Student: Stephen Roderick
Student Number: 15023712
Date: 2018
Document Type: Final Report
Supervisors: Prof Miguel Rio / Dr Clive Poole

Evaluating LoRaWAN in an Urban Environment

Comparison of measured and predicted signal strength in Watford.

An assignment submitted in part fulfilment of the degree of Master of Science Telecommunications (IGDP)

Abstract

Low Power Wide Area Networks, and LoRaWAN in particular, are proposed as a networking technology for the urban Internet-of-Things. LoRaWAN nodes transmit small payloads which are received by internet-connected gateways. Typical ranges are 2 km in built-up areas, increasing to over 10 km in open or mixed terrain. Devices are energy efficient, allowing years of operation on a single battery. It is claimed that a single gateway can serve 10^4 nodes, enabling large areas to be networked with unprecedentedly low levels of infrastructure.

This study evaluates the use of LoRaWAN as an enabler of Smart Cities. An extensive measurement campaign was undertaken. This consisted of a drive survey in Watford, using GPS-enabled nodes fitted to recycling vehicles. A second survey was also done in Jersey. A static reference installation was created to monitor long-term fluctuations in received signal strength and packet loss.

Metadata, including field strength (RSSI), was extracted and transformed using bespoke server software. Results were loaded into a data warehouse and used to tune and validate a deterministic propagation model (Bullington). A real-world network planning exercise was performed in partnership with Watford Borough Council. Composite coverage was calculated for ten candidate sites. Borough-wide coverage, greater than 90%, was predicted for three optimally placed gateways. A computer simulation was developed to model packet loss in large networks.

Keywords:

LoRa, LoRaWAN, LPWAN, Smart Cities, Propagation Modelling, Bullington

Executive Summary

Background

LoRaWAN has been proposed as a networking technology for the Internet of Things and Smart Cities. Specifications are impressive. Typical ranges are 2 km in an urban environment, increasing to 10 km in suburban and rural areas. LoRaWAN devices can be powered for several years using an AA battery. A single base station can serve up to 10,000 nodes. Bitrate is traded for increased range and power efficiency. Typical speeds are in the order of 300 bps to 5 kbps.

LoRaWAN operates in unlicensed bands at 868 MHz. It is based on LoRa modulation. Signals are encoded using frequency shifts or 'chirps'. It is the unique nature of the chirp that allows signals to be distinguished from channel noise. Use cases for LoRaWAN include asset tracking, remote monitoring, and human interface devices. It is under consideration for use in the Citizen Flood Detection Network [1].

Watford Borough Council is in the early stages of a LoRaWAN rollout. Various innovative applications are under development. This includes a city-centre display showing available car parking spaces. The Council is also partnering with Veolia to monitor the capacity of litter bins. The Council operates three gateways and has identified other candidate sites.

Aims of the Research Project

The primary focus of this work is to quantify the performance of LoRaWAN in an urban environment. The following questions are addressed:

- ☐ What are LoRa and LoRaWAN?
- ☐ How does LoRaWAN perform in an urban environment?
- ☐ What steps are required to create a validated propagation model?
- ☐ How many gateways are required to serve the borough of Watford?
- ☐ How do LoRaWAN networks scale with large numbers of nodes?

Achievements

Key achievements of the research are shown in Figure 1. In the main build phase, a static reference installation was created. This enabled measurements to be made of signal strength, packet loss, and power consumption. Server-side software was developed to collect and process test data. This was then published via a RESTful API for display in a bespoke Android app and web-based GUI.

Field measurements are essential in characterising a radio channel. In the present study, data from an extensive drive survey was used to tune and validate a LoRaWAN propagation model. The main drive survey was undertaken in partnership with Watford Borough Council. GPS-enabled nodes were

mounted on recycling vehicles. Packet metadata was collected with the coordinates of where the transmission was sent. A second drive survey was performed in Jersey. This allowed testing in a different radio access environment.

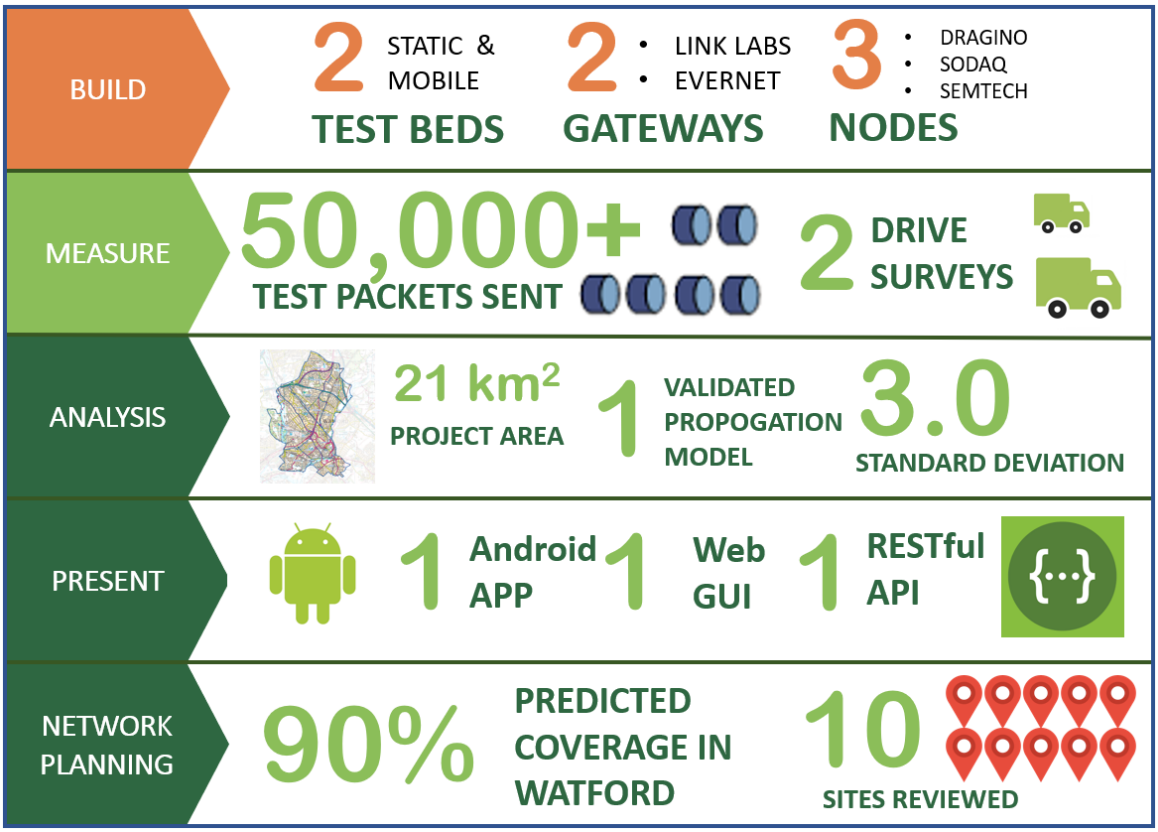


Figure 1: Infographic: Project achievements.

Data from the drive surveys was imported into an industry standard planning tool (ICS Telecom, from ATDI). A calibrated Bullington model was then used in a real-world network planning exercise. Coverage predictions were made for candidate sites. The number and location of gateways were estimated to achieve 90% coverage throughout the borough. Results from this research were used to create a technical white paper. This paper was presented to ATDI.

Conclusions

Based on results from this research, LoRaWAN is a strong candidate for Smart City applications. A maximum range of 10 km was seen. For best performance, nodes should be less than 2 km from a gateway in an urban environment and less than 4 km away in open terrain. Good coverage can be achieved with relatively few LoRaWAN gateways. In the present study, a 21 km^2 area of Watford required three gateways to achieve a composite coverage of 90%. A similar level of coverage was obtained using 10 gateways across Jersey (120 km^2). IoT applications intolerant to high levels of loss require service by multiple gateways. LoRaWAN is likely to face strong competition from 3GPP IoT standards, such as LTE-M and NB-LTE. Despite this, a large install base and competitive pricing models may ensure its future in privately operated networks.

Table of Contents

Introduction and Problem Statement

1.1 Introduction

The Internet of Things (IoT) is a pillar of a fourth industrial revolution. A revolution in how we collect and process information. It is difficult to provide an exact definition, but central to the proposition is Machine to Machine (M2M) communication, embedded systems, and human interface devices. When considering the impact of the IoT, it is common to cite the Gartner estimate that there will be 50 billion connected devices worldwide by 2020 [2]. In fact, estimates vary by tens of billions of devices and are typically being revised downwards. Despite this, IoT is likely to prove transformative. There is a current install base of between 6 and 10 billion devices.

Central to a successful rollout of the IoT is the need for a robust, scalable and secure network. Low Power Wide Area Networks (LPWANs), and LoRaWAN in particular have been proposed as a candidate technology. It has been suggested that LPWANs may eventually account for a quarter of all connected devices [3]. LPWANs trade bit rate for increased range and reduced power consumption. Recent reductions in the prices of nodes and gateways have led to increased uptake and hence greater levels of coverage. LPWANs have shown potential in Smart City applications. Data packets can be sent over long distances reducing the amount of backhaul infrastructure required. An entire city can be served by a relatively small number of base stations.

Watford Borough Council is in the early stages of a LoRaWAN rollout. Their aim is to create new applications to benefit local inhabitants. These include monitoring the capacity of litter bins and reporting on the number of available spaces in council-owned car parks. The Watford network forms part of the wider Things Connected network. Three gateways are operational, with further sites under consideration. The Council has requested assistance in modelling signal propagation around their existing gateways and in assessing the suitability of LoRaWAN for their proposed applications.

1.2 Aims of the Research Project

This study seeks to assess the suitability of LoRaWAN for IoT applications in an urban environment. Range, signal strength, packet loss and power consumption will be investigated and quantified. An industry standard propagation model will be calibrated and verified using data from a drive survey. The model will be used to make coverage predictions in a real-world network planning exercise. The research has the following specific aims:

- ☐ Quantify the range of LoRaWAN signals
- ☐ Determine rates of packet loss and how these vary with distance, time of day and spreading factor.
- ☐ Measure the power consumption of a typical LoRaWAN node and suggest ways to improve efficiency.
- ☐ Use field measurements to select, tune and validate a propagation model.
- ☐ Use the propagation model to estimate coverage for current gateways in Watford.
- ☐ Determine the number and location of gateways to achieve a borough-wide coverage of 90%.
- ☐ Investigate how LoRaWAN networks scale under increased load.

1.3 Proposed Solution

To quantify network parameters, a static reference installation will be created. A LoRaWAN base station and node will be installed at known elevations and distance apart. Test packets will be sent at regular intervals. Software in the node will vary key modulation parameters. The installation will be left in place for an extended period, allowing signal strength and packet loss to be recorded. The testbed will be used to measure the power consumption of the LoRaWAN node.

Drive surveys will be performed to measure coverage in an urban environment. GPS-enabled nodes will transmit test packets and location. Data will be used to train and calibrate a deterministic propagation model. Using the validated propagation model and an industry standard planning tool, candidate sites for gateways in Watford will be reviewed. Composite coverage figures will be used to estimate the best location for new gateways.

A computer simulation will be developed to investigate how a LoRaWAN network scales with increased load. This will provide an estimate of the number of nodes that can be served by a single gateway.

Background

In this section, we review network requirements, architectures, and standards for the IoT. We then position various candidate technologies and provide an overview of LoRa and LoRaWAN.

1.4 IoT Network Requirements

IoT applications differ from those using broadband services. Most IoT use-cases do not require either a high data throughput or persistent connections. Instead, nodes must operate under constraints such as limited battery and processing power. Key network characteristics are listed below:

- Bi-directional communication
- Scalability
- Security
- Mobility
- Energy efficiency
- Processing efficiency

Networking, power and security concerns are frequently cited as limiting factors in the rollout of the IoT. LPWANs have been proposed as a means of addressing these requirements.

1.5 Proposed Architectures

Various architectures exist for the IoT. Three and five-layer models are shown in Figure 2.

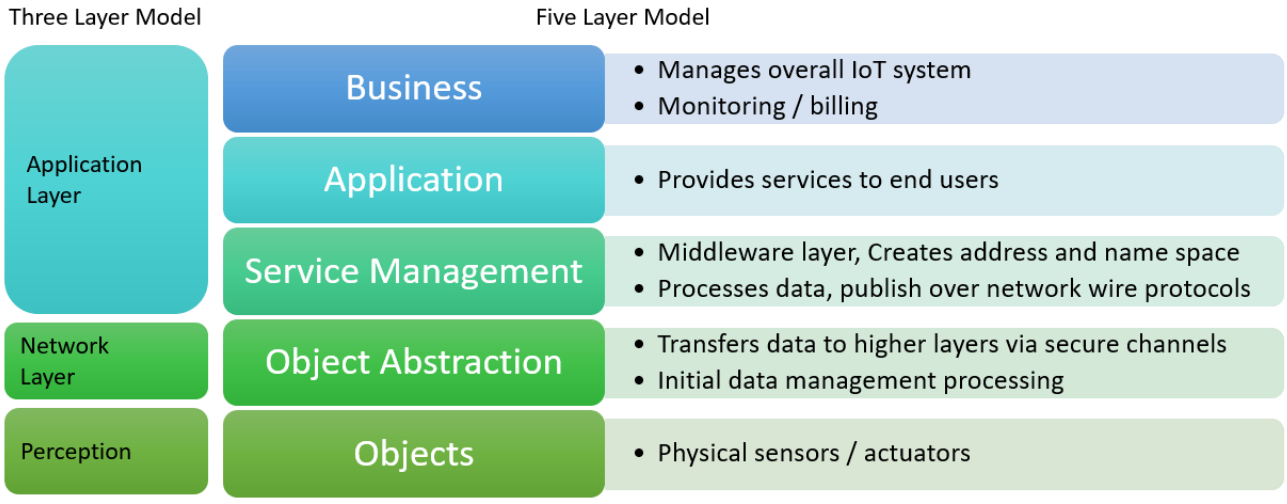


Figure 2: Reference architectures for the IoT [4].

LoRa operates in the object and object abstraction layers. LoRaWAN MAC provides functionality in the service management layer. Higher layer implementation is left to individual operators such as EveryNet and The Things Network.

1.6 Standards and Protocols

IoT standards and protocols are among the most rapidly developing and best funded in the industry [5].

Research is being carried out by several organisations, including the IETF and the IEEE. The ITU study group, SG20, has an active work programme aimed at improving the security, mobility, and scalability of IoT networks. It also has an interest in the creation of Smart Cities where LoRaWAN has a proven track record.

The LoRa Alliance is a non-profit organisation. It maintains the LoRaWAN specification and offers device certification programmes. Standards for LoRaWAN form part of a wider specification for LPWANs and Low-rate Wireless Personal Area Networks (LR-WPANs). The IEEE 802.15.4 specification describes Chirp Spread Spectrum Modulation similar to that used by LoRa. It also proposes mobility via seamless handover which is adopted by LoRaWAN.

In addition to using new protocols, IoT networks leverage industry standards such as CoAP, IPv6, and SLAAC (Stateless Address Auto-configuration). 6LoWPAN is commonly used to perform header compression.

1.7 Candidate Technologies

3GPP IoT standards, such as LTE-M and NB-LTE, have yet to be widely deployed. This leaves room for players other than the major telco operators. The current market leaders are LoRa, Zigbee, and Sigfox. Other technologies exist but are mainly targeted at niche applications. Commentators expect to see product coalescence in the coming years [6]. The main operators in the space are shown in Figure 3.

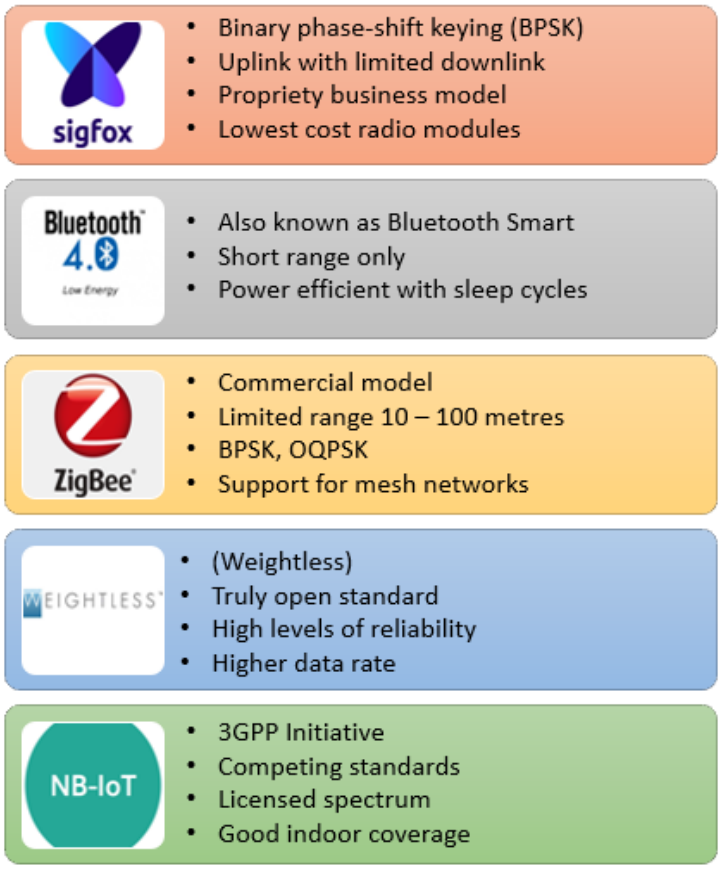


Figure 3: Description of candidate technologies.

IoT networking technologies can be positioned by considering engineering trade-offs present in their development. Engineering trade-offs relevant to the current study are shown in the figure below:

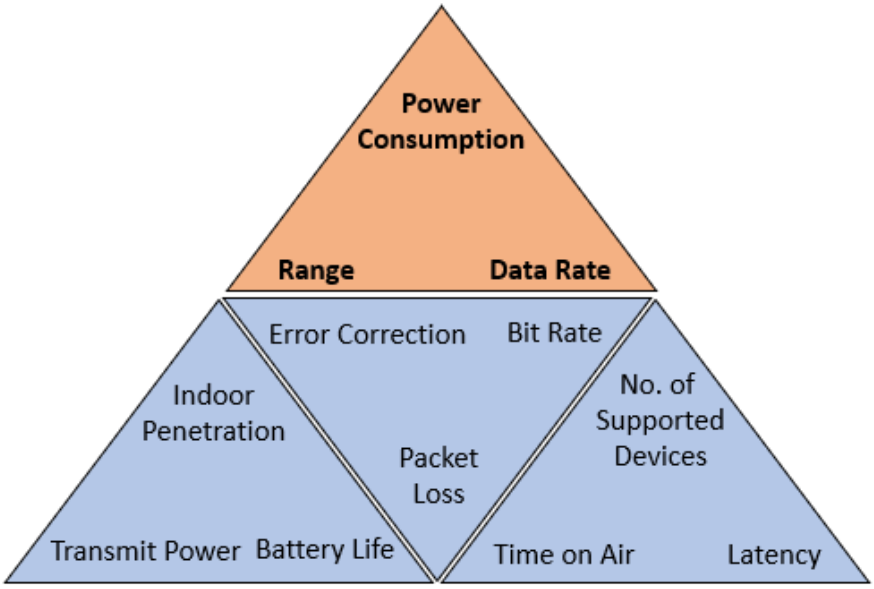


Figure 4: Trade-Offs in IoT networking protocols.

The primary engineering trade-off is between power consumption, range, and data rate. This arises as a direct consequence of the Shannon-Hartley Theorem. Figure 5 positions candidate technologies by considering bandwidth versus range:

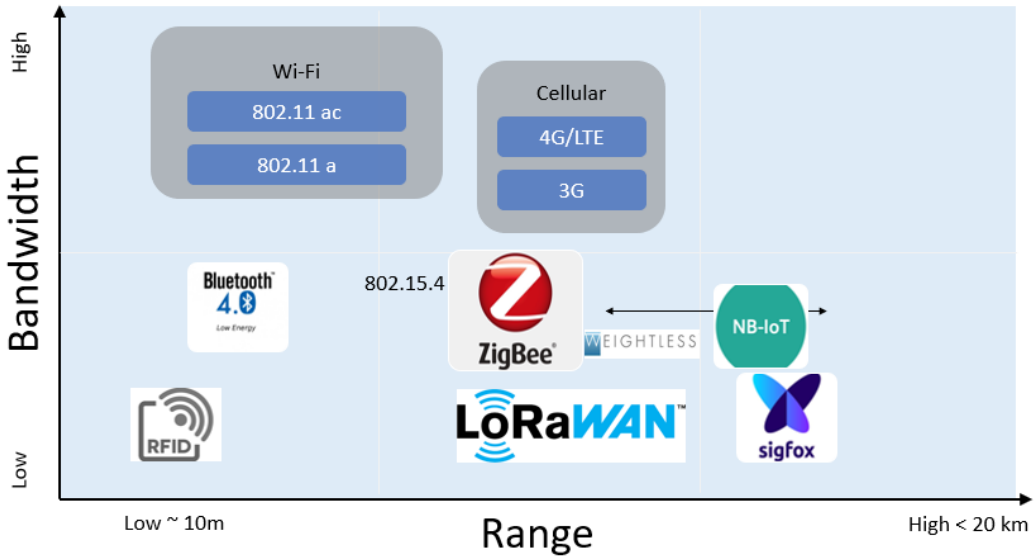


Figure 5: Positioning IoT network technologies.

Several devices on the market support multiple IoT protocols. This increases compatibility and allows a single device to leverage benefits from multiple technologies. One example of this is the Wink Hub. This supports a range of protocols, including Zigbee and Bluetooth LE. Support for multiple protocols can be achieved using a hardware or software approach. Hardware implementations use discrete

circuits. This tends to be a lower cost solution. Devices based on Software Defined Radios (SDRs) require significant processing power but are more flexible.

1.8 Applications of LoRaWAN

Examples of applications that use LoRaWAN are shown in Figure 6.

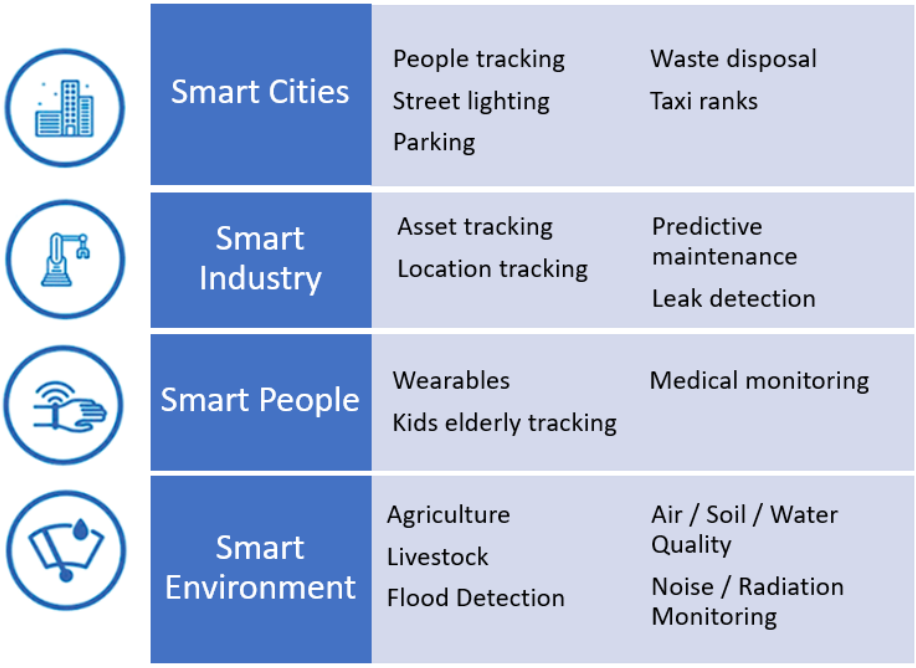


Figure 6: Current UK applications of LoRaWAN.

Most current applications are in the monitoring plane. High latency and packet loss have resulted in slow uptake in control-plane applications. Some sources suggest that the initial impact of LPWANs will be in industrial applications and that widespread adoption in domestic life will only happen when the technology is fully established. In addition to technical challenges, there are commercial factors that can limit a wider rollout. These include difficulties in commercialising the technology and in managing device obsolescence. Greatest benefits will only be realised with large-scale rollouts.

1.9 Overview of LoRa/LoRaWAN

LoRaWAN is a MAC and network layer communication protocol. LoRa is a proprietary physical layer technology. The term LoRa can be applied to both the physical and MAC layers, more accurately, it should only refer to the radio frequency modulation. The terms are disambiguated in Table 1.

LoRa	**LoRaWAN**
Wireless modulation technique	Communications protocol and architecture
Physical layer	Media Access Control Layer Network Layer
Analogous to Wi-Fi	Analogous to IP Protocol
Propriety to Semtech	Open Source (supported by the LoRa Alliance)

Table 1: LoRa vs LoRaWAN.

1.10 LoRaWAN Network Architecture

LoRaWAN supports various topologies. The most common is a *star of stars*. This is similar to a traditional cellular network. Key features are shown in Figure 7.

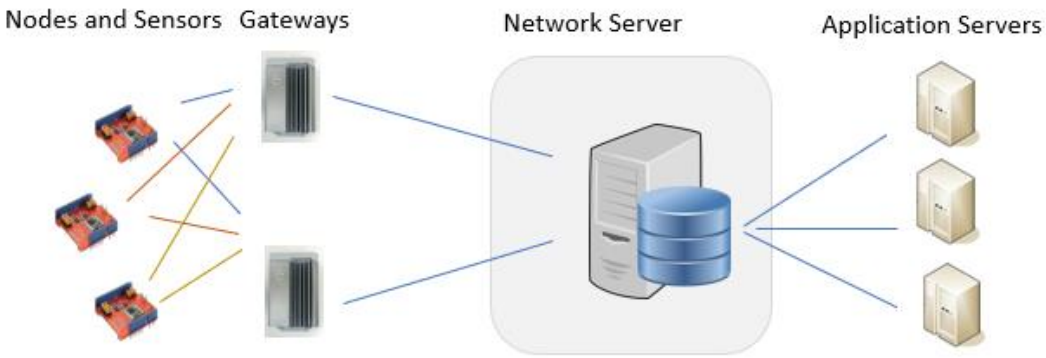


Figure 7: LoRaWAN network architecture.

Steps in an uplink connection are listed below:

- End-user devices (nodes) receive data from sensors.
- Packets of data are transmitted to base stations (gateways) using LoRa radio modulation.
- Gateways act as relays between nodes and one or more network servers.
- Network servers aggregate and process the data packets.
- Data is forwarded to an application server where it is consumed in applications.

One disadvantage of a star architecture is that gateways can be a single point of failure. This is mitigated in LoRaWAN networks because a single node is usually served by multiple gateways.

1.10.1 Note on use of Mesh Architectures

IoT technologies, such as Zigbee, use a mesh architecture. Routing between nodes can extend the range of the network and eliminate single points of failure. There are however several disadvantages. To relay data, devices must remain active. This increases power consumption. Additionally, a limited number of paths can lead to network congestion and latency. LoRaBLINK is one effort to apply a mesh architecture to LoRaWAN. The length of the LoRa preamble and use of variable bit rates tend to make operation over meshes inefficient [7].

1.11 Physical Layer

The LoRa physical layer acts as an interface between the radio channel and the MAC layer. It performs the following functions:

1. Construction of the LoRa frame for transmission over the RF link.
2. Insertion of a frame header, preamble, and CRC.
3. Determination of a channel for transmission.
4. Management of forward error correction.

Specifications for the uplink and downlink physical frames are shown in Figure 8.

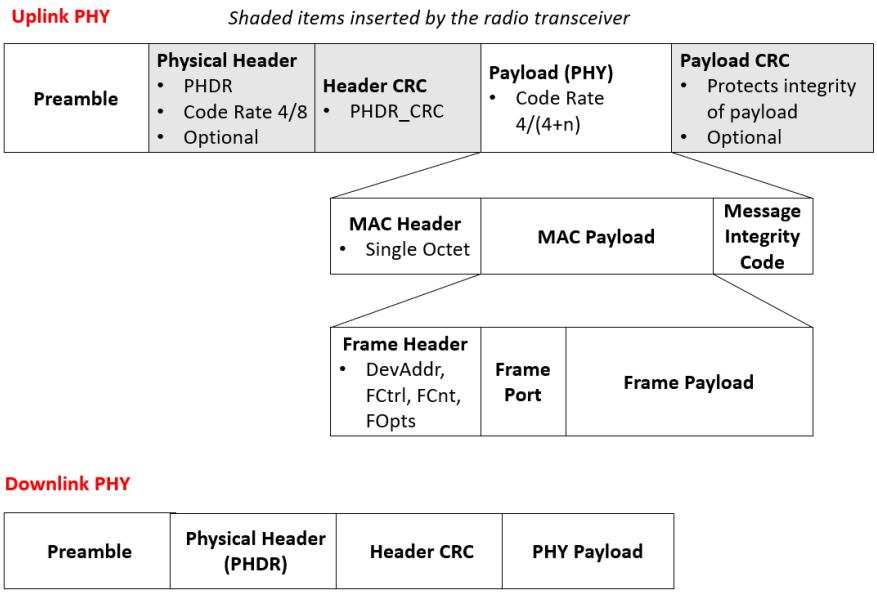


Figure 8: Uplink and downlink frame formats [8].

In any given frame, bandwidth and spreading factor remain constant. The preamble consists of a series of *up chirps*. These encode a sync word that is used to identify the LoRa network. Frame payload is encoded in Base64 with each character requiring six storage bits. Base64 is a common format for internet data transfer and is, for example, used in HTTP basic authentication. One advantage of using this over plain text is that it offers protection against un-escaped binary characters.

1.12 MAC Layer

The LoRaWAN MAC level detects changes in the physical layer. It then establishes a protocol for sending data. Specifically, it performs the following functions:

1. Manages MAC-layer services such as channel access and addressing.
2. Defines message formats for each device class.
3. Manages "Join Requests" from the end device to the server.
4. Manages "Join Accepts" from the server to the end device.
5. Manages beacon frames.
6. Manages message confirmation.

LoRaWAN media access is based on a one way 'Aloha' or half duplex protocol. Listen Before Talk (LBT) or Adaptive Frequency Agility (AFA) are not currently supported in LoRaWAN. Due to regulatory requirements, this limits the transmit power from a possible 500mW to 25mW. LoRa modulation can be combined with alternative MAC layer protocols. An example of this is Symphony Link by LinkLabs.

A robust multiple access protocol is important for LoRa as chirp modulation does not mitigate collisions well [9]. Studies show that the LoRaWAN MAC layer has an efficiency in the range 18 to 22% [10].

1.13 Modulation

A modulation scheme varies one or more waveform properties, for example, amplitude, frequency or phase. LoRa modulation is proprietary and was patented in 2014. It was developed by Cycleo, who were later acquired by Semtech. It is a derivative of Spread Spectrum Modulation (SSM). More precisely, it is a form of Chirp Spread Spectrum Modulation (CSSM). The LoRa specification states that devices close to a gateway will revert to Frequency Shift Keying (FSK). This was not seen in the current research.

1.13.1 Spread Spectrum Modulation

In spread spectrum modulation, a base signal is encoded using a high-frequency code sequence. This spreads the signal over a much wider bandwidth. The most common forms of spread spectrum modulation are frequency hopping and direct sequence. Spread spectrum modulation offers the following advantages:

- Bandwidth is traded for increased resistance to interference. This is a direct consequence of the Shannon-Hartley Theorem. Interference can be intentional or deliberate.
- Narrowband systems require significant power to overcome the noise floor. SSM devices can operate under the noise floor resulting in a lower peak power requirement.
- Resistance to fading is increased due to frequency diversity. If fading occurs only over part of the bandwidth, error correction may allow the original signal to be recovered. This is advantageous in an urban environment.
- Orthogonal spread spectrum techniques allow multiple access to the air channel.

One disadvantage is that the receiver must be aware of the coding scheme. This may require significant processing power. This can add to the cost of equipment and decrease the time that an IoT device can remain idle.

1.13.2 Chirp Spread Spectrum Modulation

CSSM was first developed in the 1940s. It has a proven track record in military and space applications. Its use by LPWANs was proposed in IEEE 802.15.4. LoRa is the first time the modulation has been applied to low-cost commercial devices [11]. In common with other spread spectrum techniques, a signal is encoded across the full channel bandwidth. Modulation is achieved using frequency patterns called chirps. CHIRP is an acronym for Compressed High Intensity Radar Pulse. They have a constant amplitude with a linear variation of frequency with time. It is the unique nature of the frequency shift that allows it to be distinguished from the channel noise. The terms up chirp and down chirp are used to distinguish frequency increases and decreases respectively. A typical time plot is shown in Figure 9.

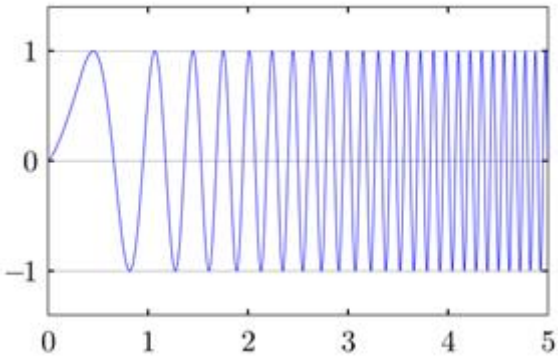


Figure 9: Up-Chirp (Linear) [12].

Chirps can be generated using fractional-N phase-locked loops (PLLs). These are relatively simple devices which help to keep device costs low [13]. CSSM has good autocorrelation properties. A frequency offset can be equated to a time difference. This simplifies decoding and increases immunity to the Doppler Shift. A spectrogram can be used to visualise LoRa signals in the frequency domain. An example is shown in Figure 10. The intensity of the line is proportional to the strength of the signal.

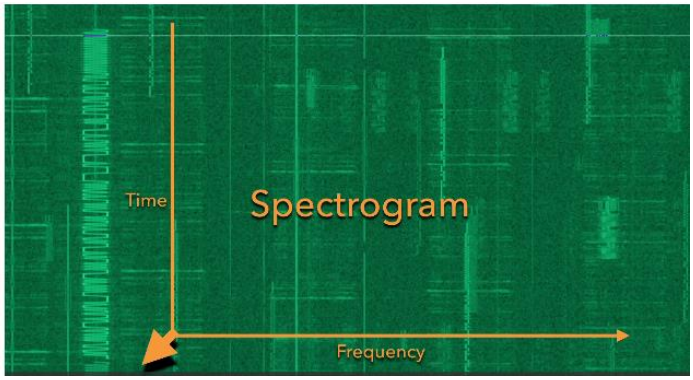


Figure 10: LoRa modulation in the frequency domain [14].

Figure 11 illustrates how chirp modulation cancels out interference. A LoRa signal and narrowband interference are decoded by applying an inverse set of chirps. The LoRa signal can be easily filtered from the interference.

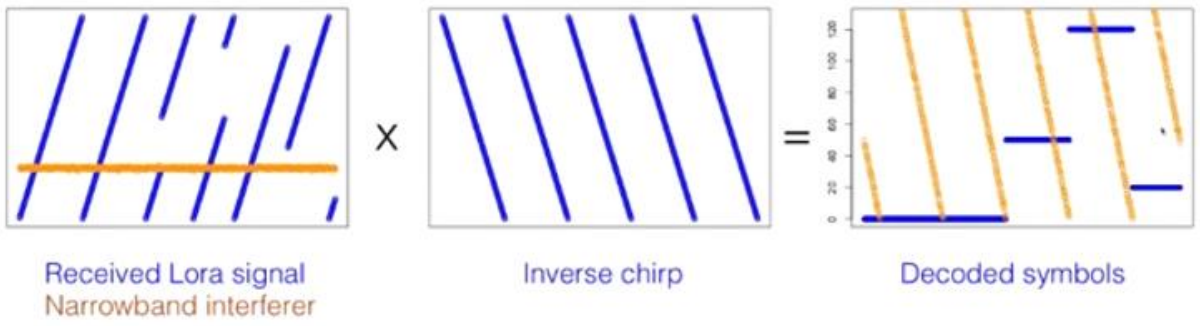


Figure 11: Narrowband interference [15].

1.14 Modulation Parameters

Modulation parameters allow optimisation for specific applications. Key parameters are described in the table below:

Parameter	Details
Frequency	Although LoRa modulation is frequency independent, most LoRa devices operate in the license-exempt ISM bands. In the UK, this corresponds to frequencies between 868 to 870 MHz. License-exempt does not mean that operation is unregulated. Limits exist on duty cycle, transmission power, and bandwidth. The Ofcom document IR2030 outlines the regulations in the 863-870 MHz band. A significant disadvantage of operating in license-exempt bands is the possibility of interference from other users.
Bandwidth	Most LoRa chipsets allow the selection of different bandwidths. All measurements in this study were done with a channel bandwidth of 125 kHz.
Spreading Factor	Spreading factor is a measure of the modulation rate. It adjusts the trade-off between bit rate and receiver sensitivity. Packets sent with a higher spreading factor have a longer transmit time but are likely to be received at greater distances. The LoRa spreading factor is formally defined as the logarithm, in base 2, of the number of chirps per symbol [16].
Low Data Rate Optimisation	This is mandatory for SF 11 and 12. Its behaviour is not well documented but when enabled reduces the bits transmitted per symbol.
Coding Rate	Determines the amount of forward error correction. In most devices, it is set to 4/5.
Adaptive Data Rate	The Lora specification describes an Adaptive Data Rate Mechanism. Packets are initially sent using SF 12. A receiving gateway can inspect the received packet's SNR value, add a margin and determine if retransmission should occur at a lower spreading factor.
Duty Cycle	Duty cycle restrictions in the ISM bands apply to a given frequency channel. There is not a one to one mapping between a LoRaWAN channel and a frequency channel. For example, if a duty cycle restriction of 1% applies to a frequency channel and three LoRaWAN channels occupy this, the duty restriction for each LoRa channel is 0.3% [17]. Duty cycle restrictions are enforced by some devices and networks. In the UK ISM bands, LoraWAN devices are limited to approximately 30 seconds per hour. In practice, duty cycles are much lower. For example, The Things Network limits transmission time to 30 seconds per day.

Table 2: LoRa modulation parameters.

1.15 Link Budget

In this section, we consider the link budget of a LoRa transmission. This will enable us to relate transmission power, receiver sensitivity, and range. Figure 13 accounts for all propagation losses and gains from the transmitter to a receiver. Receiver sensitivity is plotted as a horizontal line at the bottom of the graph. A logarithmic scale is used.

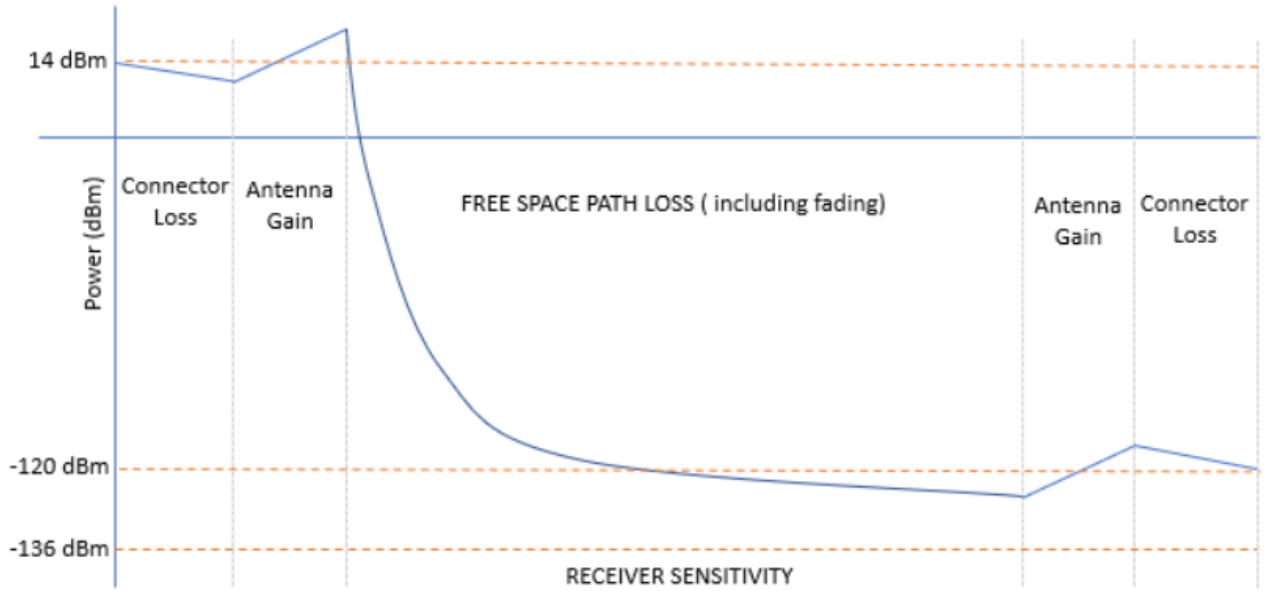


Figure 12: LoRa link budget [15].

The Effective Isotropic Radiated Power (EiRP) of the node is given by the following equation:

$$EiRP = T_x\, Power + Antenna\ Gain - Cable\ Loss$$

Due to regulations in the ISM bands EiRP is limited to 20 dBm. The link budget can be expressed as:

$$Path\ Loss\ (Link\ Budget) = EiRP - RSSI\ (Receiver\ Sensitivity)$$

The maximum sensitivity of a node-based receiver is stated as -136 dBm. This gives an overall link budget of 156 dB. Free space path loss is given by the Friis transmission equation:

$$Path\ Loss\ (dB) = -10\ log\ \frac{G_t G_r \lambda^2}{(4\pi)^2 r^2}$$

Substituting values for LoRa give a theoretical free space range of 300 km.

1.16 Nodes

Nodes operate in half-duplex mode. A switch is needed to ensure the receiver is closed during transmit. This prevents damage to the transceiver. They are categorised as Class A, B or C devices. Different classes allow optimisations of network variables. Specifically, applications can make different trade-offs of power and latency. All classes are bidirectional and unicast. In all cases, it is the end device that initiates communication. Classes differ only with regards to the downlink scheduling [16]. To achieve certification all devices must be capable of operating in Class A mode.

Figure 13 summarises the operation of the different classes:

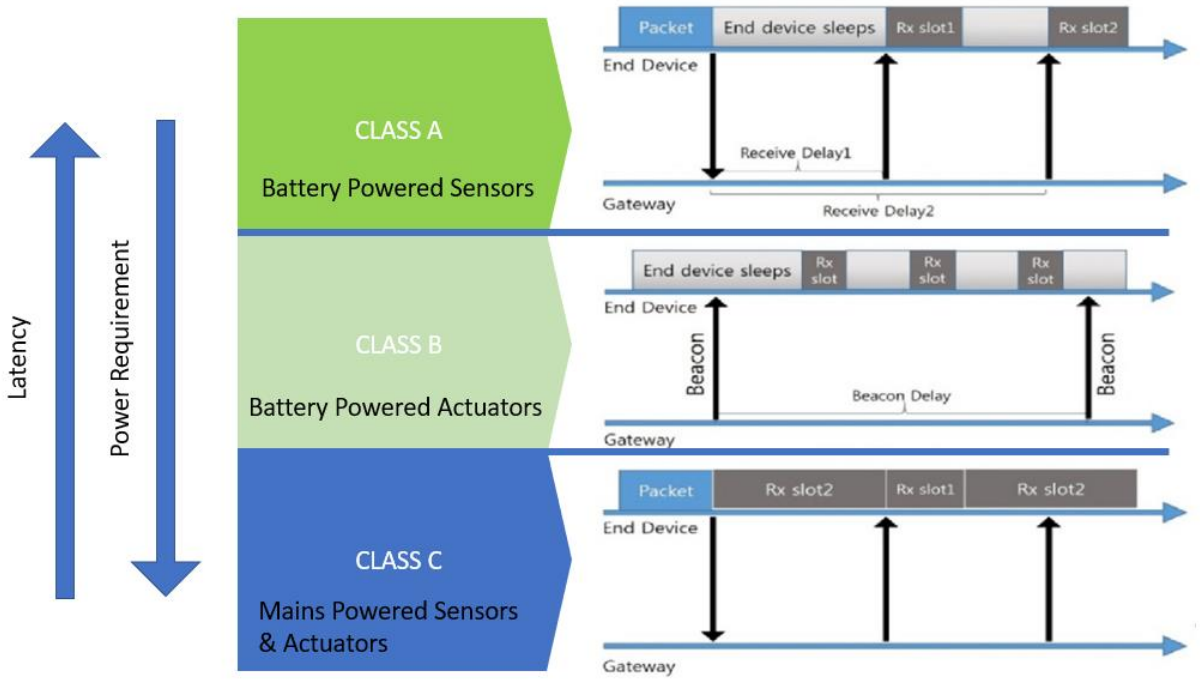


Figure 13: LoRaWAN device classes [18].

Many LoRaWAN applications require operation away from mains electricity. Novel power solutions have been suggested. Most of these use energy harvesting [19]. Two examples are given below:

- Motion Energy from mechanical, human or animal movement. For example, compression of pavements [20].
- Indoor solar power using ambient fluorescent lighting [21].

Despite these techniques, most devices will be battery powered. It is claimed that LoRaWAN nodes can operate between one to six years on a 2400 mAh battery [22].

1.17 Gateways

Gateways are single or multi-modem devices. They relay frames bi-directionally from a LoRa node to a network server. Gateways can be bought prefabricated or in kit form. Software-defined radios can also be configured as gateways. Gateway technology is continuing to improve. Semtech has recently announced a new reference design for LoRa gateways. The main objective is to implement geolocation without using GPS. Digital Signal Processing (DSP) techniques along with the Time Difference of Arrival (TdoA) and RSSI is used to estimate position. Cisco has implemented the new design in their 16 channel LoRaWAN gateway.

1.18 Network Providers

According to the LoRa Alliance, there are over 80 LoRaWAN network operators globally. In Table 3 a selection of the major network providers is listed.

Network Provider	Detail
The Things Network	This is a crowdsourced provider although a commercial model with SLAs is planned. Currently hosts around 1,500 gateways.
EveryNet	Backend provider for the Things Connected. Commercial business model. Things Connected have recently announced that they will support both EveryNet and The Things Network.
Senet	The largest LoRaWAN network in North America. Commercial business model.
Loriot	Fast growing, commercial LoRaWAN network. The business model is based on a capacity pricing model (per gateway or device).
Stream	LoRaWAN network acquired by Arm. Commercial model with 777,000 connected devices [23].

Table 3: LoRaWAN network providers.

There is also an open source LoRaWAN network provider [24]. This is available as a Docker container.

1.19 Limitations of LoRa

Several limitations of LoRa / LoRaWAN have been cited these are detailed below:

Limitation	Detail
Spectrum usage	Heavy use with 125 kHz per channel.
Reliability	It has been stated that LoRa is only suitable for applications that can tolerate up to an 80% packet loss. This limits potential use cases and makes it an unlikely choice for business-critical or control plane applications.
Latency	May not be well suited for critical machine to machine communication which requires both low latency and high reliability.
Need for standardisation	Standards bodies such as ETSI along with the LoRa alliance and special interest groups are actively working on standards.
Lack of data model	LoRaWAN lacks a unified data model with frame data being passed as bytes. IoT technology makes use of JSON or other data formats. No support for packetisation.
Limited MAC protocols	Listen Before Talk needed for greater EU duty cycle restrictions. No OTA transmission of security patches or firmware. There is no support for roaming and multicast.

Table 4: Limitations of LoRaWAN.

Some of the limitations can be addressed by good design. For example, some applications mitigate packet loss by using the principle of local autonomy. One example is in the monitoring and control of emergency lighting. LoRa has been deployed to collect metrics and perform routine system tests. Critical safety decisions are delegated to local sensors and processing.

Solution Architecture

This chapter details the hardware and software used to perform measurements and process LoRaWAN test data. Project methodology is outlined in Figure 14.

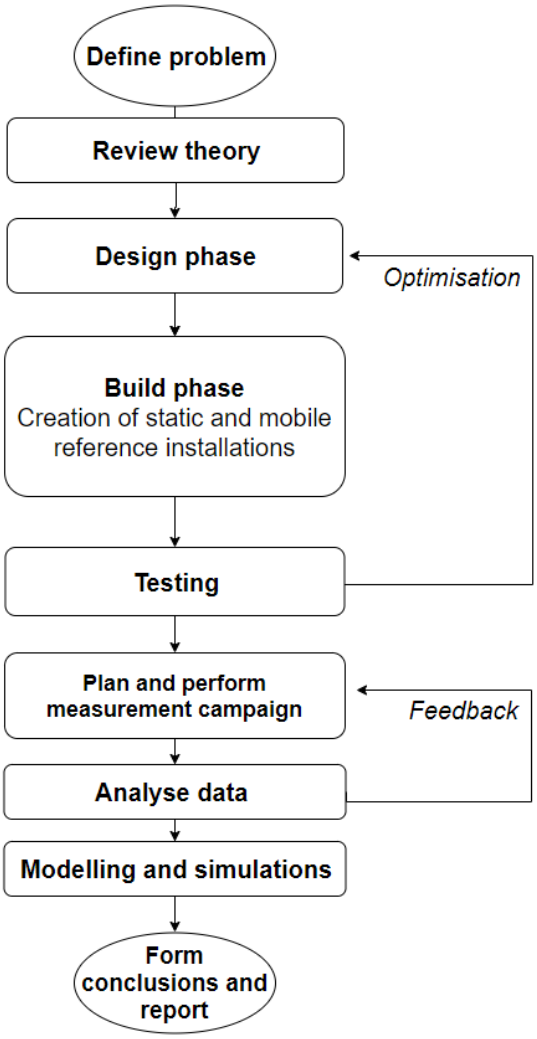


Figure 14: Project Methodology.

Testing formed an integral part of the project. Trials on the static and mobile testbeds resulted in modifications to the design. Early analysis of field data resulted in changes to the way the measurement campaign was performed.

1.20 Overview of Solution

A high-level plan of the solution is given in Figure 15. This project only considers uplink connections. Data flow is from left to right in the diagram. Transmission of LoRa packets is initiated by end-user devices. Frames are received by LoRaWAN gateways and relayed to third-party network servers. De-duplicated data is then transferred to an application server hosted for this research. Data is exposed using a RESTful web service and presented in a web-based GUI and Android application. Finally, data is exported to an offline database where it is analysed and used as a dataset for propagation modelling.

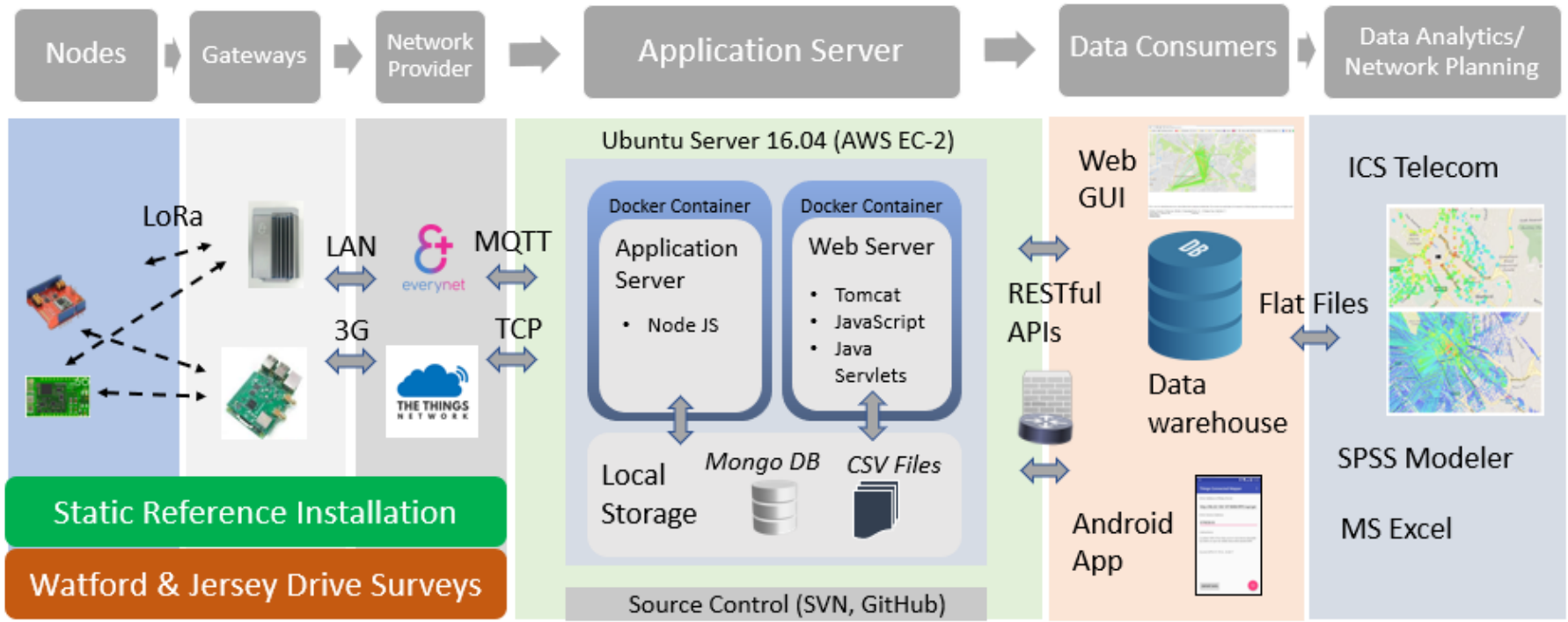


Figure 15: Solution Architecture.

The following sections detail the creation of the static and mobile reference installations.

1.21 Set up of LoRa Test Nodes

Two different LoRa nodes were used in the measurement campaign. These are discussed below:

1.21.1 Arduino Mega 2560 with Dragino Shield

The first node was based on an Arduino Mega 2560 platform with an external LoRa shield [25]. This is shown in Figure 16. A *RoyalTek GPS Module and Data Logger (REB-4216)* shield was used to provide location services. The Dragino shield is based on the SX1276 chip. This chip supports LoRa modulation. However, unlike the RN2483 chip, it does not contain an implementation of the LoRaWAN stack. It was, therefore, necessary to implement LoRaWAN in Arduino based software. This was achieved using the LoRaMAC-in-C (LMIC) Library developed by IBM. The library is 31KB in size. This limits its use on devices such as the Arduino Uno which only has 32KB of available memory. To communicate with the GPS module, the *TinyGPS* library was used.

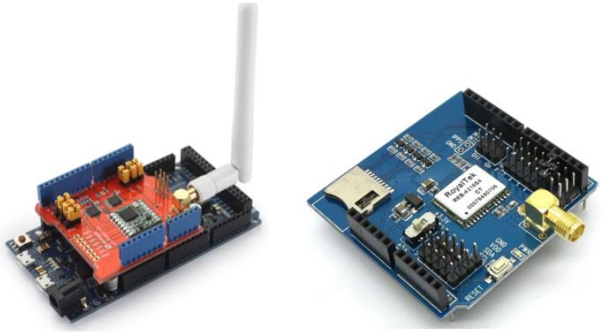


Figure 16: Arduino Mega 2560 with Dragino Shield (v1.3) (Left) External GPS Module and Data Logger (Right).

Date and time are required for scheduling test packets. In situations where the location of the node is known, an external real-time clock (RTC) was used instead of a GPS board. This is shown in Figure 17. It was found to significantly reduce power consumption compared to acquiring time from the GPS receiver.

Figure 17: External Real-time Clock (DS3231).

Bespoke software was developed to operate the test node. Features were designed to meet objectives of the measurement campaigns. The code was written in C using the standard Arduino IDE. An Apache Subversion (SVN) repository was used for backup and version control. The final version of the code was published to GitHub. Software features are listed in Table 5.

Functionality	Description
Packets sent with different spreading factor	To investigate the effect of varying the spreading factor, test packets are first transmitted at SF7. After a set interval (default five minutes) the spreading factor is incremented. When the spreading factor reaches SF12 it reverts to SF7 and the cycle repeats.
Scheduling	To schedule packets date and time are read from the GPS module or an external Real Time Clock (RTC).
GPS coordinates sent in the payload	GPS coordinates are read from the GPS module and transmitted in the payload. To ensure a fair comparison payload length is kept constant for all test transmissions.
Hibernation	To conserve power, the device is set to hibernate outside working hours (16:00 to 04:00).
Visual transmit indicator	Code was written to light an LED when a packet is transmitted. This was found useful for troubleshooting.

Table 5: Key features of software on the test nodes.

1.21.2 Sodaq One

The second node tested was the Sodaq One. This was previously marketed under the name LoRaOne. It is based on an Arduino M0 platform and contains an onboard GPS receiver. The microcontroller has a flash memory of 256KB and is powered by a 5V micro-USB port. The node uses the RN2483 chip from Microchip. This contains both the LoRa physical layer and the LoRaWAN protocol (Class A). The node is supplied with a short wire antenna. This connects to an onboard U.FL port. An SMA adapter was used to connect the board to other antennas. Software developed for the Arduino Mega node was ported to the Sodaq One.

Figure 18: Sodaq One (v2) LoRa node.

For location services, the node uses an EVA-8M GPS chip from U-Blox. Published figures state an acquisition time, from a cold start, of 26 seconds. Reacquisition times are stated as 1 second. This broadly concurs with acquisition times seen in practice. Low power operation is key. GPS connections represent a significant proportion of the overall power drain. U-Blox state power consumption as 22 mA if used continuously, reducing to 5.3 mA in power saving mode. Sodaq specialises in selling solar-powered devices. The Sodaq One comes with JST headers. These allow connection to a solar panel and LiPo battery. To get consistent results, it was found necessary to update the firmware to the latest version. The Sodaq One has an onboard accelerometer. This can be used as a hardware interrupt to wake the node when movement is sensed. In future studies, it could also be used to exclude measurements taken when a test vehicle is in motion.

1.21.3 Node Antennas

Three different node antennas were considered, these are shown in Table 6. The ANT-900MR was selected as it was found to be the most robust. It also has a right-angled SMA connector which helps keep the antenna vertical. It is 10 cm long, making it a quarter-wave monopole. No ground plane or counterpoise was used as it was assumed that the roof of the recycling vehicle would act as an informal ground plane.

To improve GPS reception, an active GPS antenna was used with the Arduino node. The Sodaq One has a built-in ceramic antenna.

Antenna Name	**Image**	**Notes**
Anaren 868 MHz antenna 66089-0830		Type: omnidirectional, short wire antenna Length: 3 cm Gain: 3 dBi at 868 MHz VSWR: 1.7:1 at 868 MHz
Molex 868/915		Type: omnidirectional, patch (microstrip) Length: 7.9 cm (feeder 10 cn) Gain: 3 dBi at 868 MHz VSWR: 1.6 at 868 MHz
ANT-900MR (EasyRadio)		Type: omnidirectional, whip Length: 10.8 cm Gain: 3 dBi at 868 MHz VSWR: ≤ 1.5 at 868 MHz

Table 6: Node antennas.

1.21.4 Comparing the Nodes

Prior to being used in production, the nodes were tested. Both were found to be stable with similar performance characteristics. Packets transmitted were received by gateways 10 km away.

The Arduino based node requires an implementation of LoRaWAN in software. This reduces available RAM but does allow full control over parameters. The disadvantage is that the required libraries are open source. Changes to the LoRaWAN specification may render the node inoperable. The Sodaq One uses the more standard RN2483 chipset. Firmware updates and end-user support are widely available. The Sodaq One is more robust and compact and has an onboard GPS receiver. This makes it more suitable for taking mobile measurements. The Arduino was used in the static testbed.

1.22 Gateway Setup

Two different types of LoRaWAN gateway were used in this research project. These are described in the next sections:

1.22.1 Raspberry Pi and Development Board (Static Reference Installation)

The first gateway to be tested was constructed using a Raspberry Pi and a LoRaWAN development board. The development board was supplied by LinkLabs and is shown in Figure 19. The board is based on the SX1301 chip and has an onboard GPS receiver (Linx R4). Connection to an external antenna is via an SMA connection.

Figure 19: Link Labs development board.

The gateway was installed and configured using code available online [26] [27]. The main software to operate the gateway uses the *libloragw* library supplied by Semtech. This is an API which interfaces with the transceiver and forwards packets to a network server. The configuration of the gateway is done in JSON. An example configuration, showing connection information, is given below:

```
  "gateway_conf": {
        "gateway_ID": "b827ebFFFE",    // MAC ID
        "server_address": "staging.thethings.network",
        "serv_port_up": 1700,
        "serv_port_down": 1700
}
```

The server was connected to The Things Network [28]. Legacy versions of the gateway software use a *poly-packet-forwarder.* This is a background service that relays packets to a local network server and a central administration server, currently located in the Netherlands. A version, without the poly-packet-forwarder, is now available. To connect to the internet, the gateway was hosted inside a DMZ. This ensured full access to the required ports and support for UDP. The advantage of using a development board rather than a commercial gateway is that all configuration options can be modified and there is full access to server logs.

As the gateway was to be left in one place for an extended period, it was necessary to protect it using a suitable case. Two commercial enclosures were tested. These are shown below:

Figure 20 Commercial cases (a) for a Raspberry Pi 3 and (b) for a Power Supply Unit (PSU).

The Raspberry Pi 3 case has a detachable lid and is designed to accommodate an external module. Unfortunately, the development board generates a significant amount of heat and more ventilation was required. The second case was designed for a Power Supply Unit. It has good ventilation, is flame retardant and has holes in the front and back suitable for passing power and antenna cables through. This was selected for use with the LinkLabs gateway.

The gateway was connected to an external collinear antenna and mounted in the author's loft. A hole was made in the ceiling of a first-floor bedroom to pass the antenna cable through. This arrangement is shown in Figure 21:

Figure 21 LinkLabs Gateway as installed.

The setup performed well although several problems were experienced. The operating system of the Raspberry Pi is hosted on an SD card. Data is being continually written to the card. Power outages can result in corruption of data. To mitigate against this the SD card was imaged after installation. This allowed for rapid recovery in the event of an outage. During the period of operation, various upgrades were made to The Things Network. This required changes to both the gateway software and configuration settings. Version control software (SVN and Git) was used to track changes to the software.

1.22.2 EveryNet Gateway

The second gateway tested was the *EveryNet Network Gateway v2.0* [29]. The gateway costs £700 and has integrated GPS and 3G. It has 49 LoRa channels with a maximum sensitivity of -141dB. The

gateway is waterproof (IP67) and designed for mounting outdoors. Power over Ethernet (PoE) is supported to reduce the number of cables required.

Figure 22: EveryNet gateway.

A test gateway was supplied on loan from the Digital Catapult. This was mounted in the author's loft in St Albans (Figure 21). The gateway was connected to The Things Connected network. This gateway, together with others hosted by Watford Borough Council, was used in the drive surveys.

The EveryNet gateway performed well and was found to be more stable than the Raspberry Pi platform. An internal battery provides backup power for eight hours. The network server can be configured to send email alerts in the event of an outage. On a couple of occasions, the gateway lost connection to the network. A hardware reset was performed by connecting a jumper cable to pins on the bottom of the device.

1.23 Gateway Antennas

Three different types of gateway antenna were tested. These are described in the table below:

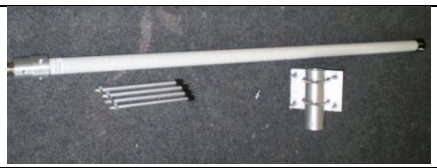
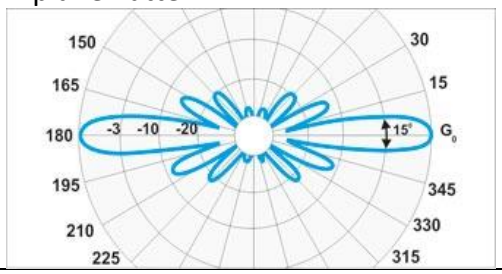
Antenna Name	Image	Notes
Cellbase2 (from Badlands Antennas)		Type: Whip antenna (half wave) Length: 17 cm Gain: 3 dbi VSWR: 1.5:1 (at 868 MHz)
Vinnant COL868/5-S		Type: Collinear Length: 70 cm Gain: 5 dbi VSWR: 1.2:1 (at 868 MHz)
Duplexers A10-868 [30]		Type: Collinear Length: 157 cm Gain: 10 dbi VSWR: < 1.3:1 (at 868 MHz) E-plane Pattern: 

Table 7: Gateway Antennas used in the Study

Antenna parameters are important in predicting signal strength. Key metrics are gain, directivity and frequency dependence [31]. These are discussed in the next sections.

1.23.1 Gain and Directivity

Antennas are passive devices therefore gain is a result of directivity. All antennas used in the study were omnidirectional. This results in a doughnut-shaped radiation pattern as shown below:

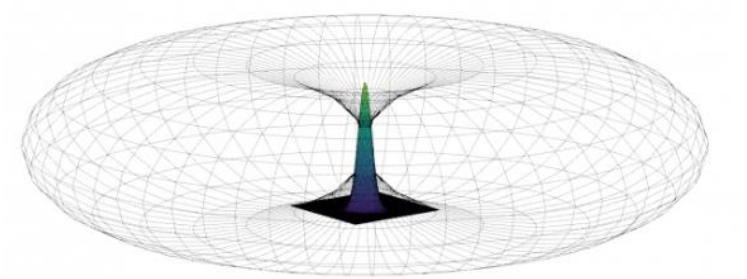


Figure 23: Radiation pattern from an omnidirectional antenna [31].

In future studies, the actual antenna patterns could be determined using an anechoic chamber. In many cases, published specifications will differ from performance in the field [32].

Accurate determination of gain is important. Regulations restrict the output power to a maximum EiRP of 14 dBm. To ensure compliance, many gateways support a parameter of *Transmit Look Up Table* (TX_LUT). This provides a mapping between requested power and the actual power used. For example, if a network server requests transmission at 14 dBm this could be mapped to 10 dBm.

1.23.2 Resonance (VSWR)

868 MHz corresponds to a wavelength of 34 cm. Typically, node antennas will be either a half or quarter wave dipole. Gateway antennas are several multiples longer. Voltage Standing Wave Ratio (VSWR) is a measure of the match between an antenna and the source impedance [32]. If VSWR is too high, typically greater than 3:1, damage can be caused to the transceiver. In practice, LoRa operates at 25 mW so damage to radio components is unlikely. Reducing the VSWR increases the efficiency of the antenna system.

As part of this study, a VNWA3 [33] Vector Network Analyser (VNA) was used to characterise a short collinear antenna. Results are shown in Figure 24:

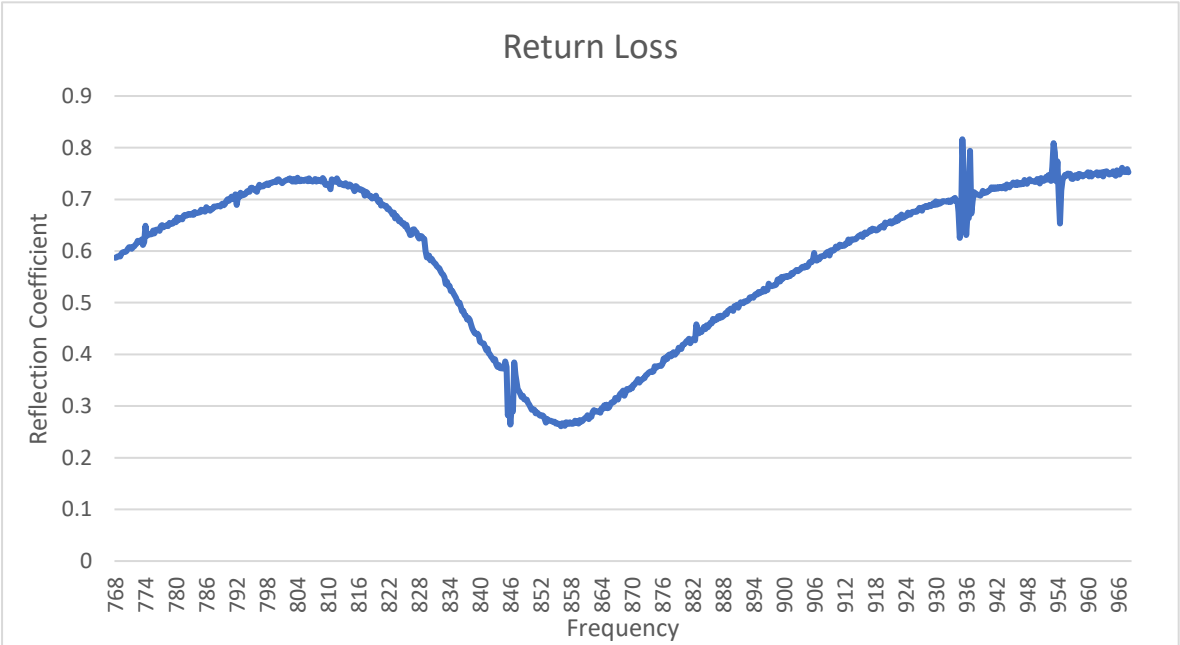


Figure 24: Return loss for a short collinear antenna.

From the return loss chart, the collinear antenna is resonant at 868 MHz with a reflection coefficient (r) of 0.32. VSWR and return loss is given by the following equations:

$$VSWR = \frac{1 + |r|}{1 - |r|}$$

$$Return\ Loss = -20\ Log_{10} r$$

This equates to a VSWR of 1.94:1 and a return loss of 9.90 dB. In practice, this means about 10% of the transmitted power will be reflected. This is an acceptable VSWR, however, it is higher than the specified figure of 1.2:1.

1.23.3 Ground Planes

For maximum antenna efficiency, a ground plane or counterpoise is recommended. A perfect ground acts as a half dipole. It reflects signals up that would otherwise be absorbed by the ground. No ground plane was used for the Cellbase2 antenna. The collinear antennas were supplied with three quarter wave radials which form a virtual ground plane.

1.23.4 Feeder Cables and Connectors

A feeder cable introduces additional losses to an antenna system. In the static installation, a 10 metre RG58 coaxial cable was used to connect the gateway to the antenna. At 868 MHz, this cable introduces a loss of 6.8 dB. One advantage of the EveryNet gateway is that it supports Power over Ethernet. This makes it easier to co-site the gateway with the antenna. Adapters where required to convert from an N-type input at the antenna to an SMA port at the gateway. Each connector was assumed to incur a loss of 0.2 dB.

1.24 Application Layer and Data Processing

The next sections detail how test packets were downloaded and presented. LoRaWAN does not define an application or presentation layer specification. Implementation is left to network operators. Figure 25 shows common approaches to retrieving data:

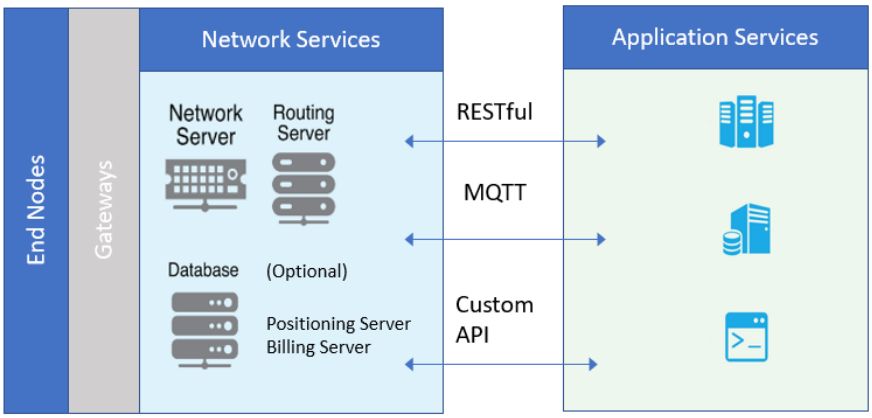


Figure 25: Higher layer architecture [34]

Mechanisms employed in this study are described below:

EveryNet / Things Connected – The current version of this platform only supports PUSH operations. Data is processed by the network servers and relayed to a nominated application server.

The Things Network – The Things Network supports two methods for accessing payload data. The first method uses MQTT. This is a lightweight publish and subscribe protocol. Messages are data-agnostic and can be formatted as binary, plain text or JSON. The Things Network publishes MQTT clients for Go, Java. Node-RED and Node.js. In the second method, data is stored in a cloud-based database. Clients query this by making RESTful API calls.

1.24.1 Hosting an Application Server

Both local and cloud-based Linux servers were set up. Features of these are listed in Table 8.

Feature	Description
Operating System	(Local) Raspbian Jessie. (AWS-EC2) Ubuntu Server 14.04
JavaScript run-time environment	Node.js
Docker Containers	Docker containers were used to facilitate the development and deployment.
Source Control	SVN (VisualSVN), Git/GitHub.
Forever library	This was used to run the node.js script as a background service
Port Forwarding (local server)	Port translation was used to expose a local port of 9090 running on the local server to the internet. Various tunnelling libraries were also investigated for this purpose. These proved difficult to manage and secure.
Static IP	Dynamic DNS services (for example, FreeDNS) were used to create a "virtual" static IP address. AWS provides long lease forward facing IP addresses.
Logging	Logging to the console (standard out) and text files facilitated troubleshooting and monitoring of the performance of the server

Table 8: Features of the application servers.

The servers performed well. AWS is an example of Platform as a Service (PaaS). This has the advantage of offering Service Level Agreements (SLA). These provide guarantees on availability and performance. In any IoT application, security is a primary concern. Access logs from the locally hosted server show malicious access attempts. AWS uses robust industry-standard security.

1.24.2 Docker Containers

Docker containers were used to package and deploy the software. Docker allows the creation of isolated environments. This means that code can be developed on one platform and then deployed with all environmental configurations to another. Server software was developed inside a container running on a Microsoft Windows PC and then transferred to AWS.

1.24.3 Server Software

To receive data from The Things Connected network, a Node.js application was written. This was based on sample code supplied by EveryNet [35]. The server listens for incoming packets on port 9090. An example of a received packet is shown in Figure 26.

Figure 26: Example of a packet received by the application server.

Node.js was also used to subscribe to the MQTT service published by The Things Network. The functionality of the Node.js applications is listed in the following table:

Feature	**Description**
Addition of GPS Coordinates	GPS coordinates sent in the payload is parsed and written to a data record. To parse the data, it is necessary to create a buffer and decode from basa64 as per the line below: `var buffer = new Buffer(args.payload, 'base64');`
Calculation of Distance	The *geolib* library was used to calculate distance between the node and a receiving gateway: `dist = geolib.getDistance({latitude: coords[0], longitude: coords[1]},{latitude: gw_latitude , longitude: gw_longitude });`
Export to different data repositories	Node.js code was modified to support export to MongoDB. This is a document database well suited to semi-structured data such as JSON. The *CSV-WRITE-STREAM* NPM package was used to output data to local CSV files.

Table 9: Features of the application server and TTN client.

1.24.4 RESTful Webservices

Data frames are forwarded from EveryNet and stored in MongoDB or CSV files. A RESTful web service (RWS) was used to publish the data to the internet. The RWS server was developed in Java 8 and deployed using Apache Tomcat. An example API is given in Table 10. A complete set of specifications is available in the Appendices.

Type	Description
Example URL:	http://localhost/RPC/api/frames?TimeInMins=60&SF=0
Description	Return a filtered array of JSON objects representing LoRa data frames stored in the data source
HTTP Method	GET
Parameters	• *TimeInMins* – the amount of history to return in minutes • *SF* – the spreading factor used to filter the data (0 returns all)
Response Status	200 (OK)
Response Content Type	application/json

Table 10: Example API specification.

Data is JSON formatted. A complete LoRa data frame is shown in the figure below:


```
{
  "time":"Tue Oct 03 2017 15:45:29 GMT+0000 (UTC)",
  "node_lat":"51.7311035",
  "node_long":"-0.3623302",
  "datr":"SF8BW125",
  "gw_lat":"51.73113",
  "gw_lon":"-0.36212",
  "payload":"\"51.7311035",
  "distance":"5733526",
  "net_type":"",
  "gw_alt":"113",
  "gw_ant_type":"",
  "gw_location":"",
  "gw_comments":"",
  "node_alt":"0",
  "node_type":"",
  "node_location":"",
  "node_comments":"",
  "rssi":"-43",
  "lsnr":"12.8",
  "counter_up":"484",
  "gw_addr":"70b3d54b114b0000",
  "dev_addr":"b22e6e0f",
  "stat":"1",
  "gw_band":"EU863-870",
  "server_time":"1507045529",
  "modu":"LORA",
  "chan":"4",
  "gateway_time":"1507045529",
  "tmst":"3385841684",
  "codr":"04-May",
  "rfch":"1",
  "freq":"868.1",
  "dev_eui":"c22b79cac5500e85",
  "rx_time":"1507045529",
  "endoffile":"-0.3623302\"",
  "id":"",
  "size":"34",
  "port":"1"
}
```


Figure 27: LoRa frame in JSON Format.

1.24.5 Security Considerations

Security is a critical factor for IoT software. To secure the web service, an API key is required for a successful connection. This is passed as a query parameter in the URL. An example of this is shown below:

```
/RPC/api/frames_secure?timeinmins=3000&sf=0&key=c225d602-###-dd4d6ea579c8
```

If the key is not supplied or is incorrect an HTTP status code of "403 Forbidden" is returned to the client.

1.24.6 Web Application

A bespoke web application was created to present the test data. The application was written using Java servlets and JavaScript. It was deployed using Node.js. The Google Maps JavaScript API was used to map the location of nodes and draw path profiles. A screenshot is shown in Figure 28.

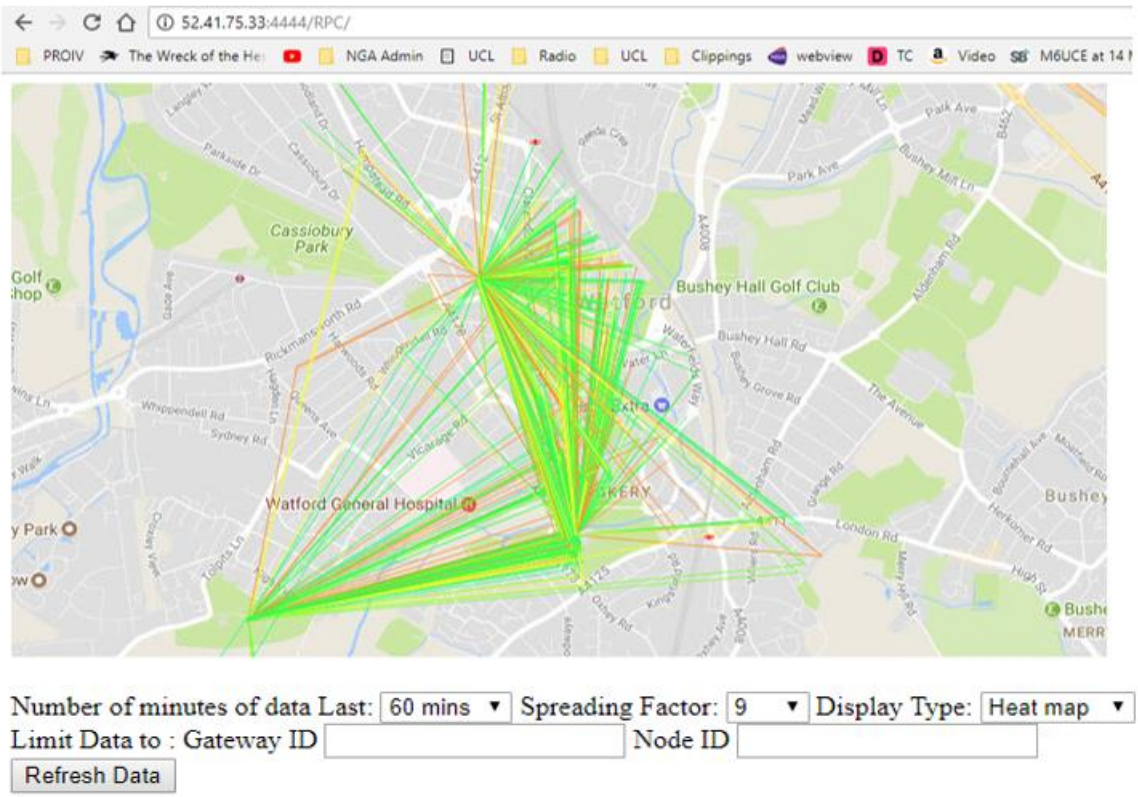


Figure 28: Bespoke Web Client used to present test data (showing path profiles).

Controls at the bottom of the application allow data to be filtered by time and spreading factor. There are also filters that select data by gateway or node. A final set of controls allow data to be exported to CSV format. Two mapping options are supported, a path profile view and a heat map. The heat map view is shown below:

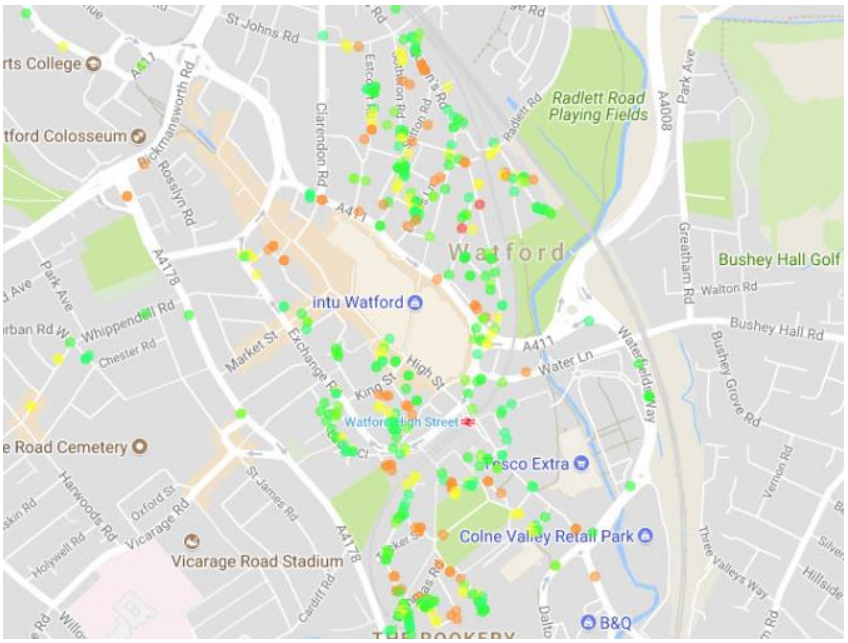


Figure 29: Web GUI - Heat map view.

Working versions of the web GUI are available at the following links:

http://bit.ly/loragui
http://54.202.150.118:8080/RPC/

A user guide is available in the appendices.

1.24.7 Android Application

An Android application was developed to process data received from The Things Connected platform. It is similar to TTN Mapper [36]. Screenshots of the app are shown in Figure 30. The main function of the app is to geocode incoming packets. This is done by comparing timestamps. Location is determined from the smartphone GPS receiver.

The app allows mapping of signal strength without requiring an onboard GPS sensor in a test node. The app was not used in the final drive survey. This was because the use of an external power pack meant that a GPS-enable node could be powered for over a week.

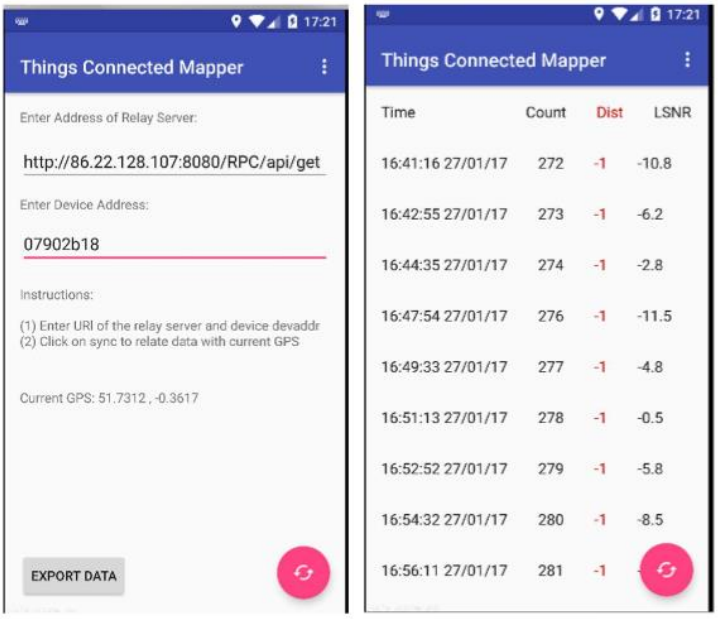


Figure 30: Bespoke Android application to present data and encode with GPS coordinates.

1.24.8 Data Processing

More than 50,000 test packets were sent and received. MS SQL Server 2017 was used to create a data warehouse. This was used to store and process the test data. A normalised data model shown in Figure 31 was adopted.

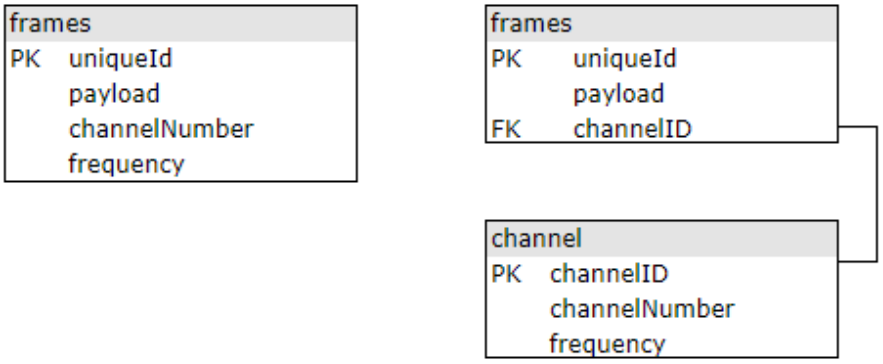


Figure 31: A single data table compared to a data warehouse with 3NF normalisation.

Data was normalised according to the third normal form (3NF). This mandates that a row of data is dependent only on its primary key and no other columns in the table [37]. The following data warehousing techniques were also applied to the database design:

Technique	**Description**
GUIDS	LoRa data as received by the EveryNet gateway has no natural unique key. GUIDS were created for use as a primary key. These are stored as a 16-byte raw data type.
Indexing	The SQL Server database is the main data source for the web and Android applications. To increase performance indexes were added to selected tables.
Views	SQL Server views were created to facilitate reporting and exporting into ICS Telecom and SPSS. Views filter the data horizontally (by selecting a subset of all records) and vertically (by selecting key columns).

Figure 32: Data warehousing techniques used in the study.

Measurement Campaigns

Field measurements are essential in characterising a radio channel. They are also useful in testing equipment and to determine Quality of Service parameters. Real-world data is especially important for systems operating in unlicensed ISM bands. This is because there is no central register of potential interference [38].

Signal strength and packet loss were selected as primary response variables. Signal strength measurements allow validation of a propagation model and packet loss is an indicator of network performance. Explanatory variables considered in the study are listed in Table 11.

Response Variables	Explanatory Variables
RSSI Packet Loss	Distance, Time of Day, Background Interference, Bandwidth, Spreading Factor, Coding Rate, Transmit Power (EIRP), Receiver sensitivity, Line of Sight, Terrain, Clutter, Transmitter and Receiver Height, Frequency (Channel), Contention, Mobility, Indoor penetration

Table 11: Response vs explanatory variables.

Two types of measurement campaign were undertaken. The first campaign was performed using a static reference installation. A test node and gateway were installed 10 metres apart. The node-based software was then used to vary spreading factor. Packets received by the test gateway were forwarded to The Things Network. Metadata was used to identify lost packets. The installation was left in place for several weeks. Data from this campaign served as a baseline for other measurements. The second set of campaigns were drive surveys performed in Watford and Jersey. These are described in the following sections:

1.25 Drive Survey - Watford

1.25.1 Project Area

The project area was defined as the Borough of Watford. This is shown in Figure 33. It has a total area of 21 km^2. As no polygon vector was available of the boundary this was manually traced within ICS Telecom. Three smaller areas were selected as the focus of the drive survey. These are shown in Figure 33 (b). These areas were chosen to include different types of terrain and a mixture of line of sight and non-line of sight locations.

1.25.2 Route Selection

Three approaches were considered, these are shown in Figure 34. In option A, measurements are taken in straight lines away from a gateway. In the second option, measurements are made circumferentially. This has the advantage that distances remain constant, so propagation will vary solely due to elevation and clutter.

Options A and B are impractical due to the layout of roads and local access restrictions. A third option, C was therefore selected. Measurements were taken along public roads. This method has the advantage of matching routes taken by recycling vehicles during their normal operation. It also eliminates the introduction of bias.

An unknown factor introduced by only taking measurements on roads is the effect of local waveguiding. In future studies, samples should be compared with those taken indoors and in open areas.

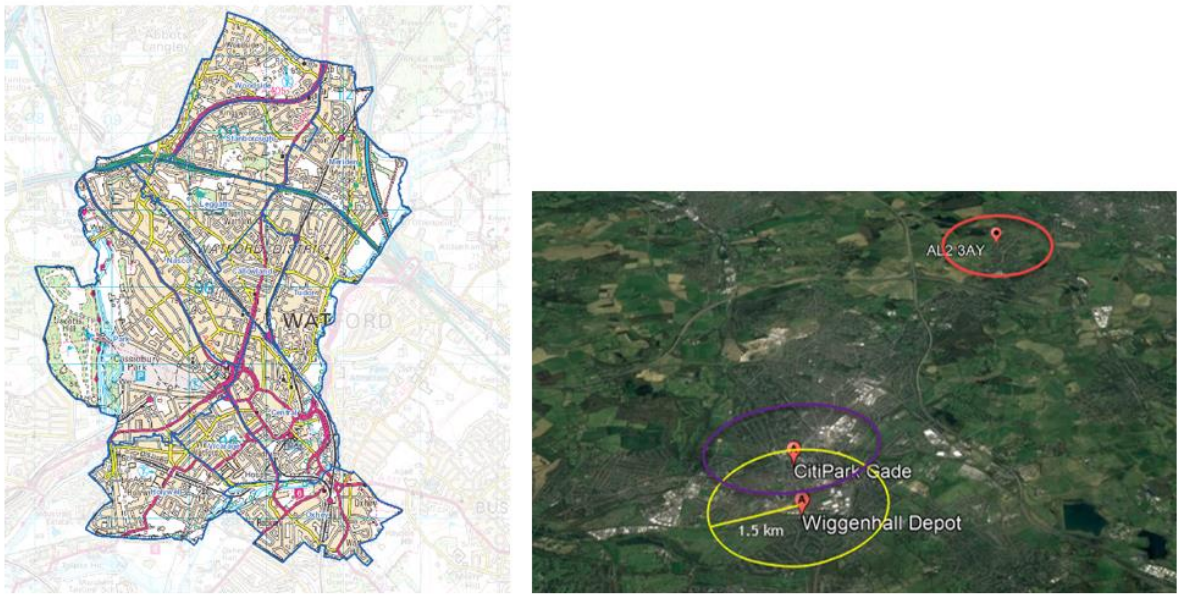


Figure 33: (a) The project area (b) Three areas selected for the focus of the drive survey.

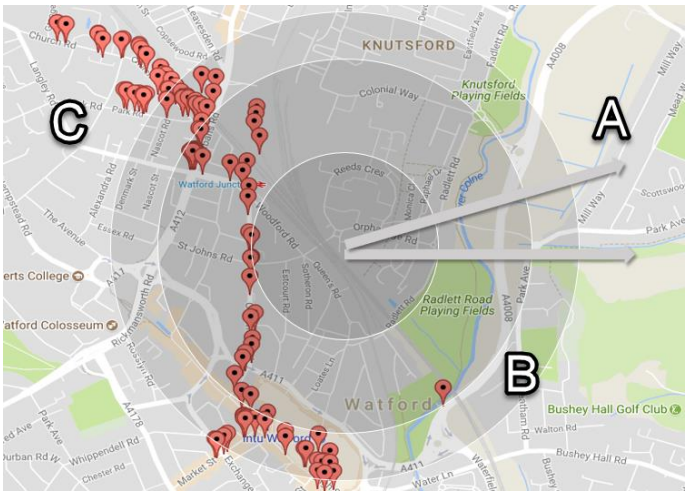


Figure 34: Different approaches to selecting routes (A) radial (B) circumferential (C) randomly along public roads.

1.25.2.1 Lee Criteria

To assist with planning, an estimate was made of the number of data points required to validate a propagation model. The Lee Criteria establishes a sampling frequency necessary to calculate a mean signal strength. It is strictly valid only in the case of mobile measurements but can serve as a baseline for other situations [39].

The criteria states that 50 samples are required over a distance of 40 wavelengths. The wavelength of LoRa signals is 34 cm. 11,000 data points are required for a 3 km path. For a test vehicle moving at

30 mph, this equates to 200 samples per second [40]. Sampling at this rate was not possible due to duty cycle restrictions. Indicative sampling at the maximum rate allowed was therefore used. This equated to 22 packets per hour. Chirp spread spectrum has good resistance to multipath effects which may help to mitigate the lower sample numbers and ensure a statistically significant outcome.

1.25.3 Test Vehicles

With permission from Veolia and Watford Borough Council, test nodes were fitted to two recycling lorries. Data was then collected during normal operation. The nodes were left in place for three weeks. Using recycling lorries had the following advantages:

- Routes cover all roads in the borough.
- Routes are consistent and repeated every four weeks.
- The lorries stop at regular intervals. Stationary measurements can be compared to those taken when the vehicles are moving.

1.25.4 Enclosures and Mountings

A case was needed to protect equipment from harsh operating conditions. A means of attaching the node to the vehicles was also required. Two solutions were tested. In Figure 35, a node is contained in a plastic case and connected to the vehicle's 12V power socket. The node can be connected either directly or using a USB power lead.

Figure 35: Connecting a test node to a 12V power socket (a) directly (b) via a USB lead.

The node was left in place for one week. The following points were noted:

- Locating the device inside the cab resulted in asymmetric insertion losses. Signals transmitted from behind the vehicle were required to pass through the length of the body to reach the sensor.
- The device was in the main operator compartment. This increased the risk of damage and of inconveniencing the crew.

An alternative solution is shown in Figure 36. Components are secured in a waterproof case. Padding is used to hold the components in place and to provide shock proofing. The antenna is passed through a hole in the side of the case. Self-amalgamating tape is used to create a waterproof seal around the lid.

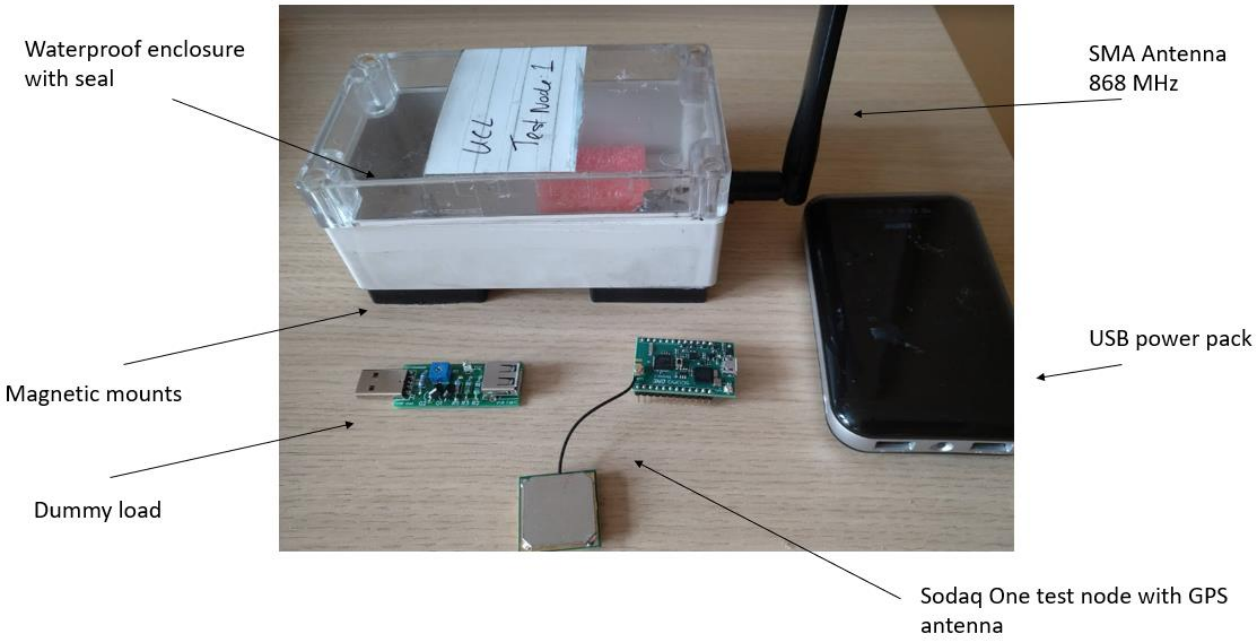


Figure 36: Sodaq One node inside a waterproof enclosure.

The case was attached to the roof using magnets as shown in Figure 37.

Figure 37: Positioning the sensor.

Ideally, the antenna should be placed in the centre of the roof to avoid azimuth pattern distortion. For safety reasons, this was not possible.

1.25.5 Selecting a Suitable Power Source

When taking mobile measurements, a portable power source is required. The table below lists the power sources tested:

Power Source	Image	Capacity	Measured Lifetime
6 * AA batteries		6 x 2400 mAh	20 hours
9V Alkaline Battery PP3		500 mAh	10 hours
Sodaq Lithium Ion Polymer		1200 mAh Output 3.7v	~ 2 days
Anker Astro E5 Power Pack		15000 mAh (stated) 12970 mAh (measured)	7 - 12 days

Table 12: Power sources tested.

The Anker E5 power pack powered the test node for longest. It was the largest of the power sources but sufficiently portable to be mounted on the top of the test vehicles. The stated capacity of the Anker E5 was 15,000 mAh. This was measured as 12,970 mAh.

1.25.6 USB Power Packs

Capacity was measured using a power meter placed between a charger and the power source. The power source was fully charged from empty and the capacity recorded. In all tests, the measured capacity was lower than the specified value. One explanation is that the stated capacity may refer to the internal Li-Ion cells. Actual devices require a boost converter to switch from 3.7 V to 5 V. This can incur losses of around 25% [41].

Unexpected behaviour was observed when using a USB power pack to power a LoRa node. When using the Anker E5 with a Sodaq One node, current was only supplied for 30 seconds. This was found to be due to protection circuits within the power pack. Power packs on the market have different threshold values. In most cases, power is cut when current falls below 10 mA. An additional load is therefore required to ensure that the current remains above this threshold. Various designs for dummy loads are available [42]. These create a current spike with a duty cycle of around 1 to 2%. For example, a 200 mA pulse of 20 ms duration every 1 to 2 seconds results in an average current draw of 10 mA. This is enough to keep most power packs from switching off. In this work, the Sotabeams Keep-Alive Load [43] was used. The board uses a potentiometer to vary the load between 6 and 150 mA. The Sotabeams load along with a USB power meter is pictured below:

Figure 38: Power Pack with USB power meter and dummy load.

1.25.7 Solar Panels

Solar cells (single-junction) have a maximum theoretical efficiency of 33%. In the present study, their use is hampered as the recycling vehicles are kept under cover when not in operation. Three different solar panels were tested. Results are discussed in the table below:

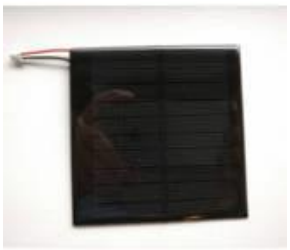
Solar Cell	**Image**	**Notes**
Telecomusers Power Pack and Solar Panel		This is a combined power pack and solar panel. In test conditions, the solar panel was not able to charge the external power pack.
Sodaq Solar Cell		Energy efficiency 17% Typical voltage: 5.5V Typical current: 170 mA This cell is designed to be used in conjunction with the Sodaq LiPo battery. It was not sufficient to power the node by itself. Using the solar cell with the LiPo battery only resulted in a small improvement in battery life.
Solar Charger, BigBlue 28W		This was the only solar panel tested capable of powering the node directly. It was used with the Anker E5 to extend battery life.

Table 13: Solar panels tested.

1.26 Drive Survey - Jersey

A drive survey was performed in Jersey in partnership with JT (formerly Jersey Telecom). A quarter-wave dipole antenna was attached to the roof of a car (shown in Figure 39). Using an external antenna eliminated asymmetric insertion losses and the metallic roof functioned as an informal ground plane. The antenna was connected to the Sodaq One node using a 5 metre feeder cable. The node was powered using the vehicle's 12V power socket.

Figure 39: Antenna mounted on a car for the Jersey drive survey.

1.26.1 Route Selection

Jersey is an ideal testbed for telecommunication services. It has a wide range of environments in a compact area. LoRaWAN has been selected as a key technology by the Sandbox Jersey initiative [44]. There are currently ten operational gateways. Routes were selected to cover all parts of the island. An example route is shown in Figure 40.

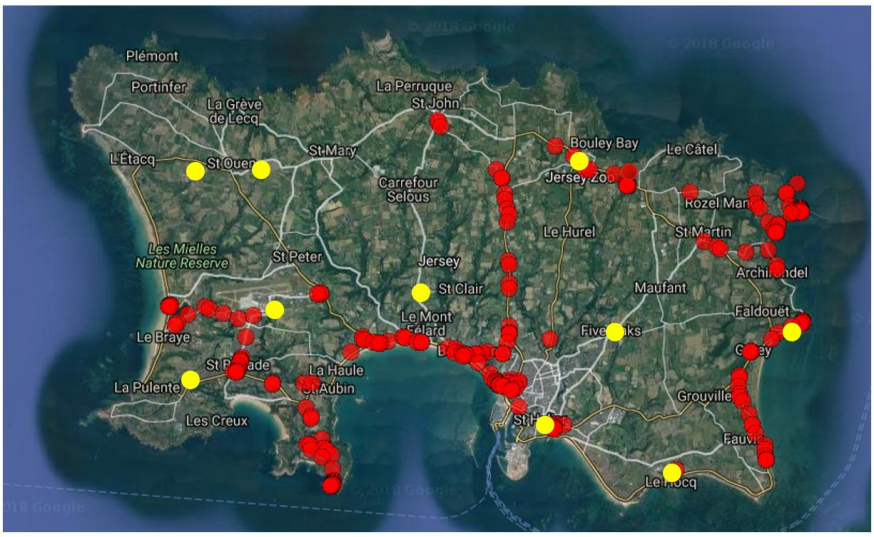


Figure 40: Example route (red) and the location of gateways (yellow).

Results

In this chapter, we present data from the static reference installation and drive surveys:

1.27 Investigating the LoRa Modulation

A software defined radio (SDR) was used to produce a spectrogram of a LoRa packet sent from the reference installation. This is shown below:

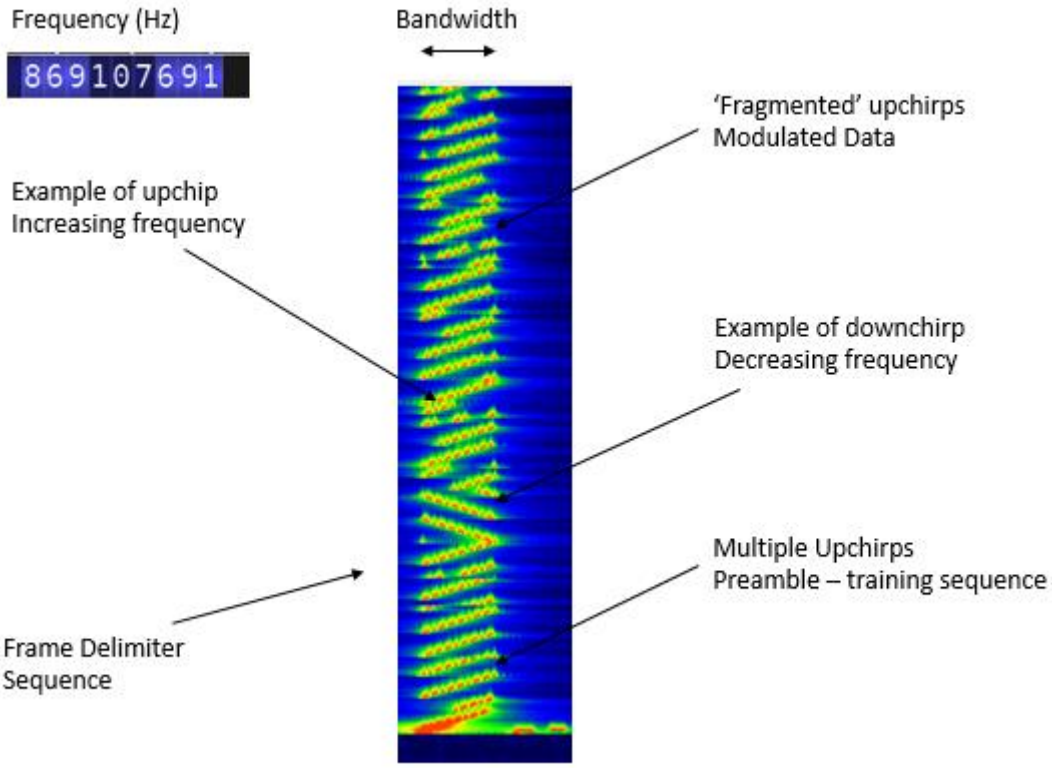


Figure 41: Spectrogram of a LoRa Packet (SF 9) using RTL2832 (NooElec) SDR and CubicSDR software.

A set of initial upchirps can be seen. This indicates the preamble and training sequence. It is followed by a Start of Frame Delimiter (SFD) of 2.25 down chips. The payload consists of a series of fragmented upchirps. The start time and duration of each up chirp is modulated. The rate of frequency change remains constant. This is seen in the waterfall as all slopes are at the same angle.

1.28 Measuring Power Consumption of a Node

The current drain of the test node was measured using a USB power meter. The current was recorded as a function of duty cycle and spreading factor. It was assumed that all packets were sent correctly and that no energy was expended in resending lost frames.

1.28.1 Current Drain vs Duty Cycle

As shown in Figure 42, the current drain increases with duty cycle. This is to be expected as a node requires significantly more power whilst transmitting. Specifications for the RN2483 chip state that it draws 40 mA whilst transmitting, dropping to 14 mA when receiving [45].

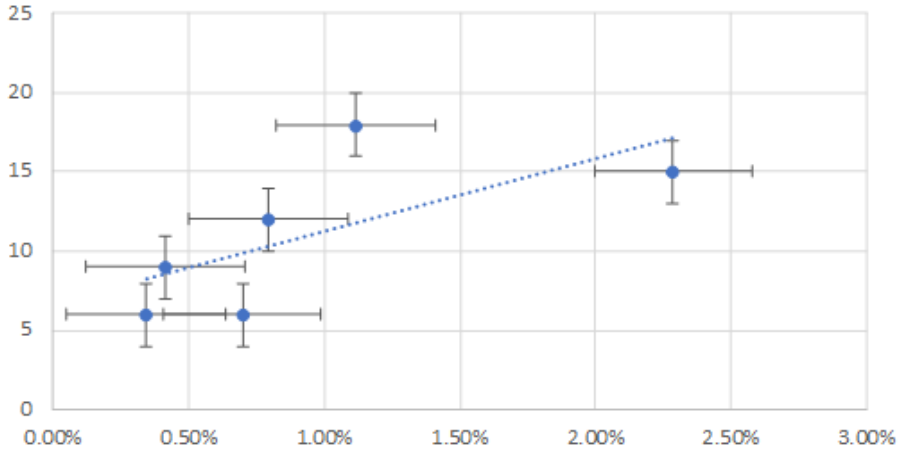


Figure 42: Current Drain (mA) vs Duty Cycle (Spreading Factor 7).

1.28.2 Current Drain vs Spreading Factor

Results are shown in Figure 43. No trend between current and spreading factor was observed. This is surprising as packets sent with a higher spreading factor have a longer transmission time. A possible explanation is that some nodes enforce duty cycle restrictions by introducing a backoff period. This would be longer for packets sent with a higher spreading factor. More work is required to understand if this could account for the results seen.

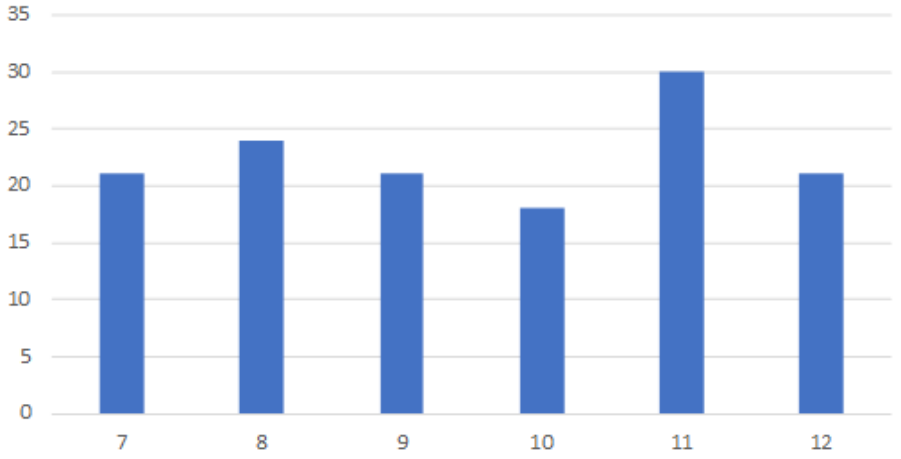


Figure 43: Current Drain (mA) vs Spreading Factor (Measured using a Sodaq One node, 1% duty cycle)

1.29 Investigating Power Saving Techniques

Various techniques for reducing power consumption were investigated. This was of practical importance as changing batteries for the drive survey required a site visit.

1.29.1 Reducing Duty Cycle

Reducing duty cycle is an efficient energy saving technique. This may not be possible as some applications require a minimum transmission rate. Additionally, low duty cycles can increase the need for nodes to perform local calculations and to persist state. This can result in increased power consumption, along with the need for more complex hardware [46].

1.29.2 Implementing Sleep Mode

To increase battery life a node can be programmed to sleep when not transmitting. The current drain of the RN2483 chip in sleep mode is specified as 10 µA. The sleep value for the node is likely to be several orders of magnitude higher [47]. In addition, some types of batteries discharge themselves resulting in a residual current. To test the use of sleep mode, a third-party library (LowPower.h) was used. This library switches compatible MCUs into a low power mode. The lowest current drain measured was 30 mA. This suggests that components other than the MCU draw significant power. To wake from sleep a hardware or software interrupt is required. The Sodaq One sensor has an onboard accelerometer. In future work, this could allow the device to be woken from sleep when the survey vehicles change location.

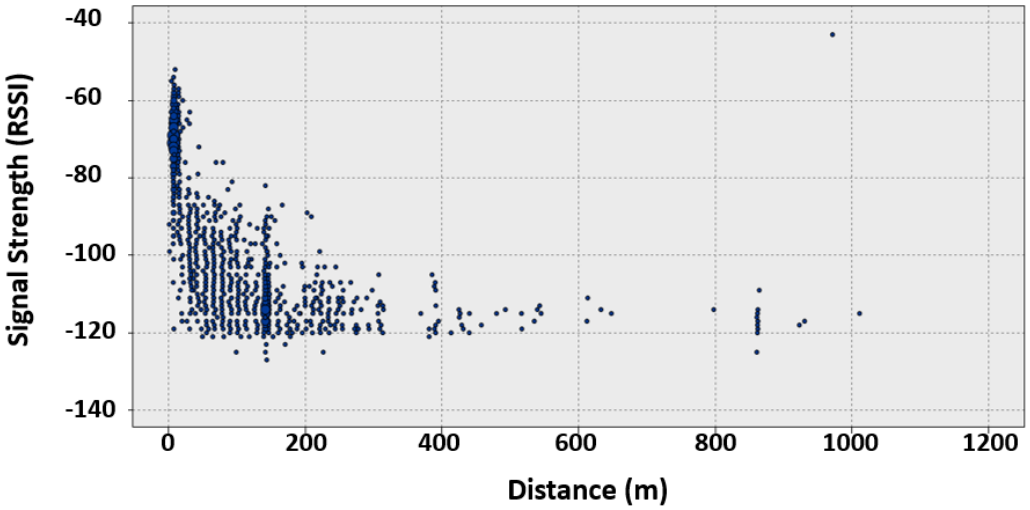
1.29.3 Use of Alternative Location Sensing

The current drain of a non-transmitting node when GPS was enabled was measured as 40 mA. This dropped considerably when the GPS chip was disabled. To achieve power efficiency the use of alternative location sensing techniques should be considered. The Cisco LoRaWAN Gateway enables positioning of nodes based on Time Difference of Arrival (TDoA) and RSSI [48].

1.30 Measurements of Signal Strength

Received Signal Strength Indicator (RSSI) is a measure of signal strength. It is expressed in arbitrary units and is normally a negative number. The higher the value the stronger the received signal. In this study, RSSI figures were obtained from the packet metadata. Ideally, RSSI values would be calibrated. To do this an antenna with known performance characteristics would measure the signal radiating from the test source.

Figure 44 shows a rapid reduction in RSSI with distance up to 100 metres followed by a more gradual decline. One possible explanation for the change in the rate at 100 metres is a loss of direct line of site. Another possibility is that close to a base station LoRa nodes revert to FSK modulation. No evidence was seen for this in the metadata.



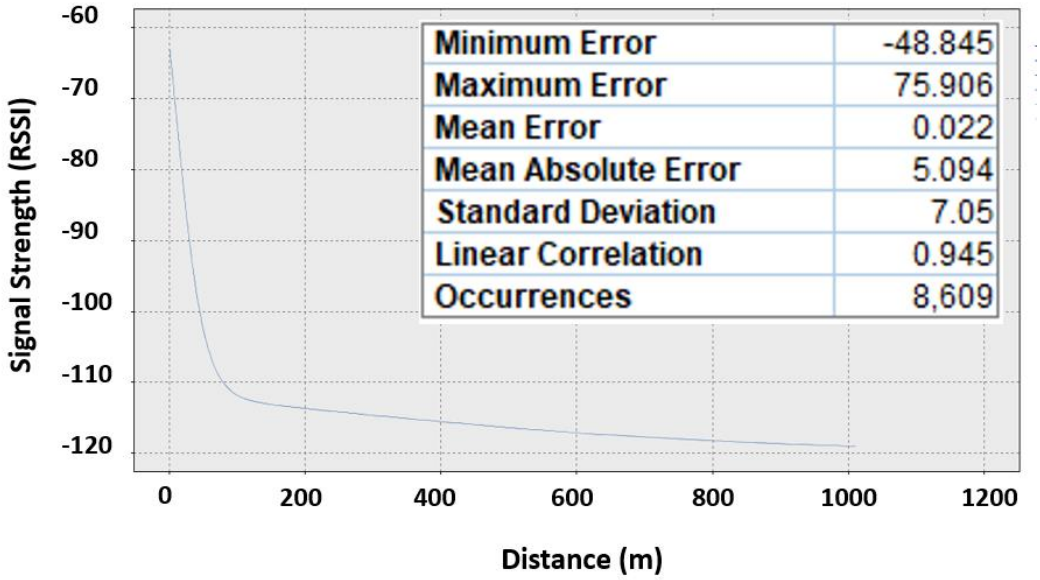


Figure 44: RSSI vs Distance (Sodaq One) (a) raw data (b) linear correlation. The height of node is fixed at 3m above ground level. Gateway is sited at Wiggenhall Depot. The dataset includes transmissions sent with various spreading factors.

The results in Figure 44 show a significant standard deviation. A possible reason for this is that elevation was not considered. An additional analysis was performed to determine how significant elevation is in predicting RSSI. Elevation was added for each GPS coordinate using the publicly available SRTM1 (Space Shuttle Radar Topography Mission) database. The relative importance of distance and elevation was ranked using SPSS Modeller.

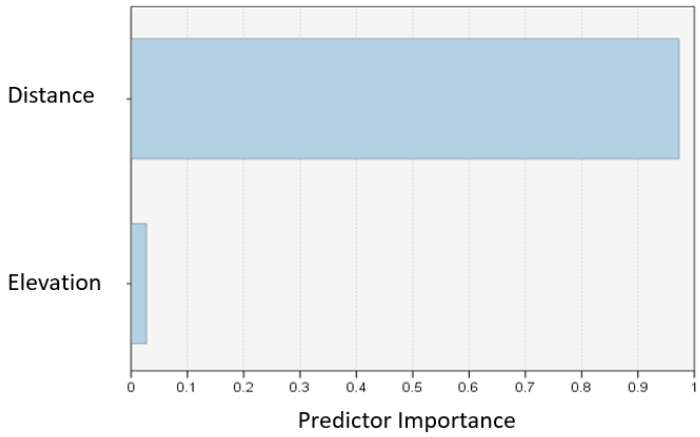


Figure 45: Graph showing relative importance of distance vs elevation in predicting RSSI.

Figure 45 suggests that, in this study, elevation has only a small effect on RSSI compared to distance. This should be understood in the context of the local terrain which was a relatively flat urban environment.

1.31 Measuring Rates of Packet Loss

Packet loss is a key network performance indicator. The amount of loss that can be tolerated depends on the application. For example, monitoring applications can typically tolerate higher levels than

those in the control plane. LoRaWAN uses frame counters to track transmitted and received packets. Both an uplink counter, *FCntUp* and a downlink counter, *FCntDown* are specified. These were used to identify packets lost in transmission.

1.31.1 Packet Loss vs Distance

Packets lost were identified in the Watford and Jersey surveys. Results are shown in Figure 46.

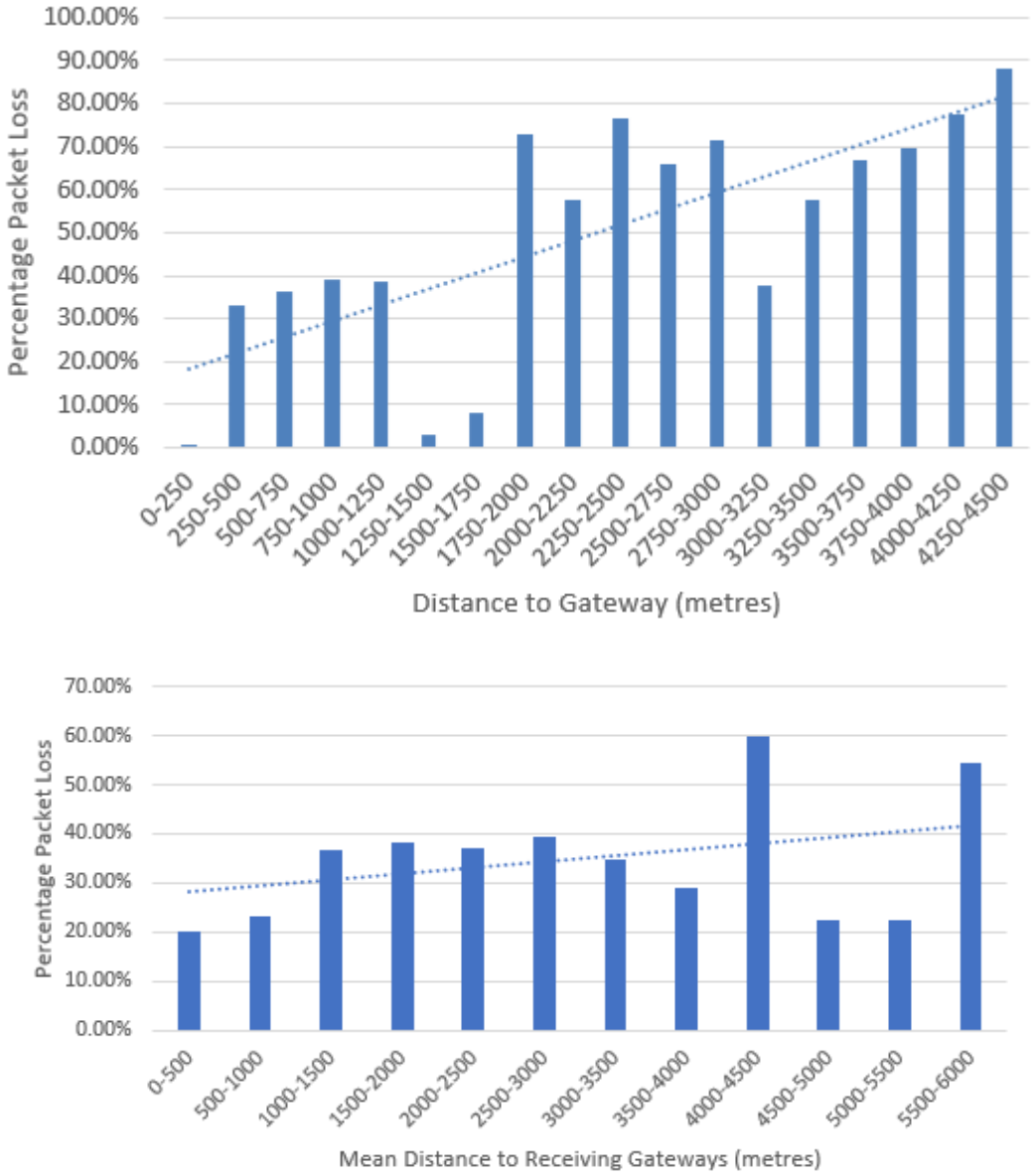


Figure 46: Packet loss as a function of distance from a gateway (a) Watford (40,000 packets) and (b) Jersey (500 packets). Results shown are for multiple receiving gateways.

In the Watford data, low levels of loss (less than 5%) were seen up to 250 metres. Packet loss then increases up to 4.5 km where 80% of all packets are lost. The Jersey data shows a slower increase with an average loss of 32% up to 4.5 km. Both studies show a higher than expected rate of packet loss. Possible reasons for this are given below:

- A large proportion of the measurements are made whilst the survey vehicles are moving. This may introduce transmission errors due to the Doppler effect.
- Bursts of packets may be lost due to the effects of local clutter. For example, a recycling vehicle may stop for a period behind a tall building.
- The test node cycles through all spreading factors. Ideally, the results would be repeated using the Adaptive Data Rate Mechanism. This selects the most suitable spreading factor and is likely to reduce the occurrence of lost packets. ADR is not currently supported by the Jersey network.

An interesting feature in Figure 46 (a) is low levels of loss observed between 1.3 and 1.6 km. Figures are shown in the table below:

Distance (metres)	Packets Sent	Packets Lost	% Loss
1300-1400	1122	34	3.03%
1400-1500	5208	141	2.71%
1500-1600	1251	87	6.95%

Figure 47: Packet loss figures for the Watford survey between 1.3 and 1.6 km.

Most of these packets were sent when the test vehicles were parked at the recycling depot. Low error rates may be explained because the vehicles were stationary and because the Wiggenhall depot is situated in open terrain away from local clutter.

A further analysis of packet loss was done comparing receipt of packets by multiple and single gateways. Results are shown in Figure 48.

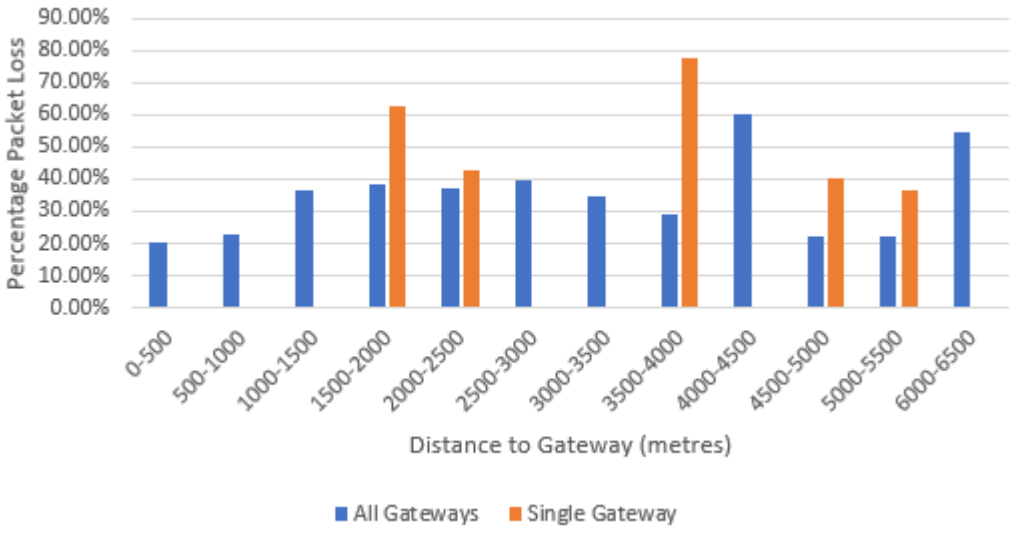


Figure 48: Packet loss: comparison between multiple and single gateways.

As expected the use of multiple gateways resulted in fewer lost packets.

1.31.2 Packet Loss vs Time of Day

One source reports a higher loss of packets at night [49]. Figure 49 shows packet loss versus time of day for the static reference installation:

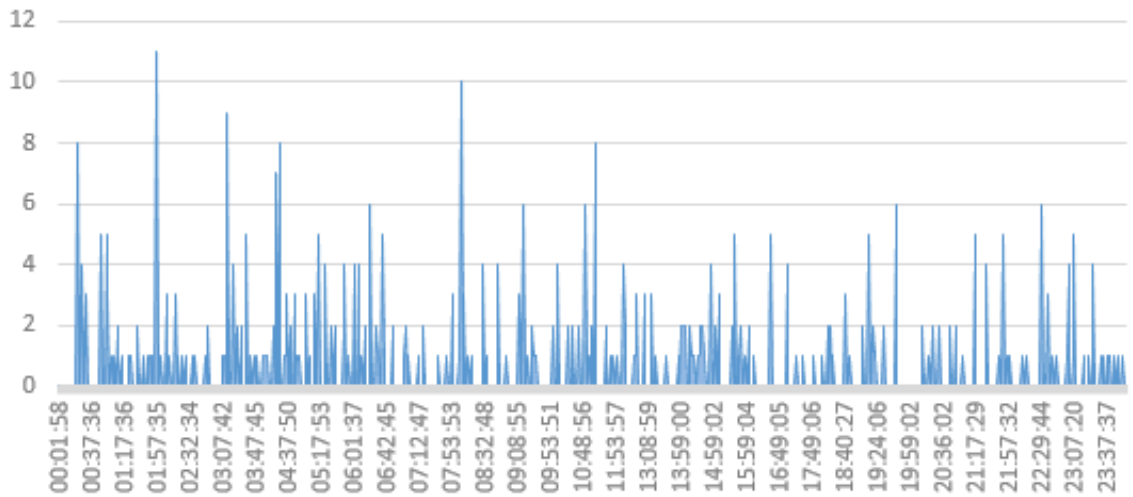


Figure 49: Packet loss (packet count) vs time of day. Taken using the static reference installation (SF7 at a fixed transmission rate).

No clear trend was observed. The same analysis was done using the data from the Watford drive survey. Results are shown in Figure 50.

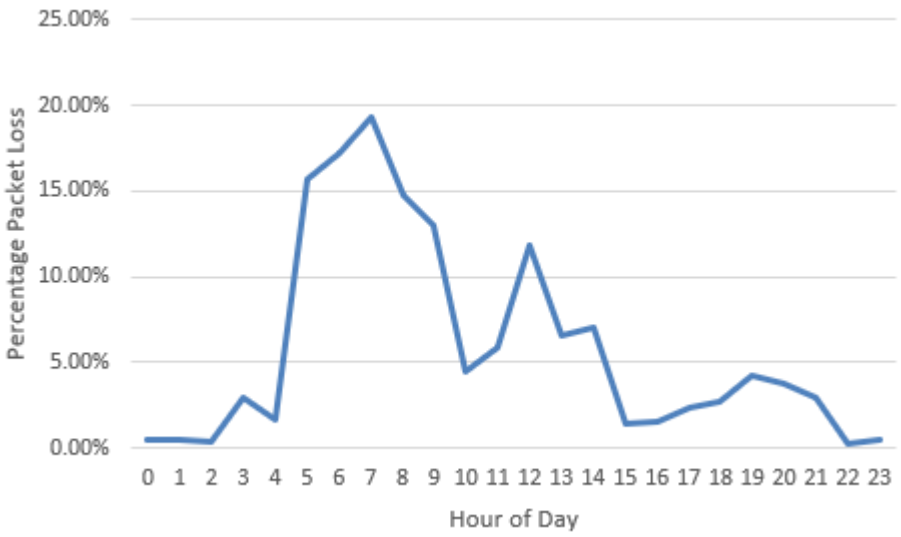


Figure 50: Packet Loss vs time of day for the Watford drive survey.

A marked increase in packet loss is seen between 4 am and 10 am. This corresponds to an active period for the recycling vehicles. With packets being transmitted whilst the lorries are travelling or in remote areas of the borough.

1.31.3 Packet Loss vs Sending Channel

An analysis was done of packet loss by sending channel. Results are shown in Figure 51.

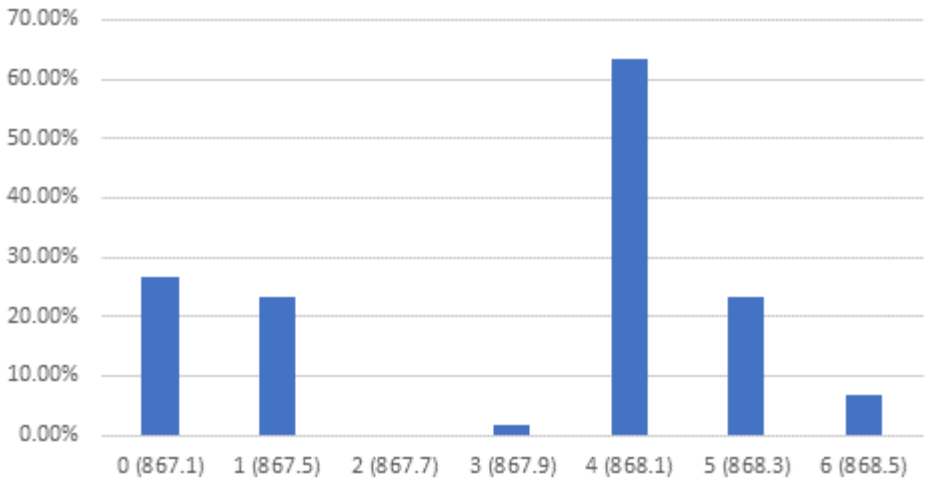


Figure 51: Packet loss by sending channel (480 samples).

There is a 60% loss for channel four compared to an average of 13% on the other channels. This was unexpected. Possible explanations are given below:

- Packet loss at some frequencies is caused by RF interference near the node or gateway. This would seem unlikely as LoRa has good resistance to narrowband interference. Additionally, the measurements were made over an extended period. In future studies, the presence of interference could be investigated using a spectrum analyser or RF probe.

- Separate power restrictions exist for each LoRa channel. It is possible that packets are being sent with different transmit powers.

- Channel selection is pseudo-random. In the above analysis, it was assumed that an equal number of packets were transmitted on each channel. The observed pattern may be due to the channel selection algorithm favouring particular channels.

Modelling and Simulations

A propagation model was created to predict coverage around the Watford gateways. Two gateways were used to train the model with a third being used for testing. Steps taken to validate the model are shown below:

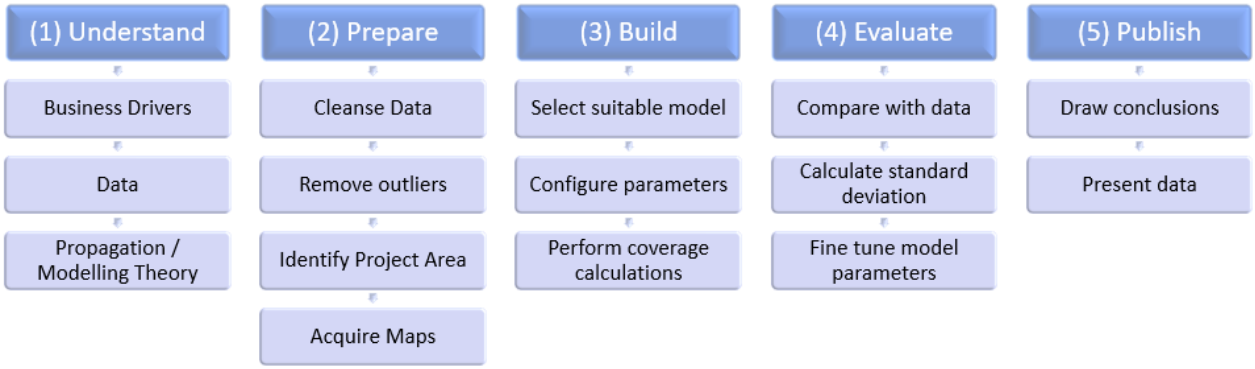


Figure 52: Steps to validate a propagation model.

1.32 Background

The primary objective in validating a propagation model was to assist Watford Council in siting LoRaWAN gateways. They wanted to know the number and position of gateways required to achieve a 90% coverage of the borough. The network planning software used was ICS Telecom. This supports a wide range of industry standard models.

1.33 Theory

Propagation models can be categorised as follows:

Deterministic – These models consider path profiles between the transmitter and a receiver. A physical understanding of electromagnetic propagation together with models of reflection, diffraction, scattering, and absorption are used to predict path losses.

Empirical – These models are statistical in nature. They are created from field measurements. They do not provide an analytical explanation of how signals propagate. A well-known example is the Okumura-Hata model. The resulting propagation equations are valid only for specific environments and frequencies. As they are based on field studies all propagation phenomena are accounted for. The typical standard deviation between predicted and measured path loss is in the range 10 to 14 dB [50].

In this work, we mainly consider deterministic models. Empirical models, such as COST-231 Hata, are useful where accurate cartographic data is not available. They have not been widely tested for LPWANs [51]. In the present study, detailed 3D maps of Watford are available. This supports the choice of deterministic or *path specific* model.

1.33.1 Attenuation

Deterministic models consider three main forms of attenuation. These are absorption, diffraction, and reflection. In urban environments reflection from buildings, sometimes called structural attenuation, is often the most significant [15]. In addition to buildings, the earth itself is a reflector. If the earth is present in the 1st Fresnel zone this can lead to significant attenuation. This is more likely in IoT applications as nodes are often close to the ground. Models of the Fresnel zone are shown in Figure 53.

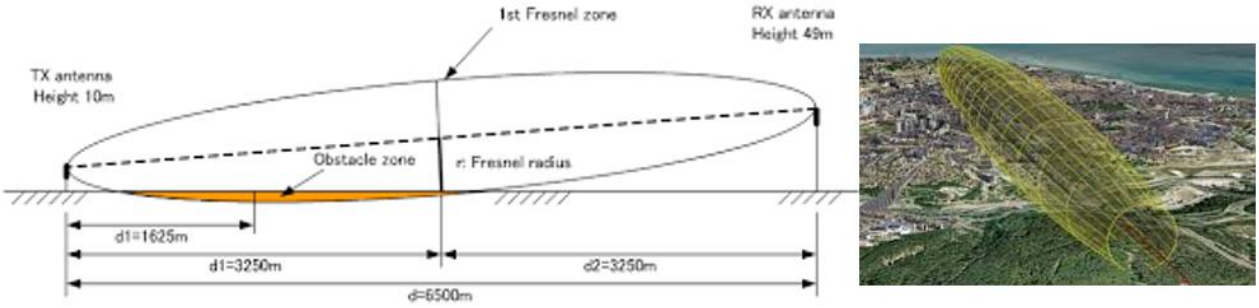


Figure 53: Modelling of the Fresnel zone.

1.33.1.1 Diffraction Modelling

Models such as Deygout-66 and Bullington are based on the Geometrical Theory of Diffraction (GTD). Obstruction loss is estimated by modelling peaks as a series of knife-edges [38]. The number of intrusions into the Fresnel Zone that can be accounted for depends on the model. The principle of knife-edge diffraction is illustrated below:

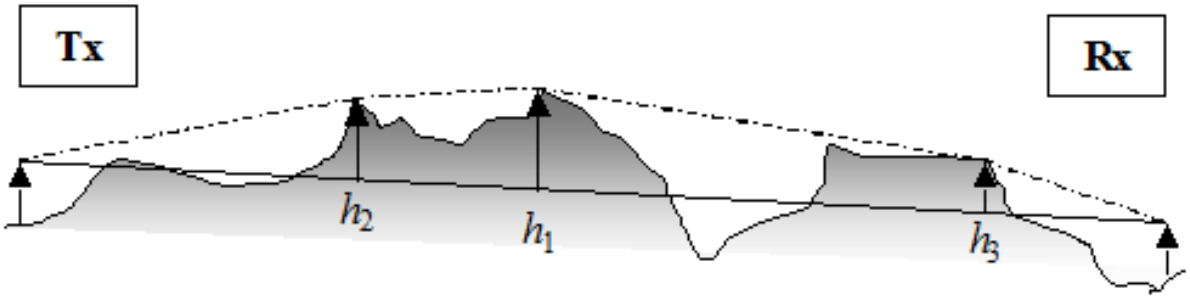


Figure 54: Knife-edge diffraction [52].

For some terrain profiles, better results can be achieved by modelling peaks as cylinders.

1.34 Propagation Modelling

The first step in creating a model was to prepare the data. Results were manually inspected (eyeballed), and obvious outliers removed. Most discrepancies in the data were found to be due to incorrect GPS coordinates. The data was then imported into ICS Telecom.

The main features of a propagation model are described below:

1.34.1 Digital Terrain Model

Propagation models require a Digital Terrain Model (DTM). This is a 3D representation of terrain. They are created using aerial photography (stereo photogrammetry) or more recently by using LiDAR. The DTM used in this study has a resolution of 25 metres. Project areas are shown below:

Figure 55: Digital Elevation Maps of project areas.

The term Digital Surface Model (DSM) is used when terrain data is enhanced with buildings and other features. In this work, a clutter layer was used rather than a DSM.

1.34.2 Clutter

Clutter is any object natural or manmade that exists on top of the bare earth topology. Type and height of the clutter are significant. For modelling purposes, different types of clutter are assigned a code. Specific attenuation (dB/m) is used to measure loss caused by clutter. This value is frequency dependent. Clutter maps of the project areas are shown in Figure 56:

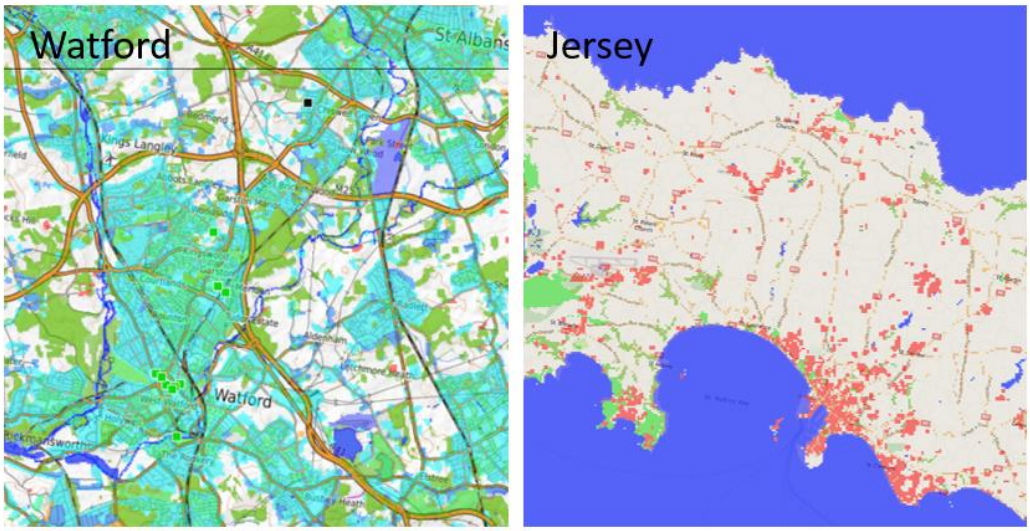


Figure 56: Example of clutter maps.

In the current study, several clutter codes were recategorised by comparing clutter maps with Google Earth imagery.

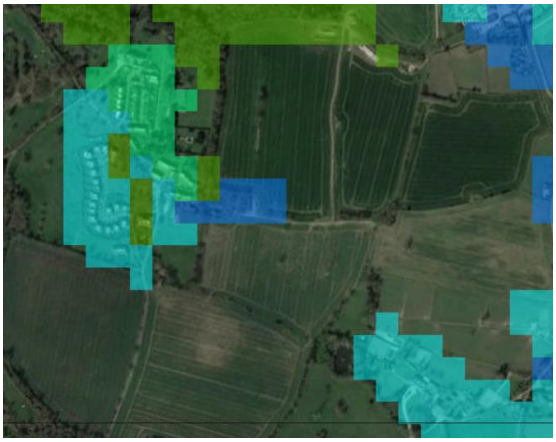


Figure 57: Example of clutter categorization.

The final map layer is a topological or street map overlay. In the current project, both Bing and Ordnance Survey maps were used. As layers are superimposed the overall resolution is limited by the lowest value. For example, there is no benefit in having a 5m clutter layer if the elevation map has a resolution of 25m.

1.34.3 Calibration

Propagation models in ICS Telecom can be trained by adjusting correction factors. These alter the way diffraction and absorption is modelled. Data from the Wiggenhall Depot and Gade Carpark gateways were used for training purposes. A third dataset, from the Chiswell Green gateway, was used to test predictions. Correction factors were adjusted until the lowest standard deviation was achieved.

1.35 Analysis of Results

Table 14 shows the performance of all models tested.

Model Name	**Correction Factors**	**Standard Deviation dB (Watford)**	**Standard Deviation dB (Jersey)**
Deygout 66	Fresnel Subpath	5.92	11.79
Deygout 66	No subpath loss	3.44	9.33
Bullington	Delta Bullington subpath	2.92	11.32
ITU-R 526 (cylinders)	Standard	6.23	17.08
Hata Cost 231 (150-2000 Mhz)	No diffraction loss No subpath loss	6.73	10.48
Extended Hata (30-3000 Mhz)	Bullington Diffraction Fresnel Subpath	4.19	9.96

Table 14: Comparison of different propagation models.

The model with the lowest standard deviation was Bullington. For the Watford data, there was a standard deviation of 2.92 dB between predicted and measured results. Sources suggest that standard deviations below 6 dB [53] are acceptable. Correlation between measured and predicted signal strength is shown in Figure 58.

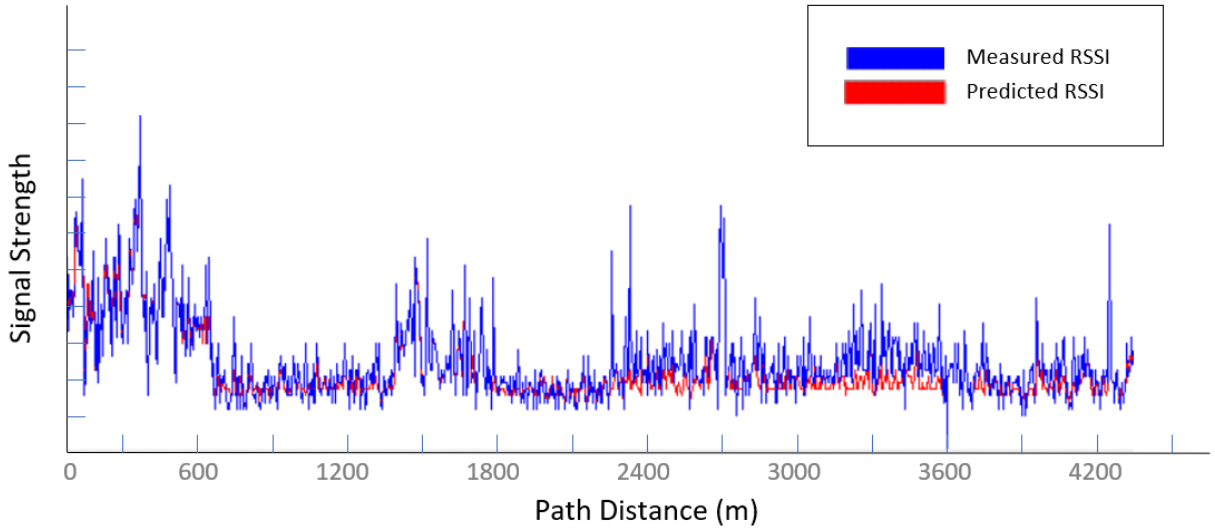


Figure 58: Measured vs predicted RSSI figures.

Performance characteristics of the Bullington model are given in Table 15.

Results	**Value**
Standard Deviation	2.99 dB
Average Error	-1.22 dB
Correlation Factor	0.89
Percentage of predictions with < 6 dB error	93.75

Table 15: Bullington model performance characteristics.

1.35.1 Creation of Coverage Maps

Coverage maps were produced using ICS Telecom. Examples of these are shown in Figure 59.

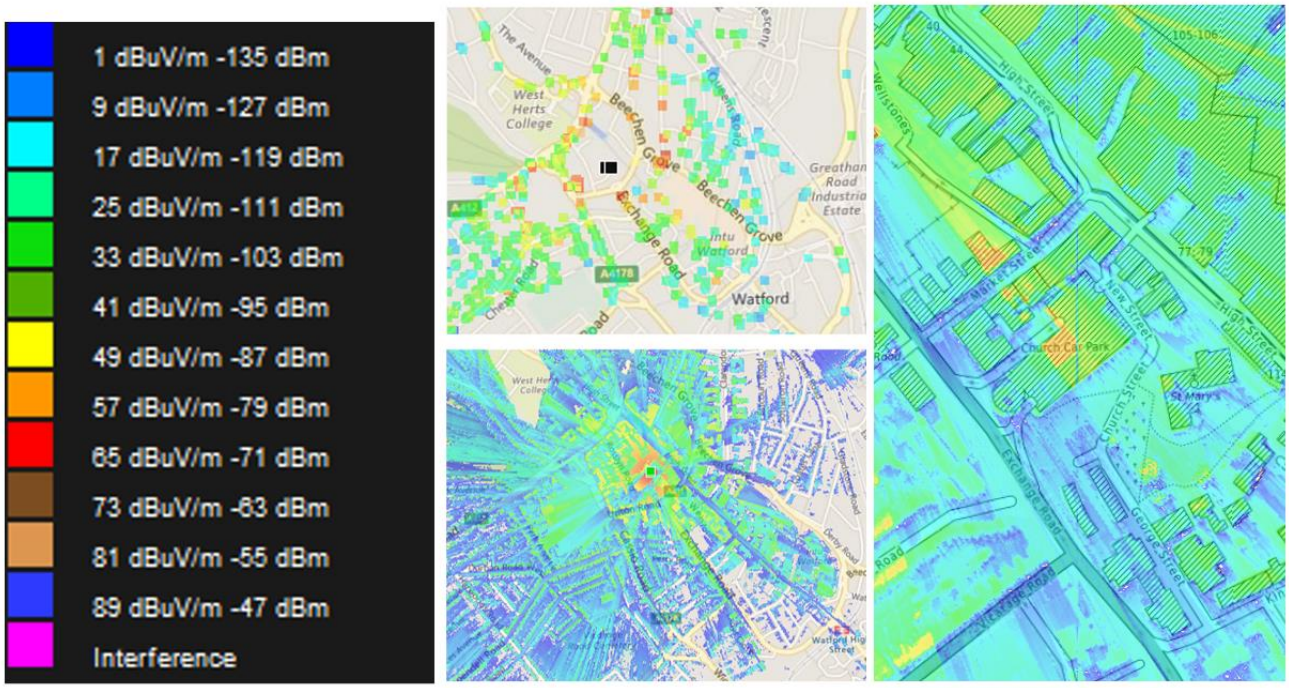


Figure 59: Example of predicted coverage maps compared with the actual results (middle top).

Detailed features of the maps are shown in Figure 60.

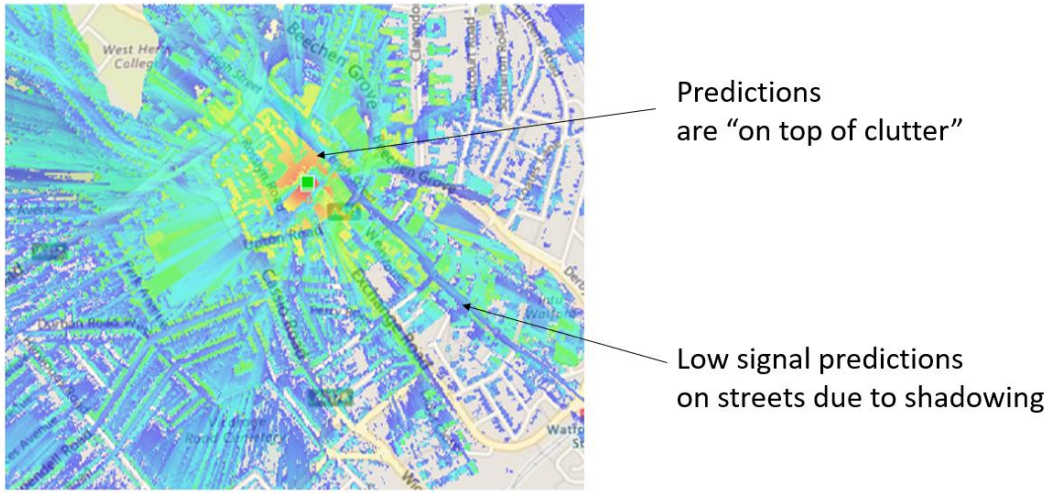


Figure 60: Features of a coverage map.

Signal strength predictions are made assuming the receiver is on top of clutter, for example, buildings. For most applications, indoor predictions are more valuable. This could be achieved by producing two separate models. These can then be combined in a single map.

1.35.2 Comparison to Line of Sight

ICS Telecom was used to analyse the line of sight profile around the Gade carpark gateway. A comparison between the line of sight profile and the Bullington model is shown in Figure 61.

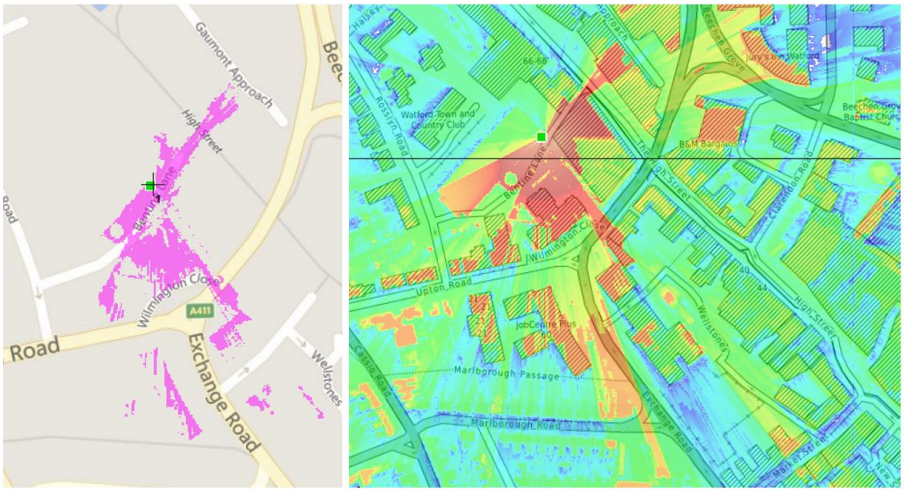


Figure 61: Comparison between Bullington model and line of sight (Gade carpark gateway).

From this comparison, line of sight is a strong predictor of the received signal strength. Were possible, a line of sight communication to a LoRaWAN gateway should be preferred.

1.35.3 Analysis of an area of Low Signal Coverage

A propagation model can be used to predict areas of poor coverage. An example area in the Watford predictions is shown in Figure 62. The low signal coverage is a result of a dip in the terrain.

Figure 62: Analysing an area of low signal coverage

Analysis of this type can support business decisions regarding the number and placement of additional base stations.

1.36 Network Planning Exercise

The propagation model was used in a real-world network planning exercise. The results will be used to plan the next stage of the Watford LoRaWAN rollout. Candidate sites for LoRaWAN gateways are shown in Figure 63.

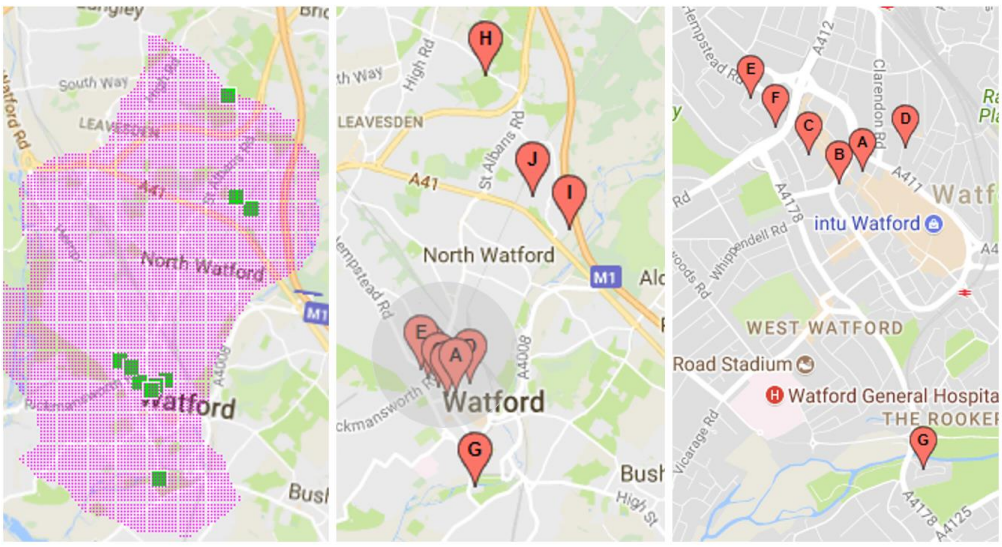
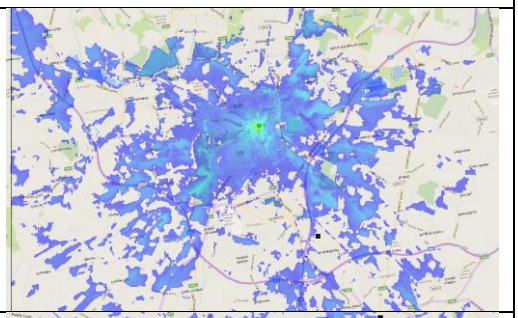
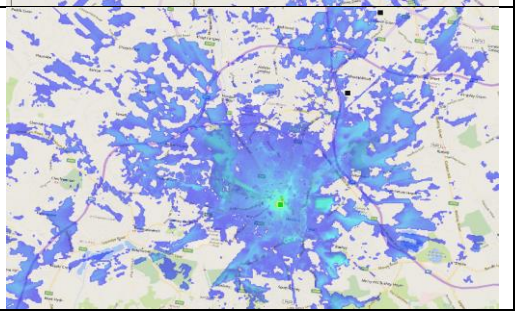
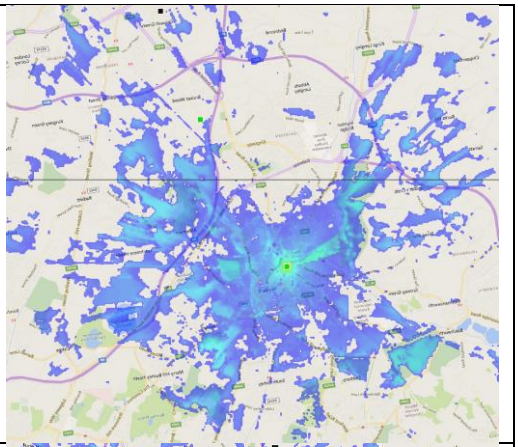
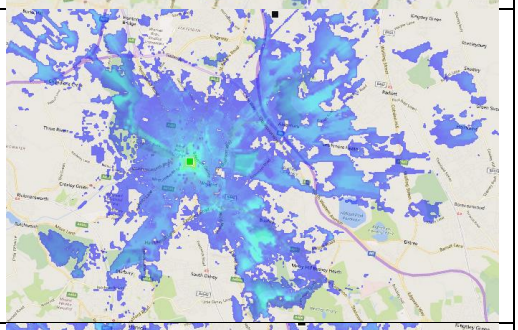
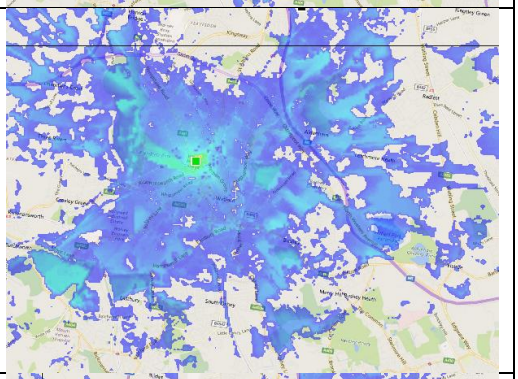
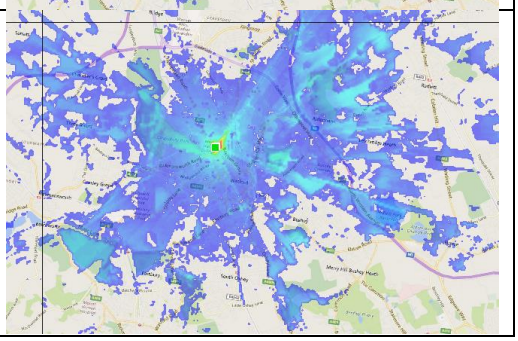


Figure 63: LoRaWAN candidate sites in Watford.

Using the Bullington model, coverage predictions were made for each site. These are shown in Table 16.

SITE REF	SITE NAME	ADDRESS	Percentage Covered	Mapped Coverage (Simulated)
A	New Watford Market	Watford House Lane, Watford, Herts WD17 1BJ	68	
B	Church Car Park	Exchange Road, Watford, Herts WD18 0JD	60	

C	Gade Car Park	Rosslyn Road, Watford, Herts WD18 0JX	59	
D	Sutton Car Park	Estcourt Road, Watford, Herts WD17 2PS	65	
E	Central Leisure Centre	Hempstead Road, Watford, Herts WD17 3HA	74	
F	Watford Town Hall	Hempstead Road, Watford, Herts WD17 3EX	73	

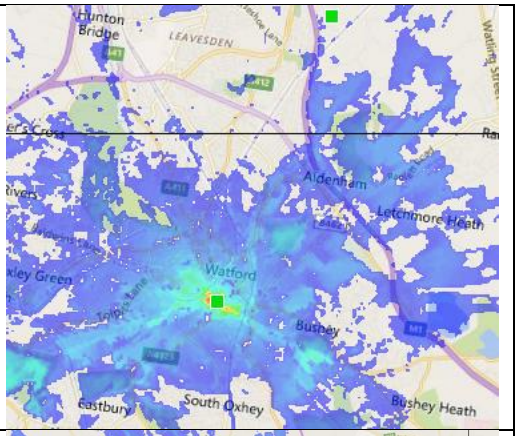
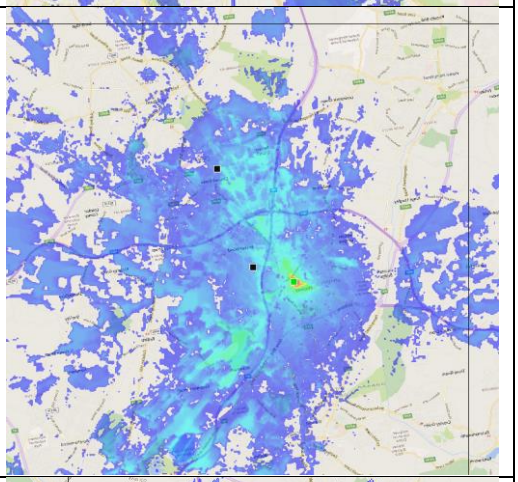
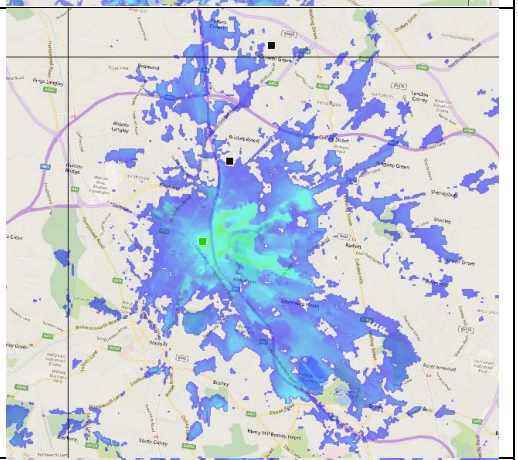
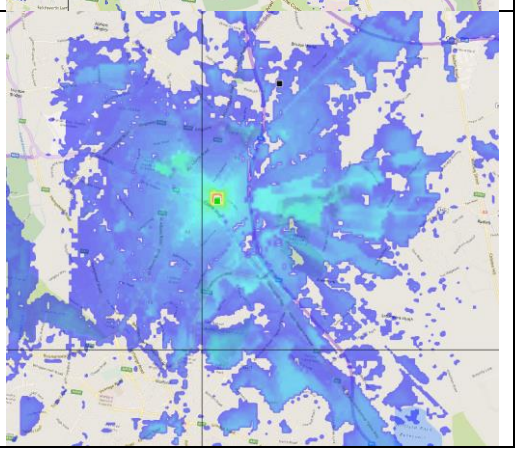
G	Wiggenhall Depot	Wiggenhall Road, Watford, Herts WD18 0FB	50	
H	Woodside Leisure Centre	Horseshoe Lane, Watford, Herts WD25 7HH	62	
I	Munden View	Garsmouth Way, Watford, Herts WD25 9DE	37	
J	Abbey View	Garsmouth Way, Watford, Herts WD25 9DZ	46	

Table 16: Single gateway simulations.

The Watford Town Hall and Central Leisure Centre sites were found to have the best-predicted coverage. These gateways are at high elevation and away from local clutter. No single site was found that met the criteria of 90% coverage. Multiple gateways are therefore required. Composite coverage predictions are shown in Table 17:

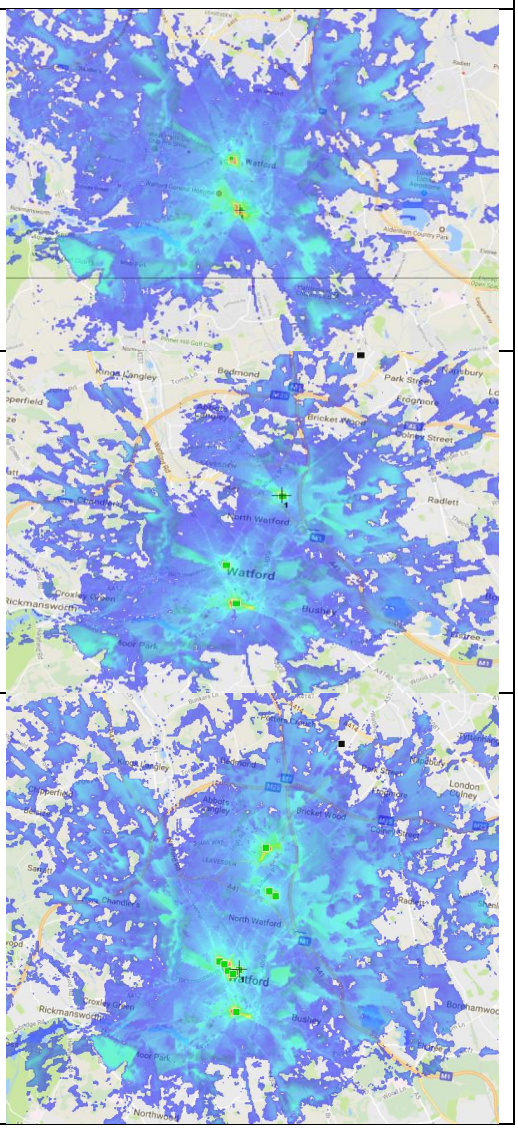
Number of GW	GWS	Coverage	Map
2	Gade Wiggenhall	71	
3	Gade Wiggenhall Abbey View	92	
10	ALL	99.6	

Table 17: Multiple gateway simulations.

The predictions show that 90% coverage of Watford borough can be achieved using three gateways sited at Gade, Wiggenhall and Abbey View. The coverage obeys a law of diminishing returns with only a 7% increase between operating three and ten gateways. Coverage may differ considerably due to changes in the local environment. For example, weather conditions or temporary structures that may obscure line of sight. The coverage predictions do not consider population density.

1.37 Analysing the Data from the Jersey Drive Survey

The Bullington model, calibrated with the Watford data, was applied to results from the Jersey drive survey. This resulted in a standard deviation of 11.32 dB. One explanation for the higher deviation is

the difference in terrain between Jersey and Watford. A lower standard deviation of 10.48 dB was obtained using a semi-empirical Okumura-Hata COST 231 model. A comparison of the two models is shown in Figure 64.

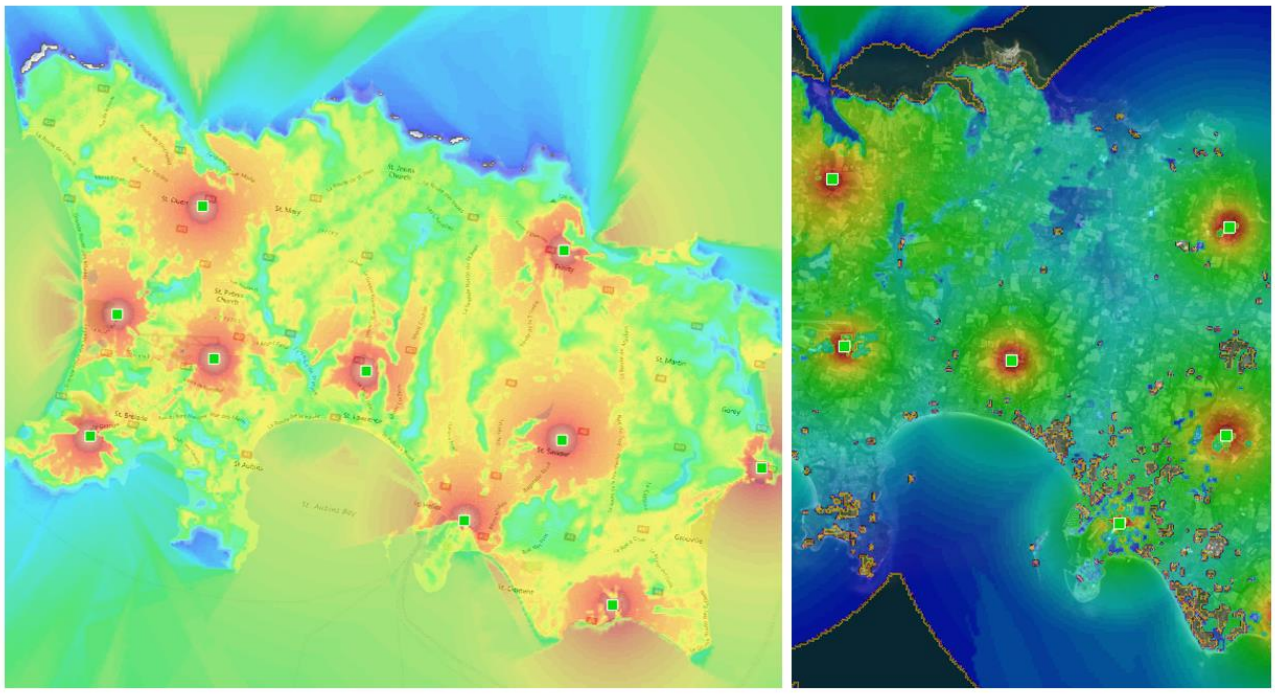


Figure 64: Comparison of Okumura-Hata COST 231 and Bullington models (Jersey).

Jersey has a total area of 120 km^2. It is served by ten LoRaWAN gateways. The Okumura-Hata COST 231 model predicts an island-wide coverage of 89%.

1.38 Simulation

Scalability is an important consideration for LoRaWAN networks. To assist Watford council with planning their network, a simulation was developed in Java. The objective of the simulation is to establish a baseline figure for packet loss. Real world packet loss will be higher due to losses incurred by the RF channel. A class and sequence diagram of the code is shown in Figure 65:

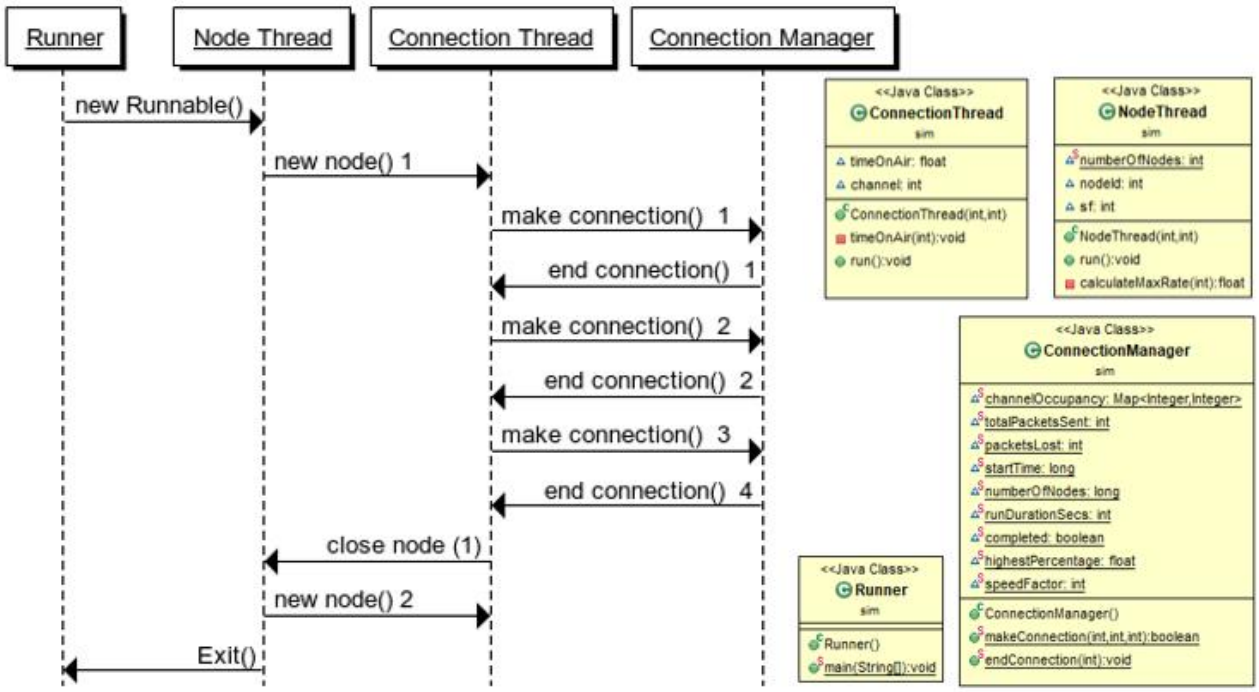


Figure 65: UML and sequence diagram (Network contention simulator).

1.38.1 Technical Details

The simulator code has the following features:

- The *runner* Java class is the primary test harness.
- This instantiates threads to simulate the behaviour of LoRaWAN nodes. These threads wait a random length of time then spawn connection requests.
- Connection threads attempt to acquire an available channel from the connection manager. Successful connections increment the channel occupancy count.
- When the connection drops, the occupancy count is decremented.
- If the selected channel is unavailable, the packet is marked as lost. The percentage of packets lost is recorded.

Table 18 lists assumptions made in creating the simulation.

Assumption	Detail
Randomised Duty Cycle	Each node was allocated a random duty cycle between 1 and 30 secs of airtime over a 24-hour period. This conforms to regulations in the ISM bands.
Fixed transmission rate	For the purposes of simplifying the simulation, each node was assumed to transmit at set intervals. An initial random delay of up to 12 hours was introduced to ensure separation between transmissions.
Fixed packet length	Time on-air figures for each spreading factor assumed transmission of a 105 byte payload [54]
Concurrent processing of incoming packets	Multi-channel gateways can process packets arriving simultaneously on different channels or on the same channel with different spreading factors. The exact number of incoming packets that can be processed simultaneously depends on the gateway hardware. For the purposes of this simulation, it was assumed that 40 packets can be processed at any one time.

Table 18: Assumptions made in creating the simulation.

1.38.2 Results

Results from the simulation are shown in Figure 66:

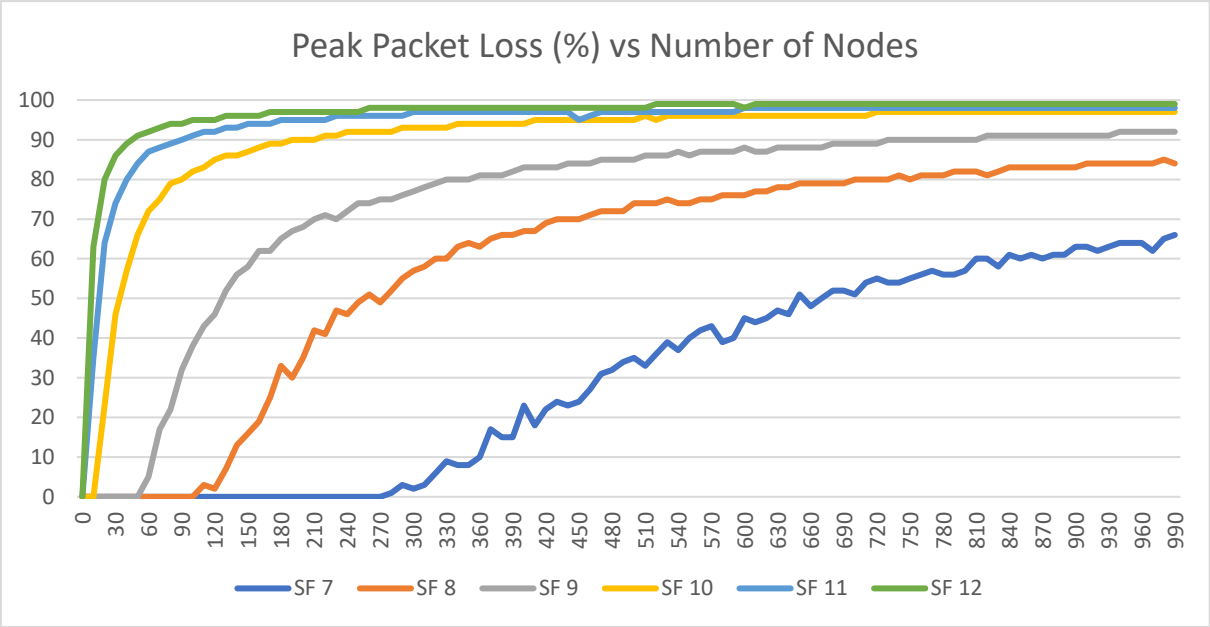


Figure 66: Results from simulation (Packet loss vs number of nodes).

Using spreading factor 7, approximately one thousand nodes can be served with a peak loss of 65%. Increasing the spreading factor to 12 reduces this number to below 30 nodes. It should be noted that this simulation is based on peak packet loss. This provides an indication of the worst-case scenario.

Conclusions

This study investigated LoRaWAN in an urban environment (Watford) and in mixed terrain (Jersey). Range, power consumption, and packet loss were quantified. A propagation model was validated and used in a real-world network planning exercise. Based on results from this research LoRaWAN is a strong candidate for Smart City applications. It is likely to face strong competition from 3GPP IoT standards, such as LTE-M and NB-LTE. Despite this, a large install base and competitive pricing models may ensure its future in privately operated networks.

Specific conclusions are listed under each project objective:

☑ **Quantify the range of LoRaWAN signals.**

In both the Watford and Jersey studies, a maximum range of 10 km was achieved. At this distance, there was a packet error rate of 80%. It is, therefore, recommended that devices are within 2 km of a gateway in an urban environment and 4 km in open terrain.

☑ **Determine rates of packet loss and how these vary with distance, time of day and spreading factor.**

Packet loss increases steadily with distance. IoT applications intolerant to high levels of loss will require service by multiple gateways. No relationship was found between packet loss and time of day. The mean distance between a node and gateway should be specified so that packets can be sent using a spreading factor of 7 or 8. This reduces time on air and network contention. Where possible the Adaptive Data Rate mechanism should be used.

☑ **Measure the power consumption of a typical LoRaWAN node and suggest ways to improve efficiency.**

Current drain of LoRa nodes was found to be low, typically around 40 mA. Power efficiency is strongly affected by the presence of onboard components, for example, a GPS receiver. Where possible, other geolocation techniques should be used such as Time Difference of Arrival (TDoA). Reducing duty cycle is an effective means of conserving power but some applications will require a minimum transmission rate. Use of sleep modes on the RN2483 chip and MCU may reduce power consumption but requires a suitable hardware or software interrupt.

☑ **Use field measurements to select, tune and validate a propagation model.**

An industry standard model (Bullington) was tuned and validated against data from the Watford drive survey. Comparison between predicted and measured RSSI values resulted in a standard deviation of 2.99 dB. An Okumura-Hata COST 231 model was used to model propagation in Jersey. This had a standard deviation of 10.48 dB.

☑ **Use the propagation model to estimate coverage for current gateways in Watford.**

Coverage predictions for Watford borough were made for both the Gade carpark and Wiggenhall depot sites. These were estimated to be 59% and 50% respectively. Far-field coverage is highly

dependent on elevation and local clutter. Where possible aim for a line of sight communication. As IoT nodes tend to be low to the ground, base stations should be mounted as high as possible (ideally greater than 30m). Gateways should be away from buildings and other obstructions.

☑ **Determine the number and location of gateways to achieve a borough-wide coverage of 90%.**

Good coverage can be achieved with relatively few LoRaWAN gateways. In the present study, a 21 km^2 area of Watford required three gateways to achieve a composite coverage of 90%. A similar level of coverage was obtained using 10 gateways across Jersey (120 km^2).

☑ **Investigate how LoRaWAN networks scale under increased load.**

A simulated LoRaWAN network was found to scale well supporting 1,000 nodes when using a spreading factor of 7. Rates of packet loss rapidly increased at higher spreading factors. For maximum scalability, multi-modem gateways should be used.

1.39 Future Work

The following suggestions are made for future research:

Propagation Modelling

- Use a building layer data to supplement Digital Terrain Maps.
- Model indoor penetration and combine with outdoor signal strength estimates to produce hybrid coverage maps.
- Use population density data (vectors) to model LoRaWAN coverage per population in Watford.

Capacity Planning

- Enhance simulation to investigate network capacity as a function of the duty cycle.
- Enhance simulation to include multiple gateways, real-world traffic data, and support for Adaptive Data Rate.
- Perform contention measurements in the field to validate the results of the simulation.

Spectrum Management

- Since LoRaWAN operates in the unlicensed ISM bands a detailed survey of interference including out of band interference should be undertaken. This would allow estimates of the capacity of the 868 MHz channel.

Enhancements to the Test Node

- An accelerometer could be used to determine if the test vehicles were moving.
- Investigate alternative location sensing technologies with the aim of reducing power consumption.
- Create a permanent mobile installation so that measurements can be repeated on a regular basis.

Expanded Measurement Campaign

- Investigate waveguiding effects by comparing urban measurements with those in open areas.
- Perform measurements of indoor and subterranean penetration. These can be used to validate assumptions about material types and building geometry.
- Quantify the effect of the Doppler shift on transmissions.

REFERENCES

[1] "The Citizen Flood Network," [Online]. Available: https://www.postscapes.com/citizen-flood-detection-network/. [Accessed 02 02 2018].

[2] Garner, "Leading the IoT - Gartner Insights on How to Lead," [Online]. Available: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf. [Accessed 01 01 2018].

[3] P. K. a. M. S. U. Raza, "Low Power Wide Area Networks: An Overview," *IEEE Communications Surveys & Tutorials,* vol. 19, no. 2, 2017.

[4] M. G. A. Al-Fuqaha, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,," *IEEE Communications Surveys & Tutorials,,* vol. 17, pp. 2347-2376,, 2015.

[5] "Internet of Things Protocols and Standards," Cse.wustl.edu, [Online]. Available: https://www.cse.wustl.edu/~jain/cse570-15/ftp/iot_prot/. [Accessed 12 11 2017].

[6] F. Rayal, "Mobile and Wide-Area IoT: LPWA and LTE connectivity," January 2016. [Online]. Available: http://www.weightless.org/about/mobile-experts-group-lpwan-report/NTZiMy9tZXhwLWxwd2EtMTYgZXhlYyBzdW1tYXJ5LnBkZg==. [Accessed 27 06 2018].

[7] Link Labs, "Lora FAQS," 2018. [Online]. Available: https://www.link-labs.com/blog/lora-faqs. [Accessed 06 07 2018].

[8] "LoRa Protocol Stack-LoRa Physical layer, LoRa MAC layer," RF Wireless World, [Online]. Available: http://www.rfwireless-world.com/Tutorials/LoRa-protocol-stack.html. [Accessed 03 02 2018].

[9] "LoRa & LoRaWAN Primer," LEVERAGE, 19 06 2017. [Online]. Available: https://www.leverege.com/research-papers/lora-lorawan-primer. [Accessed 17 01 2018].

[10] "LoRaWAN vs Haystack," Slideshare, [Online]. Available: https://www.slideshare.net/haystacktech/lorawan-vs-haystack. [Accessed 1 12 2017].

[11] P. Tracy, "What is LoRa and how does it work?," Enterprise IoT Insights, [Online]. Available: https://enterpriseiotinsights.com/20160819/internet-of-things/lora-lorawan-tag31-tag99. [Accessed 12 11 2017].

[12] "Wikipedia - Chirp," [Online]. Available: https://en.wikipedia.org/wiki/Chirp. [Accessed 02 04 2018].

[13] "LoRaWAN Part 1: How to Get 15 km Wireless," DigiKey, [Online]. Available: https://www.digikey.com/en/articles/techzone/2016/nov/lorawan-part-1-15-km-wireless-10-year-battery-life-iot. [Accessed 01 03 2018].

[14] M. Knight, "Reversing Lora," [Online]. Available: https://static1.squarespace.com/static/54cecce7e4b054df1848b5f9/t/57489e6e07eaa0105215dc6c/1464376943218/Reversing-Lora-Knight.pdf. [Accessed 01 01 2018].

[15] T. Telkamp, "LoRa crash course," YouTube, [Online]. Available: https://www.youtube.com/watch?v=T3dGLqZrjIQ. [Accessed 12 12 2017].

[16] A. Augustin, "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things," 09 09 2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5038744/. [Accessed 03 02 2018].

[17] "All you need to know about regulation on RF 868MHZ for LPWAN," DISK91.COM - TECHNOLOGY BLOG, [Online]. Available: https://www.disk91.com/2017/technology/internet-

of-things-technology/all-what-you-need-to-know-about-regulation-on-rf-868mhz-for-lpwan/. [Accessed 02 01 201].

[18] “Lora Device classes - Three device classes receive slot timing.,” Research Gate, [Online]. Available: https://www.researchgate.net/figure/Three-device-classes-receive-slot-timing_fig8_315505158. [Accessed 03 02].

[19] “The IoT Spurs the Pursuit of Low Power,” Systemdesign.altera.com, [Online]. Available: https://systemdesign.altera.com/the-iot-spurs-the-pursuit-of-low-power/. [Accessed 21 11 2017].

[20] “Pavegen - The Next Step,” Pavegen, [Online]. Available: http://www.pavegen.com/. [Accessed 05 03 2018].

[21] K. Hub, “Indoor Energy Harvesting | Harvesting Ambient Light | GCell,” [Online]. Available: http://gcell.com/knowledge-hub/indoor-energy-harvesting. [Accessed 03 03 2018].

[22] B. M. I. ,. R. V. I. a. C. G. Lluís Casals ID, “Modeling the Energy Performance of LoRaWAN,” *sensors,* 16 October 2017.

[23] R. Merritt, “Arm Grabs Stream for IoT Services,” 13 06 2018. [Online]. Available: https://www.eetimes.com/document.asp?doc_id=1333379. [Accessed 01 07 2018].

[24] “LoRa server,” forum.loraserver.io, [Online]. Available: https://docs.loraserver.io/loraserver/overview/. [Accessed 01 01 2018].

[25] “LoRa Shield for Arduino,” Dragino, [Online]. Available: http://www.dragino.com/products/module/item/102-lora-shield.html. [Accessed 02 11 2017].

[26] “Raspberry PI Link Labs LoRaWAN Gateway,” GitHub, [Online]. Available: https://github.com/mirakonta/Raspberry-PI-Link-Labs-LoRaWAN-Gateway. [Accessed 06 03 2018].

[27] “Installing a Lora Gateway,” [Online]. Available: http://bikealive.nl/lora-gateway.html. [Accessed 02 01 2018].

[28] “TTN St Albans,” [Online]. Available: https://www.thethingsnetwork.org/community/st-albans/. [Accessed 02 08 2016].

[29] “EveryNet gateway - Briefing,” [Online]. Available: http://everynet.com/wp-content/uploads/2017/06/everynet-Gateway-V2.1-Product-Brief.pdf. [Accessed 04 04 2018].

[30] “Vertical antennas A10-868 864-876 MHz,” duplexers, [Online]. Available: http://www.duplexers.eu/catalog/Pro/antennas/a10_868/. [Accessed 12 05 2018].

[31] “LoRaWAN 868MHz Antenna Test,” Coredump.ch, [Online]. Available: https://www.coredump.ch/2017/04/13/lorawan-868mhz-antenna-test-part-1/. [Accessed 01 02 2018].

[32] “Understanding Antenna Specifications and Operation - Application Note AN-00501,” Linx, [Online]. Available: https://www.linxtechnologies.com/wp/wp-content/uploads/an-00501.pdf. [Accessed 19 04 2018].

[33] “DG8SAQ Low Cost Vector Network Analyzer VNWA 3EC with Accessories,” Sdr-kits.net, [Online]. Available: https://sdr-kits.net/VNWA-3EC. [Accessed 14 11 2018].

[34] “LoRaWAN NETWORK,” Hyper Tech, [Online]. Available: http://hypertech.co.il/product/iotlpwan-networks/lorawan-network/. [Accessed 09 06 2018].

[35] EveryNet, “The EveryNet API,” [Online]. Available: https://www.thingsconnected.net/support/. [Accessed 01 01 2018].

[36] “TTN Mapper APp,” [Online]. Available: https://play.google.com/store/apps/details?id=com.jpmeijers.ttnmapper&hl=en_GB. [Accessed 03 02 2018].

[37] “Database Third Normal Form Explained in Simple English,” Essential SQL, [Online]. Available: https://www.essentialsql.com/get-ready-to-learn-sql-11-database-third-normal-form-explained-in-simple-english/].. [Accessed 19 02 2018].

[38] S. R. Simon, “Antennas and Propagation for Wireless Communication Systems,” New York, John Wiley & Sons, 1999.

[39] ATDI, “Lee Criteria,” [Online]. Available: http://www.atdi-maps.com/DOC/Radio%20Propagation%20in%20ATDI%20tools.pdf. [Accessed 01 01 2018].

[40] J. Bush, “The Significance of Rayleigh Fading in Coverage Measurements,” Berkeley Varitronics Systems, Inc., [Online]. Available: https://www.rfglobalnet.com/doc/the-significance-of-rayleigh-fading-in-covera-0001. [Accessed 03 02 2018].

[41] “Testing USB power banks,” 20 03 2016. [Online]. Available: https://stephenstuff.wordpress.com/2016/03/20/testing-usb-power-banks/.

[42] J. Hobson, “Tricking a USB Power Supply,” Hackaday, [Online]. Available: https://hackaday.com/2013/11/08/tricking-a-usb-power-supply/. [Accessed 05 01 2018].

[43] “USB battery pack keep-alive load,” SOTABEAMS, [Online]. Available: https://www.sotabeams.co.uk/usb-battery-pack-keep-alive-load/. [Accessed 14 12 2017].

[44] “Sandbox Jersey,” [Online]. Available: https://www.digital.je/choose-jersey/sandbox-jersey/.

[45] “RN2483 (433/868 MHz) LoRa Modem Datasheet,” Mircochip, 2015. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70005219A.pdf. [Accessed 10 11 2018].

[46] R. Wilson, “The IoT Spurs the Pursuit of Low Power,” [Online]. Available: https://systemdesign.altera.com/the-iot-spurs-the-pursuit-of-low-power/. [Accessed 03 03 2018].

[47] M. B. V. R. G. C. Casals L1, “Modeling the Energy Performance of LoRaWAN,” 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/29035347. [Accessed 02 01 2018].

[48] “Cisco Lorawan Gateway,” CISCO, [Online]. Available: https://www.cisco.com/c/en/us/products/routers/wireless-gateway-lorawan/index.html. [Accessed 05 03 2018].

[49] Stream Technologies, “Glasgow pioneers IoT connectivity with new LoRa™ network,” [Online]. Available: https://censis.org.uk/censis_projects/glasgow-pioneers-iot-connectivity-with-new-lora-network/. [Accessed 02 01 2018].

[50] “Okumura model,” En.wikipedia.org, [Online]. Available: https://en.wikipedia.org/wiki/Okumura_model. [Accessed 12 03 2018].

[51] “LoRa Network Planning and Sizing,” ATDI, [Online]. Available: http://www.atdi.com/lora-network-planning-and-sizing/. [Accessed 09 01 2018].

[52] ATDI, “Webinar: Propagation modelling,” [Online]. Available: https://www.atdi.co.uk/webinar-propagation-modelling/. [Accessed 03 02 2018].

[53] S. I. Popoola, “Standard Propagation Model Tuning for Path Loss Predictions in Built-Up Environments,” Springer, 2017, p. 07.

[54] “A DIY low-cost LoRa gateway,” [Online]. Available: http://cpham.perso.univ-pau.fr/LORA/RPIgateway.html. [Accessed 08 05 2018].

[55] “Lorawan World Record,” The Things Network Global Team, 08 09 2017. [Online]. Available: https://www.thethingsnetwork.org/article/ground-breaking-world-record-lorawan-packet-received-at-702-km-436-miles-distance. [Accessed 12 01 2018].

Appendix A
Comparison Between LoRa with WSPR modulation

There have been numerous attempts to transmits LoRa packets over very long distances. The current record is 702 km [55]. This was set using a weather balloon at high altitude. Protocols exist which have been specifically designed for extremely long propagation. Examples of these are FT8, JT9 and Weak Signal Propagation Report (WSPR) modulation. It is instructive to compare trade-offs in their design to those present in LoRa.

As part of this project, WSPR packets were sent using a WSPRLite beacon attached to a 40-metre long dipole antenna. Transmit power was 200 mW at a frequency of 14.096 MHz. Packets were then received by the Weak Signal Propagation Reporter Network. This is a worldwide network of internet linked receivers. The location of the received packets is uploaded to the website WSPRNET.org. Transatlantic ranges of over 6,000 km were achieved. Received packets or *spots* are shown on the map below:

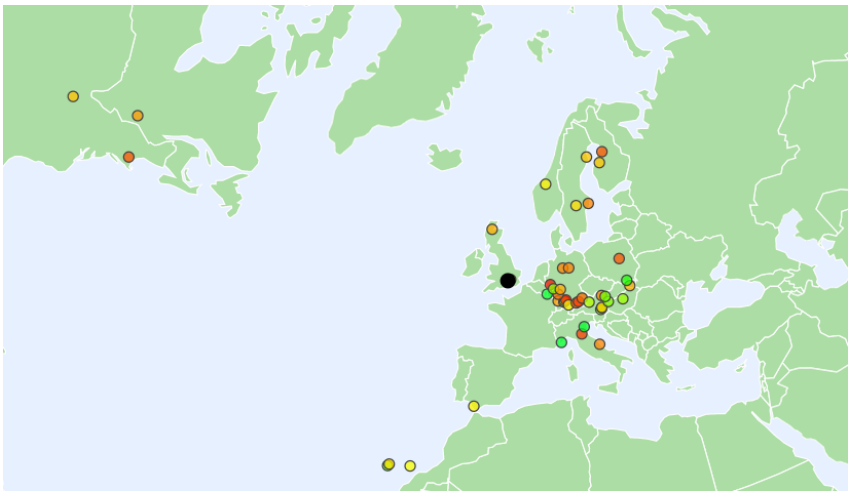


Figure 67: WSPR Spots

WSPR modulation uses frequency shift keying with a low symbol rate. 50 bits of data are transmitted over 110 seconds. Bandwidth and power are comparable to LoRa signals. WSPR also employs similar redundancy and forward error correction schemes to LPWANs. The primary difference is the mode of propagation. WSPR transmissions are in the HF portion of the radio spectrum (3 to 30 MHz). At these frequencies, signals are refracted off layers in the ionosphere. This allows for worldwide communication. LoRa and other LPWAN technology use ground wave propagation so ranges are limited to just beyond the line of sight. Interestingly, there have been reports of LoRa and Sigfox signals being enhanced by tropospheric ducting.

Appendix B
RESTful API Specification

This section describes the REST services created as part of this work to access LoRa data. All the services below have the same URL prefix

/RPC/API

The following table lists all the methods created:

URL	Method	Description
/frames	GET	Returns a filtered array of JSON objects representing LoRa data frames stored in the data source
/frames_secure	GET	As above but secured by application key
/config	GET	Returns an HTML page containing a form to update configuration information
/postform	POST	Returns an HTML page confirming configuration values have been set
/map	GET	Returns an Google Maps plot of LoRa frames received set

A more detailed specification of each service is given below:

URL:	**/frames**
Description:	Returns an filtered array of JSON objects representing LoRa data frames stored in the data source
Method:	GET
Request Data:	Query Parameters: TimeInMins - This is the number of minutes' worth of data to return. e.g. /RPC/api/frames?TimeInMins=60 SpreadingFactor - This is the spreading factor used to filter the data (Use 0 to return all)
Response Data:	Status – 200 (OK) Content Type - application/json Each element in the array is an object defining a LoRa frame

URL	**/frames_secure**
Description:	Returns an filtered array of JSON objects representing LoRa data frames stored in the data source (Secured)
Method:	GET
Request Data:	Query Parameters: TimeInMins - This is the number of minutes' worth of data to return. e.g. /RPC/api/frames?TimeInMins=60 SpreadingFactor - This is the spreading factor used to filter the data (Use 0 to return all) Key - This is a mandatory application API key e.g. c225d602-bcc8-4621-82c1-dd4d6ea579c8
Response Data:	If authenticated then as per /frames If unauthenticated throws HTTP status code 403 Forbidden

URL:	**/config**
Description:	Returns an HTML page containing a form to update configuration information
Method:	GET
Request Data:	None
Response Data:	Status – 200 (OK) Content Type - TEXT_HTML Returns user input form to change path to the CSV datafile and the amount of data to be returned (Number of minutes) 

URL:	**/postform**
Description:	Returns an HTML page confirming configuration values have been set
Method:	POST
Request Data:	None
Response Data:	Status – 200 (OK) Content Type - TEXT_HTML Example: Displays the text "values updated" to confirm change of user configuration

URL:	**/map**
Description:	Returns a Google Maps plot of LoRa frames received set
Method:	GET
Request Data:	Query Parameters: TimeInMins - This is the number of minutes' worth of data to return. SpreadingFactor - The spreading factors to display. Use 0 to display all factors e.g. RPC/api/map?timeinmins=5000&spreadingfactor=0
Response Data:	Status – 200 (OK) Content Type - TEXT_HTML Example: Displays the text "values updated" to confirm change of user configuration

Appendix C
Software

Software Design Techniques used

The Jersey / JAX-RS framework was used to simplify the creation of a LoRaWAN RWS server. Low-level code required to create a client-server is supplied by the toolkit. Java annotations are then used to implement the required API. The following example shows the method signature of a GET service with an URL or query parameter called key:

```
@GET
@Path ("/getsecure")
@Produces (MediaType.APPLICATION_JSON)
public List<Frame> getdata(@QueryParam("key") String key) throws JSONException {
```

Use of Jersey / JAX-RS.

Lambda expressions were used to efficiently parse CSV data and map to a collection or array of frame objects. An example of this is shown in the following code:

```
frames = Files.readAllLines(Paths.get(filePath, fileName)).stream()
        .map(s -> new Frame(s.split(",")[0], s.split(",")[1], s.split(",")[2], s.split(",")[3],
                s.split(",")[4], s.split(",")[5], s.split(",")[6], s.split(",")[7], s.split(",")[8],
                .....
                s.split(",")[34], s.split(",")[35], s.split(",")[36], s.split(",")[37]))
        .filter(x -> inLastMin(x.getTime())).collect(Collectors.toList());
```

Use of Lambda expressions in the RWS server.

Bespoke Web Application (GUI)

The Java class below is the main code used in a webserver to display and filter LoRaWAN data from the ThingsConnected platform. The Google Maps API is used to map data as path profiles or a heat map.

```
package api;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Collectors;

import javax.ws.rs.Consumes;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
```

```
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;

import org.json.JSONException;

@Path("/")
public class RESTService {

	// private static String filePath = "//home//pi/app";
	private static String filePath = "c:\\";
	private static String fileName = "out.csv";
	private static String timeInMins = "9999";

	@GET
	@Path("/frames_secure")
	@Produces(MediaType.APPLICATION_JSON)
	public List<Frame> getFramesSecure(@QueryParam("timeinmins") String timeInMins,
			@QueryParam("spreadingfactor") String spreadingFactor, @QueryParam("key") String
apiKey)
			throws JSONException {

		if (apiKey.equals("DFJJ-DFD-223-SD-SDSD")) {
			return getFrames(timeInMins, spreadingFactor);

		}
		return null;
	}

	@GET
	@Path("/frames")
	@Produces(MediaType.APPLICATION_JSON)
	public List<Frame> getFrames(@QueryParam("timeinmins") String timeInMins,
			@QueryParam("spreadingfactor") String spreadingFactor) throws JSONException {

		List<Frame> frames = null;

		try {

			frames = Files.readAllLines(Paths.get(filePath, fileName)).stream()
					.map(s -> new Frame(s.split(",")[0], s.split(",")[1],
s.split(",")[2], s.split(",")[3],
s.split(",")[4], s.split(",")[5], s.split(",")[6],
s.split(",")[7], s.split(",")[8],
s.split(",")[9], s.split(",")[10], s.split(",")[11],
s.split(",")[12], s.split(",")[13],
s.split(",")[14], s.split(",")[15], s.split(",")[16],
s.split(",")[17], s.split(",")[18],
s.split(",")[19], s.split(",")[20], s.split(",")[21],
s.split(",")[22], s.split(",")[23],
s.split(",")[24], s.split(",")[25], s.split(",")[26],
s.split(",")[27], s.split(",")[28],
s.split(",")[29], s.split(",")[30], s.split(",")[31],
s.split(",")[32], s.split(",")[33],
s.split(",")[34], s.split(",")[35], s.split(",")[36],
s.split(",")[37]))
					.filter(x -> filterData(x.getTime(), timeInMins, x.getDatr(),
spreadingFactor))
					.collect(Collectors.toList());

		} catch (IOException e) {
			e.printStackTrace();
		}

		return frames;
	}

	/**
	 * @param aDate
	 * @param timeInMins
	 * @return
	 *
	 *		Filter the LoRa data by time
	 */
```

```
	private boolean filterData(String aDate, String timeInMins, String aDatr, String spreadingFactor) {

		SimpleDateFormat formatnow = new SimpleDateFormat("MMM dd yyyy HH:mm:ss");
		Date date1 = null;
		try {
			date1 = formatnow.parse(aDate.substring(4, aDate.length() - 15));
		} catch (Exception e) {
			return false;
		}

		// filter on time
		if ((System.currentTimeMillis() - date1.getTime()) < (Integer.parseInt(timeInMins) * 1000 *
60)) {

			// filter on spreading factor
			if (spreadingFactor.equals("0") || aDatr.equals("SF" + spreadingFactor + "BW125")) {
				return true;
			}

		}

		// set to true for testing
		return false;

	}

	@GET
	@Path("/map")
	@Produces(MediaType.TEXT_HTML)
	public String getMap(@QueryParam("timeinmins") String timeInMins,
			@QueryParam("spreadingfactor") String spreadingFactor) throws JSONException {

		List<Frame> frames = getFrames(timeInMins, spreadingFactor);

		String str = "<!DOCTYPE html>" + //
				"<html>" + //
				"  <head>" + //
				"    <title>LoRa Coverage Map</title>" + //
				"    <meta name=\"viewport\" content=\"initial-scale=1.0\">" + //
				"    <meta charset=\"utf-8\">" + //
				"    <style>" + //
				"      /* Always set the map height explicitly to define the size of the div"
+ //
				"       * element that contains the map. */" + //
				"      #map {" + //
				"        height: 100%;" + //
				"      }" + //
				"      /* Optional: Makes the sample page fill the window. */" + //
				"      html, body {" + //
				"        height: 100%;" + //
				"        margin: 0;" + //
				"        padding: 0;" + //
				"      }" + //
				"    </style>" + //
				"  </head>" + //
				"  <body>" + //
				"    <div id=\"map\"></div>" + //
				"    <script>" + //
				"      var map;" + //
				"      function initMap() {" + //
				"        map = new google.maps.Map(document.getElementById('map'), {" + //
				"          center: {lat: 51.73113, lng: -0.36209}," + //
				"          zoom: 15" + //
				"        });" + //
				"        " + //
				"              " + //
				"              ";

		ArrayList<String> data = new ArrayList<String>();
		for (int i = 0; i < frames.size(); i++) {

			// choose colour depending on spreading factor

			String strokeColor = "";
```

```
                        if (frames.get(i).getDatr().equals("SF7BW125")) {
                                strokeColor = "#7DFF33";
                        }

                        if (frames.get(i).getDatr().equals("SF8BW125")) {
                                strokeColor = "#33FF38";
                        }

                        if (frames.get(i).getDatr().equals("SF9BW125")) {
                                strokeColor = "#33FF78";
                        }

                        if (frames.get(i).getDatr().equals("SF10BW125")) {
                                strokeColor = "#F8FF33";
                        }

                        if (frames.get(i).getDatr().equals("SF11BW125")) {
                                strokeColor = "#FF9233";
                        }

                        if (frames.get(i).getDatr().equals("SF12BW125")) {
                                strokeColor = "#FF4133";
                        }

                        str += "          var flightPath = new google.maps.Polyline({" + //
                                        "              path: [{lat: " + frames.get(i).getGw_lat() + ", lng: "
+ frames.get(i).getGw_lon()
                                        + "},{lat: " + frames.get(i).getNode_lat() + ", lng:" +
frames.get(i).getNode_long() + "}]," + //
                                        "              geodesic: true," + //
                                        "              map: map," + //
                                        "              strokeColor: ' " + strokeColor + "'," + //
                                        "              strokeOpacity: 1.0," + //
                                        "              strokeWeight: 1" + //
                                        "            });";

                }

                str += "      }" + //
                                "    </script>" + //
                                "    <script
src=\"https://maps.googleapis.com/maps/api/js?key=AIzaSyAPA4oCk_64JNHmu64WTmLqq8e1--21wYU&callback=initMap\""
                                + //
                                "    async defer></script>" + //
                                "  </body>" + //
                                "</html>";

                return str;
        }

        @GET
        @Path("/config")
        @Produces(MediaType.TEXT_HTML)
        public String writeForm() {

                String str = "<!DOCTYPE html>" + "<html>" + //
                                "<body>" + //
                                "<form action=\"postform\" method=\"post\" >" + //
                                " Path:<br>" + //
                                "  <input type=\"text\" name=\"fpath\" value=\"" + filePath + " \">" + //
                                "  <br>" + //
                                " Name:<br>" + //
                                "  <input type=\"text\" name=\"fname\" value=\"" + fileName + "\">" + //
                                "  <br>" + //
                                " Number of Mins:<br>" + //
                                "  <input type=\"text\" name=\"nsecs\" value=\"" + timeInMins + "\">" + //
                                "  <br><br>" + //
                                "  <input type=\"submit\" value=\"Submit\">" + //
                                "</form> " + //
                                "<p>Click submit to update</p>" + //
                                "</body>" + //
                                "</html>";
```

```
                return str;
        }

        @POST
        @Path("/postform")
        @Produces(MediaType.TEXT_HTML)
        @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
        public String writeFormResult(@FormParam("fpath") String fpath, @FormParam("fname") String fname,
                        @FormParam("nsecs") String nsecs) {

                filePath = fpath.trim();
                fileName = fname.trim();
                timeInMins = nsecs.trim();

                String str = "<!DOCTYPE html>" + "<html>" + //
                                "<body>" + "values updated!" + "</body>" + //
                                "</html>";

                return str;
        }

}
```

Network Contention Simulation

The classes below implement a Java thread-based network contention simulation.

```
package sim;

public class ConnectionThread implements Runnable {

        float timeOnAir;
        int channel;

        public ConnectionThread(int channel, int sf) {
                // calcualate time on air
                timeOnAir(sf);
                this.channel = channel;
        }

        private void timeOnAir(int sf) {

                switch (sf) {
                case 7:
                        this.timeOnAir = 0.05f ;
                        break;
                case 8:
                        this.timeOnAir = 0.14f ;
                        break;
                case 9:
                        this.timeOnAir = 0.29f ;
                        break;
                case 10:
                        this.timeOnAir = 1.09f  ;
                        break;
                case 11:
                        this.timeOnAir = 2.5f ;
                        break;
                case 12:
                        this.timeOnAir = 4.23f ;
                        break;

                default:
                        break;
                }

        }

        @Override
        public void run() {
```

```
                // System.out.println("starting sleep");

                try {
                        Thread.sleep((long) Math.floor(timeOnAir * 1000 / ConnectionManager.speedFactor ));
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                // System.out.println("waking from sleep");
                ConnectionManager.endConnection(channel);
        }

}

package sim;

public class ConnectionThread implements Runnable {

        float timeOnAir;
        int channel;

        public ConnectionThread(int channel, int sf) {
                // calcualate time on air
                timeOnAir(sf);
                this.channel = channel;
        }

        private void timeOnAir(int sf) {

                switch (sf) {
                case 7:
                        this.timeOnAir = 0.05f ;
                        break;
                case 8:
                        this.timeOnAir = 0.14f ;
                        break;
                case 9:
                        this.timeOnAir = 0.29f ;
                        break;
                case 10:
                        this.timeOnAir = 1.09f  ;
                        break;
                case 11:
                        this.timeOnAir = 2.5f ;
                        break;
                case 12:
                        this.timeOnAir = 4.23f ;
                        break;

                default:
                        break;
                }

        }

        @Override
        public void run() {

                // System.out.println("starting sleep");

                try {
                        Thread.sleep((long) Math.floor(timeOnAir * 1000 / ConnectionManager.speedFactor ));
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                // System.out.println("waking from sleep");
                ConnectionManager.endConnection(channel);
        }

}
```

```
package sim;

import java.util.HashMap;
import java.util.Map;

public class ConnectionManager {
	static Map<Integer, Integer> channelOccupancy = new HashMap<Integer, Integer>();

	static int totalPacketsSent = 0;
	static int packetsLost = 0;
	static long startTime = 0;
	static long numberOfNodes = 0;
	static int runDurationSecs = 0;
	static boolean completed = false;
	static float highestPercentage = 0;
	static int speedFactor = 0;

	public static synchronized boolean makeConnection(int channel, int sf, int nodeId) {

		float percentage = 0f;

		if (totalPacketsSent != 0) {
			percentage = (float) (packetsLost * 100 / totalPacketsSent);
			if (percentage > highestPercentage) {
				highestPercentage = percentage;
			}
		}

		// check if target duration meet
		long duration = ((System.currentTimeMillis() - startTime) / 1000) * speedFactor;

		if (duration > runDurationSecs) {

			if (!completed) {

				System.out.println(
						"completed run," + runDurationSecs + "," + sf + "," +
numberOfNodes + "," + highestPercentage);

				completed = true;
			}
			return false;
		}

		// increment channel
		if (channelOccupancy.get(channel) == null) {
			channelOccupancy.put(channel, 0);
		}

		int count = (int) channelOccupancy.get(channel);
		totalPacketsSent++;

		if (count++ == 40) {
			// System.out.println("lost packet");
			packetsLost++;
		} else {

			// successful connection
			channelOccupancy.put(channel, count++);

			// start TimeOut Thread
			(new Thread(new ConnectionThread(channel, sf))).start();
		}
		return true;
	}

	public static synchronized void endConnection(int channel) {

		// System.out.println("end connection on channel:" + channel);

		// decrease channel count

		int count = (int) channelOccupancy.get(channel);
```

```
                // successful connection
                channelOccupancy.put(channel, count--);
        }

}

package sim;

import java.sql.Connection;
import java.util.Random;

public class NodeThread implements Runnable {

        static int numberOfNodes = 0;

        int nodeId;
        int sf;

        public NodeThread(int nodeId, int sf) {
                this.sf = sf;
                this.nodeId = nodeId;

        }

        @Override
        public void run() {

                boolean continueThread = true;

                numberOfNodes++;

                // wait a random time
                Random r0 = new Random();
                int min0 = 0;
                int max0 = ConnectionManager.runDurationSecs / 2;
                int random_delay0 = r0.nextInt((max0 - min0) + 1) + min0;

                // random offset
                try {
                        Thread.sleep((long) (random_delay0 * 1000 / ConnectionManager.speedFactor));
                } catch (InterruptedException e1) {

                }

                while (continueThread) {

                        // select a random channel
                        Random r = new Random();
                        int min = 1;
                        int max = 40;
                        int channel = r.nextInt((max - min) + 1) + min;

                        min = 1;
                        max = ConnectionManager.runDurationSecs / 2;
                        int wait = r.nextInt((max - min) + 1) + min;

                        float minTimeToWait = calculateMaxRate(sf);

                        try {
                                Thread.sleep((long) (minTimeToWait * 1000 / ConnectionManager.speedFactor));
                        } catch (InterruptedException e1) {
                                // TODO Auto-generated catch block
                                e1.printStackTrace();
                        }

                        // if (nodeId == 1) {
                        // System.out.println("making connection");
                        // }

                        continueThread = ConnectionManager.makeConnection(channel, sf, nodeId);
                }

                numberOfNodes--;
        }
```

```
	private float calculateMaxRate(int sf2) {
		// TODO Auto-generated method stub
		float timeAir = 0;

		switch (sf) {
		case 7:
			timeAir = 0.05f;
			break;
		case 8:
			timeAir = 0.14f;
			break;
		case 9:
			timeAir = 0.29f;
			break;
		case 10:
			timeAir = 1.09f;
			break;
		case 11:
			timeAir = 2.5f;
			break;
		case 12:
			timeAir = 4.23f;
			break;

		default:
			break;
		}

		Random r = new Random();
		int min = 1;
		int max = 30;
		float maxIn24 = (float) (r.nextInt((max - min) + 1) + min);
		int secondsInDay = 86400;

		return secondsInDay / (maxIn24 * timeAir);
	}
}

package sim;

import java.util.HashMap;

public class Runner {

	public static void main(String args[]) {

		int minNodes = 1;
		int maxNodes = 1000;
		int stepSize = 10;
		int runDurationSecs = 60 * 60;
		int speedFactor = 1000000;

		System.out.println("Status, duration, sf, number of nodes");

		for (int sf = 7; sf <= 12; sf++) {

			for (int run = minNodes; run <= maxNodes; run += stepSize) {

				ConnectionManager.speedFactor = speedFactor;
				ConnectionManager.totalPacketsSent = 0;
				ConnectionManager.packetsLost = 0;
				ConnectionManager.runDurationSecs = runDurationSecs;
				ConnectionManager.completed = false;
				ConnectionManager.startTime = System.currentTimeMillis();
				ConnectionManager.numberOfNodes = run;
				ConnectionManager.highestPercentage = 0;
				ConnectionManager.channelOccupancy = new HashMap<Integer, Integer>();

				// System.out.println("starting run for " + run + " nodes at spreading factor
"
				// + sf);

				for (int i = 1; i <= run; i++) {
```

```
                                        (new Thread(new NodeThread(i, sf))).start();
                                }

                                // wait for all threads to finish before next run
                                while (Thread.activeCount() > 1) {
                                        // System.out.println(Thread.activeCount());
                                }

                                if (ConnectionManager.highestPercentage > 99) {
                                        break;
                                }

                                // try {
                                // Thread.sleep((runDurationSecs + 10) * 1000 /
ConnectionManager.speedFactor);
                                // } catch (InterruptedException e) {
                                // // TODO Auto-generated catch block
                                // e.printStackTrace();
                                // }
                        }
                }
        }
}
```

Application Server (Things Connected)

The following JavaScript code is use with Node.js to implement an application server capable of receiving LoRaWAN packets from the Things Connected platform. Contains example code from EveryNet.

```
const jayson = require('jayson');
const atob = require('atob');
const cors = require('cors');
const jsonParser = require('body-parser').json;
const btoa = require('btoa');
const fd = require('fd');
const connect = require('connect');
const app = connect();
var fs = require("fs");
var csvWriter = require('csv-write-stream');
var writer = csvWriter();
var geolib = require('geolib');
var date = new Date().toISOString().replace(/T/, '_').replace(/-/g, '').replace(/:/g, '').replace(/..+/, '');
var filepath = './log_data_' + date;

var server = jayson.server({
uplink: function(args, callback) {

var buffer = new Buffer(args.payload, 'base64');

// get coordinates of node from payload
var coords = buffer.toString('ascii').split(',');

// set location of gateway
var gw_latitude  = 51.731056;
var gw_longitude = -0.362100;

if (args.radio.gw_addr === "70b3d54b11990000") {
        console.log("watford gw");
        gw_latitude     = 51.657561;
        gw_longitude = -0.400769;
} else if (args.radio.gw_addr === "70b3d54b11990000") {
        console.log("home gw");
        gw_latitude     = 51.731056;
        gw_longitude = -0.362100;
}
else {
        console.log("unidentified gw");
        gw_latitude     = 0;
```

```
        gw_longitude = 0;
}

// calculate distance between node and gateway
dist = geolib.getDistance({latitude: coords[0], longitude: coords[1]},{latitude: gw_latitude , longitude:
gw_longitude });

var dataOut = {
        "time" : new Date(args.radio.server_time * 1000),
        "id" : "",
        "net_type" : "",
        "gw_lat" : args.radio.gw_gps.lat,
        "gw_lon" : args.radio.gw_gps.lon,
        "gw_alt" : args.radio.gw_gps.alt,
        "gw_ant_type" : "",
        "gw_location" : "",
        "gw_comments" : "",
        "node_lat" : coords[0],
        "node_long" : coords[1],
        "node_alt" : 0,
        "node_type" : "",
        "node_location" : "",
        "node_comments" : "",
        "distance" : dist,
        "rssi" : args.radio.rssi,
        "lsnr" : args.radio.lsnr,
        "datr" : args.radio.datr,
        "counter_up" : args.counter_up,
        "gw_addr" : args.radio.gw_addr,
        "dev_addr" : args.dev_addr,
        "port" : args.port,
        "stat" : args.radio.stat,
        "gw_band" : args.radio.gw_band,
        "server_time" : args.radio.server_time,
        "modu" : args.radio.modu,
        "chan" : args.radio.chan,
        "gateway_time" : args.radio.gateway_time,
        "tmst" : args.radio.tmst,
        "codr" : args.radio.codr,
        "rfch" : args.radio.rfch,
        "freq" : args.radio.freq,
        "size" : args.radio.size,
        "dev_eui" : args.dev_eui,
        "rx_time" : args.rx_time,
        "payload": buffer.toString('ascii'),
        "endoffile" : "1"
        };

        var logstream = fs.createWriteStream('out.csv',{flags: 'a'});
        writer.pipe(logstream);
        writer.write(dataOut);
        logstream.end();

/* Print out the uplink message received from the Everynet Network on the console and
* append it to the log file.
*/
// console.log("Received uplink: " + JSON.stringify(args, null, 4) + "\n");
// fs.appendFile(filepath, "Received uplink: " + JSON.stringify(args, null, 4) + "\n");
/* Get the payload (base64 encoded), which includes actual data sent by the device,
* convert it to ASCII format, print it out on the console and append it to the log file.
* We assume the device is sending a string.
*/
console.log("Payload: " + buffer.toString('ascii') + "\n");
fs.appendFile(filepath, "Payload: " + buffer.toString('ascii') + "\n");

callback(null, "100");
},
downlink: function(args, callback) {

/* Print out the downlink message received from the Everynet Network on the console and
* send back an ACK, which will then be sent to the device.
*/
console.log("Received downlink: " + JSON.stringify(args, null, 4));
var reply = {
```

```
payload: btoa("ACK")
};
callback(null, reply);
}
});

app.use(cors({methods: ['POST', 'GET']}));
app.use(jsonParser());
app.use(server.middleware());
app.listen(9090);
```

Application Server (MongoDB)

This code is a version of the previous code but uses MongoDB instead of CSV files to persist data

```
const jayson = require('jayson');
const atob = require('atob');
const cors = require('cors');
const jsonParser = require('body-parser').json;
const btoa = require('btoa');
const fd = require('fd');
const connect = require('connect');
const app = connect();
var fs = require("fs");
var csvWriter = require('csv-write-stream');
var writer = csvWriter();
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert');
var ObjectId = require('mongodb').ObjectID;
var url = 'mongodb://localhost:27017/test';

var date = new Date().toISOString().replace(/T/, '_').replace(/-/g, '').replace(/:/g, '').replace(/..+/, '');
var filepath = './log_data_' + date;

var server = jayson.server({
uplink: function(args, callback) {


var dataOut = { "distance" :-1 ,
        "time" : new Date(args.radio.server_time * 1000) ,
        "gw_lat" : args.radio.gw_gps.lat,
        "gw_long" : args.radio.gw_gps.lon,
        "rssi" : args.radio.rssi,
        "lsnr" : args.radio.lsnr,
        "datarate" : args.radio.datr,
        "gateway_eui" : args.radio.gw_addr,
        "dev_addr" : args.dev_addr,
        "counter" : args.counter_up,
        "endoffile" : "1"};


        var insertDocument = function(db, callback) {
   db.collection('lora2').insertOne(args, function(err, result) {
    assert.equal(err, null);
    console.log("Inserted a document");
    callback();
  });
};

        MongoClient.connect(url, function(err, db) {
  assert.equal(null, err);
  insertDocument(db, function() {
      db.close();
  });
});

        var logstream = fs.createWriteStream('out2.csv',{flags: 'a'});
        writer.pipe(logstream);
        writer.write(dataOut);
        logstream.end();

/* Print out the uplink message received from the Everynet Network on the console and
```

```
* append it to the log file.
*/
// console.log("Received uplink: " + JSON.stringify(args, null, 4) + "\n");
// fs.appendFile(filepath, "Received uplink: " + JSON.stringify(args, null, 4) + "\n");
/* Get the payload (base64 encoded), which includes actual data sent by the device,
* convert it to ASCII format, print it out on the console and append it to the log file.
* We assume the device is sending a string.
*/
var buffer = new Buffer(args.payload, 'base64');
console.log("Payload: " + buffer.toString('ascii') + "\n");
fs.appendFile(filepath, "Payload: " + buffer.toString('ascii') + "\n");

callback(null, "200");
},
downlink: function(args, callback) {

/* Print out the downlink message received from the Everynet Network on the console and
* send back an ACK, which will then be sent to the device.
*/
console.log("Received downlink: " + JSON.stringify(args, null, 4));
var reply = {
payload: btoa("ACK")
};
callback(null, reply);
}
});

app.use(cors({methods: ['POST', 'GET']}));
app.use(jsonParser());
app.use(server.middleware());
app.listen(9090);
```

Application Server (The Things Network)

This code collects packets sent from the Things Network

```
package sim;

var ttn = require('ttn');
var appEUI = '70B3D57ED0000648';
var accessKey = 'bmuE7OYJ5KXYJp3YNnz6bJC+oJlMaUIQzt6FN5D+Eu8=';
var client = new ttn.Client('staging.thethingsnetwork.org', appEUI, accessKey);
var geolib = require('geolib');
var fs = require("fs");
var csvWriter = require('csv-write-stream');
var writer = csvWriter();

client.on('uplink', function (msg) {


        var coords = (new Buffer(msg.fields.raw, 'base64')).toString('ascii').split(',');
        dist = geolib.getDistance({latitude: coords[0], longitude: coords[1]},{latitude: 51.731056, longitude:
-0.362100});
  console.log('', msg.metadata.server_time + "," + dist + "," + coords[0] + "," + coords[1] + "," +
msg.metadata.rssi + "," + msg.metadata.lsnr + "," + msg.metadata.codingrate + "," + msg.metadata.datarate +
"," + msg.metadata.gateway_eui);

        var dataOut = {  "distance" :dist ,
        "time" : msg.metadata.server_time ,
        "latitude" : coords[0],
        "longitude" : coords[1],
        "rssi" : msg.metadata.rssi,
        "lsnr" : msg.metadata.lsnr,
        "datarate" : msg.metadata.datarate,
        "gateway_eui" : msg.metadata.gateway_eui};


        writer.pipe(fs.createWriteStream('out.csv',{flags: 'a'}));
        writer.write(dataOut);

});
```

Android App (Main Activity)

```
package com.roderick.steve.mapper7;

import android.Manifest;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.pm.PackageManager;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Build;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.support.v4.app.FragmentPagerAdapter;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import static java.lang.Math.abs;

public class MainActivity extends AppCompatActivity implements InfoFragement.OnFragmentInteractionListener,
ItemFragment.OnListFragmentInteractionListener {

    private static String url = "";
    private static String devAdd = "";
    final Context context = this;
    private final LocationListener mLocationListener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            Log.v("WEAVER_", "Location Change");
            //textView.setText(String.valueOf(updates) + " updates");
        }

        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {

        }

        @Override
        public void onProviderEnabled(String provider) {

        }

        @Override
```

```
        public void onProviderDisabled(String provider) {

        }
    };
    ArrayList<HashMap<String, String>> contactList;
    /**
     * The {@link android.support.v4.view.PagerAdapter} that will provide
     * fragments for each of the sections. We use a
     * {@link FragmentPagerAdapter} derivative, which will keep every
     * loaded fragment in memory. If this becomes too memory intensive, it
     * may be best to switch to a
     * {@link android.support.v4.app.FragmentStatePagerAdapter}.
     */
    private SectionsPagerAdapter mSectionsPagerAdapter;
    /**
     * The {@link ViewPager} that will host the section contents.
     */
    private ViewPager mViewPager;
    private ProgressDialog pDialog;
    private ItemFragment itemFragment;
    private InfoFragement infoFragment;
    private List<Row> rowsToDisplay = new ArrayList<Row>();
    public static double latitude;
    public static double longitude;
    public static long lastGPSupdateV;

    private void handlePermissionsAndGetLocation() {


    }

    //Place this method into Activity. Did you working with onActivityResult? This method works same.
    @Override
    public void onRequestPermissionsResult(int requestCode,
                                           String[] permissions, int[] grantResults) {
        if (requestCode == 1234) {
            if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                goGps();
            } else {
                Toast.makeText(this,
                        "No permissions",
                        Toast.LENGTH_SHORT).show();
            }
        }
    }

    //This method also in Activity
    void goGps() {
        try {
            final Context ctx = this;

            final LocationManager locMan
                    = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

            LocationListener locLis = new LocationListener() {
                @Override
                public void onLocationChanged(Location location) {


                    latitude = location.getLatitude();
                    longitude = location.getLongitude();
                    lastGPSupdateV = System.currentTimeMillis();
//                    Toast.makeText(ctx,  + ", " + location.getLongitude(), Toast.LENGTH_SHORT).show();
                }

                @Override
                public void onStatusChanged(String provider, int status,
                                            Bundle extras) {
                }

                @Override
                public void onProviderEnabled(String provider) {
                }
```

```
            @Override
            public void onProviderDisabled(String provider) {
            }
        };

        locMan.requestLocationUpdates(LocationManager.GPS_PROVIDER, 60000, 10,
                locLis, null);
    } catch (SecurityException ex) {
    }
}

public static double round(double value, int places) {
    if (places < 0) throw new IllegalArgumentException();

    BigDecimal bd = new BigDecimal(value);
    bd = bd.setScale(places, RoundingMode.HALF_UP);
    return bd.doubleValue();
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);


    contactList = new ArrayList<>();


    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    // Create the adapter that will return a fragment for each of the three
    // primary sections of the activity.
    mSectionsPagerAdapter = new SectionsPagerAdapter(getSupportFragmentManager());

    // Set up the ViewPager with the sections adapter.
    mViewPager = (ViewPager) findViewById(R.id.container);
    mViewPager.setAdapter(mSectionsPagerAdapter);


    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M /* If Android is 2.3 or newer */
            && checkSelfPermission(
            Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED) {    /* And if this is first call */
        requestPermissions(
                new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
                1234);
    } else {
        goGps();
    }


    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            infoFragment.updateGPSText("Current GPS: " + round(latitude,4) + " , " + round(longitude,4));

            EditText edit = (EditText) findViewById(R.id.editText);
            url = edit.getText().toString();

            edit = (EditText) findViewById(R.id.editText3);
            devAdd = edit.getText().toString();


            Snackbar.make(view, "Syncing data from Things Connected..." + url, Snackbar.LENGTH_LONG)
                    .setAction("Action", null).show();

            new GetContacts().execute();


        }
    });

}
```

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}

@Override
public void onFragmentInteraction(Uri uri) {

}

@Override
public void onListFragmentInteraction(Row item) {

}

/**
 * A placeholder fragment containing a simple view.
 */
public static class PlaceholderFragment extends Fragment {
    /**
     * The fragment argument representing the section number for this
     * fragment.
     */
    private static final String ARG_SECTION_NUMBER = "section_number";

    public PlaceholderFragment() {
    }

    /**
     * Returns a new instance of this fragment for the given section
     * number.
     */
    public static PlaceholderFragment newInstance(int sectionNumber) {
        PlaceholderFragment fragment = new PlaceholderFragment();
        Bundle args = new Bundle();
        args.putInt(ARG_SECTION_NUMBER, sectionNumber);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.fragment_main, container, false);
        TextView textView = (TextView) rootView.findViewById(R.id.section_label);
        textView.setText(getString(R.string.section_format, getArguments().getInt(ARG_SECTION_NUMBER)));
        return rootView;
    }
}

/**
 * A {@link FragmentPagerAdapter} that returns a fragment corresponding to
 * one of the sections/tabs/pages.
 */
public class SectionsPagerAdapter extends FragmentPagerAdapter {
```

```
        public SectionsPagerAdapter(FragmentManager fm) {
            super(fm);
        }

        @Override
        public Fragment getItem(int position) {
            // getItem is called to instantiate the fragment for the given page.
            // Return a PlaceholderFragment (defined as a static inner class below).

            switch (position) {
                case 0: // Fragment # 0 - This will show FirstFragment
                    infoFragment = InfoFragement.newInstance(0, "Page # 1");
                    return infoFragment;
                case 1: // Fragment # 0 - This will show FirstFragment different title
                    itemFragment = ItemFragment.newInstance(0, "Page # 1");
                    return itemFragment;
                case 2: // Fragment # 1 - This will show SecondFragment
                    return PlaceholderFragment.newInstance(position + 1);
                default:
                    return null;
            }

        }

        @Override
        public int getCount() {
            // Show 3 total pages.
            return 3;
        }

        @Override
        public CharSequence getPageTitle(int position) {
            switch (position) {
                case 0:
                    return "SECTION 1";
                case 1:
                    return "SECTION 2";
                case 2:
                    return "SECTION 3";
            }
            return null;
        }
    }

    /**
     * Async task class to get json by making HTTP call
     */
    private class GetContacts extends AsyncTask<Void, Void, Void> {

        @Override
        protected void onPreExecute() {
            super.onPreExecute();
            // Showing progress dialog
            pDialog = new ProgressDialog(MainActivity.this);
            pDialog.setMessage("Please wait...");
            pDialog.setCancelable(false);
            pDialog.show();
        }

        @Override
        protected Void doInBackground(Void... arg0) {
            HttpHandler sh = new HttpHandler();


            Log.d("myTag", "preparing to contact url" + url);


            // Making a request to url and getting response
            String jsonStr = sh.makeServiceCall(url);

            rowsToDisplay = new ArrayList<Row>();
            if (jsonStr != null) {
                try {
                    JSONArray framedata = new JSONArray(jsonStr);
```

```
                    rowsToDisplay.add(new Row("Time                    ", "Count", "Dist", "LSNR", "Rate"));
                    for (int i = 0; i < framedata.length(); i++) {
                        JSONObject c = framedata.getJSONObject(i);

                        String time = c.getString("time");
                        Date recDate = Helper.transformDate(time);
                        String timeFormat = Helper.returnDateFormatTime(recDate);
                        String counter = c.getString("counter");
                        String devAddReceived = c.getString("devAdd");
                        Double gw_lat;
                        Double gw_long;
                        try {
                             gw_lat = Double.parseDouble(c.getString("lat"));
                             gw_long = Double.parseDouble(c.getString("long"));
                        } catch (Exception e) {
                             gw_lat = -1.0;
                             gw_long = -1.0;
                        }

                        String distance = "-1";
                        // if we have a recent GPS update then use this to calculate distance
                        long  frameTimestamp = recDate.getTime();


                        long difference = frameTimestamp - lastGPSupdateV;
                        // want abs value
                        if (difference < 0) {
                            difference *= -1;
                        }
                        if (difference < 120 * 1000) {


                           distance = Double.toString(
round(DistanceCalculator.calcDistance(latitude,longitude,gw_lat, gw_long,"K"),2));

                        }
                        Log.d("myTag",recDate + "|" + new Date(lastGPSupdateV) + "|" + difference + "|" +
distance);

                        String lsnr = c.getString("lsnr");
                        String datarate = c.getString("datarate").replace("SF", "").replace("BW", "|");


                        // filter other device addresses
                        if (devAddReceived.equalsIgnoreCase(devAdd)) {
                            rowsToDisplay.add(new Row(timeFormat, counter, distance, lsnr, datarate));
                        }

                      //  Collections.sort(rowsToDisplay);

                    }
                } catch (final JSONException e) {

                    runOnUiThread(new Runnable() {
                        @Override
                        public void run() {
                            Toast.makeText(getApplicationContext(),
                                    "Json parsing error: " + e.getMessage(),
                                    Toast.LENGTH_LONG)
                                    .show();
                        }
                    });

                }
            } else {

                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        Toast.makeText(getApplicationContext(),
                                "Couldn't get json from server. Check LogCat for possible errors!",
                                Toast.LENGTH_LONG)
```

```
                                .show();
                    }
                });

            }

            return null;
        }

        @Override
        protected void onPostExecute(Void result) {
            super.onPostExecute(result);
            // Dismiss the progress dialog
            if (pDialog.isShowing())
                pDialog.dismiss();
            if (rowsToDisplay.size() > 0) {
            } else {
                rowsToDisplay = new ArrayList<Row>();
                rowsToDisplay.add(new Row("", "", "", "", ""));
            }
            itemFragment.updateList(rowsToDisplay);
        }

    }


}
```

Node Software (Arduino, Sodaq One)

The C code below is used in the static and reference installations to send test packets from the Sodaq One node. Spreading factor is incremental at set intervals. Contains example code from the Sodaq One website.

```
#include <RTClib.h>
#include <RTCZero.h>
#include <string.h>
#include "Arduino.h"
#include <Sodaq_RN2483.h>
#include <math.h>
#include <Sodaq_UBlox_GPS.h>

#define debugSerial SERIAL_PORT_MONITOR

#if defined(ARDUINO_AVR_SODAQ_MBILI)
#define loraSerial Serial1
#define BEE_VCC 20

#elif defined(ARDUINO_SODAQ_AUTONOMO) || defined(ARDUINO_SODAQ_ONE) || defined(ARDUINO_SODAQ_ONE_BETA)
#define loraSerial Serial1

#elif defined(ARDUINO_SODAQ_EXPLORER)
#define loraSerial Serial2

#else
// please select a sodaq board
debugSerial.println("Please select a sodaq board!!");
#endif

// ABP


//Test Node 1
const uint8_t devAddr[4] = { 0x37, 0x97, 0xA5, 0xA7 };

const uint8_t appSKey[16] = { 0x0B, 0x26, 0x12, 0xD8, 0x11, 0x1A, 0xA2, 0xCF, 0xEA, 0x5B, 0x03, 0x0A, 0x87,
0x9B, 0x1A, 0x18 };
```

```
const uint8_t nwkSKey[16] = { 0x5B, 0xB9, 0xF7, 0x9A, 0x6C, 0xEB, 0xF1, 0x01, 0xAA, 0x8A, 0x68, 0xBA, 0x7E,
0x50, 0x97, 0x45 };

//Test Node 2
//const uint8_t devAddr[4] = { 0x37, 0x97, 0xA5, 0xA7 };
//const uint8_t appSKey[16] = { 0x0B , 0x26 , 0x12 , 0xD8 , 0x11 , 0x1A , 0xA2 , 0xCF , 0xEA , 0x5B , 0x03 ,
0x0A , 0x87 , 0x9B , 0x1A , 0x18};
//const uint8_t nwkSKey[16] = { 0x5B , 0xB9 , 0xF7 , 0x9A , 0x6C , 0xEB , 0xF1 , 0x01 , 0xAA , 0x8A , 0x68 ,
0xBA , 0x7E , 0x50 , 0x97 , 0x45};


// OTAA
uint8_t DevEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t AppEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
uint8_t AppKey[16] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00 };

int led = LED_RED;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

int number_of_reps = 5;
int packet_count = 1;
int set_sf = 7;

RTCZero rtc;


// day of week calc  Created by Eric Sitler
int m;          // Month Entry
int d;          // Day Entry
int yy;         // Last 2 digits of the year (ie 2016 would be 16)
int yyyy;       // Year Entry
int c;          // Century (ie 2016 would be 20)
int mTable;     // Month value based on calculation table
int SummedDate; // Add values combined in prep for Mod7 calc
int DoW;        // Day of the week value (0-6)
int leap;       // Leap Year or not
int cTable;     // Century value based on calculation table

void setupLoRaABP() {
  if (LoRaBee.initABP(loraSerial, devAddr, appSKey, nwkSKey, false))
  {
    debugSerial.println("Communication to LoRaBEE successful.");
  }
  else
  {
    debugSerial.println("Communication to LoRaBEE failed!");
  }
}

void setupLoRaOTAA() {
  if (LoRaBee.initOTA(loraSerial, DevEUI, AppEUI, AppKey, false))
  {
    debugSerial.println("Communication to LoRaBEE successful.");
  }
  else
  {
    debugSerial.println("OTAA Setup failed!");
  }
}

void setup() {
  //  //Power up the LoRaBEE
#if defined(ARDUINO_AVR_SODAQ_MBILI) || defined(ARDUINO_SODAQ_AUTONOMO)
  pinMode(BEE_VCC, OUTPUT);
  digitalWrite(BEE_VCC, HIGH);
#endif

  pinMode(GPS_ENABLE, OUTPUT);
  digitalWrite(GPS_ENABLE, HIGH);


  pinMode(ENABLE_PIN_IO, OUTPUT);
```

```
  digitalWrite(ENABLE_PIN_IO, HIGH);

  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);

  delay(3000);
  //
  //
  //  pinMode(ENABLE_PIN_IO, OUTPUT);
  //  digitalWrite(ENABLE_PIN_IO, HIGH);

  while ((!SerialUSB) && (millis() < 10000)) {
    // Wait 10 seconds for the Serial Monitor
  }

  //Set baud rate
  debugSerial.begin(57600);
  loraSerial.begin(LoRaBee.getDefaultBaudRate());

  // Debug output from LoRaBee
  LoRaBee.setDiag(debugSerial); // optional


  // connect gps
  sodaq_gps.init(GPS_ENABLE);
  find_fix(0);

  rtc.begin();
  debugSerial.println("gps_hours:" + sodaq_gps.getDateTimeString().substring(8, 10));
  debugSerial.println("gps_mins:" + sodaq_gps.getDateTimeString().substring(10, 12));

  //rtc.setHours(sodaq_gps.getDateTimeString().substring(8,10).toInt());
  /* Change these values to set the current initial time */
  const byte seconds = 0;
  const byte minutes = 20;
  const byte hours = 16;
  /* Change these values to set the current initial date */
  const byte day = 15;
  const byte month = 6;
  const byte year = 15;


  rtc.setTime(sodaq_gps.getDateTimeString().substring(8, 10).toInt(),
sodaq_gps.getDateTimeString().substring(10, 12).toInt(), seconds);
  rtc.setDate(sodaq_gps.getDateTimeString().substring(6, 8).toInt(),
sodaq_gps.getDateTimeString().substring(4, 6).toInt(), sodaq_gps.getDateTimeString().substring(1, 4).toInt());
  debugSerial.println(rtc.getHours());
  debugSerial.println(rtc.getMinutes());
  debugSerial.println(rtc.getDay());
  debugSerial.println(rtc.getMonth());
  debugSerial.println(rtc.getYear());
  // Leap Year Calculation
  yyyy = sodaq_gps.getDateTimeString().substring(1, 4).toInt();
  m =  sodaq_gps.getDateTimeString().substring(4, 6).toInt();
  d = sodaq_gps.getDateTimeString().substring(6, 8).toInt();
  if ((fmod(yyyy, 4) == 0 && fmod(yyyy, 100) != 0) || (fmod(yyyy, 400) == 0))
  {
    leap = 1;
  }
  else
  {
    leap = 0;
  }
  // Limit results to year 1900-2299 (to save memory)
  while (yyyy > 2299)
  {
    yyyy = yyyy - 400;
  }
  while (yyyy < 1900)
  {
    yyyy = yyyy + 400;
  }
  // Calculating century
  c = yyyy / 100;
```

```
// Calculating two digit year
yy = fmod(yyyy, 100);
// Century value based on Table
if (c == 19) {
  cTable = 1;
}
if (c == 20) {
  cTable = 0;
}
if (c == 21) {
  cTable = 5;
}
if (c == 22) {
  cTable = 3;
}
// Jan and Feb calculations affected by leap years
if (m == 1)
{ if (leap == 1) {
    mTable = 6;
  }
  else          {
    mTable = 0;
  }
}
if (m == 2)
{ if (leap == 1) {
    mTable = 2;
  }
  else          {
    mTable = 3;
  }
}
// Other months not affected and have set values
if (m == 10) {
  mTable = 0;
}
if (m == 8) {
  mTable = 2;
}
if (m == 3 || m == 11) {
  mTable = 3;
}
if (m == 4 || m == 7) {
  mTable = 6;
}
if (m == 5) {
  mTable = 1;
}
if (m == 6) {
  mTable = 4;
}
if (m == 9 || m == 12) {
  mTable = 5;
}
// Enter the data into the formula
SummedDate = d + mTable + yy + (yy / 4) + cTable;
// Find remainder
DoW = fmod(SummedDate, 7);
if (DoW == 0) {
  debugSerial.println("Saturday");
}
if (DoW == 1) {
  debugSerial.println("Sunday");
}
if (DoW == 2) {
  debugSerial.println("Monday");
}
if (DoW == 3) {
  debugSerial.println("Tuesday");
}
if (DoW == 4) {
  debugSerial.println("Wednesday");
}
if (DoW == 5) {
```

```
    debugSerial.println("Thursday");
  }
  if (DoW == 6) {
    debugSerial.println("Friday");
  }
  //connect to the LoRa Network
  setupLoRa();
}

void find_fix(uint32_t delay_until)
{
  debugSerial.println(String("delay ... ") + delay_until + String("ms"));
  delay(delay_until);

  uint32_t start = millis();
  uint32_t timeout = 900L * 100;
  debugSerial.println(String("waiting for fix ..., timeout=") + timeout + String("ms"));
  if (sodaq_gps.scan(false, timeout)) {
    debugSerial.println(String(" time to find fix: ") + (millis() - start) + String("ms"));
    debugSerial.println(String(" datetime = ") + sodaq_gps.getDateTimeString());
    debugSerial.println(String(" lat = ") + String(sodaq_gps.getLat(), 7));
    debugSerial.println(String(" lon = ") + String(sodaq_gps.getLon(), 7));
    debugSerial.println(String(" num sats = ") + String(sodaq_gps.getNumberOfSatellites()));
  } else {
    debugSerial.println("No Fix");
  }
}


void setupLoRa() {
  // ABP
  setupLoRaABP();
  // OTAA
  // setupLoRaOTAA();

}

void sendPacket(String packet) {

  packet = String(sodaq_gps.getLat(), 7) + "," + String(sodaq_gps.getLon(), 7);
  switch (LoRaBee.send(1, (uint8_t*)packet.c_str(), packet.length()))
  {
    case NoError:
      debugSerial.println("Successful transmission.");
      break;
    case NoResponse:
      debugSerial.println("There was no response from the device.");
      setupLoRa();
      break;
    case Timeout:
      debugSerial.println("Connection timed-out. Check your serial connection to the device! Sleeping for
20sec.");
      delay(20000);
      break;
    case PayloadSizeError:
      debugSerial.println("The size of the payload is greater than allowed. Transmission failed!");
      break;
    case InternalError:
      debugSerial.println("Oh No! This shouldn't happen. Something is really wrong! Try restarting the
device!\r\nThe network connection will reset.");
      setupLoRa();
      break;
    case Busy:
      debugSerial.println("The device is busy. Sleeping for 10 extra seconds.");
      delay(10000);
      break;
    case NetworkFatalError:
      debugSerial.println("There is a non-recoverable error with the network connection. You should re-
connect.\r\nThe network connection will reset.");
      setupLoRa();
      break;
    case NotConnected:
      debugSerial.println("The device is not connected to the network. Please connect to the network before
attempting to send data.\r\nThe network connection will reset.");
```

```
      setupLoRa();
      break;
    case NoAcknowledgment:
      debugSerial.println("There was no acknowledgment sent back!");
      // When you this message you are probaly out of range of the network.
      break;
    default:
      break;
  }
}

void loop() {
  // put your main code here, to run repeatedly:
  String packet = "12345678910ABCDEFG";

  packet_count ++;
  if (packet_count > number_of_reps + 1) {
    packet_count = 1;
    set_sf++;
    if (set_sf > 12) {
      set_sf = 7;
    }
  }

  LoRaBee.setSpreadingFactor(set_sf);



  if (set_sf == 7) {
    debugSerial.println("spreading factor:7");

  }

  if (set_sf == 8) {
    debugSerial.println("spreading factor:8");
  }

  if (set_sf == 9) {
    debugSerial.println("spreading factor:9");
  }

  if (set_sf == 10) {
    debugSerial.println("spreading factor:10");
  }

  if (set_sf == 11) {
    debugSerial.println("spreading factor:11");
  }

  if (set_sf == 12) {
    debugSerial.println("spreading factor:12");
  }


  // don't send on sat or sunday or outside working hours
  if (DoW != 0 && DoW != 1) {

    debugSerial.println(rtc.getHours());
    if (rtc.getHours() > 3 && rtc.getHours() < 19 ) {
      find_fix(0);
      sendPacket(packet);
    }
    else {
      debugSerial.println("Outside hours");


    }


  }



  delay(3 * 60 * 1000);// 3 mins
```

```
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Arduino Code (Arduino Mega with RTC)

The following code sends packets from an Arduino Mega with an external real time clock. It uses the LowPower.h library to implement a sleep cycle.

```
#include "LowPower.h"
#include "Wire.h"
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#define DS3231_I2C_ADDRESS 0x68

byte previousMinute;
static const PROGMEM u1_t NWKSKEY[16] = {  0x71 , 0xC4 , 0xA6 , 0x11 , 0x03 , 0xAE , 0x1D , 0xF8 , 0xE6 , 0x25
, 0xCA , 0xB2 , 0xD3 , 0x07 , 0x59 , 0x3C } ;
static const u1_t PROGMEM APPSKEY[16] = { 0xAF , 0x38 , 0xA5 , 0x26 , 0xA3 , 0x55 , 0x03 , 0x7D , 0x7C , 0xFE
, 0xA9 , 0x9C , 0xE3 , 0x5D , 0x2D , 0x05 };
static const u4_t DEVADDR = 0x07902B18  ; // <-- Change this address for every node!

void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[] = "Hello, world!";
static osjob_t sendjob;

const unsigned TX_INTERVAL = 60 * 3;
const lmic_pinmap lmic_pins = {
  .nss = 10,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 9,
  .dio = {2, 6, 7},
};



// Convert normal decimal numbers to binary coded decimal
byte decToBcd(byte val)
{
  return ( (val / 10 * 16) + (val % 10) );
}
byte bcdToDec(byte val)
{
  return ( (val / 16 * 10) + (val % 16) );
}

void readDS3231time(byte *second,
                    byte *minute,
                    byte *hour,
                    byte *dayOfWeek,
                    byte *dayOfMonth,
                    byte *month,
                    byte *year)
{
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
```

```
  Wire.write(0); // set DS3231 register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
  // request seven bytes of data from DS3231 starting from register 00h
  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month = bcdToDec(Wire.read());
  *year = bcdToDec(Wire.read());
}

void checkTime()
{
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  // retrieve data from DS3231
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month,
                 &year);

  Serial.print(hour);
  Serial.print(":");
  Serial.println(minute);

  if (minute % 3 == 0 && minute != previousMinute) {
    Serial.println("Triggered");

    // determine SF by time
    if (hour % 2 == 0 ) {
      Serial.println(F("Setting SF to 12"));
      LMIC_setDrTxpow(DR_SF12, 14);
    }
    else {
      Serial.println(F("Setting SF to 7"));
      LMIC_setDrTxpow(DR_SF7, 14);
    }

    do_send();
  } else {
    Serial.println("Go back to sleep");
  }
  previousMinute = minute;
}

void onEvent (ev_t ev) {
  Serial.print(os_getTime());
  Serial.print(": ");
  switch (ev) {
    case EV_SCAN_TIMEOUT:
      Serial.println(F("EV_SCAN_TIMEOUT"));
      break;
    case EV_BEACON_FOUND:
      Serial.println(F("EV_BEACON_FOUND"));
      break;
    case EV_BEACON_MISSED:
      Serial.println(F("EV_BEACON_MISSED"));
      break;
    case EV_BEACON_TRACKED:
      Serial.println(F("EV_BEACON_TRACKED"));
      break;
    case EV_JOINING:
      Serial.println(F("EV_JOINING"));
      break;
    case EV_JOINED:
      Serial.println(F("EV_JOINED"));
      break;
    case EV_RFU1:
      Serial.println(F("EV_RFU1"));
      break;
    case EV_JOIN_FAILED:
      Serial.println(F("EV_JOIN_FAILED"));
      break;
    case EV_REJOIN_FAILED:
      Serial.println(F("EV_REJOIN_FAILED"));
```

```
      break;
      break;
    case EV_TXCOMPLETE:
      Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
      if (LMIC.dataLen) {
        // data received in rx slot after tx
        Serial.print(F("Data Received: "));
        Serial.write(LMIC.frame + LMIC.dataBeg, LMIC.dataLen);
        Serial.println();
      }
      // Schedule next transmission
      //os_setTimedCallback(&sendjob, os_getTime() + sec2osticks(TX_INTERVAL), do_send);
      break;
    case EV_LOST_TSYNC:
      Serial.println(F("EV_LOST_TSYNC"));
      break;
    case EV_RESET:
      Serial.println(F("EV_RESET"));
      break;
    case EV_RXCOMPLETE:
      // data received in ping slot
      Serial.println(F("EV_RXCOMPLETE"));
      break;
    case EV_LINK_DEAD:
      Serial.println(F("EV_LINK_DEAD"));
      break;
    case EV_LINK_ALIVE:
      Serial.println(F("EV_LINK_ALIVE"));
      break;
    default:
      Serial.println(F("Unknown event"));
      break;
  }
}


void do_send() {

  // Check if there is not a current TX/RX job running
  if (LMIC.opmode & OP_TXRXPEND) {
    Serial.println(F("OP_TXRXPEND, not sending"));
  } else {



    // Prepare upstream data transmission at the next possible time.
    LMIC_setTxData2(1, mydata, sizeof(mydata) - 1, 0);
    Serial.println(F("Packet queued"));



  }


}


void setup()
{
  Wire.begin();
  Serial.begin(9600);
  pinMode(13, OUTPUT);

  // set this to a value that wont occur
  previousMinute = 99;

  //LoRa
  // LMIC init
  os_init();



  // Reset the MAC state. Session and pending data transfers will be discarded.
  LMIC_reset();
```

```
  // attempt to disable ADR
  LMIC_setAdrMode(0);
  // Set static session parameters. Instead of dynamically establishing a session
  // by joining the network, precomputed session parameters are be provided.
#ifdef PROGMEM
  // On AVR, these values are stored in flash and only copied to RAM
  // once. Copy them to a temporary buffer here, LMIC_setSession will
  // copy them into a buffer of its own again.
  uint8_t appskey[sizeof(APPSKEY)];
  uint8_t nwkskey[sizeof(NWKSKEY)];
  memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
  memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
  LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
#else
  // If not running an AVR with PROGMEM, just use the arrays directly
  LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
#endif

  // Set up the channels used by the Things Network, which corresponds
  // to the defaults of most gateways. Without this, only three base
  // channels from the LoRaWAN specification are used, which certainly
  // works, so it is good for debugging, but can overload those
  // frequencies, so be sure to configure the full frequency range of
  // your network here (unless your network autoconfigures them).
  // Setting up channels should happen after LMIC_setSession, as that
  // configures the minimal channel set.
  LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);      // g-band
  LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK,  DR_FSK),  BAND_MILLI);      // g2-band
  // TTN defines an additional channel at 869.525Mhz using SF9 for class B
  // devices' ping slots. LMIC does not have an easy way to define set this
  // frequency and support for class B is spotty and untested, so this
  // frequency is not configured here.



  // Disable link check validation
  LMIC_setLinkCheckMode(0);

  // Set data rate and transmit power (note: txpow seems to be ignored by the library)
  LMIC_setDrTxpow(DR_SF11, 14);


}


void loop()
{
  os_runloop_once();
  digitalWrite(13, HIGH);
  delay(2000);
  digitalWrite(13, LOW);
  LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);

  // on wake up check time

  checkTime();
}
```

Arduino (Arduino UNO with Dragino Shield)

This code sends LoRaWAN packets from an Arduino UNO. It contains an implementation of the LoRaWAN protocol (lmic.h).

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>

/// LoRaWAN NwkSKey, network session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
//static const PROGMEM u1_t NWKSKEY[16] = {  0xCA, 0x6A, 0x71, 0x69, 0x59, 0x76, 0x7D, 0x13, 0xDF, 0xF8, 0x29,
0xBE, 0xB1, 0xCD, 0x1B, 0x58 } ;
static const PROGMEM u1_t NWKSKEY[16] = {  0x71 , 0xC4 , 0xA6 , 0x11 , 0x03 , 0xAE , 0x1D , 0xF8 , 0xE6 , 0x25
, 0xCA , 0xB2 , 0xD3 , 0x07 , 0x59 , 0x3C } ;


// LoRaWAN AppSKey, application session key
// This is the default Semtech key, which is used by the prototype TTN
// network initially.
//static const u1_t PROGMEM APPSKEY[16] = { 0xF2, 0x0B, 0x60, 0x4D, 0xB7, 0x5B, 0x4D, 0xC6, 0x4F, 0xA3, 0xE5,
0xD9, 0x9B, 0x76, 0x86, 0xB1 };
static const u1_t PROGMEM APPSKEY[16] = { 0xAF , 0x38 , 0xA5 , 0x26 , 0xA3 , 0x55 , 0x03 , 0x7D , 0x7C , 0xFE
, 0xA9 , 0x9C , 0xE3 , 0x5D , 0x2D , 0x05 };

// LoRaWAN end-device address (DevAddr)
// See http://thethingsnetwork.org/wiki/AddressSpace
// static const u4_t DEVADDR = 0xB654BE46  ; // <-- Change this address for every node!
static const u4_t DEVADDR = 0x07902B18  ; // <-- Change this address for every node!


// These callbacks are only used in over-the-air activation, so they are
// left empty here (we cannot leave them out completely unless
// DISABLE_JOIN is set in config.h, otherwise the linker will complain).
void os_getArtEui (u1_t* buf) { }
void os_getDevEui (u1_t* buf) { }
void os_getDevKey (u1_t* buf) { }

static uint8_t mydata[] = "Hello, world!";
static osjob_t sendjob;

// Schedule TX every this many seconds (might become longer due to duty
// cycle limitations).
const unsigned TX_INTERVAL = 60 * 3;

// Pin mapping
const lmic_pinmap lmic_pins = {
  .nss = 10,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 9,
  .dio = {2, 6, 7},
};


int timer1_counter;
int seconds_elapsed;
int toggle_period;
int total_count;
int toggle;


void onEvent (ev_t ev) {
  Serial.print(os_getTime());
  Serial.print(": ");
  switch (ev) {
    case EV_SCAN_TIMEOUT:
      Serial.println(F("EV_SCAN_TIMEOUT"));
      break;
    case EV_BEACON_FOUND:
      Serial.println(F("EV_BEACON_FOUND"));
      break;
    case EV_BEACON_MISSED:
      Serial.println(F("EV_BEACON_MISSED"));
      break;
    case EV_BEACON_TRACKED:
      Serial.println(F("EV_BEACON_TRACKED"));
      break;
```

```
    case EV_JOINING:
      Serial.println(F("EV_JOINING"));
      break;
    case EV_JOINED:
      Serial.println(F("EV_JOINED"));
      break;
    case EV_RFU1:
      Serial.println(F("EV_RFU1"));
      break;
    case EV_JOIN_FAILED:
      Serial.println(F("EV_JOIN_FAILED"));
      break;
    case EV_REJOIN_FAILED:
      Serial.println(F("EV_REJOIN_FAILED"));
      break;
      break;
    case EV_TXCOMPLETE:
      Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
      if (LMIC.dataLen) {
        // data received in rx slot after tx
        Serial.print(F("Data Received: "));
        Serial.write(LMIC.frame + LMIC.dataBeg, LMIC.dataLen);
        Serial.println();
      }
      // Schedule next transmission
      //os_setTimedCallback(&sendjob, os_getTime() + sec2osticks(TX_INTERVAL), do_send);
      break;
    case EV_LOST_TSYNC:
      Serial.println(F("EV_LOST_TSYNC"));
      break;
    case EV_RESET:
      Serial.println(F("EV_RESET"));
      break;
    case EV_RXCOMPLETE:
      // data received in ping slot
      Serial.println(F("EV_RXCOMPLETE"));
      break;
    case EV_LINK_DEAD:
      Serial.println(F("EV_LINK_DEAD"));
      break;
    case EV_LINK_ALIVE:
      Serial.println(F("EV_LINK_ALIVE"));
      break;
    default:
      Serial.println(F("Unknown event"));
      break;
  }
}

void do_send() {



  // Check if there is not a current TX/RX job running
  if (LMIC.opmode & OP_TXRXPEND) {
    Serial.println(F("OP_TXRXPEND, not sending"));
  } else {

    //Serial.println(toggle_period);

    // set SF
    if (toggle_period == 20) {
      //  toggle & reset counter
      if (toggle == 0)
        toggle = 1;
      else
        toggle = 0;
      toggle_period = 0;
    }

    toggle_period ++;


    // set SF
```

```
    if (toggle == 1) {
      Serial.println(F("Sending 11"));
      LMIC_setDrTxpow(DR_SF7, 14);
    }
    else {
      Serial.println(F("Sending 12"));
      LMIC_setDrTxpow(DR_SF12, 14);
    }




    // Prepare upstream data transmission at the next possible time.
    LMIC_setTxData2(1, mydata, sizeof(mydata) - 1, 0);
    Serial.println(F("Packet queued"));
  }
  // Next TX is scheduled after TX_COMPLETE event.
}

void setup() {


  // initialize timer1
  noInterrupts();           // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;
  seconds_elapsed  = 0;

  total_count = 0;
  timer1_counter = 34286;   // preload timer 65536-16MHz/256/2Hz

  TCNT1 = timer1_counter;   // preload timer
  TCCR1B |= (1 << CS12);    // 256 prescaler
  TIMSK1 |= (1 << TOIE1);   // enable timer overflow interrupt
  interrupts();             // enable all interrupts

  toggle_period = 1;
  toggle = 0;
  Serial.begin(115200);
  Serial.println(F("Starting"));



#ifdef VCC_ENABLE
  // For Pinoccio Scout boards
  pinMode(VCC_ENABLE, OUTPUT);
  digitalWrite(VCC_ENABLE, HIGH);
  delay(1000);
#endif

  // LMIC init
  os_init();



  // Reset the MAC state. Session and pending data transfers will be discarded.
  LMIC_reset();
  // attempt to disable ADR
  LMIC_setAdrMode(0);
  // Set static session parameters. Instead of dynamically establishing a session
  // by joining the network, precomputed session parameters are be provided.
#ifdef PROGMEM
  // On AVR, these values are stored in flash and only copied to RAM
  // once. Copy them to a temporary buffer here, LMIC_setSession will
  // copy them into a buffer of its own again.
  uint8_t appskey[sizeof(APPSKEY)];
  uint8_t nwkskey[sizeof(NWKSKEY)];
  memcpy_P(appskey, APPSKEY, sizeof(APPSKEY));
  memcpy_P(nwkskey, NWKSKEY, sizeof(NWKSKEY));
  LMIC_setSession (0x1, DEVADDR, nwkskey, appskey);
#else
  // If not running an AVR with PROGMEM, just use the arrays directly
  LMIC_setSession (0x1, DEVADDR, NWKSKEY, APPSKEY);
```

```
#endif

  // Set up the channels used by the Things Network, which corresponds
  // to the defaults of most gateways. Without this, only three base
  // channels from the LoRaWAN specification are used, which certainly
  // works, so it is good for debugging, but can overload those
  // frequencies, so be sure to configure the full frequency range of
  // your network here (unless your network autoconfigures them).
  // Setting up channels should happen after LMIC_setSession, as that
  // configures the minimal channel set.
  LMIC_setupChannel(0, 868100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(1, 868300000, DR_RANGE_MAP(DR_SF12, DR_SF7B), BAND_CENTI);      // g-band
  LMIC_setupChannel(2, 868500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(3, 867100000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(4, 867300000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(5, 867500000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(6, 867700000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(7, 867900000, DR_RANGE_MAP(DR_SF12, DR_SF7),  BAND_CENTI);      // g-band
  LMIC_setupChannel(8, 868800000, DR_RANGE_MAP(DR_FSK,  DR_FSK),  BAND_MILLI);      // g2-band
  // TTN defines an additional channel at 869.525Mhz using SF9 for class B
  // devices' ping slots. LMIC does not have an easy way to define set this
  // frequency and support for class B is spotty and untested, so this
  // frequency is not configured here.



  // Disable link check validation
  LMIC_setLinkCheckMode(0);

  // Set data rate and transmit power (note: txpow seems to be ignored by the library)
  LMIC_setDrTxpow(DR_SF11, 14);

  // Start job
  // do_send(&sendjob);
}



ISR(TIMER1_OVF_vect)        // interrupt service routine
{
  TCNT1 = timer1_counter;   // preload timer



  // Serial.println(seconds_elapsed);

  if (seconds_elapsed == 60 * 2 * 3 || seconds_elapsed == 0) // 10 mins
  {
    do_send();
    seconds_elapsed = 0;
  }
  seconds_elapsed++;
}


void loop() {
  os_runloop_once();
}
```

Appendix D
User Guide for Sodaq One Nodes and Web Application

Installation

The node is based on the SODAQ One v2 (868 MHz) as listed at this site https://shop.sodaq.com/en/one-eu-rn2483-v2.html

The "with headers" option was selected as this gives most flexibility if additional components such as an LCD need to be added

To program the board, it is necessary to follow the instructions on this video: http://support.sodaq.com/sodaq-one/. NB the instructions at http://support.sodaq.com/sodaq-one/getting-started/ appear to be incorrect and may refer to an earlier version of the board.

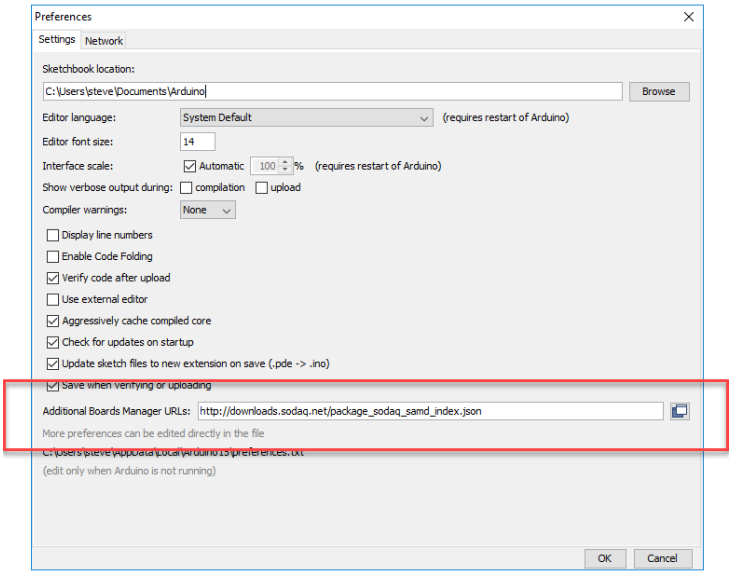
A brief outline of the steps is given below

1) On a windows PC install the latest version of the Arduino IDE currently 1.8.4. This can be downloaded from this link https://www.arduino.cc/en/Main/Software. Although there is a web version these instructions are based on the installed version.

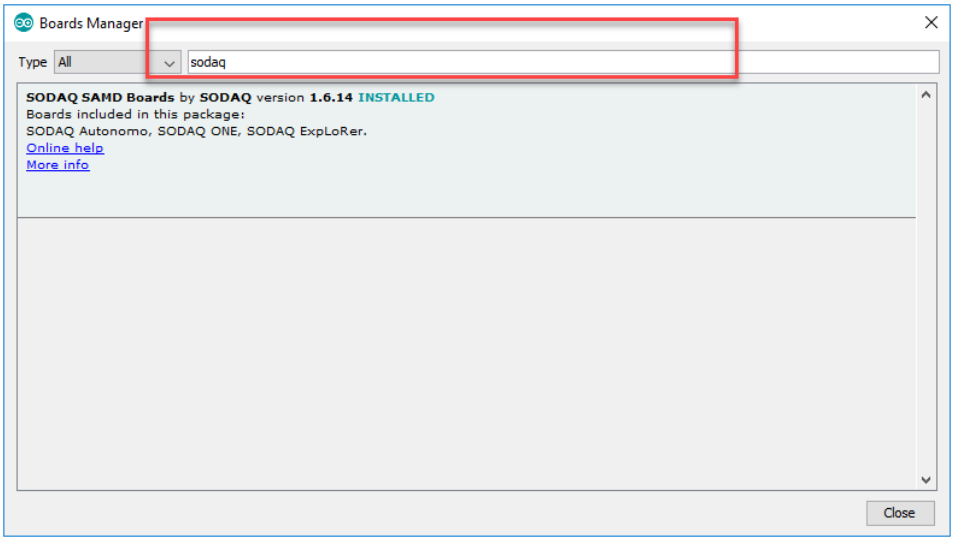
2) Add in the SODAQ URL in the configuration page as below:

In the Arduino IDE select File > Preferences

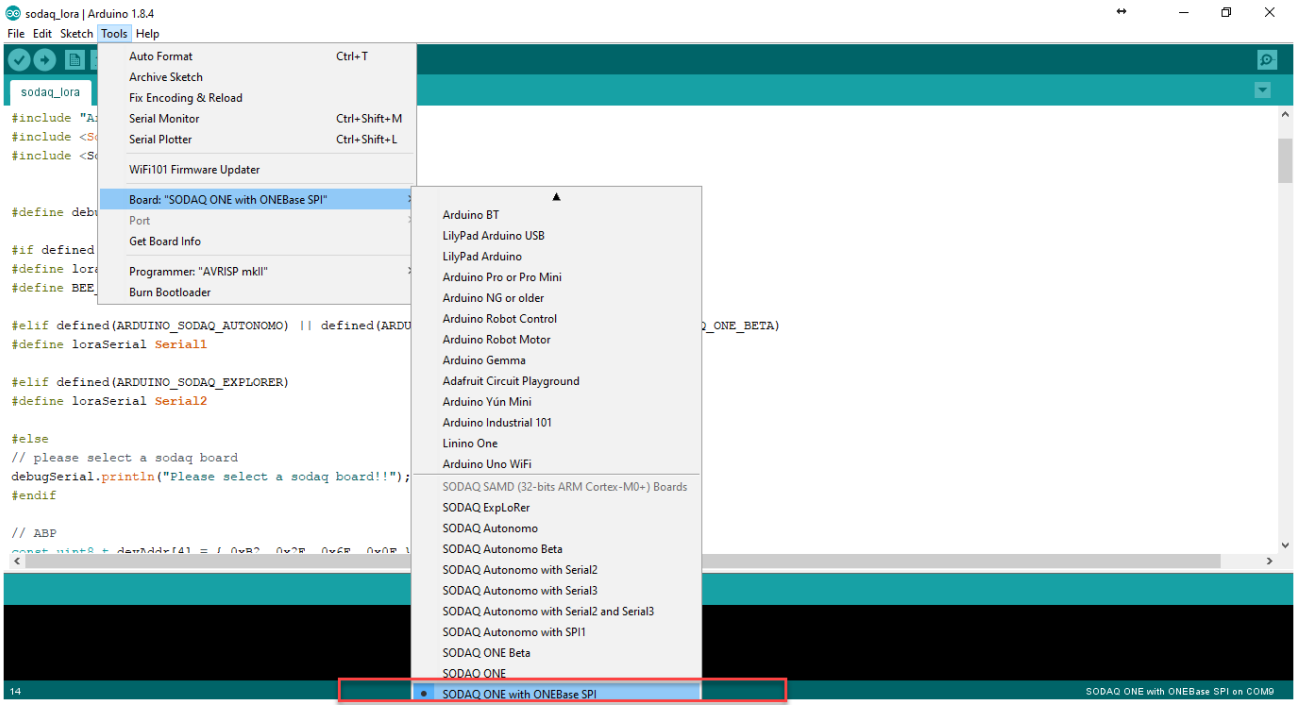
Add the URL as here:



3) Select the latest version of the board by Going to Tools > Board: ####### > Board Manager. This brings up the following:



Once the latest version of the SODAQ SAMD Boards is installed, it is necessary to select *SODAQ One with One Base SPI* as shown below:

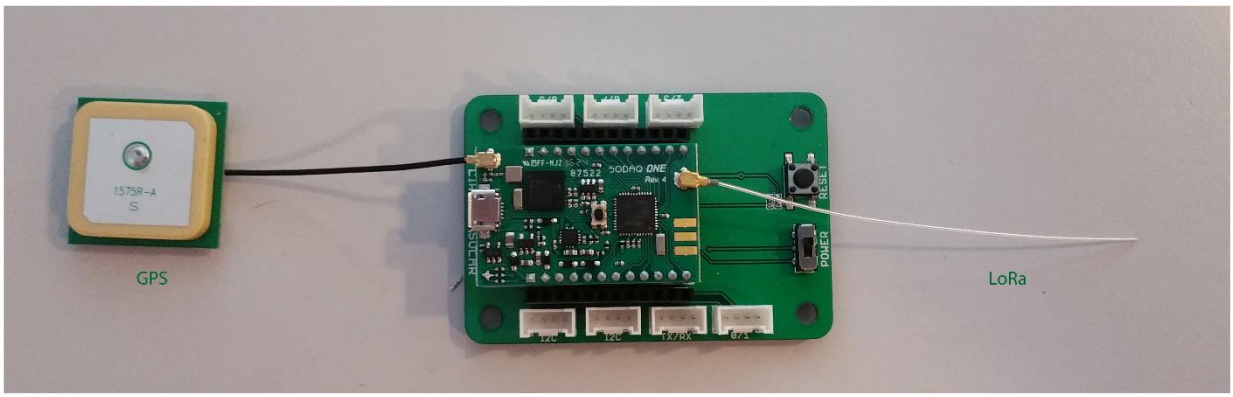


Connecting the Hardware

The main LoRa and GPS antennas and should be connected by means of the U.FL ports as below. It is simply necessary to apply pressure to the top of the port "snapping" the components together. If other external antennas are used an *U.FL to SMA adapter* is required.

To power the device the board is place into the "One Base". This has ports for connecting an external rechargeable battery. The battery is recharged either using the main USB connection on the SODAQ One v2 board or by means of a solar panel.

To program the board connect USB lead to a PC



Programming the Node

Code in the appendix to this document can be used to access the on-board GPS sensor and send LoRa packets. In addition, the Spreading Factor parameter is cycled through all available values. Five packets are sent at each value before moving onto the next value. N.B. The code uses ABP authentication rather than OTAA

N.B. To transfer the GPS coordinates of the node to the gateway it is necessary to pass this information comma separated as the payload.

The .ino file should be placed within its own directory, It should then be opened in the Arduino IDE where it can be compile and uploaded to the device.

Registering the Node

To register the node, use the normal method of adding a device to ThingsConnected. The network settings below:

Device address	462E5F
Network session key	1078EB
Application session key	A967AC

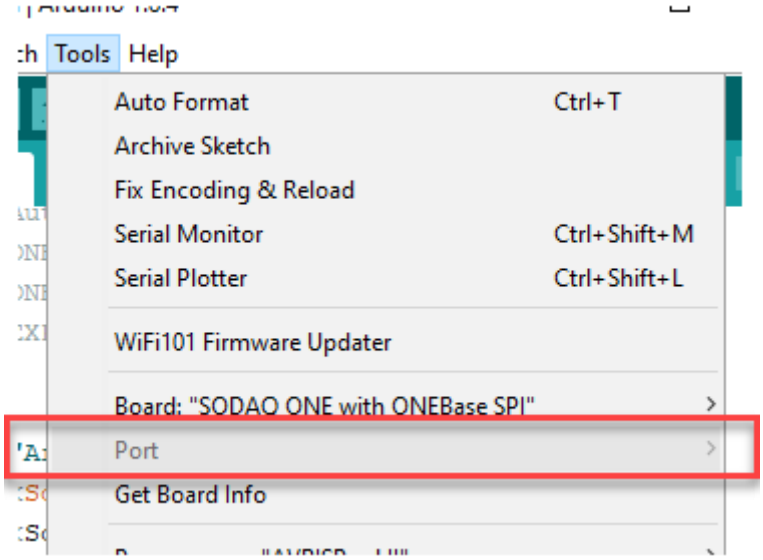
Need to be hardcoded into the Arduino .ini file before compilation and uploading. An example of this is shown below:

```
// ABP
const uint8_t devAddr[4] = { 0xB2, 0x2E, 0x6E, 0x0F };
const uint8_t appSKey[16] = {0x15 , 0x6C , 0xA4 , 0xB6 , 0x26 , 0xEF , 0x
const uint8_t nwkSKey[16] = {  0xE6 , 0xD7 , 0xEE , 0x48 , 0x29 , 0x82 ,
```

Each pair of characters are proceeded with 0x

Running the Application

Connect the node to the PC.
Select the correct port as shown below:

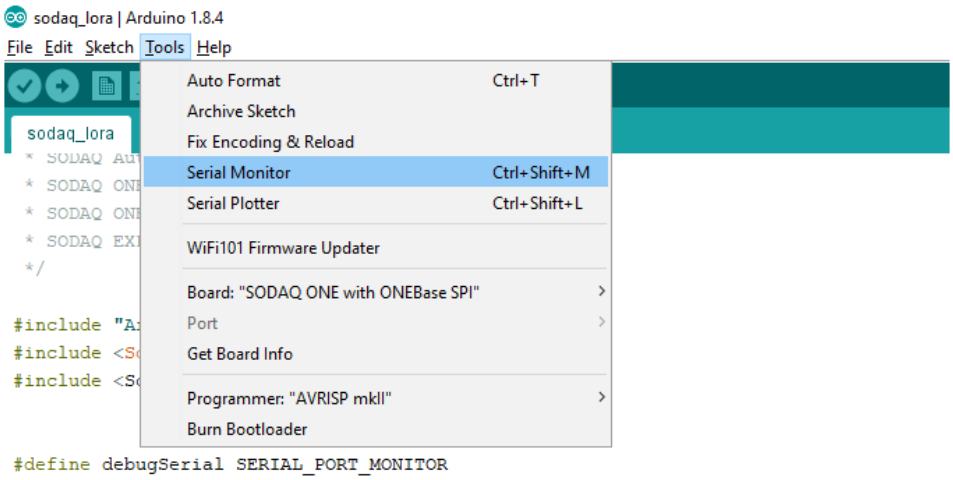


The actual port number will vary from computer to computer. If necessary, use the Windows Device Manger to determine the correct number.

To compile the code, use this control:

To upload the code, use this control:

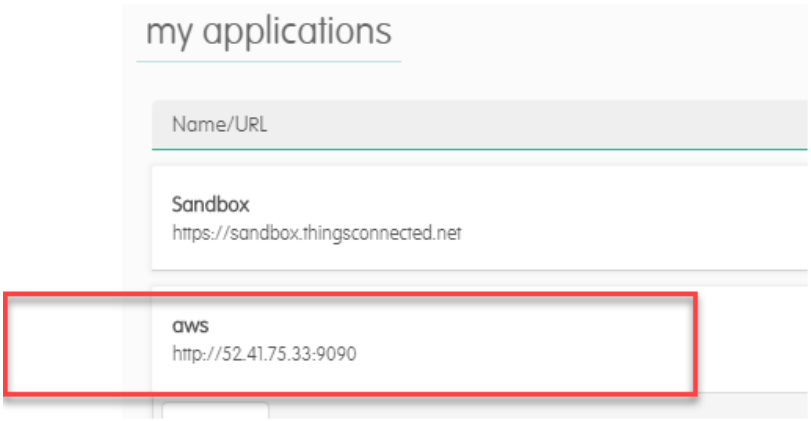
To debug open the Serial Monitor as shown below. Select the correct rate of 57600



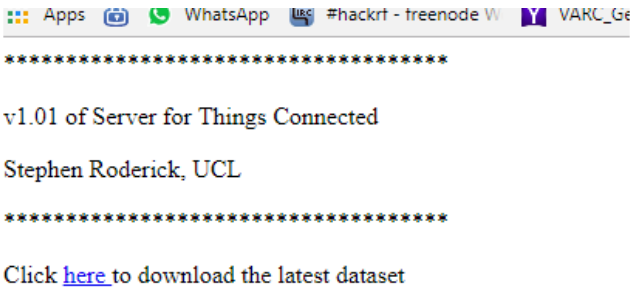
If everything is correct the node will transmit LoRa data packets. To view the data, follow the instructions in the sections below.

Connecting to An Application Server

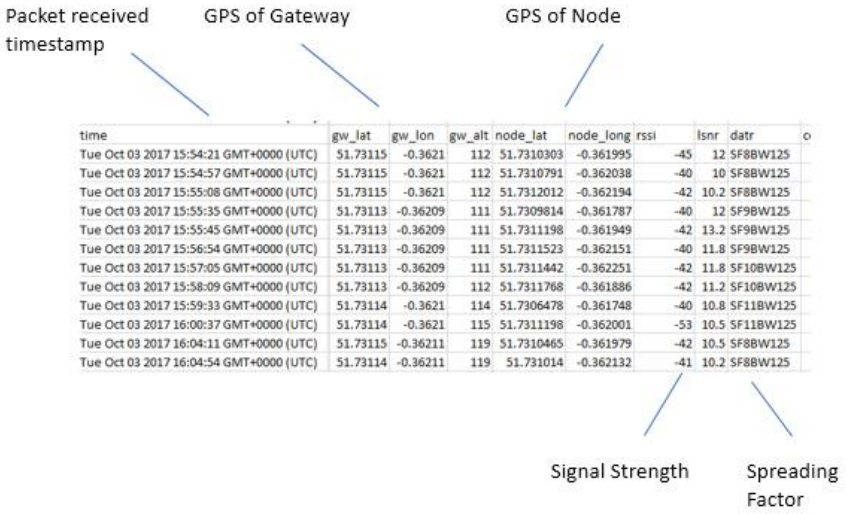
In this present study an application server is hosted on a Linux based AWS EC-2 server
This is registered with The Things Connected network:



The application server then stores all packet meta data in CSV format.



An explanation of some of the key fields in the data is given below:



time	gw_lat	gw_lon	gw_alt	node_lat	node_long	rssi	lsnr	datr
Tue Oct 03 2017 15:54:21 GMT+0000 (UTC)	51.73115	-0.3621	112	51.7310303	-0.361995	-45	12	SF8BW125
Tue Oct 03 2017 15:54:57 GMT+0000 (UTC)	51.73115	-0.3621	112	51.7310791	-0.362038	-40	10	SF8BW125
Tue Oct 03 2017 15:55:08 GMT+0000 (UTC)	51.73115	-0.3621	112	51.7312012	-0.362194	-42	10.2	SF8BW125
Tue Oct 03 2017 15:55:35 GMT+0000 (UTC)	51.73113	-0.36209	111	51.7309814	-0.361787	-40	12	SF9BW125
Tue Oct 03 2017 15:55:45 GMT+0000 (UTC)	51.73113	-0.36209	111	51.7311198	-0.361949	-42	13.2	SF9BW125
Tue Oct 03 2017 15:56:54 GMT+0000 (UTC)	51.73113	-0.36209	111	51.7311523	-0.362151	-40	11.8	SF9BW125
Tue Oct 03 2017 15:57:05 GMT+0000 (UTC)	51.73113	-0.36209	111	51.7311442	-0.362251	-42	11.8	SF10BW125
Tue Oct 03 2017 15:58:09 GMT+0000 (UTC)	51.73113	-0.36209	112	51.7311768	-0.361886	-42	11.2	SF10BW125
Tue Oct 03 2017 15:59:33 GMT+0000 (UTC)	51.73114	-0.3621	114	51.7306478	-0.361748	-40	10.8	SF11BW125
Tue Oct 03 2017 16:00:37 GMT+0000 (UTC)	51.73114	-0.3621	115	51.7311198	-0.362001	-53	10.5	SF11BW125
Tue Oct 03 2017 16:04:11 GMT+0000 (UTC)	51.73115	-0.36211	119	51.7310465	-0.361979	-42	10.5	SF8BW125
Tue Oct 03 2017 16:04:54 GMT+0000 (UTC)	51.73114	-0.36211	119	51.731014	-0.362132	-41	10.2	SF8BW125

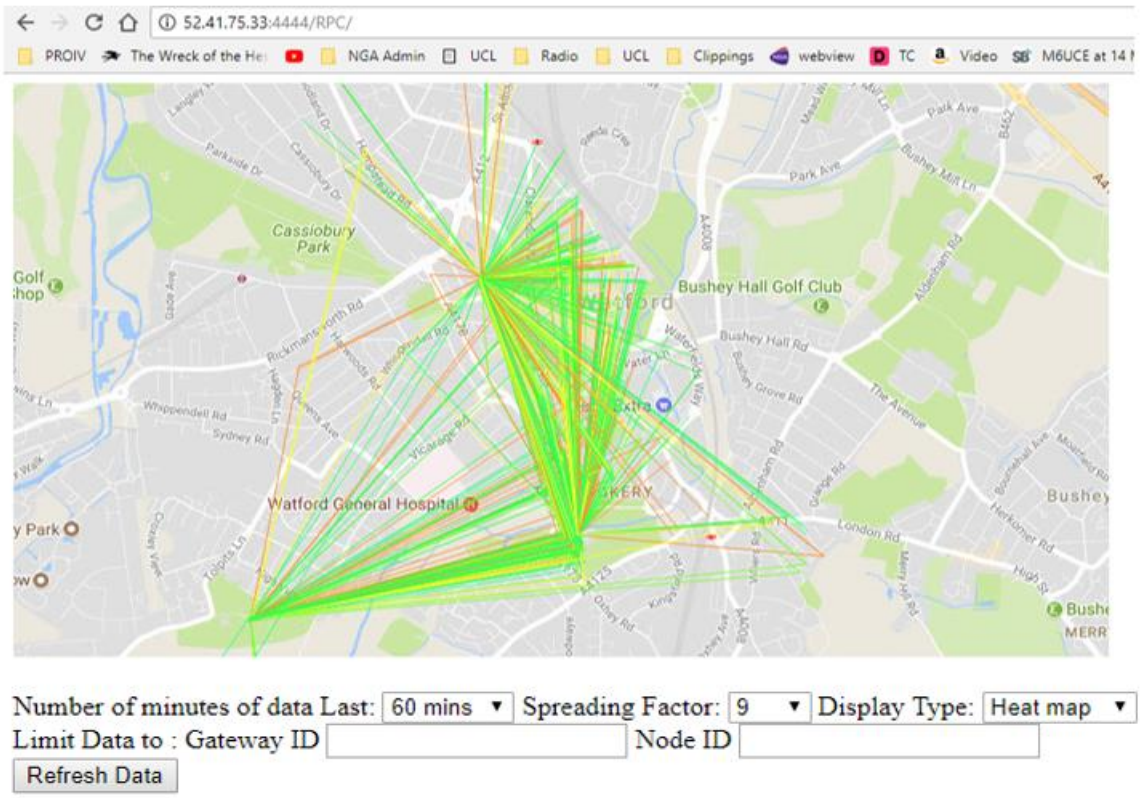
NB The CSV file contains a column that calculates the distance between the node and the gateway. Currently the calculation is incorrect. This will be resolved in the next version of the software.

To display and filter data

Navigate to

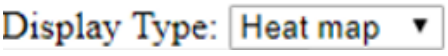
http://54.202.150.118:8080/RPC/

The following page will be displayed:



Use the controls at the bottom to filter the packet data to be returned. A specific gateway and node ID can be used to filter the data. Enter these in the controls shown below:

To change from a Path Profile view to a Heap Map use the drop down



This will display the following map view: