Implementation of a Drone Remote Identification System

Bruce Williams
972648

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Bachelor of Science

Department of Computer Science
Swansea University

May 2, 2022

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed [illegible] (candidate)

Date 02/5/2022

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed [illegible] (candidate)

Date 02/5/2022

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed [illegible] (candidate)

Date 02/5/2022

No bird soars too high, if he soars with his own wings. -William Blake

Abstract

With the number of drones increasing year on year, the number of incidents of drone misuse has been increasing too. In December 2018, hundreds of flights were cancelled due to sightings of a drone flying over the runway, the culprit was never found. This might have been different if there had been a Drone Remote Identification (Drone RID) System in place at the time. Drone RID is a similar concept to car number plates. It allows the general public and aviation officials to see the registered identifier of drones flying nearby. Aviation officials are able to look up this identifier to discover the identity of the pilot, as well as their address and contact information.

Using a Raspberry Pi, I have built an implementation of Drone RID. I have developed an accompanying Android app that observers can use to view Drone RID information and report drone misuse. I have also created a registry prototype that stores information about drones, drone operators, and reports of drone misuse. It mimics the registry of an aviation body. I then carried out an experiment to determine the effective range of my implementation, which is arguably the most critical feature of a Drone RID system. My implementation has an effective range of 50m.

The only current implementation of such a system is a system called NET-RID created by the Swiss Government. It is currently a voluntary opt-in system to it's not yet possible to determine the extent to which such a system can prevent drone misuse. Drone RID compliance will become mandatory in the UAS in September of 2023, at which point the world will see just how effective Drone RID is at mitigating the problem of drone misuse.

Acknowledgements

Thank you to my housemates Tom and Ed, for riding the towering waves of final year with me. Thank you to my supervisor, Pardeep, for your infectious enthusiasm and all your help throughout this year.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivations

In December of 2018, tens of thousands of passengers at Gatwick Airport were disrupted by a number of reported drone sightings [2]. For many in the UK, this was the first national story of drones impacting major infrastructure, and this contributed to concern about the growing problem of drone misuse. German company Drone Industry Insights forecasts drone sales to grow from 828,000 in 2021 to almost 1.4 million in 2026 [3]. With these increasing numbers in mind, it is reasonable to expect that as drone use grows, *drone misuse* will grow with it.
The core motivation of this project lies in the limitations of the already existing solutions to drone misuse. Figure 1.1 shows a number of these existing solutions: ranging from low-tech solutions, such as eagles trained to catch drones or drones equipped with nets to catch other drones, all the way to advanced techniques such as GPS spoofing and *de-authentication attacks* in order to disrupt or hijack the offending drone. The common theme of all of these solutions, bar de-authentication attacks, is that they all rely on getting a drone on the ground, retrieving it, and using the serial number on the drone to attempt to track down the pilot. Whilst downing a drone is certainly effective at putting an immediate stop to drone misuse incidents, if authorities are unable to locate the original pilot from the recovered drone, there is limited action that can be taken to prevent further misuse.

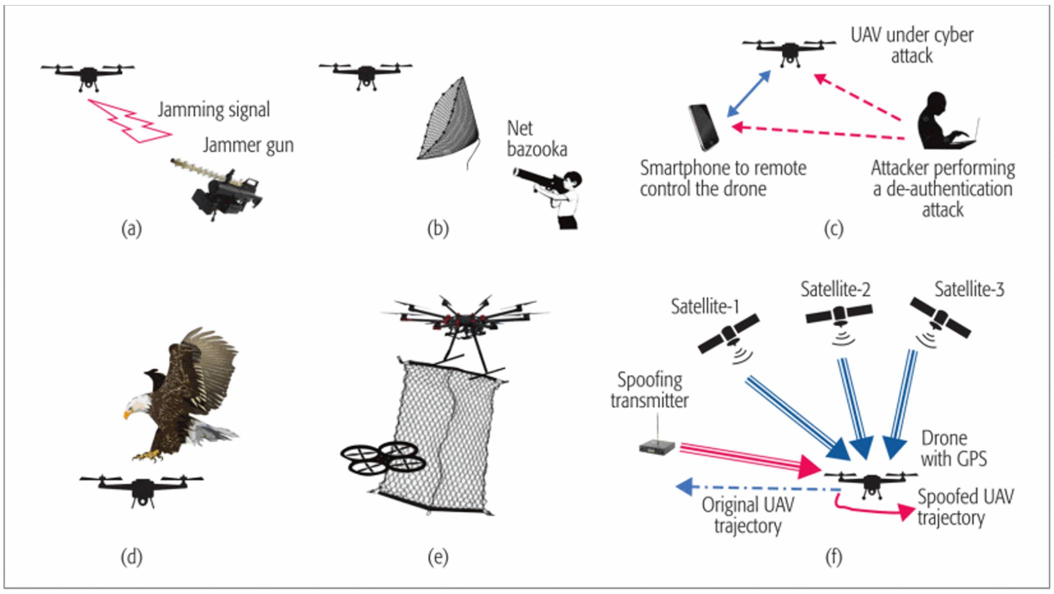


Figure 1.1: Image showing a number of current solutions to drone misuse, from IEEE Communications Magazine [1]

1.2 Objective

In this dissertation I present a project that aims to mitigate the problem of drone misuse by developing a *Drone Remote Identification* System to make malicious drone pilots identifiable and therefore accountable for their actions. I will consider the privacy and security implications of such an architecture as well as discuss the effectiveness of this technology in the real world.

1.3 Overview

The remainder of chapter 1 outlines the document structure and the key contributions of this work is organised as follows. Chapter 2 examines the problem of drone misuse in depth and introduces the concept of Drone Remote Identification, including privacy and security considerations. In chapter 3 I discuss the project management aspects of the project. In chapter 4 I discuss the specifics of my implementation as well as demonstrate what I have achieved throughout the project. In chapter 5 I discuss my field testing methodology and results. Finally, in chapter 6 I summarise the main contributions and key points to take away from this dissertation as well as discuss future work.

1.4 Contributions

The main contributions of this work can be seen as follows:

• **Android app for drone observers**

I have developed an Android app with Kotlin which allows the general public to view nearby drones, report drone misuse for specific drones, as well as allowing aviation authorities to view the identity of pilots of drones flying nearby.

• **A prototype drone registry**

I have set up a prototype database which holds information about drones, their owners, and drone misuse reports. This prototype demonstrates the system that aviation authorities could use to support Drone Remote ID technology in the real world.

• **A simple drone simulated using Raspberry Pi**

I built a simulated drone using Bluetooth Low Energy Advertising on a Raspberry Pi to field test the effective range of my implementation.

Chapter 2

Drone Misuse and Drone Remote Identification

In this chapter I will explore the problem of drone misuse in more depth, as well as evaluate existing solutions before introducing the concept of Drone Remote Identification (Drone RID).

2.1 Drone Misuse

I previously introduced the problem of drone misuse in the context of the disruption at Gatwick Airport in December 2018. However, the problem of drone misuse is by no means limited to neither Gatwick Airport nor the aviation sector. A Hungarian Government report [4] discusses the risks posed to prisons by drone misuse:

> "In November, 2016, banned items were taken by drone to an inmate in a cell on the 4th floor of the Danish Nyborg prison to assist his prison-break (White, 2016) [5]. The small UAV carried two mobile phones, a saw blade and nails, and successfully put these items down in the institution then disappeared. The payload was confiscated by the prison guards, but the drone did not get captured and its sender was not found."

Evidently the freedom to exploit airspace is dangerous when in the wrong hands. Before drones were readily available, criminals wanting to exploit airspace only had the option of using piloted aircraft which are much easier to detect and track. Commercial drones, such as the one used at Nyborg prison are "low observable" due to their small radar cross sections and high manoeuvrability [6], this means that they are difficult to track by conventional means, such as the ones detailed in section 2.2. It is likely that there will be an increase in incidents of drone misuse as drones become more affordable and easily available to the general public, therefore it is important that there are robust countermeasures in place to both neutralise these incidents, as well as prevent further incidents taking place.

2.2 Existing Solutions

The majority of current solutions to drone misuse are reactive in nature. They focus on taking a drone out of the sky in order to neutralise the threat. There are 2 stages to this approach: detection and mitigation. An Athens University survey [7] examined a number of detection and mitigation techniques:

Detection:

1. Radar Detection: Radar (with the assistance of machine learning tools) can be used to distinguish between birds and drones at long ranges (up to several hundred kilometres). However, airport radar sensors cannot detect small Unmanned Aerial Vehicles (such as drones). Radar is also the most expensive of all drone detection systems and requires various authorisations to run.
2. Radio-Frequency (RF) Detection: Whilst a study by Nguyen et al. [8] was able to detect commercial UAVs from a distance of 600m, the effectiveness of RF detection is limited in areas with lots of RF traffic, such as WiFi signals and other emitting smart devices. Unfortunately this is the case in many urban environments as well as airports and prisons.

Mitigation:

1. Electronic Interdiction: This is the use of RF signal jamming or deliberate interference to disrupt the operation of the drone by the original pilot. Ordinarily, this will result in the drone either returning to its home location, or making a controlled landing at its current position. However, Mitchel, A. [9] was able to dynamically alter GPS coordinates in real-time, which allowed them to control the drone's position and direct it to another landing zone.
2. Kinetic Interdiction: I have previously discussed measures such as counter-drone nets and birds of prey, these fall into this category. However counter-drone nets are only

effective for drones travelling at low speed and there are the use cases of birds of prey are limited due to the risks they pose to conventional aircraft at airports. Kinetic techniques are unlikely to be viable in crowded areas due to the risk of the drone's uncontrolled crashing or of the triggering of CBRNE (Chemical, Biological, Radiological, Nuclear, and Explosive) payloads.

2.3 Drone Remote Identification (Drone RID)

Of course, no solution is perfect or without drawbacks, but none of the solutions reviewed by the Athens paper have identification capability on their own. Identification of drones is the key to holding malicious drone users to account and, in turn, prevent further drone misuse. Drone RID is the broad term for any system that equips drone observers with identification capability.

2.3.1 Drone RID as a concept

Cars and other road vehicles use a standardised number plate system to identify vehicle owners. Whilst physical number plates are an effective measure for ground-based vehicles, the high speed and manoeuvrability of drones leads me to believe that a physically visible identifier would be ineffective. A digital solution must be considered, and this is where Drone RID comes in. The United States Federal Aviation Administration defines Drone RID as: "the ability of a drone in flight to provide identification and location information that can be received by other parties." [10] The FAA rule requiring most drones in US airspace to have remote ID capability went into effect on the 21st of April 2021. However, drone manufacturers have until the 16th of September 2022 to make all new drones compliant, and drone pilots have until that date the following year to be compliant. Compliance for pilots takes the form of either installing a drone software update with Remote ID built in, or by retrofitting an external Remote ID module to their drone [11].

There are 2 categories of Remote ID - Network and Broadcast, which Card et al. [6] define as follows: "Network RID defines a set of information for UAS to make available globally indirectly via the Internet, through servers that can be queried by Observers. Broadcast RID defines a set of messages for UA to transmit locally directly one-way over Bluetooth or Wi-Fi (without IP or any other protocols between the data link and application layers), to be received in real time by local Observers."

2.3.2 Architecture of a Drone RID system

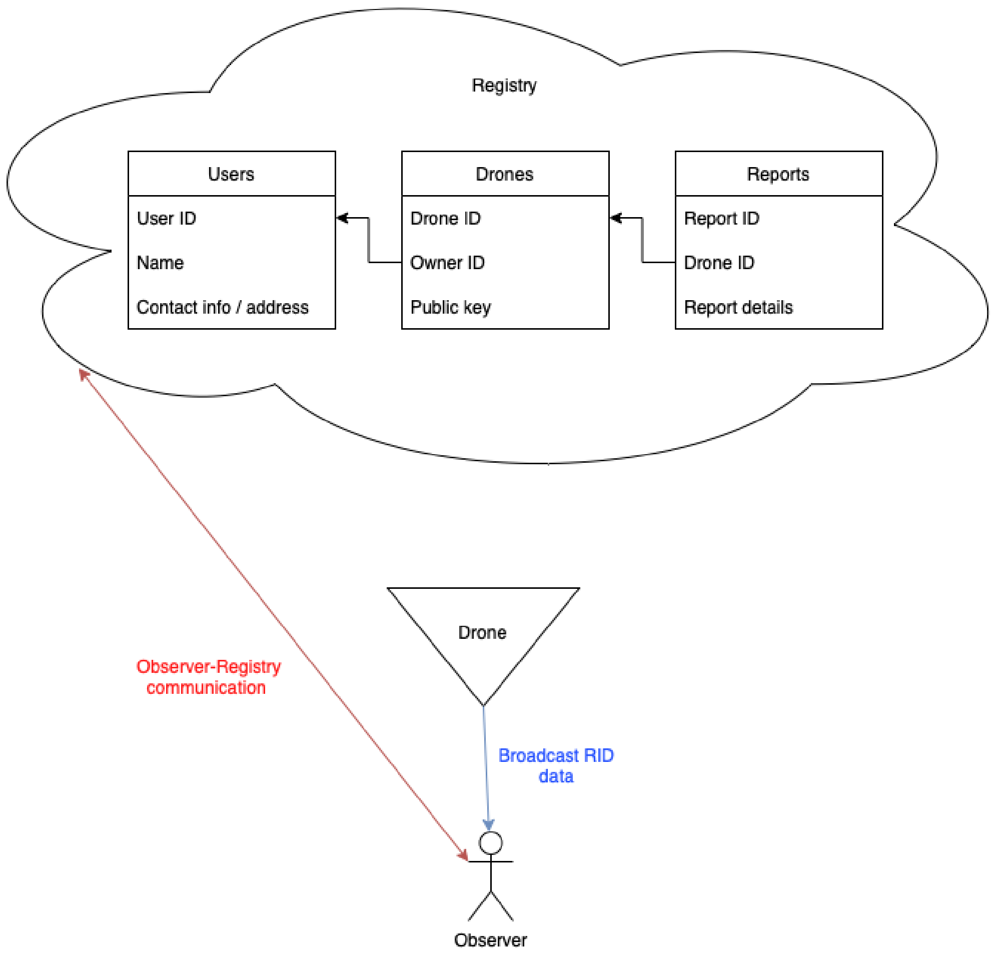


Figure 2.1: Drone broadcast RID architecture

Figure 2.1 shows a high level depiction of a Drone RID system that uses Broadcast RID. There is one way communication of RID data from drone to observer. The observer's device can then query the RID data with the registry contents over the internet to determine information about the owner. There is no direct internet connection between drone and registry

2.3.3 Privacy & Security Considerations of Drone RID

As with any new technology, there are a series of considerations to be made regarding the privacy and security of both the users of the technology, as well as wider society.

2.3.3.1 Privacy of drone users

So far, this dissertation has primarily focused on those using drones with malicious intent. However it is important to note that the majority of drone users do so with perfectly innocent intentions, and it would be unfair to treat all drone users as suspects of drone misuse. Therefore it is important to balance the privacy of drone users with the security of the general public. This balance can be maintained by ensuring personal information is only visible to authorised officials.

2.3.3.2 Security of Drone RID systems

The most critical assumption made in any Drone RID system is that the pilot of the drone identified is either the person that has registered that drone's identifier or is being directly supervised by that person. I will now analyse how sound this assumption is and, if not, how risk can be mitigated. I have identified two cases where this assumption is incorrect. The first is that the person operating the drone that has been identified by the system is neither the registered owner, nor being directly supervised by the registered owner. This can be due to either theft of the drone or hijacking of the drone. I believe the risk of misidentification due to theft can be mitigated by a theft reporting tool being built into the drone registry system, similar to how a stolen car that's caught speeding will not result in the true owner being punished. Unfortunately the risk of drone hijack cannot be as easily written off:
A survey by Zhi et al. [12] demonstrates how drones can be hijacked through the use of a de-authentication attack.

> "However, WiFi brings security risks as well. We study and find that Spark and Mavic which are two products of DJI support WiFi connecting. Although one UAV can only be connected with one terminal, we can still break the connection between UAV and terminal easily and even crack the password to control it."

Essentially, the attack works by forcing the pilot to disconnect from the drone, and then attempting to intercept the handshake when the pilot attempts to reconnect. DJI took steps to prevent this kind of attack by introducing WPA2 to encrypt the connection. However Zhi et al. propose that WPA2 is still vulnerable to dictionary attacks, particularly if the pilot uses a weak or easily-guessed password. The risk of these attacks can be significantly reduced by forcing pilots to use secure passwords to connect to their drones.

Another vulnerability in a Drone RID system is in the communications from drone to observer. The system I have described so far trusts that the communications it receives from drones is genuine and authentic, but this may not always be the case. A common problem in the world of car number plates is the use of false number plates to mask or spoof the identity of the vehicle owner or driver. There is, of course, a drone equivalent of this; it is not infeasible that somebody would reverse engineer the broadcast RID system to identify the specifics of how the RID is transmitted, which could enable them to spoof their drone identifier. The risk of identity spoofing can be mitigated through the use of cryptographic authentication.
One way this can be built into a Drone RID system by having drone users generate a public/private key pair when they register their drone, and they will add their private key to their drone when setting up Drone RID. The RID signals sent from the drone will be digitally signed using the private key of the user, this means that a copy of the RID data is taken and encrypted using the private key, before being appended it to our original data. The public key for that user stored in the central registry can then be used to verify the authenticity of the RID data. However it is important to note that adding this authentication will significantly increase the size of the RID data. This part of the system is also vulnerable to replay attacks, which in this case is when somebody copies the RID data sent out by a drone, and then broadcasts it from their own device (i.e. replay the data). This could be exploited to frame drone users and make it seem as if users have been flying in restricted areas. The extent of this vulnerability can be reduced by adding timestamps to RID data, which will then allow observers to determine if the data is being sent a long time after it has been originally transmitted.

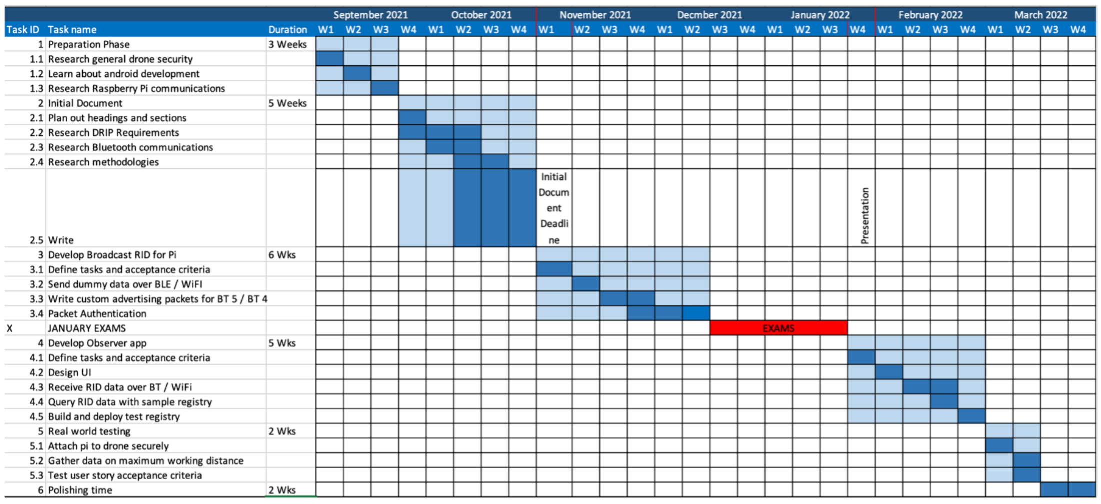
2.3.3.3 Current implementations of Drone RID

At the time of writing, the only national Drone RID system in action is the Swiss *NET-RID* system. In a 2019 report [13], the Swiss U-Space Program of the Swiss Federal Office of Civil Aviation describes the details of their system. As the name suggests, NET-RID is a Network RID system, meaning that the drones transmit identification data directly to the internet. NET-RID is an opt-in system; drone users can volunteer to be a part of the system but there is no enforcement as of yet, meaning that there is no data on the effect of this syste, on Drone misuse. I expect this to change when the European Union U-space regulations come into effect in January 2023 when remote ID will be mandatory for the majority of drones in European Union Air Safety Agency airspace [14].

Chapter 3

Project Management

3.1 Reflections on Timeline



			September 2021				October 2021				November 2021				Decmber 2021				January 2022				February 2022				March 2022			
Task ID	Task name	Duration	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
1	Preparation Phase	3 Weeks	▒	▒	▒																									
1.1	Research general drone security		█	▒	▒																									
1.2	Learn about android development		▒	█	▒																									
1.3	Research Raspberry Pi communications		▒	▒	█																									
2	Initial Document	5 Weeks				▒	▒	▒	▒	▒																				
2.1	Plan out headings and sections					█	▒	▒	▒	▒																				
2.2	Research DRIP Requirements					█	█	█	▒	▒																				
2.3	Research Bluetooth communications					▒	█	█	▒	▒																				
2.4	Research methodologies					▒	▒	█	█	▒																				
2.5	Write					▒	▒	█	█	█	Initial Document Deadline											Presentation								
3	Develop Broadcast RID for Pi	6 Wks									▒	▒	▒	▒	▒	▒														
3.1	Define tasks and acceptance criteria										█	▒	▒	▒	▒	▒														
3.2	Send dummy data over BLE / WiFI										▒	█	▒	▒	▒	▒														
3.3	Write custom advertising packets for BT 5 / BT 4										▒	▒	█	█	▒	▒														
3.4	Packet Authentication										▒	▒	▒	█	█	█														
X	JANUARY EXAMS																🟥	🟥	EXAMS	🟥	🟥									
4	Develop Observer app	5 Wks																				▒	▒	▒	▒	▒				
4.1	Define tasks and acceptance criteria																					█	▒	▒	▒	▒				
4.2	Design UI																					▒	█	▒	▒	▒				
4.3	Receive RID data over BT / WiFi																					▒	▒	█	█	▒				
4.4	Query RID data with sample registry																					▒	▒	▒	█	▒				
4.5	Build and deploy test registry																					▒	▒	▒	▒	█				
5	Real world testing	2 Wks																									▒	▒		
5.1	Attach pi to drone securely																										█	▒		
5.2	Gather data on maximum working distance																										▒	█		
5.3	Test user story acceptance criteria																										▒	█		
6	Polishing time	2 Wks																											█	█

Figure 3.1: Original Project Timeline

Despite being generous in the original estimations on the project timeline, there were, perhaps inevitably, some tasks that took longer than originally estimated or proved themselves beyond the scope of an undergraduate project. The tasks that turned out to be most problematic were the Broadcast RID packet authentication and real world testing. However despite these setbacks, I was able to complete the project by the end of March as per the original estimation.

3.2 Aims and milestones

As briefly mentioned, the Broadcast RID packet authentication and real world testing proved themselves to be more challenging than originally estimated. The original real world testing plan was to attach our Raspberry Pi RID module to a drone, and slowly fly it away in 10 metre increments until I was no longer receiving RID data on the observer device. This plan was unfortunately made infeasible due to gravity, as the weight of battery pack required to power my Raspberry Pi whilst attached to the drone was too heavy for the drone to lift. The overall weight of my Raspberry Pi with batteries was 150g, there is no official data on the payload capacity of most small drones, but droneenthusiast.com states that "A popular and frequently-used drone such as the Dji Phantom 4 can carry 1.02 lbs as a safe, and everyday payload." [15] 1.02lbs is approximately 463g so this would suggest that a Dji Phantom 4 drone would be a suitable candidate for this type of testing.
Fortunately I was able to redesign the real world testing so that not only was it now possible, but it would also provide more reliable results, and I will elaborate on this in chapter 5. The packet authentication issues were not quite as easily remedied due to the specifics of my implementation, which will be explained in chapter 4. The result of this was that I concluded packet authentication was out of scope for this project.
Despite these setbacks, I was still able to complete all of the tasks within their respective sprints which meant the project was completed on time.

3.3 Risks

Fortunately, none of the risks identified in the initial document became a reality during the course of the project. Unfortunately, there were a number of unforeseen issues which cropped up; the first of these was that just before the testing phase, the Android device that was going to be used for testing became corrupted. Due to the nature of my implementation, it is not possible to test all functionality with a virtual device. Thankfully, I was able to borrow a suitable replacement for few days so progress wasn't affected to a large extent. The other issue which became apparent was the lack of detailed documentation for Bluetooth Low Energy Broadcast Advertising, which is used in my implementation of Broadcast RID. This was hard to work around as Broadcast RID is a particularly niche concept so there weren't many options to choose from when it came to deciding on specifics of our implementation. This should have been something that was identified during the initial project research, but upon initial reading it did appear that there was a wealth of information on the subject. Fortunately it was possible to leverage the limited documentation available to produce a working proof of concept.

Chapter 4

Implementation and achievements

As previously discussed, there are two different types of Drone RID. These are Broadcast RID and Network RID. I have opted to implement a Broadcast RID system for several reasons:

- I expect my system to work in areas without network coverage
- The Swiss have already implemented a Network RID system, so developing a Broadcast RID system would be a novel challenge.
- Network RID would require additional hardware which isn't within my budget.

4.1 Implementation Overview

My implementation is based on a fusion of ideas from Card et al [6], Dr Pardeep Kumar (my project supervisor), and a scattering of other sources. In chapter 2, I discussed the high level layout of a Drone RID system, and this was comprised of three main parts. The first part being an observer device that receives data, which I have implemented by creating an Android app using Kotlin. I used Google Firebase to create a prototype registry and I used Bluetooth Low Energy on a Raspberry Pi to simulate a drone.

4.2 Registry Prototype

In the real world, this registry would be owned by the relevant aviation authority. Within this project, the registry purely acts as a prototype for what a real world registry may look like.

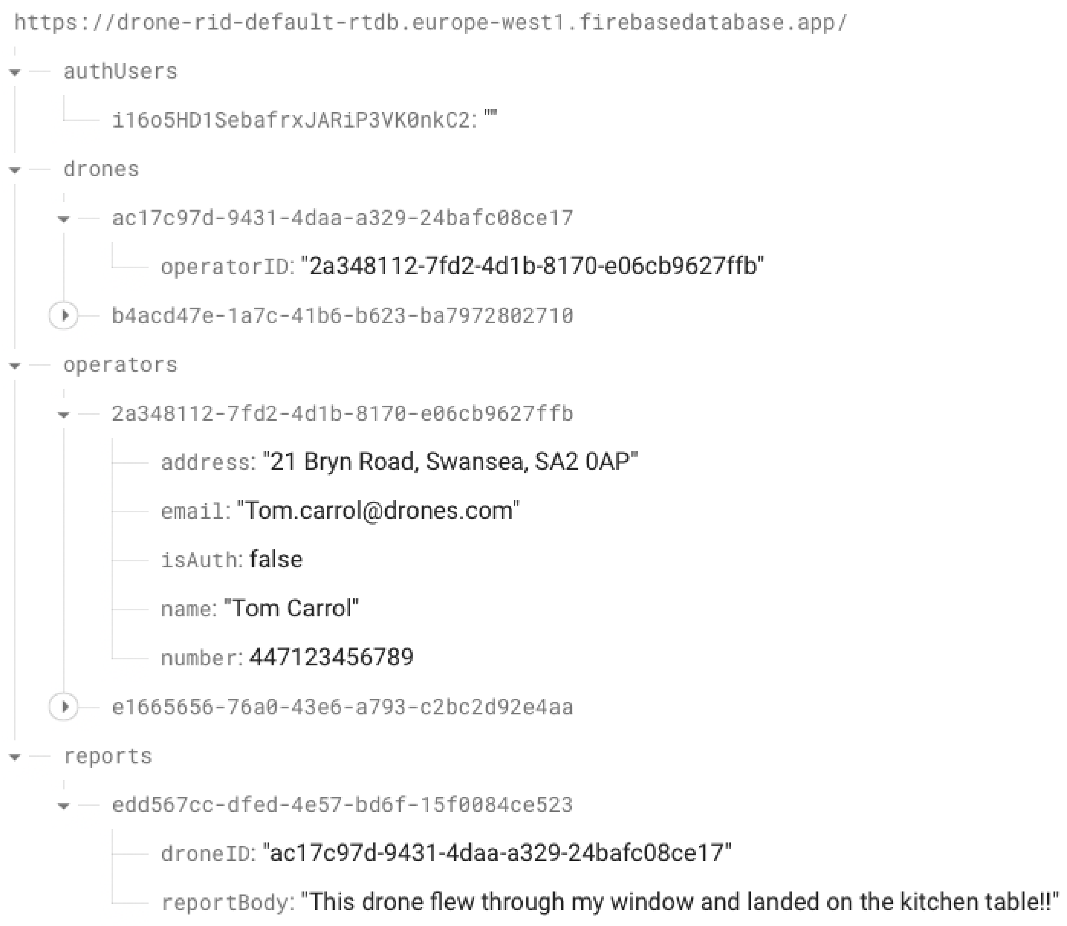
However, this doesn't mean that my registry isn't functional. My registry is responsible for three areas:

1. Operators - Storage of Operator ID, name, and contact information. If a drone belonging to this operator ID is identified for misuse, then authorities can use this information to track down the operator.

2. Drones - Storage of Drone ID, the ID of each drone's operator, and the public key of each drone if using public key authentication

3. Reports - Storage of report ID, the ID of the drone being reported as well as the details of the report.

I have used Google Firebase's Realtime Database to implement the registry. I primarily chose to use Firebase as it is well integrated with Android, which is what I have used to implement the Observer app. Realtime Database is a NoSQL database that stores data in a JSON tree, which is suitable for the simple queries that are performed in this system. Figure 4.1 shows a screenshot of our registry



```
https://drone-rid-default-rtdb.europe-west1.firebasedatabase.app/
▾ — authUsers
    └── i16o5HD1SebafrxJARiP3VK0nkC2: ""
▾ — drones
    ▾ — ac17c97d-9431-4daa-a329-24bafc08ce17
        └── operatorID: "2a348112-7fd2-4d1b-8170-e06cb9627ffb"
    ▸ — b4acd47e-1a7c-41b6-b623-ba7972802710
▾ — operators
    ▾ — 2a348112-7fd2-4d1b-8170-e06cb9627ffb
        ├── address: "21 Bryn Road, Swansea, SA2 0AP"
        ├── email: "Tom.carrol@drones.com"
        ├── isAuth: false
        ├── name: "Tom Carrol"
        └── number: 447123456789
    ▸ — e1665656-76a0-43e6-a793-c2bc2d92e4aa
▾ — reports
    ▾ — edd567cc-dfed-4e57-bd6f-15f0084ce523
        ├── droneID: "ac17c97d-9431-4daa-a329-24bafc08ce17"
        └── reportBody: "This drone flew through my window and landed on the kitchen table!!"
```

Figure 4.1: Screenshot of the registry in Firebase

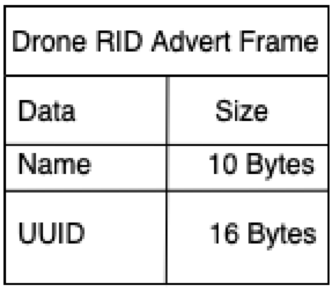
4.3 Simulated Drone

As discussed in chapter 2, it is expected that RID will be implemented either within drone operating systems, or as an external piece of hardware that can be added to drones. For my implementation I have opted to take the external hardware approach, as I wanted to make a solution that worked for a wide range of drones, rather than just one specific brand. My implementation is largely based on suggestions and requirements by Card et al's Drone Remote Identification Protocol Requirements [6]. The broad idea of my implementation is that the Raspberry Pi would be configured with the corresponding drone ID when the drone is registered. The Raspberry Pi can then be attached to medium sized drones, such as the DJi phantom range, using a 3d printed platform and tape.

4.3.1 Bluetooth Low Energy Overview

ASTM International [16] specifies four different data links for Broadcast RID, these are Bluetooth 4, Bluetooth 5 with Extended Advertisements and Extended Range, Wifi NAN at 2.4GHz and Wifi NAN at 5GHz. When researching Wifi NAN, I found that Observer devices would require specific hardware to support it [17]. As I wanted to my solution to be as accessible as possible, I opted to use Bluetooth. Card et al's requirements stipulate that Bluetooth implementation must support at least Bluetooth 4.x and optionally support Bluetooth 5, this is again to increase accessibility, as not all devices support Bluetooth 5. For these reasons I chose to use Bluetooth 4.x for my implementation. Bluetooth can be broken down into Bluetooth Classic, and Bluetooth Low Energy (BLE). Bluetooth Classic is used for connected point-to-point communication, which isn't the category that Broadcast RID falls into. BLE has built in support for connectionless broadcast communication [18], which is what I used in my implementation. The speed at which drones can travel relative to observers means that there will often not be time to form a connection before transmitting our RID data. This led me to utilise a subsection of BLE called *Advertising*. BLE advertising allows us to broadcast "frames" containing our RID data, however for Bluetooth 4.x these frames have a payload limit of 31 bytes [6] which presents a significant restraint on our data. Due to this restriction, I was unable to fit any sort of message authentication within the broadcast RID data itself. There are techniques which could be used to work around this, and I will discuss these in chapter 6, but implementing them proved to be outside the scope of an undergraduate project.

In his book, *Getting Started with Bluetooth Low Energy* [19], Kevin Townsend describes the layout of a BLE advertising system. BLE communications are defined in the Generic Attribute Profile (GATT) which establishes in detail how to exchange data over BLE. There are two roles within GATT that are relevant to our implementation: these are services and characteristics. Services are made up of characteristics, which the specific data objects we want to advertise through the service. In this implementation, our Raspberry Pi acts as our broadcaster, and it broadcasts the contents of the Drone RID service once every second.



Drone RID Advert Frame	
Data	Size
Name	10 Bytes
UUID	16 Bytes

Figure 4.2: Layout of the BLE advertising frames

In order to do this programmatically, I utilised the documentation by Bluetooth [20]. I used Python and the BlueZ library to create our advertising frames and then broadcast them using the Raspberry Pi's built in Bluetooth module. An interesting excerpt of my implementation is below, and the full code listing is in Appendix B. This excerpt shows setting up the advertising data which Bluetooth calls a *characteristic* of our Drone RID service. I then start advertising this service along with it's characteristic data.

```python
class DroneRIDCharacteristic(bluetooth_gatt.Characteristic):
  name = "name"
  drone_uuid = "uuid"

  def __init__(self, bus, index, service):
    bluetooth_gatt.Characteristic.__init__(self, bus, index, bluetooth_constants.DRONE_RID_CHR_UUID,['read','notify'],service)
    self.name = "SwanDrone"
    self.drone_uuid = "10318a23-75d6-4868-bbf9-cee1804ed43d"


class Advertisement(dbus.service.Object):
    PATH_BASE = '/org/bluez/ldsg/advertisement'

    def __init__(self, bus, index, advertising_type):
        self.path = self.PATH_BASE + str(index)
        self.bus = bus
        self.ad_type = advertising_type
        self.service_uuids = [bluetooth_constants.DRONE_RID_SVC_UUID]
        self.manufacturer_data = None
        self.solicit_uuids = None
        self.service_data = None
        self.local_name = 'Drone RID'
        self.include_tx_power = True
        self.data = None
        self.discoverable = True
        dbus.service.Object.__init__(self, bus, self.path)
```

```

def start_advertising():
    global adv
    global adv_mgr_interface
    # we're only registering one advertisement object so index (arg2) is hard
        coded as 0
    print("Registering advertisement",adv.get_path())
    adv_mgr_interface.RegisterAdvertisement(adv.get_path(), {},
                                            reply_handler=register_ad_cb,
                                            error_handler=register_ad_error_cb)
```

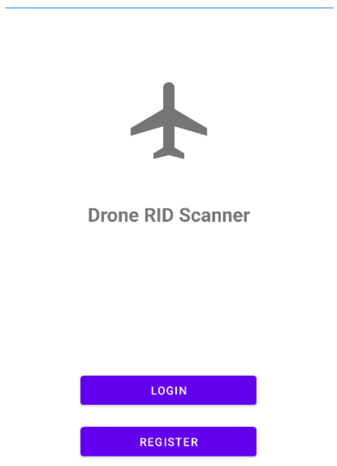
Listing 4.1: Excerpt of Broadcast RID advertising code

4.4 Observer App

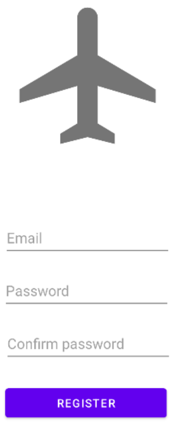
The Observer app, named "Drone RID Scanner" is an Android App developed in Kotlin. I chose to build an Android app rather than iOS because Android has a larger market share as of March 2022, with 71.7% of mobile phone owners using Android, compared to the 21.7% share of iOS [21]. The app can be broken into three parts: User Authentication, Drone Scanning, and Drone Detail.

4.4.1 User Authentication

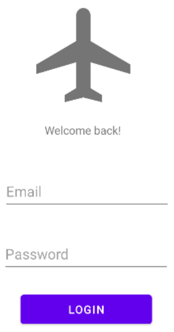
In order to protect the privacy of drone operators, I have split users into two categories: general public and authorised officials. I have used Firebase's built-in authentication to setup a simple user registration/login system in the app, and system admins are able to elevate general public users to authorised officials. Using Firebase's authentication service means that user passwords are securely stored behind Google's security, meaning that even system admins are unable to view user passwords in plaintext. This means that there is no single point of failure within the app that would result in a breach of all user passwords, significantly increasing user security and hopefully increasing user trust. Figure 4.3 shows screenshots of the authentication views in our app.



A. Welcome page



B. Register page



C. Login page

Figure 4.3: Screenshots of app user authentication pages

4.4.2 Drone Scanning

The key feature of the app is its ability to scan for nearby drones. The simulated drone transmits broadcast RID data using a specific Bluetooth Service ID, which has the form of a Universally Unique Identifier (UUID). Android allows me to filter my scanning so that only devices broadcasting on this service ID are shown to Observers, which means that observers will only see nearby drones in their app rather than lots of irrelevant bluetooth devices. If we find a drone in our scan, we load in into a *Recycler View*, which displays basic information about the drone in a card view. The user can then tap on the drone card to open the drone detail view which is where users can report drone misuse.

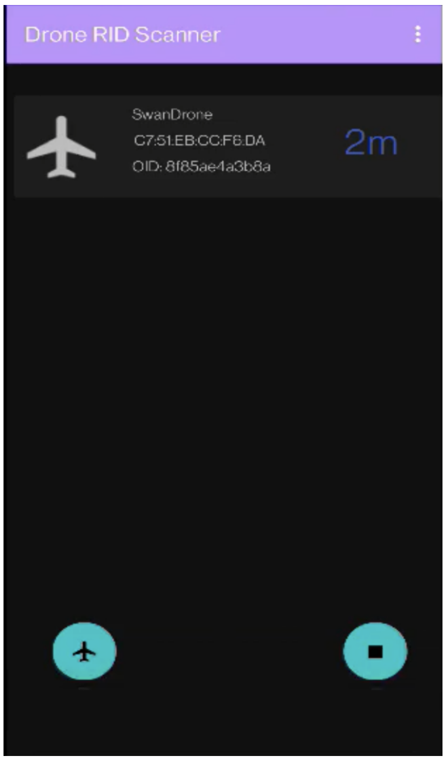


Figure 4.4: A screenshot of the scanning view of the app

The distance of a drone from an observer can be approximated using the following formula [22]:

$$Distance = \frac{MeasuredPower - RSSI}{10 * N}$$

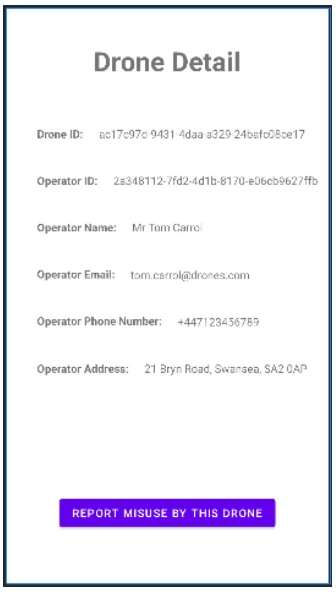
Where:

- *MeasuredPower* is the factory-calibrated constant of expected RSSI at a distance of 1 metre
- *RSSI* is the Received Signal Strength Indicator
- *N* is constant which depends on the environmental factor with a range 2-4

Values for *MeasuredPower* and *RSSI* are provided by the scan data. From my experiment detailed in chapter 5, I determined 2 to be the best value for N.

4.4.3 Drone Detail

The drone detail view is where the varying levels of user authorisation are visible. In order to protect the privacy of drone operators, we only pull identifying information about drone operators from our registry when the user is an authorised official. Regular users will be able to see only the UUIDs representing the drone and operator IDs. In the case of serious drone misuse, such as a drone flying over an airport or other restricted airspace, officials can use this information to help police track down the operator.



Drone Detail

Drone ID: ac17c97d-9431-4daa-a329-24bafc08ce17

Operator ID: 2a348112-7fd2-4d1b-8170-e06cb9627ffb

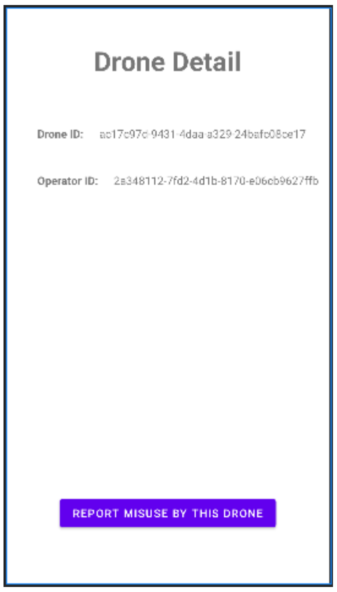
Operator Name: Mr Tom Carrol

Operator Email: tom.carrol@drones.com

Operator Phone Number: +447123456789

Operator Address: 21 Bryn Road, Swansea, SA2 0AP

A. Authorised View



Drone Detail

Drone ID: ac17c97d-9431-4daa-a329-24bafc08ce17

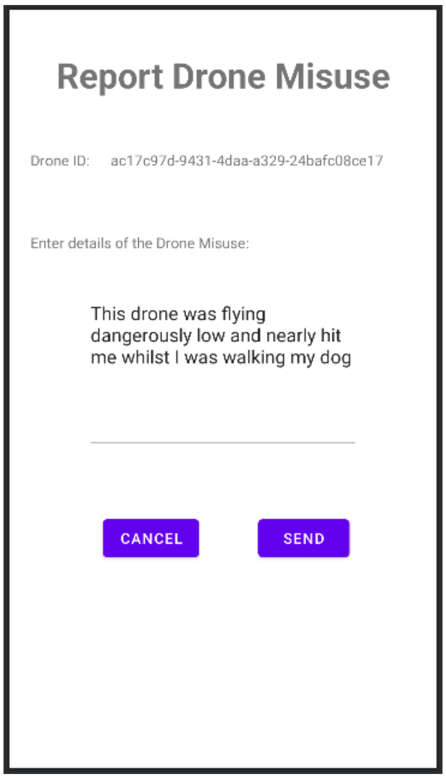
Operator ID: 2a348112-7fd2-4d1b-8170-e06cb9627ffb

B. Unauthorised view

Figure 4.5: Screenshots of Drone detail view for both levels of user authorisation

4.4.4 Drone Misuse Reporting Tool

I have built a tool into the app which allows observers to easily report drone misuse by any drones that they have come into contact with whilst scanning. The tool takes the drone ID of the selected drone and, combined with the details of the observer's report, uploads it to the registry.


Report Drone Misuse

Drone ID: ac17c97d-9431-4daa-a329-24bafc08ce17

Enter details of the Drone Misuse:

This drone was flying dangerously low and nearly hit me whilst I was walking my dog

Figure 4.6: A screenshot of the reporting tool in the app

Currently the reports can be viewed by system admins who have registry access, however there are many ways that the reporting feature could be extended which will be discussed in chapter 6.2.

Chapter 5

Evaluation of Implementation

Nordic Semiconductor state that the maximum range of Bluetooth 4.x Low Energy is 100m [23]. I hypothesised that my RID system would work at this range. My system is intended to be used outside, with clear line of sight, which are ideal conditions for Bluetooth. In order to determine whether or not my hypothesis was correct, I carried out a simple experiment.

5.1 Method

Equipment: Raspberry Pi with battery pack, Android phone, metre measuring wheel.
I identified Swansea beach as a suitable area for this experiment as it is a large open space with minimal radio interference. I set my Raspberry Pi to start advertising and walked away from it with the measuring wheel. Every 5 metres I stopped and restarted the scan to see if I was still receiving RID data, and if so, whether the RID data was correct and what the displayed proximity on the app was.

5.2 Results

Distance (metres)	Correct RID data visible?	App proximity reading (metres)
0	Yes	1
10	Yes	7
20	Yes	19
30	Yes	28
40	Yes	35
50	Yes	41
60	No	NA
70	No	NA
80	No	NA
90	No	NA
100	No	NA

The results show that my system has a maximum effective range of 50m and also shows that the proximity reading in the app loses accuracy at distances of over 30m.

Chapter 6

Conclusions and Future Work

In this dissertation I have discussed the problem of drone misuse and the flaws of current solutions. I have introduced the concept of Drone Remote ID and demonstrated the implementation of a Drone Remote ID system.

6.1 Contributions

The main contributions of this work can be summarised as follows:

- **Observer App**

  The Observer app is an accessible and easy to use app for Android which allows users to observe nearby drones, as well as report misuse. If the user is an authorised official, they are able to see personal information about the drone operator. I believe that the app is viable for real world release, though it could be improved in a few ways, which will be discussed in section 6.2.

- **Broadcast RID using Raspberry Pi to simulate a drone** I successfully implemented Broadcast RID using Bluetooth 4 Low energy. The maximum effective range of my implementation is 50m, which is viable for some cases but would require improvement before it could be released on a large scale. Drones can cover 100m in a matter of seconds so it's possible that a drone could fly directly over an observer and the observer would not be able to start a scan before the drone has gone out of range. The current implementation isn't authenticated, meaning that it is possible for somebody to spoof the ID of a drone.

- **Registry Prototype**

  One of the reasons I chose to build the registry on Firebase was due to the scalability of the Firebase platform. The prototype would be viable in the real world, though some changes to make it more usable will be discussed in section 6.2

6.2 Future Work

6.2.1 Observer App

As the experiment in chapter 5 showed, the proximity shown in the app isn't particularly accurate over 30m. Therefore I believe there is more work to be done to find a formula that can accurately calculate the proximity of drones to observer devices. The app would also benefit from a UI improvement as at present it has a very basic design.

6.2.2 Raspberry Pi Drone Simulation

As previously mentioned, the current effective range of my Broadcast RID is not quite viable for widespread release. However this can be remedied by adding improved Bluetooth hardware to the Raspberry pi, such as a larger antenna. There is also the issue of a lack of authentication. Due to the small size of advertising frames on Bluetooth 4, authentication would require the implementation of a concept called *paging*, where RID messages are broken into a number of pages that come together to form a single RID message. This is possible by adding a message ID to each frame and then having the receiving device put every received frame with the same message ID together. However this proved difficult to implement as it introduces problems of data dropout. If for whatever reason a single frame is lost, then that entire message is unusable as it cannot be authenticated.
A more viable way to overcome these problems would be to build an implementation for Bluetooth 5. Bluetooth 5 comes with a feature called Extended Advertising which increases the size of advertising frames to 255 bytes [6]. Bluetooth 5 also has an Extended Range feature which increases theoretical maximum range to 1 kilometre [6].

6.2.3 Registry Prototype

The registry could be improved through building a comprehensive front-end so that drone operators are able to register new drones, and officials are able to view reports in a clean format. These changes would make the prototype representative of a full aviation authority drone management system.

6.3 Closing Thoughts

Drone Remote Identification will become a crucial technology in order to protect people and infrastructure from drone misuse. Drone numbers have been growing year on year and will continue to do so, and it's reasonable to expect that drone misuse will grow with it. By the 16^{th} of September 2023, all drone operators in the US and EU will be required to be compliant with Drone RID and at this point we'll be able to see just how effective Drone RID is at preventing drone misuse.

Bibliography

[1] I. Guvenc, F. Koohifar, S. Singh, M. L. Sichitiu and D. Matolak, "Detection, tracking, and interdiction for amateur drones," *IEEE Communications Magazine*, vol. 56, no. 4, pp. 75–81, 2016.

[2] BBC News. (2018) Gatwick airport: Drones ground flights. [Online]. Available: https://www.bbc.co.uk/news/uk-england-sussex-46623754

[3] Drone Industry Insights. (2021) Drone market report 2021-2026. [Online]. Available: https://droneii.com/product/drone-market-report

[4] S. Prisznyák, "Drones and jails," *Scientific Bulletin-Nicolae Balcescu Land Forces Academy*, vol. 23, no. 1, pp. 43–52, 2018.

[5] M. White. (2016) Drone used to smuggle cell phones and saw blades into prison by flying it through 4th-floor window. [Online]. Available: https://goo.gl/jgwoYS

[6] C. et Al. (2022) Drone remote id protocol. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-drip-reqs/

[7] G. Lykou, D. Moustakas, and D. Gritzalis, "Defending airports from uas: A survey on cyber-attacks and counter-drone sensing technologies," *Sensors*, vol. 20, no. 12, p. 3537, 2020.

[8] P. Nguyen, H. Truong, M. Ravindranathan, A. Nguyen, R. Han, and T. Vu, "Matthan: Drone presence detection by identifying physical signatures in the drone's rf communication," in *Proceedings of the 15th annual international conference on mobile systems, applications, and services*, 2017, pp. 211–224.

[9] Michel, A. (2018) Counter-drone systems. [Online]. Available: https://dronecenter.bard.edu/files/2018/02/CSD-Counter-Drone-Systems-Report.pdf

[10] F. A. Administration. (2021, October) Uas remote identification overview. [Online]. Available: https://www.faa.gov/uas/getting_started/remote_id/

[11] (2021) Faa announces effective dates for final drone rules. [Online]. Available: https://www.faa.gov/newsroom/faa-announces-effective-dates-final-drone-rules?newsId=97022

[12] Y. Zhi, Z. Fu, X. Sun, and J. Yu, "Security and privacy issues of UAV: A survey," *Mobile Networks and Applications*, vol. 25, no. 1, pp. 95–101, January 2019. [Online]. Available: https://doi.org/10.1007/s11036-018-1193-x

[13] S. U.-S. I. Program. (2019, September) Susi's remote id demonstration of september 16 2019. [Online]. Available: https://susi.swiss/2019/09/16/remote-id-demonstration-report/

[14] de Jaeger, W. (2021, September) Switzerland launches world's first remote id network for drones. [Online]. Available: https://www.dronewatch.eu/switzerland-launches-worlds-first-remote-id-network-for-drones/

[15] DroneEnthusiast.com. (2021, April) 5 best heavy lift drones [updated 2021]- large drones that have high lift capacity. [Online]. Available: https://www.dronethusiast.com/heavy-lift-drones/

[16] A. International. (2020, February) Standard specification for remote id and tracking. [Online]. Available: http://dx.doi.org/10.1520/F3411-19

[17] A. Developers. (2021, January) Wi-fi aware overview. [Online]. Available: https://developer.android.com/guide/topics/connectivity/wifi-aware

[18] Bluetooth. (2022) Bluetooth technology overview. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

[19] Townsend, K. Cufí, C. Akiba. Davidson, R., *Getting Started with Bluetooth Low Energy*. O'Reilly Media, Inc, May 2014.

[20] M. Wooley. (2022, January) The bluetooth technology for linux developers study guide. [Online]. Available: https://www.bluetooth.com/blog/the-bluetooth-for-linux-developers-study-guide/

[21] statcounter.com. (2022, March) Mobile operating system market share worldwide mar 2021 - mar 2022. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[22] A. Ghosh. (2020, march) Note on calculating distance from rssi value of ble devices. [Online]. Available: https://thecustomizewindows.com/2020/03/note-on-calculating-distance-from-rssi-value-of-ble-devices/

[23] J. Sponås. (2018, February) Things you should know about bluetooth range.

Appendix A

App Authentication Implementation

```
package com.bruceprw.dronerid_observer



import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import android.view.View
import android.view.inputmethod.InputMethodManager
import android.widget.Button
import android.widget.EditText
import com.google.android.material.snackbar.Snackbar
import com.google.firebase.auth.FirebaseAuth

class LoginActivity : AppCompatActivity()
{
    private var mAuth = FirebaseAuth.getInstance()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.login_page)

        val loginButton = findViewById<Button>(R.id.login_button)
        val emailInput = findViewById<EditText>(R.id.login_email)
        val passwordInput = findViewById<EditText>(R.id.login_password)

        loginButton.setOnClickListener { view ->
            login(view, emailInput.text.toString(), passwordInput.text.toString())

        }
    }

    private fun login(view: View, email: String, password: String ) {
        mAuth.signInWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) {
                    task ->
                if (task.isSuccessful) {
                    val user = mAuth.currentUser!!.displayName
                    closeKeyboard()
                    showMessage(view, getString(R.string.login_success_toast))
                    val intent = Intent(this, MainActivity::class.java)
                    intent.putExtra("user", user)
                    startActivity(intent)
                }
                else {
                    closeKeyboard()
                    showMessage(view, getString(R.string.login_failure_toast))

                }
            }
    }
    private fun closeKeyboard() {
        val view = this.currentFocus
        if (view != null) {
            val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as
                InputMethodManager
            imm.hideSoftInputFromWindow(view.windowToken, 0)
        }
    }
    private fun showMessage(view: View, message: String) {
        Snackbar.make(view, message, Snackbar.LENGTH_LONG).show()
    }
}
```

Listing A.1: Login Activity

```
package com.bruceprw.dronerid_observer

import android.content.Context
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import android.util.Log
import android.view.View
import android.view.inputmethod.InputMethodManager
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import com.google.android.material.snackbar.Snackbar
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.FirebaseUser

class RegisterActivity : AppCompatActivity()
{
    private var mAuth = FirebaseAuth.getInstance()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.register_page)
        val registerButton = findViewById<Button>(R.id.register_button)
        val emailInput = findViewById<EditText>(R.id.register_email)
        val passwordInput1 = findViewById<EditText>(R.id.register_password)
        val passwordInput2 = findViewById<EditText>(R.id.confirm_password)

        registerButton.setOnClickListener{ view ->
            val email = emailInput.text.toString()
            val pass1 = passwordInput1.text.toString()
            val pass2 = passwordInput2.text.toString()
            if (pass1 == pass2) {
                createAccount(view, email, pass1)
            }
            else {
                Toast.makeText(baseContext, R.string.register_pwd_toast,
                    Toast.LENGTH_SHORT).show()
            }
        }
    }


    public override fun onStart() {
        super.onStart()

    }

    private fun createAccount(view: View, email: String, password: String) {
        mAuth.createUserWithEmailAndPassword(email, password)
            .addOnCompleteListener(this) { task ->
                if (task.isSuccessful) {
                    Log.d(TAG, "createUserWithEmail:success")
                    val user = mAuth.currentUser!!
                    val intent = Intent(this, MainActivity::class.java)
                    intent.putExtra("user", user)
                    startActivity(intent)

                } else {
                    Log.w(TAG, "createUserWithEmail:failure", task.exception)
                    showMessage(view, getString(R.string.register_failure_toast))
                }
            }
    }
    private fun showMessage(view: View, message: String) {
        Snackbar.make(view, message, Snackbar.LENGTH_LONG).show()
    }


    private fun closeKeyboard() {
        val view = this.currentFocus
        if (view != null) {
            val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as
                InputMethodManager
            imm.hideSoftInputFromWindow(view.windowToken, 0)
        }
    }
    companion object {
        private const val TAG = "RegisterUser"
    }
}
```

Listing A.2: Register Activity

Appendix B

Broadcast RID implementation

```
#This code was adapted from the Bluetooth Linux Study Guide
# Available at: https://www.bluetooth.com/blog/the-bluetooth-for-linux-developers-
    study-guide/

# Broadcasts connectable advertising packets
import bluetooth_gatt
import random
import bluetooth_constants
import bluetooth_exceptions
import dbus
import dbus.exceptions
import dbus.service
import dbus.mainloop.glib
import sys
from gi.repository import GLib
sys.path.insert(0, '.')

bus = None
adapter_path = None
adv_mgr_interface = None

class Advertisement(dbus.service.Object):
    PATH_BASE = '/org/bluez/ldsg/advertisement'

    def __init__(self, bus, index, advertising_type):
        self.path = self.PATH_BASE + str(index)
        self.bus = bus
        self.ad_type = advertising_type
        self.service_uuids = [bluetooth_constants.DRONE_RID_SVC_UUID]
        self.manufacturer_data = None
        self.solicit_uuids = None
        self.service_data = None
        self.local_name = 'Drone RID'
        self.include_tx_power = False
        self.data = None
        self.discoverable = True
        dbus.service.Object.__init__(self, bus, self.path)

    def get_properties(self):
        properties = dict()
        properties['Type'] = self.ad_type
        if self.service_uuids is not None:
            properties['ServiceUUIDs'] = dbus.Array(self.service_uuids,
                                                    signature='s')
        if self.solicit_uuids is not None:
            properties['SolicitUUIDs'] = dbus.Array(self.solicit_uuids,
                                                    signature='s')
        if self.manufacturer_data is not None:
            properties['ManufacturerData'] = dbus.Dictionary(
                self.manufacturer_data, signature='qv')
        if self.service_data is not None:
            properties['ServiceData'] = dbus.Dictionary(self.service_data,
                                                        signature='sv')
        if self.local_name is not None:
            properties['LocalName'] = dbus.String(self.local_name)
        if self.discoverable is not None and self.discoverable == True:
            properties['Discoverable'] = dbus.Boolean(self.discoverable)
        if self.include_tx_power:
            properties['Includes'] = dbus.Array(["tx-power"], signature='s')

        if self.data is not None:
            properties['Data'] = dbus.Dictionary(
                self.data, signature='yv')
        print(properties)
        return {bluetooth_constants.ADVERTISING_MANAGER_INTERFACE: properties}

    def get_path(self):
        return dbus.ObjectPath(self.path)

    @dbus.service.method(bluetooth_constants.DBUS_PROPERTIES,
                         in_signature='s',
                         out_signature='a{sv}')
    def GetAll(self, interface):
        if interface != bluetooth_constants.ADVERTISEMENT_INTERFACE:
            raise bluetooth_exceptions.InvalidArgsException()
        return self.get_properties()[bluetooth_constants.
            ADVERTISING_MANAGER_INTERFACE]

```

```python
    @dbus.service.method(bluetooth_constants.ADVERTISING_MANAGER_INTERFACE,
                         in_signature='',
                         out_signature='')
    def Release(self):
        print('%s: Released' % self.path)

def register_ad_cb():
    print('Advertisement registered OK')

def register_ad_error_cb(error):
    print('Error: Failed to register advertisement: ' + str(error))
    mainloop.quit()

def start_advertising():
    global adv
    global adv_mgr_interface
    # we're only registering one advertisement object so index (arg2) is hard
        coded as 0
    print("Registering advertisement",adv.get_path())
    adv_mgr_interface.RegisterAdvertisement(adv.get_path(), {},
                                        reply_handler=register_ad_cb,
                                        error_handler=register_ad_error_cb)


class DroneRIDCharacteristic(bluetooth_gatt.Characteristic):
  name = "name"
  drone_uuid = 0
  #notifying = False

  def __init__(self, bus, index, service):
    bluetooth_gatt.Characteristic.__init__(self, bus, index, bluetooth_constants.
        DRONE_RID_CHR_UUID,['read','notify'],service)
    self.name = "Drone1"
    self.drone_uuid = "10318a23-75d6-4868-bbf9-cee1804ed43d"
    print("Initial name set to "+str(self.name))


class DroneRIDService(bluetooth_gatt.Service):

  def __init__(self, bus, path_base, index):
    print("Initialising DroneRIDService object")
    bluetooth_gatt.Service.__init__(self, bus, path_base, index,
        bluetooth_constants.DRONE_RID_SVC_UUID, True)
    print("Adding TemperatureCharacteristic to the service")
    self.add_characteristic(DroneRIDCharacteristic(bus, 0, self))

class Application(dbus.service.Object):
  def __init__(self, bus):
    self.path = '/'
    self.services = []
    dbus.service.Object.__init__(self, bus, self.path)
    print("Adding DroneRIDService to the Application")
    self.add_service(DroneRIDService(bus, '/org/bluez/ldsg', 0))

  def get_path(self):
    return dbus.ObjectPath(self.path)

  def add_service(self, service):
    self.services.append(service)


  @dbus.service.method(bluetooth_constants.DBUS_OM_IFACE,out_signature='a{oa{sa{sv
      }}}')
  def GetManagedObjects(self):
    response = {}
    print('GetManagedObjects')

    for service in self.services:
      print("GetManagedObjects: service="+service.get_path())
      response[service.get_path()] = service.get_properties()
      chrcs = service.get_characteristics()
      for chrc in chrcs:
        response[chrc.get_path()] = chrc.get_properties()
        descs = chrc.get_descriptors()
        for desc in descs:
          response[desc.get_path()] = desc.get_properties()
    return response


dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
bus = dbus.SystemBus()
# we're assuming the adapter supports advertising
adapter_path = bluetooth_constants.BLUEZ_NAMESPACE + bluetooth_constants.
    ADAPTER_NAME
print(adapter_path)

adv_mgr_interface = dbus.Interface(bus.get_object(bluetooth_constants.
    BLUEZ_SERVICE_NAME,adapter_path), bluetooth_constants.
    ADVERTISING_MANAGER_INTERFACE)
# we're only registering one advertisement object so index (arg2) is hard coded as
     0
adv = Advertisement(bus, 0, 'peripheral')
start_advertising()

```

```
print("Advertising as "+adv.local_name)

def register_app_cb():
  print('GATT application registered')

def register_app_error_cb(error):
  print('Failed to register application: ' + str(error))
  mainloop.quit()


app = Application(bus)
print('Registering GATT application...')
service_manager = dbus.Interface(bus.get_object(bluetooth_constants.
    BLUEZ_SERVICE_NAME,adapter_path),bluetooth_constants.GATT_MANAGER_INTERFACE)
service_manager.RegisterApplication(app.get_path(), {},reply_handler=
    register_app_cb,error_handler=register_app_error_cb)

mainloop = GLib.MainLoop()
mainloop.run()
```

Listing B.1: Broadcast RID

Appendix C

Supplementary Data

The full code for this project is available on canvas.