manager, and generates and distributes authorization tokens to control data access. While the "primary" copy of all data is stored on the mobile device, service providers host data pertinent to their application and provide the necessary computation and data delivery needs. The user device also serves as a lightweight naming and lookup service, and gives a user information about a friend's choice of service providers, as well as pointers to find the friend's content on the provider. The standardized Polaris APIs ensure that friends will be able to access each other's social content even if they chose different providers for the same application. Polaris makes liberal use of caching for static data, further reducing the mobile device's role in ordinary transactions, and improving data availability. Finally, users can choose between multiple providers of the same OSN service (*e.g.* photo-hosting) based on performance, security, or cost. Decoupling different OSN services (*e.g.* photos, direct messages, blogs) allows providers to monetize the user data they have access to, while helping Polaris preserve privacy against untrustworthy providers.

[1] Mobile devices include smartphones, tablets, e-readers, *etc.*

Roadmap. We begin in Section 2 by describing the Polaris architecture, its key challenges, and our approaches to addressing each of them. Next, in Section 3, we describe basic OSN operations in Polaris using examples. We then discuss how to use primitives in Polaris to build core components necessary for today's full-featured OSNs in Section 4. Finally, we describe an initial prototype of Polaris for Android phones in Section 5, and conclude in Section 6.

2. ARCHITECTURE

In this section, we present the high level architecture for Polaris, our social network platform designed to improve user control over their data while providing a viable economic environment for third-party OSN providers. We begin by summarizing existing approaches with similar goals, and use it to set the context for our work. We then discuss our architecture, highlighting key challenges to our approach and describing how the Polaris architecture meets these challenges.

2.1 Existing and Alternative Architectures

The popularity of OSNs and the commensurate rise in privacy concerns have motivated a number of proposals for distributed OSNs that do not require users to give up their data to any particular (potentially untrustworthy) provider. Some propose users maintain their own data, and leverage P2P distributed hash tables (DHTs) to store, search, and access data [4, 7]. These DHT-based OSNs face the same challenges as all P2P systems, such as maintaining data integrity and availability in high-churn, distributed scenarios [3], as well as the inherent security risks in DHT systems.

Another proposed alternative is to use cloud-based hosting services to store social data and apply end-to-end encryption to preserve privacy. In Vis-a-Vis [12], users store and manage their data using virtual machines hosted in the cloud, while Persona [2] advocates using Attribute-Based Encryption and group key management to securely distribute social data. Contrail [13] is a mobile device-centric OSN that leverages a store-and-forward service hosted in the cloud to relay encrypted data to and from users.

All of these approaches achieve strong privacy by eliminating centralized OSN providers and encrypting all data. However, the use of blanket encryption significantly impairs economic incentives for third-party cloud services to host and manage social data for free, since it is impossible to do contextual advertising based on anonymized streams of encrypted data. This shifts monetary costs and management efforts to end-users. We believe these efforts are unlikely to gain traction, because history has shown that users are often unable or unwilling to take on the efforts or monetary costs of managing complex systems [5]. In this case, users are more likely to remain at existing, centralized OSNs, even if this means sacrificing privacy completely.

2.2 Leveraging Mobile Devices

Our primary goal is to create an OSN architecture that improves privacy for OSN users, while maintaining high levels of availability and performance associated with centralized OSNs. Since we believe users are unable or unwilling to take on the associated costs, our approach is to design an architecture that provides sufficient economic opportunities to motivate OSN providers to play the data hosting and management role.

Our insight is to leverage user's mobile devices as the control center for a distributed OSN platform. Mobile devices such as smartphones are an excellent fit for this task for several reasons. First, mobile devices are ubiquitous, and tightly tied to each individual. Second, unlike personal computers or laptops, mobile devices have much higher availability than desktop PCs. Finally,

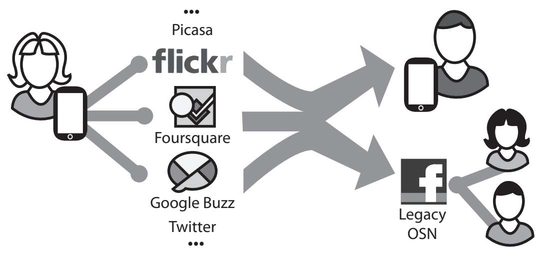

Figure 1: The Polaris system. Users select commoditized OSN providers for different application domains. Users retrieve social data directly from their friend's providers. Users on legacy OSNs receive updates from Polaris via site-specific APIs that bridge the two OSNs.

devices like smartphones and digital assistants already manage people's social contacts and other sensitive personal information, making the transition to hosting OSNs a natural one.

Despite their many advantages, the use of mobile devices as an OSN platform also presents several challenges:

- **Resource Constraints.** The high availability of mobile devices comes with two caveats: limited battery life and a low bandwidth network connection. These constraints limit the size and amount of social data that mobile devices can host.
- **Data Availability.** Energy and bandwidth constraints mean that some social data (*e.g.* photos) cannot be hosted on the mobile device. Instead, they must be hosted on highly available infrastructure with high capacity storage and bandwidth.
- **Network Restrictions.** Mobile devices are not universally accessible over the Internet: cellular providers routinely firewall access to their networks, and force mobile devices to access the Internet through proxies. These restrictions complicate using mobile devices as servers for social data.
- **Device Fallibility.** Mobile devices are often broken, lost, or stolen. Data on the device needs to be backed-up frequently to facilitate account restoration.

Note that we do not include storage constraints in our list of challenges above. Multimedia social content such as a user's photo library can grow to GBs in size. However, most modern devices have expandable storage via multi-gigabyte SD cards. If storage does become an issue, users can use desktops to provide permanent backup for content, and erase multimedia files after they have been replicated to the provider.

2.3 Commoditizing OSN Services

The four challenges outlined above prevent mobile devices from being the sole component in a distributed OSN. *Commoditized social network services* can mitigate these issues. Much of the heavy lifting can be offloaded to third party service providers that utilize their own solutions for high availability (*i.e.* clouds). This helps mobile devices preserve power so they can provide the user interface and host sensitive data. Since no single provider has access to all of a user's data, this minimizes the threat a single provider can pose to privacy. Furthermore, the standardized APIs that Polaris services must support ensure that a user's data remains portable in the event that any one provider is compromised or violates her expectations of privacy. Figure 1 depicts the overall Polaris system.

Economic Incentives. We believe that there are ample economic incentives for providers to offer OSN services, even if they are not the center of the Polaris OSN. In Polaris, all APIs use HTTP REST over SSL, so communications are secure from eavesdropping. However, there is no requirement for data itself to be encrypted; this capability is left as an additional feature that commoditized services may offer. This means that services can make money the traditional way, by serving contextual ads along with hosted content. Services can also sell large scale data-mining and analysis capabilities, even though they may only host a specific type of data, and only for a subset of all Polaris users. Even when user's identities are anonymous, it is still possible to mine streams of social data en-mass to perform real-time sentiment and popularity analysis, producing valuable results and generating revenue.

Creating an open market of commoditized services opens up new avenues for competitive revenue generation. Requiring services to support standardized APIs creates a wide-open playing field for companies to enter the market and compete for customers by innovating and differentiating their services. One way to do this is for providers to offer different tiers of service. For example, a photo hosting service might offer a basic, free package, but also a paid, premium package that enables hosting of higher resolution images, more storage space, and advanced online editing tools.

As mentioned above, service providers can also use security and privacy as a selling point. Providers can offer more security conscious users anonymous, fully-encrypted data storage facilities for a fee. This fosters an environment where security conscious users with higher levels of technical acumen can be catered to specifically, while still allowing them to interact with other, more cost conscious users who use less secure commoditized services.

2.4 Distributed Access Control

While commoditized services address data availability issues and enhance privacy in an economically sustainable way, they also create an additional challenge for Polaris:

- **Distributed Access Control.** For social data hosted directly on the user's mobile device, controlling access is trivial. However, hosting social data across a multitude of distributed services requires that we develop very reliable distributed access control mechanisms to preserve security and privacy.

To fully leverage commoditized services, users need a clear, simple, unified way of managing access to their data that is hosted remotely. To solve this challenge, the Polaris application on each mobile device can be seen as a security "kernel." This kernel acts as the sole identity provider for the user, manages access control lists (ACLs) for the user's data, and issues authorization tokens to other users and services. Polaris pushes these policies to the user's commoditized services via standardized APIs so that they obey the data access rules the user has defined.

Since all security sensitive operations (adding new friends or services, changing privacy settings, *etc.*) must serialize through the kernel on the user device, this allows Polaris to decouple data management and enforcement of security policies. The mobile device only needs to deal with a minimal number of policy management tasks that are well within the limits of its battery and bandwidth constraints. Meanwhile, the bulk of the user's data is stored away from the kernel of the system by external providers. This model affords Polaris very good performance and security isolation: *i.e.* even if an individual component (mobile device, commoditized service) goes offline, the user's security policies are unaffected, and overall data availability remains high.

3. POLARIS BASICS

In order to further clarify the operation of our system, we now go into greater details of some of the basic operations of Polaris. These

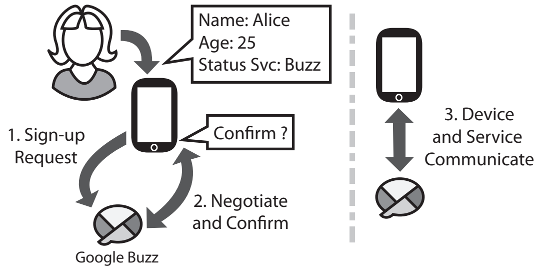

Figure 2: **Setting up Polaris. (1) Alice creates a profile and signs up for commoditized services. (2) Alice's device and the service negotiate, then Alice manually confirms the sign-up. (3) Alice's device and her services can now communicate freely.**

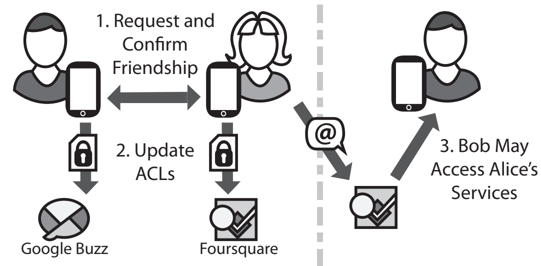

Figure 3: **Friending a new user. (1) Alice and Bob exchange friend requests and manually confirm the relationship. (2) Each user sends updated ACLs to their services. (3) Alice and Bob can now directly access each others'services.**

descriptions are meant to provide the high-level intuition of how the final system will operate, without delving into the exact specifics of APIs and protocols. In Section 5 we provide more concrete details of an actual prototype of Polaris.

First Steps. Figure 2 depicts the initial setup process for Polaris. A hypothetical user named Alice installs the Polaris client software on her mobile device and inputs the details of her social profile. Polaris automatically generates a unique, cryptographic identifier for Alice. Alice then selects what commoditized services she would like to use for things like updating her status, storing her photos, *etc.* Polaris contacts each service and requests a new user sign-up. This initiates a negotiation process during which Alice's identity is verified, and the service informs Alice of its terms and conditions, along with what personal data of hers it needs to access in order to function. Alice is free to reject requests if she feels the service is requesting too many privileges and select a different commoditized service to fill the role. If Alice finds the service acceptable, she confirms the new account sign-up, and Polaris and the service exchange unique authorization tokens to verify each other in the future. Alice can now interact with the service's APIs, and the service can push notifications to her mobile device.

The list of commoditized services that a user leverages are stored as part of her social profile by Polaris. When a user's friends query her device for profile information they also receive this list of services. They can use this information to locate the remainder of the user's content that is hosted remotely. Users can revoke a service's access to their data at any time by deleting its authorization token. Similarly, services that users only wish to use temporarily can be issued tokens with a-priori expiration dates.

User Identity and Bootstrapping Communications. Polaris utilizes two methods for users to uniquely identify themselves. The first is an OpenID URL, which provides a useful rendezvous point for others wishing to contact the user. This URL can resolve to a light-weight proxy that maintains a persistent connection to the user's mobile device. Maintaining such a connection is unlikely to significantly affect the device's battery life. For example, Google maintains a persistent connection to all Android phones with minimal impact on these devices. Proxy indirection is necessary due to firewalls that and NATs that impede direct connections to mobile devices on cellular networks.

The second method uses the globally unique, routable identifiers that are already associated with mobile devices. As we will demonstrate in Section 5, it is possible to use a data/SMS protocol to bootstrap traditional network connections to mobile devices. For example, consider two friends in the Polaris OSN, Alice and Bob, who have exchanged phone numbers. If Alice wants to view Bob's profile, Alice can send Bob a query over SMS stating her current IP and asking for Bob's current IP. Bob replies with his current IP, at which point Alice and Bob can use UDP hole punching techniques to communicate directly over the Internet.

The data/SMS communication approach has the advantage of alleviating the need for proxies to circumvent cellular network firewalls. Furthermore, by using the mobile device's data connection for the majority of data transfer we leverage the fastest connectivity available to the device while minimizing the use of costly SMS messages. In the event that a user is uncomfortable giving out their mobile number, they can use a phone number aliasing service like Google Voice to mask their true number, or rely on an alternate channel such as e-mail or instant messaging. If the identifier associated with an alternate channels is compromised it can simply be discarded and replaced without affecting the user's primary phone number or e-mail account.

Friending and Access Control. Controlling access to information stored on users' mobile devices is insufficient for a privacy centric OSN: access control must also extend to data hosted by commoditized services. Part of the standardized API that commoditized services must support is the ability for users to upload and manage ACLs for data stored on that service. This extends the privacy protection offered by Polaris from the user's mobile device onto remote services as well.

Polaris supports two methods for "friending" other users. The first is proximity based: users in direct contact can quickly and securely trade information using the local area networking capabilities of their mobile devices. The second method involves more traditional, asynchronous friend requests. Friend requests can be directed to users via their OpenID or their phone number, at which point the request is stored pending acceptance.

Figure 3 shows the process of friending and establishing distributed access controls in Polaris. Alice and Bob bootstrap the friendship process by sending friend requests to each other. Each user's device contacts the other in order to perform identity verification, confirm the friendship, exchange authorization tokens, and retrieve profile data from the new friend. During this exchange, Alice sets the access privileges that Bob has to her data, and vice-versa. Once Alice has confirmed the friendship, her device pushes updated access control policies to her commoditized services. This serves the dual purpose of informing the services she is now friends with Bob, and telling them what social data (if any) Bob is allowed access to. Bob can then query Alice's services (which he learned about during the initial exchange of profile information) to retrieve her social data, without having to contact Alice directly. Alice is

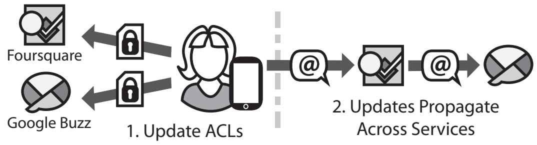

Figure 4: Service composition. (1) Alice updates her ACLs to give Foursquare write access to Buzz. (2) Updates to Foursquare can now be forwarded to Buzz.

free to update her access policies at any time, or unfriend Bob entirely by deleting his authorization token from her ACLs.

Because all commoditized services use standardized APIs for identity verification and authorization, users are able to access data hosted on their friend's services, even if they do not have an account there. Friends also cache each other's profile data, which mitigates the impact of mobile devices being unavailable. Users push profile change notifications directly to friends in order to maintain consistency. The privacy of browsers is protected, since users can not tell when data cached on other's devices or hosted by third parties is accessed.

Finding a technological solution to ensure that commoditized services faithfully honor user's ACLs is an extremely difficult challenge. Instead, we believe that a social solution to this problem is more practical. In the event that a commoditized service fails to uphold users' privacy policies, users can simply abandon that service and move to a more secure competitor. Users do not have this freedom with current OSNs because their friend links are not portable to other providers. However, in Polaris this migration is made simple by the fact that the APIs commoditized services must implement include functions that enable data and friend portability. Thus, it is in each service's best interest not to violate user's privacy, as such a breach is sure to be well publicized (probably via social news-media) and will certainly cause a user exodus.

Service Composition. In many cases, it is useful for commoditized services to be able to interact directly, without user intervention. We refer to this as *service composition*, and Polaris supports this functionality. Figure 4 illustrates the process of composing commoditized services. Suppose Alice uses a geolocation service like Foursquare that is capable of publishing location updates to her status. If Alice wants to use this feature, she can modify her ACLs to give Foursquare write access to Buzz on her behalf, using a new access token. At this point Foursquare can publish updates directly to Alice's status on Buzz without contacting her mobile device. Note that Alice's personal authorization token to Buzz and the token generated for Foursquare are distinct: Alice can revoke Foursquare's access to Buzz at any time by deleting its token and pushing the change to Buzz.

4. TOWARDS A FEDERATED OSN

Thus far we have outlined the basic architecture of Polaris, focusing on how we lay the foundations for an economically viable distributed OSN that leverages commoditized OSN services. In this section we will examine how to use the basic, primitive operations enabled by Polaris to construct higher level functionality that is necessary for a modern, full-featured OSN.

Search. A key function of OSNs is search, as this is how users locate their friends. In the absence of centralization it is not clear how to replicate this functionality in a distributed OSN scenario. Our solution is to treat search as another federated service, alongside standard services like status update and photo hosting. Polaris users can register themselves with any centralized search directories that they choose, along with some portion of their personal information to enable keyword search. Canonical identifiers like users' OpenIDs and phone numbers are used to bootstrap communications between devices.

In this model, there is ample opportunity for search services to differentiate themselves. Just as Facebook used to be divided into networks, search services can cater to specific regions, companies, schools, cultures, *etc.* Search services can also implement additional security features, such as requiring users to verify their affiliations by providing corporate/edu e-mail addresses, or verifying users region by checking the area code of their phone number or their GPS coordinates. In this way, a whole ecosystem of search services can flourish, each catering to different segments of the population and offering varying levels of security.

Incremental Deployment. After the initial deployment of Polaris, there will be a transition period during which early adopters migrate, while their friends remain at existing OSNs. Without any mechanism to support incremental deployment, Polaris users will be disconnected from friends on other OSNs, which is a significant barrier to Polaris' adoption. Fortunately, most existing OSNs have APIs that allow data to be read and written by external apps. The Polaris client can forward updates to, and read updates from, friends on legacy OSNs using these APIs. Because Polaris only leverages official API channels, it is unlikely to run afoul of system administrators at existing OSNs.

Real-time Communication. Polaris leverages a publish/subscribe model to enable push-based, real-time data communications. Commoditized services that are expected to publish updates are allowed to view users' friend lists, thus making it simple to perform routing of data. Each user's mobile device pushes her access control lists to the user's service providers. This allows Polaris to support fine-grained control over dissemination of updates, enforcing the data access rules the user has defined. Finally, Polaris supports both forward-direction messages (*e.g.* new updates) and reverse-direction ones (*e.g.* comments, mentions, and likes).

Social Applications. Polaris' focus on leveraging open APIs means that third-party social applications should have no trouble interfacing with users and their devices. The same mechanisms that Polaris uses to identify other users and authorize commoditized services can be used to secure interactions with social applications. With respect to user's privacy, Polaris actually simplifies the process of using social applications. Social applications can be thought of as a special case of the normal friending process: users can assign data access control policies to applications the same way they can to their friends. Additionally, because users have complete control of their identity in Polaris, they cannot get opted-in to social applications without their knowledge or consent.

5. A POLARIS PROTOTYPE

We have implemented a prototype of Polaris as a standard "App" on the Android smartphone platform. Android is a rapidly growing, open-source platform for smartphones with 28% of the smartphone market in the USA [9]. The Android Marketplace allows users to download and install over 90,000 applications [1]. Android applications are Java based and run on a modified Linux core. Applications have access to both core Java libraries and Google libraries for phone specific functions.

Our Polaris prototype uses an SQLite database to store all of the user's information, including profile fields, friend lists, ACLs, and settings related to the user's service providers. Access to services is provided by Polaris plug-ins. Our current prototype has plug-ins supporting Twitter, Google Buzz, and Youtube, as well as implementations of our custom, Polaris-only APIs.

Global Reachability. As mentioned in Section 1, current generation mobile devices on cellular networks reside behind a firewall that blocks incoming connections initiated from Internet hosts. This means services running on mobile devices must find alternate means to accept incoming requests. Our Polaris prototype can receive incoming requests over two communications channels. The first channel is Android specific: it uses Google push notifications as a proxy to relay messages to the Polaris application running on a mobile device. If necessary, messages relayed in this manner can instruct Polaris to send outgoing HTTP GET requests for additional content associated with the message.

A second, more general solution is to use SMS messaging as the communication backbone. The Polaris prototype registers a SMS broadcast receiver with the Android OS, which triggers when SMS messages with a specified prefix arrive. Polaris SMS messages contain an action followed by a sequence of parameters. One example usage of this channel is requesting and accepting new friends. Polaris notifies the user of incoming friend requests via Android's Status Bar notification system, at which point the user can accept or deny the request. One SMS message is required for sending a friend request, and one is required for the reply.

Portability. Our prototype's dependence on Android is specific to this implementation and not fundamental. Since many messages are asynchronous and do not require real-time responses, we can envision replacing them with a web-based polling system. Alternatively, users can bootstrap communications by tunneling through a free messaging service such as Gtalk or Skype.

The primary obstacle limiting Polaris' portability is the openness of other mobile OS platforms. While all of the major smartphone platforms include push-notification APIs, not all allow third-party applications access to the SMS inbox (iOS being the prime example). As other new mobile platforms emerge (Windows Phone 7, ChromeOS) we will evaluate them for suitability as target platforms for Polaris.

6. LIMITATIONS AND ONGOING WORK

This paper describes first steps in realizing the Polaris OSN. While we have addressed some of the core technical challenges, several key issues remain the subject of our ongoing work. We summarize them here and outline plans to address them.

Energy Consumption. Energy Consumption is a key concern for all mobile applications. Intuitively, using Polaris on a smartphone is similar in energy usage to using a Facebook reader. Polaris will send and receive additional control messages for modifying access control lists and friendship links. We are currently preparing for a detailed trace-based event-driven energy consumption study of Polaris components and protocols.

Security and Auditing. From a security perspective, a distributed OSN architecture significantly increases the attack surface and the difficulty of detecting and defending against attacks. As ongoing work, Polaris requires a detailed security analysis as well as design of carefully placed auditing mechanisms in the system.

Availability and Scalability. Polaris strives to maximize data availability by caching profiles and hosting social data on commoditized services. Additional trace-based experiments will be used to demonstrate the feasibility of our approach to maintaining high-availability. Furthermore, these experiments will examine potential scalability challenges in Polaris, such as heavy-hitter operations like aggregating news-feeds from distributed sources.

Device Migration. The existing Polaris prototype assumes that users only have a single, persistent mobile device. Future extensions will accommodate multiple devices per person, as well as facilitate migrating Polaris accounts between devices in the event that they are replaced, broken, or stolen. These features can be implemented by expanding the role of the proxy service to store encrypted account backups and support multiple devices per user.

Incentivizing Providers. Besides migrating users away from centralized OSNs, successful adoption of Polaris also requires fostering the growth of multiple providers in each application domain. Market forces already create tension between central OSNs like Facebook and competing service providers like Zynga games. We will encourage and simplify the move away from central OSNs by releasing open-source implementations of Polaris APIs in several languages used by web applications.

Application Limitations. Applications that require visibility into significant portions of the social graph do not work well in distributed OSNs like Polaris. While the lack of a central provider might eliminate the need for graph-wide analysis tools, this is still a limitation of the architecture.

7. REFERENCES

[1] ANDROLIB. Android market statistics, July 2010. `http://www.androlib.com/appstats.aspx`.

[2] BADEN, R., BENDER, A., SPRING, N., BHATTACHARJEE, B., AND STARIN, D. Persona: An online social network with user-defined privacy. In *Proc. of SIGCOMM* (2009).

[3] BLAKE, C., AND RODRIGUES, R. High availability, scalable storage, dynamic peer networks: Pick two. In *Proc. of HotOS* (2003).

[4] BUCHEGGER, S., SCHIÖBERG, D., VU, L.-H., AND DATTA, A. Peerson: P2p social networking: early experiences and insights. In *Proc. of SNS* (2009).

[5] CARLSON, N. Debunked: Why you'll never have to pay for facebook. CNN.com, June 2010.

[6] COOK, B., MESSINA, C., RECORDON, D., AND HALFF, L. Oauth, 2010. `http://oauth.net/`.

[7] CUTILLO, L. A., MOLVA, R., AND STRUFE, T. Safebook: Feasibility of transitive cooperation for privacy on a decentralized social network. In *Proc. of IEEE WoWMoM AOC* (2009).

[8] FITZPATRICK, B. Openid. OpenID Foundation, 2010. `http://openid.net/`.

[9] KAFKA, P. Is android really outselling apple?, May 2010.

[10] NIELSEN. Smartphone penetration over 50% in 2011, 2010. `http://www.gpsbusinessnews.com`.

[11] PRODROMOU, E., VIBBER, B., WALKER, J., AND COPLEY, Z. Ostatus, 2010. `http://ostatus.org/`.

[12] SHAKIMOV, A., VARSHAVSKY, A., COX, L., AND CÃĄCERES, R. Privacy, cost, and availability tradeoffs in decentralized osns. In *Proc. of WOSN* (2009).

[13] STUEDI, P., MOHOMED, I., BALAKRISHNAN, M., RAMASUBRAMANIAN, V., WOBBER, T., TERRY, D., AND MAO, Z. M. Contrail: Enabling decentralized social networks on smartphones. Tech. Rep. MSR-TR-2010-132, Microsoft Research, 2010.

[14] TOOTOONCHIAN, A., SAROIU, S., GANJALI, Y., AND WOLMAN, A. Lockr: Better privacy for social networks. In *Proc. of CoNEXT* (2009).