Project Title

Distributed Systems – Project Proposal

Valentin Trifonov
ETH ID 13-941-679
vatrifon@student.ethz.ch

Simon Ringeisen
ETH ID 13-934-930
rsimon@student.ethz.ch

Jan Eberhardt
ETH ID 13-925-417
ebjan@student.ethz.ch

Caroline Creidenberg
ETH 15-907-421
ccreiden@student.ethz.ch

Fabian Ulb
ETH ID 13-931-951
fabianu@student.ethz.ch

Felix Wolf
ETH ID 13-927-983
fewolf@student.ethz.ch

ABSTRACT

In this document we are describing the initial planning of an Android app, its purpose is to serve as a quick and easy way for people in an emergency situation to call for help to people within their vicinity. The goal of the app is that the people in the surrounding area will be able to receive the emergency alarm through peer-to-peer or a server based system and respond to the emergency faster than any official emergency system would be able to.

1. INTRODUCTION

Traditional emergency solutions tend to have one issue making them unsuitable for many modern emergency situations, that is the need for a longer conversation about the problem with an officer and the long response latency. In some situations, much faster help is needed. We try to tackle this problem by providing an application which is very easy and fast to start, issuing an alert to people nearby informing them of the emergency as well as the location, and therefore enabling people helping people where needed, even in situations where an internet connection might not be available.

As good as this sounds, there are a number of challenges to solve for this, particularly concerning fast and reliable multicasts and consistency for proximity-based communication.

First of all, there is the problem of reliability - in an emergency situation, you want to have your alert reaching other people by any means possible. We try to tackle this by using multiple different means of communication and detection: First of all, with a working internet connection, we will pursue a centralized approach, a firebase database to store the locations of people and a server to evaluate the location informations and send push notifications via Google Cloud Messaging (GCM) to notify people in the vicinity of the alert origin. Firebase is a scalable real-time database which was originally developed by an independant company and later got bought by google [3]. We choose this approach for scalability reasons. See section 2.1 for more details. This provides a large range as well as a reliable way of communication.

Secondly, without a working internet connection, we take a P2P approach. There is p2pkit [14], a framework developed by a spinoff of the ETH, Ueppa, which is developed exactly for proximity-detection, and works even without an internet connection. Though this is a nice approach, the current development status of this framework might not be sufficient for our requirements. We started a first series of tests with modest results but we will evaluate our options here further. One consideration is to replace p2pkit with the Android Wifi-P2P framework [1] or use it as a supplement. In case we arrive at the decision that it is insufficient we will focus ourselves on the centralized approach and take P2P as a best effort offline approach. Consult Section 2.2. for more details.

The combination of these approaches should hence provide reliability on the edge whatâĂŹs possible with currently available technology.

2. SYSTEM OVERVIEW

It's important to remember that we have two app backends that essentially do the same thing. They can be thought of as two different channels to communicate. If one is not available, we will just try to use the other option and if both are available that is fine as well.

2.1 Centralized approach

As mentioned in section 1 the centralized approach uses a central server and a database. All clients, meaning all phones on whom our app is running, register by sending their ID to the App Engine Server [8]. They get back a key, which they can use to write to the database and write their location into. This location doesn't need to be accurate, it just needs to be a rough approximation. Also this location must be updated from time to time, this could be every hour, twice a day or even only daily. This allows us to store a rough location estimation of all the clients. Now if one client needs help, he sends an alarm request to the server. This request also contains his/her exact current location. The server then goes through the database and finds clients that are somewhat in the near. This could be 10 km or 100 km or even more depending on the update interval of the locations. Then the server sends a push notification via Google Cloud Messaging (GCM) to the clients which are in this larger range. See [4] for Firebase with Google App Engine and [2] for GCM push notifications. Then the clients can decide whether the request is not too old (as GCM's may be delayed when the client was offline) and if the user who needs help is within a useful range. It then connects to the firebase and looks to see if someone who is closer to the person who needs help has already taken action. If not, the app starts an alarm. As soon as the user decides to help, the app starts putting his/her current location into the server. Like this, the other users can see, whether someone has already taken action or not. The alarm can be revoked by inserting a corresponding message into the firebase.

Why don't we use something like Google Nearby? With Google Nearby Messages one has a range restriction (approximately 30 meters) [13]. Since this is not enough for our purpose and we want to have flexible control of the range in which we want to alert people, we will build our own approach there. Therefore our system gets highly dynamic and would theoretically also work for any other kind of proximity based application. Why do we use firebase and GCM and not our own simple server with, for example, simple MySQL database? We want our overall service to be reliable and highly scalable. We want to build a system which could theoretically be used by thousands of people. With firebase we are safe on the database side, because they claim

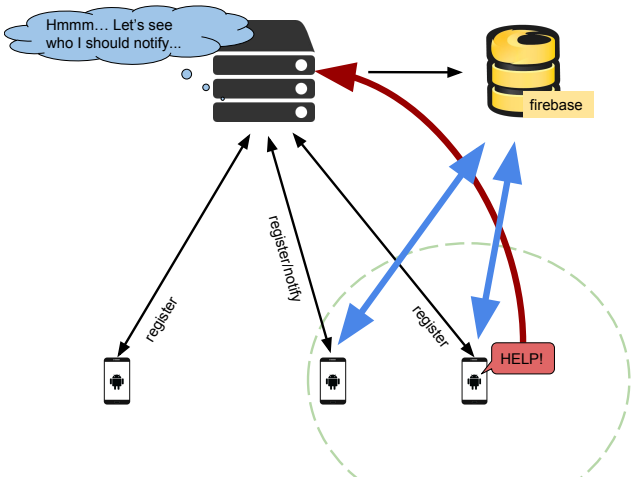

Figure 1: Server Structure

they can handle millions of concurrent connections. Also we do not need to worry about security issues, encryption etc. We could theoretically always have an open connection to the firebase database directly from each client. This would require much energy according to [15]. Therefore we rely on the GCM service there and send push notifications from our Google App Engine [8] server via GCM to the client. Also with Google App Engine we do not need to worry about security and scalability issues. There are neat APIs for Android [5] and a REST Interface for the Server [6].

2.2 Decentralized approach

The Decentralized approach is currently based on the p2pkit. It can be divided in two parts, an active and a passive mode. In the passive mode each device has to keep track of the available peers in its vicinity. This gets implemented with a List, where discovered peers are inserted and lost peers are removed. Depending on which P2P technology we utilize, it could be that we can only deal with approximate times of actual existing peers.

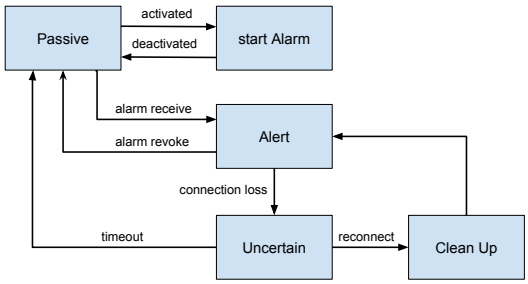

Figure 2: Peer Logic

The App switches to active mode if an alarm is received or sent. If an alarm was sent, then the List of available peers is used to send them alarm messages. After an alarm was sent the user is given the possibility to revoke the alarm. The messages contain a source ID (UUID) and some information about the source location. If an alarm is received, the source id is stored until the alarm gets revoked. As long as the device has a connection to the device from which it got the source ID, it sends alarm messages to its available peers. To limit the number of resends, the devices keep track of the length of the path from the actual source to the device and if a certain threshold is reached it won't send any further messages. If a revocation with the source ID is received, it sends revocation messages to all its peers and goes back to passive mode. If the connection to the device from which it got the source ID is lost, it also won't send further alarm messages. If the connection with an involved device can be obtained, the device has to inquire if the alarm was not revoked, else the device will go back to passive mode after a certain timeout. In all states the user gets the possibility to return to passive mode.

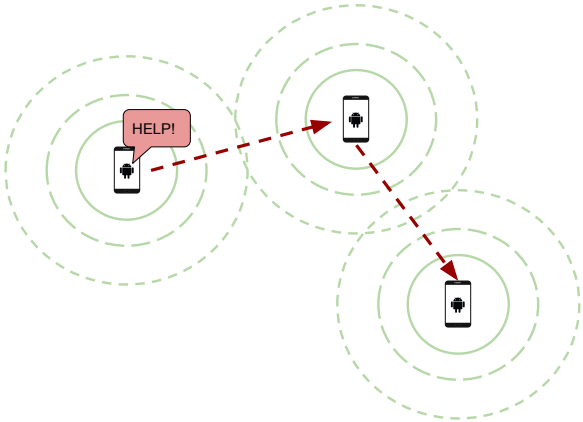

Figure 3: Peer Connection

Here the main challenge is to achieve consistency and fast communication between all devices with a reasonable battery usage. A first series of tests has shown certain issues in this domain, especially concerning the consistency.

2.3 App Overview

The picture below is a general overview of the system from an Android perspective and how we plan to implement the two separate channels.

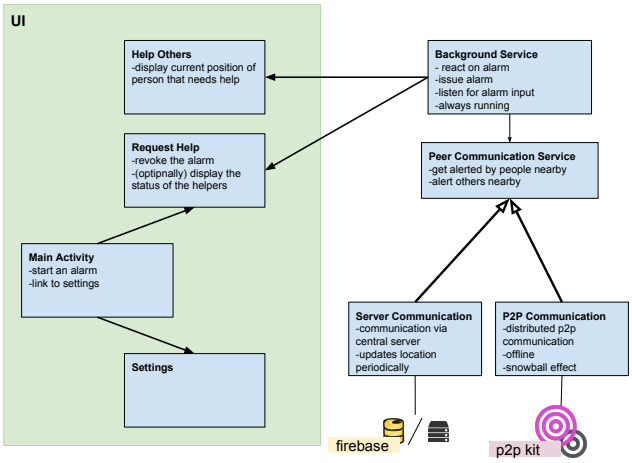

Figure 4: App Overview

2.4 UI

The UI is pretty simple and straightforward. We have a Main Activity where you can start the alarm, turn the service as a whole either on or off (this means you won't receive alarms if it's off and you also will not be able to start

an alarm) and a button to open the settings. The following settings will be offered: the user can decide which button combination can trigger the alarm, turn off the online version and work with the p2p version only (this will only be implemented if the p2p communication will be sufficient) and enter a custom message which will be displayed to persons that get the alarm. If the user triggered the alarm he gets to a screen where he can abort or revoke said alarm and see the status of surrounding people, people on the way etc. (this is an optional feature). If somebody receives an alarm, there will be a standard android notification and by clicking on it, the user will get to a screen where he/she sees a map with his/her position and the location and message of the person in need. He/she is then able to either accept or dismiss the alarm depending on their ability to help or not.

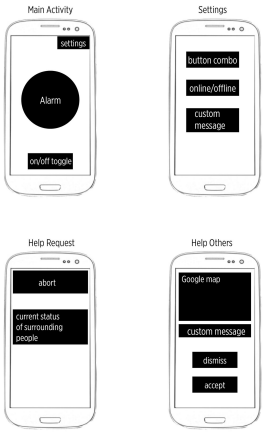

Figure 5: User Interface

3. REQUIREMENTS

Because of a potential use of Bluetooth Low Energy we require minimum sdk 18. We will be testing our project with Nexus 5, LG G3 and Nexus 6. In the case that the p2pkit integration is successful, we will furthermore require that the p2pkit (P2P Services by Ueppa AG) is also installed on the device and that bluetooth and wifi are enabled, not necessarily connected, while the app is in use. The server implementation requires an internet connection and also the location service of android should be turned on. Libraries we will use:

1. Google App Engine [8]
2. Google Cloud Endpoints [9]
3. Google Cloud Endpoints Android API [10]
4. Google Cloud Messaging [11]
5. Google Cloud Messaging Android API [12]
6. Firebase [3]
7. Firebase Android API [5]
8. Firebase token generator [7]
9. p2pkit [14]
10. potentially Android Wifi P2P [1]

4. WORK PACKAGES

Breakdown the work to subtasks to meet the project requirements. Define and describe these tasks.

1. **GUI & Activities**
   - 1.1. **implement basic GUI**
   - 1.2. **Structure the project**
     provide interfaces for the different work packages
   - 1.3. **(Optional) Alarm Feedback & Helper Proximity**
     Add a feature where the one needing help receives feedback about people on their way to him and their distance
2. **P2P backend**
   - 2.1. **P2P Test Phase, Basic Functionality**
     test if the p2pkit is applicable for our needs (in terms of speed and reliability). If it is not, we will not pursue this option further
   - 2.2. **Discovery Logic**
     implement discovering nearby people, keep track of peers in your vicinity
   - 2.3. **Alarm & Revocation Logic (Snowball effect, Broadcast)**
     implement sending and revoking alarms to nearby people and reacting to these events
3. **Centralized Backend - Client Side**
   - 3.1. **Database Setup (Firebase)**
     integrate a firebase db into our app, periodically update user location in the database
   - 3.2. **GCM integration**
   - 3.3. **Server Interface**
     Communication to the server and reacting to push notifications (send current location, start alarm)
   - 3.4. **Alarm logic**
     check if alarm is still valid and user is in a useful range, react to alarm if nobody else already did, handle reaction of user
   - 3.5. **(Optional) Alarm Feedback & Helper Proximity**
     Add a feature where the one needing help receives feedback about people on their way to him and their distance
4. **Centralized Backend - Server Side**
   - 4.1. **Implement server and integrate firebase db**
   - 4.2. **GCM integration**
   - 4.3. **Alarm Logic**
     react to user alarm events and find nearby people
   - 4.4. **(Optional) Alarm Feedback & Helper Proximity**
     Add a feature where the one needing help receives feedback about people on their way to him and their distance
5. **Miscellaneous**
   - 5.1. **Alarm Activation Logic**
     implement a listener which monitors button events for a special combination and then triggers the alarm

5.2. **always-on Android background process & IPC**
figure out how to have the background code running in a separate process and how to communicate to it from the activities

5.3. **(Optional) consider ways on how we could reduce spam alerts**

5. MILESTONES

1. **Implement the basic building blocks - by 2.12.2015**
Caroline: WP1.1, WP1.2
Felix, Fabian: WP2.1, WP2.2, WP2.3
Jan: WP3.1, WP3.3
Valentin: WP3.2, WP3.4
Jan and Valentin: WP5.2
Simon: WP4.1, WP4.2, WP4.3

2. **Code review - continuously**

   2.1. **We review our code in pairs (Caroline/Simon, Felix/Fabian, Jan/Valentin)**

3. **Put the basic building blocks together - by 8.12.2015**

4. **Test, fix and optimize - until 18.12.2015**

   4.1. **WP5.1 done by Caroline**

5. **Consider adding some of the optional stuff**

   5.1. **WP1.3**

   5.2. **WP3.5**

   5.3. **WP4.4**

   5.4. **WP5.3**

6. REFERENCES

[1] Android Wifi Peer-to-Peer. `http://developer.android.com/guide/topics/connectivity/wifip2p.html`. Accessed on 12 Nov 2015.

[2] "App Engine Backend with Google Cloud Messaging" Template. `https://github.com/GoogleCloudPlatform/gradle-appengine-templates/tree/master/GcmEndpoints`. Accessed on 12 Nov 2015.

[3] Firebase. `https://www.firebase.com/`. Accessed on 12 Nov 2015.

[4] Firebase App Engine Tutorial. `https://www.firebase.com/blog/2015-10-01-firebase-android-app-engine-tutorial.html`. Accessed on 12 Nov 2015.

[5] Firebase Java API. `https://www.firebase.com/docs/android/api/`. Accessed on 12 Nov 2015.

[6] Firebase REST API. `https://www.firebase.com/docs/rest/`. Accessed on 12 Nov 2015.

[7] Firebase Token Generator. `https://github.com/firebase/firebase-token-generator-python`. Accessed on 12 Nov 2015.

[8] Google App Engine. `https://cloud.google.com/appengine/docs`. Accessed on 12 Nov 2015.

[9] Google Cloud Endpoints. `https://cloud.google.com/appengine/docs/python/endpoints/`. Accessed on 12 Nov 2015.

[10] Google Cloud Endpoints Android API. `https://cloud.google.com/appengine/docs/java/endpoints/consume_android`. Accessed on 12 Nov 2015.

[11] Google Cloud Messaging. `https://developers.google.com/cloud-messaging/downstream`. Accessed on 12 Nov 2015.

[12] Google Cloud Messaging Android API. `https://cloud.google.com/tools/android-studio/messaging/add_module`. Accessed on 12 Nov 2015.

[13] Google Nearby Messages. `https://developers.google.com/nearby/`, see youtube video at start page. Accessed on 12 Nov 2015.

[14] p2pkit by Uepaa. `http://p2pkit.io/`. Accessed on 12 Nov 2015.

[15] Stack overflow. `http://stackoverflow.com/a/27325402`. Accessed on 12 Nov 2015.