DISTANCE BASED APPLICATION TRIGGER

Mehdi Yosofie, Philipp Schlieker

January 24, 2019

CONTENTS

SUMMARY

Context awareness is an important issue in today's IoT world. Our project proposes a trigger that is activated based on the distance to a given object. In our scenario, we make use of the Bluetooth Low Energy (BLE) standard which allows the distance measurement between two objects. We use a BLE beacon together with the Bluetooth sensor of the smartphone in order to measure the distance and then trigger an API.

1 INTRODUCTION

The main idea of this project is to build an Android app that allows to trigger IoT devices based on position / distance thereby adding context awareness. One standard way to do this, e.g. employed by IFTTT, is the use of GPS or the connectivity of WiFi networks. Both of these technologies allow for great radius but are quite limited in their accuracy. Thus, we propose to use Bluetooth Low Energy to estimate the distance. This is done by using a beacon that continuously sends out a broadcast signal using Bluetooth. This signal is then received by a sensor which estimates the distance based on the signal strength. In our scenario we make use of a Raspberry Pi as beacon and API. The sensor is then provided by an Android phone.

2 PRACTICAL APPLICABILITY OF THE PROJECT

This type of project can be used in numerous scenarios. A few of which are the following:

1. **AUTOMATICALLY UNLOCKING A DOOR** Our prototype can be used to automatically unlock the front door of a house once the inhabitant arrives, and thereby replacing the normal key. In contrast to WIFI, the accuracy of BLE is high enough to *only* keep the door unlocked while standing right next to it.

2. **SWITCHING APPLIANCES** Another use case would be switching on and off appliances based on one's position in the room. One example would be a desk lamp which is switched on once the user is sitting at a desk and it is dark outside.

3. **SENSOR FOR SMARTHOME APPLICATIONS** Another use case is the use as a sensor for SmartHome Applications and thereby adding a wide range of possibilities. It is for example possible to switch on the light, adjust the room temperature, and turn on music once a person enters a specific room.

4. **PREVENTING MISUSE OF DEVICES** Another option would be to prevent the misuse of devices. This can be done by only allowing the invocation of a command when the person is in a specific location. This could be used by AirBnB to prevent guests from unlocking the front door when they are not there or by companies to enforce security policies.

3 CONCEPTUAL APPROACH & ARCHITECTURE

The used sensor is the Bluetooth sensor of an Android Smartphone. The Bluetooth Low Energy (BLE) Beacon is sending its signal constantly and periodically. The Smartphone (held by a human) comes closer to the Beacon and detects the beacon. As soon as the beacon is detected by the smartphone (by the human), it sends (the human automatically sends) a HTTP request to our web server to open the door or to turn on the light. Our project architecture can be seen in figure 1.

We simulated the Beacon furthermore by a NodeJS application on PC. Thus, debugging and testing was faster and possible on every machine.

The HTTP request arrives on the web server (a Raspberry Pi). The Raspberry Pi handles the request by activating the lamp or opening a lock via its GPIO.

The Raspberry Pi can be reached over a public IP address. We used Balena as deployment architecture for the Raspberry Pi. Balena [1] is a IoT cloud deployment platform where Balena images can be downloaded and flashed on the Raspberry Pi. The image auto-pulls the code which the developer pushes on his git repository. On Balena, the Raspberry Pi gets a public IP address. We installed Localtunnel [2] on the Raspberry Pi to resolve the IP address to a DNS name and have a nicer server name. There is no need of port forwarding.

4 SENSORS USED

We mainly used the Bluetooth sensor of the Android smartphone in order to detect the BLE beacon. Furthermore the WiFi card of the phone and Raspberry Pi were used for communication. The GPIOs of the Raspberry Pi are used for the action itself.

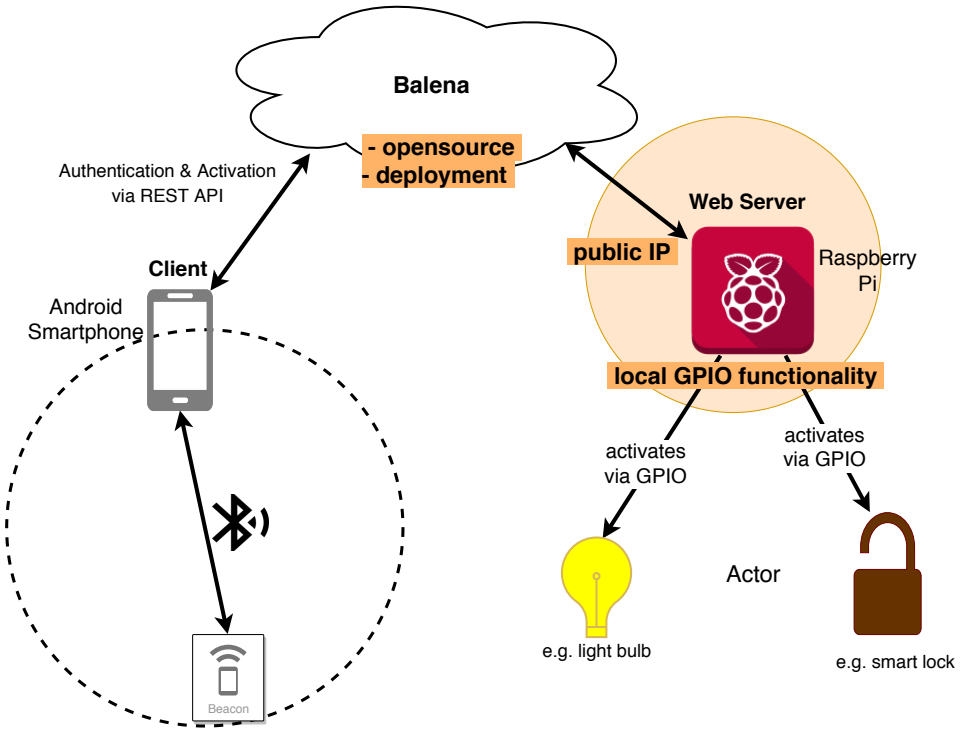

Figure 1: Project Architecture

5 PROBLEMS FACED

During the project, we faced various issues. Most of these were due to a lack of experience, especially in terms of Android, since we both had almost zero Android experience.

1. **USE OF BLE ANDROID LIBRARY** The first challenge was to find an adequate Bluetooth Library for Android, which supports the Eddystone Beacon simulated by the Raspberry Pi, as well as the iBeacon provided by the chair. Once the library was selected, it took quite a long time to get it working. One of the issues here were the missing errors when required Android permissions were not set.

2. **BACKGROUND DETECTION ON ANDROID 8+** Prior to Android 8 it was possible to keep a service running in the background in order to detect beacons when the application is in the background. Starting with Android 8, this is not possible any more [3] since services are limited to run only once every 15 minutes. As a solution, a new API is provided which wakes up a given application once a defined beacon is in range. Unfortunately, this did not work for us, thus, we were forced to keep our application permanently active by creating a persistent notification.

6 WORK ITEMS

Building our prototype involved various work items which we completed as follows:

1. **CREATING BLE BEACON** Creating a NodeJS application which serves as a BLE beacon both on the laptop and the Raspberry Pi.

2. **DEPLOYMENT TO RASPBERRY PI** In order to have an easy way to deploy new versions of our application, we decided to use Balena. Balena allows to run Docker images on a Raspberry Pi and manages the deployment.
3. **FINDING BLE LIBRARY FOR ANDROID** Finding a BLE Library for Android which supports both the Eddystone beacon (Raspberry Pi) and the iBeacon by the university.
4. **FOREGROUND DETECTION** Enabling the Android application to find the beacon and then estimating the distance while being in the foreground.
5. **BACKGROUND DECTECTION** Enabling the Android Application to find the beacon while being in the background. In order to do this, we needed to create an additional Application class which can permanently run.
6. **RASPBERYPI API** Creating an API on the Raspberry Pi which interacts with the GPIOs.
7. **MAKING API REQUESTS WITH ANDROID** Adding the API Request on the Android phone once the smartphone is near enough.
8. **PUBLISHING RASPBERRY PI API** In order to be independent of the network setup, the Raspberry Pi publishes the API using a URL with localtunnel.me. Thus, independently of a local network connection, the API can be reached over the Internet.
9. **TESTING** We used the feature branch system of Git: implementing and pushing, the other tests and merges into main branch.

7 WORK SPLIT

Using Git issue feature, we assigned tasks to us and shared the progress with each other. Especially, the merge request feature helped a lot in keeping a stable version. Overall, Philipp worked on the Beacon detection and Mehdi focused on the rest of the Android App. Testing was done by both.

8 CONCLUDING REMARKS

During the last months, we were able to develop a working prototype including our desired feature set. The prototype proves that such a setup is possible, however it outlines some limitations such as battery consumption. This could be for example further improved by only scanning for beacons when the person is in a certain geofence.
All in all, this has helped us to extend our knowledge gained from the lecture. We are looking forward to further work on our prototype and to get it working in reality.

REFERENCES

[1] https://www.balena.io/.

[2] Roman Shtylman. https://localtunnel.github.io/www/.

[3] David G. Young. Beacon detection with android 8. http://www.davidgyoungtech.com/2017/08/07/beacon-detection-with-android-8, Aug 2017.