Musubi: Disintermediated Interactive Social Feeds for Mobile Devices

Ben Dodson  Ian Vo  T. J. Purtell  Monica S. Lam
Computer Science Department
Stanford University, Stanford, CA 94305
{bjdodson,ianvo,tpurtell, lam}@cs.stanford.edu

ABSTRACT
Mobile devices play an integral role in our online activities, most notably in the domain of personal and social computing. This paper presents Musubi, a mobile social application platform that enables users to share any data type in real-time feeds created by any application on the phone. Musubi is unique in providing a disintermediated service; all communication is supported using public key encryption thus leaking no user information to a third party.

Despite the heavy use of cryptography to provide user authentication and access control, users found Musubi simple to use. We embed key exchange within familiar friending actions, and allow users to interact with any friend in their address books without requiring them to join a common network apriori. Our feed abstraction allows users to easily exercise access control. All data reside on the phone, granting users the freedom to apply applications of their choice.

In addition to disintermediating personal messaging, we have created an application platform to support multi-party software with the same respect for personal data. The SocialKit library we created on top of Musubi’s trusted communication protocol facilitates the development of multi-party applications and integrates with Musubi to provide a compelling group application experience. SocialKit allows developers to make social, interactive applications without needing to host their own servers.

1. INTRODUCTION

Smart phones are fast becoming the device of choice when it comes to personal and social computing. We take pictures on the phone, we play music on the phone, and we can even remotely control our TVs. Through Bluetooth, NFC (Near-Field Communication), and QR codes we can build ad-hoc personal device networks. Eventually we will have all our credentials on the phone so we can unlock paid contents anywhere without logging in.

As the gateway to everything personal and digital, the phone is an extension to ourselves. Being constantly on and connected, the phone has the ability to let us share and celebrate all our daily moments with our friends and loved ones.

For those who do not wish to give up the totality of our digital selves to a for-profit third party, the phones can allow us to interact with others without intermediation. This paper presents Musubi which allows users to share virtually anything with our contacts without sacrificing personal data privacy or ownership.

1.1 Disintermediation of Social Interactions

Everything social is built with intermediation today, from simple applications like Tic-Tac-Toe to full-blown social networks. While earlier social software would collect their own friends, now almost all social software is developed on platforms like Facebook to get access of users’ friends through Facebook Connect. More and more personal information is available online and collected by a single party. Although today’s social networks are providing better sharing controls, ultimately they gain ownership of the personal data. While many teenagers and young adults are not worried about privacy, they are concerned about their image on their social feeds, which is affected by not just what they post, but what their family, friends, and friends-of-friends post as well.

With the availability of personal smart phones, we now have the opportunity to create disintermediated social services. This is attractive not just for privacy-conscious individuals, but for friends to share anything with each other from silly pictures, random thoughts, to dark secrets, without worries.

Phones have persistent connections to the internet, however, mobile IP networks disallow incoming connections to the devices because of security concerns. We created the Trusted Group Communication Protocol, or TGCP, to enable phones to contact each other through a cloud relay while preserving information security. Data is protected using public key cryptography, and the decrypted data is only available on the end users devices.

1.2 A Decentralized Trusted Social Graph

In the Musubi system, there is no single central social graph. The graph is decentralized; our friends’ contact information is stored in our address book. In Musubi, we can interact with anybody on our phone’s address book, without having to ask users to sign up on an external social network. Individuals are identified by our public keys. PKI systems for personal identity are notoriously hard to use [30]. Our goal is hide the security mechanisms so well that even a child can use them. We capture existing friendship by securely identifying the public keys of contacts within the user’s native address book. Second, we leverage smart phone technologies such as NFC and Bluetooth to allow users to seamlessly connect with each other as they interact in the physical world. Finally, we provide a group-centric user interface that aggregates application activity and communication into interactive private social feeds.

1.3 Sharing Everything Interactively

Our goal is to let users share everything on the phone interactively with each other. They can share not just static
content like status updates and photos, but content from any mobile application. Moreover, we want to enable the sharing of collaborative applications, all disintermediated like the core Musubi system itself. We make that possible by creating a library called SocialKit that lets developers focus simply on the client software. The library automatically helps connect participants, exchange messages among participants by simply reading and writing to a feed, and maintain distributed state by writing and reading from an automatically replicated social database. There is no need to host a server to run such apps; moreover, no friends’ information is leaked to any third party.

1.4 Contributions

This paper presents the design of a complete social sharing platform for smart phones. The contributions of this paper include:

- We have developed TGCP, a trusted communication protocol for exchanging encrypted messages amongst friends and within groups on the mobile phone. This is the backbone of our disintermediated services.
- We have created a social application platform that enables the creation of multi-party apps with disintermediated communication.
- Musubi allows users to interact with their friends right out of their address books. Users can share just about everything on friends’ feeds. The Musubi feeds let collaborative apps advertise their progress on the same feed; users can launch any of the shared apps with a consistent touch-and-interact interface.
- We have developed a large collection of Musubi applications that range from simple sharing of to-do lists to engaging multi-player games. All these disintermediated apps can be developed quickly using our application platform and requiring no servers.

2. USER EXPERIENCE

A primary goal for Musubi was to make the experience feel as integrated and natural as possible. This includes reducing the frictions associated with setup, interactions with friends, and integration of applications.

We provide some sample scenarios to show how Musubi is integrated into the social experience on mobile. We note that as described below the user is unaware of the underlying cryptographic operations involved.

Ready from install. Michael installs Musubi on his phone. Immediately he is able to send and receive messages from all of his friends in his address book. He sends his mother a picture of himself smiling at work, and quickly receives a message back from his mother saying “I’m so proud of you Michael, love you!”

Acquiring new friends. Michael meets new customers everyday. It is his job to keep up a good relationship with all of them in order to keep their business. Now when he meets new associates, instead of exchanging business cards he simply touches his phone to theirs, and they automatically become friends over Musubi. Michael now manages all of his business associates on his phone.

Easy access control. Michael has been using Musubi for some time now, and now belongs to a few groups: “College Buddies”, “The Scotts”, and “Dunder Mifflin Paper.” He likes this separation of friends, family, and coworkers, as he can act casually in one group and not worry about how people in the other group might perceive him. He takes a picture of himself drinking with some friends at the bar and posts it to his “College Buddies” group without any concern that his mother, who is opposed to alcohol, might see the picture.

Group sharing. Michael organized a company outing for his sales team for doing a good job this past quarter. He bought them all tickets to the zoo. He wants to keep a record of the experience, so he creates a new group “Dunder Mifflin Zoo Trip.” He broadcasts the group locally using GPS and password protects it. His employees check “Nearby Groups” on their phones, enter in the password, and join Michael’s new group. Michael creates a to-do list of goals for the day with the TodoBento application, as seen in Figure 1. Everyone turns on the “Share photos” mode, and all of the pictures they take on their phone while this mode is enabled are automatically shared with just the “Dunder Mifflin Zoo Trip” group.

Non-Musubi applications. One of Michael’s favorite applications on his phone is PicSay, a free photo-editor he downloaded from the market. It is not a Musubi application. Michael sees that Jim posted a picture of a cat in the “Dunder Mifflin Paper” feed and can’t help but think that it would be hilarious to see the cat with a monocle. He long presses the cat picture, presses edit, and loads the cat picture into PicSay. After he puts the finishing touches on his cat, he presses the back button and his well-dressed cat automatically appears in the feed.

Easy multi-party applications. Michael gets a notification from Musubi that some of the guys in the office are starting a poker game and he wants to join in. He doesn’t have the poker game installed, but when he clicks on the game he is automatically brought to the market page for the poker application. After he installs it, he joins the office poker session and begins playing.

3. A DECENTRALIZED TRUSTED SOCIAL GRAPH

We build upon the familiar address book model: once friends have exchanged contact information they are able to contact each other without needing to sign up with a common third-party provider. In this way, Musubi is built on top of a decentralized social graph, with users knowing only who their friends are. We base our system on a couple of principles, enforced with the aid of encryption, to ensure
that communication between individuals and within a group is trustworthy and eliminates spam.

1. Two people can communicate with each other if and only if they have each other’s public keys in their address books. Mutual key exchange makes two parties friends.

2. Group memberships are maintained by the group members who share a key pair that governs access to the group. Sharing the group key-pair and a partial group list is enough to allow a new person to join an existing group.

We show how to provide users a simple-to-use system where Musubi handles all aspects of key management for communication and group management for handling user interactions en masse.

3.1 Key Issuance

By focusing on mobile, we can take advantage of the fact that the phone is a single-owner device. Accounts can be bound to a device eliminating the need for account registration. Upon installation, a public/private key pair is generated on the user’s behalf, allowing the user to participate in Musubi interactions with virtually no setup.

Users have the option of associating their public keys with their web identities to facilitate key distribution and revocation. For example, public keys can be stored with web finger providers [32] such as Google. There has also been a proposal to place public keys on Facebook profiles using SocialKeys [16]. Lastly, we can create a key directory service that allows users to bind their key with a publicly known identity via OpenID [23] authentication. If users lose or leak their secret key, they can simply update their public keys.

Users are not required to back their key up and can instead take full responsibility for handling the distribution of their public key. This is ideal for those who are not in a position to link their account with an identity that is directly associated with them, for example activists and protesters.

3.2 Importing Friends

As with phone numbers and emails, public keys can be imported into the address book. We can optionally take advantage of a user’s address book in conjunction with publicly hosted key directories to automatically acquire all of their friends’ keys. This avoids the bootstrapping problem that plagues emerging social networks of how to import existing friends into a new system. From the moment a user links one of their existing identities to Musubi, the user is set up in the system and able to interact with all of their friends in their address book who also have Musubi installed.

We initialize the address book with public keys as follows:

1. A user installs Musubi and authenticates with OpenID or Facebook Connect to claim ownership of an addressable identifier.
2. Musubi publishes their public key in the key directory.
3. Musubi computes a one way hash of the public identifiers of all of the contacts in their address book and uploads this to the key directory. Hashing the identifiers prevents the key directory from discovering friends’ identities unless they are also using the key directory.
4. The key directory receives a list of public keys for contacts who have connected to the directory service.

Figure 2: A new Musubi client can use the directory service to quickly make friend connections.

5. If a friend later adds himself to the key directory, the directory notifies the user’s friends of his new public key.

An additional benefit of using the address book in this fashion is the ability to add friends before they become Musubi users. This is an advantage over existing models where friendship requests can only be initiated once a user is registered in the system.

3.3 Inviting Existing Friends

Users may want to request a friendship with a person who is not already a Musubi user. The user can send a friend request through an external channel such as email. This request is an HTTP link that can be intercepted by Musubi to initiate the friending process. If Musubi is not installed, the link leads to a webpage prompting the user to install it.

3.4 Making New Friends

For users who meet for the first time and have NFC enabled phones, the easiest and most direct way is to exchange public keys is through physical contact. Musubi takes advantage of the natural handshake interaction that people perform when they introduce themselves. By touching phones, we can utilize NFC to directly exchange public keys, thereby establishing friendship.

Groups allow for mass acquisition of friends. Once a user has created a group, they can extend invitations in a similar manner to friend invitations. Group invitations are represented as HTTP links that can be intercepted by Musubi to initiate the joining process of acquiring and inviting the memberlist, adding those public keys to the user’s address book. If Musubi is not installed, the link leads to a webpage that prompts the user to install it. The link can be distributed through any medium, for example NFC, email, a GPS-based group broadcasting server, or even on a website, and any group member is able to generate an invitation.

4. TRUSTED GROUP COMMUNICATION PROTOCOL

Mobile phones are the primary platform of a modern social network. These devices are always on, often connected, and intimately involved with our daily lives. Unfortunately,
many of the techniques used to create alternative social network are not optimal on a mobile device. For example, a standard P2P architecture where clients directly connect to each other to exchange messages will not work for smartphones. The 3G network service provided for these devices usually does not allow for incoming connections to be accepted by the device. This limitation mandates the use of an external service to buffer messaging between smartphones. Furthermore, aggregating non-critical message delivery allows for significantly reduced power consumption [18].

To preserve the principle of disintermediation, the Musubi social application platform relies on end-to-end encryption. Musubi uses the Trusted Group Communication Protocol, or TGCP, for the required message transport primitives for the application platform. TGCP is a multicast messaging system that provides identity-based routing, server-hosted message queues, and connection aggregation.

4.1 Trusted Identity-Based Messaging

Instead of using human readable names, such as an email address, the TGCP protocol specifies that a public key be used as a global identifier. Messages are addressed to individuals by their public keys; the content of the messages are encrypted and signed with the senders’ private keys. A user can hand the encrypted message off to any server to forward the message to a recipient; as only the recipients can decrypt the messages with their secret keys. Since the messages are signed, the recipient can authenticate the sender. Servers can even cache large amounts of data for clients without imposing any risk to privacy. This design preserves the disintermediation principle because the messaging subsystem is not privy to the content of the messages. However, the server is given a pseudonymous view of the structure of groups and frequency of messages.

4.2 Message Format

The content of the message consists of a header and a payload. The payload is encrypted with a 128-bit AES key using CBC mode with PKCS5 padding. Each message is encrypted with a different key thus allowing future messages to be obscured from unwanted participants if they are removed from a group. The header includes a copy of the symmetric message key for each recipient. The individual copies of the symmetric key are encrypted with the RSA public key of each recipient using ECB mode with PKCS1 padding. The sender’s public key and an RSASSA-PKCS1-v1_5 signature using the SHA1 hash function for the full message are also included in the header to prevent tampering [10].

| signature | from | to0 | key0 | ... | toN | keyN | body |

Figure 3: A TGCP packet is encrypted with a per-message key and signed for authenticity.

4.3 Group Management

Joining a group implies two actions on the part of a TGCP client: finding all members of a group, and tracking an identifier for the group scope, or feed, in which the members participate. Beyond the basic message transport mechanism, TGCP defines a scheme for managing group membership, shown in Figure 4. Each group is identified by an RSA key-pair. When membership control messages are sent between participants, they are encoded using the group private key rather than the sender’s private key. This allows for a new person who is not yet a friend to seamlessly join the group. Each participant in the group has added the public key for the group to its whitelist thus allowing for group control messages to reach them.

Figure 4: Any group member can invite new participants by sharing the group key pair and a partial list of group members with another TGCP client.

To invite someone to a group, we give a snapshot of group membership (at least one member) and the key pair for the group to the invitee. They send a JoinNotification message signed using the group private key to the members of the group. When the other members of the group get the request, they add the public key to their list of trusted friends. Then the existing members compare the membership view that the invitee sees with their own view of the members. If there are additional members the invitee does not yet know about, the client replies with a JoinResponse object that includes its view of the group membership. This message is again encrypted with the group key. The invitee sends a JoinNotification message to any new members that are discovered as peers respond to the join requests.

Because of the requirement that individuals exchange public keys to become friends, asynchronous friending over an external medium requires a secret to be exchanged that allows communication to proceed through the whitelist. TGCP treats a friend invitation as a degenerate case of a group invitation. When a user wants to invite a friend, he or she generates a key-pair which will be added to the whitelist. The user gives his or her public key and the group key-pair to the invitee over an external medium. The invitee can now use the dummy group key to send a message to the invitee without being blocked by the whitelist. After the bidirectional friendship is established, the invitor can remove the dummy group key from its whitelist.

4.4 Federation

One possible implementation of TGCP is a centralized cluster that handles the routing of messages between all mobile phone users. We have implemented this design to explore the higher level challenges inherent in implementing a decentralizable application platform. For the purpose of geographic distribution and choice of different providers, a federated design is necessary. In a federated model, each public key is associated with a server, or home agent, that serves as the designated contact point for a mobile device.

Federated TGCP aggregates all social activity into a single stream of messages to and from the home agent. The protocol implies a network topology where many clients are
connected to a single server, and each server is connected
to many other servers. The servers participate in the peer-to-peer network that routes traffic between devices, while
the client simply pushes and pulls messages to and from its
self-designated home agent, as in Figure 5.

TGCP is intended to operate using semi-trusted networks
of home agents, such as those that might be provided by
telecom operators. Messages are addressed to a set of indi-
viduals by specifying their public keys with an encoded
TGCP packet. Once more than one server is available for
message routing, it is necessary to designate which server
is pertinent to the specific recipient of a message. The ex-
pected home agent for a particular keyed identity is also
encoded in the TGCP packet. This allows the common case
of messaging to happen without requiring servers to consult
a global database of designated home agents. The client can
easily keep track of the home agent for its contacts because
the home agent designation changes infrequently.

5. MUSUBI FEEDS

Musubi is designed to be a highly engaging application
platform. Users can share everything, including not only
status updates and photos, but arbitrary content generated
by any application on the phone as well as interactive ap-
lication states that solicit active and ongoing participation
from group members.

5.1 The Feed Abstraction

All objects shared within a group are organized in the
abstraction we call a feed. A feed can be a long-lasting con-
nexion with a mostly fixed group of people. A feed can also
be associated with a physical place such as someone’s house,
those that might be provided by
telecom operators. Messages are addressed to a set of indi-
viduals by specifying their public keys with an encoded
TGCP packet. Once more than one server is available for
message routing, it is necessary to designate which server
is pertinent to the specific recipient of a message. The ex-
pected home agent for a particular keyed identity is also
encoded in the TGCP packet. This allows the common case
of messaging to happen without requiring servers to consult
a global database of designated home agents. The client can
easily keep track of the home agent for its contacts because
the home agent designation changes infrequently.

5.2 Interactions on a Feed

Besides letting users type in status updates or post pic-
tures, users can interact with content on the feed in several novel ways.

Data Import and Export. Musubi supports the basic
kind of sharing available on today’s mobile devices. For
e example, a link can be shared from a web browser with a few clicks; the user elects to share a link, chooses Musubi,
and then chooses the friends or feeds with which the link
should be shared. Later, the link can be shared from the
feed over email, or launched in the browser.

In-Place Editing. Users can edit objects in the feed and
reshare them easily. They click on the object, pick among
a list of applications installed on the phone that support
editing that object, and the data is shared upon exiting the
program. As seen in Figure 7, two clicks allows a user to
invoke their favorite photo editing application to edit a pic-
ture that has been shared in a feed, and when he is done, the
content is instantly shared with the group. This basic inter-
activity is enabled through the “edit” intent on the Android
platform. The Musubi feed provides the context of how the
data is shared.

Virtual presence. Musubi takes advantage of being
“mobile-first” by supporting ongoing engagement that ex-
tends beyond Musubi’s core application. The phone is an
important personal accompaniment and is increasingly privy
to our daily goings-on. Musubi makes it easy for friends to
continually share activities through the phone, giving them
a sense of companionship the whole time.

With a click on the feed of their choice, users can elect
to share a variety of ongoing activities from their phone as
“presence updates” to a feed. For example, users can enable
GPS location sharing. This feature is similar to several other
services such as Loopt, Google Latitude, or Google+, but
without intermediation.

The “Camera presence” shares any newly captured images
with a group whenever a photo is captured with the built-in
Android camera application. A group of friends can enable
camera presence during a night out to automatically share
their captured experience with each other without requiring
Figure 7: Using the PicSay photo editor to modify a picture in a feed.

a centralized service. We can also broadcast the music we listen to and even the TV shows we watch. Sharing such data with feeds provides fresh context for us to interact with friends while limiting the scope of what we share.

Multi-party applications. All the modalities described so far show how users can share information of existing applications on mobile devices, provided that these applications support the proper intents. Musubi further enables new kinds of disintermediated interactions in the form of multi-party applications by building upon the platform primitives provided by Musubi, as described in the next section. Interactions with these applications are analogous to the above; an app adds a visual representation of itself to a feed, and clicking on the feed entry launches the application for further interaction.

6. MULTI-PARTY APP PLATFORM

A survey of developer experiences today reveals that writing multi-party games for mobile phones remains a daunting task. Consider the exercise of writing a basic game of Tic Tac Toe for phones, played across two remote devices. In the standard model, we must (1) choose an API for pairing friends, such as Facebook Connect; (2) set up a server to handle the pairing; (3) manage push notifications from server to client; (4) write the mobile client.

Musubi reduces the task to only writing the mobile client. We expose the platform’s interaction primitives through an API allowing application writers to simply write the mobile client code. They do not have to set up a server. Note that we do not preclude the development of server-client apps; we are simply providing an alternative model that was not available before.

6.1 Overview

A decentralized multi-party application needs
1. a group of participants,
2. a sequence of objects that represent interactions between members,
3. distribution or replication of these objects across members’ devices.

The feed is a basic example of such an application. We further extend the feed primitives to multi-party application developers so apps can be developed as a subfeed within a feed. The main feed serves as a form of “bulletin board” of arbitrary posts from a mix of applications, where users can issue and accept invitations to new app sessions. Users can easily join these sessions even if they do not have that app installed. Moreover, they do not have to monitor the status of independent applications, users are alerted of state updates from any applications on the general feed.

In this model, an application is bound by a parent feed and communicates purely with feed primitives. Musubi thus provides an identity firewall that enables applications to send messages to a users’ friends without learning their identities.

Our current interprocess communication makes use of Android’s built-in mechanisms including Content Providers and Intents. On other mobile platforms like iOS, applications are sandboxed and we cannot provide this level of communication. Instead, we can support simplified html5-powered applications hosted within our process.

The following describes the application platform we have built for the Android OS. The API is accessible directly as a Content Provider representing feed data. We also provide a library called SocialKit to further simplify the development of multi-party applications.

6.2 Subfeeds

The abstraction of a subfeed is used for grouping messages from the same application session as well as for grouping user responses to messages within a feed. Technically, every object in a feed is the head of a subfeed. Each object has a unique identifier generated from the TGCP’s packet signature which we reference for our semantic graph. The subfeed is then a collection of objects that lists the head as a “parent.”

Independent subfeeds allow different app instances (such as two different games of Poker) to maintain state independently. Musubi supports a mechanism for updating the view within a feed that is associated with an app instance by sending a specially typed object to the subfeed. The rendering is available on devices even if the application is not yet installed.

Figure 8 demonstrates how a poker application interacts with a Musubi feed. The feed has two different application instances for our application, each representing a game played with different people. The poker app associates internal state information with each session which is shared across all participants. It also periodically updates the visual representation of the game so it can be rendered in Musubi as seen in Figure 10.

There are two ways in which an application can attach itself as a subfeed to a feed. The user can first pick the people whom he wishes to interact with and then the application, or he can pick the app first and then choose participants.

In the first case, the user launches an application within a Musubi feed. When a user views a feed, a button press allows him to choose an application to launch, adding it to the feed. The application is automatically handed a subfeed identifier, which it can use for subsequent communication.

In the second case, an application can call into Musubi to prompt the user for a feed. On Android, this is achieved by having the app send an intent for resulting data, which Musubi populates with the feed identifier. During the process, a user may create a new feed for the application.

6.3 Messaging
Once an app is connected to a feed, it can send arbitrary data objects to it. The communication passes through Musubi, which creates the message shown in Figure 9. The message contains the application’s data and also includes a timestamp, an identifier for the sending application, and a sequence number indicating how many messages the user has sent to this feed. Musubi signs the object with the user’s private key and sends it out on the TGCP network.

Objects are composed of properties that address various needs of application development. An object must contain a type (a short string identifier), and can optionally contain JSON-formatted data as well as raw binary. The size of a message is practically limited to about 1 MB due to constraints on the devices. When an object is received, the timestamp, sending application’s identifier, and user signature are also made available to an application.

Musubi social database is presented as a Content Provider, allowing applications in controlled ways. In Android parlance, the Content Provider allows us to enforce access controls, as we can determine the application that sent a query.

If an application is given access to a feed, it can obtain limited information about the users of that feed, including their public key, nickname, and profile picture. This is enough information to visually represent the user and to establish a communication with them by way of Musubi. More sensitive data such as phone numbers, email addresses, and physical addresses are by design not accessible. We believe a user should not have the ability to share this data on behalf of a friend, preferring instead for this data to be opted-in for sharing from that friend’s device.

Currently, applications are restricted to seeing only data generated from their application from any member in the feed. We are actively exploring models that allow willing applications to exchange data.

### 6.4 Social Database

Objects entered into the social database are available to applications in controlled ways. In Android parlance, the Musubi social database is presented as a Content Provider, capable of running queries over the feeds and objects it maintains. The Content Provider allows us to enforce access controls, as we can determine the application that sent a query.

If an application is given access to a feed, it can obtain limited information about the users of that feed, including their public key, nickname, and profile picture. This is enough information to visually represent the user and to establish a communication with them by way of Musubi. More sensitive data such as phone numbers, email addresses, and physical addresses are by design not accessible. We believe a user should not have the ability to share this data on behalf of a friend, preferring instead for this data to be opted-in for sharing from that friend’s device.

Currently, applications are restricted to seeing only data generated from their application from any member in the feed. We are actively exploring models that allow willing applications to exchange data.

### 6.5 Distributed State

Even with the friend management and connectivity provided by Musubi, writing distributed applications remains a challenge. We must also help developers manage distributed state, tackling transportation and state management details while providing a straight-forward, usable API.

To help develop applications over the distributed datastore, Musubi provides APIs to manage common application styles. For example, the TurnBasedGame API maintains consistency for applications in which only one user at a time is allowed to update the application’s state. The API allows a developer to set a JSON object representing the application’s state, to specify whose turn is next, and also to embed a text, html, or bitmap thumbnail representing the application for display in the main feed. We believe this greatly simplified API can support a variety of applications.

The universal identifier given to the app instance’s feed object can also be used to set up data connections outside of Musubi. We have demonstrated how this identifier can be used to run application sessions using the Junction API [6].

Junction allows applications to communicate in realtime, as if they were in a chatroom. A unique session identifier acts as a capability for joining that session. Junction supports communication over a variety of channels, including XMPP, Bluetooth, and a local LAN. In each case, a device acts as a “switchboard,” routing messages and establishing a global ordering. While objects in Musubi are implicitly persisted, Junction messages are transient. Junction also provides its own system for managing distributed state with itsProps abstraction, which may be preferable to Musubi’s APIs for some styles of applications.

The app instance identifier can also be used to maintain state on a central server. Musubi is still useful for establishing the application’s membership and embedding it in a social feed. Such deployments have additional maintenance costs and scalability challenges, but greatly simplify state consistency issues for many types of applications. This technique also allows a vast number of legacy applications to interoperate with Musubi.

### 7. EXPERIMENTAL RESULTS

#### 7.1 Implementation

Our initial prototype of Musubi was developed on the Android platform, with an iPhone version in progress. We have made Musubi available in the Android Market since July of 2011. Our codebase is completely open source and available on GitHub [15].

Our experimental implementation of TGCP uses a stock distribution of the popular AMQP server RabbitMQ. Each individual owns a message queue on the server named according to their public key. When a person sends a message, they create a fanout exchange that multicasts message to the other recipients. Then, they send a public a message to the fanout, which causes it to be distributed to all of the recipients’ message queues. The message queues are durable so that messages are guaranteed to be delivered. The client acks messages from the server as it consumes them to ensure that each message is processed by the client at least once. The client itself keeps track of the signatures of messages it has received and fully processed to avoid repeatedly handling the same message. Our single server instance was able...
7.2 User Experience

Anecdotally, our research group has been using our prototype continuously for over four months. On a larger scale, Musubi and our whiteboard application have been deployed at a charter school in New Jersey to 150 K-3 students. Teachers have responded favorably to the ability to share a private, collaborative drawing space with their students, using it for situations such as helping a student practice their cursive.

We have also conducted a study with two groups of 15 students each, aged 10-12, at a local Montessori school [28]. The platform was exceedingly popular with the children, so much so that parents were concerned about how to limit usage. The children had little trouble adapting to the system, making excessive use of picture taking and photo editing with PicSay, while remaining completely unaware of the underlying cryptographic operations. The students had mostly positive feedback, with a number of them proclaiming that "Musubi is AWESOME!"

7.3 Application Experience

The Musubi application provides a suite of functionalities including group management, app communication, and distributed state management. We describe below a sample of the different kinds of collaborative apps built for Musubi.

Interactive Sharing. Musubi makes it trivial to build interactive sharing applications among friends. TadPoll allows members to conduct quick polls. A user runs TadPoll in a feed and enters a question he would like people to answer. The question is rendered as an html thumbnail in the feed, and clicking it prompts users to respond to the poll. Each answer is persisted in the social database, and when a new answer is inserted, the html thumbnail updates to show current answers from the aggregated results. The basic TadPoll implementation is approximately 175 lines of code. In the same vein, TodoBento helps friends keep track of a common to-do list; friends can put up new items and check them off when completed. These apps will more likely be used if they are frictionless and integrated into the regular communication flow.

Turn-Based Apps. Many casual games are turn-based apps. Instead of requiring developers to manage scalable web services, Musubi lets all these apps be written with only client software. Tic-Tac-Toe is implemented using SocialKit’s TurnBasedGame API. Tic-Tac-Toe sessions are launched from within a feed, with Musubi prompting the user to select an opponent. Tic-Tac-Toe can be written in less than 200 lines of code, leveraging the TurnBasedGame API to maintain state and display players’ names and images, as well as to represent the game’s state as an html thumbnail in the encapsulating feed. Tic-Tac-Toe is particularly easy to write because its state is known in full to all players; other compelling games share this property, including Chess, Checkers, Connect 4, and many more.

weHold’em and WordPlay are two fancier turn-based games implemented using SocialKit. WeHold’em is a game of Texas Hold’em poker for 2 to 8 players, and WordPlay is a Scrabble variant for up to 4 players. In weHold’em, each player joins with a fixed amount of money, and the game is played in rounds until a player’s money runs out. WordPlay runs a single game per application instance. Both games publish html thumbnails to the feed to update their state.

Unlike Tic-Tac-Toe, both weHold’em and WordPlay involve application state that should be kept private to each player. We keep state private by simply not rendering it on devices that do not have access, however a curious user could determine other player’s private state by exploring their local database. Because Musubi is designed for use with friends, this is not necessarily a grave concern. It is also possible to run these games with cryptographically verified privacy [17].

Real-Time Collaboration. weTube and wePaint are two real-time applications we had previously built using Junction. weTube is a collaborative playlisting application. A weTube session is started by invoking the application from a feed. Users can choose media from the web to add to a playlist, including YouTube videos or links to music tracks. Links can be voted up or down, and are sorted based on their popularity. WePaint is a collaborative whiteboard app. Friends can draw with each other in real-time, with Junction’s Props abstraction handling consistency across devices.

Junction applications are ad-hoc in design, and Musubi provides a context in which they may run. Integration took approximately 20 lines of code per app, and each can be shared directly from a feed. wePaint can also be launched atop an existing image in the feed, making itself available using the “edit” intent. Musubi passes the object’s universal identifier as an argument to the editor, which allows wePaint to establish a unique, shared editing session.

Making Existing Apps Social. Jinzora is an independently developed media streaming application that connects to a personally-hosted cloud-based service to browse and stream a music collection. Jinzora integrates with Musubi in two ways. First, it can publish playback information to feeds as a form of presence sharing, letting friends know what you are currently listening to. Since Jinzora’s music is hosted in the cloud, a friend can click on a music entry to play it back immediately.

Second, Jinzora can be launched as a feed app, sharing a session across devices. When Jinzora detects that it has been launched as such, it allows the client to be put in “jukebox” or “remote” mode. When in jukebox mode, Jinzora listens for messages to update the current playlist or to control playback, turning the device into a media renderer. Remote mode shares playback requests over Musubi rather than handling them on the local device.

The basic jukebox functionality took roughly an hour to implement, and required about 50 lines of code. Presence sharing required about 10 lines. Note that presence updates occur on a main feed, and jukeboxing occurs within a subfeed.

Discussion. We have shown that a large range of applications, from simple polling among friends to complete media browsing and streaming apps, can be integrated into the Musubi social experience seamlessly. We found the uniformity in the user interface to be very helpful because users know how to interact with new apps instinctively. Unlike other collaborative apps, these Musubi apps have no access to friends’ contact information. And except for Jinzora, which streams from a media server, none of these apps have a server component. weHold’em and TodoBento are written by non-Musubi developers, who succeeded in building these
apps without our involvement. This suggests that the API is usable and complete enough for at least these apps.

8. RELATED WORK

8.1 Groups as a Primitive

Group management in online social networks are a popular feature that enhances community interactions. The burden of getting group functionality tailored for efficacy in each and every social application is daunting, and the primitives need to become a part of a basic social operating system. When properly integrated into a system, group-based communication systems can increase usage by more than double [4]. Systems such as Cluestr, SocialFlows, and Facebook SmartLists all strive to eliminate the overhead of creating groups, yet they do not leverage the possibility of real-time acquisition of groups from mobile sensors nor do they provide a mechanism for ongoing private communication with the group members [8, 13, 26].

8.2 Alternative Social Application Platforms

A few companies, such as Skiller and OpenFeint, have attempted to provide game developers with a managed, back-end networking service so that developers need not worry about implementing and hosting a server. These companies, however, address this problem with a centralized solution and do not provide the same guarantees of privacy that encryption offers [19, 25]. Many systems have been built that try to enhance the ability of the individual to control the flow of information about them. Decentralization and encryption are the techniques that support preservation of data ownership and information flow control. Diaspora was a newsworthy example of a distributed social network because they were able to rapidly garner the interest and money of a crowd of people who were yearning for a safe social option [24]. There are a plethora of other alternative social network protocols in the open source community, each with a slightly different architecture [1, 29, 31]. SocialVPN suggests that the communication interface can be any IP based protocol and that members of groups should be connected via a virtualized subnet [11]. Distributed social communication systems, such as Contrail and MobiClique, have also already made their way to the mobile phone, however, these systems do not address the difficulties the architectures present to distributed multi-party application developers [27, 20].

Google’s Plus social network emphasizes the need for private correspondence by making users specify the precise groups they wish to share with, similar to how email works [7]. Although this network encourages users to consider the audience of their posts, it does not provide any guarantees about data ownership and privacy beyond what is outlined in the EULA. FlyByNight takes a different approach and simply uses existing centralized social networks to manage friend relationships, while storing only encrypted content on a central application server [12]. Technically it provides disintermediation, but it requires all participants to join the same proprietary social network.

8.3 Identity and Routing

External identities can be authenticated in many ways ranging from standards like OpenID [23] or OAuth [9] to proprietary system such as Facebook Connect. Our system allows for real world identities to have a public key attached using these federated identity systems, but we do not require the Musubi identity to be bound to a “real world” account. This opens up the possibility to have pseudonymous identities, with the caveat that the structure of the social graph is still visible to the network operators. This is distinct from the anonymization provided by services like TOR [5] which obscure the endpoints of internet communication channels.

Because of the difficulties associated with P2P overlay routing schemes [3], the Musubi design depends on reliable traffic relay services. The reliable, but not fully trusted, provider model is more similar to existing systems for electronic mail and instant messaging than P2P routing techniques [14, 33]. TGCP allows delivery of messages across a variety of network links because it is an identity-based routing system [2] with embeddable routing hints.

Authenticator is a framework for establishing secured connections between friend on existing social networks [22]. After negotiating a shared key using the messaging primitives of existing social networks, it encodes future messages within the available platform-specific sharable types. It does not address the issue of data sharing between applications, but rather solves the problem of passing a secret message to a friend programmatically using existing channels. SocialKeys also proposes a mechanism for bootstrapping cryptographically secured identities through existing social networks by repurposing existing data sharing APIs [16].
9. CONCLUSION

This paper presents the Musubi social sharing platform which enables users to share and interact with friends on the phone without having to give up privacy to any third-party service providers. We have focused on creating a usable system, making sure the cryptographic operations used in enforcing privacy and security do not get in the way of the usability of the software. We have made the social experience very natural on the phone; we can interact with anybody on the phone’s address book, without having to ask users to sign up on an external social network. Our informal user study involving about thirty children suggests that the software is attractive. We have made Musubi and a host of Musubi applications (Jinzora, wePaint, weHold’em, weTube) available in the Android Market, and all the source is open and available on GitHub. Videos showing some of Musubi’s features are available on http://mobisocial.stanford.edu/musubi.

Disintermediation of phone interactions is accomplished with the design of the Trusted Group Communication Protocol (TGCP). The data are routed according to the public keys, thus enabling many possible relatively simple implementations. Making this TGCP protocol available through our application-level API, we enable easy development of many collaborative decentralized applications that honor privacy and without the need of hosting servers.

As we have demonstrated with the Musubi applications described in this paper, we now have the ability to create small and large decentralized social apps without having to worry about disclosing our friends’ information to a third party. Even non-Musubi applications can share information privately through our system using Android intents. We envision one day that there will be hundreds of thousands of Musubi applications, and all of the native applications will share within Musubi feeds.

10. ACKNOWLEDGMENTS

We would like to thank Aemon Cannon, Chanh Nguyen, and Kuzuya Yokoyama for their help with the Musubi project. This research was funded in part by NSF Programmable Open Mobile Internet (POMI) 2020 Expedition Grant 0832820, the Stanford MobiSocial Computing Laboratory, and a Stanford graduate student fellowship.

11. REFERENCES


