Final Report

Project Title:
A Location-Based Time Capsule Application on Android

Team Members:
Meng Cheng, Qin Huang, Yiteng Zhang, Zhong Ren

I. Overview

Our location-based time capsule application is motivated by the time capsule in our real life. People can leave messages or photos inside a time capsule and bury it at a certain location, which is to be opened at a later time in the future.

Based on the real-life example, our application extends the concept of time capsule by integrating the social networking features, which enables users to communicate with other users via time capsules. In other words, our app allows users to define a time capsule in three dimensions: time, location, and user visibility. Users can store a text message inside a time capsule, and optionally specify some additional information about what time to unlock the capsule, at what location to trigger the capsule, and/or who will be notified about the capsule.

To provide a seamless user experience, our app client periodically updates the location of the user to the server for back-end processing. Once the user enters pre-specified location during the scheduled time of a certain time capsule, the device will automatically receive a push notification via Google Cloud Messaging (GCM) containing the content of the capsule (provided that the user has been configured as the receiver of the capsule).

Admittedly, the use of location service (especially fine-grained localization service like GPS) will unavoidably reduce the battery lifetime of mobile devices. To address this challenge, we propose an adaptive update mechanism to dynamically adjust the update interval according to the status of both the client and the server. We will introduce our approach in later section.

The following part of this report goes as follows. In section II, we describe the client part of our location-based time capsule application. In section III, we introduce the server of our app, which bridges the client and the GCM server. In section IV, we present the conclusion and some possible future work of our app.

II. Client

Our application’s client side basically has two parts: the user interface and the background service.
1. User Interface

Specifically, the user interface part can be separated as login session and main operating session. In the login session, there are three functionalities, login, register and setup IP address. After you finishing these preparations, you will enter the main operating session, where there are receive box and sent box for capsules as well as contact list and other settings.

Interacting with server using AsyncTask

The android class AsyncTask allows us to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers. We implemented a function searchRequest() to construct a http request to the server, and a function processResponse() with JSON parsing to deal with the response from the server. The task would go through two steps: doInBackground(Params…) would be first invoked, and call the request function searchRequest(). After the background computation finishes, onPostExecute(Result) would be invoked, and call the response function processResponse(). According to the response result from the server, the client side will choose corresponding action.

Easy, smooth navigation using Fragment

A Fragment is a piece of an application’s user interface or behavior that can be placed in an Activity. Basically our app has four fragments in the main interface. Using FragmentManager (Android’s method), we can switch between any of the four fragments easily. Also, we include the RadioButton widget to enhance the user experience.

Send capsule to other users from the contact list

An independent contact list is embedded in one of the four fragments. The contact list will automatically loaded with the data from the server when the user logs in. Not only can the user view his/her friends in the list, but also he/she can just click the “send” button after the friend’s name to send the capsule. The contact list is implemented by adding a composite widget ListView into the fragment layout file.

Directly selecting any location on the map

Android platform is bundled with Google Maps, which helps us to create this location-based application. You can search for the destination for the capsule in the search bar, or you can press the “self” button to locate your location. After this, you can now do more, that is, you can click anywhere on the map to select the new place to put your capsule. This is implemented by an Android method named GestureDetector. After you decide your location, you should press the “save” button to save this information and return to the capsule editing interface.

Conveniently select any date/time for the capsule
By including two time widgets, TimePicker and CalendarView, you can easily add your time choice by several touch actions on the screen. Also we consider the special case: if you press the “save” button to save the time information but you forgot to choose the date from the calendar or the time from the time picker, we will attach the current time to the capsule’s time information. Moreover, the default valid time of the capsule is one day. In other words, the GCM service would try to push the capsule to the receiver in one day until it succeeds.

2. Background Services

Besides the user interface visible to the users, our app also runs two background services: one is to update the current location of the device to the server, the other is to periodically wake up the first service when the app is running in background or when the screen is off.

Background service to update location of device

To let the server keep informed about each user using our app, the client periodically sends data to server and updates its location information. This behavior is achieved by running a background service. The service implements a LocationListener, which listens to the location events obtained by either GPS or network localization. Once the location is updated, the background service will broadcast a message containing the latest location of the device.

Our app also contains a location broadcast receiver, which handles this broadcasted location message, and then creates an asynchronous task to communicate to server to update its location. As provided by the Android framework, the location update only happens when the location listener detects any location change, which is fairly energy-efficient.

Background service to periodically wake up device

Although the above-mentioned background service can effectively update the device location, it only works when the application is running in foreground. To enable the app to update the device’s location even if the device is in sleep mode (the screen is off), we implement another background service to wake up the location update service even if the screen is locked.

We utilize the partial wake lock, one feature provided by Android, which can wake up the CPU to do the required processing. The wakeup background service periodically obtains a partial wake lock to run the location update background service, and then release the lock to put the device back to sleep mode. One novel part of our app is that the wakeup interval is dynamically adjustable according to the response from the server. The underlying the assumption is that if there are no potential capsules near the user, the wakeup interval should be longer. This feature significantly reduces the unnecessary CPU load and network usage, which makes our app energy-efficient. More detail on the adaptive behavior of our app is explained in next section.
III. Server

Our server is implemented based on the reference implementation provided by Prof. Yang in Homework 3. Basically, our server has three parts: (1) the application server to handle HTTP request, carry out corresponding actions, and return JSON response; (2) the MySQL database to store the persistent data; (3) the part to send notification via GCM server to the client.

1. Application Server

Based on the reference implementation provided by Prof. Yang in Homework 3, our server is able to receive http request from client and return JSON response to the client. The server is connected to a MySQL Database via Java Database Connectivity (JDBC), which stores the information of users, time/location capsules. As there are many forms of communication between server and client, we create a protocol that defines how client should contact with the server and exchange information. For example, the communication protocol for getting the contact list is shown as follow. The complete protocol we have implemented can be found in the file HTTP/JSON Protocol.pdf under the project directory.

Request: http://<server>/contacts?user_id=<userid>
Response: {"contacts":[{"id":1,"name":"zr36"},{"id":2,"name":"yry3"}]

The major functionality of the application server is to store the time capsule in the database, and, more importantly, to decide when to push GCM notification to the client based on the current time and the location of each client. To simplify our prototype implementation, we define two types of capsules: the time-based capsules scheduled to be pushed at a certain time, and the location-based capsules which are triggered when the receiver(s) enter a certain location.

To process the time-based capsule, our server has a pre-scheduled timer thread to periodically (every 20 second in our current implementation) search the database to see if there exist any time capsule whose start time has passed the current time. If so, for each matched capsule, the server will encapsulate the related information of a time capsule in database (i.e. name of sender, receiver’s device id, title, message content, etc.) to a TimeCapsule object and call the GCM part function to push the capsule to the receiver. Since the search is done periodically and the interval is very short, and the GCM push is instantaneous, the target user of any time capsule will receive the time capsule almost as soon as the current time reaches the sender’s predefined time in the capsule.

To handling the location-based capsule, however, there are more issues to be considered. First, unlike the time, the location of each device is entirely different, and has to be processed independently. Second, it is quite expensive in terms of computation cost to calculate the distance between the device and each location capsule using latitude and longitude coordinates.
Third, the location of each device has to be obtained via the HTTP requests, which causes the device to spend a lot of energy.

In order to address these issues, we use an adaptive approach to search for matched location capsules in the database. For each device running our app, whenever it updates its location coordinates, the server will add search the database using a square window. In other words, we would instantly calculate the upper and lower bound of latitude and longitude around device’s current location, and search for only those location capsules whose location is within that square. The key point is that the size of the square is not fixed, but varies depends on the search result. Suppose the initial side of the square is 1 km. If one device updates its location and the server found no time capsules around that device, the size of the search square will be doubled next time when this device reports location update. In the meanwhile, if the server found no location capsules nearby, it sends back a response to the device telling it a longer interval for location update. This mechanism improves the energy efficiency of the client app. On the other hand, if the server finds one or more location capsules within the square, it will cache these capsules in the server’s memory to enable faster matching in the future. (due to limitation of time, we didn’t finish the caching feature of our server). Because each location capsule might be defined to be visible within distinct size of ranges, these cached capsules are not pushed to the client unless the client device enters its own range. Once the location capsule is finally to be triggered, the server creates a new thread to push the capsule to GCM and in turn to the client.

2. Database Design

The server is connected to a MySQL Database via Java Database Connectivity (JDBC), which stores the persistent information of users and time/location capsules. The following figure shows the model of our database.
3. Google Cloud Messaging (GCM)

We use GCM (Google Cloud Messaging) to push capsules to users as cloud messages. GCM is a service that allows the server to push notification to the users' Android-powered device. This could be a lightweight message telling the user there is new data to be fetched from the server (for instance, a movie uploaded by a friend), or it could be a message containing up to 4kb of payload data (so apps like instant messaging can consume the message directly). The GCM service handles all aspects of queuing of messages and delivery to the target Android application running on the target device. GCM is fast, instantaneous, and energy-efficient. Since the server should push capsule to device via GCM server, the database connected with server should store device key for GCM of each user. We have both implemented GCM-related part in server and client. In server, we have a sender function to read in capsules as object and send them to corresponding device. In client, the app will check with GCM server that if the device has registered on login of user. If the device is not registered, device will register itself with GCM server and get a unique device key from GCM server. After a device key is obtained, the device will upload or update its device key with app server. The app server keeps the latest device key of each user.

IV. Conclusion and Future work

In conclusion, our location-based time capsule app on Android successfully delivers the functionality to define location-based capsules, send capsules to other users, and receive instantaneous notification of capsules via GCM.

In terms of future work, we could further optimize the location search to enable in-RAM caching of capsules and faster search. We also envision that our app could link with some SNS like Facebook, Twitter to enhance the social networking feature of the app. In addition, the app could be able to store photos or audio inside the capsule to enrich the user experience.