Shyam Tailor

Anonymous Proximity Beacons from Smartphones

Computer Science Tripos – Part II

Downing College

May 13, 2018

Proforma

Name:	**Shyam Tailor**
College:	**Downing College**
Project Title:	**Anonymous Proximity Beacons from Smartphones**
Examination:	**Computer Science Tripos – Part II, July 2018**
Word Count:	**11845**[1]
Project Originator:	**Dr Robert Harle**
Supervisor:	**Dr Robert Harle**

Original Aims of the Project

This project investigates the usage of Bluetooth-enabled smartphones for human proximity detection. Smartphones can transmit identifiers, which can be heard by other smartphones nearby. A naive solution where each smartphone continually transmits a fixed identifier would make it trivial for a third party to track users; this project explores a solution to make tracking difficult by using mutable pseudo-anonymous identifiers that can only be reversed by a de-anonymisation server. An example application is developed: the server can return location information to requesting users, emulating the functionality of static proximity beacon systems — except without the deployment costs.

Work Completed

I met all my success criteria. I built an Android smartphone client, and a de-anonymisation server which users could obtain location information from. Three different anonymisation schemes were built for Bluetooth Low Energy, which is currently widely adopted. Each scheme was benchmarked, and the system was subjected to real world testing: a deployment was done, and power consumption tests were completed. Extension work was completed to improve the quality of location information returned to users, and to explore a novel anonymisation scheme made possible by new features in Bluetooth 5, which was recently standardised.

[1]This word count was computed using `texcount -sum -inc -utf8 -sub=chapter diss.tex` for chapters 1–5.

Special Difficulties

Adoption of Bluetooth 5 (standardised December 2016) has been slower than expected. Few flagship smartphones have been released with support, and it is not necessary to support all new features to claim compatibility. A similar situation has occurred with development kits; full support can be expected around December 2018. The Bluetooth functionality used in this dissertation is extended by Bluetooth 5; from a programmer's perspective, very little modification is required to use it. The extended functionality may increase power consumption however, but I am unable to quantify this at time of writing.

Declaration

I, Shyam Tailor of Downing College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed [signature]

Date [date]

Contents

Chapter 1

Introduction

A ubiquitous system for reliable and precise localisation indoors remains elusive; proposed systems have significant barriers to adoption. For many applications, however, precise indoor location is unnecessary: sometimes we just need *context*. For example, you might want to know if you are 'at' or 'near' something, and not where you are precisely. *Beacon*-based systems can provide this information: proximity to a beacon gives you context (known as *microlocation*).

Beacons have been built with Bluetooth Low Energy (BLE) [29, 27], which was standardised in 2010. To date, however, deployments of beacons are rare. My project evaluates an innovative solution to this issue: instead, use stationary personal devices such as smartphones as dynamic beacons, *without compromising the privacy of the device owner*. The material developed can also be applied to static beacon deployments to combat *spoofing*.

I achieved the core aims of my proposal, which required the implementation of a client application, and a de-anonymisation server. I completed two extensions that improve location estimation quality as well as developing a novel approach using Bluetooth 5 (standardised in 2016).

In this chapter I provide the motivation for this project, and setup the problem I am solving; finally, I cover existing work.

1.1 Motivation

Historically, indoor positioning systems focused on providing high precision location—accurate to within 2-3 metres minimum. WiFi or Bluetooth fingerprinting [49, 34] involves an initial scanning phase to capture a signal strength map for a building; localisation requires finding where in the map the user is. The Bat system [38] used ultrasound receivers installed in the building. Users wear a device that emits an ultrasonic pulse picked up by receivers with a known location. The receivers infer distance from the propagation time; the user's location is found by solving for the intersection of these distances from

the receivers. These systems have barriers to adoption: the fingerprint scanning phase must be redone if furniture moves, while the Bat required a dense installation of receivers.

Beaconing systems have been developed using RFID [26], radio beacons, and IR [37] and visible light signals [30]. BLE beacons are a special type of radio beacon, but are popular due to adoption of BLE in smartphones. BLE beacons are not useful for obtaining precise location because of the problems of *multipath propagation*: distance measurements are too inaccurate for multilateration techniques to work.

Beacons are subject to the same adoption issues. Fine grain microlocation can only be obtained with a dense deployment of beacons, which is expensive and requires maintenance. Another issue is *spoofing*: beacons transmit fixed identifiers which serve as a key into a database. These can be captured and replayed to fool devices into believing they are in a location where they are not. Using fixed identifiers also means other parties can create an alternative database of beacons. This represents a commercial challenge: there is little incentive to invest in a deployment if it can be used to power a rival service.

Human proximity detection is possible with BLE [43], but it is essential that individuals cannot lie about their identity; otherwise, data collected may be incorrect. Proximity information is useful in a wide variety of fields: examples include modelling the spread of epidemics, or understanding workplace interactions. The quality of interaction in the workplace is an important factor in productivity [31].

Another application is *context-aware sensing* [44], where energy consumption is reduced by using the cheapest possible sensor to determine the context of the user. The relationships between the user's context and sensor information is learned dynamically, and it may be possible to incorporate social information into the learning algorithm.

1.2 Problem Outline

The system assumes a set of stationary devices that broadcast an identifier using Bluetooth. The devices may be standard, dedicated beacons or personal devices. In traditional deployments, beacons advertise a fixed identifier. A personal device doing this could be tracked by third parties, however.

Personal devices should transmit pseudo-Anonymous IDentifiers (AIDs) which can be detected by other devices. AIDs are reported to a server for de-anonymisation; personal devices are *not* able to reverse AIDs themselves. The server can therefore deduce which users are in close proximity. The proximity information is used for location in this dissertation: devices can report their location to the server, and the server can return location information to devices which are co-located. This is useful if a device is unable to obtain *any* location fix.

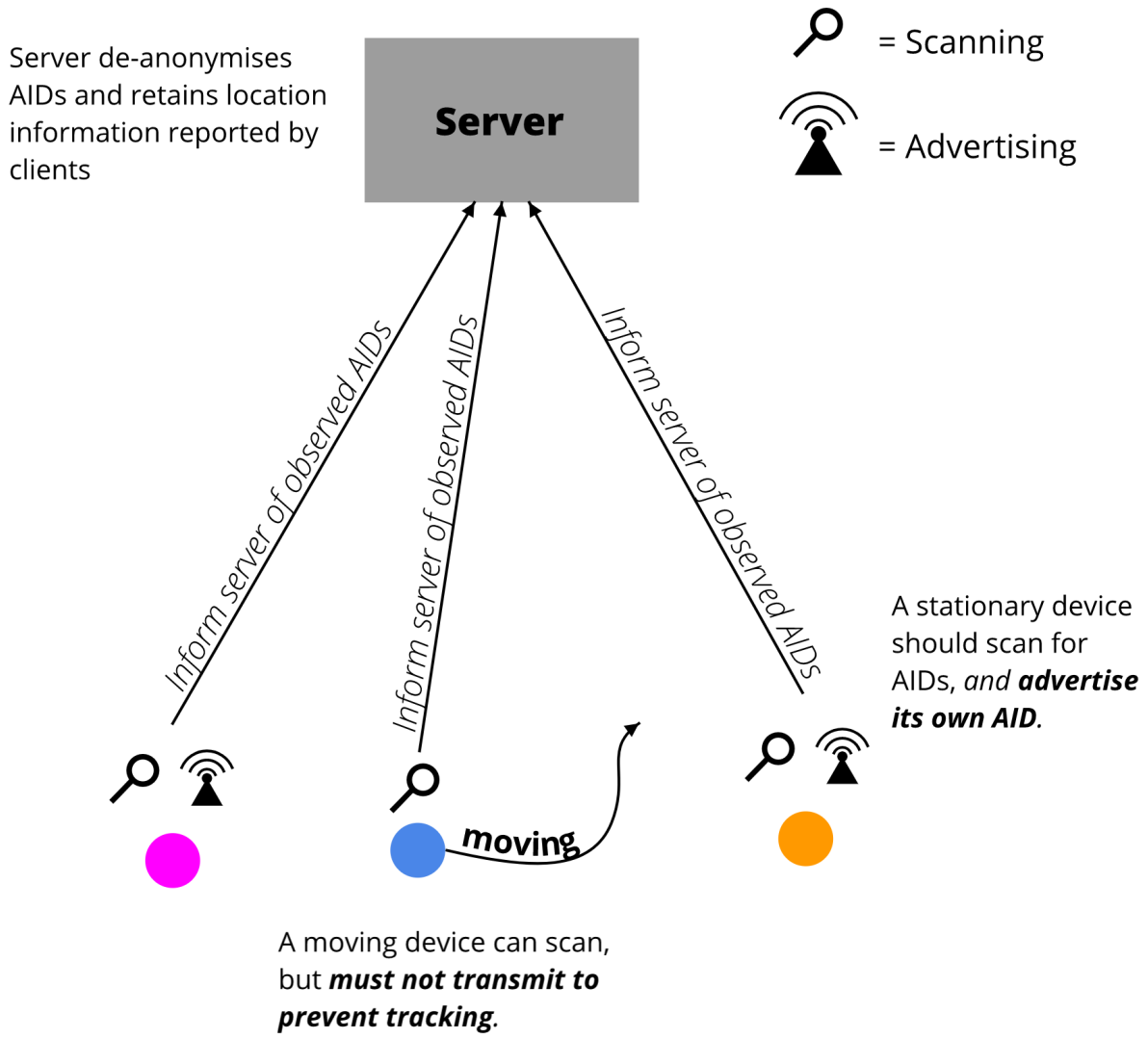


Figure 1.1: Interactions of devices with each other and the server.

To avoid tracking, devices *cannot* transmit while they are moving; they may still scan, however. This is indicated in Figure 1.1.

1.3 Related Work

1.3.1 Anonymous Beaconing

Anonymous BLE beacons have become commercially available in recent years. Estimote's [11] system involves assigning each beacon a unique key during manufacture, which is used to create a *visible identifier* that is forwarded to a server to obtain the real ID; no more details are available. Google have released the Eddystone EID product, which operates in the same way. Technical details are available [39], and I explore this

scheme as one method of achieving the requirements.

Anonymous beaconing has been studied in the RFID domain; relevant to this project is work done on creating identifiers for banknotes [40].

1.3.2 Human Proximity Detection

BLE is an effective technique for human proximity detection [43]. The accuracy of proximity detection was 97% for a 10 second window, and it is possible to deploy such a system using consumer hardware.

1.3.3 Localisation in Ad-Hoc Networks

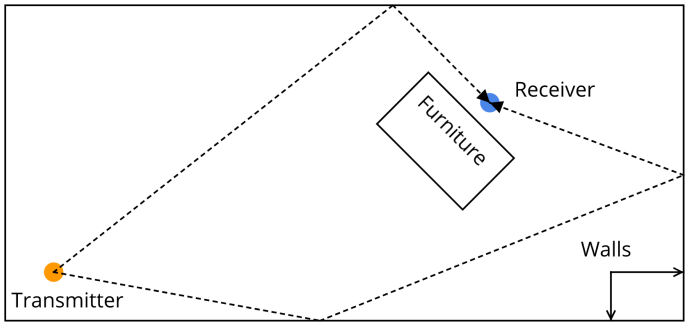


Figure 1.2: Multipath propagation.

Localisation of nodes from connectivity graphs has been studied for ad-hoc networks [33, 48]; unfortunately, the techniques are not applicable to this project. BLE does not provide accurate distance measurements due to multipath effects, and personal devices cannot obtain *very* accurate location fixes indoors. It may be possible to workaround inaccurate location fixes using angulation techniques, where the device knows the direction of incoming packets, but Bluetooth does not support this feature. The techniques developed could be reused with a different beacon platform supporting these measurements in the future, however.

Chapter 2

Preparation

In this chapter, I discuss Bluetooth and the capabilities introduced by the newest specification. I move on to the theory behind using a Kalman filter to smooth signal strength measurements. Finally, I state the starting point for this project, the software I have used, and present the requirements.

2.1 Bluetooth 5

Bluetooth (*Classic*) was introduced in the 1990s for exchanging data over short distances between two devices. It is designed for *continuous* data transfer with high throughput: use cases include file transfer and audio streaming. The protocol is power-hungry, and previous attempts to use it for proximity detection were restricted to scanning once every few *minutes* [31].

The introduction of Bluetooth Low Energy (*BLE*) in Bluetooth 4.0 [2] removed continuous streaming to reduce power consumption; it was designed for short interactions with IoT devices and wearables. BLE has *advertisements*: broadcast packets with a maximum length of 31 bytes. Some bytes are lost to protocol overheads, so the number of bytes that can be specified in the advert is actually fewer; we will return to this in the implementation chapter. Connection-orientated functionality for BLE exists, but is unnecessary for this project. BLE advertisements are *stateless*, unlike connections, which reduces power consumption for listeners; additionally, anyone can listen to an advertisement. BLE devices are either masters (which scan for other BLE devices) or peripherals: advertisement transmission requires the device to support peripheral mode. A device can be a master and a peripheral concurrently. BLE also supports randomising the advertisement MAC address for privacy reasons.

Bluetooth 5 was standardised in 2016 [3], and introduces improvements to BLE. The main improvements are increasing advertisement packet length and the effective range.

Advertisement packet length has been increased to 255 bytes. Another feature is *chaining*, where several packets can be combined to form a single logical advertisement

of up to 1650 bytes.

Bluetooth 5 offers techniques to improve range to $\geq$ 40m, compared to BLE's 10m [6]. This is achieved through *forward error correction*, physical layer *coding* techniques, and increased maximum allowed transmission power. Increasing range increases power consumption, however.

2.2 Signal Strength Filtering

Signal strength measurements are commonly used with BLE to class distances [29]: if the signal is stronger than a threshold, then the two devices are almost certainly close. The measurements are noisy and need smoothing, however.

The log-distance path loss model [47] describes radio propagation inside buildings under ideal circumstances; it says that signal strength decays exponentially with distance, and that the signal strength measured by a receiver will follow a Gaussian error model.

We can smooth signal strength measurements using a Kalman Filter [48], which estimates an unobserved variable using noisy measurements and is suited to a Gaussian error model. We define a process model, describing how the system's state evolves over time, and a measurement model, relating the state of the system to the measurements made. For path loss filtering, the process model is defined as:

$$\begin{aligned} x_t &= Ax_{t-1} + Bu_{t-1} + \varepsilon_{t-1} \\ &= x_{t-1} + \varepsilon_{t-1} \end{aligned}$$

This says that the current state is the same as the previous state, plus some *process noise*; the process model allows for a control input u in general, but we assume a situation where neither the transmitter or receiver are moving and therefore drop the term.

We define a measurement model describing how state x_t results in measurement z_t as:

$$\begin{aligned} z_t &= Hx_t + \delta_t \\ &= x_t + \delta_t \end{aligned}$$

The measurement should equal the state with additional *measurement noise* since we can measure it directly. Both ε_t and δ_t are zero-mean Gaussian random variables, whose variances need to be chosen; in this case, the measurement noise is much larger than the process noise.

2.3 Starting Point

This project is inspired by a Part III project by Augustin Zidek [50], which was converted into a paper accepted to a conference in March [51]. I am a co-author on

this paper; my contributions were doing battery drain experiments and suggesting modifications to the anonymisation schemes described. My implementation is completely separate, and I had no access to Augustin's codebase. I implemented three different anonymisation schemes for BLE; Augustin's implementation used only one. My extension work considers Bluetooth 5, a technology not released when Augustin did his project.

The most closely related Tripos courses were *Mobile and Sensor Systems* and *Security I/II*. Small amounts of code were written prior to the start of the project to familiarise myself with libraries: namely for JSON serialisation of Java classes, movement detection and how to enable BLE peripheral mode. Code was copied from libraries' websites with little modification.

I had never written any mobile applications beyond "Hello World" examples. Although Android uses Java, there are several concepts that are specific to Android, and its standard library is significant.

2.4 Software Used

2.4.1 Smartphone Platform

I implemented my client application on Android. Android applications are usually written in Java, a language I am familiar with; iOS requires learning Swift and the application model. I also did not own an iOS device. It is difficult to sideload applications on iOS; to do a deployment, I would have to pay a fee to Apple.

2.4.2 Programming Languages

I used Java and Python for this project.

Java: Android development is done with Java. I used Java for the server implementation since I could share modules between the client and server, and it simplified serialisation. Java also has an extensive standard library with support for cryptography and networking.

Python: Python was used in the evaluation because there are many scientific libraries available, which I used to construct many of the diagrams in this dissertation. Python is also suitable for processing log files.

I made use of **Bash** scripts to automate benchmarks.

2.4.3 Development Environment

Git: I used git repositories for the client and server for version control. I used branches to implement large changes, allowing me to backtrack and context switch. Git's sub-module functionality was used to share components between the server and

client: changes made for the client would be visible for the server (and vice-versa). Code was regularly committed and pushed to GitHub which provided one form of backup.

Unit Testing: Unit tests were written using JUnit; they are a method of sanity checking and ensuring that changes do not cause breakages. Over 100 unit tests were written, and were run before committing any code.

IDEs: I used Android Studio [7] for client development as it is recommended and supported by Google. Server side development was done using IntelliJ [25]. These IDEs are similar as they are developed by the same company, reducing the time required to become familiar with both.

2.4.4 Benchmarking

During development, it is necessary to assess the benefit of optimisations made. Naively benchmarking Java code gives poor results, however. Benchmarks must account for optimisations applied dynamically by the Java Virtual Machine (JVM). Pitfalls include:

1. Dead code elimination.
2. Constant folding.
3. Loop unrolling and SIMD.
4. Garbage collection pauses.
5. Non-determinism: different JVMs can optimise the same code differently.

Benchmarks were written using the JMH [10] microbenchmarking library in the OpenJDK, which accounts for JVM behaviour.

2.5 Requirements Analysis

I used the *MoSCoW* method to expand on the requirements and extensions in the proposal.

2.5.1 Client

1. **Must** only transmit identifiers when the device is still.
2. **Must** implement transmission and scanning functionality for BLE.
3. **Must** target Android 5.0 — the version BLE peripheral mode was introduced and in wide use at time of writing.
4. **Must** have a method of determining whether a device has started moving, with latency under 5 seconds.

5. **Must** be able to communicate with the server confidentially.
6. **Must** implement some method of obtaining absolute location information.
7. **Must** have one method of creating anonymous identifiers that does not require contacting the server.
8. **Should** incorporate other methods of creating anonymous identifiers.
9. **Should** log important events for later analysis.
10. **Could** be power optimised.
11. **Won't** implement a Bluetooth 5 advertiser or scanner — there is insufficient hardware in wide use for a deployment.

2.5.2 Server

1. **Must** offer clients access control for their location.
2. **Must** be able to de-anonymise an identifier within 1 second of receiving it even with 1000 devices registered, *without heuristics*.
3. **Must** provide the following location information to requesting clients:
   (a) The number of devices in the near vicinity.
   (b) If one of the devices in the near vicinity has provided access control permissions to the requesting client, then disclosing that device's absolute location to the requesting client (if it exists).
4. **Must** implement all anonymisation methods implemented on the client.
5. **Should** provide the location of devices which have granted access to the requesting client, even if they are not visible to each other directly (i.e. multiple hops away).
6. **Could** inform neighbouring devices if they are particularly close to each other spatially.
7. **Could** implement Bluetooth 5 based anonymisation schemes.

2.6 Software Development Model

The components for the client and server have well defined requirements and interfaces; a *waterfall model* was therefore used to build a complete implementation with one anonymisation scheme. Further anonymisation schemes were easily added since I had used interfaces in my initial implementation.

Chapter 3

Implementation

I discuss the implementation of the client, before moving on to the server. Finally, I discuss anonymisation schemes; I look at existing schemes for BLE, and then at my proposed scheme for Bluetooth 5.

3.1 Client

The client is an Android application written in Java, with the application logic implemented in a service. A service is an application component *without a UI*, and which does background processing. There are background and foreground services: foreground services do work that the user is aware of; background services run operations which are not directly noticeable to the user. Android will kill background services regularly to conserve resources, so a foreground service is necessary. Foreground services must display a notification to the user when they are running [20]; this is acceptable for the purposes of the project. The service is invisible to the user apart from the notification. There is a GUI to monitor what the client is currently doing.

3.1.1 Bluetooth

Here I discuss the advertising and scanning components implemented for BLE. I am constrained by the Bluetooth specification [2]: the format *must* use advertising data (AD) elements defined in the specification for listeners to be able to interpret it. The format is described in Figure 3.3.

The packet uses two AD elements: the first specifies the transmission power; the second specifies data associated with a 2 byte manufacturer identifier. Each AD element contains length and type fields, followed by data. For transmission power, there is 1 byte: transmission power in dBm. For manufacturer data, the first 2 bytes are the manufacturer identifier; further data can be freely specified [2]. A 2 byte manufacturer identifier has been assumed (`0xDBEA`) which has not been assigned by the standards committee [14]; if the project is released publicly, a manufacturer ID would need to be obtained. Not assuming this value would require a 16 byte ID to be specified instead [2],

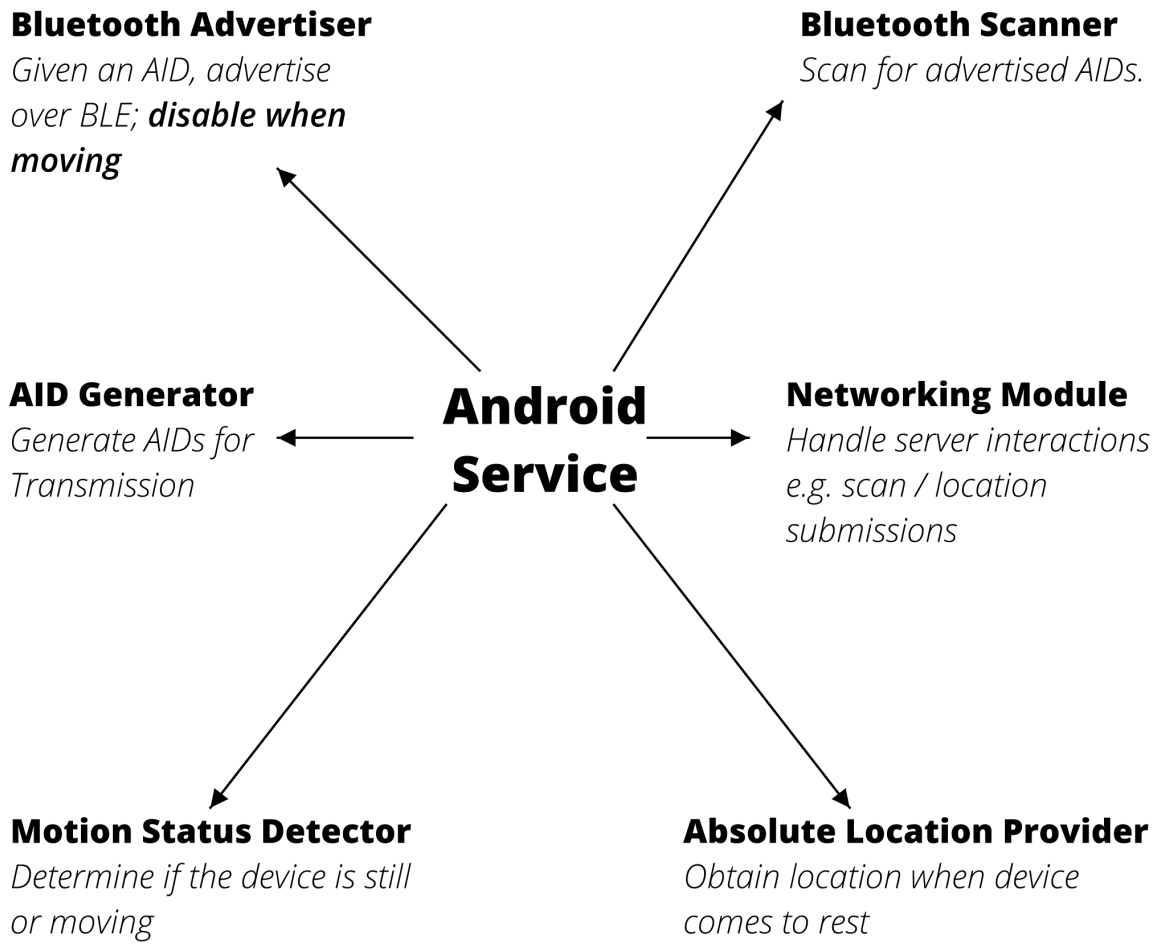


Figure 3.1: Main components in the client.

restricting the length of the AID. Code for constructing the advertisement is given:

```
1 AdvertiseData constructAdvertisementData(byte[] data) {
2         final byte[] advertised = concat(PREAMBLE_BYTES, data);
3
4         return new AdvertiseData.Builder()
5                 .addManufacturerData(MANUFACTURER_CODE, advertised)
6                 .setIncludeTxPowerLevel(true)
7                 .setIncludeDeviceName(false)
8                 .build();
9 }
```

There are 16 bytes for the AID; we discuss methods for generation later in this chapter. 6 bytes remain for future improvements.

Advertising is a cheap operation, so we use the maximum power advertisements the device supports; privacy implications are discussed in the evaluation.

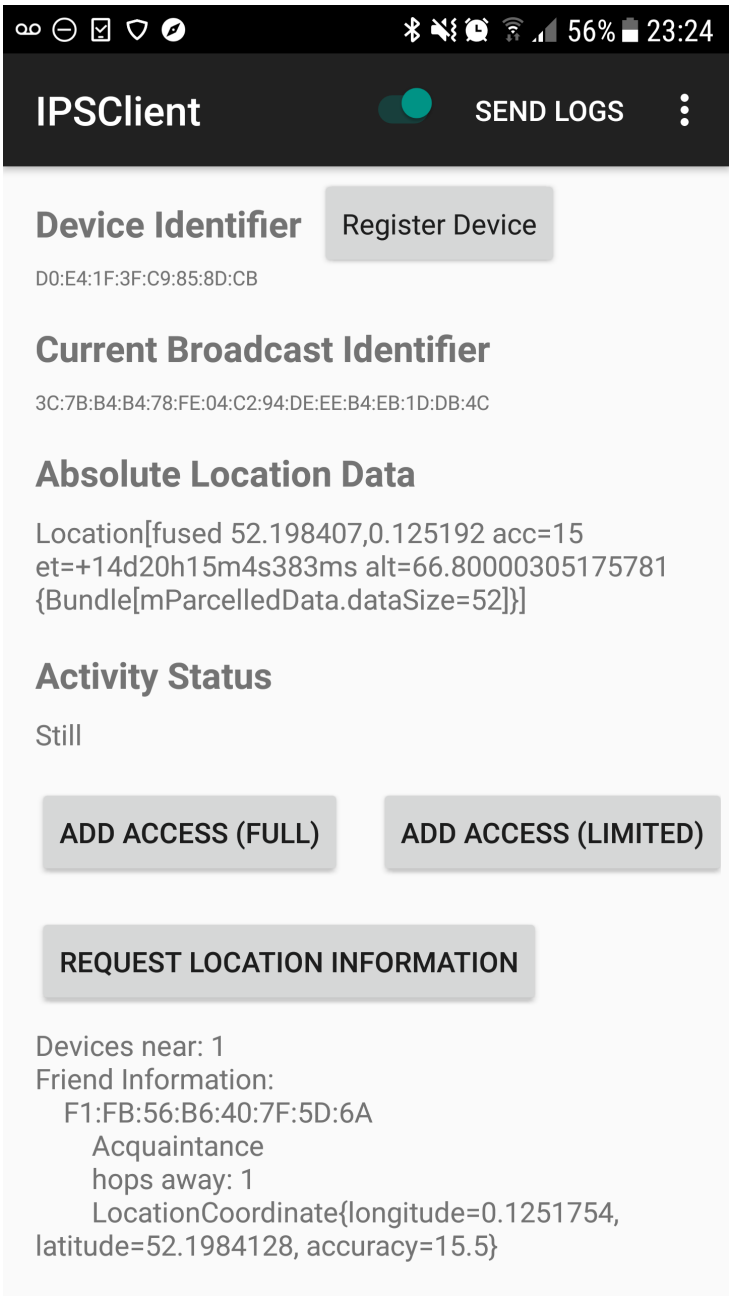


Figure 3.2: Test Client GUI.

Continually scanning is expensive. The client does not enable the scanner continuously when the device is still to save energy: it is duty cycled with a ratio of 25% and a period of 120 seconds. When movement begins, continuous scanning is enabled: the scan could reveal (previously unknown) relative proximity information between devices. As discussed in the evaluation, the proportion of time spent moving is small, so this does not have a significant impact on battery life.

Counter-intuitively, advertising is cheaper than scanning despite higher instantaneous power use: scanning must be enabled continuously whereas advertising has a duty cycle

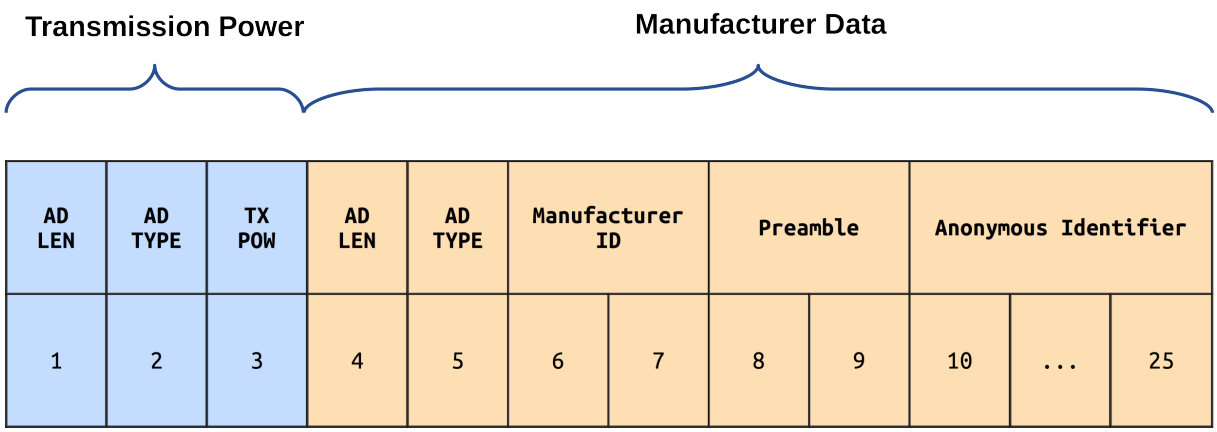


Figure 3.3: BLE payload format.

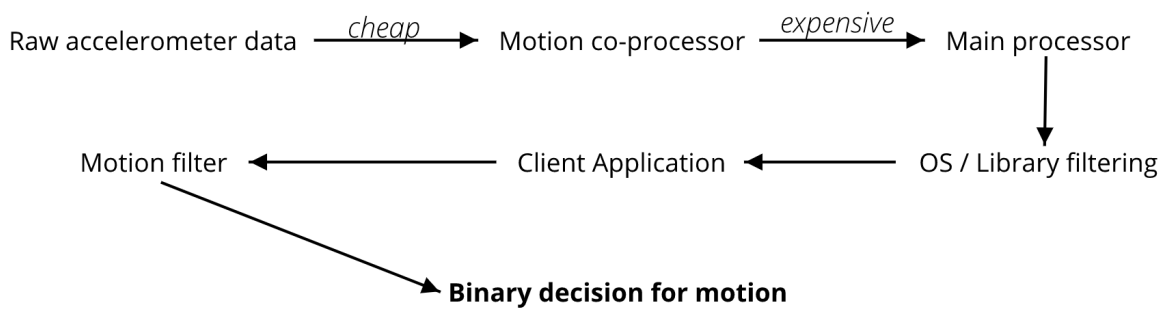
of around 1%. Scanning also requires further CPU involvement, unlike advertising.

The scanner listens for BLE packets; each packet is checked for data associated with the manufacturer identifier used, a valid preamble, and transmission power information. A scan record is then constructed with the AID, transmission power, received signal strength indicator (RSSI) and the device's current motion status.

The client does not de-duplicate scan records: it is useful for the server to have all the measurements. RSSI is a noisy measurement, so it is estimated from multiple measurements whenever possible. The server also needs to know when the last observation was made to decide if two devices are co-located at a later time.

Android disables the scanner after about 15 minutes; this is undocumented behaviour, and would result in AIDs not being observed when they should have been. Scanning was restarted if it had been activated for 10 minutes continuously using an event-loop primitive in the standard library.

3.1.2 Motion Sensing



Accelerometer data is used directly to determine whether the device's motion status.

Both the Android significant motion sensor [18] and the activity recognition API in Google Play Services [12] had too high latency: it took *at least* 10 seconds for them to report that motion had started.

Polling the accelerometer is cheap: modern devices have co-processors that collect data from the accelerometer continuously. The expensive operation is transferring the data to the main processor [19, 13]; for this application, some latency is acceptable. Power consumption can be reduced by instructing Android to deliver the accelerometer data when it becomes available (up to a maximum delay of 1 second).

The client retains a wakelock, which prevents the CPU from fully sleeping; I will expand on why this is necessary in Section 3.1.7. It is good practice to duty cycle the accelerometer reading to retain battery by shutting the CPU down; since the CPU is kept awake by the wakelock, however, there is no benefit to doing this: data is read continuously to ensure accurate decisions are made.

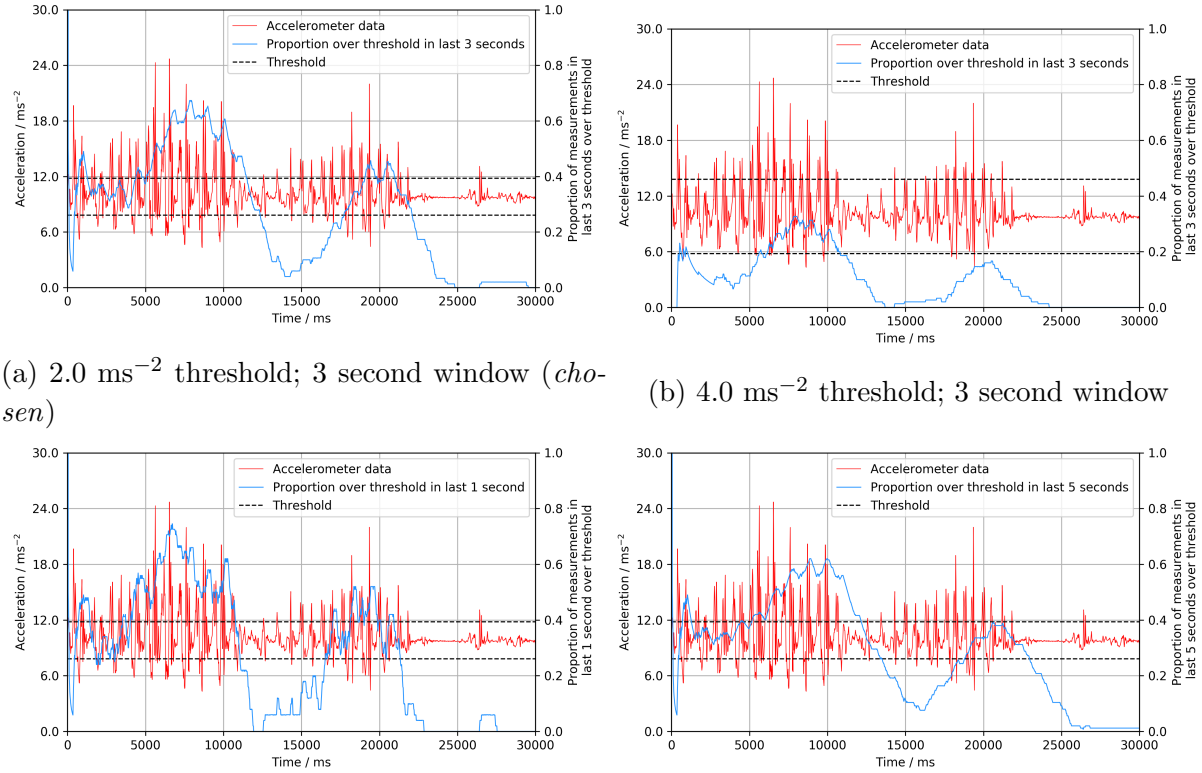
The accelerometer data is fed into a binary classification algorithm to decide if the device is moving. This decision is passed to the service, which disables the Bluetooth advertisement if the device is moving. When motion has stopped, advertising is restarted with a new AID.

Classification

Advanced techniques are unnecessary to decide if a device is moving or not. A widely used procedure is to threshold the measurements: if the measured acceleration magnitude deviates from gravity by more than a certain amount, then the device is moving. A simple procedure uses less CPU time, reducing energy usage. The threshold used by my application is 2.0 ms^{-2}, chosen through a sensitivity analysis on walking data collected from myself (Figure 3.4); an alternative analysis with a threshold of 4.0 ms^{-2} is given, indicating that it is too high.

Transient changes need to be ignored; an example is Figure 3.5. A window of measurements are retained and motion is reported if more than a certain proportion were over the threshold. The window period was chosen as 3 seconds, and the proportion was 20%; these parameters were chosen using the sensitivity analysis (Figure 3.4). Both parameters could be updated by the server.

Parameter choice is a compromise between latency and false positive reduction. We want to ignore very short periods of motion, where the user's location does not change significantly: a new location measurement is taken when the device becomes still, using energy and providing no new information since the location measurement is too imprecise to distinguish between the two positions. A 3 second latency is the worst case, meeting the 5 second requirement.



(a) 2.0 ms^{-2} threshold; 3 second window (*chosen*)

(b) 4.0 ms^{-2} threshold; 3 second window

(c) 2.0 ms^{-2} threshold; 1 second window

(d) 2.0 ms^{-2} window; 5 second window

Figure 3.4: Sensitivity analysis motivating algorithm design. Accelerometer data was taken from my smartphone (Samsung Galaxy S7) while I was walking, with a short pause after 11 seconds.

The algorithm is *sticky*: it takes 15 seconds of no activation events being detected before reporting the user as still. A further delay (chosen uniformly using a secure random number generator) of up to 10 seconds is used, on top of the minimum delay, to reduce the risk of an adversary correlating transmissions from the same device. Pause length trades proximity data granularity against individual user's privacy: too long pauses means short interactions are missed. The numbers chosen reflect this observation.

3.1.3 Location

Absolute location data is obtained through the fused location provider in Google Play Services, which combines data from sources such as GPS, accelerometer and WiFi [15]. The location data is usually more accurate than what can be obtained from GPS alone, especially indoors.

When a device starts moving, the location measurement is invalidated, and is updated once motion has ceased. The location is logged, and sent to the server if the device has WAN access.

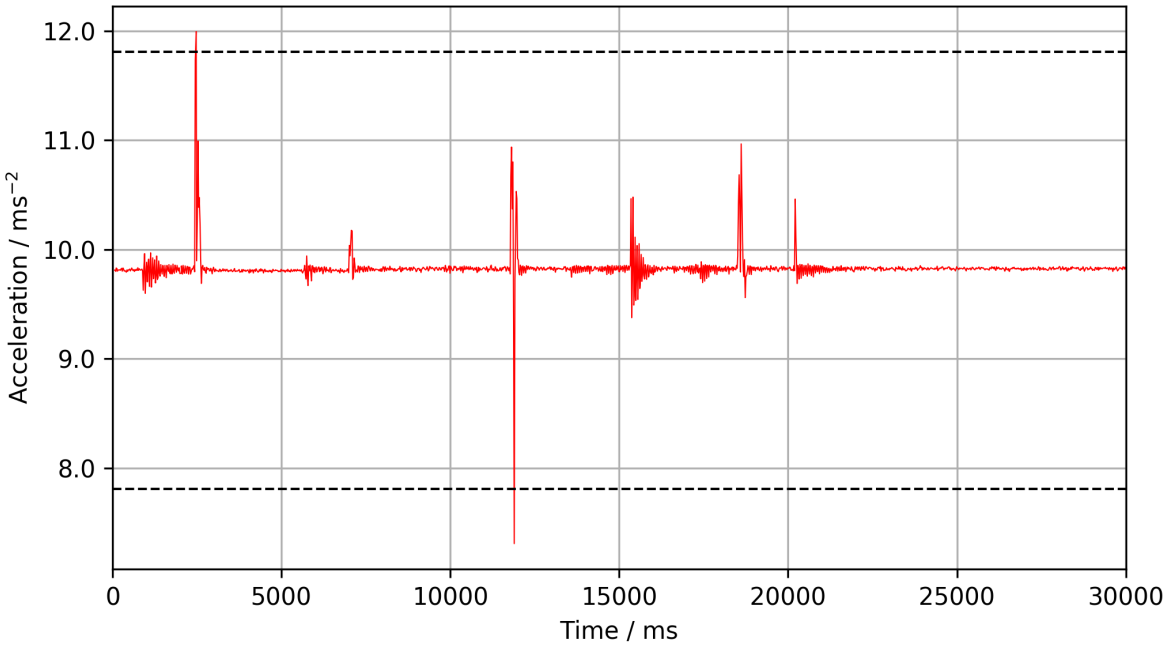


Figure 3.5: Accelerometer data taken from my smartphone. Threshold values are marked with black lines. This scenario involved moving the phone on a table; a naive algorithm would (*unreasonably*) report motion.

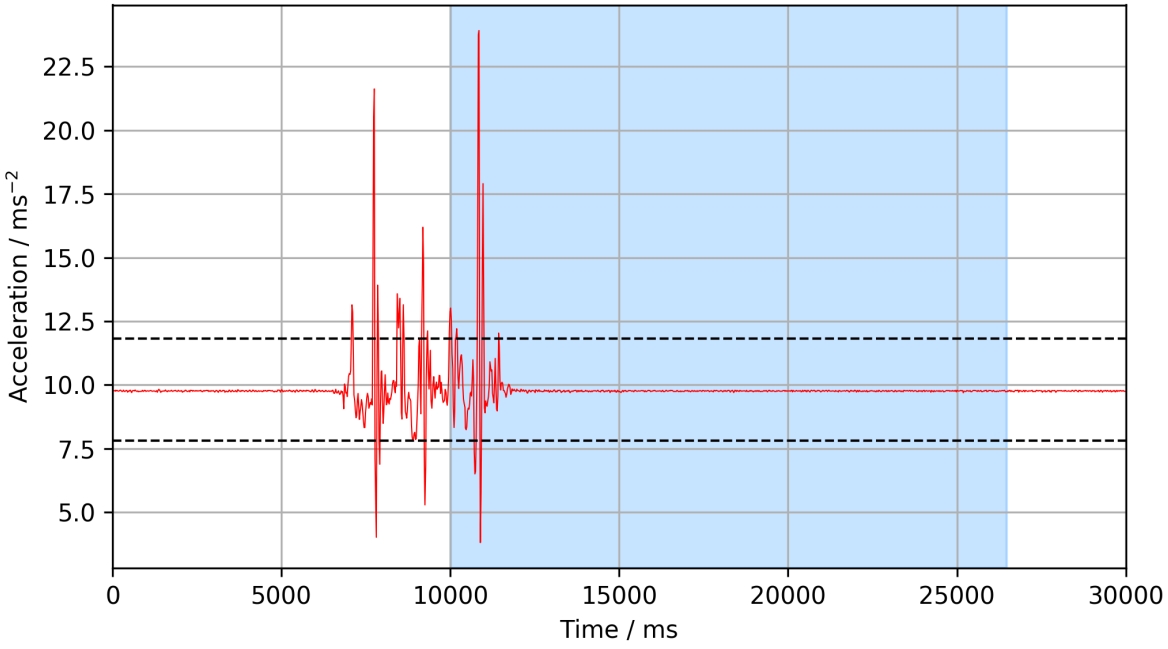


Figure 3.6: Walking data collected from my smartphone. As expected, the algorithm activates when I start walking, with a delay of approximately 3 seconds. Periods when the algorithm would report motion shown in blue (no additional random delay added).

3.1.4 Network

The client communicates to the server using the interface in Figure 3.8 over HTTPS, since the information sent is sensitive.

There is a trade-off between observation latency and battery consumption. The compromise is to batch scan records, with a period of 60 seconds being chosen when the device is still. The period is decreased to 15 seconds during periods of motion so that the location graph is updated as soon as possible. This is implemented using an event loop; the device does nothing unless there are outstanding records.

Location measurements are sent when they are made. This behaviour is justified by Section 4.4.3: devices move infrequently relative to the rate scan records are sent at.

3.1.5 Logging

Events are timestamped and logged for later analysis. The application records:

1. Motion status
2. AIDs transmitted and the generation scheme
3. Scan records
4. Location data
5. Interactions with the server
6. Battery percentage (every 10 minutes)

I used Logback [17] as the logging framework; my requirements could be fulfilled by an existing library so it is unnecessary to write my own framework. Log events were output to rolling files kept on the device until they were sent to the server. Logs are sent to the server when devices are charging and connected to WiFi; the GUI also provides a button to force submission. This functionality is implemented with Android's JobScheduler framework [21] to minimise battery impact.

Logs are stored in compliance with Android's suggested security practices [8].

3.1.6 Fetching of Parameters

The following settings can be obtained from the server:

1. Whether to enable data collection
2. Periods for batching scan records
3. Scanner period parameters

4. Motion algorithm parameters (Section 3.1.2)

Parameters are obtained every 12 hours using the JobScheduler [21] framework. They are logged, and the service is restarted with them.

This functionality was added in case it was necessary to change parameters during the deployment; otherwise I would need to distribute a new application to all participants. It was not necessary to change parameters during the experiment.

3.1.7 Android Restrictions

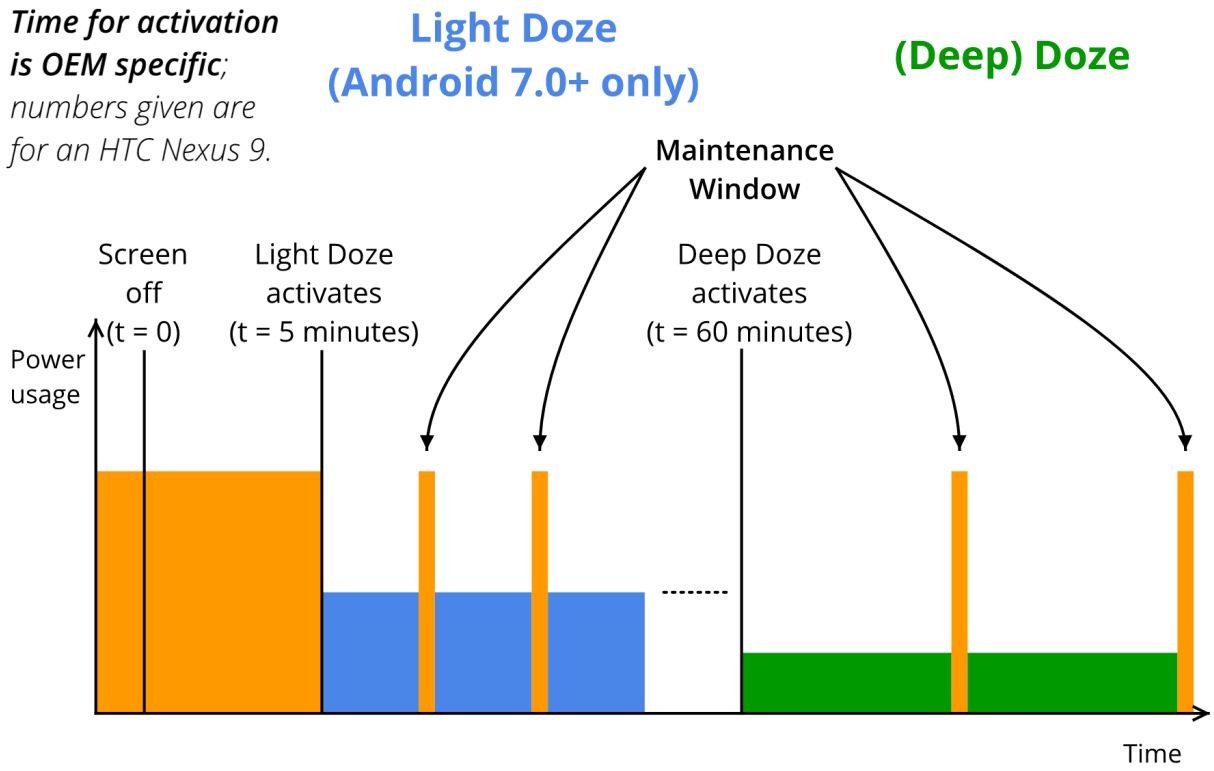


Figure 3.7: Doze activation timeline.

Doze was introduced with Android 6.0 in late 2015, and imposes restrictions on applications to improve battery life [22]. Background processing and access to the network is deferred into maintenance windows. Android 7.0 introduced a 'light' Doze mode, which activates more quickly, but takes less severe measures; the timeline for activation is in Figure 3.7.

I get around these restrictions by adding a wakelock to the foreground service; a wakelock prevents the device from fully sleeping. Doze ignores the wakelock unless the user whitelists the application [22], so the user is prompted to whitelist the application after installation. The client ceased functioning if the device had been idle for longer than 10 minutes without a wakelock.

If the wakelock is released, it is impossible to predict when it will be possible to re-capture it. Alarms are ineffective: they can only wake the device up for a short period every 9 minutes [22] — too infrequent for this use-case.

A similar experiment with dedicated hardware achieved 20 hours runtime using a battery of 100 mAh — 3.5% the size of a typical smartphone battery [43]. This experiment did not use GPS or WAN, however.

3.2 Server

In this section, I discuss the implementation of the server.

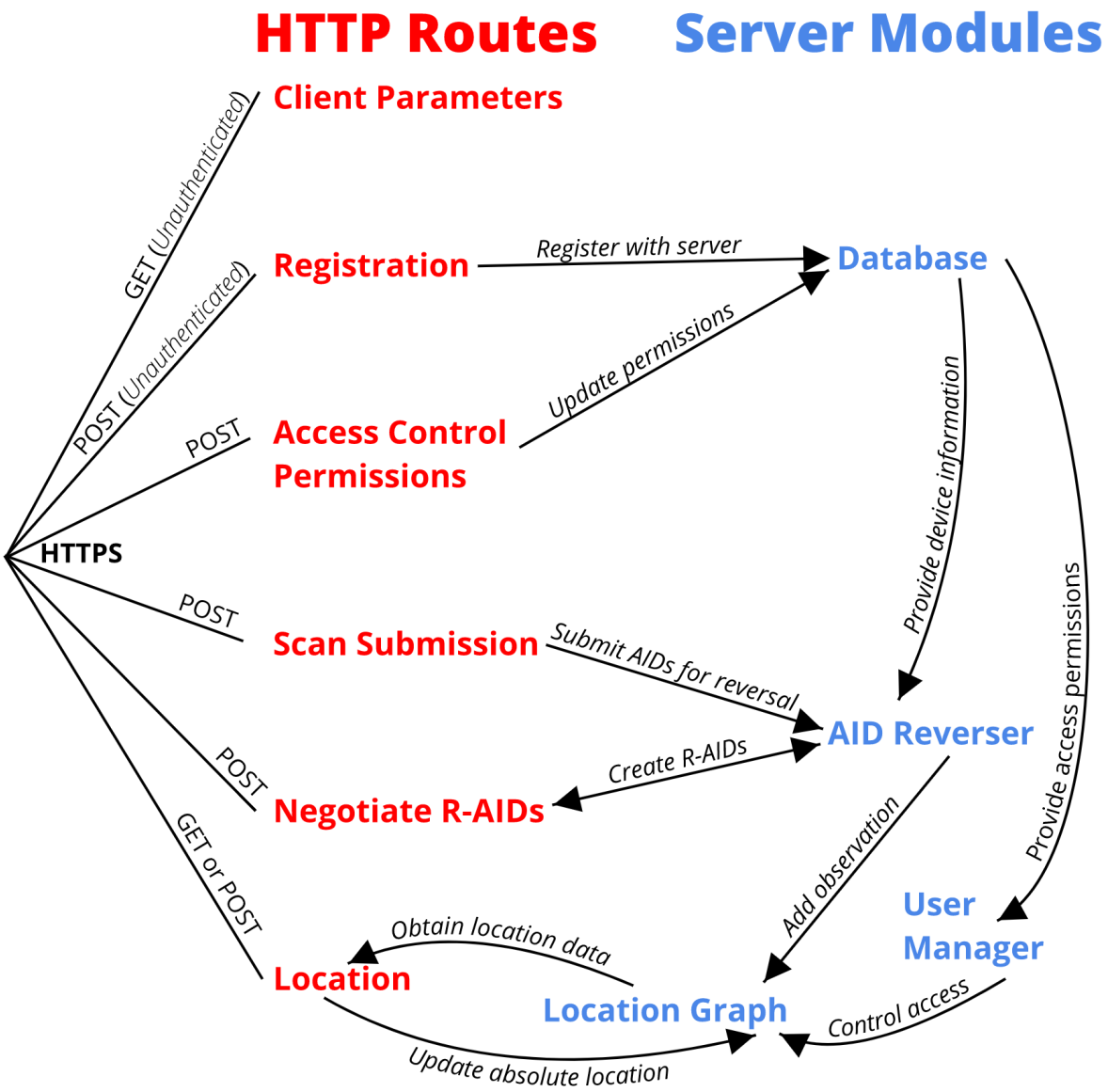


Figure 3.8: Interactions with server.

3.2.1 Client-Server Interface

The client and server communicate over HTTPS, with HTTP basic authentication [35] used to protect routes where necessary. The server's interface is visible in Figure 3.8.

3.2.2 Databases

An HSQLDB [16] database is used to store information. Figure 3.9 displays the tables in the database.

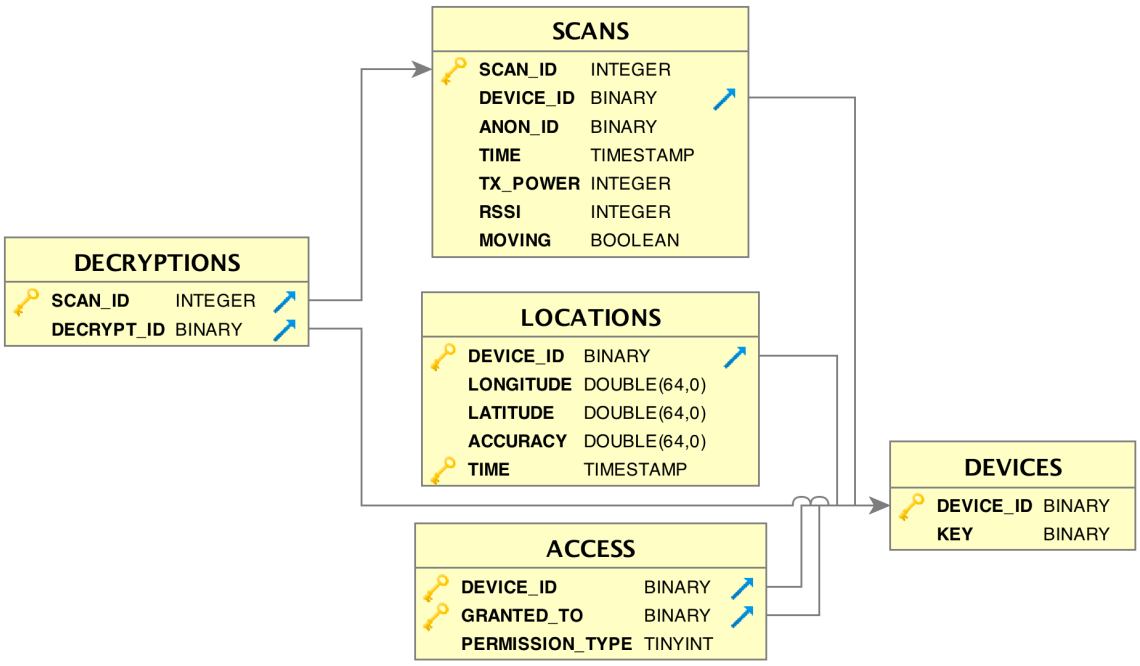


Figure 3.9: Tables in the database, with primary keys and foreign key constraints indicated.

The `devices` table stores device identifier and key pairs; in a production scenario, it would be good practice to encrypt this table. The `scans` table stores scan records submitted; when an AID is reversed, a record is inserted into the `decryptions` table. The `locations` table stores location updates submitted, and the `access` table stores access control information, explained in Section 3.2.3.

3.2.3 Access Control

For privacy reasons, the server cannot return location information about a user to arbitrary requesting users; there are, however, use-cases where a user may wish to share their location with others. The server maintains a whitelist for each user recording which other users may access their location data. Two levels of access are identified: [51]

1. **Friend**: if A is a friend of B, then A is permitted to track B: the device identifier for B is returned unchanged.

2. **Acquaintance**: if A is an acquaintance of B, then A is allowed to track B, but with *a time limited pseudonym*: the true identity of B is not revealed to A. This access may be useful for safety purposes, where it is necessary to know if users are in a building, but not their identity.

Pseudonym generation functionality is implemented using a hashtable with time-expiring keys: if the required key does not exist, then a random identifier is generated, and entry is inserted into the hashtable. Entries expire after 10 minutes. If devices A and B are acquaintances of C, then A and B receive *different* identifiers for C. The random identifier is checked against the database to ensure that it does not correspond to any valid device identifier; it is re-generated if required.

3.2.4 AID Reversal

When an observation is submitted, the server begins reversing it. The computation is asynchronous: the submitting client receives an immediate response from the server acknowledging the submission, but no other information. Each AID scheme is different, and are described in Sections 3.3 and 3.4.

3.2.5 Location Graph

As an extension, I implemented a graph which can return location information to a querying node, using the JUNG graph library [9].

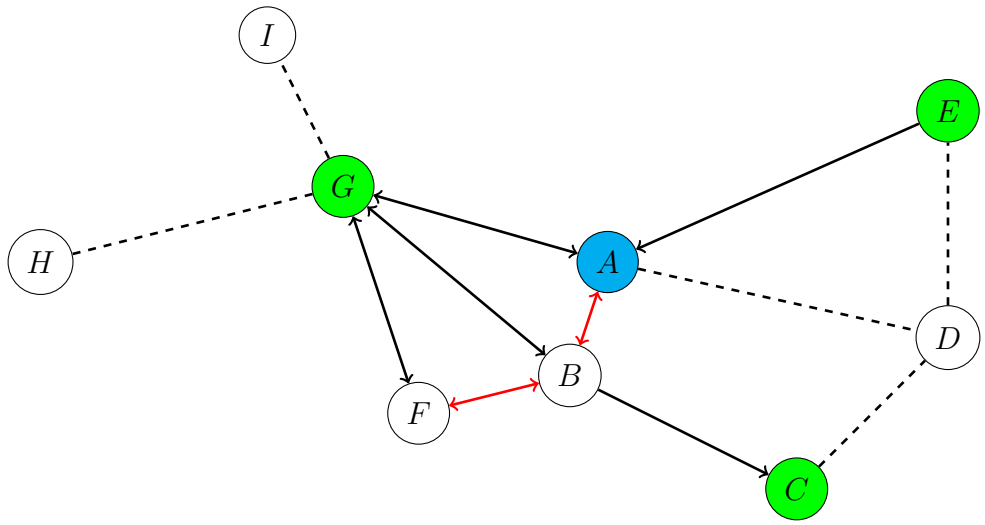


Figure 3.10: Example location graph. $X \to Y$ indicates X has observed Y recently, and the edge is *live*; close edges are red. Dashed edges are not live. Here, A is requesting location information, and green nodes have sent their location (access permissions not indicated).

Exploitation of *spatial locality* is essential for one anonymisation scheme. If device A observes device B, then A is likely to observe devices B can see as well. The design of the graph explicitly takes account of this fact. We will return to this in Section 3.3.1.

Each device is a vertex in the graph. Edges are undirected: they represent a data structure which stores observations made in either direction. Each edge includes methods to query whether the edge is currently deemed as *live*, and if it is live, whether the two devices are in close proximity (deduced from RSSI measurements). An edge is live if there has been a recent observation in either direction: within the past 3 minutes, to account for the period used in the client for scanning.

Each vertex stores location data with its timestamp. If the device submits scan records that indicate the device was moving more recently than the location timestamp, the location is invalidated. The measurement is also invalidated if the server receives no correspondence from the device for 10 minutes: we assume that the device cannot inform the server of movements. As shown in the evaluation, it is uncommon for a device to be still for longer than 10 minutes continuously.

When a scan record is received indicating it was made with the observing device moving, an edge between the two devices involved is created if it did not already exist. If the edge already existed, all observations are flushed: they are stale. This edge is *not live*, but provides connectivity in the graph, increasing the probability that using the graph as a heuristic returns a hit.

```
1  void addObservation(DecryptedScanRecord record) {
2          final DeviceNode observing = getNodeFromId(record.getDeviceId())
       ;
3          final DeviceNode observed = getNodeFromId(record.
       getDecryptedIdentifier());
4
5          ScanEdge connectingEdge = graph.findEdge(observing, observed);
6
7          if (connectingEdge == null) {
8                  connectingEdge = new ScanEdge(observing, observed,
       OBS_STALENESS_MILLIS);
9                  graph.addEdge(connectingEdge, observing, observed);
10         }
11         // observations flushed if observer was moving
12         connectingEdge.addScanRecord(record);
13
14         // measurement was made while moving -> invalidate location
       measurement if necessary
15         if (record.isMoving()) {
16                 synchronized (observing) {
17                         final Optional<Instant> timeOfMeasurement =
       observing.getTimeOfLocationMeasurement();
18                         timeOfMeasurement.ifPresent(locTime -> {
19                                 if (locTime.isBefore(record.
       getObservedAt()))
20                                         observing.invalidateLocation();
21                         });
```

```
22                      }
23              }
24  }
```

A device is removed from the graph if it does not report any observations, and no other devices observe it, for 10 minutes. Edges are only removed if one of their end-vertices is removed. This makes the heuristic more robust: a pair of devices may lose connection to WAN and therefore not report observations of each other, but still transmit AIDs which are reported by other devices which *do* have WAN access. The downside to not pruning edges is increased memory usage; in this case I consider having a better heuristic a worthwhile trade-off.

Deciding Proximity

Path loss (transmission power minus received signal strength) measurements are smoothed using a Kalman Filter (theory described in Section 2.2). Parameter choice is not an exact science: experimentation and domain knowledge are required. Measurement variance was chosen as 25 dBm^2, and process variance was chosen as 0.1 dBm^2. A cutoff path loss value of 70 dBm was chosen: experimentally this corresponded to a separation of 1 m between devices. If observations in *either* direction had an 80% probability of being under this cutoff then the devices are close. A divergence in path loss in opposite directions indicates multipath or shadowing effects increasing path loss, so we ignore the larger estimate.

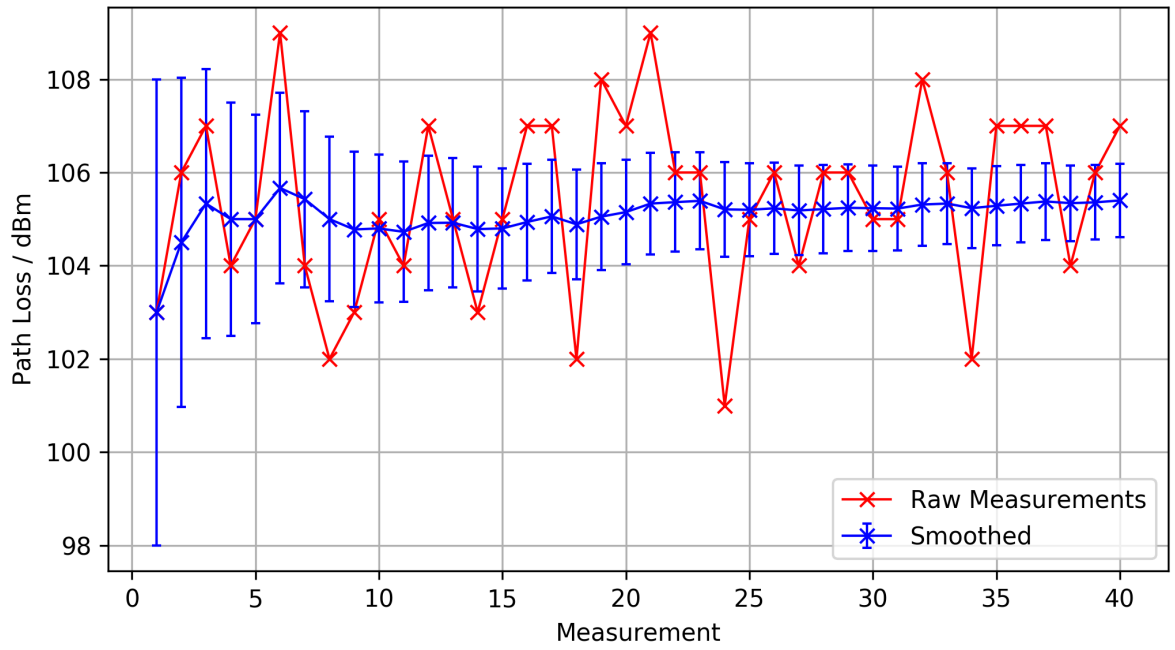


Figure 3.11: Path loss measurements filtered with a Kalman filter.

Absolute Location Estimation

Estimating absolute location data for a requesting device is impossible with Bluetooth. Each smartphone can only obtain a location fix with a claimed accuracy of 15-20m indoors; RSSI is an unreliable measurement since it is easily affected by environmental factors. Robust techniques do exist for probabilistic localisation, such as Monte Carlo localisation [48]. Convolving the probability distributions for location and signal strength together results in distributions that are unhelpful for localisation, however. Signal strength is used for augmenting the proximity information — not for full localisation.

3.3 Bluetooth 4 Schemes

This section provides details for the BLE methods for anonymisation. I cover three methods to create 16 byte long AIDs. Each method has already been published [50, 36, 51]. A name has been given to each method to disambiguate them.

16 bytes is insufficient to use public key cryptography with a sensible key size. It cannot be assumed that devices *always* have a network connection, and it is preferable to avoid contacting the server constantly to reduce power consumption.

3.3.1 E-AIDs

On registering with the server, devices receive a 64 bit device identifier (DID) and a 128 bit device key (DK).

Client Side

Device i generates an AID by concatenating its device identifier, DID_i, with an 8 byte randomly generated nonce, n. The 16 bytes are encrypted using AES in Electronic Code Book (ECB) mode [50, 51]. ECB mode is appropriate since *exactly one block* is being encrypted; additionally, we avoid the overhead of an IV.

$$\text{AID}_i = \text{ENC}_{\text{DK}_i}(\text{DID}_i || n)$$

Whenever the device needs to create a new AID, it regenerates n and re-runs the encryption.

Server Side

When the server receives an AID it has no indication which device generated it; it is impossible to pre-compute AIDs for each device. It must try *every* (DID, DK) pair until it finds a DK which returns a plaintext with the first 8 bytes matching the corresponding DID.

A device does not have to contact the server at all in order to change its AID using this scheme, fulfilling a success criterion.

The key search is linear in the number of devices registered with the server. The search is, however, embarrassingly parallel, and there are heuristics.

Heuristics

Three heuristics have been identified to reduce the search space:

1. **Caching**: AIDs are retained for 20 minutes after they were *last obtained* from the cache.
2. **Access Control**: If a user A has allowed another user B to have access to their location, it is assumed that A and B will be co-located frequently.
3. **Spatial Locality**: The location graph is queried for devices in the graph reachable from the reporting device: the query includes edges which are not considered "live" currently. Spatial locality can be exploited using other means, e.g. cell tower information [51] or geo-IP; these coarser heuristics were not implemented, however.

Code for searching in case of a cache miss is given:

```
1  Optional<DeviceIdentifier> search(DeviceIdentifier receivedFrom,
       AnonIdentifier anonIdentifier, boolean parallel) {
2
3      Collection<DeviceRegistration> registrations;
4
5      // Spatial locality heuristic
6      if (locationGraph != null) {
7          registrations = devices.getRegsForIds(locationGraph.
       getSuggestedForReversal(receivedFrom));
8          Optional<DeviceRegistration> matching = reverseAgainstRegs(
       anonIdentifier, registrations, parallel);
9          if (matching.isPresent())
10             return Optional.of(matching.get().getDid());
11     }
12
13     // Access permissions heuristic
14     registrations = devices.getRegsForIds(permissions.
       allDevicesAllowingAny(receivedFrom));
15     if (registrations.size() > 0) {
16         Optional<DeviceRegistration> matching = reverseAgainstRegs(
       anonIdentifier, registrations, parallel);
17         if (matching.isPresent())
18             return Optional.of(matching.get().getDid());
19     }
20
21     registrations = devices.getAllRegistrations();
```

```
22      Optional<DeviceRegistration> matching = reverseAgainstRegs(
        anonIdentifier, registrations, parallel);
23      if (matching.isPresent())
24          return Optional.of(matching.get().getDid());
25      else
26          return Optional.empty();
27  }
```

Optimisations

I implemented two major optimisations:

1. **Caching Cipher Instances**: Cipher initialisation takes significant time.
2. **Queuing Identical Submissions**: If an AID is undergoing reversal, further submissions of the same AID are queued; all callbacks are fired at the same time.

3.3.2 S-AIDs

Google developed the Eddystone EID scheme to combat spoofing of Bluetooth beacons, and for preserving user privacy with wearable devices [36]. This method requires that the client has a clock loosely synchronised with the server.

Google argue this technique is suitable for static Bluetooth beacons; in reality, static beacons drift beyond the acceptable deviation of 43 seconds (for Google's implementation [39]) without any external references. Smartphones *do* have access to external references, and are a better choice for this scheme.

Client Side

Device i generates an AID by encrypting a timestamp since an epoch, padded out to 16 bytes [39]. The timestamp is synchronised to 30 second periods in this implementation. The AES cipher is applied in ECB mode; as with E-AIDs, this is acceptable since it is *exactly one block*.

$$\mathrm{AID}_i = \mathrm{ENC}_{\mathrm{DK}_i}(\lfloor T/P \rfloor)$$

where T is the current timestamp, and P is the period being synchronised to.

Server Side

The server can predict the AIDs generated by each device; therefore, it can pre-compute the AIDs that can be reported in a given time period and avoid the search procedure. Reversing an AID reduces to a lookup in a hashtable.

The server ignores AIDs that were created over a certain number of periods ago. There are two good reasons for doing this:

1. **Memory concerns**: Ericsson predict there will 6 billion smartphones users in 2020 [4]. Assuming an install base of 1 billion devices, we need 24 GB to store the (DID, AID) pairs per period; this estimate ignores all data structure overheads.

2. **Spoofing**: It is only possible to spoof another device for as long as the AID is valid: once it has expired, the server will ignore it.

The server does not consider more than 12 periods, and the client does not use an AID generated for longer than 10 periods, so the client's clock can be out of sync with the server by ± 1 period.

3.3.3 R-AIDs

This scheme requires a WAN connection, unlike the previous two schemes; the power consumption impact is reduced by *batching* requests — the client does not need to contact the server every time it needs change its AID.

The client sends an *authenticated* request to the server, which creates a list of AIDs using a secure random number generator. The server calculates a lifetime, and enters each AID into a hashtable supporting time-expiring keys; all the AIDs expire at the same time. The list of AIDs and their lifetime is returned to the client for advertising. The server can vary the number of AIDs provided, and their lifetime, or reject the request if it is experiencing high load: the client can fall back onto another technique. For this implementation, the server always returns 20 AIDs lasting for 20 minutes.

An observation of an AID generated with this method is reversed by a lookup in the hashtable. When an AID expires its hashtable entry is deleted, so an adversary cannot continue spoofing another device.

The client uses R-AIDs if possible, then S-AIDs, then E-AIDs, to minimise server reversal time.

3.4 Bluetooth 5 Schemes

In this section, I discuss a novel anonymisation scheme for Bluetooth 5 that exploits the increased advertisement packet size. The packet is large enough to make public key cryptography viable. To my knowledge, I am the first to propose an anonymisation scheme for Bluetooth 5.

3.4.1 Benefits of Public Key Cryptography

Public key cryptography can improve the server's scalability. We assume that the server has a public key known by clients. There are two important properties that are being exploited:

1. **Constant de-anonymisation time**: The server takes constant time to decrypt each AID: there is no expensive key search.

2. **Reduced memory requirements**: It is possible to design the plaintext so server-side pre-computation is unnecessary.

There are the usual downsides with public key cryptography: leakage of the server's private key represents a single point of failure for the system.

1. **All AIDs become readable**: Possession of the private key allows you to decrypt all AIDs. If the device identifier is in the plaintext, it is trivial for an adversary to link together AIDs; the scheme I chose offers clients *deniability*, discussed in the evaluation.

2. **No breach feedback**: The server has no way of knowing whether its private key has been leaked or not.

One solution for these issues is to regularly rotate the server's key pair. Changing the key pair requires devices to obtain the key pair from the server, precluding the usage of static (dumb) beacons.

3.4.2 Proposed Format (P-AID)

We continue to assume that devices register with the server, and receive a device identifier (DID) and a *symmetric* device key (DK) as before; the key size is increased to 256 bits, recommended by cryptographic primitive using the key.

Client Side

The server needs to verify a device's identity without including the shared secret in the plaintext, in case the server's private key is leaked. AIDs generated with this scheme should be *timestamped*, so that they expire. The plaintext in Figure 3.12 aims to achieve these requirements.

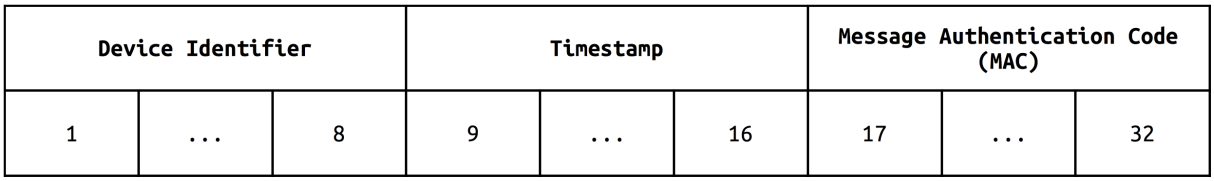


Figure 3.12: P-AID plaintext format.

The plaintext consists of the device identifier concatenated with the Unix timestamp. The HMAC-SHA256-128 algorithm is then used to generate a MAC and is concatenated to the end of the plaintext. A public key encryption scheme encrypts the entire plaintext using the server's public key to produce an AID.

It is unnecessary to use digital signatures: non-repudiation is *not* a desirable property, and creating a signature is more expensive than creating a MAC. The MAC offers devices *deniability*, assuming their device key has not been leaked as well.

Server Side

Reversal is the same regardless of the public key encryption scheme used. First the AID is decrypted and parsed. The server retains a high watermark: any AIDs timestamped before this are rejected. The MAC is verified last since it is the most expensive check.

```
1  // Skip if age is too old
2  final long now = Instant.now().getMillis();
3  if (now - aidMillis > ServerParams.PUB_KEY_AID_MAX_AGE_MINUTES * 60 * 1
       _000)
4          return Optional.absent();
5
6  // Skip if in the future by > 1 minute
7  if (aidMillis - now > 1000 * 60)
8          return Optional.absent();
9
10 // Get registration from DB
11 final Optional<DeviceRegistrationV5> registration = devicesV5.getKeys(
       did);
12
13 if (!registration.isPresent())
14         return Optional.absent();
15
16 // Check MAC
17 final byte[] correctMac = RsaAidGenerator.createMac(
18         Bytes.concat(didDecrypted, millisDecrypted),
19         registration.get().getMacKey()
20 );
21
22 if (!Arrays.equals(correctMac, macDecrypted))
23         return Optional.absent();
24
25 return Optional.of(did);
```

3.4.3 RSA

The issue with RSA is key size: even with the size extension provided by Bluetooth 5, the recommended key size of 2048 bits cannot be used without resorting to chaining (described in Section 2.1). The next smallest common key size (1024 bits) is already considered to be insecure [32], so this project uses a non-standard key size of 1792 bits.

Chaining is avoided because *all* the constituent packets of the advertisement must be received; this is unlikely if the observing device is moving.

The client encrypts with the server's public key using the RSA PKCS #1 standard with OAEP padding, creating an AID of length 224 bytes [41].

3.4.4 Elliptic Curve Cryptography (ECC)

ECC uses a smaller key size for the same security level as RSA, meaning a shorter AID can be generated. We use the Elliptic Curve Integrated Encryption Scheme (ECIES), a hybrid encryption system [1].

ECIES is a generic framework: it does not prescribe which key derivation function (KDF), or authenticated encryption scheme *must* be used, although primitives are provided in the specification for cross-compatibility. The following primitives have been chosen:

- **KDF**: KDF2 with SHA256.

  This primitive is recommended by the IEEE, who standardise ECIES [1]. The library used did not support any other KDF [28].

- **MAC**: HMAC-SHA256-128.

The plaintext is encrypted by using the KDF as a stream cipher [1]. It is also possible to use the KDF to obtain a key for a block cipher, and encrypting with the block cipher in a suitable mode of operation. This method requires an initialisation vector, lengthening the AID.

Assuming the ECC key size is chosen to be 256 bits (corresponding to a 128 bit security level [32]), the length of the AID can be calculated as:

1. 257 bits for the curve point.
2. 256 bits to encrypt the plaintext defined in Figure 3.12.
3. 128 bits for the MAC.

Giving a total of 81 bytes.

3.4.5 Including GPS Data

With ECIES there is enough space for devices to share their location without using the server as an intermediary. The trailer in Figure 3.13 can be appended onto AID transmissions by devices which have been able to obtain a location fix.

Each value has the following interpretation:

Latitude			Longitude			Altitude		LACC	AACC	Timestamp		
1	...	8	9	...	16	17	18	19	20	21	...	28

Figure 3.13: Location trailer format.

- **Latitude and Longitude**: 8 byte IEEE754 floating point numbers representing degrees.
- **Altitude**: Signed 2 byte integer representing metres above sea level.
- **AACC and LACC**: Unsigned bytes representing altitude and location accuracy in metres.
- **Timestamp**: Unsigned 8 byte integer Unix timestamp (in milliseconds) for the measurement.

The definition of accuracy mirrors the one used in the Android standard library: the radius of 68% confidence (i.e. the true value has a 68% chance of lying inside the radius around the estimated value).

Location measurements may leak information facilitating linkage to a static identity: e.g. measurements taken with Google's location library commonly report an accuracy of 15m (Section 4.4.5). Fuzzing is a possible solution: the measurements are useful for deducing *context* (discussed in the introduction), which does not always require very accurate measurements.

3.4.6 ECC compared to RSA

The shorter AID produced by using ECIES is preferable for 3 reasons:

1. Longer transmissions are harder for the listener to receive without error because of environmental factors, and collision domain increasing with message length.
2. Longer AIDs result in longer messages being sent to the server over WAN, using more power.
3. Transmitting a longer advertisement requires more power.

3.5 Summary

I have presented the major components of the Android client and the de-anonymisation server. Possible anonymisation schemes were then covered: first for BLE, then for Bluetooth 5.

Some subtle implementation details are discussed in the security evaluation (Section 4.5).

Chapter 4

Evaluation

In this chapter, I show that the movement algorithm works across different individuals. I assess the server's feasibility, and analyse how it scales with the number of devices using the system. I then analyse data from a deployment of my client, and comparing it to another data set. Finally, I consider some attack vectors an adversary may use.

I will open-source the code after publication of examination results. I am not obligated to do this by third-party libraries.

4.1 Accelerometer Data

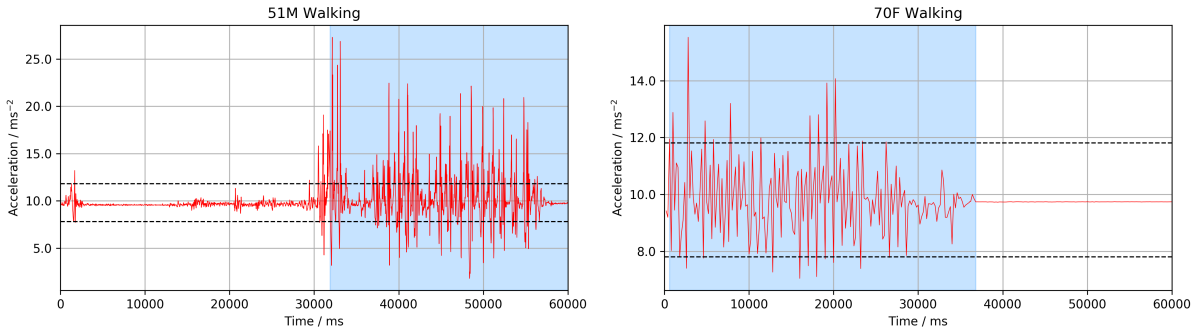


Figure 4.1: Accelerometer traces for two individuals; periods when the algorithm would report motion shown in blue.

I obtained ethical approval to record accelerometer data from volunteering participants using an application installed on their smartphones; participants gave consent to link their data with their age and gender, and were asked to behave as normal. People walk differently based on age and gender: you would expect heavier footsteps from young men than elderly women, for example. Additionally, different smartphone models provide different accelerometer data: there is calibration and sample rate variation.

Figure 4.1 shows that the motion algorithm activates as expected for both individuals, including a difficult case with an elderly woman using a device with a low sample rate.

The algorithm's high sensitivity will cause some false positives but this is acceptable. *False negatives* are unacceptable as this facilitates tracking.

4.2 SpaceLab Data Set

Following sections will refer to the SpaceLab data set [43], which was collected in 2016 from 25 employees at an architectural company over a period of a month. Participants wore special wrist mounted devices, which behaved as BLE beacons whilst also scanning for other BLE beacons in the vicinity; a step detection algorithm also ran on each device. The data set also included observations of static beacons that were installed into the building.

4.3 Anonymisation Schemes

4.3.1 Benchmarking Precautions

Benchmarks were run using OpenJDK 8 running on Ubuntu Linux 16.04, using the precautions described in Section 2.4.4. The machine had an i5-3570 processor and 16GB of RAM.

4.3.2 Reversal Performance

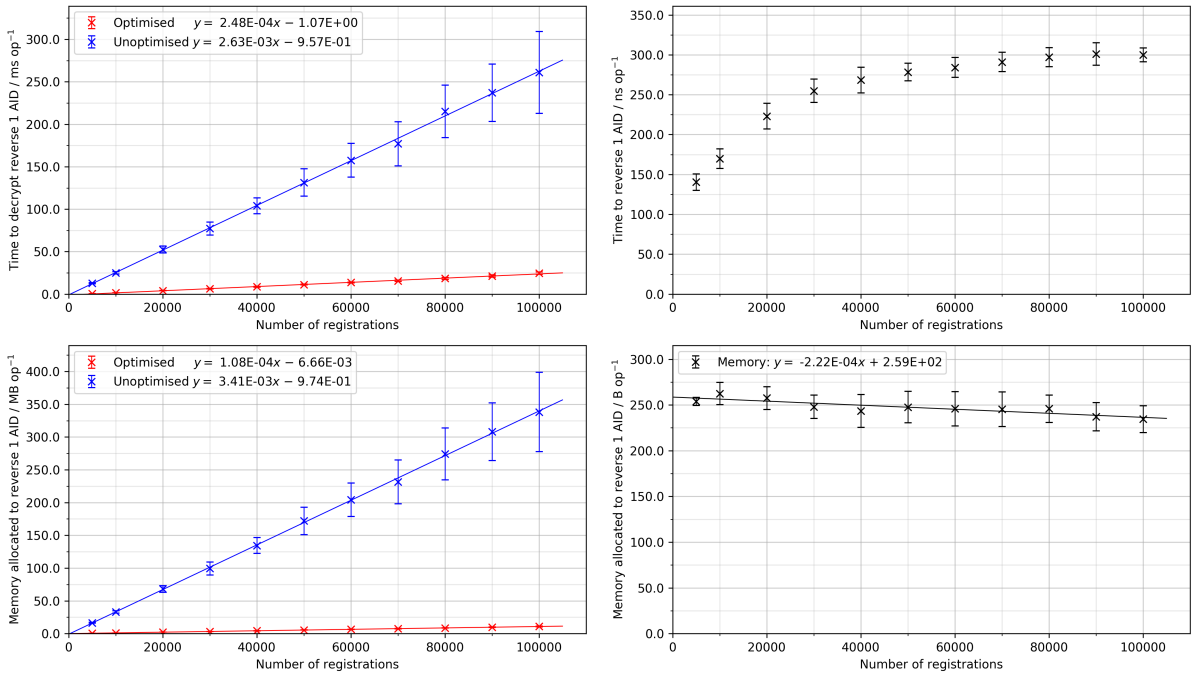
Here I benchmark how each AID scheme scales in time and memory with the number of devices registered with the server. These are basic load tests, and ignore heuristics. I demonstrate that E-AID reversal throughput scales linearly with the number of threads.

Load Testing

As seen in Figure 4.2a, E-AID reversal scales linearly in both time and memory with the size of the search space. This is expected, indicating my code is correct. The worst case latency if the heuristics are ineffective of 25 ms with a search space of 100,000.

Figure 4.2b shows different behaviour for S-AIDs: the average time scales non-linearly with the search space, while memory usage is constant and less than 300 bytes per reversal. The non-linearity is because the performance of the underlying hashtable varies with the load factor; it is unrelated to the hash function used. The hash function's uniformity was verified using the birthday paradox: as expected, the mean number of hashes generated before a collision was approximately $\sqrt{\frac{\pi}{2} \cdot 2^{32}}$.

P-AIDs use constant time and constant memory, shown in Figure 4.3; the memory usage is significantly more than S-AID, however. The results indicate a trade-off: ECIES uses an order of magnitude more memory, but an order of magnitude less time compared to RSA. This observation does not take precedence over the reasons given in Section 3.4.6 for why the client would use ECIES: the server side reversal is trivially distributable



(a) E-AID (b) S-AID

Figure 4.2: Average time and memory usage for reversal of E-AIDs and S-AIDs.

regardless of which encryption scheme is used.

I have verified two success criteria pertaining to the server's scalability: *all* the techniques could scale to a single building containing 1000 users, and each technique has been temporally bounded so that predictions about scalability can be made. For S-AIDs, performance is closely related to the underlying hash table's performance.

E-AID Parallelism

As seen in Figure 4.4, the procedure for reversing E-AIDs is *embarrassingly parallel*: computations for two separate AIDs are independent. Greater throughput can be obtained by devoting a single thread to each reversal and running several reversals in parallel, rather than using multiple threads to speed up one reversal. This is because of communication overheads: dividing the search space amongst several threads implies synchronisation. The downside to using one thread per reversal is increased latency; this is not an issue since the system does not have strict real time deadlines, and even over a search space of 100,000 registrations, the latency is less than 25 milliseconds. Both methods of exploiting parallelism were implemented and can be combined to balance latency against maximum throughput.

4.3.3 Heuristics

For the E-AID scheme to scale, heuristics that reliably decrease the size of the search space must exist. Here we consider how effective the caching and spatial locality heuristics

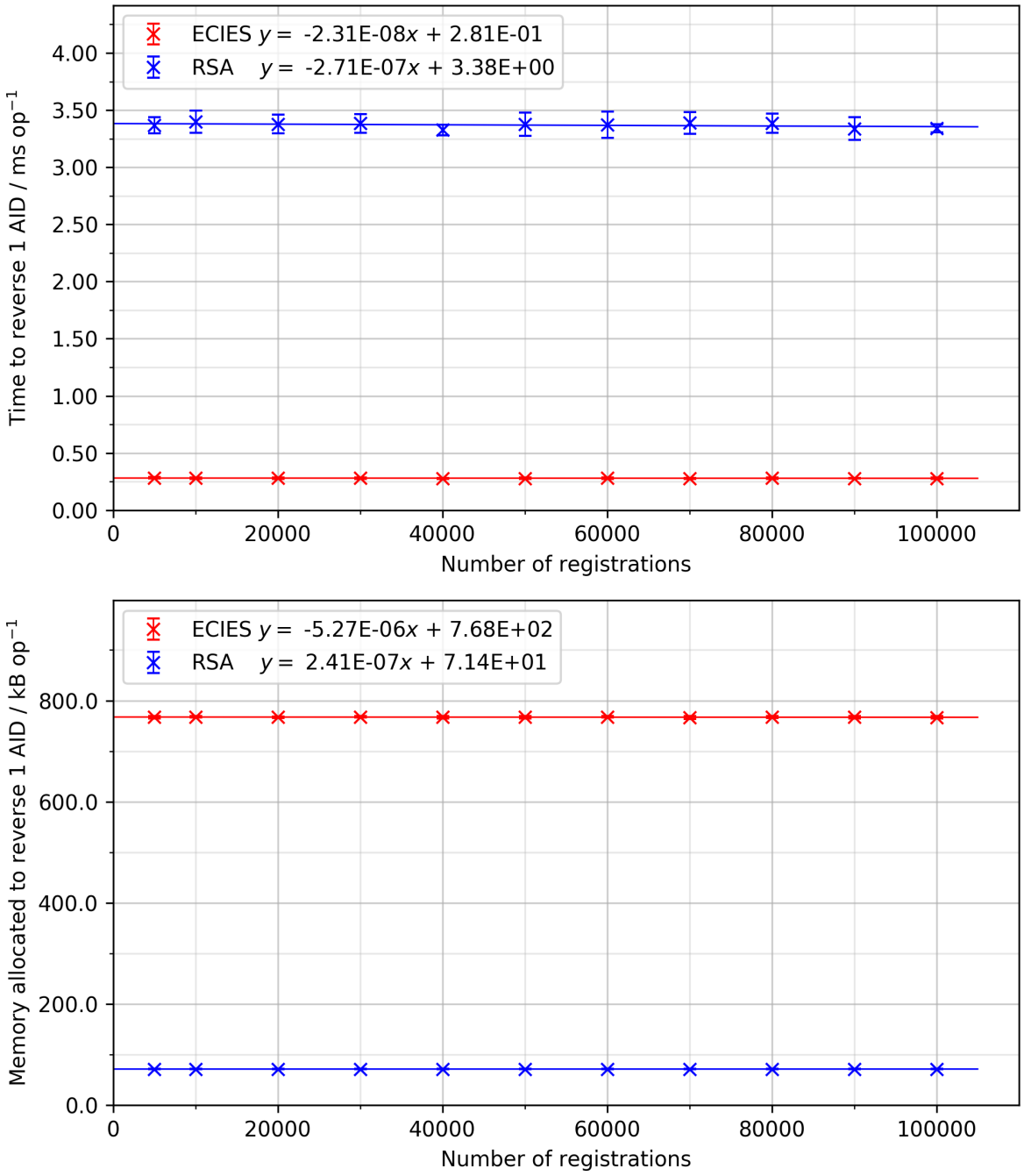


Figure 4.3: Average time and memory usage for reversal of P-AIDs generated with RSA and ECIES.

discussed in Section 3.3.1 are.

Spatial Locality

The spatial locality heuristic was tested using the SpaceLab data set. A Python script built an observation graph from the data; to simulate the delay in the server receiving scan records from clients, and the need to reverse them, observations are not inserted into the graph until 135 seconds have passed since they were made. The graph prunes

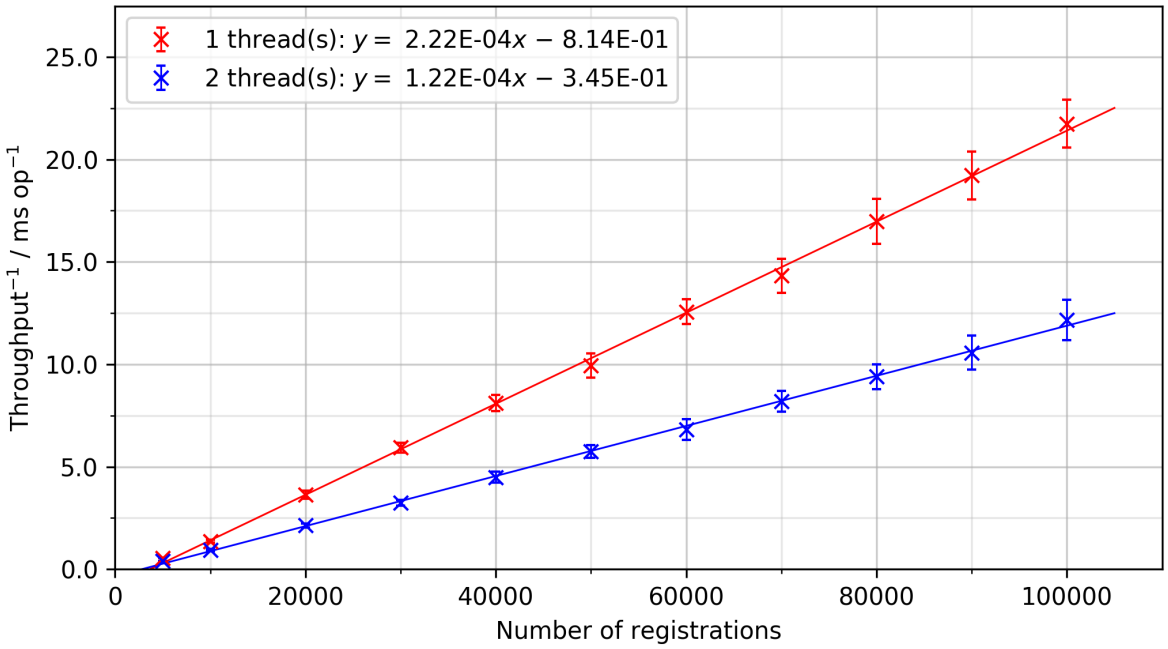
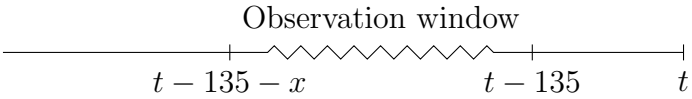


Figure 4.4: Reciprocal throughput for E-AID reversal with 1 thread and 2 threads. This quantity can be interpreted as the mean time between successful AID reversals, over all threads.

nodes as described in Section 3.2.5. Any observations made when the transmitting device was moving were discarded.

The observation window (x in the diagram) is varied: a device must be observed or make an observation at least once in the window, or it is pruned from the graph. For each observation in the logs, the graph was queried to see if there was a path between the observing and observed vertices: if there was a path we regard the heuristic as having predicted the observation.



As Figure 4.5 demonstrates, the heuristic has excellent predictive power with this data set, with over 99% of observations being predicted with a window size of 10 minutes. Extending the window increases the predictive power of the heuristic, but diminishing returns apply after the window size reaches 3 minutes.

The graph was tested for robustness by randomly discarding edges; with only 20% of edges remaining, more than 70% of observations could still be predicted. Details are omitted for space reasons.

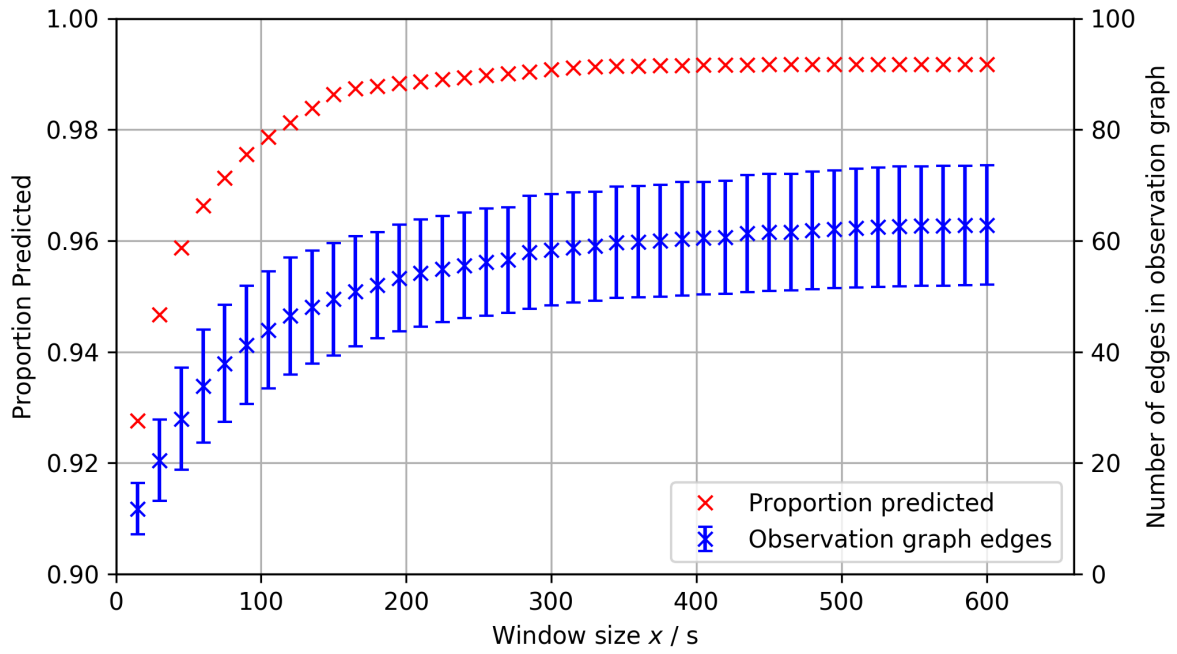


Figure 4.5: Plot showing the proportion of observations predicted by the graph, with the number of edges in the graph. For one day's data (250,000 observations).

Caching

Later in this chapter I discuss the deployment of the client. In the deployment, devices saw the same AIDs repeatedly (median 9 repetitions); this can be seen in Figure 4.13.

In a dense deployment, a device will have several neighbours. For the SpaceLab data set, each device can see a median of 5 other devices [51]; this figure will increase with the range improvement provided by Bluetooth 5. If many devices can see the same AID, then there is even greater benefit to caching.

4.4 Real World Testing

In this section, I discuss the battery impact of the system, and analyse data obtained from a deployment of my client.

4.4.1 Battery Impact

A test application was written to assess the battery impact of core operations used by the client. The application was run on a Google Nexus 9 running stock Android 7.1.1 with WiFi disabled, and no other applications running; the screen was switched off immediately once the test was started. The Nexus 9 incorporates battery monitoring hardware not usually found in consumer devices [24], which allows battery charge measurements to be obtained.

To ensure that the test application continued to run once Doze activated, the procedures described in Section 3.1.7 were used. Experimentally, it was found that the device entered the light doze 5 minutes after the screen had been switched off, and a deep doze 55 minutes later.

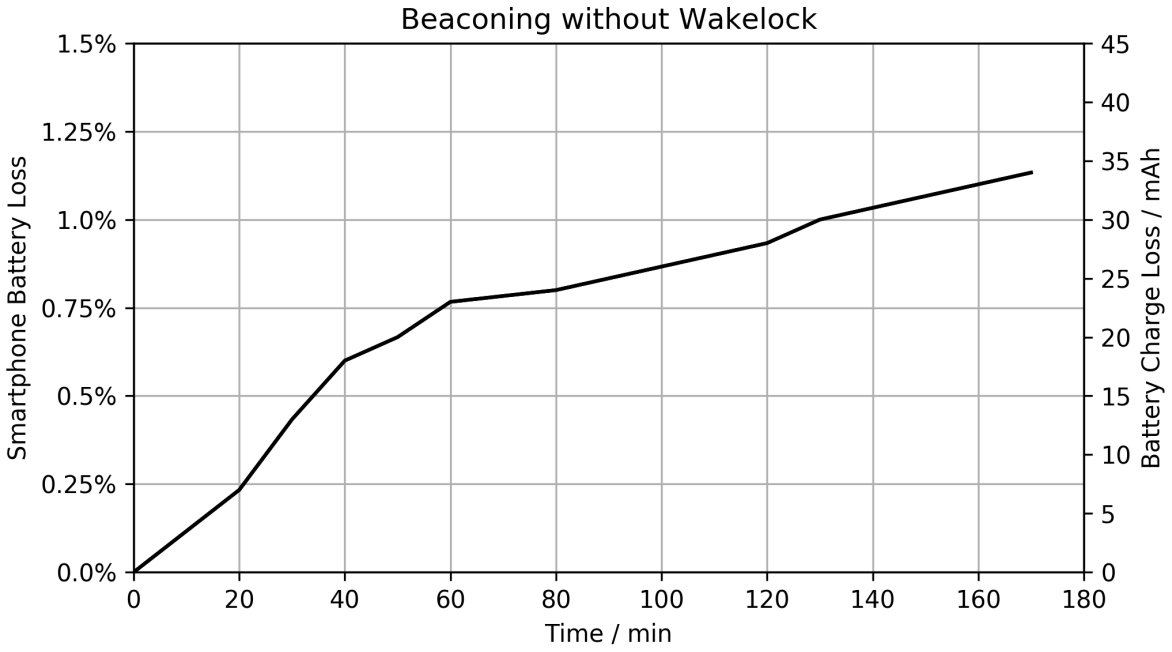


Figure 4.6: Battery drain whilst beaconing without a wakelock. The gradient changes after 1 hour, when the device enters the deep Doze state (indicated by Battery Historian). Battery percentage was calculated assuming a battery size of 3000 mAh.

Measurements were taken using Android's Battery Historian profiler [23] and are reported in Table 4.1. Values given for the second and third hour represent consumption when the system has entered a deep Doze state, and are consequently lower than the corresponding value for the first hour.

Test	Description	Hourly Draw / mAh
Control	No application running	7.7, 6.0, 6.0
Control with Wakelock	Test application holds a wakelock	38.3, 31.4, 27.9
BLE Advertising	Advertising at maximum power at frequency of 10Hz; no wakelock	26.4, 6.0, 6.0
BLE Advertising with Wakelock	Advertising at maximum power at frequency of 10Hz; wakelock held	45.5, 33.2, 34.4
BLE Scanning with Wakelock	Scanning continuously next to a device beaconing at 10Hz	70.3, 65.0, 65.0
E-AID generation with Wakelock	Generating E-AIDs every 10ms	47.8, 42.0, 43.7

Table 4.1: Energy consumption measurements. The hourly draw column gives the mean consumption for the first, second and third hours from the test start.

Advertising is a very cheap operation: approximately 20 mAh for the first hour, and then negligible afterwards. A listening device verified that the Nexus 9 was beaconing. These results are not unexpected: the BLE subsystem can operate without CPU

involvement after being given a command, so the steady state cost becomes the cost of Bluetooth transmission. Scanning is more expensive, with an extra 35 mAh required per hour compared to the relevant control; this was explained in Section 3.1.1 For E-AID generation, over 350,000 AIDs were generated per hour, introducing a cost of approximately 10 mAh over the control — implying the cost to generate one is negligible.

Battery Historian assigned a battery drain of between 20-25 mAh (0.66–0.83%) per hour on my Samsung Galaxy S7 (running Android 7.0) with the client running; this is lower than the 'control with wakelock' experiment on the Nexus 9. The S7 is possibly more efficient, and the wakelock cost could be amortised amongst other applications installed on my device. Power drain due to CPU usage was negligible: the *true* cost of running the application is unclear because the cost measured is dominated by the wakelock.

Deployment participants found the battery consumption acceptable: one did not even notice it.

4.4.2 Deployment

Ethical approval was received to deploy my client to volunteering participants, allowing me to validate that the system worked as designed. Other data sets were used to validate that the data collected was sensible.

There were 7 devices in the deployment; most students own iPhones, limiting the number of people I could ask to join the experiment. Real time scan record and location submission were disabled to reduce battery drain; devices logged when they would have contacted the server, however. The experiment ran for 14 days.

Poor manufacturer support for Android devices is a criticism of the platform, and I identified three instances where smartphones did not comply with the behaviour described in the Android documentation:

1. **Nonfunctional JobScheduler implementation by manufacturer**: One smartphone had a JobScheduler implementation which never started jobs; log submission and parameter updating did not work. Logs were lost from this device. It was possible to reconcile all observations made of this device, however, including the AID generation technique.

2. **Undocumented Service killing**: One manufacturer ignored the rules regarding Doze [22] and killed *all* services within 1 hour. My application restarted the service several times a day, so some data was still collected.

3. **Incorrect RSSI measurements**: One device had a known bug where all RSSI measurements were a constant, physically implausible, value.

Despite these issues, there was sufficient data to validate that the client was working as designed and to draw some conclusions.

Privacy Considerations

Participants were warned that the data collected could co-locate them with other participants, and that sufficient location data can de-anonymise them. A warning was given regarding the application's battery drain.

A random identifier was associated with each log; personal data is not associated with logs.

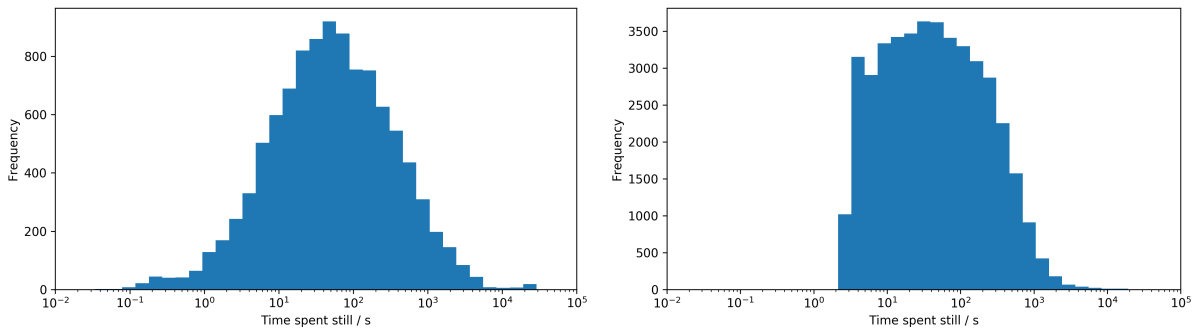
Validation

All AIDs observed in the logs were reconcilable with another device's AID, and the technique used to generate it; timestamps were checked to ensure that the observation occurred when the AID was being transmitted. This check proves that the Bluetooth components were working.

Devices were checked for co-location when an observation was made. This was done using absolute location data when it existed; as expected the client only picked up AIDs in the same location.

4.4.3 Motion Data Analysis

Data collected from my deployment was compared with the SpaceLab data set [43].

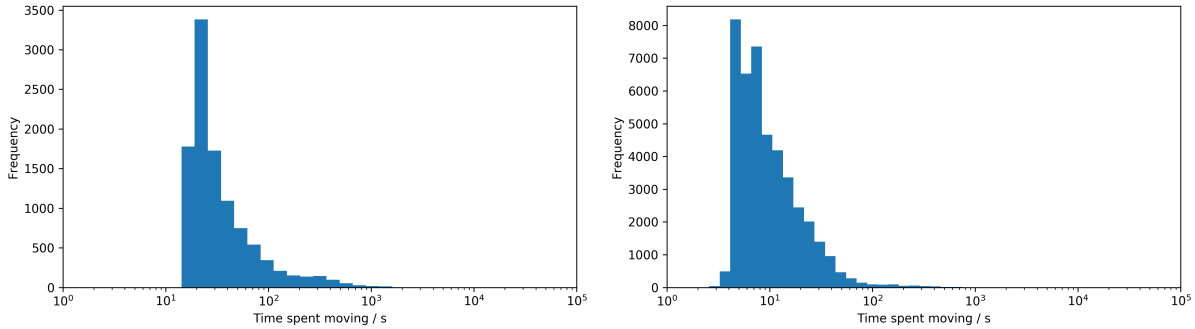


(a) Deployment (b) SpaceLab

Figure 4.7: Histograms showing length of time spent still by devices.

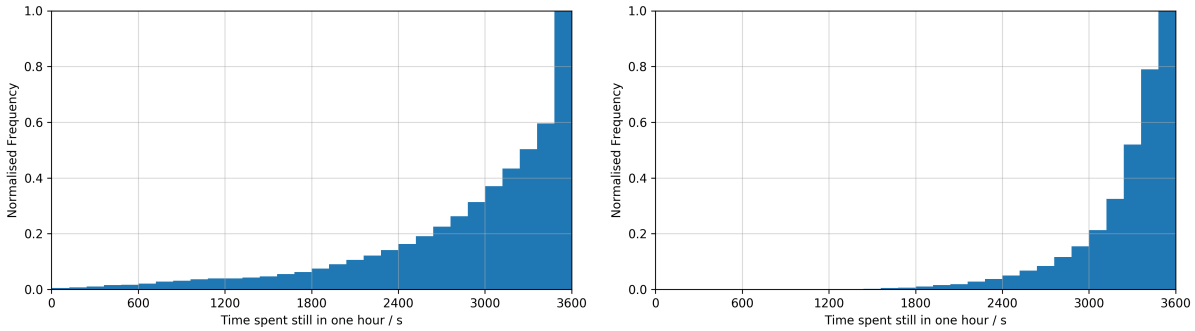
Figure 4.7 indicates that devices in my deployment had a number of still periods lasting less than 1-2 seconds; this is not visible in the SpaceLab data. The SpaceLab deployment used a step detection algorithm which instead included short still periods as part of a series of steps.

As seen in Figure 4.8, the data from my deployment and from the SpaceLab data have a similar shape. The two histograms are translated relative to each other: my motion algorithm returns to reporting still slower than the SpaceLab algorithm. Together, these figures demonstrate that the motion-sensing component was performing as expected.



(a) Deployment (b) SpaceLab

Figure 4.8: Histograms showing length of time spent moving by devices.



(a) Deployment (b) SpaceLab

Figure 4.9: Cumulative histograms showing length of time spent still per hour — and therefore beaconing.

Figure 4.9 demonstrates that each device spends a significant portion of each hour still (median 56 minutes for my deployment) — and therefore beaconing. The difference in distributions is explained by noting that the SpaceLab data was collected in an office environment, while data from my deployment considers *all* times of the day, including periods when the participant might be sleeping or moving outside.

4.4.4 AID Analysis

	E-AIDs	**S-AIDs**	**R-AIDs**
Generated	796	248	13997
Observed	34	13	1073
Observed / Generated	0.0427	0.0524	0.0767

Table 4.2: Number of AIDs generated and observed by other devices for each scheme. AIDs are counted only once if observed by multiple devices.

As seen in Table 4.2, R-AIDs were used the most: the client's preferred order for generation was R-AIDs, then S-AIDs, then E-AIDs, as mentioned in Section 3.3.3. S-AIDs were used less commonly than expected, since the client cannot use the scheme

if it is uncertain whether its clock is synchronised with the server; it is unreliable to check on Android when the clock has changed so the client relies on receiving a server timestamp within a certain tolerance (within 5 seconds[1] of the client's clock) before using the scheme. If the client was able to communicate with the server, then it used the R-AID scheme instead, explaining the low number of S-AIDs seen. The client also always started with E-AIDs, since it had not negotiated with the server yet, and was unconfident that its clock was synchronised.

The proportion of R-AIDs observed is significantly larger than the other two schemes; this is down to a bias in the selection of my participants, who are Cambridge students with access to Eduroam. Usage of R-AIDs requires a network connection which, in Cambridge at least, is more likely to be found indoors than outdoors, thanks to the excellent Eduroam coverage. Consequently, users are more likely to be in close proximity when using R-AIDs.

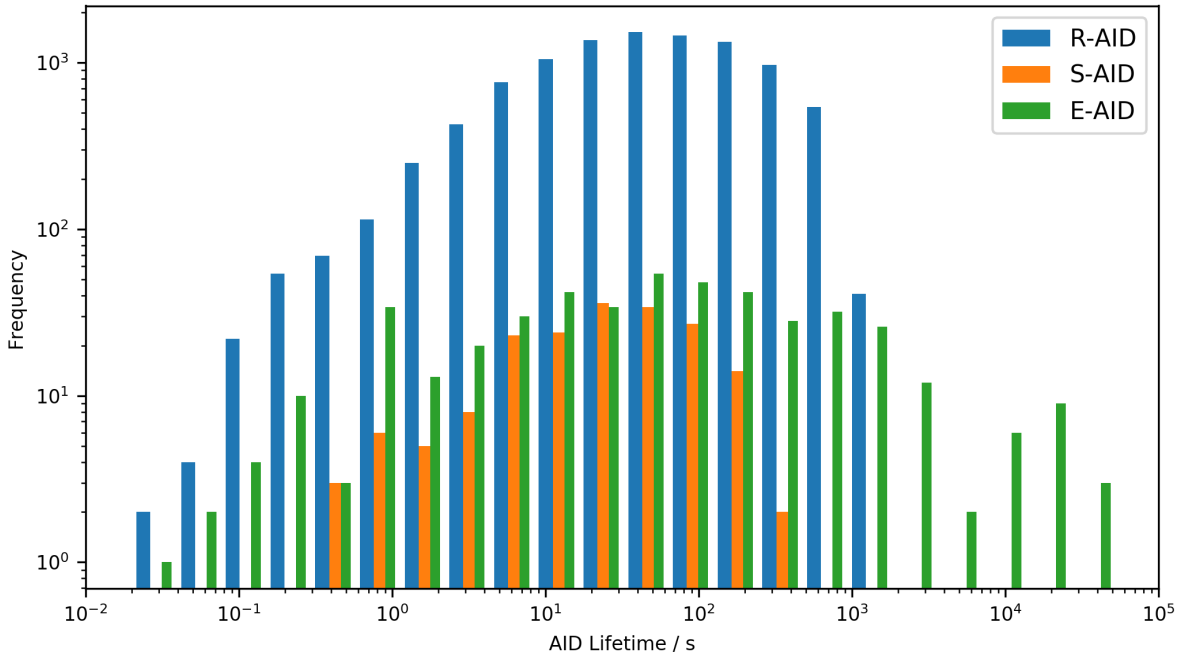
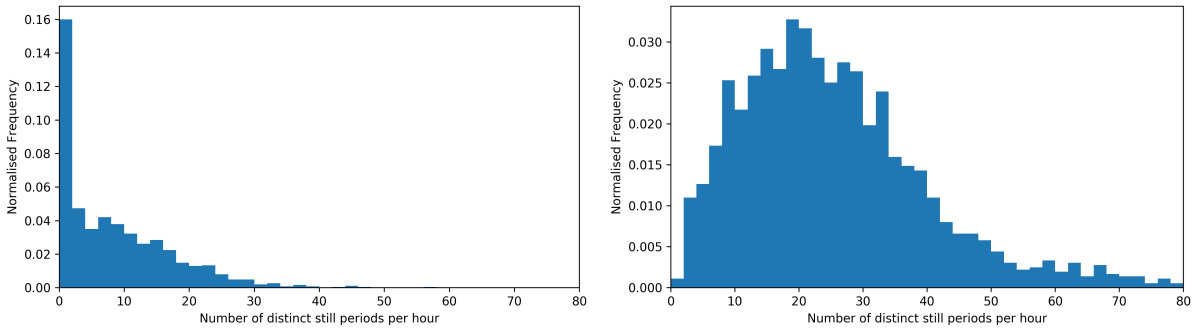


Figure 4.10: Histogram showing the lifetime of AIDs generated using each scheme. Note logarithmic axes.

Figure 4.10 shows that R-AIDs expire within 20 minutes and S-AIDs expire within 5 minutes with no exceptions; this shows that the client was executing as it should. E-AIDs form the tail of the histogram, since they do not expire. Because of AID expiry, this histogram has a smaller tail and a wider central peak than Figure 4.7.

Figure 4.11 indicates the number of distinct still periods per hour: this is analogous to the number of AIDs used per hour, *if the AIDs do not expire*. In this diagram, we see a divergence between my data and SpaceLab: the wrist-mounted devices used in the

[1]Comfortably within the latency expected when contacting a server in London using HTTP



(a) Deployment (b) SpaceLab

Figure 4.11: Histograms for the number of distinct still periods per hour.

SpaceLab experiment are more prone to false-positives. My deployment included times when people were sleeping.

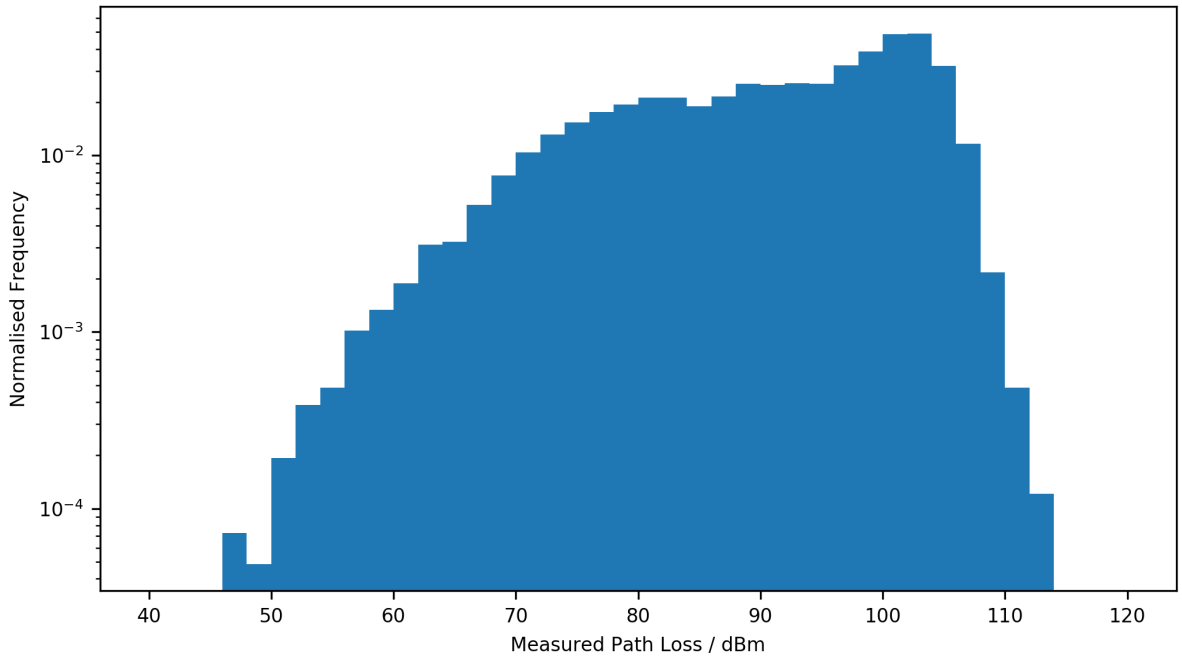


Figure 4.12: Histogram for observation path loss. Path loss is transmission power minus received power.

Path loss for observations is shown in Figure 4.12; a smaller value indicates that the two devices were positioned close to each other. The majority of time users are not very close to each other, as expected: only a small proportion of measurements are under the 70dBm cutoff for deciding closeness.

The number of repeat observations made by a single device for a given AID is shown in Figure 4.13: clearly there is merit in caching E-AIDs. Repeat observations are required for RSSI filtering to work; with a median of 9 and mean of 21.3 repetitions, the filtering

algorithm will have enough data for reliable inference in most cases.

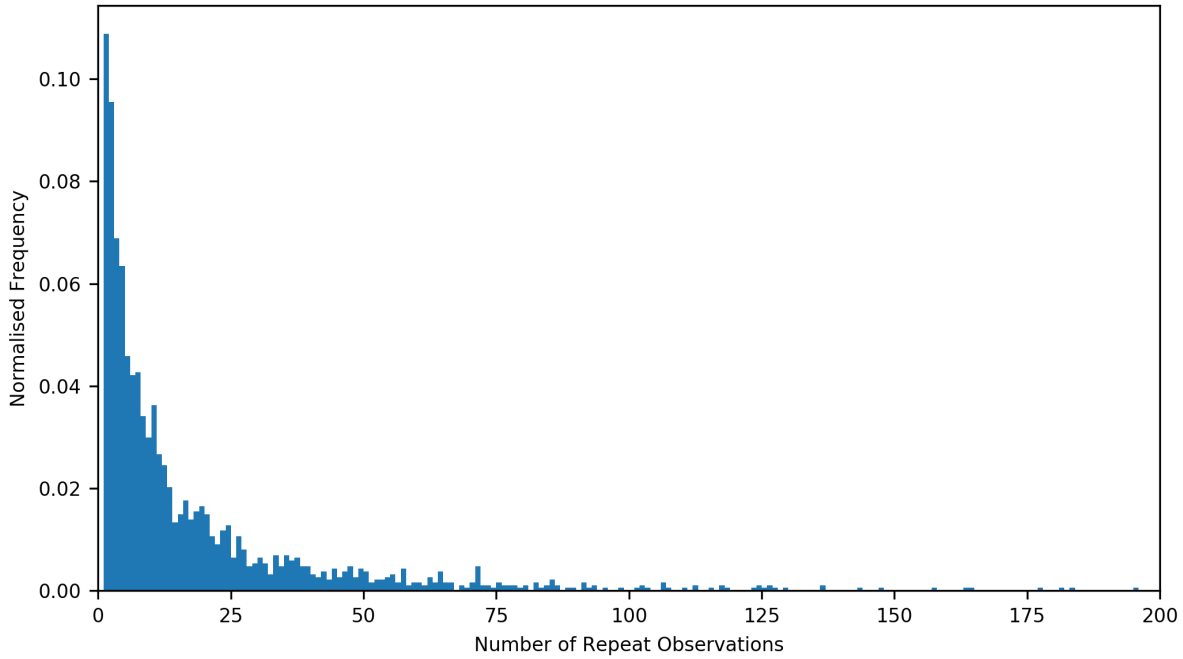


Figure 4.13: Histogram showing the number of repeat AID observations.

4.4.5 Location Analysis

92.4% of observations made were of devices which had a location fix. Of these observations, 67.9% were of devices with a location fix with an accuracy under 20m.

Figure 4.14 indicates that the location library frequently claims a location accuracy of 15m. It is unclear how this value is calculated since the library is closed source, but this accuracy is claimed so frequently that I believe it is a default: the accuracy values should not be trusted with too much confidence. Determining how accurate the library is indoors requires ground truth measurements, which, as discussed in the introduction, are difficult to obtain without significant set-up work.

4.5 Security Evaluation

In this section, I consider various attacks an adversary might use, and discuss mitigations implemented. I cover material that has been studied extensively by existing work; I then look at material more closely related to my implementation.

The security of E-AIDs and S-AIDs is discussed in depth in their respective papers [51, 39]: they are secure since the adversary must do a brute force search to decrypt an AID. This search yields the key for *one device*: all other devices are still protected if one key is compromised.

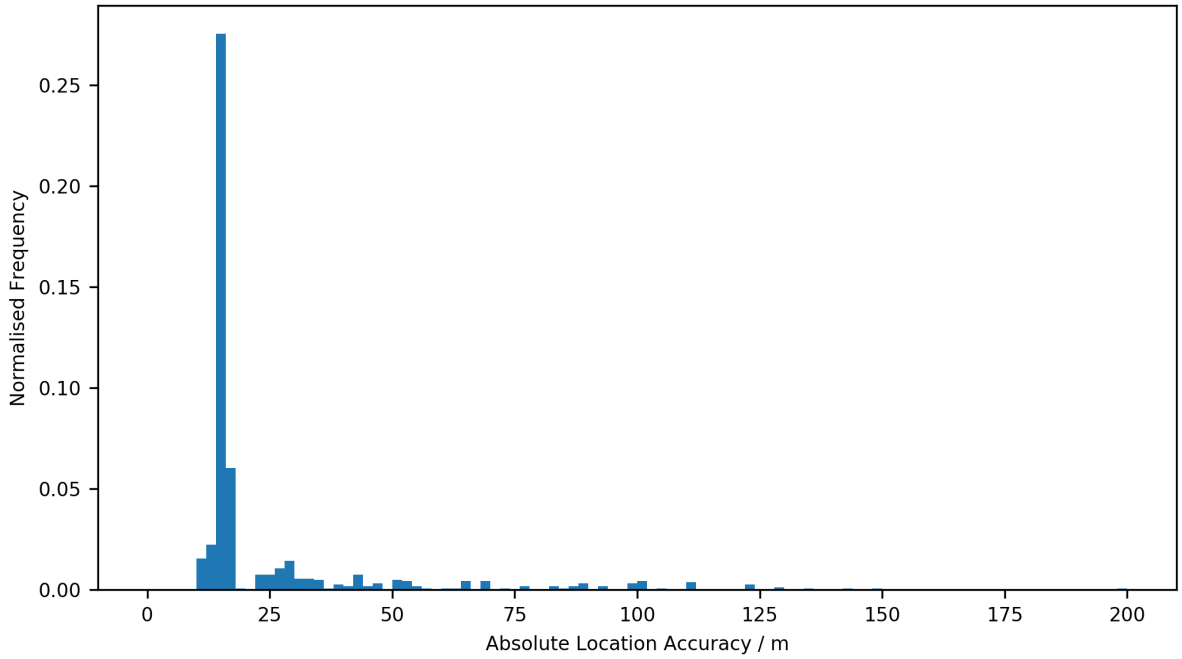


Figure 4.14: Histogram showing accuracy of absolute location data for devices which were later observed. Locations are included only once if the device is observed multiple times.

Another consideration is the linkability of AIDs generated with the same key; if they could be linked, then the block cipher (AES) is not an implementation of a pseudo-random permutation [39] as is assumed.

Denial of service attacks are viable: an attacker could jam frequencies used by Bluetooth, or inundate the de-anonymisation server with unresolvable AIDs. There is little that can be done to mitigate jamming, but the server could use the usual methods for handling denial of service attacks, such as dropping requests from certain IP addresses; the server could ignore clients sending unresolvable AIDs at an application level as well.

Low node density is an issue: consider the example of one smartphone in the entire building using the system. Regardless of any mitigations applied by the client, it is trivial to track the phone. Other technologies, such as WiFi tracking, could already do this, however [51].

Any production system would need to consider key distribution and revocation carefully. Google have provided details for their solution [39].

The following sections discuss material specific to my implementation, or not already discussed in depth by existing work.

4.5.1 Spoofing

If an adversary can impersonate other users, then the proximity data collected may be inaccurate.

E-AIDs

One issue not originally noted [50] is that E-AIDs never expire. A device's previous AIDs are only invalidated by re-negotiating a new (DID, DK) pair with the server.

I propose [51] making the nonce a sequence number, and incrementing the sequence number each time a new AID is needed; the server records the latest sequence number for each device, and ignores AIDs that resolve to earlier sequence numbers. This modification does not put a temporal limit on AID lifetime, and instead requires observations of a new AID to be reported before old AIDs are invalidated. It is, however, easy to implement for devices with inaccurate clocks, and does not significantly increase state maintained by the server.

Other Schemes

For S-AIDs and R-AIDs, if the server receives a stale AID the entry will not exist in the hashtable, and is ignored. With P-AIDs the server can check the timestamp.

4.5.2 Random Lifetimes and AID Expiry

With all schemes except E-AIDs, an AID could expire while the device is still stationary. It is inappropriate to generate a new AID and restart immediately, since an adversary could correlate the two. The client uses a random pause before restarting transmission when an AID expires. The pause length is chosen with a secure random number generator sampling from a uniform distribution between 0 and 60 seconds. The choice of pause length is a trade-off between the availability of the system, and the privacy of individuals.

An adversary could deduce information about a user by analysing the periods with which AIDs are changed. During the deployment it was common for a given device to use the same scheme for an extended period of time — e.g. if it had no network connection, but an accurate clock it would use S-AIDs. Since each scheme can have a different validity lifetime, an attacker could link AIDs together by noting that they all expired after the same amount of time. The solution is to randomly expire AIDs before they reach their maximum lifetime; this is done with a secure random number generator sampling from a uniform distribution.

4.5.3 Power Matching

Advertisement power and frequency may be characteristic to a particular device, facilitating linkage. These settings should be randomised each time a new AID is transmitted. As noted in Section 4.4.1, transmission is cheap, and the system operates

most effectively if devices transmit at the maximum power and frequency they can. The distributions sampled from should therefore be skewed to favour higher power and frequency advertisements.

Unfortunately, fine control over these parameters was unavailable in the targeted Android version.

4.5.4 Deniability

Google discuss an extension to the S-AID scheme [39]. At each timestep, the device key can be updated as:

$$\mathrm{DK}_{t+1} = \mathrm{SHA256}(\mathrm{DK}_t)$$

Doing this means that if Eve captures Alice's key at time T, any AIDs generated by Alice before T are *still* unlinkable by Eve, assuming SHA256 preimage resistance.

This idea can be extended to P-AIDs. If Eve has the server's private key, but *not Alice's device key*, then they cannot be *sure* that an AID with Alice's device identifier was generated by Alice, since they cannot check the MAC for forgery: this gives Alice some plausible *deniability*. Applying the above technique to device keys for P-AIDs means that if Eve obtains Alice's device key at time T, Alice can deny that they generated any AIDs Eve observed before T.

4.5.5 Attacks on P-AIDs

Deduction of the device identifier from the ciphertext implies that the encryption scheme does not offer confidentiality. For linkability, consider the following experiment for public key encryption scheme $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ and security parameter l:

Experiment $\mathrm{Link}_{\mathcal{A},\Pi}(l)$
Setup

1. The challenger generates a bit $b \in_R \{0,1\}$ and a key pair PK, SK $\leftarrow \mathrm{Gen}(1^l)$.
2. Adversary $\mathcal{A}$ is given input 1^l.
3. The challenger generates a device identifier (DID) and key (DK) and defines the following function:

   $$\mathrm{Plain}(T) = \mathrm{DID}||T||\mathrm{MAC}_{\mathrm{DK}}(\mathrm{DID}||T)$$

   This corresponds to the plaintext defined in Figure 3.12. Also define $m = \mathrm{len}(\mathrm{Plain}(\cdot))$.

Rules for the interaction

1. $\mathcal{A}$ is given PK and oracle access to $\mathrm{Dec}_{\mathrm{SK}}(\cdot)$ and $\mathrm{Plain}(\cdot)$.

2. $\mathcal{A}$ outputs two values:
   (a) $T_{\mathcal{A}}$
   (b) $M \in \{0,1\}^m$
3. The challenger computes C, and returns it to $\mathcal{A}$; C is calculated as:
   (a) $b = 0$: $C \leftarrow \text{Enc}_{\text{PK}}(\text{Plain}(T_{\mathcal{A}}))$
   (b) $b = 1$: $C \leftarrow \text{Enc}_{\text{PK}}(M)$
4. $\mathcal{A}$ continues to have oracle access as before, but may not ask for $\text{Dec}_{\text{SK}}(C)$.
5. Eventually, $\mathcal{A}$ outputs b'; if $b' = b$ then $\text{Link}_{\mathcal{A},\Pi}(l) = 1$, else 0.

Encryption scheme Π prevents AID linkability if for all probabilistic, polynomial-time adversaries $\mathcal{A}$ there exists a negligible function negl such that $P(\text{Link}_{\mathcal{A},\Pi}(l) = 1) \leq 1/2 + \text{negl}(l)$.

If Π is secure against adaptive chosen-ciphertext attack (CCA2), then it also prevents AID linkability. Proof of this requires reducing the interaction above to the usual interaction defined for indistinguishability under adaptive chosen-ciphertext attack (IND-CCA2) [42]; the definition above is a weaker requirement than CCA2 security, since $\mathcal{A}$ can choose *any* pair of messages in the IND-CCA2 interaction; in this case, one message has been instantiated for the adversary compared to the definition for IND-CCA2. Both RSA OAEP and ECIES are CCA2-secure encryption schemes [42].

We provide oracle access since an adversary could infer information about an AID from location information obtained from the server. Note that the above interaction protects a device whose device key has been leaked: if the server's private key has not leaked, $\mathcal{A}$ cannot infer the plaintext. Also note that an adversary cannot deduce the difference between AIDs from two separate devices which it knows the plaintext for.

The reader may be aware that 'authenticate-then-encrypt' is insecure; *it is not relevant in this context.* Authenticate-then-encrypt applies when combining a chosen-plaintext attack (CPA) secure encryption scheme with a MAC to produce an authenticated encryption scheme.

4.5.6 Location Forgery

Location measurements can be forged: devices cannot trust advertised measurements blindly.

Digital Signatures

A common pattern for verifying information is to have it certified by a mutually trusted third party (TTP). It is proposed that the de-anonymisation server can be the TTP.

Assuming a device has a WAN connection, the server could digitally sign the location trailer with the private key; the device broadcasts the returned signature with the location trailer. This scheme only works under the assumption that the server only makes signatures for trusted clients, however. With the Elliptic Curve Digital Signature Algorithm (ECDSA), a 64 byte signature can be produced with a 256 bit key; this is small enough to be transmitted with an AID and the location measurement.

Decentralised Verification

Schemes exist for establishing trust in ad-hoc networks, but are unsuitable since they require static identifiers [46]. Threshold cryptography, where the signing key is distributed over several nodes, could be thwarted by an adversary who controls many nodes.

4.6 Summary

I verified that my movement algorithm works for different users. The de-anonymisation server's performance has been assessed against the success criteria. Data obtained from a deployment was used to validate correct operation of the client. I discussed attack vectors an adversary could use to disrupt the system.

Chapter 5

Conclusion

I have built a system for obtaining proximity data using smartphones, but which prevents third parties from tracking or spoofing individuals easily. This required the implementation of an Android smartphone client, and a de-anonymisation server. As an example application, I emulated the functionality provided by traditional Bluetooth beacon deployments, which remain rare because of the setup cost; there are *many* other uses for the proximity data, however. I benchmarked different anonymisation schemes, performed battery drain tests, and did a deployment to verify that all the success criteria in my proposal were met. Possible attack vectors were also considered as part of the evaluation. Since the client's battery drain is acceptable, and the server is scalable, it is possible to deploy this system more widely.

I completed two of my planned extensions. Most notably, I proposed a new identifier format exploiting functionality introduced by Bluetooth 5.

Before submitting this work for publication, I would like to assess the battery impact of features introduced in Bluetooth 5. This is difficult as of May 2018 because adoption has been slower than expected.

There are worrying applications for the technology developed in this project. It could be applied in *mass surveillance*: assuming it is possible to install a client on a large proportion of phones, it would be *easy* to infer social relationships. This assumption is not unfounded: two companies have applications which have a market penetration of over 70% in the US [5].

5.1 Future Work

1. Location estimation using angle of arrival information. Future beaconing platforms could include this information for usage with positioning systems, allowing positioning to within 10s of centimetres [45].

2. Using mutual information in observations made between two devices. The path loss should be identical in each direction, but in reality multipath propagation and

shadowing cause differences. There is little research in this area.

3. Learning context from social relationships e.g. inferring that you are at the gym because the people nearby also go to the gym.

Bibliography

[1] IEEE Standard Specifications for Public-Key Cryptography – Amendment 1: Additional Techniques. *IEEE Std. 1363a-2004*, September 2004.

[2] *Bluetooth Specification v4.2*, December 2014.

[3] *Bluetooth Specification v5.0*, December 2016.

[4] *Ericsson Mobility Report*, 2017 (accessed April 25, 2018). `https://www.ericsson.com/en/mobility-report`.

[5] *The 2017 U.S. Mobile App Report*, 2018 (accessed April 26, 2018). `https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/The-2017-US-Mobile-App-Report`.

[6] *Bluetooth 5 ICs/Solutions*, 2018 (accessed April 26, 2018). `https://www.nordicsemi.com/eng/Products/Bluetooth-5`.

[7] *Android Studio*, 2018 (accessed April 4, 2018). `developer.android.com/studio/index.html`.

[8] *App Security Best Practices — Android Developers*, 2018 (accessed April 4, 2018). `https://developer.android.com/topic/security/best-practices.html`.

[9] *Java Universal Network/Graph Framework*, 2018 (accessed April 4, 2018). `jung.sourceforge.net/`.

[10] *JMH*, 2018 (accessed April 4, 2018). `openjdk.java.net/projects/code-tools/jmh/`.

[11] *Secure iBeacon advertising with Estimote Secure UUID*, 2018 (accessed April 4, 2018). `developer.estimote.com/ibeacon/secure-uuid/`.

[12] *Activity Recognition API — Google Developers*, 2018 (accessed January 12, 2018). `developers.google.com/location-context/activity-recognition/`.

[13] *Batching — Android Open Source Project*, 2018 (accessed January 12, 2018). `source.android.com/devices/sensors/batching`.

[14] *Company Identifiers*, 2018 (accessed January 12, 2018). `www.bluetooth.com/specifications/assigned-numbers/company-identifiers`.

[15] *Fused Location Provider API — Google Developers*, 2018 (accessed January 12, 2018). `developers.google.com/location-context/fused-location-provider/`.

[16] *HSQLDB*, 2018 (accessed January 12, 2018). `hsqldb.org/`.

[17] *Logback*, 2018 (accessed January 12, 2018). `logback.qos.ch/`.

[18] *Motion Sensors — Android Developers*, 2018 (accessed January 12, 2018). `developer.android.com/guide/components/services.html`.

[19] *Sensors — Android Open Source Project*, 2018 (accessed January 12, 2018). `source.android.com/devices/sensors/`.

[20] *Services — Android Developers*, 2018 (accessed January 12, 2018). `developer.android.com/guide/components/services.html`.

[21] *Intelligent Job Scheduling — Android Developers*, 2018 (accessed March 24, 2018). `developer.android.com/topic/performance/scheduling.html`.

[22] *Optimising for Doze and App Standby — Android Developers*, 2018 (accessed March 24, 2018). `developer.android.com/training/monitoring-device-state/doze-standby.html`.

[23] *Analyzing Power Use with Battery Historian — Android Developers*, 2018 (accessed March 26, 2018). `developer.android.com/topic/performance/power/battery-historian.html`.

[24] *Measuring Device Power — Android Open Source Project*, 2018 (accessed March 26, 2018). `source.android.com/devices/tech/power/device`.

[25] *IntelliJ IDEA*, 2018 (accessed May 12, 2018). `https://www.jetbrains.com/idea/`.

[26] *Active RFID Tags*, 2018 (accessed May 6, 2018). `https://www.zonesafe.net/features-benefits/active-rfid-tags/`.

[27] *Beacons – Google Developers*, 2018 (accessed May 6, 2018). `https://developers.google.com/beacons/`.

[28] *Bouncy Castle Cryptography APIs*, 2018 (accessed May 6, 2018). `https://www.bouncycastle.org/java.html`.

[29] *iBeacon – Apple Developer*, 2018 (accessed May 6, 2018). `https://developer.apple.com/ibeacon/`.

[30] *Indoor Positioning – Philips Lighting*, 2018 (accessed May 6, 2018). `http://www.lighting.philips.com/main/systems/lighting-systems/indoor-positioning`.

[31] Nadav Aharony, Wei Pan, Cory Ip, Inas Khayal, and Alex Pentland. Social fmri: Investigating and shaping social mechanisms in the real world. *Pervasive Mob. Comput.*, 7(6):643–659, December 2011.

[32] Elaine Barker and Allen Roginsky. Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. NIST Special Publication, NIST, November 2015.

[33] L. Doherty, K. S. J. pister, and L. El Ghaoui. Convex position estimation in wireless sensor networks. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 3, pages 1655–1663 vol.3, 2001.

[34] R. Faragher and R. Harle. Location fingerprinting with bluetooth low energy beacons. *IEEE Journal on Selected Areas in Communications*, 33(11):2418–2428, Nov 2015.

[35] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, IETF, June 1999.

[36] Google. Eddystone Ephemeral Identifier. `https://developers.google.com/beacons/eddystone-eid`.

[37] A. Harter and A. Hopper. A distributed location system for the active office. *IEEE Network*, 8(1):62–70, Jan 1994.

[38] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The anatomy of a context-aware application. *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 59–68, 1999.

[39] Avinatan Hassidim, Yossi Matias, Moti Yung, and Alon Ziv. Ephemeral identifiers: Mitigating tracking & spoofing threats to ble beacons. 2016.

[40] Ari Juels and Ravikanth Pappu. Squealing euros: Privacy protection in rfid-enabled banknotes. In Rebecca N. Wright, editor, *Financial Cryptography*, pages 103–121, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[41] B. Kaliski and J. Staddon. PKCS #1: RSA Cryptography Specifications. RFC 2437, IETF, October 1998.

[42] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[43] A. Montanari, S. Nawaz, C. Mascolo, and K. Sailer. A study of bluetooth low energy performance for human proximity detection in the workplace. *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 90–99, March 2017.

[44] Suman Nath. Ace: Exploiting correlation for energy-efficient and continuous context sensing. ACM, June 2012.

[45] R. P and M. L. Sichitiu. Angle of arrival localization for wireless sensor networks. In *2006 3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, volume 1, pages 374–382, Sept 2006.

[46] Asad Amir Pirzada and Chris McDonald. Establishing trust in pure ad-hoc networks. In *Proceedings of the 27th Australasian Conference on Computer Science - Volume 26*, ACSC '04, pages 47–54, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[47] Theodore Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2001.

[48] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[49] Moustafa Youssef and Ashok Agrawala. The horus location determination system. *Wirel. Netw.*, 14(3):357–374, June 2008.

[50] Augustin Zidek. Bellrock – your phone as an anonymous BLE beacon. `http://augustin.zidek.eu/wp/wp-content/uploads/2015/02/Master_thesis.pdf`, June 2016.

[51] Augustin Zidek, Shyam Tailor, and Robert Harle. Bellrock: Anonymous Proximity Beacons from Personal Devices. In *IEEE International Conference on Pervasive Computing and Communications*, March 2018.

Appendix A

Consent Form

Anonymous Proximity Beacons from Smartphones (Revised)
Participant Information

Before you decide to take part in this study it is important for you to understand why the research is being done and what it will involve. Please take time to read the following information carefully and discuss it with others if you wish. If you have any questions, you should contact the individual running the experiment.

Background and Purpose

This experiment will involve collecting data by installing an Android based application onto your smartphone. The purpose of the experiment is to evaluate the effectiveness of a system that allows you to determine your location indoors, without requiring the installation of devices such as Bluetooth proximity beacons into the building. This system works by transmitting identifiers over Bluetooth, which cannot be linked to you by other devices; other devices can scan for these identifiers, but must send them to a location server, which is able to make sense of them. There may also be static beacons installed in buildings; these behave similarly to the software installed on the smartphones, and are present to provide additional data for analysis.

With the participant's consent, short periods of raw accelerometer data will be collected, and associated with their age and gender. The accelerometer data is used to validate that the application detects periods of motion correctly. The acceleration observed by your smartphone will vary according to your gender and age, which is why this data is being collected.

Only accelerometer data is associated with your gender and age: all other data collected from the experiment is referred to with an identifier only.

Instructions for Participants

Taking part in this study is entirely voluntary; refusal or withdrawal from the experiment will not result in any penalty or loss to you, now or in the future.

The study involves installing an Android application onto your device. The application should be left running in the background, and you should proceed to use your smartphone as normal.

The application will log information about: accelerometer data from the device, the identifiers that are generated and transmitted by the device, any identifiers that the device can see (with signal strength information), and absolute location data (obtained via GPS). Interactions with the application server are also recorded.

If you wish to provide raw accelerometer data, you can press a button in the user interface, which opens a dialog asking you to tag the activity you are about to do. The data is then

collected for 2 minutes, and sent to the server. Once the collection has started, you may proceed to use your smartphone as normal. After two minutes have elapsed, no more data is collected unless you explicitly tell the application to start another recording.

Risks and Benefits

The installation of the application may result in degraded battery life; this depends on the device, but in testing the additional battery drain (on a Samsung Galaxy S7) has been observed to be around 0.7% per hour.

It is possible to infer from the data whether you are co-located with another participant in the study.

Accurate absolute location data (recorded via GPS) will be stored in the data. It is possible in some cases to de-anonymise participants given sufficient location data. The intended use of this data is to determine when participants are co-located with each other, and what their relative locations are. If the data is released, all location data will be processed to remove absolute references.

There is no intended benefit for the participant of the experiment. Participants will not receive any reward for participation.

The data will be anonymised and not linked to a name, such that if it is leaked, it will not be possible to associate the data with an individual. The results will be analysed for a dissertation, which will be submitted in May 2018.

All data ***except raw accelerometer data*** will be identified only by an identifier, with personal details kept in a locked file or secure computer with access only by the immediate research team.

Raw accelerometer data is associated with your age and gender, but is not linked to any other data in the experiment.

Ethical Review and Contact

This experiment has received ethical approval from the Computer Laboratory Ethics Committee.

Primary contact: *Shyam Tailor, email:* *sat62@cam.ac.uk*

UNIVERSITY OF CAMBRIDGE

Anonymous Proximity Beacons from Smartphones (Revised)

Consent Form

- I have read and understood the Participant Information Sheet;
- I have been given the opportunity to ask questions and have had them answered to my satisfaction;
- I agree to take part in this study;
- I understand that my participation is voluntary and that I am free to withdraw at any time without giving a reason;
- I consent to the procedures used by the investigators to ensure that the data collected from me is anonymous;
- I consent to the my data being retained and being used in future studies.
- I do / do not (*delete as appropriate*) consent to having raw accelerometer data collected from me, and associated with my age and gender.

Participant Name:

Participant Signature:

Date:

Researcher Name:

Researcher Signature:

Date:

Appendix B

Project Proposal

Computer Science Tripos Part II Project Proposal

Anonymous Proximity Beacons from Smartphones

S. A. Tailor, Downing College

October 19, 2017

Project Originator: Dr R. Harle

Project Supervisor: Dr R. Harle

Director of Studies: Dr R. Harle

Project Overseers: Prof J. Crowcroft & Dr T. Sauerwald

Introduction and Description of the Work

The invention of Bluetooth Low Energy (BLE) has made it viable to manufacture beacons that provide location information to nearby devices that also have BLE; unfortunately, installations of beacons are still rare. This project aims to solve this problem by re-purposing stationary personal devices as the beacons, using BLE peripheral mode, while also addressing the privacy concerns associated with such a set-up. The end product would consist of a client application which runs on a user's (Android) smartphone, and a de-anonymising server which users can request location information from.

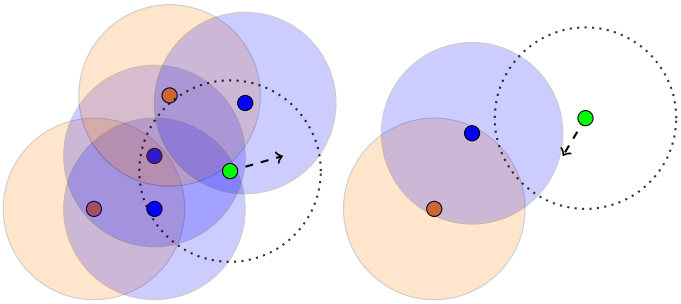


Figure 1: Example scenario. The green nodes represent a moving device, which is scanning for the beacons, but is not itself a beacon. Blue nodes are beacons which have not got access to their absolute position; orange nodes are beacons which do have this information. The larger circles represent the range of each device.

The client application would produce an anonymous identifier, which is transmitted by the device, and received by other devices when they scan for beacons. These devices then report that they observed the anonymous identifier to the server, which can then reverse the anonymous identifier back to the corresponding device identifier. Devices only beacon when they are not moving; otherwise they may still scan for nearby beacons, but they will not be advertising themselves. There are many methods to generate the anonymous

Figure 2: The corresponding connectivity graph for the scenario above. In this case, there are three disconnected sub-graphs.

identifiers: one method would be to use symmetric encryption, with keys shared between each device and the server.

The language of choice for this project will be Java: it is Android's default language for application development, and is also suitable for for server side development since there is good support for cryptography and networking in the standard library. Sharing a language between client and server development also simplifies serialisation, and allows the re-use of common modules.

Starting Point

The core of this project is based off of a Part III project by Augustin Zidek [1], which has subsequently been turned into a paper which is currently in review. It should be noted that not all schemes described by the paper have been implemented, and there is new Bluetooth technology that has been introduced since that project was completed.

The relevant Tripos courses are primarily: *Mobile and Sensor Systems* and *Security I/II*. Small amounts of code have been written to experiment with libraries: namely for JSON serialisation, movement detection, and how to enable BLE peripheral mode. Most of it is demo code straight from the corresponding library's website, with small modifications.

Resources Required

For this project, I will be using my own computer, a 2015 MacBook Pro with a 2.7GHz Intel Core i5 processor and 8GB of RAM. I accept full responsibility for this machine, and I have made contingency plans to deal with hardware and/or software failures. If this machine should fail, I can continue work on an MCS machine. Code will be backed up to an external hard drive, and to a cloud service; it will also be pushed regularly to an online GitHub repository.

The main piece of software for Android development is Android Studio, which is freely available from Google's website. Similarly, for server side development, I will need a JDK, which is widely available online, and also installed on the MCS machines already.

Android based smartphones with BLE will be needed to deploy the client on, and to do testing with. I currently own one such device, and have access to three more at home. If required, it should be possible to loan further devices from the department (confirmed by my supervisor). One possible extension will require the use of a smartphone with Bluetooth 5 and Android with version 8.0 or above to evaluate fully; my supervisor has confirmed that the lab should be able to loan me appropriate devices to do the testing.

Work to be Done

The main components of the project are the following:

1. Implementation of an Android client that generates anonymous identifiers which are transmitted by the smartphone.

   There are several methods that can be used to generate these identifiers, and all should be implemented so that they can be evaluated against each other. The smartphone should also attempt to provide GPS data if it can, to aid with absolute positioning for other devices which are unable to get a GPS signal.

   Methods to create anonymous identifiers include:

   (a) Unique symmetric key shared between each client and the server. The device identifier can then be encrypted with a nonce to create an anonymous identifier.

   This is the default algorithm; this method has been employed in industry. [2]

   (b) Device can ask the server to provide a list of anonymous identifiers, and choose one of them to transmit.

   (c) Hashing the device identifier with the time stamp (in some coarse interval) to create the anonymous identifier.

2. Implementation of a de-anonymising server that can receive scan reports from devices, and can convert the anonymous identifiers back into the corresponding device identifier.

   The server should also be able to reply to requests from smartphones their location: proximity information is the core aim, but in some cases it may also be possible to provide absolute information. It is also desirable to implement some kind of access control, so that users can specify if other devices are able to track their location ("friend" devices).

3. Benchmarking and evaluation of the system.

   This could involve the following:

   (a) Comparison of the different schemes to create anonymous identifiers, and the asymptotic performance of the de-anonymising server when there is a large number of users.

   (b) What heuristics are possible for de-anonymisation, and their effect on overall server performance.

   (c) The impact on the smartphone's battery life.

   (d) Using the system to collect data.

   (e) Simulations using real world data sets, such as from the Bat indoor positioning system [3] and the SpaceLab BLE data set. [4]

Success Criteria

The project will be deemed a success if it is possible for a user to acquire location information, without being easily tracked. More concretely:

1. The server must be able to scale to 1000 users in the same building and still be able to reverse an anonymous identifier within 1 second of receiving it. This time limit should be achievable even in the case it is not possible to apply any heuristics.
2. There must be some form of access control for location: a user must be able to specify "friend" devices, who can obtain location information about them.
3. Users must be able to request their location, and the system should return proximity based information, and absolute location data if it is available. This must be done without leaking the identity of other devices, except of those who have given permission for the requesting user to see their location.
4. Devices must be able to change their identity without having to contact the server.
5. The de-anonymisation process should be bounded temporarily, so that it is possible to evaluate how far the system will scale.

Possible Extensions

1. Expanding the location component of the project to increase the accuracy of location queries. This could take the form of building a full graph, and attempting to estimate absolute position from several absolute measurements which are connected to the requesting device.

   There is also the possibility of using Received Signal Strength Indicator (RSSI) values obtained during scans to place weak constraints on the distance between nodes in the location graph. While it is not possible to make accurate inferences from RSSI values, since it does not follow a strict $1/d^2$ fall-off as is theoretically expected, it should be possible to roughly class distance: if the RSSI is over a certain threshold, then the distance between the two devices is almost certainly small.
2. Exploring the use of public key cryptography, which has only recently become possible with the introduction of larger advertising packet sizes in Bluetooth 5. It is not quite possible to fit a payload encrypted using RSA-2048, but it should be possible to use elliptic curve cryptography (which has significantly smaller key sizes, but still wouldn't fit into pre-version 5 advertising packets).
3. More advanced modelling of existing data sets (such as considering Bluetooth 5 range improvement and the system's effectiveness as a function of range).

Timetable and Milestones

The schedule is broken into 15 two week long periods, with the first period beginning on 20/10/17.

1. Further research on the libraries as needed, and any initial basic set-up, such as web server configuration. Begin development of the Android client.
2. By the end of the period, the Android client should be mostly finalised, except for anonymity features. The implementation should allow swapping in the code for anonymous identifier generation when it has been developed.
3. Begin work on the server. It should be able to register new devices, receive scan reports from devices at this stage, and handle requests to change a user's "friend" devices.

   Also begin work on **one** method to create anonymous identifiers.
4. Finish work on anonymous identifier generation and decryption. It should be possible after this period to have a device be able to generate an anonymous identifier, and for the server to be able to correspondingly reverse it back to a device identifier.
5. Work on building the location component; devices should be able to obtain very basic location data, such as the number of devices nearby, and the number of hops from the nearest absolute location reported, with the coordinates of the report.
6. Begin testing. Aim to ensure that the system works as intended with the first anonymous identifier method implemented, with a small scale deployment.
7. Write code for other methods of anonymous identifier generation. Write benchmarks to test the asymptotic performance of the different anonymous identifier generation methods.
8. Write progress report. Devise some heuristics to improve de-anonymisation performance, and implement them. Pre-process data sets into a suitable format.
9. Create test framework for use with the data sets from previous indoor positioning studies. Collect data for real world studies, and explore impact of the heuristics.
10. Continue work on real world data sets and collect battery life data from Android client. If running on schedule, start looking at implementing the extensions.

    Extension 1 (improved location data) is estimated to take about 3–4 weeks of work, and will involve creating a graph of observations. Adding the collection of RSSI data should be trivial, although making useful inferences from it will be more challenging.

    Extension 2 (public key encryption) is estimated to take around 1–2 weeks of work. It should be fairly easy to add it as another scheme for anonymous identifier generation; the main problem will be to evaluate it against the already implemented schemes, under realistic conditions.

    Extension 3 (more advanced modelling of data sets) is open ended, and there are several ways to approach it; this extension would be attempted last.
11. Continue work on extensions.

12. Continue work on extensions. Begin a skeleton draft of the dissertation.
13. Begin writing the dissertation.
14. Continue writing dissertation, and aim to finish draft.
15. Proofreading and submission.

References

[1] A. Zidek, "Bellrock – your phone as an anonymous BLE beacon." `http://augustin.zidek.eu/wp/wp-content/uploads/2015/02/Master_thesis.pdf`.

[2] Google, "Eddystone Ephemeral Identifier." `https://developers.google.com/beacons/eddystone-eid`.

[3] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, "The anatomy of a context-aware application," *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 59–68, 1999.

[4] A. Montanari, S. Nawaz, C. Mascolo, and K. Sailer, "A study of bluetooth low energy performance for human proximity detection in the workplace," *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 90–99, March 2017.