Multilateration based system for localization of nodes with LoRaWAN

Daniel Saul
University College London

Supervisor: Dr Miguel Rio

March 19, 2017

Abstract

I wish I were an abstract.

Contents

Chapter 1

Introduction

The Internet of Things (IoT) is one of the most hyped of technology buzzwords at the moment but the term was coined as long ago as 1999 by Kevin Ashton. Back then, all data on the internet was generated or uploaded by humans, whether text, images or information. But Ashton envisioned a future of computer systems generating and collating data on their own, with little to no input from us. Today we are in the situation where

1.1 Low Power Wide Area Networks

1.2 LoRaWAN

1.2.1 Things Connected

1.3 Motivations

1.4 Project Outline

Chapter 2

Localization

2.1 Received Signal Strength

2.2 Angle of Arrival

2.3 Time of Arrival

2.4 Time Difference of Arrival

2.4.1 TDOA Equations

2.4.2 Hyperbolic Solver Algorithms

2.4.3 Problems

Chapter 3

Implementation

The implemented test setup consisted of a simple custom node, the existing Things Connected LoRaWAN infrastructure across London and custom server-side software. The node transmits its real location as calculated by the on-board GPS which is received by LoRaWAN gateways and uploaded to the server. These real coordinates of the transmission location can then be later compared to the estimated location by the multilateration algorithm.

3.1 Node Hardware

A simple node was designed, built and programmed in C, consisting of an ESP8266 microcontroller, u-blox MAX8C GPS receiver and a Microchip RN2483 LoRa transceiver, as shown in figure 3.1. The GPS receiver used a basic chip antenna whilst the LoRA transceiver used an 868MHz 'rubber ducky' antenna. The specific hardware is not particularly important however, but rather the functionality is.

The GPS receiver was configured to pedestrian mode since the primary mode of testing would be a person walking with the node. The LoRaWAN configuration parameters (device address, network session key and application session key) were hard-coded into the microcontroller firmware.

The microcontroller follows a basic loop. In each iteration, set at roughly every 10 seconds, the microcontroller requests data from the GPS (latitude, longitude, altitude, number of satellites and time) and loads it into a struct datatype. A counter in the microcontroller's flash memory, such that it is persistent across power cycling, is incremented and also added to the struct. The entire 20 byte struct is then transmitted as a binary blob payload by the LoRa transceiver. Other functionality allowing transmission on a button press and short bursts of continuous data transmission was also implemented but not ultimately used.

The main file of the node firmware can be seen in appendix A.

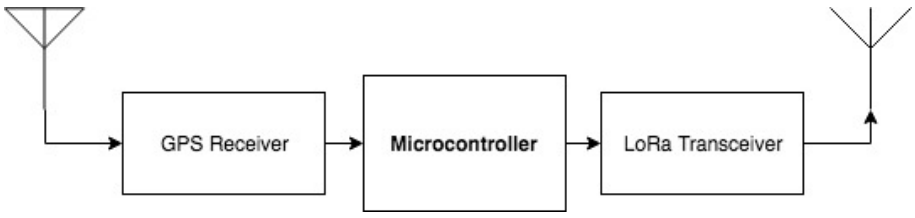

Figure 3.1: Node Diagram

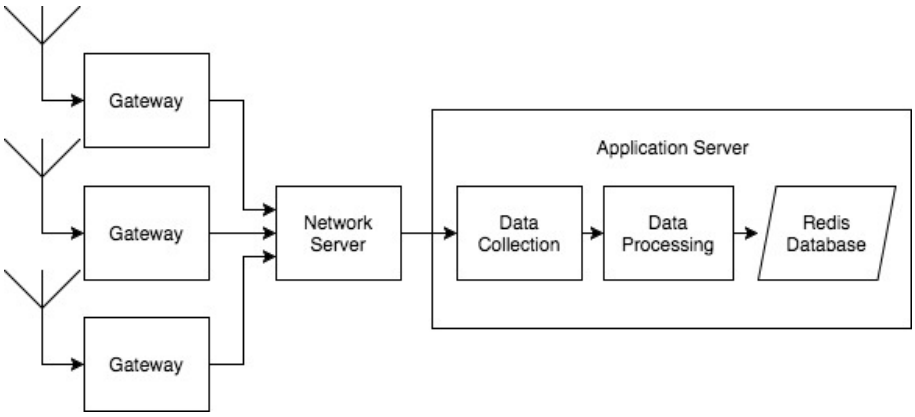

Figure 3.2: LoRaWAN to Server Diagram

3.2 Server-side Software

When a gateway receives a packet from a node, it uploads it to the network server. The data can then be passed by the network server over an HTTP connection to the relevant application server, if the application server has implemented the necessary JSON-RPC methods [1]. A python script to be run on a public server with open port 8080 is found in appendix B.1 and implements the 'uplink' and 'post_uplink' methods. The 'uplink' method provides data as soon as one gateway has received a packet whilst the 'post_uplink' method should provide data on all receiving gateways a few seconds later. Unfortunately after testing, it was found that this latter method is not functional and that therefore obtaining details on all gateways is not possible this way.

Therefore, a workaround was found by using the unofficial API running the Things Connected dashboard. The last 20 received packets from each gateway are available from a 'messages' endpoint. By continuously polling this endpoint at a rate faster than that at which the node is transmitting, all data from all gateways can be collected. Figure 3.2 shows a simplified system diagram for the LoRaWAN and server-side systems. A python script to perform an HTTP POST request to login and continually fetch data from the unofficial API through HTTP GET requests is found in appendix B.2. The fetched data is published to a Redis Pub/Sub channel which is subscribed to by another script, found in appendix B.3, where the data is processed. The payload of each packet is decoded and the binary blob unpacked into the constituent pieces of data. If the packet payload already exists in the Redis database, the details of the new gateway receiving the packet is appended to the gateway list. If it doesn't already exist, the entire packet data is freshly inserted into the database.

It is at this point in the system that any final positioning algorithm would ultimately be included. Instead, another python script, in appendix B.4, allows all the received data to be exported from the database to a JSON file for analysis and testing.

3.3 Localization Software

Chapter 4

Data Analysis & Results

Chapter 5

Conclusion

Chapter 6

Example Chapter

6.1 Aims

I aim to do things, as shown in equation (6.1). I will show how to do this in Section 6.2.

$$f(x) = \dot{x} + Bu(t) + Cx(x) \tag{6.1}$$

6.2 Apparatus

50% of my study.

A diagram of how to set up the apparatus is shown in figure 6.1.

6.3 Observations

6.4 Discussion

These results can be seen in table 6.4.

Figure 6.1: Apparatus setup

Table 6.1: Predicted and measured beam deflections

Thing	thing	thing	thing
Another	another	another	another

Bibliography

[1] Everynet Core API v1.0, 2017.

Appendix A

Node Firmware

Appendix B

Server-Side Software

B.1 Python Everynet API Script

B.2 Python Fetch Script

B.3 Python Processing Script

B.4 Python Export Script