Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

SEPIA Mobile: Teaching Privacy to Android Apps

Mathias Karlsson

Semester Thesis 04.07.2011 - 02.12.2011

Advisors: Dr. Martin Burkhart
Dilip Many
Supervisor: Prof. Dr. Bernhard Plattner

Computer Engineering and Networks Laboratory (TIK)
Department of Information Technology and Electrical Engineering

Abstract

Mobile devices offer an excellent platform for social applications since we carry them around the whole time. Social applications are all based around sharing private information with other people. But we do not only share this information with the people we choose but we also have to make it available to the service provider. The user has to agree to the terms of the service provider which can go as far as complete ownership of any content made available through the service.

This semester thesis explores the possibilities to provide privacy measures to mobile applications on Android systems using the SEPIA library developed at ETH Zurich. As an example application a friend locator application is developed. This service informs the user whenever friends are close similar to Google Latitude or Foursquare. Previous work explored ways of preserving the location privacy by randomization or truncation of the location information. Additionally some employ cryptography to hide the location information completely from the central service provider. Our approach is different from previous work because it maintains the advantages of a centralized service with regard to the mobile device (e.g. minimal communication, connectivity). Using the SEPIA library, the shared information is secure without the need for performing complex cryptographic operations on the mobile devices or establishing a PKI for all the mobile devices. The privacy peers are able to perform closeness checks over a set of friends without learning the location of either party. The implemented prototype Android application serves as proof of concept that it is possible to develop viable privacy-preserving mobile apps using the SEPIA library.

Acknowledgements

I am very thankful for the support of my two tutors Dr. Martin Burkhart and Dilip Many. I was able to profit greatly from their technical expertise and supportive advice as well as their great flexibility and up front feedback which I greatly appreciated. I also want to thank my wife Jagoda Karlsson for her support and motivation. I also thank Prof. Dr. Bernhard Plattner for his feedback in the intermediate review and presentation.

Contents

Chapter 1

Introduction

Smartphones hold a vast amount of privacy sensitive data and are often criticized for compromising user privacy. In this work we explore the possibilities of secure multiparty computation (MPC) for mobile devices to enable privacy preserving services.

As an example application a friend-locator application was chosen. This is an application that enables the user to get information of close friends and then for example arrange a meeting. Prominent examples for non privacy preserving friend locator applications are Google Latitude [1], foursquare [2] or Gowalla [3]. To make such an application work we have to decide whether two friends are close or not. This can be done either with or without the help of a service provider. Normally, this is done with a central service provider because of the limitations of mobile devices (partial connectivity, limited bandwidth). Commonly, the mobile device sends its current location to the service provider and then receives the location-based data and provides it to the user. The service provider therefore knows the exact location of the device and is able to profile the user. The user is forced to trust the service provider to not misuse and keep the the information confidential. The past few years have shown that this trust is not always justified. This not only increased the privacy concerns of the users but also attracted the spotlight of public attention to this topic, summoning concerned organizations and regulators to protect the user. The following public discussions showed that the opinions about what is private information and how it should be handled are vastly different.

With MPC, this central service provider can be split into several parties that by themselves know nothing about the user, but still provide the desired service to the user. The user now only has to trust a subset of all the parties and can be sure that even if some data is leaked it cannot be misused. SEPIA, developed at ETH Zurich, offers a an efficient framework for MPC which has already proven very efficient for aggregation of network security and monitoring data [5].

The goal of this work is therefore to first adapt the SEPIA MPC library [5] for the android platform and then develop an example application around it.

Chapter 2

Prior Work

The topic of privacy-preserving location-aware services (i.e. friend locator) has been of growing interest, especially since the commercial success of smart-phones. Several ways were explored to approach location privacy preservation but many still provide the functionality using peer-to-peer protocols or with the help of a central entity.

An early approach was to transform the coordinates with a distance-preserving coordinate transformation [9] and send it to a central service provider. It is easy for the service provider to calculate the distance between two friends but it also learns the exact distance of two friends and can therefore exclude certain locations. Later, homomorphic coordinate encryption [11] was explored. In this case both friends have to be online to complete the operation and, depending on the protocol, certain information is leaked to the friends.

The above approaches have the disadvantage that no minimum privacy is guaranteed since multiple friends together are able to triangulate the exact location [8]. This can be prevented with a grid based approach. At first the exact location is generalized (i.e. using a cell ID or a cell center coordinate) and then sent to the service provider. The service provider is able to compare the distance of the generalized locations and can inform the friends about their proximity [7]. The provider only learns the generalized location. This can be prevented by encrypting the cell identifier [6]. It is also possible to provide results of different precision using several grids of different granularity [10]. With this approach, a certain minimum privacy (minimum privacy of the friend with the smallest minimum privacy) is guaranteed even if friends collaborate with the service provider. But this guaranteed minimum privacy comes at the cost of service precision and therefore results in false positives as well as false negatives. The cells of two friends might fulfill the distance requirement but not match the exact locations. Additionally, neighboring cells have to be checked since the friends could stand next to each other but in different cells. Instead of checking neighboring cells it is also possible to use overlapping shifted grids [8].

All the approaches using encryption have the disadvantage that either secrets have to be shared between the devices or public keys of the friends have to be distributed in an authenticated

way, effectively requiring a PKI.

Implementing a privacy-preserving mobile application with SEPIA brings several advantages. Assuming a solid SEPIA privacy peer infrastructure, it is possible to maintain the performance advantages of a centralized service. One such advantage is the permanent connectivity of the service provider. Therefore it is not necessary that the mobile devices are always connected for the service to work. It also minimizes the required communication for the mobile devices which can be important with regard to costs. Additionally such a service has usuallymuch more computational power available than mobile devices, especially with regards to cryptography. Since the privacy sensitive information is safe due to multiparty-computation [5] no cryptographic keys have to be exchanged between mobile devices greatly simplifying the mobile application.

Chapter 3

Location Representation

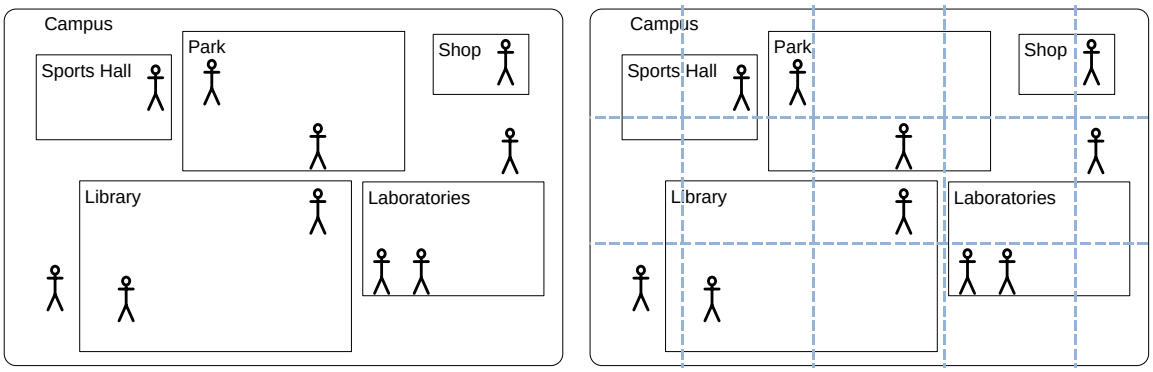
In this chapter first the requirements of a privacy preserving location for a friend locator service are established. The subsequent sections discuss two possible mappings from the real location to a privacy preserving location and finally their advantages and disadvantages with regard to a real world friend locator app for mobile devices.

3.1 Privacy Preserving Location

In friend locator services such as Google Latitude [1] or foursquare [2] the user generally sends his precise location to the service provider which subsequently forwards this information to the friends. The user then has an overview over all the friends' locations, no matter the distance, which does not retain much privacy.

At first we have to define what kind of location privacy we would like to preserve. Most likely I don't want to let everybody know where I am exactly to preserve a certain minimum privacy. This minimum privacy can be defined with a *minimum privacy range*. Obviously the privacy comes at the cost of accuracy of the service but as long as the *minimum privacy range* is within a reasonable range (e.g. 100 meters) the friend locator service still works and provides the user with sufficient amount of privacy. As an additional privacy feature we propose a *maximum location sharing range*. This limits the distance the location is shared with other friends, e.g. 7 kilometers. In context of a friend locator service this makes sense since there is no point of revealing the location to friends that are far away.

The difficult part is to find a suitable mapping of the given location information, latitude and longitude, to a privacy preserving location which still allows to find close friends. Intuitively in a first approach one would choose circles or randomization around the present location [9]. But these simple approaches are vulnerable to triangulation and therefore violate the *minimum privacy range* or are prone to false results. Because of this the location has to be discretized to preserve the privacy.



(a) Discrete Location (b) Map Cell

Figure 3.1: Simple examples for the two discussed location mappings

3.2 Discrete Locations

The basic idea of *Discrete Locations* is to have a set of predefined locations like foursquare [2] or Gowalla [3]. Such a location can be a shop, pub or university, basically anything that can be defined by a coordinate and a name (see figure 3.1(a)). The user then can choose to check in to a location which normally works as follows. First the mobile device checks its coordinates and sends it to the service provider and then receives nearby locations which it presents to the user (e.g. as marks on the map). Second the user decides to check in to a location which tells the mobile device to inform the service provider about the chosen location. If the wanted location does not exist the user can create a new one sending the precise location with a name and maybe some additional information. The service also sends the information of friends of the user to the device that is displayed to the user e.g. as icons on the map.

With regard to friends a certain minimum privacy depending on the location size is preserved. But this basic design clearly violates the privacy since the service provider learns the exact location and the chosen location ID. This can be prevented by downloading the location database to the mobile device and only sending the location ID. In this case at least a certain minimum privacy is preserved. Alternatively the location ID can be cryptographically protected (e.g. encryption, multiparty-computation). This complicates the check for close friends greatly since the service provider is no longer aware of the location. But depending on the protection it is still relatively easy to find friends checked in to the same location. This effectively means that the *maximum location sharing range* is equal to the *minimum privacy range*. Additionally it assumes that for each friend the user wants to preserve the same privacy but in reality one might want to make a difference between friends or acquaintances.

Using cryptographic protected location information the *Discrete Locations* approach cannot solve the *close friend not same location problem*. This means that for two friends that are checked

in to two different locations which are very close to each other the service cannot determine the closeness. Imagine two shops next to each other. The friends can be as close as a few meters but would not be informed.

This approach actually does not solve the mapping problem on its own. It only provides the user with a list of close by locations and delegates the really difficult part of mapping the precise location to a privacy preserving location to the user.

3.2.1 Advantages

- User is in control whether to check in or not
- User solves the mapping problem
- Location definition understandable to the user (e.g. building, square, etc.)
- Discrete locations with unique IDs
- Efficient check with equality comparison

3.2.2 Disadvantages

- Locations have to be predefined and provided to the user
- Precision of the service depends on the location (i.e campus vs. pub)
- Privacy depends on the location (i.e campus vs. pub)
- Same privacy for all the friends
- Cannot solve the *close friend not same location problem*

3.3 Map Cell

A map cell is a predefined map area wherein the mobile device is at a given time (e.g. field in a grid, see figure 3.1(b)). The cell can be identified by a defined cell ID or coordinate (i.e coordinate of the center or a chosen corner. In this case the procedure would works as follows. First the mobile device checks its coordinates and calculates the identifier for the cell which is sent to the service provider. Second the service provider checks whether any friends are close by and informs the user. This can be done by comparing the cell IDs [10] or comparing the distances of cells with a threshold [7].

The service always provides a certain minimum privacy depending on the cell size. With regard to both the service provider and the friends since never a precise location is transmitted. When the location ID is additionally protected by cryptographic means it conceals the location to the service provider completely. But this also complicates the close friend check greatly. This effectively means that the *maximum location sharing range* is equal to the *minimum privacy range*. Additionally the simple approach of using one grid not only assumes that the user wants to preserve for

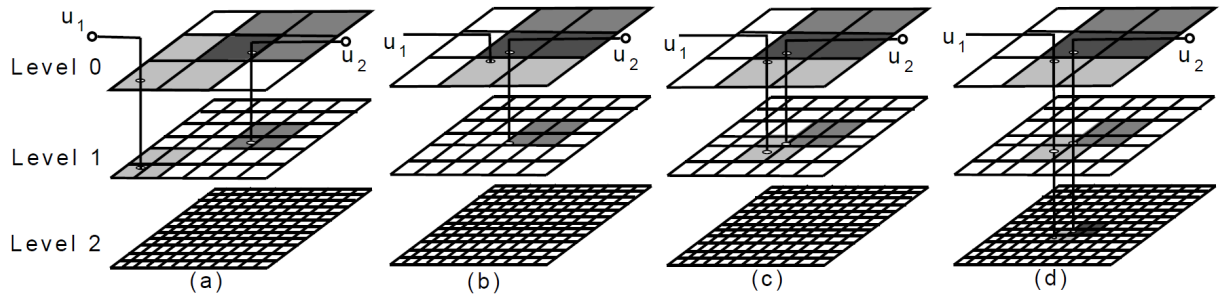


Figure 3.2: Multiple Resolution Layers, taken from [10, Figure 1]

each friend the same privacy but also that all the users have the same privacy needs.

The simple *Map Cell* approach also cannot solve the *close friend not same location problem* since two friends could stand next to each other but the cell border is between them. This is particularly unintuitive for the user if it splits up a location such as a building, room or a square (see figure 3.1(b)).

These shortcomings can be overcome using multiple grids. By using grids with different resolutions similar to [10] (see figure 3.2) it is possible to provide the user with a multitude of settings. This effectively decouples the *maximum location sharing range* and the *minimum privacy range*. This also enables the possibility to set different settings on a per friend basis (e.g. for Alice 10x10 meter up to 10x10 kilometers, Bob 100x100 meters up to 10x10 kilometers). Even the *close friend not same location problem* can be solved with overlapping grids [8]. These quite significant advantages sadly don't come for free. Instead of performing one check per friend it is now necessary to perform one for each resolution in every grid (number of grids times number of resolutions).

Last but not least the really tricky part is to find such a cell mapping for the earth that fulfills all these requirements. For that, the quadrature of the sphere problem must be solved, at least approximately. But to be able to take advantage of all the possibilities, additional viable overlapping mappings have to be established.

3.3.1 Advantages

- Preserves user specific minimum privacy
- Precision can be friend dependent
- Privacy can be friend dependent
- Can solve the *close friend not same location problem*
- Automatic location updates possible
- No local storage required
- Efficient check with equality comparison

3.3.2 Disadvantages

- Location mapping abstract for the user
- Must solve mapping of current location to cell ID (quadrature of the sphere)
- Much more information might have to be sent to the service provider
- Many more checks might have to be done by the service provider

3.4 Comparison

With respect to user control and usability, the two location mappings are roughly equivalent. *Discrete Location* gives the user more control and information due to additional information or tags associated with the location. But it can be quite tedious to always check in to the proper location and can easily be forgotten. This could be improved with a locally stored set of bookmarked locations to which it will automatically check in. In contrast, *Map Cell* updates its location continuously without user interaction. To prevent updates of unwanted locations it could be either turned off or put in a private mode.

Regarding the limited memory and bandwidth of a mobile device it would be preferable to not store and update large amounts of data on the device. This favors *Map Cell* since it is only computed on the mobile device whereas *Discrete Locations* needs frequent location information updates.

Using the simple *Map Cell* approach with only one grid and one resolution the load for the service provider is the same as for *Discrete Location*. But when the additional options of *Map Cell* are offered, the service provider has to cache more data and perform more checks. On the other hand for the *Discrete Location* a location database has to be frequently updated and provided to the users. This cannot be outsourced to the users as a *register new location* feature since this would violate the users privacy because it is necessary to send a precise location information along with a location name.

The user privacy and service precision are inversely coupled, the more privacy is needed the less accurate the service is. This holds for both definitions. But whereas for *Discrete Location* the privacy and precision depend on the predefined location with *Map Cell* it is possible to choose different settings per friend. A friend dependent privacy would better reflect reality since we do not trust everybody equally. Additionally a minimum privacy can be chosen which is not possible with *Discrete Locations* since it is defined by the location size.

With regard to the implementation both approaches have their own obstacles. The *Discrete Locations'* main challenge is to create and provide the data to the user. Whereas for the *Map Cell* the earth surface has to be approximated well enough into 'equal' areas with as little error as possible.

Chapter 4

Application Design

In this chapter the design of the entire application is described. First it will give a brief overview of the whole application. In the following sections the main parts are described in more detail.

4.1 General Application Design

The applications as a whole consists of several components that interact with each other (see figure 4.1). There are two functional layers. The *Administration Layer* handles all administrative tasks (e.g. user registration, etc.). The second layer is the *Computation Layer* wherein all the privacy relevant computation takes place. Certain components interact with both layers. On the mobile devices the friend locator application is running. Its task is to interact with the user and communicate with the other components. In SEPIA terms this application takes the role of an input peer.

4.2 User Administration Service

The user administration service resides only in the *Administration Layer* and provides all the needed administrative functionality to the other components. It maintains a list of all registered users and their relationships with the associated settings. This list is provided to the privacy peers. The users interact with the user administration service through the application on the mobile device to perform administrative tasks.

4.2.1 Operations available to the Privacy Peers

- Download the friendship list

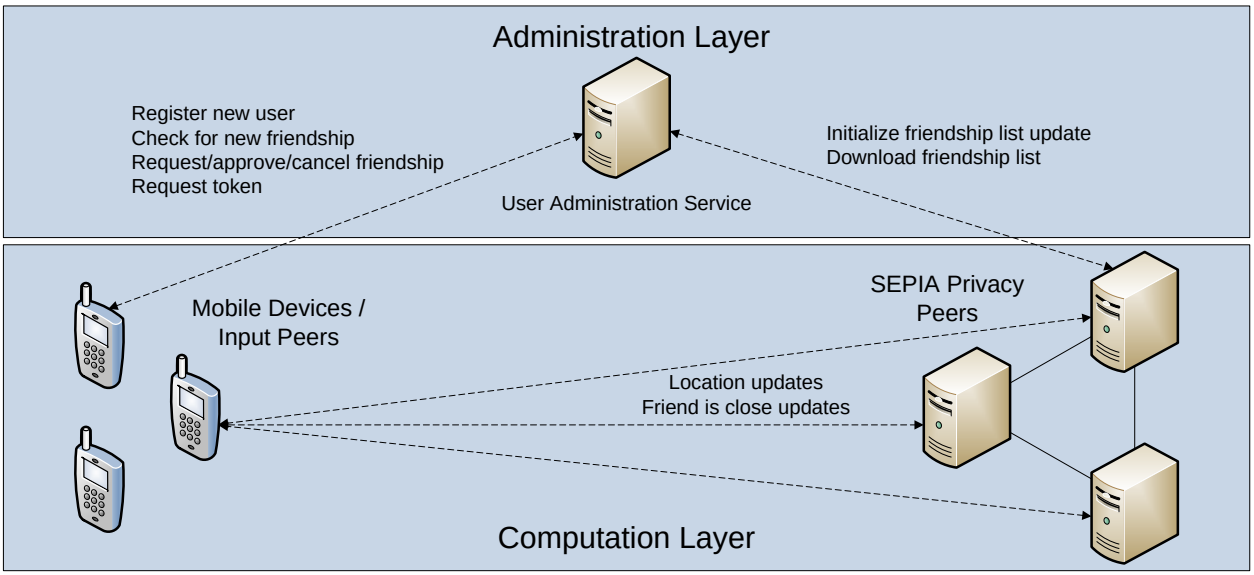


Figure 4.1: Friend Locator application design

4.2.2 Operations available to the Input Peers

- Register new user
- Request friendship
- Check for new friendship request
- Approve friendship
- Cancel friendship
- Request token

The user administration service also initiates the update mechanism for the privacy peers. Otherwise it could happen that the friendships lists are inconsistent among the privacy peers leading to disagreement for which pairs the isFriendClose check should be executed or not.

It is assumed that such a service exists. It will not be implemented at this stage since it would be out of scope of the work. The main focus of this work is on the computation layer. For testing purposes the values (e.g. friends list, user credentials, etc.) are directly given to the other components.

4.3 Privacy Peers

The privacy peers interact with other components on both layers. On the *Administration Layer* they receive updates of the friendship list from the user administration service. This information is required for isFriendClose checks to know which friends should be checked and to respect the

privacy settings for the relationship. On the *Computation Layer*, all the privacy relevant operations occur.

The privacy peers receive location updates of the mobile devices as Shamir shares which are stored. When an isFriendClose check is triggered the relationship list is checked to verify the friendship and to get the relationship settings. Then the corresponding Shamir shares are fetched, the isFriendClose function evaluated and the result is sent to the mobile devices. The isFriendClose check is triggered after a location update is received. It could also be done as a separate request but for simplicity it is done as one.

4.3.1 Operations available to the user administration service

- Initialize friendship list update

4.3.2 Operations available to the mobile devices

- Update location

4.4 Mobile Device/Input Peer

The input peers also interact with other components on both layers. On the *Administration Layer* they can register as a new input peer. Afterwards the user can request, approve and cancel friendships through the application. To get informed about unapproved friendship requests the mobile device has to query the administration service. On the *Computation Layer* the mobile device can update its location and receive information if a friend is close.

4.4.1 Platform independent Function Description

The device first checks its precise location using GPS or WLAN identifiers. This location information is given to a location digester. The location digester takes the provided location information and creates a privacy preserving location information depending on the used location mapping (see section 3.2 or 3.3). The created Shamir shares are sent to the privacy peers which will perform an isFriendClose check and send the result to the mobile devices. After receiving a friend is close update and there are additional friends close the user will be notified. These actions are ideally performed in the background to keep the information up to date. To interact with the user a user interface is needed.

4.4.2 Actions available to the user

- Register user

- Request friendship
- Approve friendship
- Cancel friendship
- Show friend status
- Activate/deactivate automatic location update
- Manually initiate a location update

4.4.3 Design for Android

As already mentioned in 4.2 the user administration service and therefore the whole *Administration Layer* will not be implemented since it is out of the scope of this work. Because of that all these functionalities won't be further discussed in the following design definitions.

Android knows four basic types of application components - activity, background service, data provider and broadcast receiver. In the friend locator application we have two types of tasks. First the automatic location update and user notification after a friend is newly close. This does not need user interaction and is best implemented as a background service because it keeps running in the background. The background service is basically the android implementation of SEPIA input peer. Second everything that is in direct interaction with the user (UI related). These functionalities have to be implemented as activities.

Functionalities of the background service

- Update location
- Receive friend update
- Notify user

Functionalities of the activity

- Show list of friends with status
- Check for friends (manual location update)
- Activate/deactivate/turn off background service

The *update location* and *receive friend update* functions are of special interest since they interact with SEPIA. *update location* retrieves the precise location, creates the Shamir shares and sends them to the privacy peers. While *receive friend update* processes the received friend information given by the privacy peers.

The location digest should be defined as an interface. Doing so it will be possible to easily implement different location definitions, depending on the applications need.

Chapter 5

Implementation

In this chapter all the major decisions and experiences made during the implementation are described.

5.1 Android Version

First an Android version had to be chosen for which the application will be developed. The test devices available are Google Nexus One which at the time supported up to Android 2.3.4 (currently 2.3.6). Even if version 2.3 is widely adapted still a large number devices are still using 2.2. Version 2.2 and 2.3 together make the vast majority of the currently available devices and since any Android application running on v2.2 also runs on v2.3 version 2.2 was selected as basis for the implementation.

5.2 Location Definition

As location definition the *Discrete Location* (see section 3.2 on page 6) was chosen. Even if at first sight the *Map Cell* approach might have more advantages in regard of privacy preservation, it has severe disadvantages with regard to an implementation with SEPIA.

Using the *Map Cell* definition we have to find a mapping effectively cutting the world into cells. This can be done using the Military Grid Reference System [4] which is used by the NATO militaries. The system assigns to each cell a unique string that must be mapped to a java long for sepia to work. With regard of the possible several overlapping grids and different resolutions the privacy peers would have to perform a significant amount of equality checks for each buddy check (number of grids times number of resolutions). Also finding a working mapping with overlapping cells is no trivial task and simply out of scope of this work.

In case of the *Discrete Location* it was possible to get a sample of the Gowalla [3] locations

as a SQL dump which assigns a unique integer for each location. This requires only one equality check on the privacy peers and simplifies the implementation greatly.

Regardless the choice if the implemented protocol of either definition works the other one will also work since the simplest version of *Cell Map* (one resolution, one grid) requires the same checks on the privacy peers as *Discrete Location*.

5.3 Projects and Code Breakdown

The main requirement regarding the implementation was to use the Java SEPIA library with as little modifications as possible. This requirement is necessary to not break the existing SEPIA protocols or create a separate Android branch of the SEPIA library. To ensure this compatibility the first step was to get a SEPIA tutorial peer running in an Android environment and then gradually adjust towards the desired design.

Eventually adjustments and cuts to the desired design were necessary because it was either in conflict with the existing code or would simply demand too much time. This section will therefore shortly describe the functions of the application's main parts and point out the differences to the previously described design.

5.3.1 Code Base

The code is organized in three projects.

1. The already existing Java SEPIA library [5] contains the basic functionality such as peer start up for a Java peer, connection handling, base operations. Here only structural but no functional adjustments were made to make an override easier.

2. The Friend Locator Java library which uses the SEPIA library. It contains all the protocol relevant code such as the input and privacy peer, protocol resp. listeners, message and data container. This data container acts as a shared object between Android and the SEPIA input peer.

3. The Friend Locator Android application which uses the Friend Locator library. It not only consists of all the Android relevant code but also holds the necessary code overrides of the SEPIA library. These overrides are required because the Android runtime environment is different from the Java runtime environment. The project also holds all the necessary information for the SEPIA input peer (config file and certificate key store) and the SQLite database of the Gowalla locations.

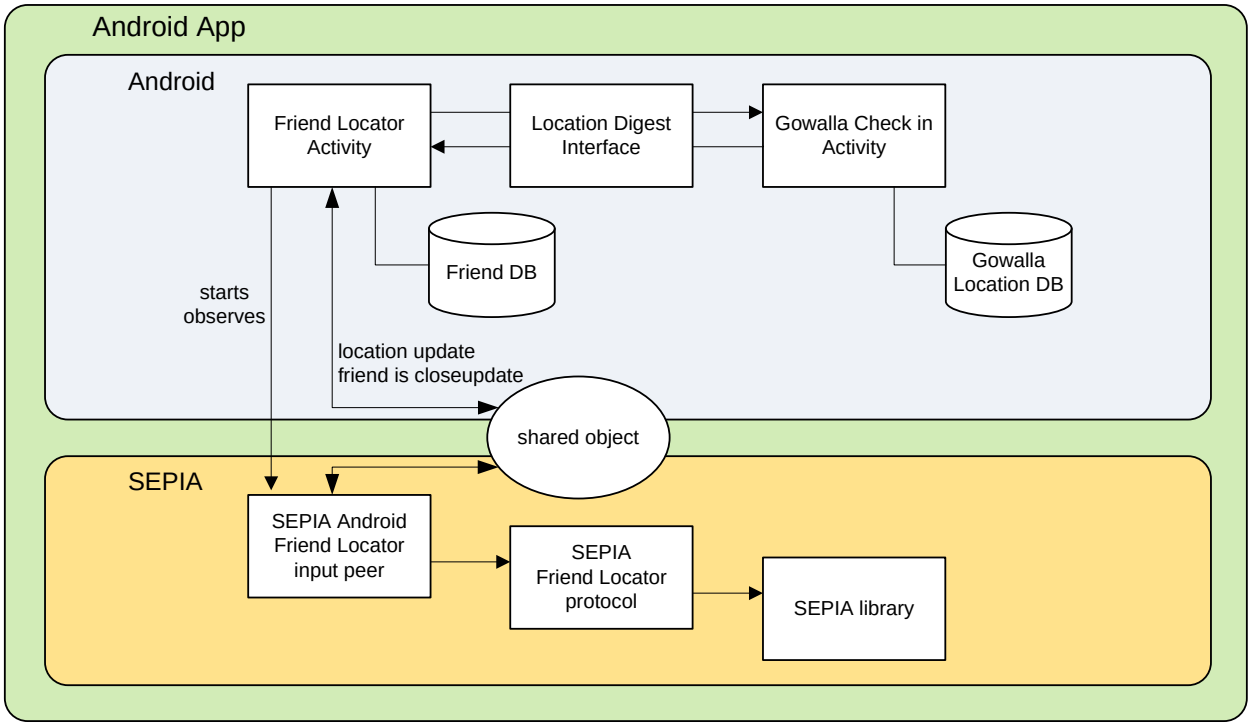


Figure 5.1: Functional Android Friend Locator design

5.3.2 Android Application

The resulting Android application (apk) holds several different main functionalities (see figure 5.1). The Android part is again split in two activities. The Friend Locator activity provides the user with the basic overview and lets him check in to close by Gowalla locations. This starts the Gowalla check in activity which queries the local database and provides a list of close by locations available to check in. Upon check in the results are then returned to the Friend Locator activity which then forwards the information to the shared object. At the beginning it also starts up the Android SEPIA input peer. Upon notification the received results are processed and the list of close friends is updated. As a second part the necessary SEPIA functionalities for the Android input peer are included. The functionality is described in the following section. And lastly it holds all the additional data for the application (Gowalla database, SEPIA config file, SEPIA certificate key store).

5.3.3 Android Input Peer

The SEPIA Android input peer is started by the Friend Locator activity in the onCreate event. This start up is not performed using the standard SEPIA MainCmd and PeerStarter code since the Android RE behaves differently. The start up is executed using the MainCmdAndroid and PeerStarterAndroid. In the process the Friend Locator activity is set as an observer of the peer

starter instance. During start up a listener thread for each privacy peer is created. These threads listen for protocol messages and forward them to the input peer. When new location data is set in the shared object a function of the input peer is triggered it creates shares which are sent to the privacy peers. Upon receiving a friend update the input peer will set the information in the shared object and notify the the starter which subsequently notifies the Friend Locator activity.

Even though the input peer code resides in the Java project the input peer cannot be started as a standard Java SEPIA input peer. This is due to the fact that the shared object is created by the Friend Locator activity and set by the PeerStarterAndroid instance. Without the shared object the input peer is not working. A standard Java SEPIA input peer uses the start up procedure given in the SEPIA library.

5.3.4 Privacy Peer

The SEPIA privacy peer is the component that kept most similarity to the its previous tutorial version. The main difference is that the privacy peers don't reinitialize completely after every computation round. The received location information is kept locally for each input peer until the next update. The privacy peers maintain a listening thread for each input peer waiting for a location update. Upon receiving an update an isFriendClose check is triggered over a specified set of friends.

5.3.5 Friend Locator Protocol

In the tutorial protocol the tutorialPeer2PrivacyPeer and tutorialPrivacyPeer2Peer classes specified when which peer is sending or expecting which information. With the change to an asynchronous protocol between the input and privacy peers this is no longer necessary since the input peers can receive information at any time. The protocols therefore were redesigned to listener threads that periodically check if a message was received.

The computational protocol between the privacy peers remained mostly unchanged and is therefore synchronous. This privacy peer to privacy peer protocol (pp2pp protocol) specifies which functions are executed during the isFriendClose check MPC (prepare location data, equality check, prepare results, reconstruction, return results).

5.3.6 The isFriendClose Check

This isFriendClose check, which is mentioned several times throughout the report, is a functional collection of several functions residing in different classes, that together perform the task of checking whether friends are close or not. The pseudo code below describes the performed sub tasks during the check. After the pseudo code is a brief description in which classes and functions these sub tasks are implemented.

Pseudo Code

```
create a list of all friends with valid location information
if the friends list contains two or more friends then
  for all friend pairs possible by the friends list do
    fetch the location Shamir shares for both friends from the cache
    schedule a SEPIA equality check
  end for
  execute all the SEPIA equality checks in parallel
  for all SEPIA equality check results (i.e. Shamir shares) do
    fetch share from equality check result list
    schedule a SEPIA reconstruction
  end for
  execute all the SEPIA reconstructions in parallel
  for all friend pairs do
    write boolean results from the reconstruction result list to a result matrix
  end for
  send friend is close update to all friends for which a check was performed
end if
```

The isFriendClose check starts in the notificationReceived function of the privacy peer. After the suffiecientSharesForComputationReceived function creates the friends list and checks if enough data for the computation is available the pp2pp protocol is started. The pp2pp protocol specifies which MPC functions the privacy peer has to execute during the MPC. At the end the sendResultsToPeers function of the privacy peer is called to send the results back to the input peers.

5.4 Challenges

During the implementation many obstacles were encountered that had to be overcome. Several have already been briefly mentioned in the previous section and might be mentioned here again for completeness.

5.4.1 Android Runtime Environment

The Android runtime environment is different of the Java runtime environment. It does not necessarily support exactly the same functionalities as the Java RE. For example Android supports the BouncyCastle key store instead of the Java key store. Android also evolves very quickly which does not only fix problems. In the current version 2.3.6 the WiFi connection is not very stable. Prior to version 2.3 all included files larger than one MB must be compressed. This does not apply

to either badly compressible or highly compressed file types such as jpg pictures. The easy work around usually is to simply rename the file. Starting version 2.3 it was decided that this constraint should be removed. Also very special are the Android access rights. These access rights are specified in the application manifest and required to gain access to system components such as GPS, network, SD card etc. This list is also shown to the user previous to installation requesting the permission. During development these access rights can lead to wasted time since even if all the code is working properly the application will not run due to missing access rights. Last but not least the runtime environment can behave very different because it is designed to run on a mobile device. Android might decide to pause the application or even if the activity is closed the process still exists and the listening treads are still running and don't terminate because they are not woken up to terminate themselves.

5.4.2 Gowalla Database

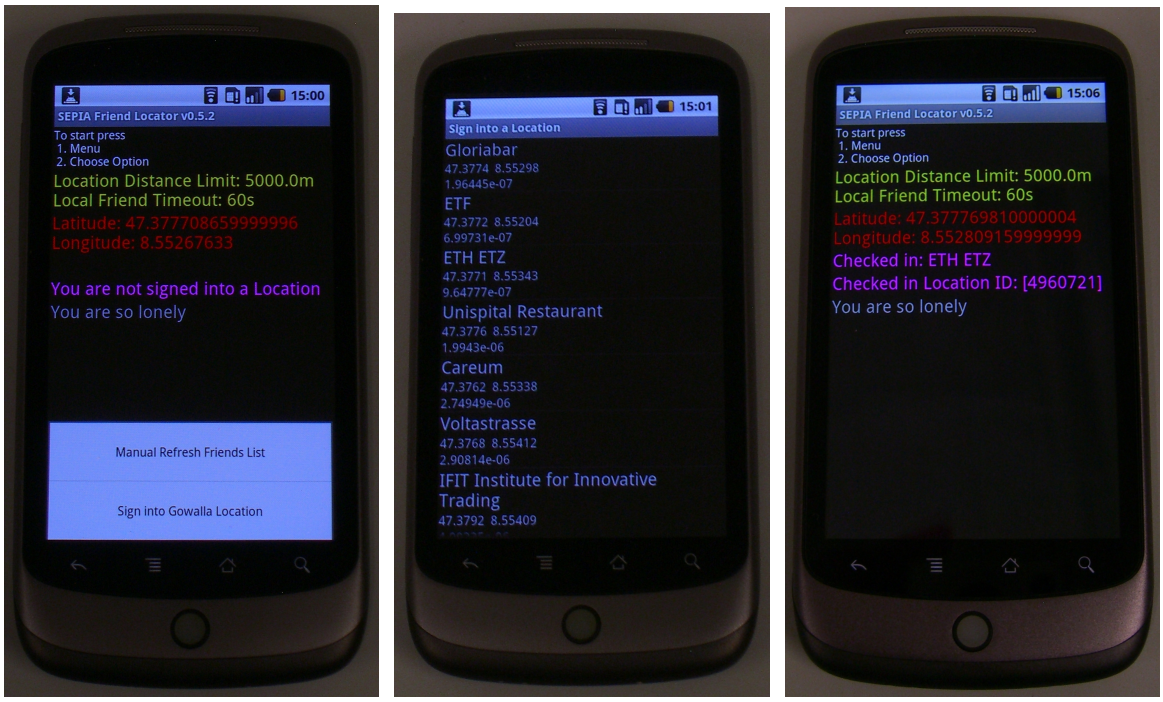
The Gowalla location MySQL database dump has a size of roughly 120 MB and contains nearly 1.7 million entries. For a mobile application this is quite a lot. To reduce the size the contained locations were limited roughly around Switzerland effectively limiting the latitude between 45 and 48 degrees and longitude between five and eleven degrees. This step reduced the size to a little bit more than two MB containing 29028 locations. Android actually does not support MySQL it rather supports SQLite. SQLite has a quite limited set of functions and uses different command character set. Therefore before importing the dump had to be adjusted. After the import of the data an Android specific table hat to be created where Android will later store its metadata.

5.4.3 SEPIA

In context of developing, the asynchronous protocol design was probably the greatest challenge since some basics are defined in the SEPIA library that should not undergo major reworking because of this new protocol. But also the design of the input and privacy peers, which now run in different runtime environments and yet share a common code base, was very challenging. Last but not least the specification of a communication interface between the SEPIA input peer and the Android activity was quite tricky, because neither the input peer nor the Android activity should have to know details of its counterpart.

5.4.4 Performance Evaluation

The performance evaluation remains future work due to the limited time. But it is quite safe to make certain estimations with regard to the performance. Due to previous work with SEPIA it is known that up to over two thousand equality checks can be performed per second [5]. In the current implementation the required checks are quadratic to the size of the set. With optimizations



(a) not checked in (b) Gowalla location check in (c) checked in

Figure 5.2: SEPIA Mobile friend locator Android UI

(e.g. recheck only after certain timeout, etc.) and the assumption that the updates are evenly distributed the required checks could be lowered to linear to the size of the set. Based on this, the SEPIA peers could already handle quite a large set and the SEPIA privacy peers are unlikely to become the bottleneck of the application, as long as the set does not grow too large.

5.5 Application Demo

Finally I would like to give a short overview of the application from the users point of view. The demo is shown on three Google Nexus one devices. These devices are supposedly used by three friends (friend one, friend two and friend three, from left to right). Friend one and two have the Android version 2.2.1 and friend three 2.3.6. This explains the sometimes different look of the interface. Friend one and two were not upgraded since version 2.3.6 has some WiFi connection problems. The location where the demo takes place is the electrical engineering building ETZ of ETH Zurich.

In figure 5.2 you can see the different steps of the check in on the Android application. Figure 5.2(a) shows the display of a just started friend locator application. On top it displays the current settings. In this case the locations displayed to check in are limited to a maximal distance of 5

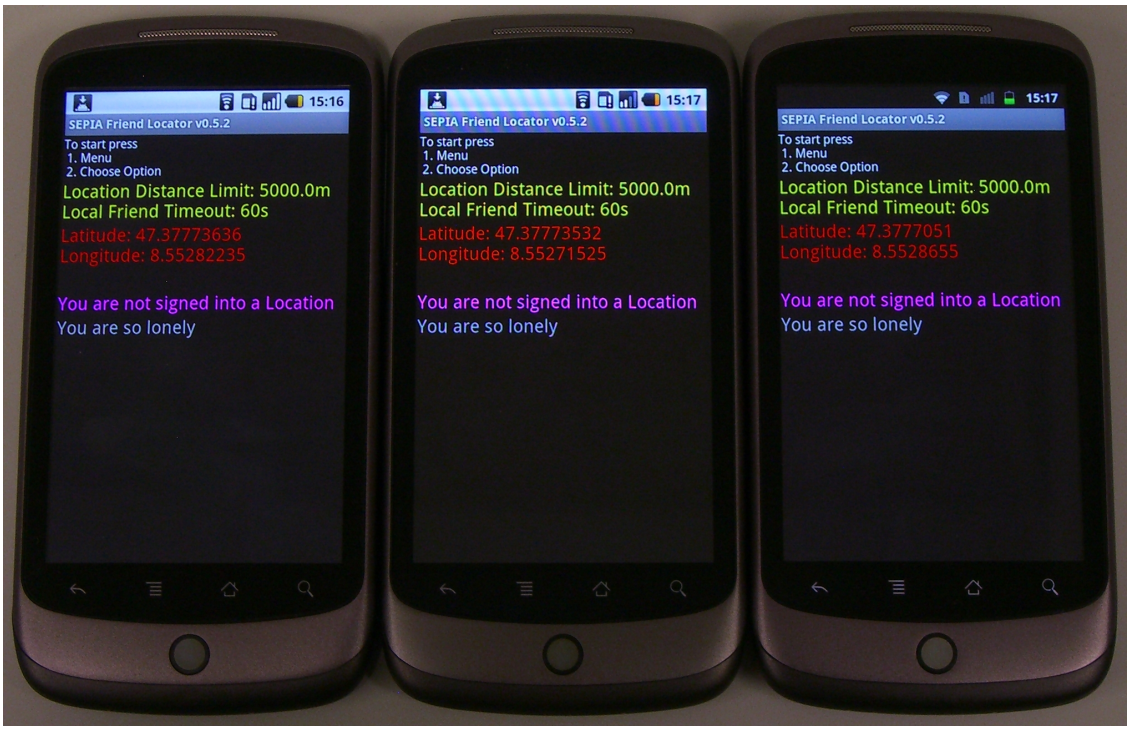


Figure 5.3: No device checked in in any location

kilometers and friends for which no update was received for longer than 60 seconds are timed out. Below (in red) the precise location of the device is shown. Upon pressing the menu button and selecting *check in* the *Gowalla activity* is started and shows the nearby locations as seen in figure 5.2(b). It displays the name of the location and below the coordinates of that location. The value in the third line represents a distance measure to that location. To check into a location simply press on the location and check in. This closes the *Gowalla activity* and returns the user to the *friend locator activity* which now shows the name and location ID (see figure 5.2(c)) of the checked in location.

In figures 5.3, 5.4 and 5.5 one can observe how the list updates when friends move. First nobody is checked into any location. Therefore the list shows the standard message when no friends are close, *You are so lonely*. Then, everybody goes working and checks into ETH ETZ which is the electrical engineering building. This can be seen in figure 5.4 where all the friends are checked into the same location. The friends list shows the close by friends with the name and below the unique identifier and the time stamp it received the update. Then, *Friend one* decides to make a break in the location Gloriabar and updates his location. The service propagates this information to all the other devices. *Friend one* is now lonely again in the Gloriabar were as *friend two* and *friend three* are still checked in the same location. If now *friend two* and *friend three* join *friend one* the devices would show the same view as in figure 5.4.

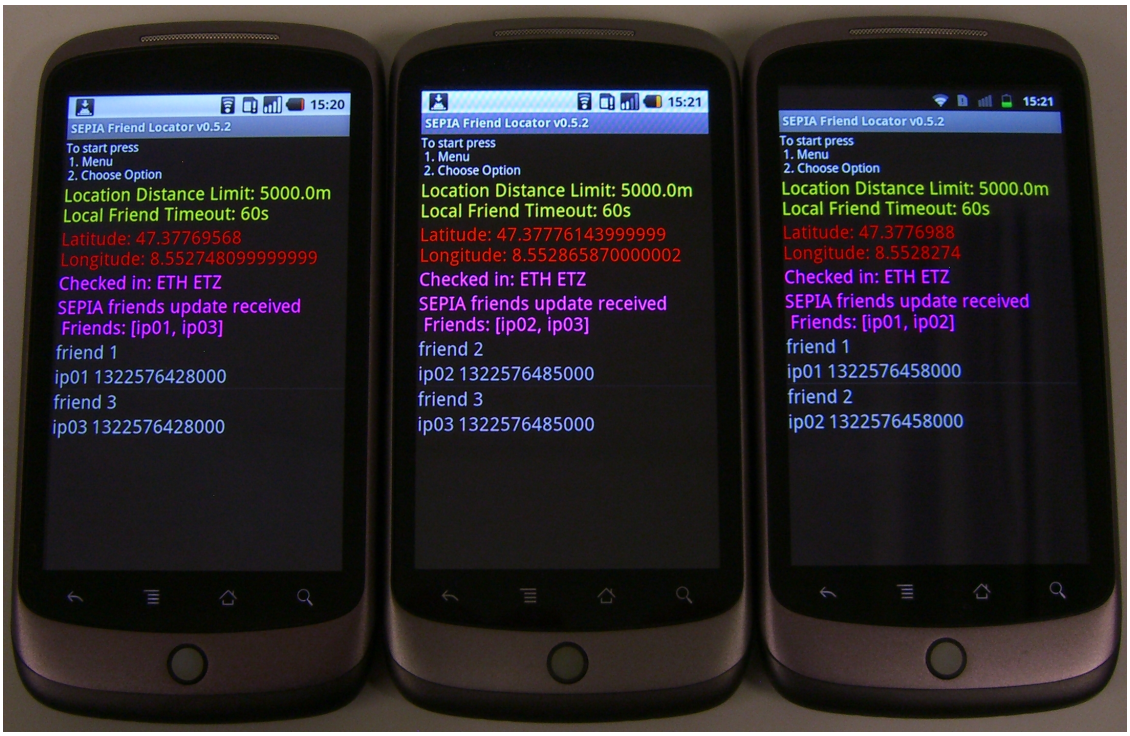


Figure 5.4: All devices checked in the same location

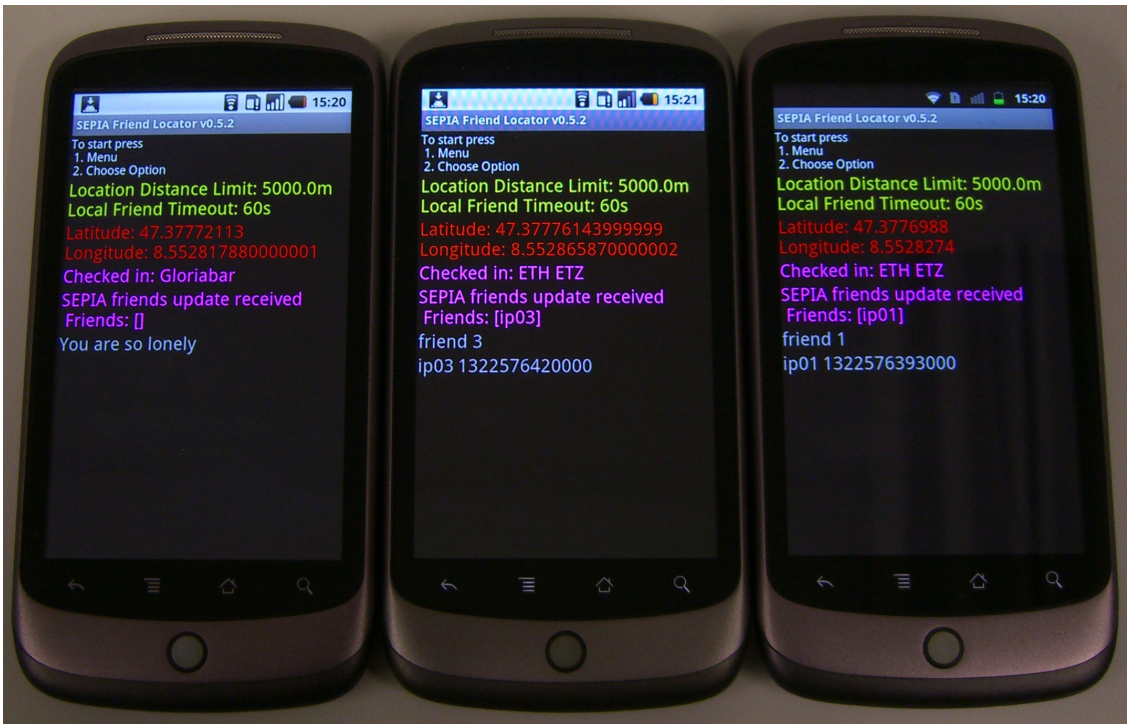


Figure 5.5: Friend 1 changed its location

Chapter 6

Conclusions

Privacy

Privacy problems regarding mobile devices are of growing concern as already mentioned in chapter one. As an example of such a privacy sensitive application we chose a friend locator app. This app provides the users with information about the whereabouts of their friends. In the service design of popular applications we detected the centralized service provider as the main privacy breach since it has access to all the location information of each user. But one might also want to keep a certain minimum privacy with regard to ones friends. In chapter three we therefore had a look at the problem of privacy preserving location definitions. Two possible approaches were discussed both with distinct advantages and disadvantages.

Application

Based on this information we built a privacy preserving friend-locator application. To actively preserve the privacy the user must check into a location. The size of the location provides a certain minimum privacy with regard to friends. The central service provider was replaced with SEPIA privacy peers therefore protecting the privacy of the user. Each privacy peer cannot extract any information of the given share. Only a majority of the privacy peers together are able check whether two users are in the same location or not. But why should it be better to trust many privacy peers than one central service provider? First it is not necessary to trust all the privacy peers as long as the user trusts enough of the privacy peers for the protocol to work. Second he does not need to put the same amount of trust into a privacy peer as he would have to put into the central service provider. He only has to trust that the privacy peer will not share the information with any other privacy peer and follows the protocol (semi-honest). This works best if the privacy peers are reluctant to trust each other and therefore unwilling to share their information.

The current prototype proves that the concept works. It is still very far from an Android Market ready application. This is especially due to the fact that the administration layer is not present.

Because of that many variables and files had to be directly included in the application. This information could otherwise be provided dynamically.

Android and Java

First getting in touch with the Android environment is very difficult until you know how the basics work. The tutorials on the Android developer page help greatly with that but it is sometimes very hard to find help concerning some more advanced problems. For starters also no mobile device is required because the SDK offers Android virtual devices that can be set up with the right version and subsystems. The main disadvantage of the simulator is that it is quite slow even if it is possible to speed it up with the right settings.

Porting a Java application designed for the Java runtime environment to the Android RE seems to be viable as long as the mobile device can fulfill the system requirements. But during the port process one should really rethink the application design because the Android RE might not behave as expected. Depending on the application it is possibly easier to design the application from scratch rather than refit it for Android.

Future Work

One of the main parts required to elevate this application to a real world app for the Android Market would be the administrative service that provides all the required information to the users and privacy peers. Additionally the user interface could be more appealing to the users (i.e. notification). Many smaller changes (e.g. usage of a background service etc.) could greatly improve the resilience and reduce power consumption. On the privacy peer side the required checks can be reduced with rechecking only after a certain time. Also with many small changes the resilience could be improved. As already mentioned, also a detailed performance evaluation has yet to be performed.

Bibliography

[1] https://www.google.com/latitude, visited Aug 2011.

[2] foursquare. https://foursquare.com/, visited Aug 2011.

[3] Gowalla. http://gowalla.com/, visited Aug 2011.

[4] Military grid reference system. http://en.wikipedia.org/wiki/Military_grid_reference_system, visited Aug 2011.

[5] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *19th USENIX Security Symposium*, Washington, DC, USA, August 2010.

[6] Sergio Mascetti, Claudio Bettini, and Dario Freni. Longitude: Centralized privacy-preserving computation of users' proximity. In *Proceedings of the 6th VLDB Workshop on Secure Data Management*, SDM '09, pages 142–157, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] Sergio Mascetti, Claudio Bettini, Dario Freni, X. Sean Wang, and Sushil Jajodia. Privacy-aware proximity based services. In *Proceedings of the 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, MDM '09, pages 31–40, Washington, DC, USA, 2009. IEEE Computer Society.

[8] Arvind Narayanan, Narendran Thiagarajan, Michael Hamburg, Mugdha Lakhani, and Dan Boneh. Location Privacy via Private Proximity Testing. In *NDSS '11: Proceedings of the 18th Annual Network & Distributed System Security Symposium*. The Internet Society, February 2011.

[9] Peter Ruppel, Georg Treu, Axel Küpper, and Claudia Linnhoff-Popien. Anonymous user tracking for location-based community services. In *LoCA*, pages 116–133, 2006.

[10] Laurynas Šikšnys, Jeppe R. Thomsen, Simonas Šaltenis, Man Lung Yiu, and Ove Andersen. A location privacy aware friend locator. In *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases*, SSTD '09, pages 405–410, Berlin, Heidelberg, 2009. Springer-Verlag.

[11] Ge Zhong, Ian Goldberg, and Urs Hengartner. Louis, lester and pierre: three protocols for location privacy. In *Proceedings of the 7th international conference on Privacy enhancing technologies*, PET'07, pages 62–76, Berlin, Heidelberg, 2007. Springer-Verlag.