BLE Jamming; Location Music Player

Brandon Spada
CS391: Wireless Networks
University of Hartford
spada@hartford.edu

Amanda Boeni
CS391: Wireless Networks
University of Hartford
Brick, New Jersey
Boeni@hartford,edu

Cong Nguyen
CS391: Wireless Networks
University of Hartford
New York, NY
cnguyen@hartford.edu

Andrew Kuczma
CS391: Wireless Networks
University of Hartford
Granby, CT
kuczma@hartford.edu

***Abstract*— This document talks about using Bluetooth Low Energy Beacons for a location music player and the research of sent and received packets of data between Android device and Bluetooth Low Energy beacons.**

Keywords*— *Bluetooth Low Energy, RSSI, Spotify Web API, iBeacon, Bluetooth GATT

I. Introduction

Bluetooth Low Energy, or BLE, is much like normal Bluetooth technology, but provides a great reduced power consumption while also being a cheaper alternative that offers a similar communication range. This project is based around the idea of playing music based on the location of a handheld device. Imagine a silent disco, but instead of manually switching the channel on your headset, the music channel will change depending on your location within a room or building. This project was implemented using Spotify, Android app studio, and Kotlin; the eventual goal of this project is to get position data by measuring packet loss from device, to BLE beacon, back to the device, but this article will focus on an RSSI implementation of this particular idea. The Beacon that we used for this project were BlueCharm BC037 iBeacon devices, which were very useful in retrieving basic information, such as the RSSI values.

Fig 1: iBeacon device

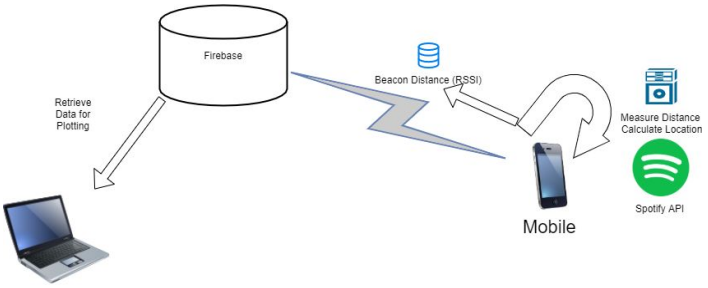
II. Ease of Use

A. User Friendly App

Using Android App Studio, we were able to make an app that connects to the BLE beacons and implements Spotify to the Android device. The application development process was the most important stage of the project.

B. Planned Research and Change

For this documentation, the original planned project was to get a specific playlist on Spotify to play when closer to any one beacon using time of flight (ToF). If you were closer to beacon one, you would get the associated song



playing out of your Android device. However, due to strict time constraints, at the end of the semester, it was suggested to track packet sending and receiving from device to beacon with physical and digital obstacles, but that too was not practical within our current two-month window. This project utilizes RSSI (Received Signal Strength Indication) to measure the distance between BLE beacons and the mobile device as a result.

Fig 2: Designing the system

C. Abbreviations

- *BB - Blackboard*
- *IEEE - Institute of Electrical and Electronics Engineers*
- *BLE - Bluetooth Low Energy*
- *ToF - Time of Flight*
- *B1 - Beacon One*
- *B2 - Beacon Two*
- *B3 - Beacon Three*

- *RSSI- Received Signal Strength Indicator*

III. PHASES

In this segment, the different project phases are discussed along with the work done during these phases.

A. Proposal / Design

For the design of the project, this diagram represented the plan to get an Android App to connect to a constantly updating database to store BLE Beacon information on:

1. Connection Signal Strength
2. Distance
3. Overall Delay (per millisecond)

Two main ideas were discussed on how to calculate the location with the distance, signal, and delay: Trilateration and Time of Flight. Trilateration is the process of determining absolute or relative locations of points by measurement of distances, using the geometry of circles, spheres, or triangles. This would have been a good way to accurately pin-point where the Android device is in regards to the three different BLE beacons trying to get their own song to play. However, the main problem with this was the delay between device and the beacons, this would make trilateration too inaccurate. Time of Flight, or ToF, is a method for measuring the distance between a sensor and an object, based on the time difference between the emission of a signal and its return to the sensor. So using the packets sent between the Android device and the BLE beacon, we can therefore measure the distance with more accuracy, accounting for the delay and the signal strength; so when the songs on the Android device update, we can calculate and show the user the delay between songs playing. Once the user moves, if it shows a 10 second delay, then the user knows not to move until the song changes to make sure the BLE beacons can represent the location the user is in by the song played.

B. Implementation

a) Android App

The application is a basic blue background with a Spotify logo in the lower right hand corner with a button that allows the user to be directed within the web browser app that allows for spotify to give permission to the app. Once clicked and successfully connected to the app, the button will change into a “Connected” status that is non-interactable. Along with the Beacon names: 1, 2, and 3; the app shows the connection signal strength.

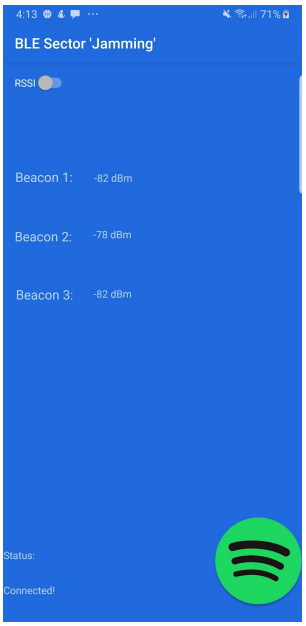


Fig. 3: The application’s appearance.

b) Spotify Connection

Using Spotify Web API [1], we connected their services into the Android app. Tapping the “Connect” button on the app, the app will be redirected to the web service app on the Android device (Chrome, Explorer, Safari, etc.) where Spotify will ask the user to log into their profile in order to access the music that will be played through your device; this includes allowing Spotify to share your service to the app.. Then once the app is reopened, the button will change to the “Connected” status and the API will change the playlist whenever one person moves out of the specified sector.

c) Bluetooth Connection

We connected to the iBeacons using the Public API for the Bluetooth GATT Profile server. This class provides Bluetooth GATT server role functionality, allowing applications to create Bluetooth Smart services and characteristics.

The beacons themselves are cost-effective and small, roughly 1.5 inches in diameter and less than an ounce. The replaceable battery within is rated to last more than a year of constant usage, not to mention easily accessible, as any corner store or market may sell batteries such as these. Using the Eddystone app on android was an easy way

In addition, we also implemented a Bluetooth Adapter in the Kotlin class. It represents the local device Bluetooth adapter. The BluetoothAdapter lets you perform fundamental Bluetooth tasks, such as initiate device discovery, query a list of bonded (paired) devices, instantiate a BluetoothDevice using a known MAC address, and create a BluetoothServerSocket to listen for connection requests from other devices, and start a scan for Bluetooth LE devices in the vicinity. Since we need to scan for each MAC-address-specific device, we need to check the scanner for those specific addresses every time the scanner is called.

```
//Setting rssi ..... first implementation...
if(result?.device?.address == "80:6F:B0:6C:94:2B"){
    rssi1 = result.getRssi()
```

Fig 4: example of looking for device-specific RSSI

d) Location Tracking

One of the methods we used was track the location of the device was trilateration. Using the received RSSI values from the beacon. We were able to calculate the distance between the device and the beacon individually, using the following equation:

$$distance = 10^{(RSSI-A)/-10n}$$

In code:

```
Math.pow(10.0, ( txPower - rssi) / (10.0 * signalPropagationConstant))
```

The value 'A' and 'signalProgationConstant' was set to 2.7 which represents an environment with few obstacles. We tested the algorithm in an environment that is a little less ideal because there were some minor obstacles in-between the beacons and the device, making RSSI very jittery, though this is to be expected even in ideal environments due to the nature or RSSI.

The algorithm to calculate the X and Y coordinates is trilateration, which allows for us to calculate a somewhat accurate point for the beacons.

$$(x-x_1)^2+(y-y_1)^2=r_1^2$$
$$(x-x_2)^2+(y-y_2)^2=r_2^2$$
$$(x-x_3)^2+(y-y_3)^2=r_3^2$$

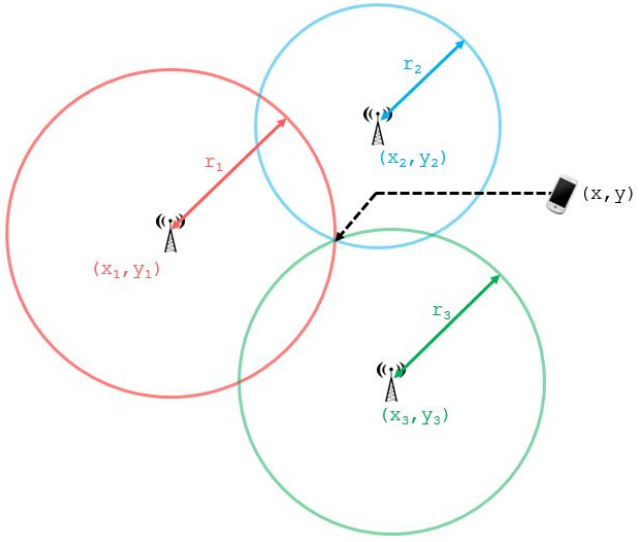


Fig. 5: An example of trilateration.

The r values are the distance between the beacon and the device.

In addition, the RSSI displayed on the app works but the displayed values have considerably high variation in a short period of time that make the location tracking to be less accurate. So we used the average value of the RSSI in the period of 10 seconds.

```
algorithm QuadrantsSort is
input: da, db, dc
output: enum quadrant q such that q = "Top Right",
"Top Left", "Bottom Right", "Bottom Left"

if: da < db and da < dc and (da + dc)/2 < (dc + db)/2
then: set q "Bottom Right"
else if: dc < db and da < dc and (da + dc)/2 < (dc + db)/2
then: set q "Bottom Left"
else if: db < da < dc and (da + db)/2 < (db^2 + dc^2)^-2
then: set enum "Top Right"
else if: db <= dc < da and db < (da^2 + dc^2)^-2
then: set q "Top Left"
(note: <= is used in order to help solve cases that may be
in the middle of the quadrants; in practice this should never occur.)
return q
```

Fig. 4: The algorithm used to determine the quadrant that the user is in, in pseudocode.

Finally, the algorithm takes the distance data from the previous function, and determines the quadrant the user might be in. While crude, it does the job quite well, despite the finicky nature of RSSI when it comes to distance detection. In fact, due to it being so general, it allows quite a bit of flexibility with quadrant detection, even with multiple obstacles (such as other humans blocking certain paths).

e) Spotify Playlist

We divided the room into four sections. Top left, top right, bottom left and bottom right. Everytime the device enters a different section and the app checks the location, it would play a random song in the specific playlist assigned to that section.

C. *Final Result Report*

As expected, the app connected successfully to each of the bluetooth beacons with little trouble. The button on the display which connects the device to the Spotify API also works as expected, but the Spotify API might take a few seconds to load up. Everytime the device is connected, the apps runs Spotify and play the specific playlist.
Furthermore, the app did change the playlist after moving into different sections, however, the change did not follow the theoretical result that we were expecting. This was due to the high variation of RSSI value, as the calculated location of the device varies greatly from higher RSSI. Since distances can change drastically, this might accidentally change the sector that the device is in.

```
distA: 1.4307229891937576
distB: 1.9201419386388017
distC: 3.1220829994678336

graph2X: 4.897966610288024
graph2Y: 3.5796277958574283
```

Fig 6: Distance calculations and conversion to (X,Y) components

The stated goal of the project is true; the playlist does change per sector of the room the user is in. However, in terms of precise location tracking, the app does not do the best job, as this distance can vary drastically.

D. Future Work

For future work, the accuracy of the distance measurements are the most important thing to fix. With proper distance detecting, a closer reading from Android device to each beacon would increase the location tracking algorithm accuracy by a high degree. RSSI is also to blame for accuracy woes, for it simply is not made to be used in this fashion and is not a stable value for such use. In order to remedy this, a viable alternative would need to be found. The reliability of BLE in crowded space is also a concern; getting this to be more accurate and efficient is something that is to be worked on. Placing the beacons higher up (ie. taped on walls) worked the best, but more testing with different altitudes could also change the location tracking and overall accuracy. Placing the beacons on the ceiling, and thus decreasing potential obstacles, is a promising way to approach this, so finding appropriate brackets for hanging or fastening might be another step to take. With more time to mature and some further research, this project has a very good chance of becoming a powerful and effective method for real-time position tracking with BLE.

IV. Research

There were a few learning curves for the group. This section focuses on the learning challenges that lead to speed bumps in the project timeframe.

A. Kotlin

Kotlin is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin is designed to interoperate fully with Java, and the JVM version of its standard library depends on the Java Class Library, but type inference allows its syntax to be more concise. With the language being really connected and similar to Java, we were able to learn and implement this project fairly quickly.

B. Android Studio

Not having background experience with Android Application development proved to be a challenge in the beginning. However, we used Android Studio which has a great user interface and there were a lot of useful resources online to help us understand the process of developing an application quicker.

The Android Studio libraries were a tremendous help to understand the classes needed to implement the apps and the required knowledge of the Android libraries.

C. BLE Signal and Packets

The BLE Signal was strong in our test cases since the experimented distance was relatively small, but the larger spaces in which it was further tested created a high level of variation in the output. It is typically more useful for the purposes it is designed to do, but is great for a quick and simple option for position detection such as this where precision is not necessarily required and funding is lacking.

D. *FireBase*

Firebase is a mobile and web application development platform owned by Google. Our group used it to store our database. The variables sent are RSSI values, calculated distances and x-y based location of the device. The database was constantly updated while the app was running. We used Firebase due to ease of use, accessibility and integrated on Android Studio.

Acknowledgment

Dr Andrew Jung provided helps with the direction of the implementation. In addition, he offered innovative ideas for the group to research. In the end, due to time constraints, we needed to use RSSI instead of packet handling. Additionally, Mike Jakovic provided some help with troubleshooting and creating the quadrant location algorithm.

References

[1] Spotify Web API. “Endpoints return JSON metadata about music, artists, albums, and tracks from Spotify Data Catalogue. Web API also provides access to user related data, like playlists and music that the user saves.” https://developer.spotify.com/documentation/web-api/

[2] Android Studio. Official IDE for Android operating system and app development, Kotlin preferred language, but also supported by languages such as Java and C++. https://developer.android.com/studio

[3] A built-in platform introduced in Android 4.3 that supports BLE and provides APIs that apps can use to discover devices, query for services, and transmit information. Uses Generic Attribute Profile (GATT) for sending and receiving short pieces of data over a BLE link.. https://developer.android.com/guide/topics/connectivity/bluetooth-le

[4] Your Bibliography: 101 Computing. (2019). Cell Phone Trilateration Algorithm | 101 Computing. [online] Available at: https://www.101computing.net/cell-phone-trilateration-algorithm/ [Accessed 5 Dec. 2019].