Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

# Efficient Firmware Update Transmission for LoRa Low Power Wide Area Technology

Cian Guinee
14317069

Supervisor: Dr. Jonathan Dukes

April 11, 2019

A Dissertation submitted in partial fulfillment
of the requirements for the degree of
Master in Computer Science

# Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: ____________________ Date: ____________________

# Abstract

The growth over the past two decades of applications leveraging Wireless Sensor Networks on the Internet of Things, has lead to an exponential increase in the number of active IoT end nodes. With such a vast user base, it is reasonable to expect that best practice software engineering processes be used in the development of applications for the Internet of Things, just as they are expected to be used on software applications for any other platform. Such process include that of Continuous Development: the idea that software should be constantly updated to add functionality or address bugs. This process becomes difficult for nodes on the Internet of Things when the problem of transmitting firmware update packages to these nodes is considered.

The type of devices on the Internet of Things come with certain problems and constraints unique to this domain. Relying largely on battery power, often incurring size and weight limitations and requiring cost-effectiveness to be sustainable are just some of the properties of an IoT end node that lead to challenging problems for developers. This leads to devices being constrained particularly by low energy consumption requirements, to ensure long battery life, and limited computational resources, to ensure compact, lightweight construction and low cost devices. LoRaWAN is a spread-spectrum Low Power Wide Area Network communication technology that aims to allow the propagation of data over long distances, while still ensuring low computational and energy costs to devices using it.

This work aims to outline a more efficient means of transmitting firmware updates to end nodes on the Internet of Things by making use of LoRaWAN technology. It will build upon work done in previous dissertations with the same subject matter, and in particular attempt make use of the Class B mode of operation offered to LoRaWAN devices, by designing, testing and evaluating three different protocols for transmitting firmware update data over this device class.

# Acknowledgements

I would firstly like to extend my thanks to Dr. Jonathan Dukes, whose guidance and input over the course of this research was invaluable.

I would also like to extend gratitude to the staff of Pervasive Nation, who provided infrastructure and support over the last six months. In particular I would like to thank Brian Murphy, who helped me through configuration issues early on.

Finally, I would like to thank my parents Ann & Brian, whose never ending support and encouragement over the course of my academic endeavours has made me not only a better student, but a more well rounded person, something for which I cannot ever thank them enough.

# Contents

# List of Figures

# Nomenclature

| | |
|---|---|
| WSN | Wireless Sensor Network |
| IoT | Internet of Things |
| HMI | Human Machine Interaction |
| M2M | Machine to Machine |
| LPWAN | Low Power Wide Area Network |
| LPWA | Low Power Wide Area |
| LoRaWAN | Long Range Wide Area Network |
| MAC | Media Access Control |
| PN | Pervasive Nation |
| NB-IoT | Narrow Band Internet of Things |
| MTC | Machine Type Communication |
| mMTC | Massive Machine Type Communication |
| RFID | Radio-Frequency Identification |
| OTA | Over-the-Air |
| LWM2M | Lightweight Machine to Machine |
| BW | Bandwidth |
| SF | Spreading Factor |
| SF-$n$ | Spreading Factor $n$ |
| CR | Code Rate |
| $T_s$ | Symbol Period (*seconds*) |
| $R_s$ | Symbol Rate (*symbols/second*) |
| $R_c$ | Chirp Rate (*chirps/second*) |
| API | Application Programming Interface |
| $R_x$ | Receive/Receipt |
| $T_x$ | Transmit/Transmission |
| ACK | Positive Acknowledgement |
| NACK | Negative Acknowledgement |
| PSR | Piggybacked Selective Repeat |
| TSR | True Selective Repeat |

# 1 Introduction

The Internet of Things (IoT) has seen rapid growth over the past decade, and the expectation is that this growth will only continue. With this growth, of course, the number of resource constrained end-devices, used to collect data in the form of sensors, as well as effect change in their surroundings in the form of actuators, will increase exponentially - most applications will make use of multiple sensors/actuators. With this emergence of IoT to the mainstream, rigid software engineering practices become increasingly important, and none more so than the practice of continuous development, that is the constant release of new features and/or bug fixes to existing, previously deployed software. The requirements of IoT end-nodes impose certain constraints on these devices, which must be overcome in order to allow the aforementioned software engineering methods to be put into practice. This work, in its entirety, aims to design an efficient data transmission protocol, making use of the existing Low Power Wide Area Networking technology LoRaWAN, for eventual use in the transmission of firmware update packages to IoT end-nodes.

The following chapter will establish the context in which this work was undertaken. Furthermore, it will outline some of the motivation behind this work, and posit some of the potential applications of the results of the dissertation, providing an understanding of the benefits of the research done in this study, over the course of the past year.

## 1.1 Context

This section aims to provide the reader with some understanding of the technologies used in this work and potential problems facing these technologies which will be addressed by the work done for this dissertation.

### 1.1.1 Wireless Sensor Networks and The Internet of Things

The Internet, as a technology, has evolved well past any expectations initially held for it in the past three decades with the advent and widespread adoption of the World Wide Web. From the early days of Web 1.0, a document sharing platform used mostly by academic institutions, the web has shown tremendous growth and continues today to become an increasingly important part of every day life. From the time of Web 1.0, the Internet has seen several radical shifts in how it is used, first of all, through the emergence of dynamic content during Web 2.0, which facilitated massive growth in usage of the Internet by significantly reducing the barrier to entry into using the Internet to create their own personalised content. Web 3.0 aims to reduce the communication barrier between human and machine by giving heterogeneous technologies the means by which to work together [18], and furthermore giving applications an awareness, when interacting with users, of context and intent. Though this is very much still a work in progress, it proves the future for Internet technology is to continue to grow closer to users. Despite the fact that Web 3.0 is still being developed, Web 4.0 has also started emerging in recent times, and introduces the idea of being constantly connected to the Internet allowing constant use of personalised services, and communication with other users [12]. Central to this idea of an interconnected existence is not only being interconnected with people, but also a connection with the objects we interact with on a day to day basis.

Central to these ideas of an interconnected world is the need for technology that is aware of its surroundings, technology that can take input from the real world and, if required, take subsequent action in the real world based upon this input.

#### Wireless Sensor Networks

Early work in this area began in the later parts of the 20th century, when advancements in micro-electro-mechanical systems (MEMS) [2]. Originally, WSNs were designed as independent, localised networks with no connection to the wider Internet. WSNs were designed to be deployed and managed on site and monitor and/or change a range of conditions in the world around them in a variety of applications such as habitat monitoring, healthcare, and traffic control [43]. As Internet technologies evolved, and the demand for increasingly remote control of WSNs gained support, the Internet of Things emerged as the solution to these, and several other, problems with WSNs and grew quickly in popularity, evidenced by the search trend in Figure 1.1.

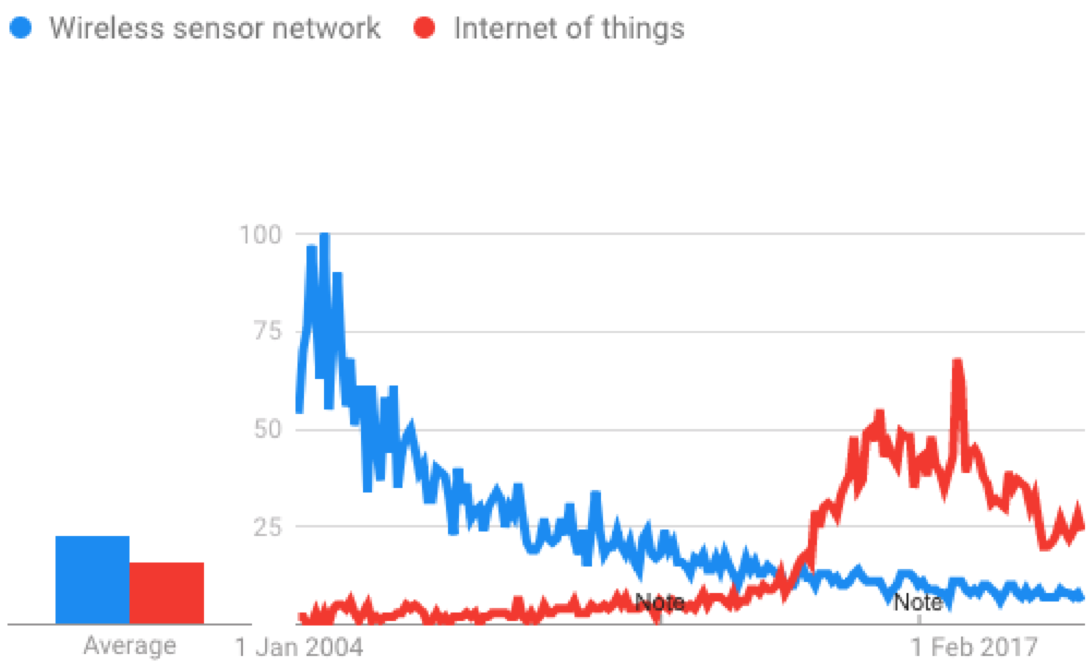


Figure 1.1: A Google Trend showing the increase in popularity of IoT, compared with the decline of searches for WSNs

## The Internet of Things

The Internet of Things (IoT) is defined as an interconnection of physical and virtual things known as Nodes to form a global infrastructure for the information society based on existing and evolving interoperable information and communication technologies for enabling advanced services [21]. This definition exhibits the differences between WSNs and IoT networks. Firstly, IoT puts an emphasis on interoperability. This is to say that devices and gateways embrace the heterogeneous nature of sensor networks, and attempt to set out methods in which different devices can work with one another regardless of underlying hardware and software differences. Extending this idea even further, the requirement for interoperability on the Internet of Things continues to allow nodes to work with services and systems exposed to them by their connection to the cloud. This leads to the second aspect of the Internet of Things, which is their less localised nature and the interconnection of individual networks to the Internet.

The exposure of these networks to the Internet provides several advantages. Foremost among these advantages, it allows nodes to connect, directly or more commonly through the use of fog nodes, to the cloud which exposes a multitude of services to the application. This allows computational work not explicitly required by the node in order to function to be performed by more capable devices, freeing the device from anything other than configuration and gathering sensor data. This is not only more suited to the computational constraints of IoT sensor nodes, but also reduces energy consumption significantly. Recent developments in the area of edge computing are further stretching the abilities of what can be achieved by these services. For example, when used in conjunction with fog nodes, effective, adjacent caching allows low latency interactions for end users allowing a broader range of applications for IoT systems, most notably, facilitating multimedia applications [37].

Moving away from localised networks and exposing networks of devices to the Internet has also allowed much of the network management work to become truly remote. This means that administrators are, for the most part, no longer required to be on site to perform regular maintenance and control operations.

### 1.1.2 Low Power Wide Area Networking

Low Power, Wide Area Networks (LPWAN) are a group of long range data transmission technologies which offer constraint aware connectivity to low power devices which are distributed across a wide geographic region [33]. On any device, transmitting and receiving data is often one of the most expensive actions in terms of both computational power and energy cost. LPWA technologies aim to address this by making certain trade offs, often concerning network throughput, in order to keep power consumption down. A more informative review of some of the options available to developers and how they differ will be given in more detail in the following chapter.

Several LPWANs exist currently, with varying levels of coverage, support, distance, energy cost, and monetary cost, and for this work, the technology used is LoRaWAN . The LoRaWAN stack, illustrated in Figure 1.2, consists of a MAC layer developed and maintained by the LoRa Alliance sitting atop the proprietary LoRa chirp spread spectrum modulation scheme developed by Semtech. This allows developers to develop applications on the application layer without having to concern themselves with low level networking concepts.

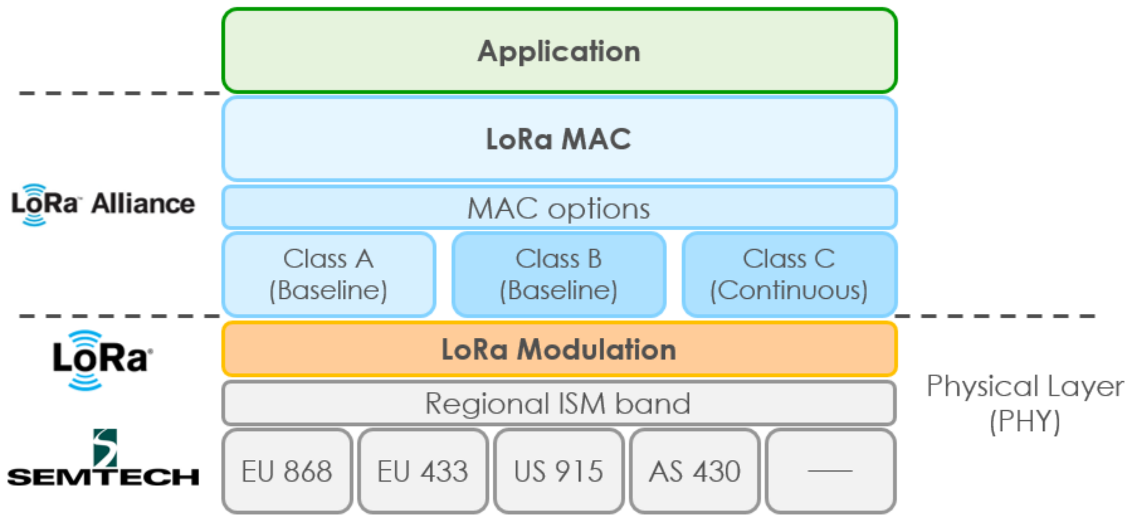


Figure 1.2: A visual representation of the LoRaWAN stack from *https://zakelijkforum.kpn.com/lora-forum-16/what-is-lora-and-lorawan-8314*

Some of the examples where LoRaWAN has seen been implemented include in first response situations, an application in which it is practical due to its ability to handle weak or noisy signals [10]. It has also seen an uptick in use in industrial scenarios due to its flexibility and scalability [38].

Part of this flexibility built into LoRaWAN devices comes in the form of the three different classes of operation it offers. This allows devices to be built with only the LoRaWAN components it absolutely requires. The three different classes offer varying levels of functionality, each with different applications in mind. The LoRaWAN classes are set out as follows:

### Class A

Class A (for *All*) devices open two short receive windows after sending an initial uplink transmission. This uplink transmission is scheduled and sent by the end node, based on the end node's needs. As the name would suggest all LoRaWAN devices must minimally implement the Class A behaviour. LoRaWAN Class A is the lowest power option made available by the specification, but this comes with a trade-off in terms of the amount of data that can be transferred with any one uplink/downlink cycle, as the receive windows are short and scheduling is left up to the end device.

### Class B

Class B (for *Beacon*) devices must also be able to exhibit the characteristics of a Class A device when needed. Additionally, Class B devices schedule regular receive windows with the gateway, allowing for a greater number of receive windows per uplink, along with a schedule the gateway is aware of. This opens up the opportunity for higher throughput per uplink, naturally at the expense of a higher power consumption.

### Class C

Finally, LoRaWAN Class C (for *Continuous*) devices, aside from implementing Class A behaviour, allow a continuous receive window to be opened, closing only when the device is transmitting information. The continuous mode is the most exhaustive mode of activation set out in the LoRaWAN specification in terms of power consumption, but also allows the lowest latency for data transmission. At this point it is also worth noting that, as defined in the LoRaWAN standard, all devices must implement Class A, but devices supporting Class B must not implement Class C and vice-versa.

### Pervasive Nation

LoRaWAN demands a certain amount of infrastructure to be in place in order to support development for the technology. Of course, it would be impractical and massively costly to setup this alone for the purpose of this project, and so an alternative solution had to be chosen. Headquartered in TCD, the Pervasive Nation group, self-described as "Ireland's IoT Testbed," are a centre for future research in IoT, and particularly LPWA, technologies, consisting of members from many of the country's top academic institutions. Together, these institutions aim to provide infrastructure throughout the country for different LPWA technologies, and cooperate with researchers to provide a means by which they can test their applications.

For this work, the Pervasive Nation infrastructure was utilised and furthermore, support was provided through contact with the staff of Pervasive Nation. Unfortunately, as the

Implementation and Evaluation chapter of this work will go on to describe in more detail, the Class A functionality, for the duration of this project, was the only LoRaWAN operation mode supported by the infrastructure. Though tests were being performed by the people at Pervasive Nation on supporting Class B functionality, staffing changes in the group led to support remaining incomplete for Class B over the PN infrastructure. This issue was solved through the use of simulated Class B devices, parameterised using data from both the LoRaWAN official specification and previous work pertaining to this area.

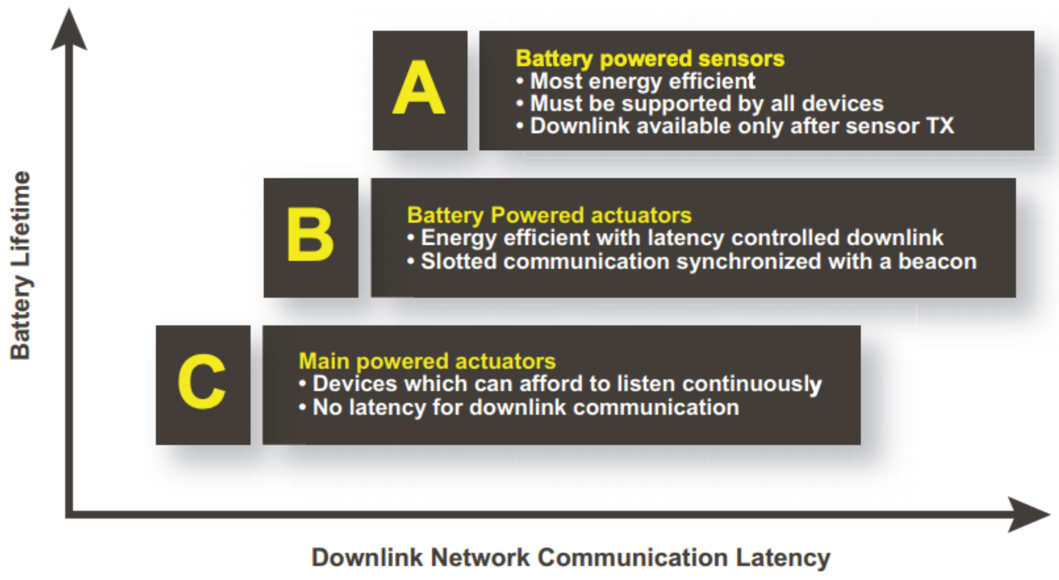

A
Battery powered sensors
• Most energy efficient
• Must be supported by all devices
• Downlink available only after sensor TX
B
Battery Powered actuators
• Energy efficient with latency controlled downlink
• Slotted communication synchronized with a beacon
C
Main powered actuators
• Devices which can afford to listen continuously
• No latency for downlink communication
Battery Lifetime
Downlink Network Communication Latency

Figure 1.3: Overview of LoRaWAN classes comparing energy consumed with latency on the downlink [23]

### 1.1.3 Applications of IoT Technology

With the advent and continued growth of IoT technology, new applications for this technology are emerging constantly. The following section will give some examples of these applications, giving a sense of applications for which the conclusions in this work may be useful.

#### Industry and Manufacturing

One area that has taken great benefit from the development of IoT technologies, as evidenced by Figure 1.4, is that of industry and manufacture. The first wave of automation

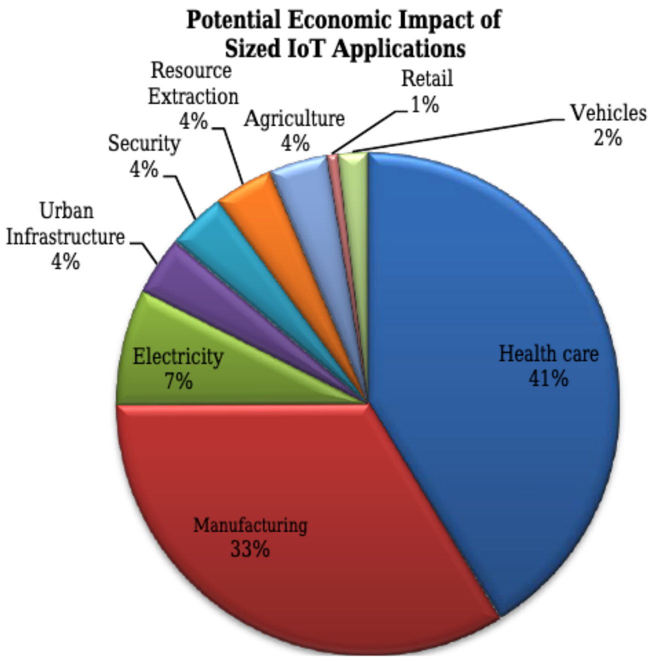


Figure 1.4: Projected Market Share for Dominant IoT Applications by 2025 [3]

and digitisation in industry commonly referred to as Industry 3.0 began in 1969 when Modicon presented the first programmable logic controller that enabled digital programming of automation systems [14]. This lead to massive changes in how factories operated by combining many individual, purpose built electronics and computers, each of which was designed for one task specifically. Combining these many individual pieces of equipment allowed factory owners and corporations to transform their places of work from slow, fully manned assembly lines to fully automated, much more effective and efficient assembly lines. This transformation began a total paradigm shift in the world of industry, and in doing so, also laid the groundwork for its successor.

The predictably named Industry 4.0, still very much in its early stages, is the successor to this wave of automation. Building on the advancements of its predecessor, the five major features of Industry 4.0 are defined as digitisation, optimisation, and customisation of production; automation and adaptation; human machine interaction (HMI) [25]. We can see from these features several areas where the use of IoT technologies is implicit in the development of this new model of Industry. Machines, under this new paradigm, will be considered part of cyber-physical systems, in which they are communicating and cooperating with one another, being constantly optimised and improving through the collection and processing of data. All of this will be enabled only by the continuing advancement of

Internet of Things technology.

### Smart Cities

Another area in which IoT has been discussed as a very important component is Smart City technology. With each passing year, all major cities around the world are becoming more and more connected. Whether it be adaptive street lighting [1], real time passenger information for transport systems [20], or environmental monitoring in city centres [35], new smart city applications for IoT technologies are popping up continuously.

With governments becoming increasingly aware of the benefits that technology can bring to a city, it is just a matter of time until we begin to see a very high demand for smart infrastructure, and this will of course require constant upkeep and maintenance from the central government.

### Farming and Agriculture

An application of IoT technology that has seen rapid growth, notably in but not limited to Ireland, is the area of smart agriculture. This is an example of a use case for IoT technology where vast improvements can be made by its introduction into daily life for farmers in both small farms on which communities are dependent using IoT nodes to monitor climate and soil conditions [22] and larger, more industrial agriculture settings, with benefits such as management and productivity evaluation, and allowing computationally verifiable methods to ensure traceable produce from farm to table [40].

### Smart Healthcare

The healthcare industry is always on the lookout for ways in which it can improve. The stakes involved in this domain don't allow for avoidable mistakes or unnecessary harm due to human error. It is for these reasons that the healthcare industry has always made efforts to adopt new technologies that come along, in an attempt to continuously improve patient experience and reduce errors and inefficiencies.

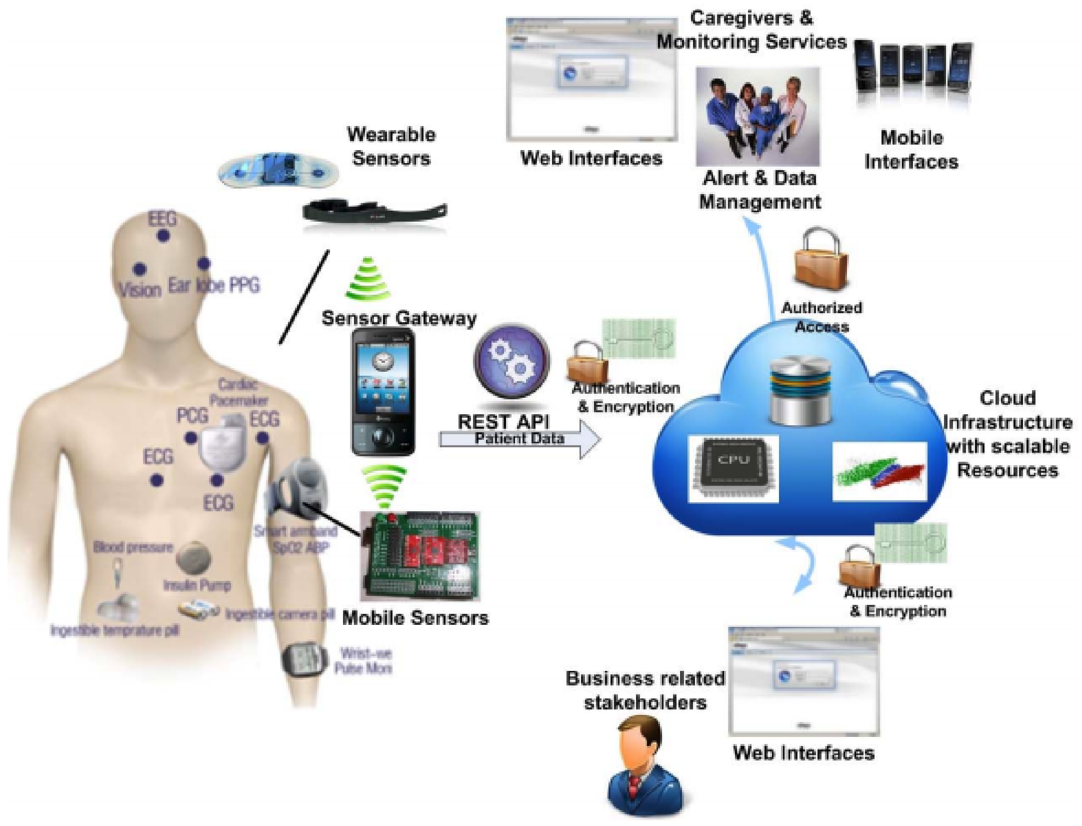


Figure 1.5: An example of a potential architecture for a smart care facility [13]

One example of a use for IoT technology in healthcare is in patient monitoring and management. Proposed systems make use of IoT connected wearables to monitor patients throughout their stay in a healthcare facility [13]. Not only is this monitoring very useful for facility staff, who are informed in changes in patient condition, allowing them to respond quicker, but also for facility management. Furthermore, the vision into the future would be that medical technology such as pacemakers, insulin pumps, blood pressure machines, hearing aids etc. all be IoT nodes which, if needed, could begin sharing information with caregivers if required. An example structure for a smart care facility is given in Figure 1.5

Mentioned in the preceding section on the Internet of Things, another responsibility of a cloud connected IoT device discussed was collecting data for use by cloud services. In making use of this, wearable patient monitoring nodes will allow patient experience to be tracked from the time they enter the facility to the time they leave. Useful information such as wait times in different departments, surges in number of patients at a given department at a given time and patient stay durations will all be collected through the use of raw data from the device, put through a number of cloud based services.

This section has given some idea of the applications for the work to follow. In all of the applications listed above, addressing issues and adding new features to existing systems will be a strict requirement. With these examples in mind the next section will go on to discuss how this is achieved through firmware updating.

### 1.1.4 Firmware Updates

The nature of software in modern computing is very dynamic. This is to say that software is no longer delivered, deployed and left alone. Rather, modern software development is an incremental process, with constant improvements being rolled out into running systems. In most modern software projects, a set process will be used to provide continuous updates over some agreed upon period (i.e. weekly, monthly etc.). This practice is in place to allow for issues to be addressed and features to be added, ensuring quality to end users, and with that in mind it is unfeasible to expect this process to change in the development of software for IoT applications. Delivery of this software, more commonly referred to in this case as firmware, must also allow for incremental updates to be applied regularly to all such devices.

A constant problem within the area of the Internet of Things, to which no one obvious answer has yet been found, is this problem of updating device firmware in the post deployment phase. Once these end nodes have been taken into the field, many options for pushing firmware to devices are ruled out. Between constraints regarding power, and constraints added by the environment to which they are deployed, many new challenges are faced by these end devices. In response to these challenges, methods for updating device firmware must conform to a certain set of important requirements.

During the dissemination of these software bundles, from both gateway to end node and, if applicable, end node to end node, protocols for the distribution of firmware must be aware of device constraints. This means that when designing a system for Over the Air firmware updating, both the means of communication from a hardware perspective, and the protocol designed to carry out these updates, must be vigilant regarding use of power and processor-heavy tasks. Aside from this, it is also important that software bundles are kept small, and the number of transmissions/receipts must be kept to a minimum. This, for the most part, means ensuring only modified parts of the firmware are transmitted.

There are also problems in the delivery of firmware over the air when security is called into question. It is crucial that end nodes can verify the update being pushed has come from a trusted source, and due to the constraints on end devices, many common cryptographic answers to this question are not possible. Error correction and failsafe measures are another problem with the dissemination of over the air updates - a corrupt piece of software could render nodes unusable until the error is caught and corrected. The concerns listed in this paragraph will, however, be considered out of scope for the research at this stage, leaving room for further study in this area at a later time.

## 1.2 Aims

This work aims to investigate LoRaWAN as a means by which to disseminate firmware update packages to IoT end nodes, and in particular, do so by making use of the Class B mode of operation offered by LoRaWAN 's specification. Transmission and Receipt of radio communications is one of, if not the most, expensive operations carried out by nodes on the Internet of Things. With this in mind, it is believed that reducing the number of uplinks required to transfer a certain amount of data will result in lower power consumption by nodes. This will require seveeral different technologies and domains to be studied, namely WSNs and the Internet of Things, LPWANs, and Incremental Firmware Updating methods and technologies.

This work will build upon work already set out in a similar Master's Dissertation, completed last year by Kevin O'Sullivan of TCD [31]. In this work, LoRaWAN is investigated, and along with this, it defines several designs for protocols with which to transmit incremental firmware updates over the LoRaWAN Class A mode of operation. The work done during the course of this project will extend the work done previously by taking into account the protocol designs developed for Class A devices, and attempting to make them compatible with Class B nodes. Furthermore, alternative approaches to protocol design with Class B at the forefront of consideration will be investigated and, as such, several new protocols will also be proposed.

Once design is completed on protocols for update transmission over Class B, proposed protocols will then be evaluated, with several different metrics considered, allowing comparison and contrast to be performed both between the new protocols designed, and between the performance of the previously proposed protocols for Class A.

Similar to the protocols set out by the preceding work, the protocols in this work are designed to reduce data overhead, keeping packet count and size of these packets on both the uplink and downlink to a minimum.

Finally, to briefly scope the project, the only factor considered in the design and testing of this work will be data transmission. This is to say that other parts of the firmware update process, for example: Control Protocols, Error handling and recovery, Bootstrapping images once on the nodes etc. will not be covered or investigated during the course of this dissertation. This will be left up to future work, and will be discussed in the future work section of this text.

## 1.3 Methodology

The methodology for carrying out the research described in the introduction of this work requires that several tasks be undertaken.

The first task will be to implement Class A behaviour on one or more devices, and establish a firmware updating process between the device and the server. This will be achieved using sample code, provided by Pervasive Nation, in conjunction with custom device code to reimplement the firmware updating protocol set out by the dissertation which this work extends. This reimplementation will not change behaviour of the protocol, however it is worth noting that it will allow for the introduction of LoRaWAN Class B behaviour as well as the already tested Class A.

A number of protocols will then be investigated, with the resulting designs for a firmware update protocol over LoRaWAN being implemented and evaluated later in the process.

After this, an investigation will be done into parameterising an emulation of Class B behaviour and, once complete, an attempt will be made to implememt an emulated Class B device as close to the parameters found during the investigation as possible. A server to communicate with the emulated Class B devices will also be implemented for each protocol tested, however it will only differ from a real application server for the protocol in question by how it communicates with the device. Core protocol logic will remain the same across real and emulated scenarios.

Directly proceeding the successful emulation of a Class B device, the protocols designed in the Design section will be implemented on the emulators, and tested. Evaluation of the results of these tests will be carried out, drawing comparisons and contrasts between both Class A and Class B for the updating process, and between the three protocols designed for Class B.

Finally, some conclusions drawn from the evaluation of the protocols will be presented, to allow a verdict to be given on the most optimal solution investigated throughout the course of this dissertation for updating firmware over the air on constrained IoT devices.

## 1.4 Dissertation Structure

The remaining sections of this dissertation are set out as follows:

Chapter 2, "Background and State of the Art," will discuss work to date in the field of Dynamic Firmware Updating over the IoT, starting with the aforementioned previous Master's Dissertation and moving on through Wireless Sensor Networks and the Internet of Things, and LPWAN technologies and the options available to IoT developers at the time of writing.

Chapter 3, "Protocol Design," describes the protocols being proposed in this work and discusses the design choices made during their creation, and how they account for different constraints of both the end nodes themselves and the LoRaWAN stack and infrastructure.

Chapter 4, "Implementation and Evaluation," describes in detail the end node devices, the PN infrastructure and how the protocols were implemented on the nodes and on the server, as well as the simulation of Class B devices. It goes on to present an evaluation of the experiments carried out with the previously described implementations and the results presented by these experiments.

Chapter 4, “Conclusions,” will evaluate the outcomes of the experimental results with regard to the aims and objectives set out at the start of the project, and from this attempts to draw a set of conclusions. It will also list proposed future work to be undertaken in the field.

# 2 Background & State of the Art

## 2.1 Firmware Updating on WSNs & the IoT

The nature of software in modern times is much less static than it once was. As explained briefly in the introduction, software is no longer deployed and left alone. Modern development teams instead continue support for long periods of time after the deployment of a product or piece of software. These requirements do not change when we consider device software for IoT nodes, despite their constrained nature. This software, referred to often in this context as firmware due to its being the bridge between hardware and software, must also be allowed to be updated in the field to address bugs and allow addition to or extension of current features. The fact that this firmware will be distributed to IoT nodes does present a certain set of challenges to be addressed by the mechanisms designed to achieve firmware updating, however practical solutions to these problems are essential to the growth of the IoT by allowing it to scale without requiring large amounts of human interaction. The first part of this section will involve setting out some of the requirements of a firmware update, moving on to discuss some of the challenges presented by IoT nodes in the context of firmware updating. Once a review of these difficulties has been undertaken, existing solutions to these challenges will be explored, with a particular focus on how and where these solutions fall down, and how they can be improved. Throughout this process, attention will be drawn, when relevant, to the problems to which this work could present solutions.

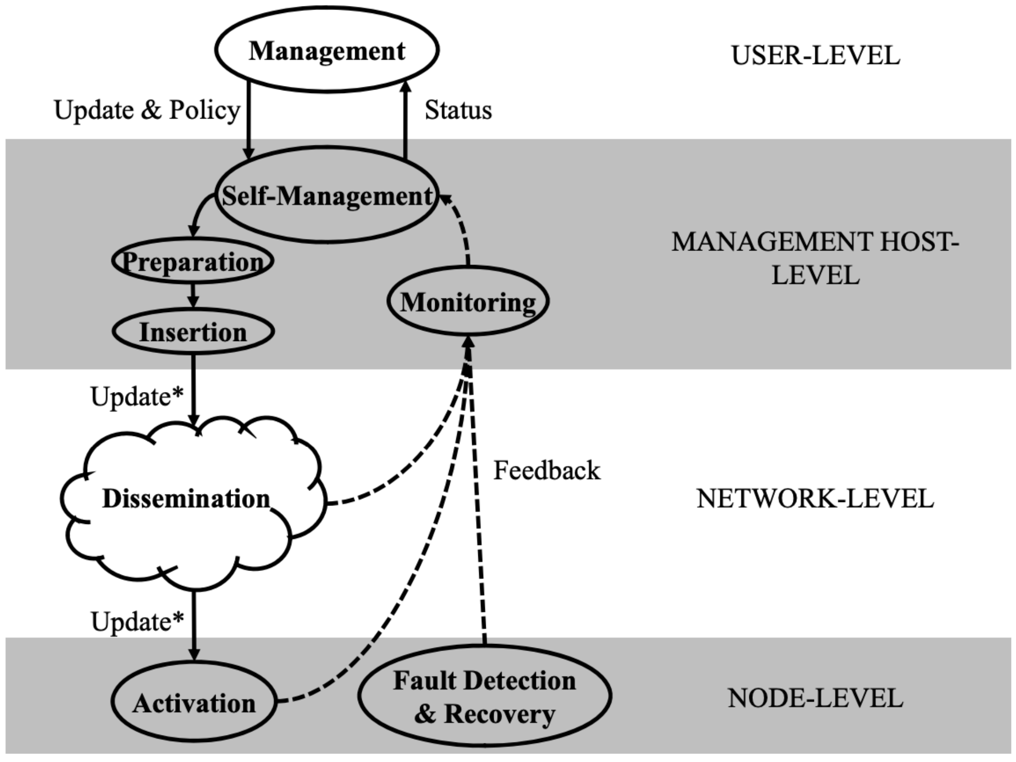


Figure 2.1: Components of an Autonomous Software Updating Mechanism [9]

### 2.1.1 Requirements of a Proposed Solution

Several studies have been carried out relating to the domain of updating firmware over the Internet of Things. In the course of these such investigations into this area, several requirements have been defined. One particular piece of research [9] lists the requirements for firmware updating mechanisms as follows:

1. **Low Intrusiveness:** During the process of updating, day to day operation of the device must remain as uninterrupted as possible. Furthermore, updates must be as automatic as possible, rather than manual, requiring direct connection to update providers.

2. **Resource Awareness:** As previously discussed in detail, a certain set of restrictions are imposed by the nature of IoT devices on resources available to applications. Like any other application on an IoT node, firmware updating mechanisms must be aware and considerate of these constraints. Particularly relevant to this work is the need to keep wireless communication to an absolute minimum, due to its very expensive nature both in terms of device resources and energy consumption.

3. **Security:** Though not addressed in this work, it is important to mention the requirement for security and certainty of integrity of update packages. Security is a challenge facing many areas of IoT application development [44], and the process of firmware updating on IoT networks is no different in this aspect.

4. **Scalability:** With the vision of the Internet of Things being for all humans and objects to be interconnected, the potential scale of applications on the IoT have the potential dwarf that of applications in other domains. It is another necessity, then, that a firmware updating mechanism designed to distribute code to the component devices of these massive systems must be able to scale well through huge numbers of nodes.

### 2.1.2 Challenges of Dynamic Firmware Updating on IoT

The following presents some of the challenges inherent in building applications for the Internet of Things, and when relevant explains how these challenges must be overcome in order to effectively transmit firmware updates over the Internet of Things.

#### Interoperability

The Internet of Things intends to provide not only machine to user interconnection, but also flawless machine to machine interaction [16]. The nature of the IoT, however, asserts that it consist of many different devices each performing unique tasks and providing different functions to its users as well as to the IoT network. The heterogeneity of the IoT, while crucial to its operation, however, is one of the more challenging aspects in developing applications that make use of it.

In the context of providing dynamic firmware updates, this variability in node type presents a few problems. Firstly, having many different devices means that overarching control protocols for the transmission of firmware updates will have to manage error handling, recovery, bootstrapping and security across these devices, each of which will have a different method of addressing these areas. The degree of complexity when addressing this issue only serves to increase with the scale of the application [36]. Finally, if code developed is not in some way reusable, it will serve to slow down the process further.

### Resource Constraints

Discussed in the last chapter with regard to devices on the Internet of Things, this review aims to point out the impact that these previously discussed constraints introduced in IoT applications will have on the firmware updating process.

Foremost among the implications constrained devices have in the process of disseminating firmware updates is their lack of computational power. Computational issues come in several ways for devices on the IoT. Firstly, the receipt and acknowledgement of packets coming across a network can put strain on the limited computational power, as these operations are expensive both in terms of energy consumption and computation. To reduce the impact these problems have on an IoT system, which severely limits prospective solutions in terms of available bandwidth, time spent active and throughput across the network [34].

Less pertinent in this work, but still a notable issue is the fact that having lower computational resource requires appropriately lightweight control protocols, particularly when it comes to tasks like bootstrapping, security and integrity verification. While certain solutions have been proposed and implemented, this remains one of the biggest problems with OTA updating on the IoT. One such solution which has gained popularity due to its awareness of device constraints is the LWM2M protocol [32].

### Energy Constraints

Another restriction imposed on IoT devices is a reliance on low energy consumption. This is particularly important if we consider large scale deployments of IoT nodes in some application. Many nodes will be reliant on battery power, and carelessness in consumption of power could lead to frequent battery drain and replacement. If we consider a farming application, for example, where end nodes are deployed to keep track of soil temperature with the intention of alerting users to potential dangers to crops. If an average farm size of around 200 acres [19] is considered, the number of devices deployed will be huge, and the distances between each node will be great. This means that with inefficient use of energy, massive amounts of work will have to be done just to keep the network running, inevitably requiring battery changing on a daily basis. One of the functions of an IoT application is to minimise the need for human interaction where possible. Considering the example just set out, it is clear that to facilitate this, energy consumption is a metric of which every application must be acutely aware.

Software updates are, of course, an expensive operation where energy consumption is concerned. Between the fact that large amounts of data are being transferred, and the requirement for its integrity disallowing the use of data fusion algorithms [8], the expense of disseminating software updates across a network of IoT nodes is one of the most costly operations a node can perform.

In this particular work, energy consumption will be the most prominent benchmark used to evaluate the solutions proposed. This is due in part to its importance as a whole when developing IoT applications, and also due to the fact that it can be easily estimated based on number of uplinks, downlinks, dropped packets and retransmits. In this context, this a particularly useful feature of energy consumption as a benchmark, as many of the experiments carried out were emulation based, and being able to estimate consumption from easily collectable metrics allowed accurate conclusions to be drawn despite the lack of real world hardware and infrastructure.

### 2.1.3 Incremental Firmware Updates

Incremental software updates are an approach to software update rollouts that require users only to acquire the parts of the code that have changed. These changes in the software are referred to in this process as 'deltas,' coming from the Greek letter delta, the mathematical symbol used to denote change. These deltas are subsequently compressed and sent over the network to the end-devices.

In [39], several different approaches for delta generation and compression are discussed. It was found that, if algorithms are chosen well, a combination of differentiation and compression could lead to significant energy savings, however on the other hand, poor choices could lead to worse performance than transmission of the update as a whole package, when decompression and integrity checks were considered.

Choosing a responsible method for generating deltas is clearly important to the energy usage of the device as a whole, however in this project, only the data transfer part of the update process is considered. For this reason, the delta generation facet to OTA firmware updating is considered out of scope. It is, however, beneficial to have some understanding of this process.

With some idea of the requirements of a reliable firmware updating mechanism and the process of generating update packages, some understanding can be shown for where a data transfer protocol, which is constraint aware, would fit into the bigger picture of an OTA firmware updating process for IoT applications in their post-deployment phase. With this in mind, this review of background information will move to some of the specific technologies pertaining to this dissertation, which is to say technologies that will help solve the issue of developing performant and reliable data transmission for delivering firmware update packages to IoT end nodes.

## 2.2 LPWAN Technologies

With the massive, and ever increasing, growth of the Internet of Things in the last two decades, the need for efficient systems of wireless communication is ever-increasing. The majority of devices on the Internet of Things are low power end nodes, and for this reason, it is important that proposed communication mechanisms are aware of device constraints. This rules out common radio communication systems such as WiFi or cellular networking, as the energy consumed by each of these is much higher than what low power nodes can afford. Aside from resource awareness, communication systems must also be considerate of factors like cost. The number of end nodes within a network on the Internet of Things could be very large on a larger system, and having hardware that is not cost effective will inhibit growth of IoT technology. LPWANs offer a cost effective solution to these issues, and solve many of the problems that will be faced particularly in the dissemination stage of OTA firmware updates, some of which are listed in the previous section.

Low Power Wide Area Networks (LPWANs) is the name given to a collection of constraint aware low power communication mechanisms, designed specifically for use on the Internet of Things, that allow communication over distances of up to 40km in rural areas and up to 10 years battery life, while keeping device and network subscription costs to a minimum [27]. This section will consider several different LPWAN technologies currently in use and emerging in the Internet of Things today. It will also compare the use of LPWANs with other solutions for wireless communication on IoT end nodes. Finally, it will discuss some of the LPWANs currently in use in both research and practical applications, availability of infrastructure and/or plans to expand current infrastructure to support new low power wide area communication technologies. An examination of planned and potential applications of the technology will also be carried out.

### 2.2.1 Current LPWAN Technologies

A number of different LPWAN technologies have emerged in recent years, each with its own advantages and disadvantages. In this section, several different LPWAN systems in use today will be discussed. Also discussed here will be some of the applications for which these various networks prove useful. Where applicable, alternative communication options will also be examined, to give a sense of why a LPWAN technology may be chosen over its alternatives.

#### Sigfox

The second LPWAN technology for discussion is Sigfox. Operating in the unlicensed ISM bands, the Sigfox network uses phase shift keying to send data across its network. Initially supporting just uplink transmission, Sigfox now allows bidirectional communication between end node and network server. While its cost is low due to effective antenna design and per unit subscription, Sigfox falls down compared to LoRaWAN when it comes to data transfer, with a limit of 140 messages per day, a maximum uplink payload of 12 bytes, and a maximum downlink payload of just 8 bytes [28].

Also seen in [28], however, is the advantage Sigfox offers over LoRaWAN in terms of range, doubling that of LoRaWAN in both an urban and rural setting. Sigfox communication nodes, similar to LoRaWAN Class A devices, largely operate in sleep mode, only waking to transmit data, and as such a similar battery life can be observed by both.

Applications for Sigfox, despite its advantages, are limited, however, and most uses of the technology are seen in academia and research. Several applications of LPWANs are considered in [15], and it can be observed in this work that other technologies seem more beneficial in practical applications than Sigfox, particulary considering the firmware updating application.

### NB-IoT

NB-IoT (Narrow Band Internet of Things) is a LPWAN technology that uses existing cellular networks to communicate with IoT end nodes. It is built from existing LTE functions, with many features stripped away, allowing for a simpler standard, leading to reduced cost and more efficient performance [6].

The reliance on licensed bands can impact cost negatively, but also provides somewhat of a guarantee of robust and reliable transmission. NB-IoT is similar to LoRaWAN in its relatively high uplink and downlink payload sizes of 125 bytes and 85 bytes respectively [6].

Many applications of NB-IoT centre around the idea of Machine Type Communication, and specifically, so-called massive MTC (mMTC)[29]. The network's reliability makes it a good candidate for supporting large amounts of devices transferring massive amounts of data, and the drawbacks in terms of cost can be outweighed by the benefits of its robustness.

### DASH7

DASH7 is another LPWAN technology that operates on sub 1GHz bands, similar to Lo-RaWAN
and Sigfox, which has its origins in the active RFID standard. Using this network, tags send transmissions to gateway servers in an asynchronous manner, with the tags initialising contact with the gateway whenever a transmission is required with no need for periodic synchronisation [42], similar to LoRaWAN Class A.

While ranges of up to 10km have been suggested, testing has indicated that the technology is much more effective at distances of up to 1km [7]. This makes DASH7 the shortest range technology discussed here, but it does have advantages in terms of its throughput and high payload size of 256 bytes, and its ease of deployment due to low reliance on infrastructure.

Applications of this newer technology are still being explored, but the outlook for this particular technology is not as bright as for the others discussed in this section. It seems that on many levels, DASH7 is outclassed by its competitors. In explorations of the feasibility of DASH7 for industrial applications, it has been suggested to be unsuitable for such use [17].

While there are many other LPWAN technologies, both deployed and in development, the three described above offer a good idea of the variety in the field, and the advantages and disadvantages of taking different approaches in the design of an LPWAN technology. With some of the technologies in use now described, the discussion here will move to focus on applications LPWAN technology.

### 2.2.2 Availability of LPWAN Infrastructure

While many solutions, some outlined previously, exist for LPWAN communication, their applications vary, as does the availability of infrastructure for each. This section of the review will focus on the use of LPWAN communication paying attention to availability, particularly in this country, as well as listing some applications in which they can be put into use.

One of the major obstacles in the research and application of LPWAN technology can be the overhead in terms of development and deployment of required infrastructure. Not only does radio communication technology require costly infrastructure for broadcast in the form of multiple antennae stationed in strategic locations, but technologies must adhere strictly to local regulations and restrictions on radio broadcasting.

Sigfox is the network that has seen the most uptake in terms of coverage, with Ireland becoming the sixth country in the European Union to achieve full coverage for the network. This was due in large part to the operator, VT-Networks, who raised funding and developed the infrastructure in place today. Due to its full coverage of the country, Sigfox can be seen in use in a variety of research and practical applications.

Where coverage and infrastructure are concerned, NB-IoT immediately has an advantage over its competitors based on its reliance on existing cellular infrastructure. Any area of the country supporting LTE communication will also support NB-IoT. Currently, the

Irish cellular operator Vodafone offer their network for NB-IoT applications, allowing a vast infrastructure with high coverage to be used. The reliability of having an established network appeals to the developer of NB-IoT applications, however the cost per device offered by the provider can be a limiting factor. This high cost may see the network used less in academia and research as in for-profit applications, but this largely remains to be seen, with the service only rolling out in 2017.

DASH7 is another LPWAN technology which does not have a high overhead where construction of infrastructure is concerned, due to the communication on the DASH7 network being based on the active RFID standard. Information on practical and research applications of DASH7 were, however, difficult to come by, which suggests uptake in the technology has been slow. This may be to the limiting factors discussed in the preceding description of the technology.

### 2.2.3 Applications of LPWAN Technology

LPWAN technology offers a wide range of applications. In the discussion that follows, some of the many applications of the technology will be examined.

#### Farming and Agriculture

A very obvious application of LPWAN technology, and without question relevant here in Ireland, is in the area of IoT connected smart farms. Vast areas of the world are used for agriculture, and it is an area in which many applications of IoT technology have been suggested, and successfully deployed.

LPWANs offer an obvious advantage over shorter-range communication mechanisms when it comes to farming, as nodes in a smart farm network will be deployed over hectares of land. Rather than attempting to deploy multiple gateways which can be used to communicate over short range with end nodes, it makes more sense to keep overhead in terms of initial deployment and maintenance of units to a minimum by using LPWAN.

The research done in [26] outlines the design of a system for use in agriculture which uses LoRaWAN as its main communication mechanism. The system outlined in this work is basic, but it gives a sense of the advantages of using LPWAN technology in this particular application, as well as indicating the readiness of the technology to support applications like the one described in the cited paper.

### Renewable Energy

The world is moving ever closer to relying fully on renewable energy, taking advantage of natural conditions in the surrounding area. Notable, for example, in Ireland are its many hydroelectric plants and wind farms, both offshore and inland. As the worlds demand for renewable energy increases, so does the need for an efficient system to monitor new infrastructure, and IoT has been shown to offer many advantages when integrated in such systems. Using LPWANs as a communication technology for these systems makes sense, as again, maintenance and initial deployment work must be kept to a minimum, and on a wind farm that covers several acres, using LPWANs as a communication system ensures this, by removing the need for multiple gateways.

Some examples include a monitoring system for energy providers, which could be applied to systems existing in Ireland or further afield [11]. This example uses LoRaWAN as its communication mechanism, but again this could be changed to suit project requirements.

### Smart City Systems

One application of IoT technology that is seeing a constant increase in use around the world is in the development of so-called smart cities. From simpler tasks such as providing real time information to commuters, to more difficult issues such as air quality and environment monitoring in cities, new smart city applications are constantly being deployed around the country.

This particular application of LPWAN technology tests the limitations of many of the systems discussed earlier, as urban environments tend to be filled with obstacles to communication, and this puts many of the LPWANs discussed earlier at the lower end of their range capabilities. They still, however, offer the best solution when we consider the size of modern cities. Furthermore, if we take Dublin as an example, the infrastructure for implementing LPWAN largely exists already, if we consider that NB-IoT, LoRaWAN and Sigfox coverage are all available in the city. This would allow independent systems being developed separately by separate entities some choice in the technology they use for communication, and could encourage development in the area.

This section has outlined some of the LPWAN technology currently available for use in IoT research and practical applications. An examination of some of the available LPWAN technologies has also been carried out. LoRaWAN as an option in this section was omitted, though some comparisons with LoRaWAN were still drawn, as this particular technology is described in detail in the next section of this review.

## 2.3 LoRaWAN

The following section will take a closer look at the LoRa and LoRaWAN technologies which will be put to use throughout this project as the chosen LPWAN technology with which to design protocols for firmware update transmission.

### 2.3.1 The LoRa Physical Layer (LoRa PHY)

LoRa , a modulation technology acquired by Semtech in 2012, when still in its early stages, and since developed and maintained by the same company, is a proprietary modulation technology based on Chirp Spread Spectrum modulation, which utilises orthagonal spreading factors which enable variable data rates, and Forward Error Correction . This modulation scheme presents LoRa with an increased resistance to channel noise, long term relative frequency, doppler effects and fading [30].

#### Chirp Spread Spectrum Modulation

The LoRa physical layer makes use of Chirp Spread Spectrum modulation to facilitate reliable and robust transmission over long range. This modulation scheme uses increasing or decreasing frequency patterns, such as those pictured in the Figure above, to encode data coming across the network. These increasing and decreasing modulations of the frequency are known as up-chirps and down-chirps respectively. This results in a final signal that

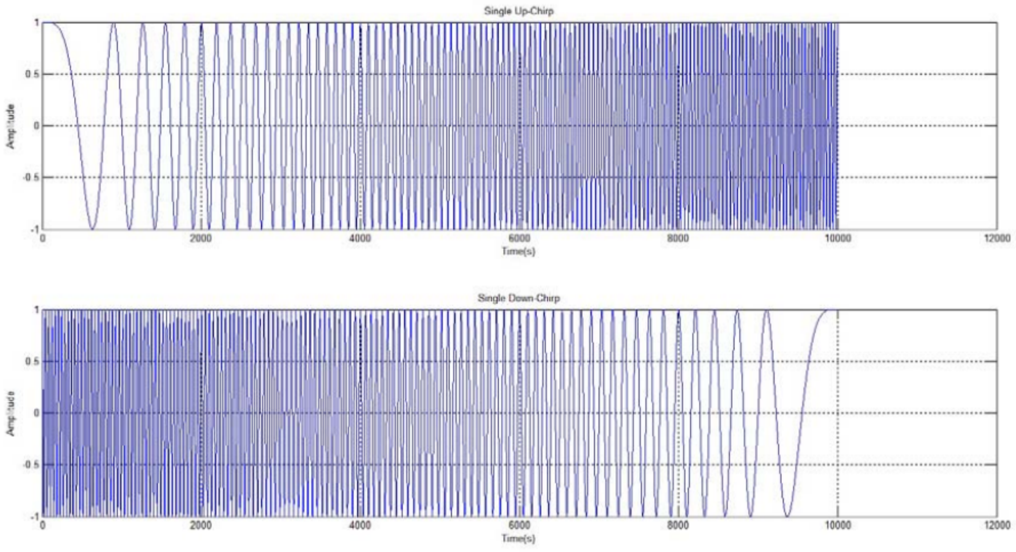


Figure 2.2: Examples of an up-chirp and down-chirp respectively, used to encode data on the LoRa physical layer [30].

consists of a preamble, a series of up-chirps followed by two quarter down-chirps, and the encoded data, which will be a series of up/down chirps [4].

As alluded to above, the LoRa modulation scheme is very customisable relative to its competitors. There are several parameters to the construction of a LoRa signal that allow this. Namely, these are Bandwidth ($BW$), Spreading Factor ($SF$) and Code Rate ($CR$). Signals are generated by LoRa as $2^{SF}$ chirps per signal, covering the entire frequency band. As mentioned above, a LoRa signal begins as a series of up-chirps followed immediately by two down chirps. If the maximum frequency of the band is reached before a chirp is complete, the frequency will wrap around again, starting at the minimum frequency and continue back up the band as the chirp continues. The amount of information encoded in a single signal is directly related to the chosen spreading factor. Since $2^{SF}$ exist in a signal, the amount of data which can be encoded per signal is $SF$ bits [5].

The rate at which chirps occur is decided by the bandwidth available, with one chirp occurring per second, per Hertz of bandwidth. This has several knock on effects on the modulation, with an increase of one spreading factor leading to a halving of the chirp span, and a doubling of the symbol duration. Furthermore, increasing bandwidth will also increase transmission speed proportionally. The symbol period ($T_s$), symbol rate ($R_s$) and chirp rate ($R_c$) can be defined by equations (1), (2) and (3) respectively [5][31]:

$$T_S = \frac{2^{SF}}{BW} \tag{1}$$

$$R_s = \frac{1}{T_s} = T_S = \frac{2^{SF}}{BW} \tag{2}$$

$$R_c = R_s \times 2^{SF} \tag{3}$$

### Code Rate and Forward Error Correction

Additionally, LoRa allows for error correction by inserting a forward error correction code into symbols. When this is taken into account, along with the supplied code rate, a usable bit rate for a given configuration can be determined by equation (4) below [5]:

$$R_b = SF \times \frac{BW}{2^{SF}} \times CR \tag{4}$$

Code rate, in this instance, is used to describe the proportion of data bits that carry useful information to the redundant bits used for error correction in a symbol. This is made more clear perhaps by the following example: If a LoRa symbol contains 5 bits, and uses a code rate of $\frac{4}{5}$, the number of useful data bits will be 5 and the number of redundant bits is $5 - 4 = 1$ [31]. Code rates permitted by LoRa 's modulation scheme are $CR \in \{1, 2, 3, 4\}$.

## 2.3.2 The LoRaWAN Protocol

The LoRaWAN network protocol is a set of standards that define what exactly comprises a LoRaWAN network. Currently, the LoRaWAN specification is maintained by the LoRa alliance, a non-profit organisation with over five hundred member companies. The following section will outline a portion of the details laid out in this standard, in particular the details relevant to this work.

### Topology

The LoRaWAN network consists of three components. The first of these are known as end-devices. End-devices are the small, resource constrained end nodes that collect data on location. The second component of the network are LoRaWAN gateways. End-devices connect to one or more LoRaWAN gateways in a star-of-stars model. The final component of the network is the LoRaWAN network server, which coordinates the delivery of packets from end-devices to their corresponding application servers, authenticates data and schedules data for downlink to end-devices via the gateways. Network servers connect to device gateways using a standard IP connection.

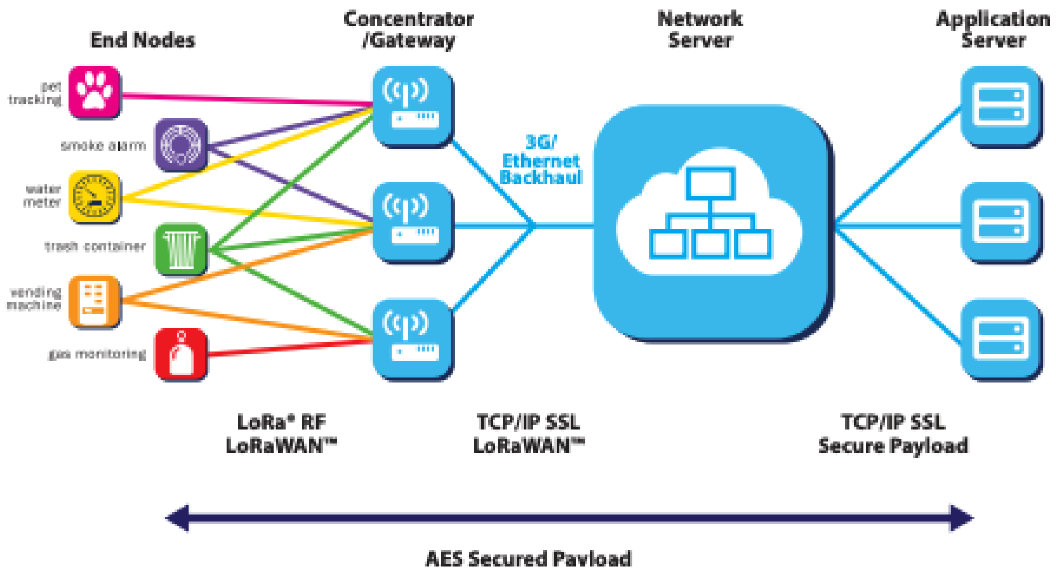


Figure 2.3: An overview of LoRaWAN 's network topology [23].

The application server, not strictly part of the LoRaWAN specification, is a regular IP server that can send and receive packets from devices through the use of a restful API, exposed by the infrastructure provider/network server. The application server facilitates a certain level of flexibility, as multiple application servers can be written to be used in different, independent applications using data from the same devices.

## LoRaWAN Classes

Discussed briefly in the introduction, LoRaWAN offers three different classes of device, which is simply a definition of how the device will operate. The following is some of the information available on what differentiates the classes, what advantages/disadvantages are inherent in each and some intended applications of each device class.

The first class made available by LoRaWAN is Class A (*'All'*), which all devices must implement minimally. In this class, end-devices send uplink messages to their gateways at their own discretion. Immediately after sending these uplink transmissions, a receive $(R_x)$ window is opened on the end-device. If the gateway has already scheduled a packet, it will send this on receiving an uplink. After a certain delay, if no packet is received a second $R_x$ window will be opened on the device. This second receive window delay is configured beforehand, during device registration, thus making the network server aware of it, allowing packets to be sent at the correct time. An illustration of this is seen in Figure 2.4.

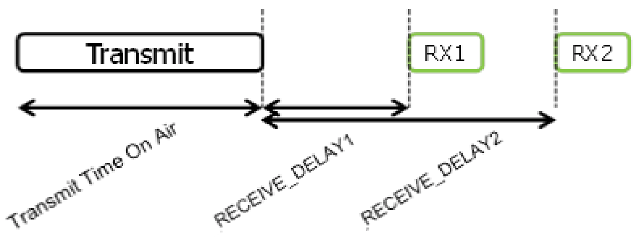


Figure 2.4: Timing slots on LoRaWAN Class A [24].

LoRaWAN Class A operation is the least power consuming mode of operation offered in the specification. Periodic uplinks, sent at the discretion of the end device, mean that the number of communications between end-devices and their gateways is kept to an absolute minimum and the communications hardware spends most of its time in sleep mode.

LoRaWAN Class A was intended to be used by sensor nodes to transmit data they had collected back to the application server via the LoRaWAN gateway [23], again meaning that these nodes would only do the minimum amount of networking required by their applications. This is to say that the intention for Class A was to be a unidirectional mode of operation for these sensor nodes, allowing them to send small amounts of data with minimal, if any, downlink data being received. This presents a problem when the application of firmware updating is considered, a process which a series of consecutive back and forth transmissions between the application server and the end-devices.

In [31], several protocols were designed for the transmission of firmware over LoRaWAN Class A. Under evaluation, a modified version of Piggybacked Selective Repeat ARQ performed most optimally from the selection of protocols tested. While the protocol did perform well under the circumstances, operating under Class A leads to more uplinks being sent and a potentially longer transmission time, all culminating in a poor comparative energy consumption.

For this reason, it was decided that this work would aim to extend and further the investigations done in this previous dissertation, making use of LoRaWAN Class B to attempt to break some of the limiting factors imposed by Class A, and lead to a better overall energy performance.

Class B (*'Beacon'*), the second mode of operation offered under the LoRaWAN specification involves a more regular uplink/downlink schedule between the end-device and gateway, enabling more efficient two-way communication. The intended application of LoRaWAN Class B is for use in devices that control actuators - devices controlling equipment capable of changing the environment surrounding them. For this application, two way data transfer is required to send control instructions to actuators, and more importance is given to the problem of transmitting data from the LoRa gateway to the end-device.

In Class B operation, an end-device the gateway to which it is connected will periodically be synchronised through the transmission from the end-device to the gateway, called a 'Beacon,' allowing the gateway and end-device to agree upon timing. Once this agreement, or 'Beacon Lock,' is complete, the device will open a series of receive windows, in this instance named 'ping slots,' during which the device can be sent downlink data by the gateway. The period of time between beacon lock transmissions is called the *beacon period* and the time between ping slots named the *ping period*. Class B operation is illustrated in Figure 2.5.

It follows from this description that Class B is ideal for a firmware updating application. The amount of data the end device has to receive is a constant, regardless of what device class is used - the firmware update will stay the same size. This is to say that no change

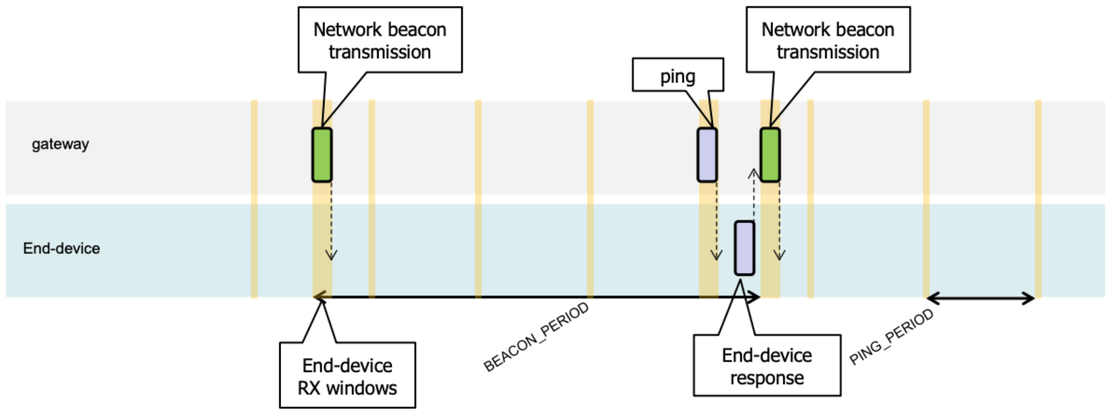


Figure 2.5: An illustration of LoRaWAN Class B operation [24].

of device class can offer any improvement on the amount of data being sent to the device and therefore it is difficult to improve energy efficiency in a firmware updating application on the downlink. Where an optimisation does exist, however, is in the uplink of responses from the end-device to the LoRaWAN gateway. Using Class B, more data can be sent from the gateway to the end-device per uplink and this fact opens up an opportunity to improve energy efficiency of data transfer using LoRaWAN Class B functionality.

Class C, the final mode of operation offered by LoRaWAN is intended for use on the gateway, and thus will only be briefly discussed here for the sake of completeness. For end-device firmware updating, Class C will not be considered.

In Class C (*Continuous*) operation, the timing structure is much the same as that of Class A. The difference with this mode of operation, however, is that after the second $R_x$ delay, the $R_x$ window opened will stay open continuously until the next uplink is performed, hence this mode's name. As expected, this is a very costly mode of operation in terms of energy consumption, and thus is not intended for use on end-devices. An illustration of this operation can be seen in Figure 2.6.

### LoRaWAN Data Link Layer & Packet Scheduling

In the previous section, it was discussed that LoRaWAN end-devices can connect to more than one gateway at once. This is useful for LoRaWAN applications, especially if an application where end-devices may be mobile is considered. In keeping with this, a LoRaWAN end-device sends uplinks to every gateway to which it is connected. This is then forwarded

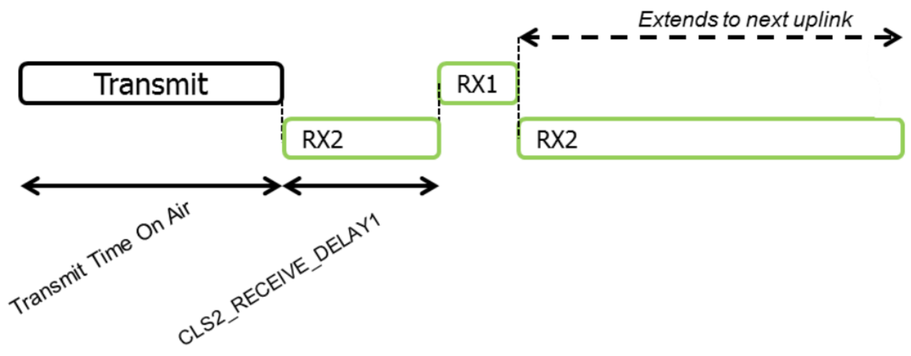
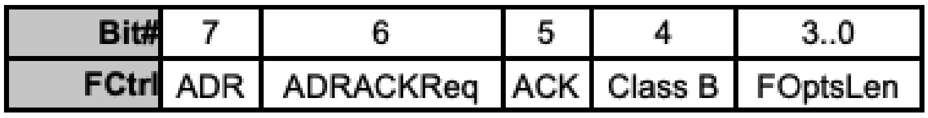


Figure 2.6: An illustration of LoRaWAN Class C operation [24].

from the gateway or gateways receiving it to the network server.

In the reference work for this project [31], an intricate description of the Class A uplink and downlink packet structure can be seen, and this will not be duplicated here. For the most part, the packet structure for Class B is the same as that of Class A, with one exception: the RFU bit in the FCtrl part of the packet in Class A is unused, but it gains a function in Class B operation. With this bit set to 1, the network server knows this device has switched to Class B operation, and it is prepared to use scheduled ping slots rather than uplink initiated $R_x$ windows.

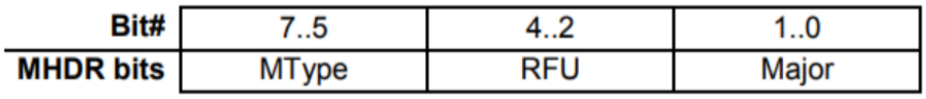


| Bit# | 7 | 6 | 5 | 4 | 3..0 |
|---|---|---|---|---|---|
| FCtrl | ADR | ADRACKReq | ACK | Class B | FOptsLen |

Figure 2.7: FCtrl slot breakdown in LoRaWAN Class B operation [24].

On the downlink, the structure of a Class B packet is exactly the same as that of Class A, the only difference in this mode of operation being the way in which they are scheduled and sent. This is to say that the same amount of data can be sent by a Class B downlink frame as in a Class A downlink frame.

It is also worth mentioning at this point that LoRaWAN Class B messages can be multicast - one message can be scheduled to be sent to many devices at the same time. Multicast messages only differ from unicast messages on some minor details. Firstly, a multicast message can not carry MAC commands in either the FOpt field or the downlink payload on port 0, the port reserved for configuration between the device and network. This is due to the fact that the same security and robustness cannot be guaranteed with multicast messages as with their unicast counterparts. Second, messages must be of type ‘unconfirmed.’ This means they will not require acknowledgement the LoRaWAN MAC layer. Accordingly, the ACK and ADRACKReq bits in the FCtrl slot must be set to 0 and the MType field in the MAC Header must be set to the unconfirmed value [24]. Though multicast does present an interesting opportunity for the application being discussed in this work, particularly regarding the dissemination to multiple end-devices of the same firmware update package, the protocols in this work are not designed to accommodate multicast which, within the allotted time, was decided to be considered as out of scope. This does, however, open up an interesting opportunity for the functionality to be investigated further as an extension to this work.



| **Bit#** | 7..5 | 4..2 | 1..0 |
|---|---|---|---|
| **MHDR bits** | MType | RFU | Major |

Figure 2.8: LoRaWAN MAC Header (MHDR) structure [31]

Devices on LoRaWAN must all minimally implement Class A operation, as previously discussed. The network, therefore, has been designed such that it can accommodate dynamic device switching between Class A and Class B. On the network, all end-devices begin operation as Class A devices. Upon receipt of a beacon lock message from the gateway, the device can subsequently enter Class B operation, now that it is synchronised with the gateway and, by extension, the network server.

In some applications, there will arise a scenario where the Class B device can no longer receive beacon messages from the gateway. In this case, the beacon and ping slots on the end device are widened gradually, to accommodate any clock drift that may have occurred over the time it is out of contact with the gateway.

In the event that the beacon is lost, the end-device will continue operating in Class B mode for 120 minutes, after which time it will switch back to Class A operation, until a beacon lock is re-established. This is known as beacon-less operation, and the device will use its own internal clock to keep time, expanding ping and beacon slots as previously mentioned to accommodate potential drift in the internal clock. If any beacon is received during this time, the device can operate beacon-less for 120 minutes again before switching back to Class A.

It is also worth noting that the LoRa physical layer operates on the 863-870 MHz frequency band, the open band in Europe. This frequency band is subject to strict duty cycle restrictions, which mean that devices can not be active - that is transmitting or receiving data - for more than 1% of their active time. Practically, this means choosing a certain period of time, e.g. a day, an hour, a minute, and setting a certain interval the device has to wait before transmitting after the preceding transmission. This scheduling is done by both the application and the network server, with packets transmitted in violation of the duty cycle being dropped by the network. This is important to the application at hand, because gateways and end-devices of all classes are subject to the same duty cycle restrictions, and this means there is no practical use of firmware transmission time as a metric in the later evaluation of the protocols designed for this work.

### 2.3.3 Firmware Updates over LoRaWAN

A previous work that will heavily influence many of the design decisions taken throughout the course of this research, and thus is worth highlighting individually, is"An Evaluation of LoRa Low-Power Wide-Area Technology for Firmware Update Transmission [31]." In this work, an experimental protocol for the transmission of firmware updates over the LoRaWAN infrastructure was designed and evaluated. This work details the use of LoRa Class A behaviour in transferring firmware updates from gateway to end node, highlighting the areas in which LoRa technology can improve upon existing solutions, and providing an important benchmark with which to compare any further research making use of the same technologies.

Furthermore, this thesis highlights many of the areas in which updating can be improved and even goes on to layout suggestions for developing protocols for firmware update transmission for constrained devices over LoRaWAN Class B.

# 3 Protocol Design

The following chapter will describe the process through which the protocols created for this project were designed. This discussion will begin with a quick overview of how the update transmission will work over LoRaWAN Class B. Moving on from this, the firmware packet assembly process will be described. Subsequently, some definitions for the chapter will be given to allow understanding of what will be used to evaluate the designs. Finally, the 3 protocols designed for this work will be discussed in detail. This chapter aims to give some sense of the design choices that were made during the creation of these protocols, their influences and differences, and what data will be relevant to their evaluation in the next chapter.

## 3.1 Design Considerations

### 3.1.1 Advantages of Using Class B over Class A

A Class B end-device will periodically contact the gateway/network server with an uplink to achieve beacon lock. Transmission of firmware segments can be done in response to these periodic uplinks, which can carry a payload in addition to being used for the purpose of beacon lock.

In [31], it is discussed that due to the fact packets must be scheduled in advance, the sequence for sending updates will always be one step out of synchronisation between the end-device and the update protocol. This is to say that, as explained in the referenced work, the uplink sent by the end-device, requesting packet $n$, will be responded to with a downlink containing packet $n - 1$.

Fortunately, Class B operates differently to Class A and will serve to eliminate this inefficiency. After an uplink is sent in Class B, the server will wait one ‘ping period,’ a term explained in section 2.3, before sending the first packet. This allows the packet received in response to be the exact packet that was requested, as no pre-scheduling is required. Class B, in this way, eliminates one of the inefficiencies found in the firmware update process over Class A even before entering the protocol design phase.

Another improvement, which will be discussed in more detail during the evaluation of the protocols designed here, is the number of total uplinks sent. The expectation, by using Class B, is that the number of uplinks required to transfer a firmware update delta will be significantly less than that of Class A. If the example of Class B operation given in the previous section is considered, where for every uplink there are four downlink ping slots opened, in a best case scenario where no packets are dropped, and no retransmission is required during the process, the Class B device will need one quarter of the uplinks to receive the same amount of data. This is another advantage inherent in using Class B over Class A.

### 3.1.2 Segment Size and Segment Indexing

The lowest possible downlink packet size made available by LoRaWAN is at a spreading factor of 12 and a bandwidth of 125kHz. For the purpose of this work this will be the packet size considered, though changing the packet size should be easily configurable and is an operation that can be performed on the application server. For optimal efficiency, an update should be completed in the minimum amount of packets necessary, and it is for this reason that with each downlink, the maximum size of 51 bytes will be used.

For any firmware delta over this size of 51 bytes, which most if not all update deltas will surpass, the image will need to be broken down by the server into $n$ packets of 51 bytes. As this work extends the work done in [31], the same packet structure is used in this work. This packet structure involves prepending an index to the firmware segment so it can be reassembled on the end-device upon completion of the transfer.

### 3.1.3 Error Correction

In the context of firmware update delivery, it is of the utmost importance that every segment reaches the end-node. In networking, there exists the construct of an **A**utomatic **R**epeat re**Q**uest (ARQ), which handles the loss or damaging of a packet during transmission by retransmitting the packet in question, using a pre-defined behaviour, upon detecting its loss.

Some error correction methods use positive acknowledgement, meaning that upon receiving a packet, the receiver in this case the end-device, will send an acknowledgement (ACK) to the receiver, notifying it that the segment in question has been received. In an ARQ protocol using positive acknowledgement, the server will determine if a segment has been lost using the acknowledgements it has received. In the simplest example, this could mean if a server receives an ACK for segment $n$, and subsequently packet $n + 2$, it could determine that packet $n + 1$ has been lost. In some cases, protocols will also use timers to determine packets have been lost. In such protocols if an acknowledgement is not received for segment $n$ within a predetermined amount of time, packet $n$ will be rescheduled. Once this determination has been made, the packet will be rescheduled using the rescheduling behaviour defined in its error correction protocol. In Figure 3.1, an example is given of a simple stop-and-wait ARQ protocol, using positive acknowledgement.

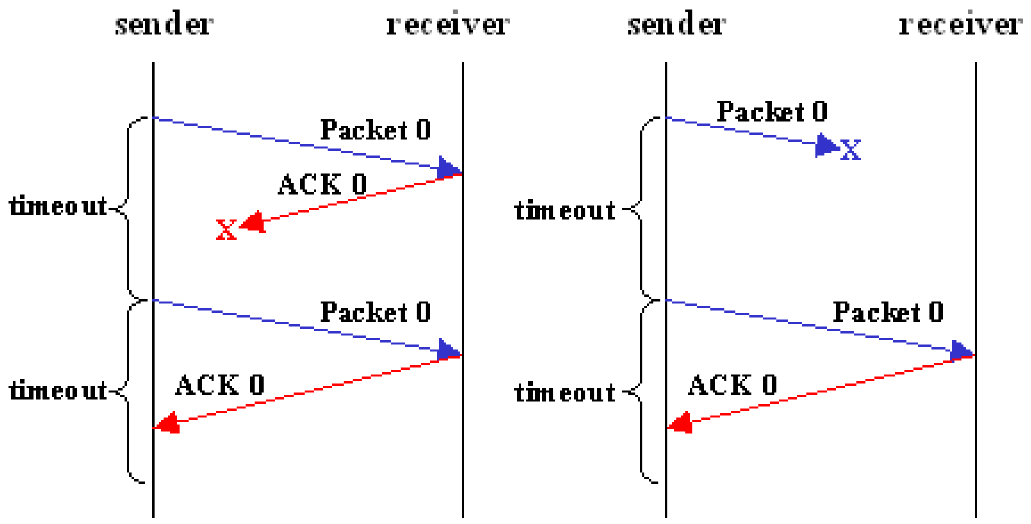


Figure 3.1: An example of Stop-and-Wait ARQ using positive acknowledgement from *https://www.isi.edu/nsnam/DIRECTED_RESEARCH/DR_HYUNAH/D-Research/stop-n-wait.html*.

Other examples of ARQ protocols will use negative acknowledgement. This is, as expected, the opposite of positive acknowledgement - packets that are dropped are determined by the receiver, and upon making this determination, the receiver will send a message to the server/sender, acknowledging the *loss* of a packet. Using this information, the server can reschedule the packet, again according to its pre-defined behaviour. An example of this can be seen in Figure 3.2, which shows NACKing in the context of the Go-Back-N ARQ protocol (for furhter explanation of this protocol, see [41]). In some protocols, NACK and ACK are used in conjunction. This is not the case, however, in any of the protocols designed for this work. The protocols designed here will use the periodic uplinks required to operate on LoRaWAN to send ACK/NACK messages to the server. Dropped packets can therefore be acknowledged by the packet indices not present in the uplink payload, in the case of ACK protocols, or indices present in the uplink payload, in the case of NACKing. This will become clearer as the protocols designed for this work are explained.

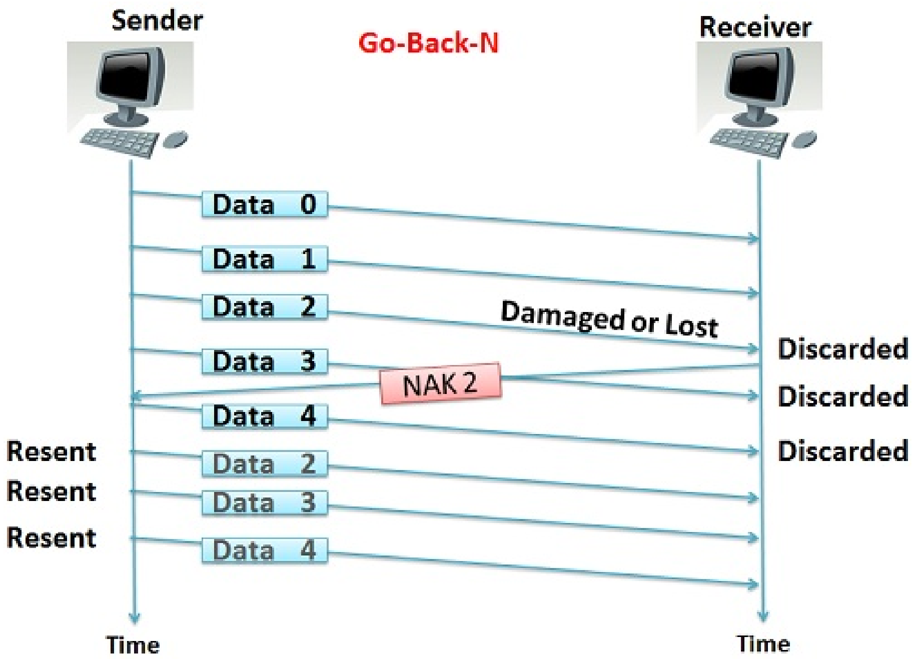


Figure 3.2: An example of NACKing in the context of the Go-Back-N protocol (explanation in [41]). Image shown here from *https://techdifferences.com/difference-between-go-back-n-and-selective-repeat-protocol.html*.

## 3.2 Design Metrics, Definitions & Considerations

In this project, the objective is to deliver an entire firmware update delta, accounting for transmission errors, at the optimal efficiency. To determine what is meant by ‘efficiency’ in this context, some evaluation metrics must be defined, and this is the intention of the following section.

### 3.2.1 Total Uplinks

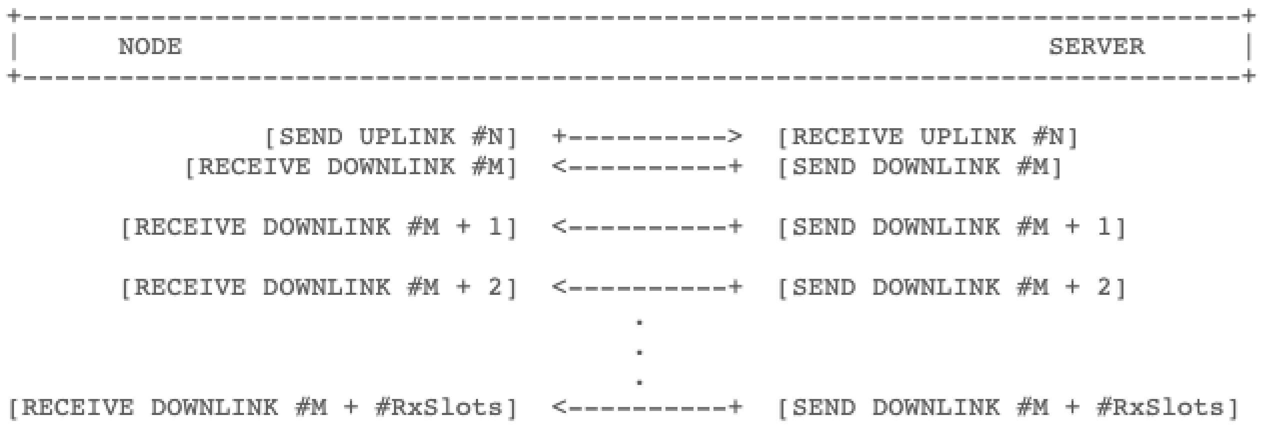
In order to send a complete firmware update package, a certain number of uplinks must come from the LoRaWAN end-device, to both request and acknowledge firmware update segments. Each one of these uplinks takes a certain amount of computational resource and uses a certain amount of energy. Transmission is one of the most expensive operations a low-power device can perform, and thus the less transmissions a device sends, the less energy it will consume, making the process more efficient.

In a Class A firmware update, such as those defined in [31], the number of uplinks will be, in the best case, equal to the number of segments in the update plus an additional one for the first uplink, for which no data will be received. As previously explained, this will be improved upon by the use of Class B.

It is worth noting at this point that this metric will be used in the comparison of Class A with Class B. More emphasis is put on this metric in this comparison as when comparing the three protocols designed for Class B, as the three protocols, operating over Class B, will show similar results, and will not highlight the meaningful differences between the protocols, whereas in comparing Class A and Class B the difference is expected to be dramatic.

### 3.2.2 Effectiveness Ratio

An effective uplink is a communication from the device which receives in response meaningful and unique data. In the context of this work, the definition of an effective uplink changes slightly when compared with the definition given in the work it extends [31]. In the reference work, an uplink is considered effective if unique data is received in the downlink response. In this work, however, a number of downlinks equal to the number of $R_x$ slots will be sent per uplink in keeping with the definition in the LoRaWAN specification [24] for Class B operation. This requires the definition of an effective uplink to be modified slightly as follows: In this work, an effective uplink is an uplink for which all segments received in response contain unique and meaningful data. An example of this is seen in Figure 3.3:



```
+------------------------------------------------------------------------------+
|      NODE                                                        SERVER      |
+------------------------------------------------------------------------------+

              [SEND UPLINK #N]  +---------->  [RECEIVE UPLINK #N]
         [RECEIVE DOWNLINK #M]  <----------+  [SEND DOWNLINK #M]

     [RECEIVE DOWNLINK #M + 1]  <----------+  [SEND DOWNLINK #M + 1]

     [RECEIVE DOWNLINK #M + 2]  <----------+  [SEND DOWNLINK #M + 2]
                                      .
                                      .
                                      .
[RECEIVE DOWNLINK #M + #RxSlots]  <----------+  [SEND DOWNLINK #M + #RxSlots]
```

Figure 3.3: Effective uplink sequence diagram

Effective uplinks, as a metric alone, however, will not give the full picture of how efficient an update transmission was. For this reason, a new metric has been introduced for this work - Effectiveness Ratio. This metric, defined simply as the average number, in a transmission, of unique and meaningful packets received per downlink. The effectiveness ratio hopes to give an indication of the quality of the data received with each downlink, with quality here meaning the number of unique, useful packets received. A higher effectiveness ratio will, in general, mean less downlinks were required to send an update over the air.

### 3.2.3 Uplink Size

In transmission of uplinks, it makes sense to assume that with more data to transfer, the amount of time a device needs to spend actively transmitting will be increased, which will in turn lead to a higher energy consumption. The less data a device has to send with each uplink, the less data it has to transmit to the gateway, thus meaning a smaller uplink size will lead to a more efficient firmware update.

### 3.2.4 Ineffective Uplinks

In the case of ineffective uplinks, there are two different occurrences where an uplink will be considered ineffective. These two types of ineffective uplink, named after their respective analogue in [31], are described in the following text. It is important to note that while the two ineffective uplink types are considered analogous to their counterparts in [31], their definitions will be modified in this work to fit a Class B context.

#### Ineffective Data Uplink

In keeping with the definition of what an effective uplink is given in the previous section, an ineffective uplink, for the purpose of this work, will be considered to be an uplink for which the corresponding sequence of downlinks contains one or more repeated segments. This is illustrated in Figure 3.4.

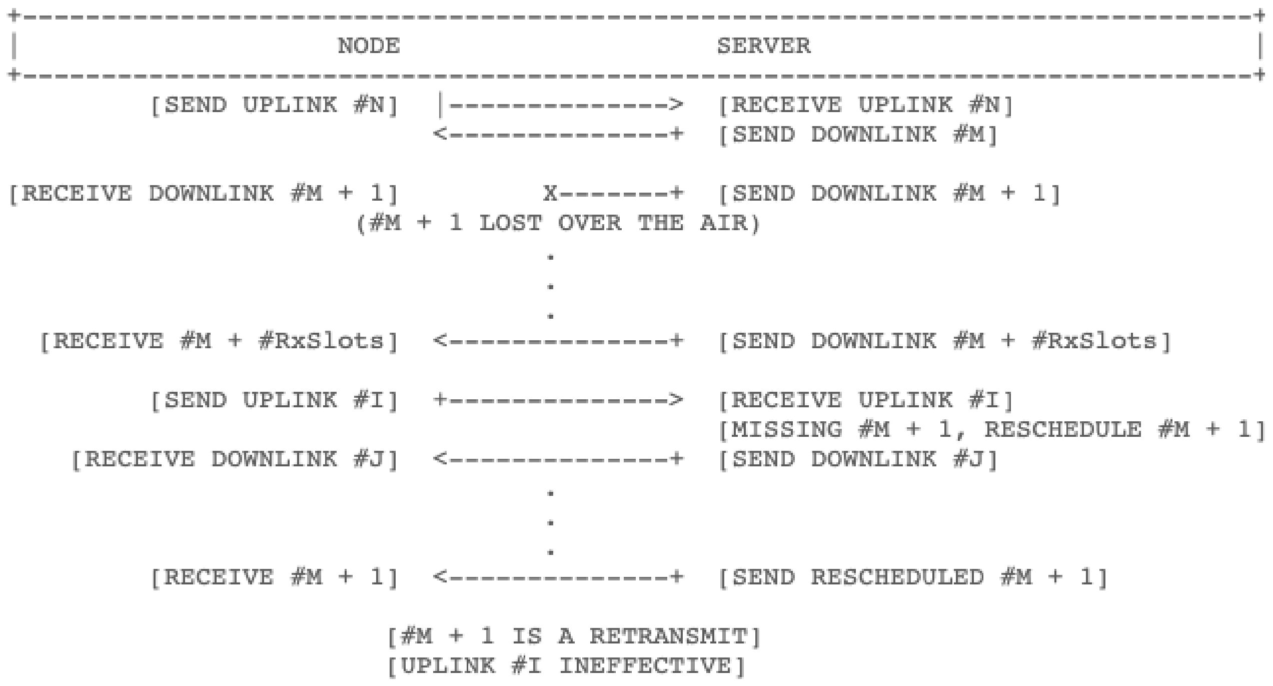
#### Total Ineffective Response Uplink

A total ineffective response uplink is an uplink for which no data is received in **any** of the ping slots. This type of uplink can occur for two different reasons:

1. No more segments to be sent
2. Gateway restricted by duty cycle (see below)

In radio communication, a *duty cycle* is the name given to the percentage of a devices total operational time it is permitted to be active on the band in which it operates. As LoRaWAN uses the unlicensed band, it is subject to EU limitations for this band and as such its devices can not be active for more than 1% of their operational time, i.e. their duty cycle is 1%.

On LoRaWAN devices, this means, in practice, choosing a window of time, for example 100s, and ensuring that in this 100 seconds window the device only transmits for at most 1 second. It is also important to note that both devices and gateways are restricted to the same duty cycle, meaning no large transmission time difference is expected between Class A and Class B operation - the reason transmission time is not used as a metric in this work.

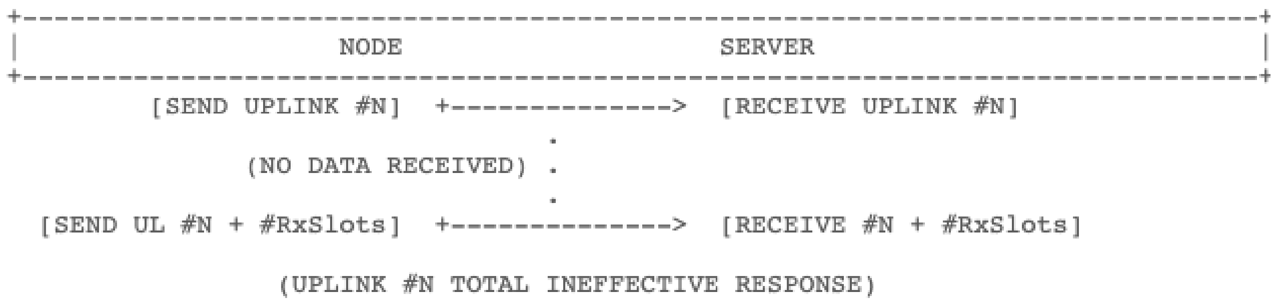


```
+------------------------------------------------------------------------------+
|                    NODE                     SERVER                           |
+------------------------------------------------------------------------------+
         [SEND UPLINK #N]  |--------------->  [RECEIVE UPLINK #N]
                           <---------------+  [SEND DOWNLINK #M]

[RECEIVE DOWNLINK #M + 1]         X-------+  [SEND DOWNLINK #M + 1]
                       (#M + 1 LOST OVER THE AIR)
                                  .
                                  .
                                  .
  [RECEIVE #M + #RxSlots]  <---------------+  [SEND DOWNLINK #M + #RxSlots]

         [SEND UPLINK #I]  +--------------->  [RECEIVE UPLINK #I]
                                              [MISSING #M + 1, RESCHEDULE #M + 1]
    [RECEIVE DOWNLINK #J]  <---------------+  [SEND DOWNLINK #J]
                                  .
                                  .
                                  .
         [RECEIVE #M + 1]  <---------------+  [SEND RESCHEDULED #M + 1]

                         [#M + 1 IS A RETRANSMIT]
                         [UPLINK #I INEFFECTIVE]
```

Figure 3.4: Ineffective data uplink sequence diagram

Using Class B also improves over protocols designed for Class A, reducing the likelihood of uplinks being **totally** ineffective - when no data at all is received for the uplink in question, as more than one segment is sent per downlink.

An illustration of a total ineffective response uplink can be found in Figure 3.5.



```
+------------------------------------------------------------------------------+
|                    NODE                     SERVER                           |
+------------------------------------------------------------------------------+
         [SEND UPLINK #N]  +--------------->  [RECEIVE UPLINK #N]
                                  .
               (NO DATA RECEIVED) .
                                  .
  [SEND UL #N + #RxSlots]  +--------------->  [RECEIVE #N + #RxSlots]

                 (UPLINK #N TOTAL INEFFECTIVE RESPONSE)
```

Figure 3.5: Total ineffective uplink example

With some understanding of the considerations taken during the design phase, the three protocols designed for this work will now be described in detail.

## 3.3 Uplink & Downlink Packet Structure

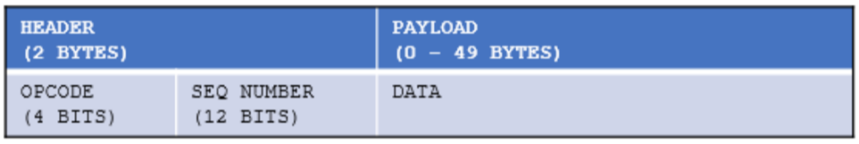
In every protocol designed as part of this dissertation, the same basic packet structure is used on both the uplink and the downlink. These protocols differ in how packets are acknowledged and rescheduled, but the basic structure will remain as described here. It is also worth noting that this structure is very similar to that described in [31], with mostly minor adaptations for use on Class B. In cases where structure has been modified, the packets have still been designed such that they will work, without modification to the server or device code, with the Class A protocols described in the referenced work. This is beneficial in applications where certain devices in a system do not support Class B operation, as the protocol allows for updates to be transmitted to both classes. Furthermore, if this work was extended and a control protocol for the update process was designed, it could allow for the configuration of the server to send an update to multiple devices of both classes at the same time.

In LoRaWAN applications, the application payload will fill the Frame Payload slot of the uplink/downlink payload described in the LoRaWAN specification [24]. As previously mentioned, for the purpose of this work, the worst case payload size of 51 bytes will be considered, though no change should be required to use the same packet structure with higher payload sizes.

### 3.3.1 General Packet Structure

The general packet structure for a packet in this work, as in [31], contains a 2 byte header, consisting of a 4 bit opcode, which allows different types of packet to be determined. In this work, two opcodes are defined, 0 meaning a normal packet being sent, and 1 a final packet. These opcodes apply to both uplink and downlink packets, if a downlink is received with opcode 1, the device will respond with opcode 1. If the device has responded with opcode 1, and all expected packets are accounted for, the server marks the update as finished. If not all packets are acknowledged, the server will resend with opcode 1 until all segments are accounted for.

The header also contains a 12 bit sequence code. Though not required in this work, the sequence code is intended to aid in firmware reassembly on the device. It could also become important if packet indices begin to overflow their allocated 1 byte (i.e. if there are more than 255 firmware segments). In the packet payload, which will occupy, at most, the remaining 49 bytes, contains some amount of data, of size less than or equal to 49 bytes. This general packet structure is outlined in the diagram in Figure 3.6, from [31].

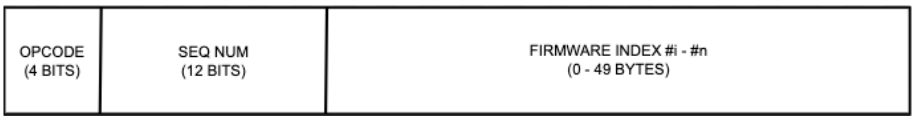


| HEADER (2 BYTES) | | PAYLOAD (0 – 49 BYTES) |
|---|---|---|
| OPCODE (4 BITS) | SEQ NUMBER (12 BITS) | DATA |

Figure 3.6: General Uplink & Downlink Packet Structure [31].

## 3.3.2 General Uplink Packet Structure

Following from the general packet description, uplink packets will contain the same header described in said description. In the uplink data payload sector, however, there are specific data that will be carried. This section of the packet will contain a list of indices which, depending on the protocol being implemented, will represent indices of packets that are being ACKed or NACKed. End-devices will maintain a queue of which packets they have received or lost and use this information to populate this field. A diagram of this structure can be seen below.



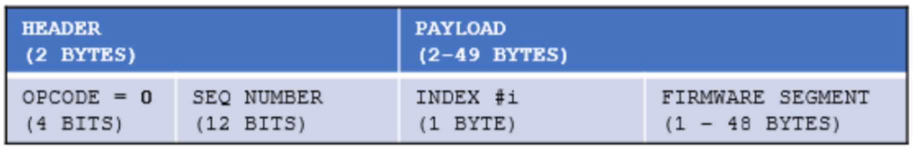
| | | |
|---|---|---|
| OPCODE (4 BITS) | SEQ NUM (12 BITS) | FIRMWARE INDEX #i - #n (0 - 49 BYTES) |

Figure 3.7: Uplink Packet Structure.

## 3.3.3 General Downlink Packet Structure

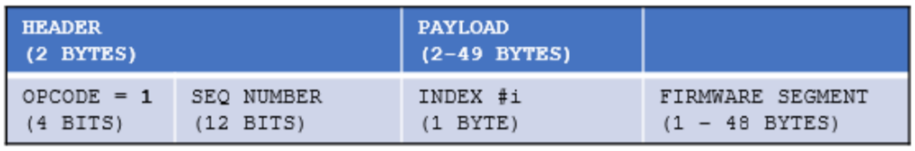
Downlink packet headers do not differ from the general/uplink structure, as expected. The payload for a downlink is unchanged from that described in the reference work, and Figure 3.8 shows the diagram from [31] of a downlink packet in the normal case.

Figure 3.9 shows the diagram for a final downlink packet - notice the opcode of 1 instead of 0.



| HEADER (2 BYTES) | | PAYLOAD (2-49 BYTES) | |
|---|---|---|---|
| OPCODE = 0 (4 BITS) | SEQ NUMBER (12 BITS) | INDEX #i (1 BYTE) | FIRMWARE SEGMENT (1 - 48 BYTES) |

Figure 3.8: Normal downlink packet structure [31].



| HEADER (2 BYTES) | | PAYLOAD (2-49 BYTES) | |
|---|---|---|---|
| OPCODE = 1 (4 BITS) | SEQ NUMBER (12 BITS) | INDEX #i (1 BYTE) | FIRMWARE SEGMENT (1 - 48 BYTES) |

Figure 3.9: Final downlink packet structure [31].

## 3.4 Protocol 1: Piggybacked Selective Repeat ARQ

The first protocol being evaluated in this work is the most efficient protocol designed in [31]: Piggybacked Selective Repeat ARQ, and an in detail description of how the protocol works over LoRaWAN Class A can be found in the referenced work. In this work, however, a slight modification has been made to the protocol to adapt it for use in Class B applications. This modified version of the protocol will be what is described in the following text, but the protocol from the previous dissertation may be referenced in relation to the adapted version being described here, so as to mention where the adaptation differs from the originally described protocol.

### 3.4.1 Implementation

In the work that this dissertation has extended, the Piggybacked Selective Repeat protocol accounts for only one 8 bit index acknowledgement at a time on the uplink. This means that the end-device can only uplink acknowledgement for one update segment at a time, if it conforms to the specification given for the protocol.

If the application of this protocol over LoRaWAN Class B is considered, the protocol as defined will not work. Between uplinks in Class B operation, multiple downlinks are sent, sending multiple segments It follows that multiple segments should also be able to be acknowledged. This work will make a minor modification to the protocol defined in [31], in order to remedy this. Namely, a change is made, described in the preceding general uplink packet structure section, to the payload field of the packet. In [31], only one 8 bit index is permitted per uplink. This adapted version of the protocol will allow packet indices to be queued on the device, and sent in one uplink. Uplink size in the LoRaWAN specification is not restricted to 8 bits, rather uplink payloads can also contain 51 bytes. Each index will be one byte, with sequence number incrementing if the overall index reaches a number greater than 255, and the index restarting from 0. Uplink in this protocol will be the same as the packet seen in Figure 3.7. Indices sent in uplinks for this protocol are **positive acknowledgements** (ACKs) - i.e. If the server sends [#0, #1, #2, #3, #4] and receives in response an uplink payload containing [#0, #1, #2, #4], the server can determine that the end-device has received segments 0 - 2, dropped 3 and received 4.

Downlink packets will be structured as seen in Figure 3.8 and 3.9, depending on whether or not the final packet is being sent. Downlink packets are sent one-by-one, each containing one index and segment respectively. The difference in this version of the protocol is that, when using the Class B implementation, more than one of these packets can be sent per uplink.

On the server, before the update begins, the raw firmware update data is divided into segments, and then packaged as a series of downlink packets. A send queue is then initialised, containing all packets, in sequence, of the update. A separate, immutable store of these packets is also kept, to avoid having to rebuild packets in the case of packet loss. If a packet is lost and the server becomes aware of this through missing acknowledgement, the packet is rescheduled - in this protocol packets are rescheduled by appending them to the end of the send queue, removing the need for a 'sliding window,' used in a true implementation of selective repeat. Also worth mentioning at this point is the fact that for this protocol, and indeed all three in this work, timers are not used, due to the fact that LoRaWAN will require an eventual uplink. This is to say that an eventual uplink response is, for this purpose, a certainty. Furthermore, choosing a timeout in this case would be difficult. LoRaWAN devices can often be out of network reception for extended periods of time, especially if end-devices are in some way mobile, but this does not necessarily mean that packet loss has occurred - again, packets lost will be confirmed in the guaranteed eventual uplink.

This is also the only protocol in this work that was tested with both Class A and Class B. In the case of this protocol, and indeed all the protocols in this work, it is easy to configure on the server how many receive windows an end-device is setup to open in between uplinks. If set to 1, the protocol will work with Class A devices. If set to greater than 1, the server will operate with Class B devices. The full operation of this protocol can be seen in Figure 3.10

### 3.4.2 Adapted Piggybacked Selective Repeat Sequence Diagram

```
          +------------------------------------------------------------------------------+
          |                       NODE                       SERVER                      |
          +------------------------------------------------------------------------------+

                         [SEND UPLINK 1]  +----------------->  [RECEIVE UPLINK 1]
                   (REQUEST SEQUENCE #1)                      [SCHEDULE SEQUENCE #1]
|                   [WAIT PING_PERIOD 1]         .            [WAIT PING_PERIOD 1]
                                                 .
                                                 .
                    [RECEIVE SEGMENT #1]  <-----------------+  [SEND SEGMENT #1]
                   (ADD #1 TO ACK QUEUE)
                    [RECEIVE SEGMENT #2]  <-----------------+  [SEND SEGMENT #2]
                   (ADD #2 TO ACK QUEUE)         .
                                                 .
                                                 .
                    [RECEIVE SEGMENT #N]  <-----------------+  [SEND SEGMENT #N]
                   (ADD #N TO ACK QUEUE)                      (N = #RxSlots)
                         [SEND UPLINK 2]  +----------------->  [RECEIVE UPLINK 2]
                        (SEND ACK QUEUE)                      (CHECK ACKS RECEIVED)
                                                              (#1 -> #N DELIVERED)
                        (REQUEST SEQ #2)                      [SCHEDULE SEQ #2]
                      [WAIT PING_PERIOD]         .            [WAIT PING_PERIOD]
                                                 .
                                                 .
               [RECEIVE SEGMENT  #N + 1]  <-----------------+  [SEND SEGMENT #N + 1]
               (ADD #N + 1 TO ACK QUEUE)         .
                                                 .
                                                 .
             (#I NOT ADDED TO ACK QUEUE)         X---------+  [SEND SEGMENT #I]
                                    (PACKET #I LOST OVER THE AIR)
                                                 .
                                                 .
                                                 .
                    [RECEIVE SEGMENT #J]  <-----------------+  [SEND SEGMENT #J]
                                                              (END OF SEQ #2)
                         [SEND UPLINK 3]  +----------------->  [RECEIVE UPLINK 3]
                        (SEND ACK QUEUE)                      (CHECK ACKS RECEIVED)
                                                              (ACK #I MISSING)
                                                              (RESCHEDULE #I)
                                                 .            [SCHEDULE SEQ #3]
                                                 .
                                                 .
                         [SEND UPLINK M]  +----------------->  [RECEIVE UPLINK M]
                   (M REQUESTS LAST SEQ)                      (ACKS OK, SCHEDULE SEQ #N)
                                                 .            (N, LAST SEQ, CONTAINS SEG #I)+
                                                 .                                          |
                                                 .                                          |
+------------------+[RECEIVE SEGMENT #I]  <-----------------+  [SEND SEGMENT #I]<------------+
|                (#I ADDED TO ACK QUEUE)         .
+->(UPLINK M INEFFECTIVE, DUPLICATE #I)          .
                                                 .
           [RECEIVE FINAL SEG, OPCODE 1]  <-----------------+  [SEND FINAL SEG, OPCODE 1]
                      (ADD TO ACK QUEUE)
           [SEND FINAL UPLINK, OPCODE 1]  +----------------->  [FINAL UPLINK RECEIVED]
                                                              (OPCODE 1, ALL ACKS PRESENT)
                                                              (UPDATE FINISHED)
```

Figure 3.10: Sequence Diagram for Class B Adapted Piggybacked Selective Repeat Protocol

### 3.4.3 Initial Evaluation

This protocol has one main advantage over the protocol it is derived from. This is that it allows less uplinks to be sent in transmission of the same amount of data. As discussed previously, this means that less energy will be used in transmission of the data over Class

B. This does come at the cost, however, of a larger uplink size and therefore a longer transmission time. It will have to be determined, when proper infrastructure is available, if this trade-off is worthwhile, but this work will assume that it is, under the hypothesis that for every byte in a Class A operation of this protocol, or the protocol from which it is derived, a 16 bit header is also sent, but in Class B operation this 16 bit header is sent $\frac{1}{n}$ of the times it is sent over Class A, where $n$ is the number of ping slots opened per Class B uplink.

## 3.5 Protocol 2: True Selective Repeat ARQ

In order to serve as a benchmark for the piggybacked, windowless selective repeat adaptations in this work, a sliding window, true implementation of selective repeat ARQ was chosen to be implemented.

### 3.5.1 Implementation

Selective Repeat ARQ makes use of a so called 'sliding window' when it sends packets. This sliding window, of length $n$, initially transmits packets $1 - n$. Upon receiving acknowledgement for packet 1, the window slides up one place, and this continues while sequential ACKs are received. If a packet is dropped, the window will stay in its current position, and any unACKed packets from within the bounds of the window in its current position will be retransmitted. It is important to note that the window only moves when its first sent packet is acknowledged.

The only difference between this protocol and a selective repeat protocol seen in networking is, again, it forgoes the use of timers, for the same reasons given previously. Aside from this difference, packet rescheduling and flow control are performed in the same way. In this implementation, window size is configurable on the server, and it makes the most sense to choose a window size equal to the number of ping slots available for Class B operation - i.e. If there are 5 ping slots per beacon period, the sliding window will be of size 5.

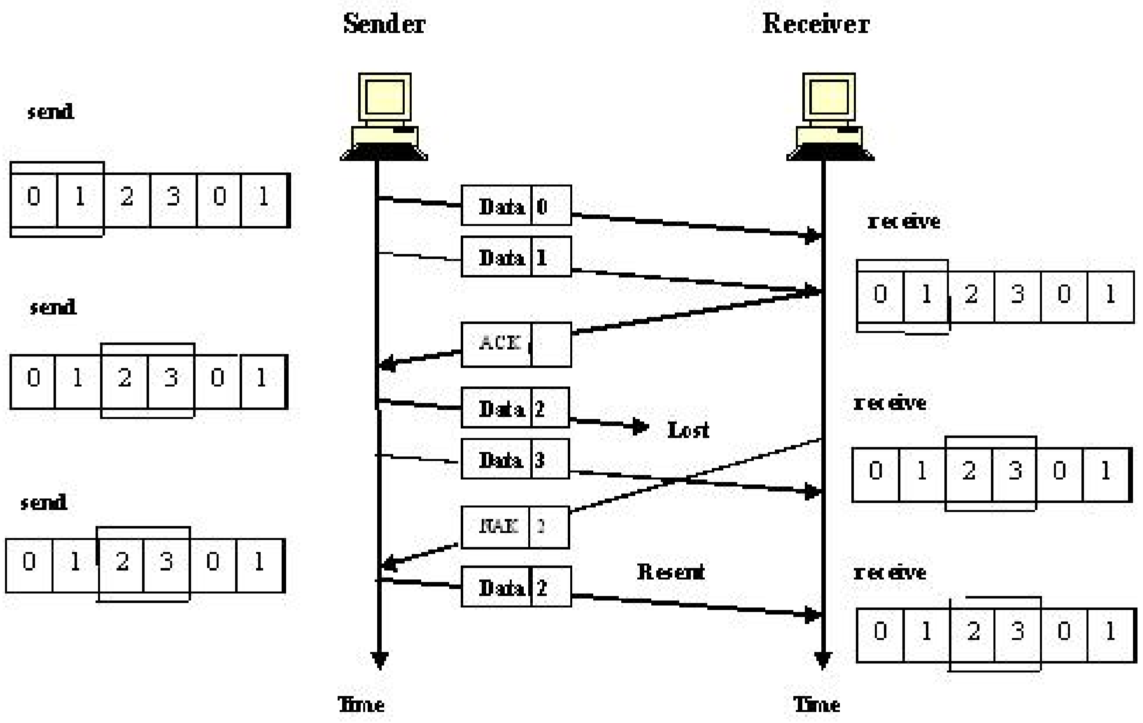


Figure 3.11: Sequence Diagram for Selective Repeat ARQ from *http://enggedu.com/tamilnadu/university_questions/question_answer/be_mj_2007/5th_sem /cse/CS1302/part_b/12_b.html.*

## 3.5.2 Initial Evaluation

This protocol will not perform optimally compared to the other protocols described in this work. Due to the sliding window that halts in the case of out of sequence delivery, if an uplink with out of sequence acknowledgements is sent, the resulting sequence of downlinks will not necessarily use all of the ping slots available to the server, and as such it will, in any non-perfect case, take more uplinks to transfer the same amount of data as with the other protocols. Compared with the previously described protocol, this protocol will have similar uplink size, also meaning no optimisation to this end. It is expected the results in the next chapter will show this protocol to perform the worst of the three listed in this work.

## 3.6 Protocol 3: NACKed Piggybacked Selective Repeat ARQ

### 3.6.1 Implementation

The third protocol is almost exactly the same as the first protocol described here. The only difference between the two is that rather than sending positive acknowledgements for packets, only lost packet indices are sent. The device keeps track of out of sequence packets, and sends indices which have not been received. In the case of rescheduled packets, the device can check which segments it still needs to receive and deduce the next sequence of packets it should receive, provided it knows the number of $R_x$ slots it opens per uplink.

## 3.6.2 NACKed PSR ARQ Sequence Diagram

```
          +-------------------------------------------------------------------------------+
          |                        NODE                       SERVER                       |
          +-------------------------------------------------------------------------------+

                         [SEND UPLINK 1]  +----------------->  [RECEIVE UPLINK 1]
                   (REQUEST SEQUENCE #1)                       [SCHEDULE SEQUENCE #1]
                    [WAIT PING_PERIOD 1]         .             [WAIT PING_PERIOD 1]
                                                 .
                                                 .
                    [RECEIVE SEGMENT #1]  <-----------------+  [SEND SEGMENT #1]

                    [RECEIVE SEGMENT #2]  <-----------------+  [SEND SEGMENT #2]
                                                 .
                                                 .
                                                 .
                    [RECEIVE SEGMENT #N]  <-----------------+  [SEND SEGMENT #N]
                                                               (N = #RxSlots)
                         [SEND UPLINK 2]  +----------------->  [RECEIVE UPLINK 2]
                 (SEND EMPTY NACK QUEUE)                       (CHECK NACKS RECEIVED)
                                                               (#1 -> #N DELIVERED)
                        (REQUEST SEQ #2)                       [SCHEDULE SEQ #2]
                      [WAIT PING_PERIOD]         .             [WAIT PING_PERIOD]
                                                 .
                                                 .
               [RECEIVE SEGMENT  #N + 1]  <-----------------+  [SEND SEGMENT #N + 1]
                                                 .
                                                 .
                                                 .
                (#I ADDED TO NACK QUEUE)         X----------+  [SEND SEGMENT #I]
                                    (PACKET #I LOST OVER THE AIR)
                                                 .
                                                 .
                                                 .
                    [RECEIVE SEGMENT #J]  <-----------------+  [SEND SEGMENT #J]
                                                               (END OF SEQ #2)
                         [SEND UPLINK 3]  +----------------->  [RECEIVE UPLINK 3]
                       (SEND NACK QUEUE)                       (CHECK NACKS RECEIVED)
                                                               (NACK #I PRESENT)
                                                               (RESCHEDULE #I)
                                                 .             [SCHEDULE SEQ #3]
                                                 .
                                                 .
                         [SEND UPLINK M]  +----------------->  [RECEIVE UPLINK M]
                   (M REQUESTS LAST SEQ)                       (NO NACKS PRESENT, SCHEDULE SEQ #N)
                                                 .             (N, LAST SEQ, CONTAINS SEG #I)+
                                                 .                                           |
                                                 .                                           |
+------------------+[RECEIVE SEGMENT #I]  <-----------------+  [SEND SEGMENT #I]<------------+
|                                                .
+->(UPLINK M INEFFECTIVE, DUPLICATE #I)          .
                                                 .
           [RECEIVE FINAL SEG, OPCODE 1]  <-----------------+  [SEND FINAL SEG, OPCODE 1]

           [SEND FINAL UPLINK, OPCODE 1]  +----------------->  [FINAL UPLINK RECEIVED]
                                                               (OPCODE 1, NO NACKS RECEIVED)
                                                               (UPDATE FINISHED)
```

Figure 3.12: Sequence Diagram for Class B NACKed Piggybacked Selective Repeat Protocol

## 3.6.3 Initial Evaluation

This protocol, in terms of total uplinks, will show the same performance as its positively acknowledged counterpart, it is expected. Where it will show some increase in performance is in the size of uplink packets, and thus the time spent transmitting uplinks. Under the assumption that less packets will be dropped than successfully transmitted, this protocol using NACKs instead of ACKs could significantly reduce the number of 8 bit indices sent in its uplinks. For example, if a certain test drops 20% of the packets sent, the NACKed version of the protocol will transmit 25% of the 8 bit indices that the ACKed version would send. This is expected to be reflected in the experimental results.

# 4 Implementation and Evaluation

The previous chapter gave an overview of the protocols designed for use in this work, and further, an indication as to why they were designed in the way that they were. This chapter will discuss the implementation of these designs, and their subsequent evaluation, comparing performance briefly between Class A and Class B. Moving on, this chapter will uncover the protocol for Class B transmission which was most able to address the issues set out in the preceding chapters with regard to the previously established metrics.

This chapter will also go over issues faced during the implementation of the protocols, including a major issue that was cause for a significant change to the implementation as a whole.

## 4.1 Issues - Pervasive Nation

When the research for this work began, an initial meeting was held with Pervasive Nation, the IoT research group and LoRaWAN infrastructure provider for this project. The purpose of this meeting was to establish the feasibility of getting a project working over Class B on the Pervasive Nation infrastructure. It was mentioned that test applications had already been deployed to a restricted Class B testbed on the PN infrastructure, and the intention was to make this available to all users of Pervasive Nation in the near future - mid to late winter was forecast at the time of this meeting.

Over the course of the next few months, a string of configuration problems were with the sample code provided by Pervasive Nation for Class B devices. This was not an issue, as it was to be expected, but it did mean some time without contact with Pervasive Nation as the issues were resolved and code was, eventually, flashed to the devices.

Over this time, however, several staffing changes occurred at Pervasive Nation, with the group now being restricted to one employee at TCD. With less staff to deal with new projects and user issues, the Class B project was put on hold by Pervasive Nation. This was discovered upon contacting Pervasive Nation in January.

With Class B infrastructure being unavailable, and with no plans from PN to implement it before the deadline for this work, new solutions were examined. Several ideas were examined with relation to improving upon the Class A protocols seen in [31], as well as the possibility of designing a control protocol for use in Class A updates. Upon investigation, the protocols in [31] were found to be close to optimal, and there were deemed to be no opportunities for extensive changes. In the case of designing a control protocol for Class A updates, a solution was needed, however with the research done up to this point focusing on transmission protocols, and the limited time available due to these issues, it was decided that this was also not feasible.

After deliberation, it was suggested that, using [31] and the LoRaWAN specification [24], it was possible that parameterised device emulations could be written for Class B devices. This allowed the same basic server logic to be used for each protocol as in a real device application, varying only in how packets were transferred.

Despite the absence of Class B infrastructure posing quite a major issue, the emulation of devices allowed the research done in this work to produce meaningful results. The intention is still, however, that the protocols designed here be implemented in the future on real-world devices when the infrastructure becomes available. This will be discussed further in the future work section of the final chapter.

## 4.2 Hardware and Software Platforms

### 4.2.1 Devices

#### Hardware

Due to the issues discussed in the last section, a hardware platform was only available for use with Class A tests. This came in the form of a board produced by Pervasive Nation built for use on their infrastructure. This board was designed with the intention that it could be easily integrated into existing systems to allow sensors/actuators to communicate over LoRaWAN . It uses the ST Microelectronics STM32L0 processor and SX1726 LoRa radio in conjunction to achieve this communication. The popularity of the STM32xx microprocessor family makes resources and community support readily available, and this proved to be helpful during the resolution of the aforementioned configuration issues. An overview of this board is shown in Figure 4.1, taken from the Pervasive Nation documentation. Not highlighted in this picture is the antenna that allows the device to communicate over LoRaWAN .

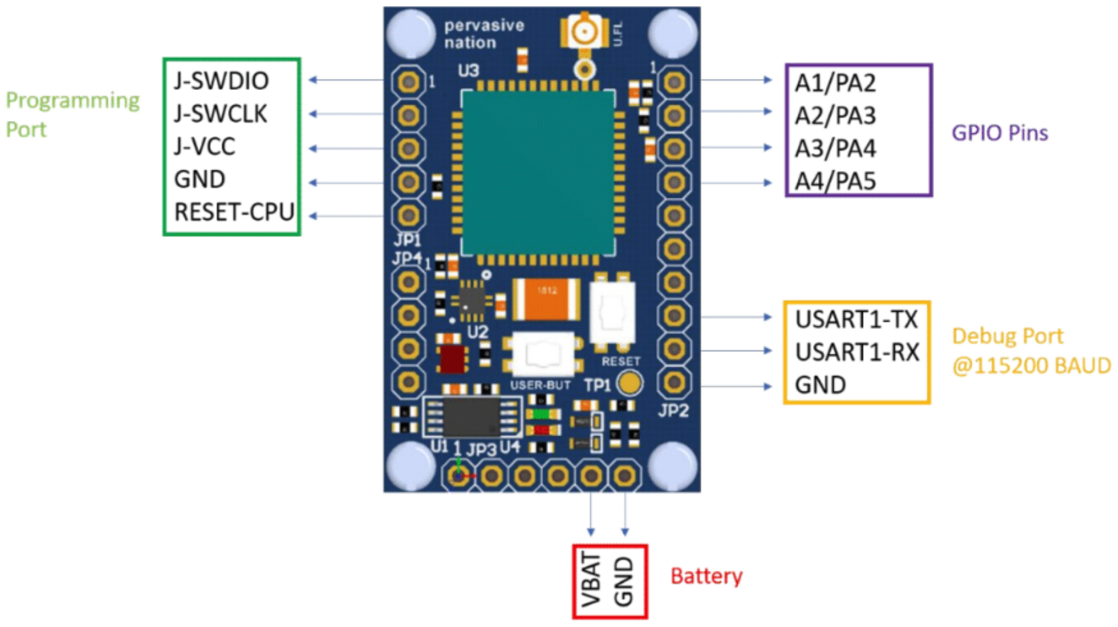


Figure 4.1: An overview of the Pervasive Nation LoRa board, from their documentation.

### Software - Class A Device

The PN board could not be programmed directly over USB, and required the use of an intermediate device to allow software to be flashed. After the initial meeting with Pervasive Nation, a Keil uVision project was provided by their staff for use in writing programs for the device. By using a previously configured project as the basis for the work done in this project, some of the initial configuration steps were eliminated, though configuration for programming the device was still difficult, as mentioned in the previous section of this chapter.

After correctly configuring the project, the device could be flashed from Keil uVision, through the intermediate 'ST-Link' compatible board as shown in Figure 4.2. The language used for all code put on real-world devices was C, as is standard in the embedded device space.

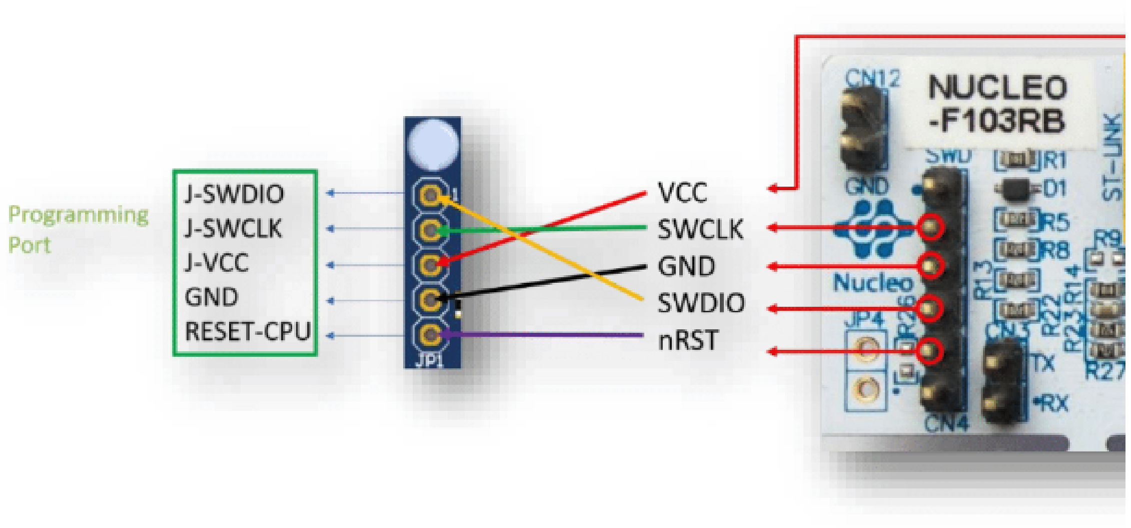


Figure 4.2: Setup for programming the PN LoRa board through an intermediate device, from the PN documentation.

### Class B Emulation

In order to overcome the lack of available infrastructure for Class B applications, software emulations of Class B devices were instead created. These emulations were parameterised using data from both the LoRaWAN specification [24] and the reference work [31]. This allowed the emulation of device performance at different spreading factors with a reasonable degree of accuracy, though this could only be verified by the comparison of real-world devices with emulations. For this reason, an emulation of a Class A device was also implemented, to establish credibility for emulated devices.

Device emulations were implemented in Python. Uplinks were replaced with HTTP requests to the application server, which would respond with data in the exact same encoded byte-stream format as would be expected by real devices. The number of segments the server would respond with deviated slightly from the real application server, however. In the case of a real device server, upon receiving an uplink, the server schedules packets to be sent one by one, leaving the downlinking of these packets in correct the ping slots to be decided by the LoRa gateway/network server. For the emulation servers, devices would be sent a number of packets equal to the number of receive windows configured on the emulated device. This had no bearing on the resulting emulation, so it implemented on the emulation in this way rather than sending packets one at a time to save on unnecessary HTTP requests and to allow the number of uplinks to be consistent with the expected real-world numbers. Once these segments were sent to the device, the device would drop packets at a spreading-factor dependent percentage based on the data collected regarding packet loss at different spreading factors in [31].

This emulation allowed enough data to be gathered to result in meaningful data, which it is expected in future real-world device testing, will prove to be a satisfactory estimation of real-world data.

### 4.2.2 Servers

#### Infrastructure

For communication with real devices, servers were run on AWS EC2 instances which allowed them to be exposed to the wider Internet. This allowed the Data Access Sub System (DASS) API, exposed by the LoRa network server to allow applications to schedule and receive transmission data, to use callback URLs to notify the server of delivery of packets.

In the case of servers written for communication with Class B emulators, all application servers and devices were run in a local environment, as no communication with other devices on the Internet was required.

#### Software

In the case of the server communicating over the PN infrastructure, the server would receive uplinks from the device and, in response, use the previously mentioned DASS API to schedule new downlink packets, conforming to the scheduling behaviour defined in the protocol being implemented.

For servers communicating with emulated devices, the same underlying protocol logic was used, with the only differences between emulation servers and real-world servers being firstly, that different endpoints were called by devices directly and secondly, responses containing update segments were sent directly to devices, naturally with no scheduling being done with any LoRa infrastructure. Most importantly, error control and retransmission behaviour, as defined in the protocol being implemented, remained the exact same as on real-world devices.

All server implementation was done in Python, using the Flask web framework to expose endpoints that the DASS API or emulated devices could call to receive/schedule new data. One last detail important to note is that on the server, the number of receive windows being opened by the device was configurable. Being able to change this number allowed servers to be used with Class B at varying ping slot configurations as well as Class A devices, by setting the number of ping slots to 1.

## 4.3 Performance Evaluation

With the implementation of Class A device code, Class B emulated devices and application server logic for each protocol complete, a series of experiments were carried out, each aiming to provide answers to a different question. These experiments were as follows:

1. Real world Class A device vs Emulated Class A device
2. ACKED Piggybacked Selective Repeat - Class A vs Class B
3. True Selective Repeat vs ACKED PSR vs NACKED PSR

Due to time constraints imposed by the configuration challenges faced earlier in the project, and the unexpected need to write device emulators for Class B, the first two experiments only measured the total number of uplinks required in the transmission process. The reason this metric was chosen over effectiveness ratio and ineffective uplink metrics is that these two metrics can be assumed to be worse the higher the total uplinks metric climbs, and the average uplink size metric is largely irrelevant in experiments involving Class A.

Each experiment uses an average-min-max graph to illustrate results. This was chosen as it gives a good overview of the data being presented, allowing the reader to ascertain a good idea of best case, worst case and normal performance without having to review a large amount of data.

In all experiments which use Class B functionality, the number of receive windows used was 5. This number was chosen as it is close to the example shown in the LoRaWAN specification, and as there is no clear instruction as to how many receive windows can be used, it was decided to stay close to this example, so as not to show results that were potentially unachievable in practice with real devices.

### 4.3.1 Experiment 1 - Real Device vs Emulated Device

Experiment 1 aims to establish credibility for the emulated devices by comparing the performance of a real-world device with that of an emulation of the same device. This comparison uses the same protocol for each device - the Piggybacked Selective Repeat (PSR) implementation using positive acknowledgement that was described in Chapter 3.

For this experiment, 10 tests were carried out sending a 1kb update delta using a spreading-factor of 12. The results are discussed on the next page.

**Results**

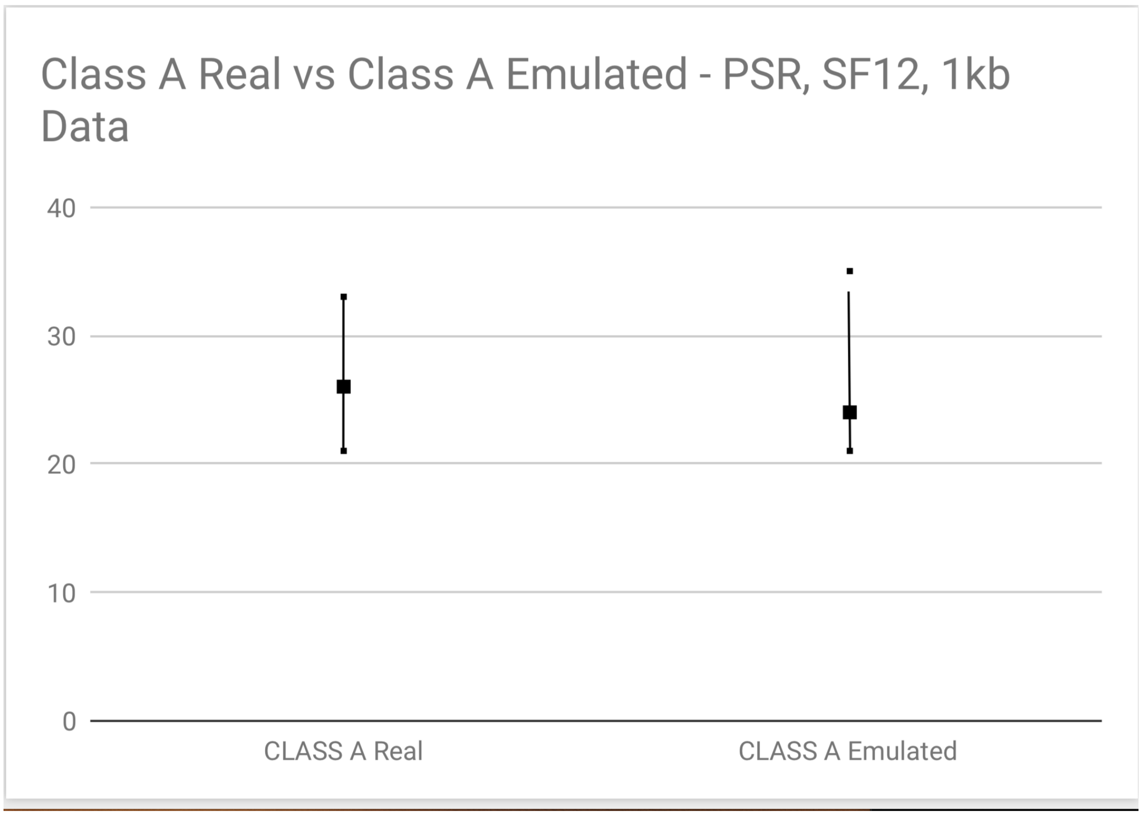


Figure 4.3: Experiment 1 results.

As can be seen from the results obtained in experiment 1, the real device and its emulated counterpart show similar performance, both averaging close to 25 uplinks required to deliver 1kb of data over 10 tests. The emulated device does see a higher variance in the number of packets it drops compared to the real device. One possible explanation for this is that the drop percentages obtained from [31] resulted from tests done in several different locations, meaning these drop percentages account in some way for LoRaWAN tower reception. The Class A real device used in this experiment was positioned in the same place for each test, and thus the reception to the nearest tower remained constant, which may have resulted in the smaller deviation from the mean observed on the real device.

Despite this small discrepancy, the Class A emulation showed itself to be similar enough to the real device to establish some credibility for the emulations used in the following experiments.

### 4.3.2 Experiment 2 - Class A vs Class B

Experiment 2 aims to show the advantages of using Class B in place of Class A for firmware update transmissions. Again making use of the Piggybacked Selective Repeat implementation using positive acknowledgement, 10 tests were carried out, each transmitting 1kb of data. The results, discussed on the next page, show a comparison of the total number of uplinks sent to receive this 1kb data, and will illustrate the reduced energy cost incurred when using Class B.

**Results**

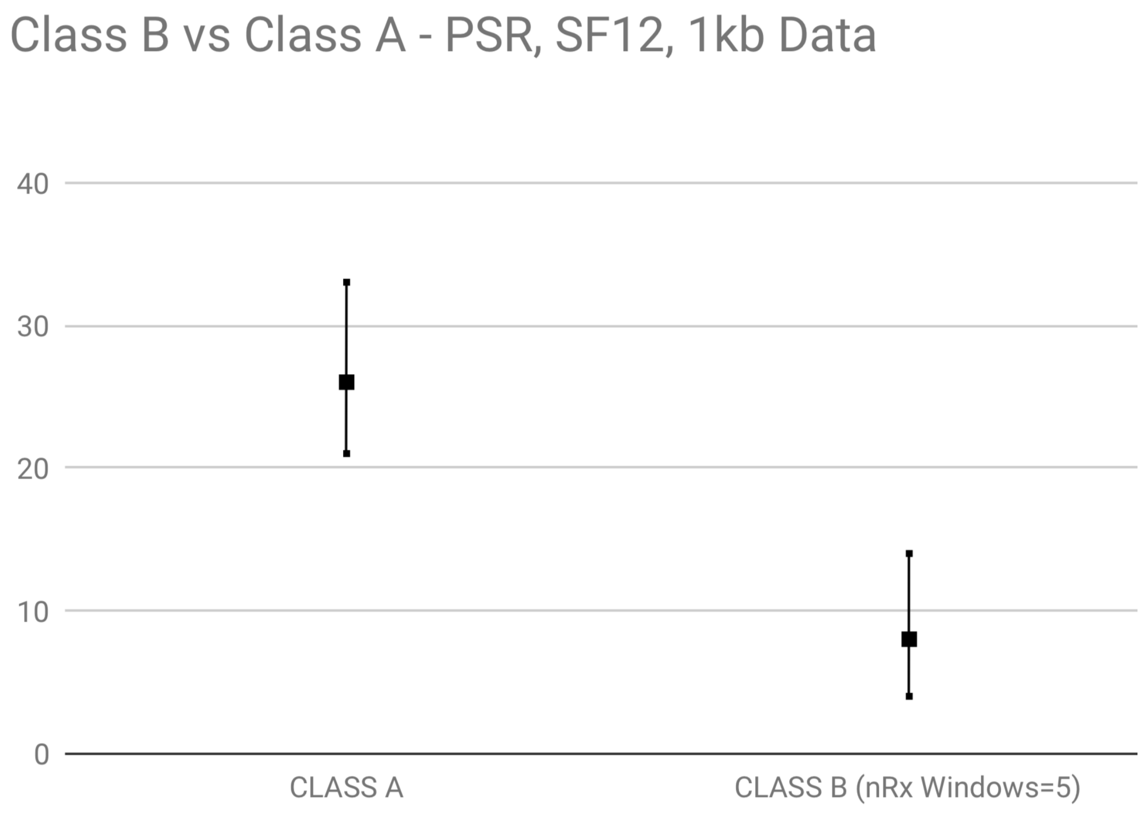


Figure 4.4: Experiment 2 results.

The results obtained from this experiment are quite clear, making the advantages of using Class B very obvious. Due to the fact that Class B can receive multiple segments per uplink, the number of uplinks sent by such a device are reduced, in theory, by a factor of $n$ where $n$ is the number of $R_x$ windows opened by the Class B device per beacon period (explained in Chapter 2).

This experiment has shown this theoretical reduction factor to be very close to fact in practice, with this particular test showing a 3.5$\times$ reduction in number of uplinks for a Class B device that opens 5 $R_x$ windows. It is expected that with more runs of this test, the observed reduction factor would get ever closer to the theoretical reduction factor.

### 4.3.3 Experiment 3 - Protocol Comparison

The final experiment, also the most extensive, aims to make clear the best protocol, of the three designed in Chapter 3, for use in transmitting firmware updates over LoRaWAN Class

B. For each protocol, 40 total tests were carried out - 10 tests transmitting a 1kb update delta at SF-7, 10 tests transmitting a 2kb update at SF-7 and the other 20 being the same 1kb and 2kb transmissions at SF-12. These delta sizes were chosen based on estimates for compressed incremental update packages, which are generally observed to be in the 1-2kb range for non-major changes, and give a good indication how the transmission changes as deltas to be sent increase in size. Tests were carried out on spreading factors 7 and 12 to provide some variance in drop likelihood, though if time had permitted, it would have been more complete to test the full range of spreading factors.

**Effectiveness Ratios**

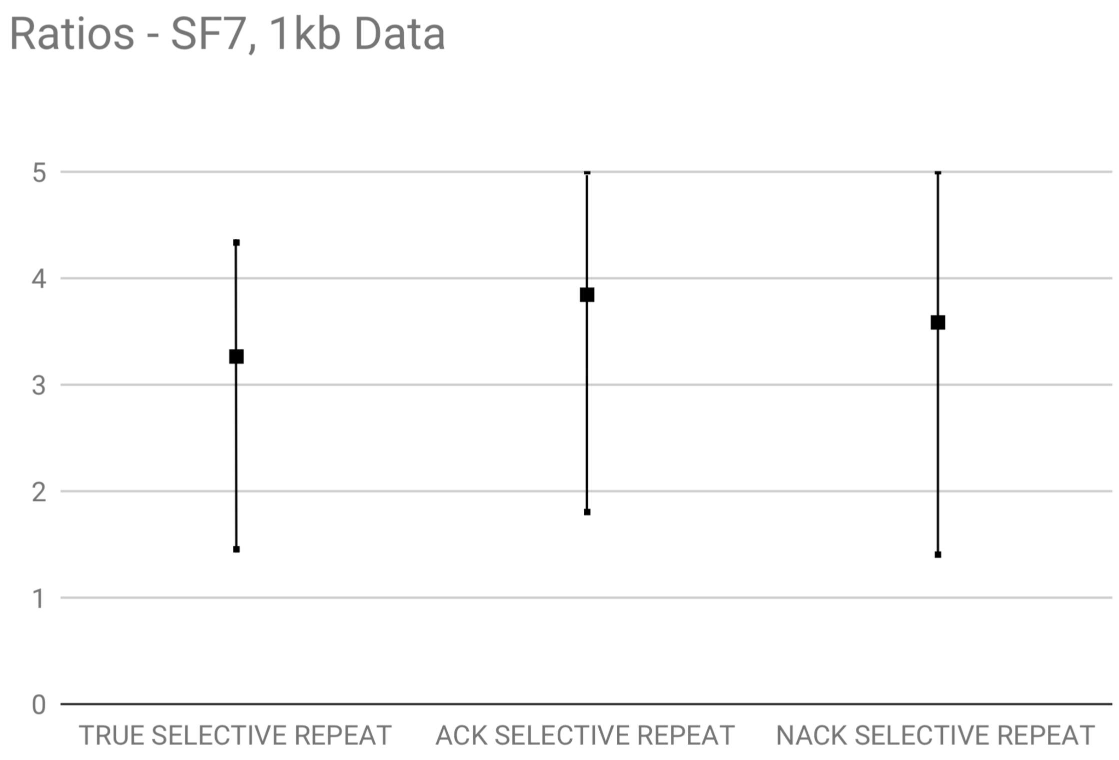


Figure 4.5: Results - Observed Effectiveness Ratios (SF7, 1kb Delta).

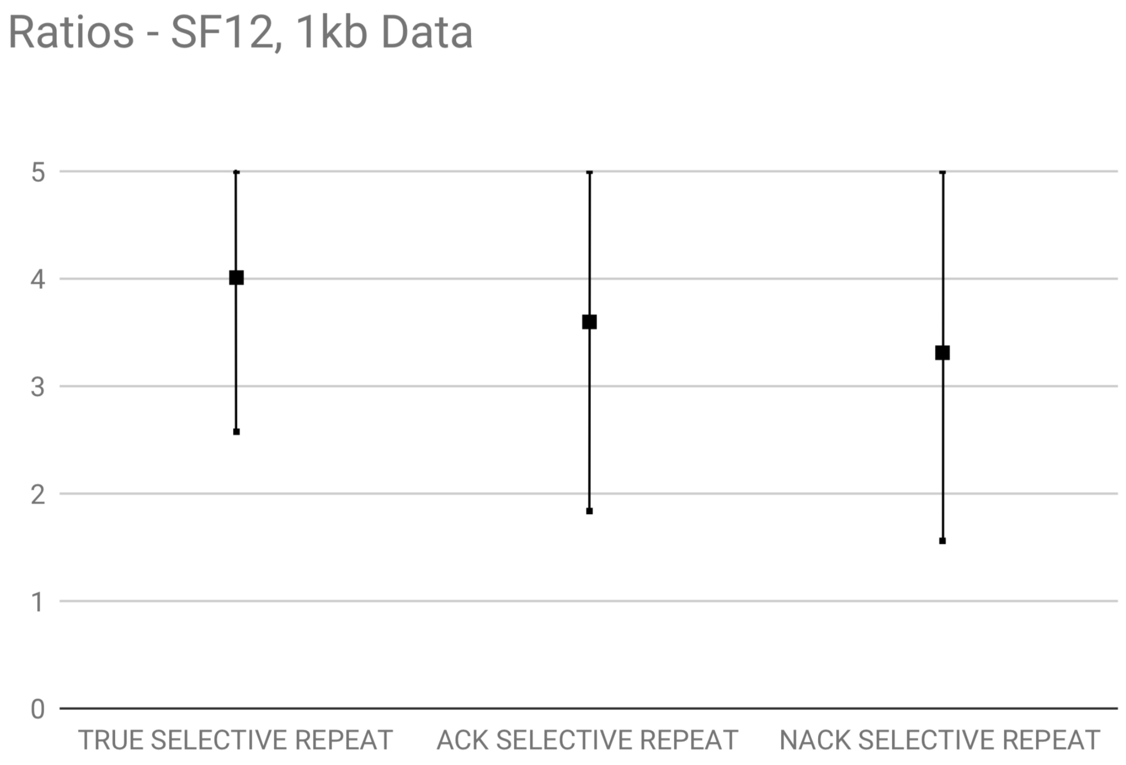


Figure 4.6: Results - Observed Effectiveness Ratios (SF12, 1kb Delta).

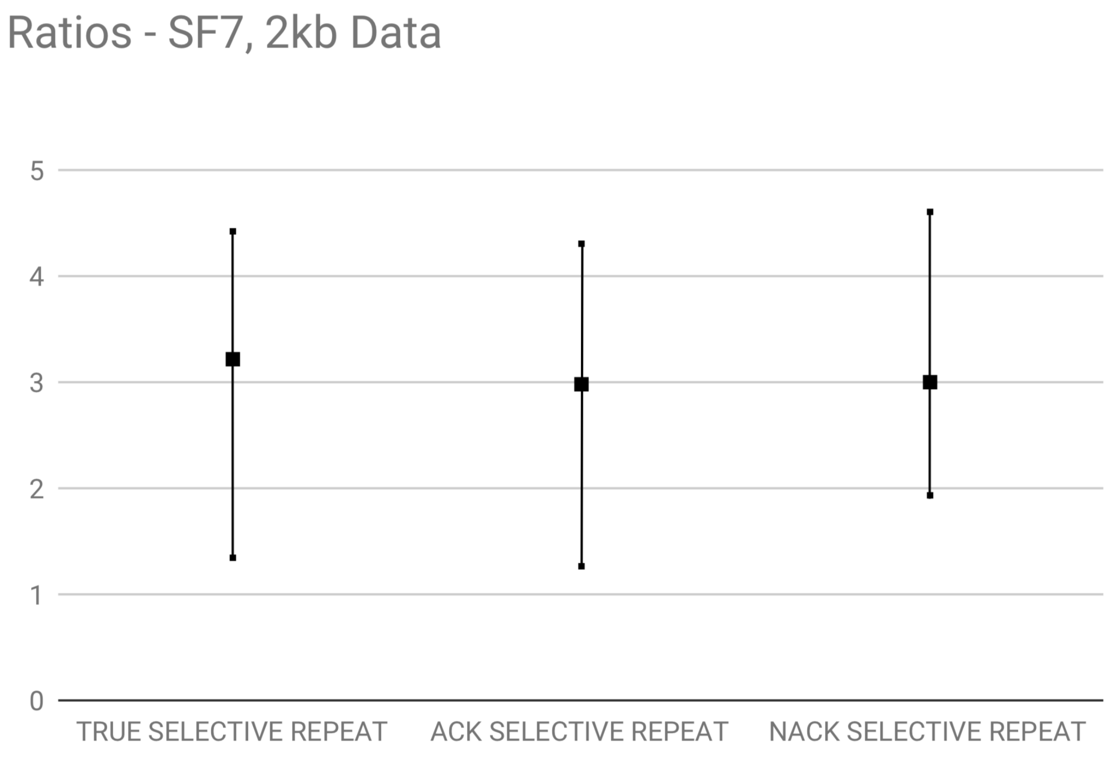


Figure 4.7: Results - Observed Effectiveness Ratios (SF7, 2kb Delta).

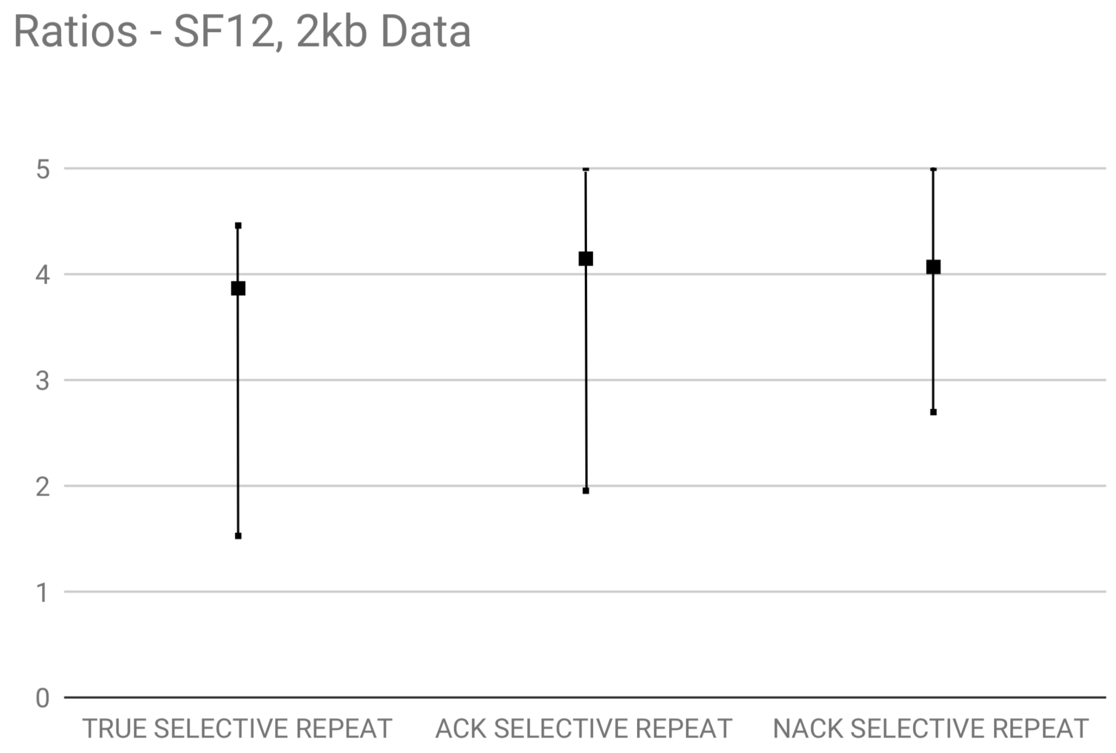


Figure 4.8: Results - Observed Effectiveness Ratios (SF12, 2kb Delta).

As previously explained, the effectiveness ratio aims to quantify how effective a protocol is by averaging the number of effective downlinks received per uplink.

For the 1kb data size tests, several observations can be made. Firstly, at spreading factor 7, the number of lost packets is slightly higher than on spreading factor 12. This is to be expected based on the values used to parameterise the emulations. In terms of the best performance, the positively ACKed version of PSR performs better with the higher loss at SF7, whereas the True Selective Repeat implementation performs better with less packet loss. This is somewhat consistent with what would be expected, in that TSR performs better with less packet loss, as losing more packets with TSR will lead to the window stopping, in turn leading to a higher number of duplicate packets being sent, reducing effectiveness ratio.

This is not consistent, however, with the results expected of the other two protocols with higher loss rate. It would have been expected instead that these two protocols perform on a similar level to, if not better than the TSR implementation. As only 10 tests were carried out at each spreading factor, it is possible that this result was just due to the pseudo-random nature of the packet loss in emulated devices, and perhaps testing another

10 times would yield different results.

For the 2kb data size tests, the results are much closer, though the true selective repeat protocol unexpectedly comes off slightly ahead in the SF7 test. Interestingly, a perfect ratio, i.e. $(r = nR_x)$, is never observed in the true selective repeat protocol's tests, whereas on the more reliable spreading factor of 12, this result is observed using both other protocols. Again as expected, the performance of the ACKed and NACKed PSR come quite close to each other, with slightly lower minimums being observed with the NACKed version.

All in all, the effectiveness ratio results do not produce a clear winner when it comes to the protocols designed for this work, with all three performing quite similarly on all accounts. It is expected, however that the average uplink sizes that follow this set of results will present clearer findings.

### Average Uplink Sizes

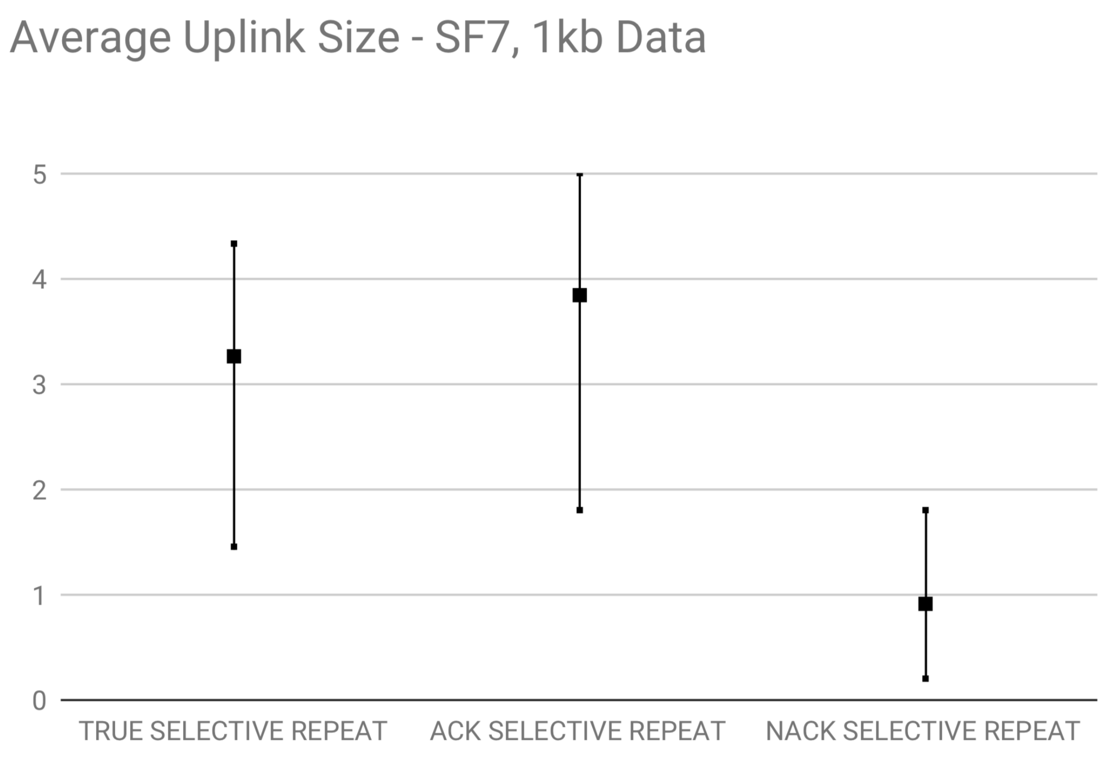


Figure 4.9: Results - Observed Average Uplink Size (SF7, 1kb Delta).

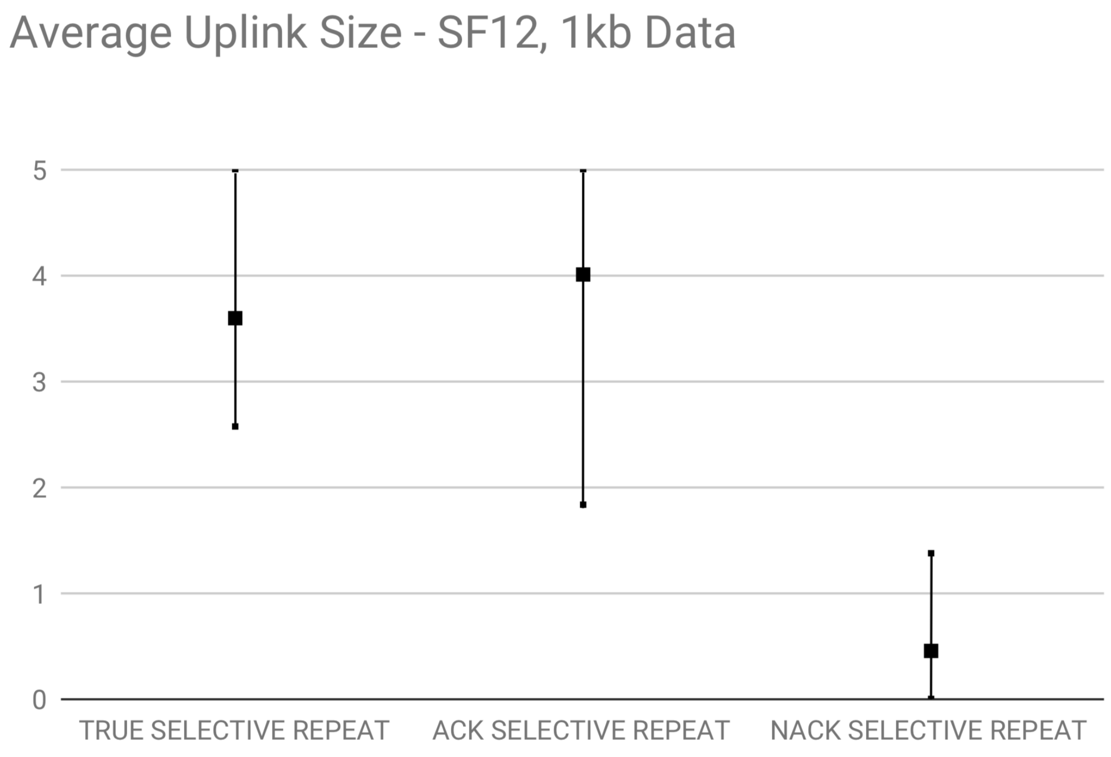


Figure 4.10: Results - Observed Average Uplink Size (SF12, 1kb Delta).

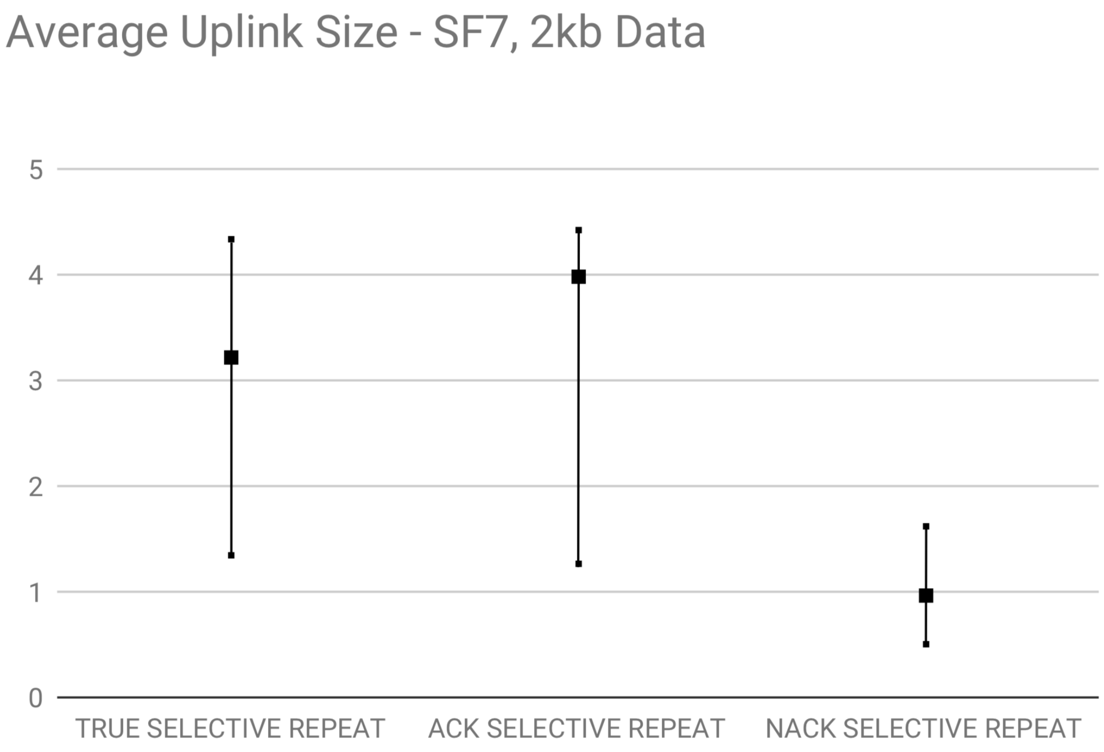


Figure 4.11: Results - Observed Average Uplink Size (SF7, 2kb Delta).

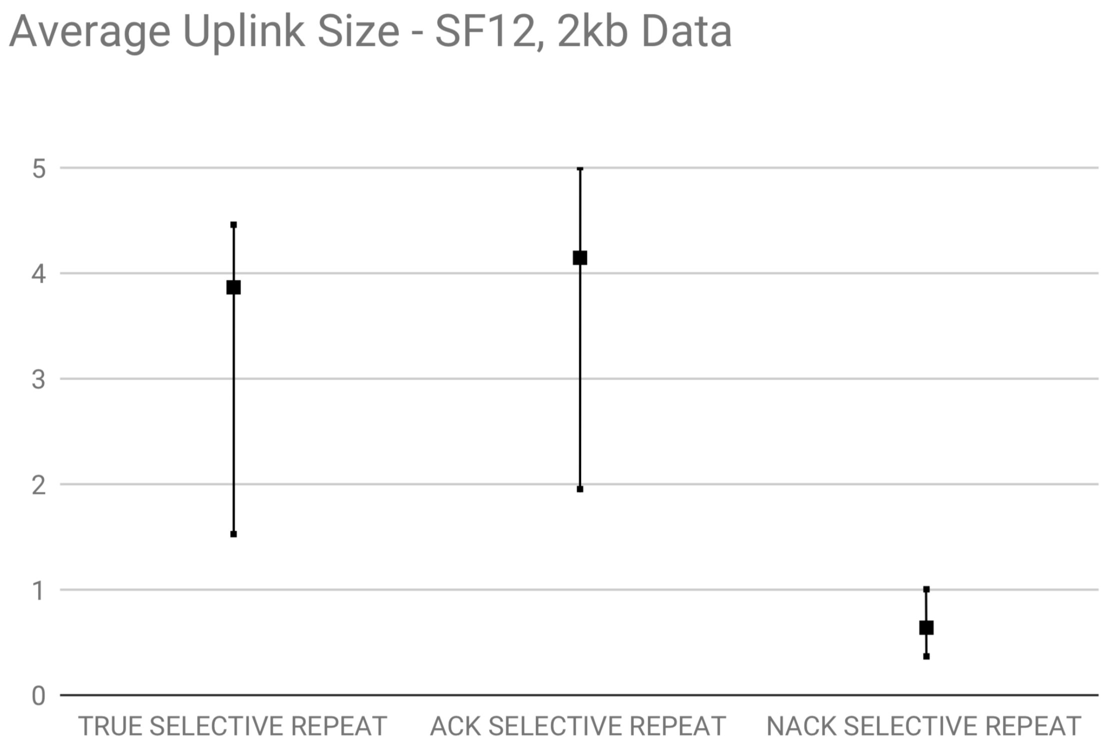


Figure 4.12: Results - Observed Average Uplink Size (SF12, 2kb Delta).

As expected, this metric has revealed a clear divide between the NACKed PSR and the other two protocols. Across all tests, the average uplink size is less than 1, compared to the other two protocols where we see the average uplink size lie in the 3.5 to 4 region. This should mean, with a real implementation on a Class B device, that a significant decrease in transmission time, and thus energy, will be observed.

It is clear from these results that NACKed PSR is by far the better protocol in terms of energy cost where average uplink size is considered.

**Ineffective Uplinks**

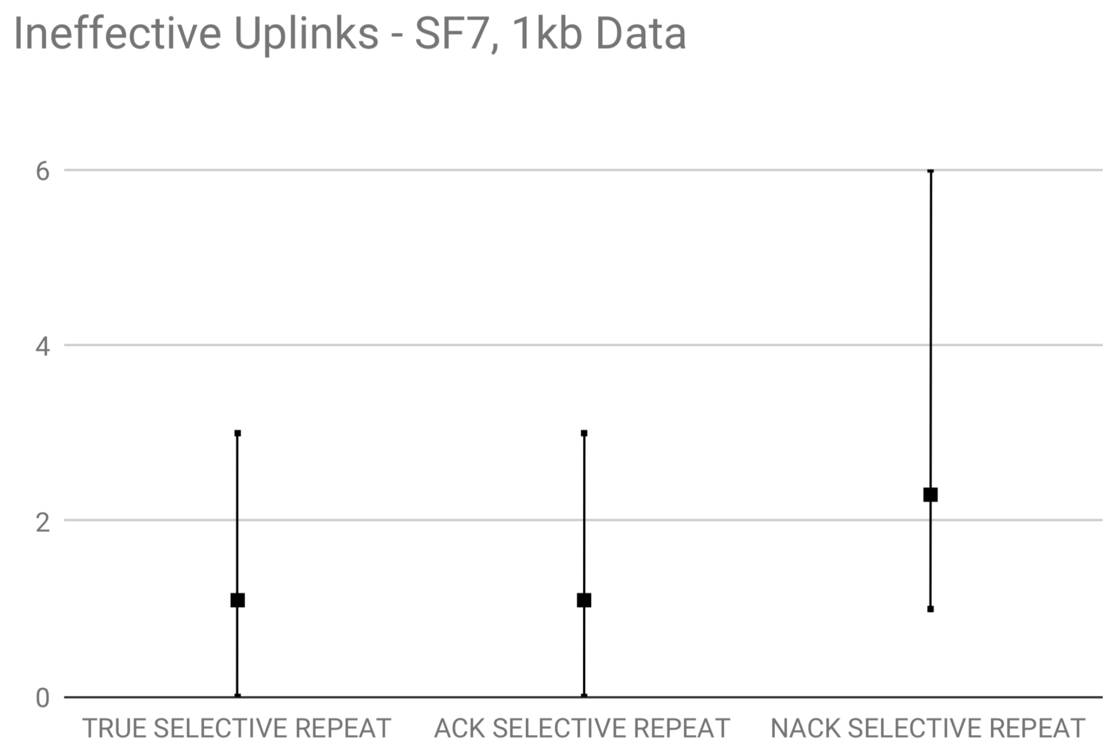


Figure 4.13: Results - Observed Ineffective Uplinks (SF7, 1kb Delta).

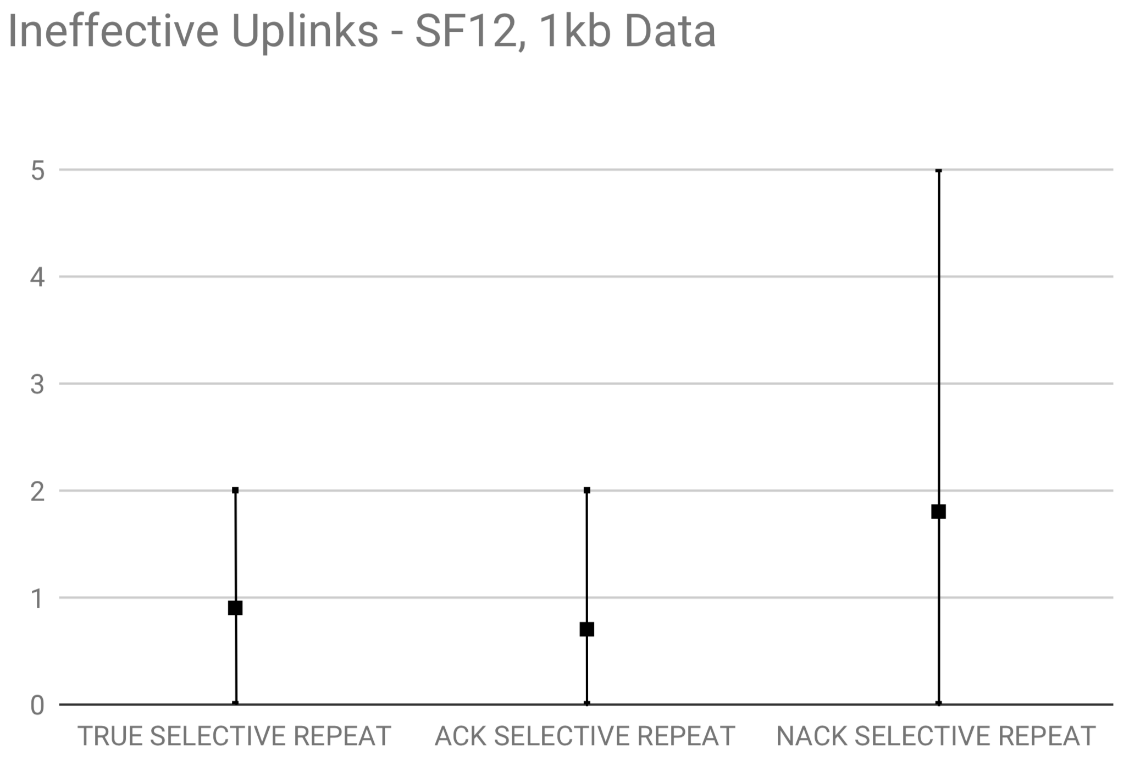


Figure 4.14: Results - Observed Ineffective Uplinks (SF12, 1kb Delta).

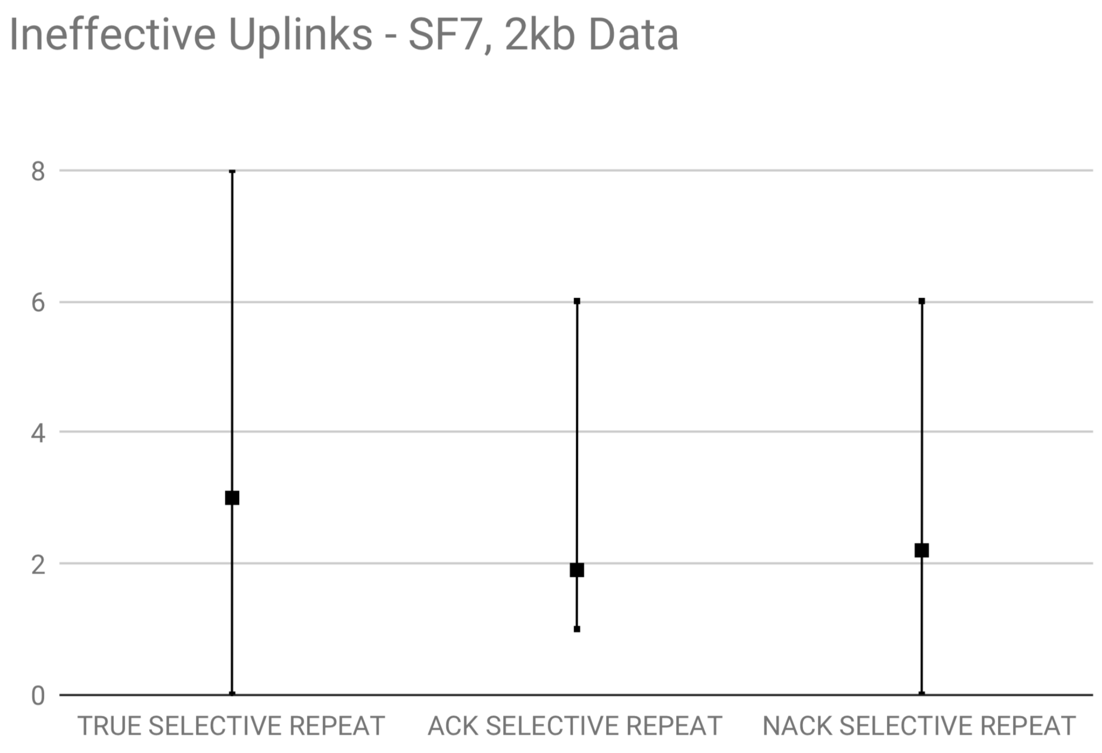


Figure 4.15: Results - Observed Ineffective Uplinks (SF7, 2kb Delta).

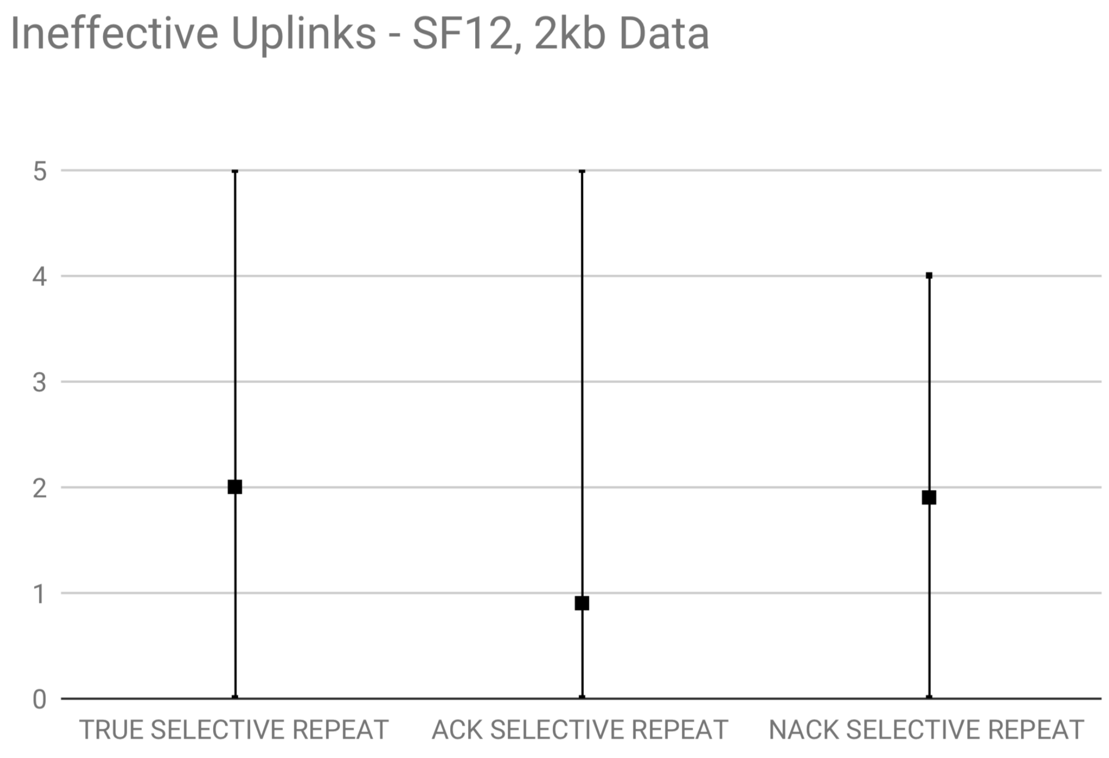


Figure 4.16: Results - Observed Ineffective Uplinks (SF12, 2kb Delta).

The results seen here echo in a lot of ways what is seen in the ratio results, which is to be expected as more ineffective uplinks will lead to a worse ratio.

In the 1kb data tests, NACKed PSR performs the worst of the group, but not by much which would again suggest that these observations could have been different in a different set of tests, and would more likely be closer to what is expected the more tests that are run. Deviation from the mean in this set of observations was also quite high, which suggests that the emulations may have been too variant in their rates of packet loss, and again implementing these protocols for testing with real devices would likely yield more predictable results.

In the case of the 2kb tests, the results are more expected than in the 1kb tests, backing up the idea mentioned above that the performance of each protocol will begin to conform to what is expected with more tests, as in this case, twice the number of packets are to be sent, which produces the same effect as doubling the number of tests run.

Again, based on this data, no clear winner is initially obvious, though ACKed PSR performs marginally better than the other two protocols. This is not definitive, however, and the results that could be provided by running these experiments on real world devices will likely produce much more conclusive results.

## 4.4 Discussion

The results presented in this chapter varied in how effective they were in presenting the case for LoRaWAN Class B in this context, but they did allow some conclusions to be drawn.

To begin, the first experiment showed a certain level of similarity between real and emulated devices. This proved to be helpful in that it gave emulations some credibility, however this is not necessarily reflected by the performance of the Class B emulators. There is, however, no precise way to prove the emulation to be less effective than real devices where Class B is concerned until there is real infrastructure available.

With this said however, it must be kept in mind that the 'inconclusive' tests in the 3rd experiment may just be inconclusive due to the fact that the protocols will perform similarly where these metrics are concerned, though TSR would still be expected to perform worse than the other two protocols in these two metrics, which, in the tests carried out in this work, it did not. Again, with real device testing or even perhaps more tests across more spreading factors using emulation, results would likely converge to what is expected.

The test of Class B vs Class A for update transmission was perhaps the most conclusive, and this is quite important. Despite being difficult to find a winner out of the three protocols designed for this work, this experiment showed quite clearly that there will be a significant benefit to using Class B in this context. The use of Class B is likely to reduce energy consumption by a factor proportional to the number of $R_x$ windows opened by a Class B device, and this is very beneficial to the end device.

As alluded to in all of the above observations, there is no clear best protocol of the three designed for this work. The only clear definition between the three protocols is the average uplink size, a metric by which NACKed Piggybacked Selective Repeat comes out on top by a wide margin. This is quite a significant result, as lowered transmission time on the uplink is likely to lead to considerable energy cost reduction. For this reason, the most promising of the three protocols in terms of reducing energy usage is NACKed Piggybacked Selective

Repeat.

# 5 Conclusion

Building on a previous dissertation, which evaluated the use of LoRaWAN Class A technology in firmware update transmission, this work set out to answer two main questions:

1. Is LoRaWAN Class B a better for firmware update transmission than Class A?
2. What is the best protocol for update delta transmission over Class B?

Using a combination of real world devices and infrastructure as well as device emulation for the unavailable Class B infrastructure, an attempt was made to answer these questions.

To evaluate the update process on Class B, three firmware transmission protocols were designed: True Selective Repeat ARQ, ACKed Piggybacked Selective Repeat ARQ and NACKed Piggybacked Selective Repeat ARQ. The resulting evaluations of these protocols were not wholly conclusive as to which performed the best, though the last protocol mentioned, NACKed Piggybacked Selective Repeat ARQ, is expected to perform the best out of the three if further testing is carried out, especially if this testing is done on real devices rather than emulations.

Each protocol was evaluated with two different update sizes, based on real world update delta sizes, and each of these update delta sizes were in turn transmitted on two different spreading factors offered by LoRa. Spreading factors do not have much impact, it would seem, where update time is not considered, though higher drop rates on different spreading factors can impact results and it would be worth testing a larger variety in the future.

The question of whether Class B is better for update transmission than Class B, however, was answered fairly definitively, and it is expected that tests on real devices will reflect this. A significant decrease in the number of uplinks needed to transfer data will directly cause less energy consumption, meaning the updating process will be much more effective if Class B is used.

To conclude, the use of LoRaWAN Class B over Class A does present LoRaWAN as a much more likely candidate for the transmission of firmware updates over long distances. Class A is meant as a largely unidirectional means of communication, with downlink data expected by the device to be small and infrequent. Class B, on the other hand, is built for bidirectional transmissions, and while it does consume more energy in the same operational time as a Class A device, the benefits in terms of uplink reduction and potentially faster transmission times are likely to result in a net overall reduction in energy usage by devices which use LoRaWAN Class B to receive updates.

## 5.1 Future Work

This section will outline some potential work to be undertaken in the future, based on the results of the research done for this work.

### 5.1.1 Implementation on Real Class B Devices

It has been alluded to several times in the course of this work that more conclusive results will likely be found if real Class B devices are used rather than best-guess emulations. To truly assess the performance of the protocols outlined in this work, and the feasibility of using Class B for update transmission, tests must be carried out in real life situations, with experiments testing factors such as network coverage, mobile deployments and their effects on update transmission, and of course time taken for update transmission. Using the work done in this dissertation, implemented on real Class B devices, the findings could be verified and an optimal protocol could be better selected.

### 5.1.2 Multicast Functionality

Multicast functionality is offered by LoRa in Class B operation. This means that dissemination to multiple nodes could potentially be significantly easier through the use of Class

B over Class A. This is again, however, something that requires real-world infrastructure to test and thus could not be completed in this work, but would be a worthwhile topic to investigate as the idea of using LoRaWAN as a means by which to transmit firmware updates moves towards real world application.

### 5.1.3 Control Protocols & Bootstrapping

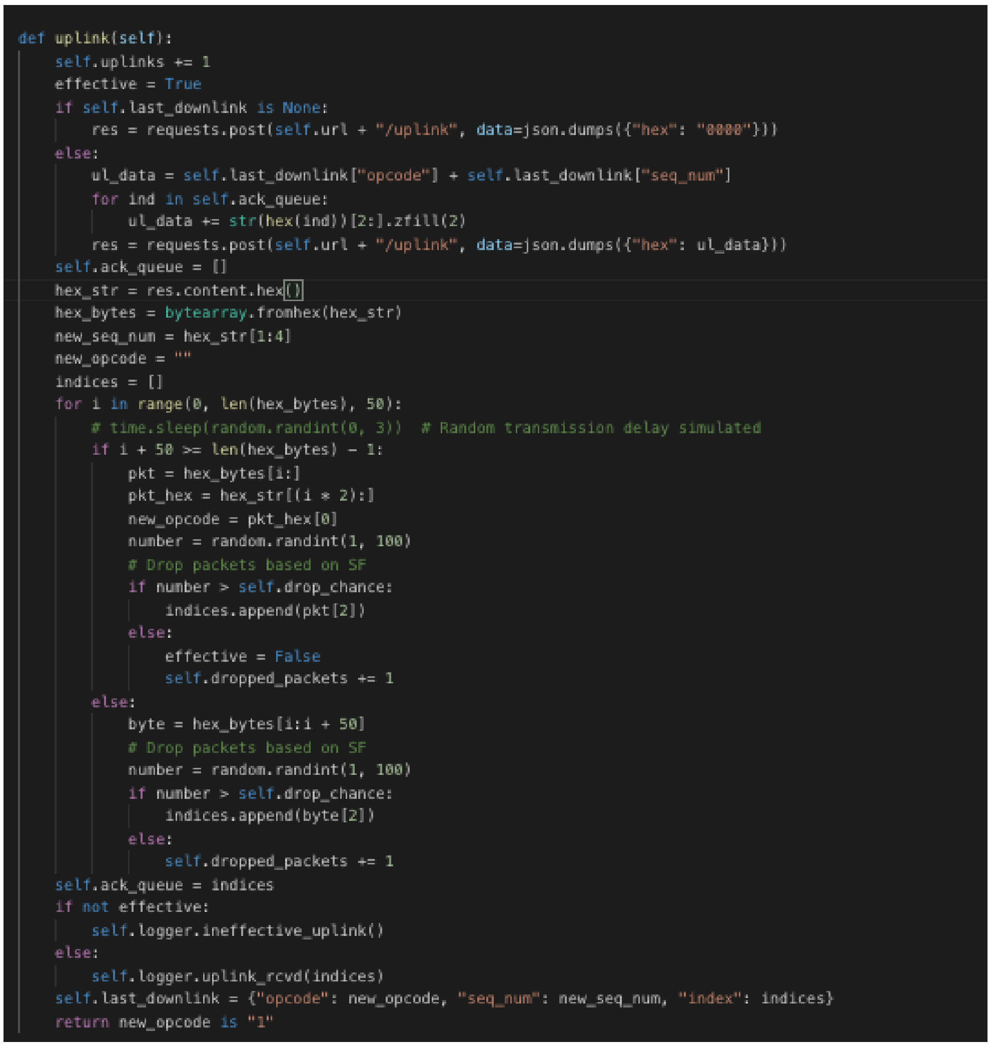
Again considering the progression of this research towards being applied in real world systems, a control protocol to determine things like what devices are updating, how much data devices have received, handling devices going offline mid-update, class switching and many more must be developed. Known as a control protocol, this protocol oversees the transmission and dissemination to devices in a system and adds another layer of robustness to the update process.

Furthermore, upon arrival on the device, updates must be arranged and put into the end-devices memory at the correct locations in order to run when the device restarts, followed by the device restarting itself to run this newly organised code. This process, known as bootstrapping, must also be investigated and could potentially be developed alongside a control protocol in future research.

With these future considerations in mind, there is still a lot of work to be done, but groundwork has definitely been laid to allow a fully functional update mechanism for LoRaWAN devices, and with the work of future researchers, a firmware updating procedure could be developed for use in real world applications.

# A Appendix

## A.1 Sample Class B Device Emulation Code



```python
def uplink(self):
    self.uplinks += 1
    effective = True
    if self.last_downlink is None:
        res = requests.post(self.url + "/uplink", data=json.dumps({"hex": "0000"}))
    else:
        ul_data = self.last_downlink["opcode"] + self.last_downlink["seq_num"]
        for ind in self.ack_queue:
            ul_data += str(hex(ind))[2:].zfill(2)
        res = requests.post(self.url + "/uplink", data=json.dumps({"hex": ul_data}))
    self.ack_queue = []
    hex_str = res.content.hex()
    hex_bytes = bytearray.fromhex(hex_str)
    new_seq_num = hex_str[1:4]
    new_opcode = ""
    indices = []
    for i in range(0, len(hex_bytes), 50):
        # time.sleep(random.randint(0, 3))  # Random transmission delay simulated
        if i + 50 >= len(hex_bytes) - 1:
            pkt = hex_bytes[i:]
            pkt_hex = hex_str[(i * 2):]
            new_opcode = pkt_hex[0]
            number = random.randint(1, 100)
            # Drop packets based on SF
            if number > self.drop_chance:
                indices.append(pkt[2])
            else:
                effective = False
                self.dropped_packets += 1
        else:
            byte = hex_bytes[i:i + 50]
            # Drop packets based on SF
            number = random.randint(1, 100)
            if number > self.drop_chance:
                indices.append(byte[2])
            else:
                self.dropped_packets += 1
    self.ack_queue = indices
    if not effective:
        self.logger.ineffective_uplink()
    else:
        self.logger.uplink_rcvd(indices)
    self.last_downlink = {"opcode": new_opcode, "seq_num": new_seq_num, "index": indices}
    return new_opcode is "1"
```

Figure A.1: Sample code for receipt of an uplink on an emulated device

# Bibliography

[1] Abinaya, B., Gurupriya, S. and Pooja, M. [2017], Iot based smart and adaptive lighting in street lights, *in* '2017 2nd International Conference on Computing and Communications Technologies (ICCCT)', IEEE, pp. 195–198.

[2] Akyildiz, I. F., S. W. S. Y. and Cayirci, E. [2002], 'Wireless sensor networks: a survey.', *Computer Networks* **38**, 393 – 422.

[3] Al-Fuqaha, A., guizani, m., Mohammadi, M., Aledhari, M. and Ayyash, M. [2015], 'Internet of things: A survey on enabling technologies, protocols and applications', *IEEE Communications Surveys & Tutorials* **17**, Fourthquarter 2015.

[4] Angrisani, L., D'Arco, M., Dassi, C. and Liccardo, A. [2018], Lora signals classification through a cs-based method, *in* '2018 IEEE 4th International Forum on Research and Technology for Society and Industry (RTSI)', pp. 1–5.

[5] Augustin, A., Yi, J., Clausen, T. and Townsley, W. [2016], 'A study of lora: Long range & low power networks for the internet of things', *Sensors* **16**(9), 1466.

[6] Ayoub, W., M. M. N. F. S. A. E. . P. J. C. [2018], 'Towards ip over lpwans technologies: Lorawan, dash7, nb-iot.', *2018 Sixth International Conference on Digital Information, Networking, and Wireless Communications (DINWC), Digital Information, Networking, and Wireless Communications (DINWC), 2018 Sixth International Conference on* p. 43.

[7] Ayoub, W., Nouvel, F., Samhat, A. E., Prevotet, J.-C. and Mroue, M. [2018], 'Overview and measurement of mobility in dash7', pp. 532–536.

[8] Brown, S. and Sreenan, C. [2013*a*], 'An energy benchmark for software updates on wireless sensor nodes'.

[9] Brown, S. and Sreenan, C. [2013*b*], 'Software updating in wireless sensor networks: A survey and lacunae', *Journal of Sensor and Actuator Networks* **2**(4), 717–760.

[10] Buyukakkaslar, M. T., Erturk, M. A., Aydin, M. A. and Vollero, L. [2017], 'Lorawan as an e-health communication technology', **2**, 310–313.

[11] Choi, C.-S., Jeong, J.-D., Lee, I.-W. and Park, W.-K. [2018], 'Lora based renewable energy monitoring system with open iot platform', pp. 1–2.

[12] de Boer, P. S., van Deursen, A. J. and van Rompay, T. J. [2019], 'Accepting the internet-of-things in our homes: The role of user skills.', *Telematics and Informatics* **36**, 147 – 156.

[13] Doukas, C. and Maglogiannis, I. [2012], Bringing iot and cloud computing towards pervasive healthcare, *in* '2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing', IEEE, pp. 922–926.

[14] Drath, R. and Horch, A. [2014], 'Industrie 4.0: Hit or hype?[industry forum]', *IEEE industrial electronics magazine* **8**(2), 56–58.

[15] Gaddam, S. C. and Rai, M. K. [2018], 'A comparative study on various lpwan and cellular communication technologies for iot based smart applications', pp. 1–8.

[16] Gazis, V., Goertz, M., Huber, M., Leonardi, A., Mathioudakis, K., Wiesmaier, A. and Zeiger, F. [2015], Short paper: Iot: Challenges, projects, architectures, *in* '2015 18th International Conference on Intelligence in Next Generation Networks', IEEE, pp. 145–147.

[17] Grabia, M., Markowski, T., Mruczkiewicz, J. and Plec, K. [2017], 'Design of a dash7 low power wireless sensor network for industry 4.0 applications', pp. 254–259.

[18] Hatzivasilis, George, A. I. A. G. A. D. B. A. K. V. F. K. S. G. [2018], 'The interoperability of things: Interoperable solutions as an enabler for iot and web 3.0.', *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2018 IEEE 23rd International Workshop on* p. 1.

[19] Heady, E. O. and Sonka, S. T. [1974], ‘Farm size, rural community income, and consumer welfare’, *American Journal of Agricultural Economics* **56**(3), 534–542.

[20] Hickman, M. D. and Wilson, N. H. [1995], ‘Passenger travel time and path choice implications of real-time transit information’, *Transportation Research Part C: Emerging Technologies* **3**(4), 211–226.

[21] Kathjoo, Mujtaba Yousuf, K. F. A. T. B. M. [2018], ‘A comparative study of wsn and iot.’, *2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC), Advances in Electronics, Computers and Communications (ICAECC), 2018 Second International Conference on* p. 1.

[22] Lipper, L., Thornton, P., Campbell, B. M., Baedeker, T., Braimoh, A., Bwalya, M., Caron, P., Cattaneo, A., Garrity, D., Henry, K. et al. [2014], ‘Climate-smart agriculture for food security’, *Nature climate change* **4**(12), 1068.

[23] LoRa Alliance [2015], A technical overview of LoRa and LoRaWAN, Documentation, LoRa Alliance.

[24] LoRa Alliance Technical Committee [2017], LoRaWAN™ 1.1 Specification, Documentation, LoRa Alliance.

[25] Lu, Y. [2017], ‘Industry 4.0: A survey on technologies, applications and open research issues’, *Journal of Industrial Information Integration* **6**, 1–10.

[26] Ma, Y.-W. and Chen, J.-L. [2018], ‘Toward intelligent agriculture service platform with lora-based wireless sensor network.’, *2018 IEEE International Conference on Applied System Invention (ICASI), Applied System Invention (ICASI), 2018 IEEE International Conference on* p. 204.

[27] Mekki, K., Bajic, E., Chaxel, F. and Meyer, F. [2018*a*], ‘Overview of cellular lpwan technologies for iot deployment: Sigfox, lorawan, and nb-iot.’, *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Pervasive Computing and Communications Workshops (PerCom Workshops), 2018 IEEE International Conference on* p. 197.

[28] Mekki, K., Bajic, E., Chaxel, F. and Meyer, F. [2018*b*], ‘Overview of cellular lpwan technologies for iot deployment: Sigfox, lorawan, and nb-iot’, pp. 197–202.

[29] Narayanan, S., Tsolkas, D., Passas, N. and Merakos, L. [2018], 'Nb-iot: A candidate technology for massive iot in the 5g era', pp. 1–6.

[30] Noreen, U., Bounceur, A. and Clavier, L. [2017], A study of lora low power and wide area network technology, *in* '2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)', IEEE, pp. 1–6.

[31] O'Sullivan, K. [2018], An Evaluation of LoRa Low-Power Wide-Area Technology for Firmware Update Transmission, Master's thesis, Trinity College Dublin, Dublin.

[32] Rao, S., Chendanda, D., Deshpande, C. and Lakkundi, V. [2015], Implementing lwm2m in constrained iot devices, *in* '2015 IEEE Conference on Wireless Sensors (ICWiSe)', IEEE, pp. 52–57.

[33] Raza, U., Kulkarni, P. and Sooriyabandara, M. [2017], 'Low power wide area networks: An overview', *IEEE Communications Surveys & Tutorials* **19**(2), 855–873.

[34] Samie, F., Tsoutsouras, V., Bauer, L., Xydis, S., Soudris, D. and Henkel, J. [2016], Computation offloading and resource allocation for low-power iot edge devices, *in* '2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)', IEEE, pp. 7–12.

[35] Sanchez, L., Muñoz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., Ramdhany, R., Gluhak, A., Krco, S., Theodoridis, E. et al. [2014], 'Smartsantander: Iot experimentation over a smart city testbed', *Computer Networks* **61**, 217–238.

[36] Sarkar, C., Nambi, S. A. U., Prasad, R. V. and Rahim, A. [2014], A scalable distributed architecture towards unifying iot applications, *in* '2014 IEEE World Forum on Internet of Things (WF-IoT)', IEEE, pp. 508–513.

[37] Satyanarayanan, M. [2017], 'The emergence of edge computing', *Computer* **50**(1), 30–39.

[38] Sisinni, E., Carvalho, D. F., Ferrari, P., Flammini, A., Silva, D. R. C. and Da Silva, I. M. [2018], Enhanced flexible lorawan node for industrial iot, *in* '2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)', IEEE, pp. 1–4.

[39] Stolikj, M., Cuijpers, P. J. and Lukkien, J. J. [2013], Efficient reprogramming of wireless sensor networks using incremental updates, *in* '2013 IEEE International Conference

on Pervasive Computing and Communications Workshops (PERCOM Workshops)', IEEE, pp. 584–589.

[40] TongKe, F. [2013], 'Smart agriculture based on cloud computing and iot', *Journal of Convergence Information Technology* **8**(2).

[41] Towsley, D. [1979], 'The stutter go back-n arq protocol', *IEEE transactions on Communications* **27**(6), 869–875.

[42] Weyn, M., Ergeerts, G., Berkvens, R., Wojciechowski, B. and Tabakov, Y. [2015], 'Dash7 alliance protocol 1.0: Low-power, mid-range sensor and actuator communication', pp. 54–59.

[43] Zhang, J. and Varadharajan, V. [2008], 'A new security scheme for wireless sensor networks', *IEEE Globecom 2008* pp. 128–132.

[44] Zhao, K. and Ge, L. [2013], A survey on the internet of things security, *in* '2013 Ninth International Conference on Computational Intelligence and Security', IEEE, pp. 663–667.