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

School of Computer Science and Statistics

An Evaluation of LoRa Low-Power Wide-Area Technology for Firmware Update Transmission

Kevin O'Sullivan

Supervised by Dr. Jonathan Dukes

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

M.A.I. Engineering

Submitted May, 2018

Declaration

I, Kevin O'Sullivan, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Signed: ____________________ Date: ____________________

Summary

This work aims to evaluate LoRa low-power wide-area technology for wireless firmware transmission. LoRa, a chirp spread spectrum derivative developed by Semtech, is one of the emerging low-power wide-area technologies.

Low-power wide-area technologies trade data rate for long range transmission at low power consumption. Wireless communication is typically the most energy-intensive operation of a wireless sensor node and as a result low-power wide-area (LPWA) technologies look promising for many applications.

Wireless firmware updating has always posed problems in resource constrained wireless sensor networks. However, considering the low data rate constraints of low-power wide area technologies and with firmware update images typically in the order of tens or hundreds of kilobytes, this poses an even greater challenge. Taking the limitations of LoRa into consideration, this work designs two wireless firmware transmission protocols for LoRa Class A devices. An incremental firmware update approach is adopted, significantly reducing the size of updates, with firmware update deltas of less than 7000 bytes considered in this work.

The two designed protocols are variants of the Stop and Wait ARQ and Selective Repeat ARQ protocols. The Stop and Wait ARQ protocol was shown to have many inherent inefficiencies due to the unconventional bi-directional communication nature of LoRa Class A devices. The Selective Repeat ARQ variant, incorporating piggybacked acknowledgements, showed promising potential with a theoretical efficiency approaching 100% as the number of required packets increases.

Due to the inherent inefficiencies of the Stop and Wait ARQ protocol it was only tested for illustrative purposes. The Piggybacked Selective Repeat ARQ protocol was extensively tested. The commercially available LoRa RN2483 module was used for the experiments. Experiments consisted of transmitting a arbitrary firmware update delta of 5000 bytes at each of the six available spreading factors (or data rates). In total each spreading factor was tested eight times over the course of one week.

The experiments showed that LoRa is capable of 7000 byte firmware transmission times of less than one hour at the lower spreading factors 7, 8 and 9. At higher spreading factors this time significantly increases to between 3 and 9 hours at spreading factors 10 and 12 respectively. A discovered bug in the RN2483 module contributed to longer than expected transmission times, particularly at the higher spreading factors.

The optimum efficiency of the Piggybacked Selective Repeat ARQ protocol for a 5000 byte update is 98%, however, efficiencies of greater than 70% were rarely achieved. Again, the bug significantly contributed to inefficiency in the firmware transmission process and upon resolution, efficiencies closer to that of the optimum efficiency are expected.

Lastly, the distance between gateway and node was found to have a significant effect on packet reception. For reliable communications (and subsequently reliable firmware transmission) at lower spreading factors, gateways should be positioned within 2 km of the node with good line of sight. At higher spreading factors this distance can be increased to 5 km. Higher levels of noise floor at busy city locations was also found to contribute to poorer packet reception at lower spreading factors.

Abstract

Wireless communications are typically the most energy-intensive operation of a wireless sensor node. Newly emerging low-power wide-area (LPWA) technologies trade data rate for long range transmissions at low power. Wireless firmware updating has always posed problems in resource constrained wireless sensor networks. Considering the data rate constraints of LPWA technologies and with firmware update images typically in the order of tens or hundreds of kilobytes, this poses an even greater challenge to wireless firmware updating.

This work evaluates LoRa, a LPWA technology, and its ability to transmit firmware images to Class A devices. An incremental firmware update approach is adopted, with firmware update image deltas of 7000 bytes and less considered in this evaluation. Two firmware transmission protocols are purposefully designed for LoRa bi-directional communication and evaluated. The evaluation considers all six available spreading factors, which trade data rate for range and reliability. The results showed that at the lower spreading factors (higher data rates), LoRa can comfortably transmit a 7000 byte update in less than one hour. The corresponding time rises to 9 hours at the lowest data rate, however, a bug in the LoRa module is expected to contribute greatly to this time, and upon its resolution a time closer to 5 hours is expected. This work also found that in urban environments gateways should be located within 2 km of the node with good line of sight for adequate reliability at lower spreading factors. This distance can be increased to approximately 5 km for higher spreading factors.

Acknowledgements

I would like to thank Dr. Jonathan Dukes for his guidance and input throughout my dissertation.

I would also like to thank the team at Pervasive Nation for not only providing the infrastructure for such a project but also their continuous support in debugging any issues. A special acknowledgement to Brian Murphy whose willingness and readiness to help debug any issues was greatly appreciated.

Finally, I would like to thank my parents for their support and encouragement throughout all my academic endeavours to date.

Contents

List of Figures

List of Tables

Nomenclature

M	Maximum MAC Payload	$Bytes$
N	Maximum Frame Payload	$Bytes$
C	Channel Capacitity	$Bits/s$
CR	Code Rate	
B	Bandwidth	Hz
BW	Modulation Bandwidth	Hz
G_p	Processing Gain	dB
N	Average Noise of Interference Power	$Watts$
R_b	Bit Rate	$Bits/s$
R_c	Chip Rate	$Chips/s$
R_sc	Symbol Rate	$symbols/s$
SF	Spreading Factor	
S	Average Received Signal Power	$Watts$
ToA	Time on Air	s
T_{off}	Time off Air	s
T_s	Symbol Period	s
ABP	Activation By Personalisation	
ACK	Acknowledgement	
ADR	Adaptive Data Rate	
AES	Advanced Encryption Standard	
API	Application Programming Interface	
AppKey	Application Key	
AppSKey	Application Session Key	
ARQ	Automatic Repeat reQuest	
BLe	Bluetooth Low Energy	
BW	Bandwidth	

Nomenclature

CRC	Cyclic Redundancy Check
DASS	Data Access Sub-System
DCU	Dublin City University
DevAddr	Device Address
DER	Data Extraction Rate
DIT	Dublin Institute of Technology
DR	Data Rate
DSSS	Direct Sequence Spread Spectrum
EU	European Union
EUI	Extended Unique Identifier
FCnt	Frame Count
FCtrl	Frame Control
FHDR	Frame Header
FNwkSIntKey	Forwarding Network Session Integrity Key
FOpts	Frame Options
FPort	Frame Port
FRMPayload	Frame Payload
GSM	Global System for Mobile Communications
IoT	Internet of Things
ISM	IIndustrial, Scientific and Medical Radio Band
LPWA	Low Power Wide Area
LPWAN	Low Power Wide Area Network
LTE	Long-Term Evolution
M2M	Machine to Machine
MAC	Medium Access Control
MHDR	Medium Access Control Header

Nomenclature

MIC	Message Integrity Code
NwkKey	Network Key
NwkSEncKey	Network Session Encryption Key
OTAA	Over-The-Air Activation
PER	Packet Error Rate
PELR	Packet Error Loss Rate
PHY	Physical
PHDR	Physical Header
QoS	Quality of Service
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RFU	Reserved for Future Use
RNSS	Radio Network Sub-System
RSSI	Received Signal Strength Indicator
RX1	Receive Window 1
RX2	Receive Window 2
SF	Spreading Factor
SNR	Signal-to-Noise Ratio
SNwkSIntKey	Serving Network Session Integrity Key
TDOA	Time Difference of Arrival
TEC	Technology and Enterprise Campus
UCD	University College Dublin
UMTS	Universal Mobile Telecommunications System
WSN	Wireless Sensor Networks

1 Introduction

1.1 Background

The evolution of the world's telecommunication capabilities has coincided with the advancement of society throughout the ages. Telecommunication is today defined as *communication over a distance by cable, telegraph, telephone or broadcasting.* The word's origin however has a simpler meaning: *tele*, a Greek prefix meaning *distant*; and *communicare*, from the Latin *to share* [1]. From humble beginnings of smoke signals and drums, society has made great strides in the pursuit of *sharing at a distance.* Just over four hundred years ago, London was notified of the arrival of the Spanish Armada by a series of hilltop beacons from Plymouth [2], a mere 300 km. Today any person with a computer or mobile phone with internet access can broadcast a message to anyone in the world in a matter of seconds. The pioneers of telegraphy, telephony, radio, satellite and internet over the past one and a half centuries laid the foundation for what today is being called the *Fourth Industrial Revolution*, the fusion of the physical of times passed with the digital of now (Figure 1.1 [3]).

Internet of Things, Cloud Computing, Big Data, Artificial Intelligence, Machine Learning, Industry 4.0 - these are just some of the words that have become common vernacular over the past decade. With the invention of the steam engine, powered by an abundance of coal resources, came the first industrial revolution. Today, with the emerging field of machine learning, powered by an abundance of big data, society is on the brink of this next industrial revolution. This abundant source of data, just like

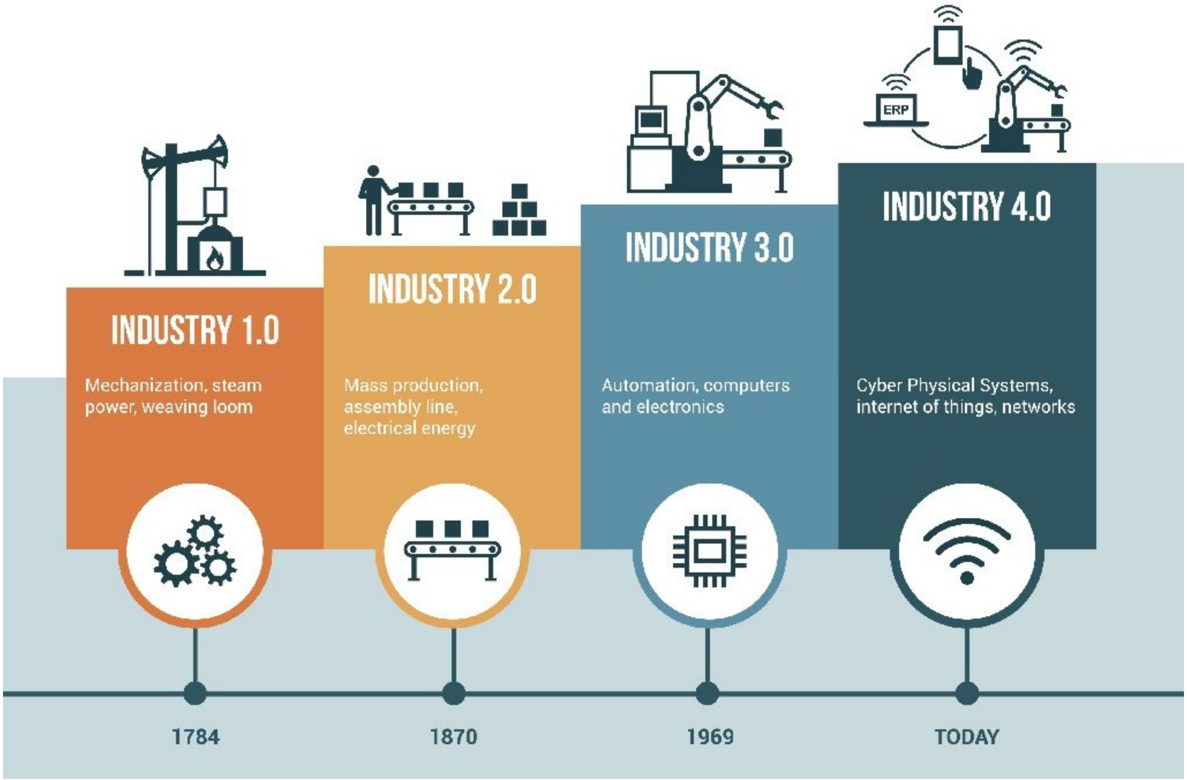

ERP
INDUSTRY 1.0
Mechanization, steam power, weaving loom
INDUSTRY 2.0
Mass production, assembly line, electrical energy
INDUSTRY 3.0
Automation, computers and electronics
INDUSTRY 4.0
Cyber Physical Systems, internet of things, networks
1784
1870
1969
TODAY

Figure 1.1: Industrial Revolutions Timeline [3]

coal, must be mined and collected. The advances in wireless technologies over the past 20 years has paved the way for wireless sensor networks and the Internet of Things. It is these wireless sensing devices that collect the valuable data which drives the engines of machine learning and artificial intelligence.

The term Internet of Things was originally coined in 1999 in the context of supply chain management [4]. Since then, and particularly in recent years, the term has come to encompass a wide range of industrial applications (transport, healthcare, smart factories and farms) and a wide range of consumer applications (smart homes, smart TVs, smart watches, etc). The internet revolution of the late 1990s and early 2000s led to the interconnection between people at unprecedented scale and pace. The next revolution will be the interconnection between objects to create a smart environment [5]. Until the emergence of smart-phones the devices connected to the internet shared a large degree of traits - stationary with access to power resources and large computational capabilities - and the specific protocols the internet was built on adhered to these traits. Typical 'things', however, do not share these traits. 'Things' may be mobile, in harsh environments, with limited power and computational resources. With the emergence of open wireless technologies such as Bluetooth, RFID, Wi-Fi and cellular, the ideology of ’things’ fuelling a data driven world has become a reality. A growing world population, world hunger, resource depletion, energy crisis, global warming - these are just some of the challenges the world is faced with today. IoT has the potential to help society face the global challenges of today and tomorrow. By 2020 the number of connected IoT devices is expected to surpass the number of human subscribers with 212 billion smart IoT entities expected to be deployed [6]. The whole annual economic impact of the IoT market is predicted to be in the range of $2.7 trillion to $6.2 trillion by 2025 [7]. Figure 1.2 illustrates the projected market share of dominant IoT applications [7] [8].

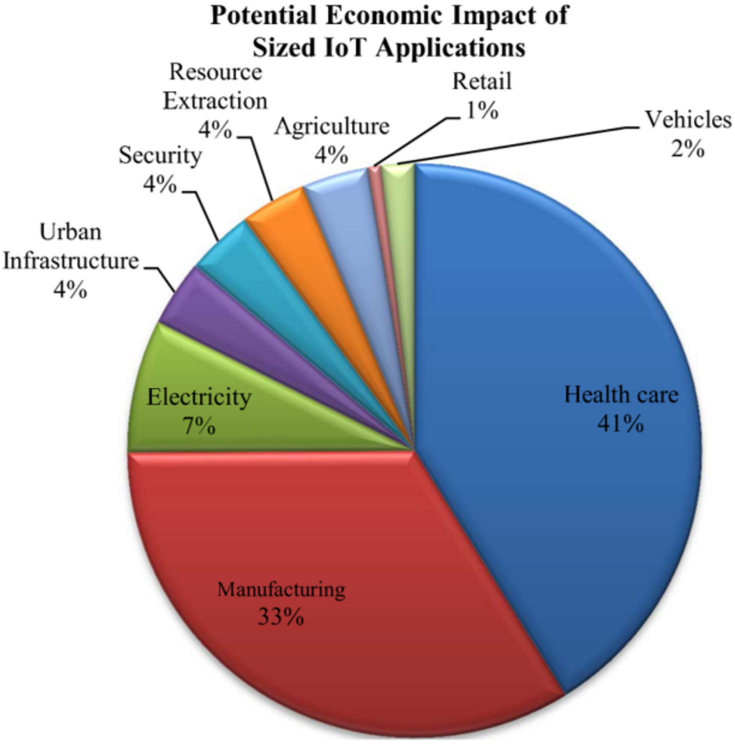


Figure 1.2: Projected Market Share Dominant IoT Applications by 2025 [8]

Communication technologies connect heterogeneous objects to deliver specific smart services. WiFi, Bluetooth (and BLe), Z-Wave, RFID and LTE are all examples of communication protocols used for IoT devices today [8]. Bluetooth and WiFi are two communication protocol standards that define a physical and MAC layer for bandwidth wireless communications within a short range. Bluetooth is oriented to connecting close devices requiring medium to high bandwidth, serving as a substitute for cables, while WiFi is oriented towards computer-to-computer very high bandwidth communications. LTE is a standard wireless communication for high-bandwidth, long range data transfer between mobile phones based on GSM/UMTS network technologies [9] and has been adapted into IoT applications. Figure 1.3 illustrates the various wireless technologies and their trade-offs in terms of bandwidth and range. WiFi and LTE, while offering high-bandwidth at short to long ranges, are power hungry and as such not suitable to IoT applications with limited power resources. Bluetooth, while offering medium to high bandwidth at close ranges with low power consumption, is not suitable to IoT applications requiring long communication ranges. The limited energy resources and the long range requirements of many IoT applications lead to the development of low-power wide-area network (LPWAN) technologies. LPWAN technologies have been designed specifically for long battery life applications requiring very little data to be transmitted over long distances. With a phenomenal range of a few to tens of kilometres [10] and battery life capabilities of ten years and beyond, LPWA technologies are promising for the internet of low-power, low-cost and low-throughput things. LPWA technologies achieve long range and low power operation at the expense of low data rate (typically in orders of tens of kilobits per second) and higher latency (typically in orders of seconds or minutes) [11]. LPWA technologies are considered for those use cases that are delay tolerant, non-mission critical and do not need high data rates. For this reason, LPWA technologies are not suitable for many industrial IoT, vehicle-to-vehicle and vehicle-to-infrastructure applications [12]. However, they still meet the needs of a plethora of applications for smart cities, smart metering, home automation, wearable electronics, logistics, environmental monitoring, amongst others [11]. Low-power wide-area networks represent a novel communication paradigm which will complement traditional cellular and short range wireless technologies in addressing the diverse requirements of IoT applications [11]. Approximately one fourth of the overall 30 billion IoT/M2M devices are to be connected to the Internet through LPWA networks using either proprietary or cellular technologies [13].

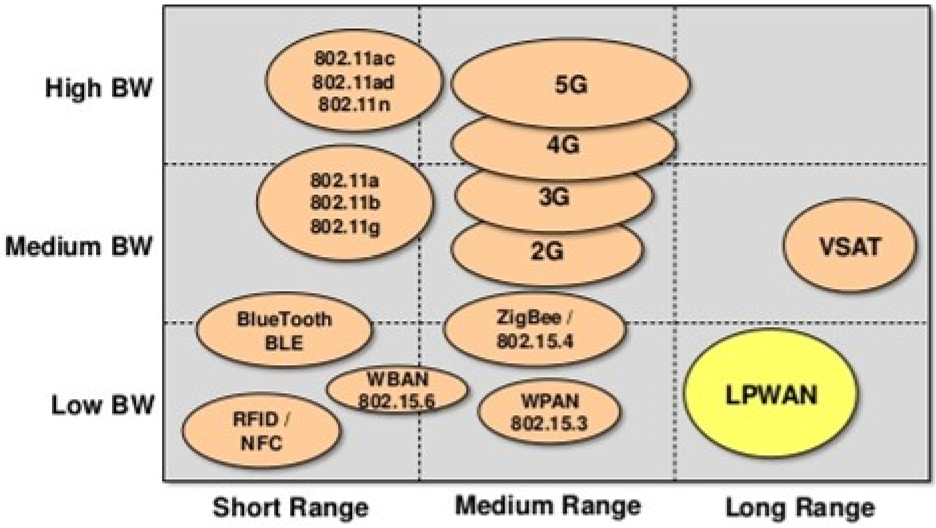


Figure 1.3: Wireless IoT Connectivity Technologies [14]

A critical issue in the effective deployment of these networks is the ability to update firmware after deployment. Owners of smart appliances (smartphones, smartwatches, etc) today will already be accustomed to routine firmware updates.

Traditional firmware update tools are ineffective for wireless sensor networks (WSN). They do not take into account the severe resource constraints at individual nodes or the impact of wireless communications [15]. The resource constraints in terms of energy, memory and processing power present a challenging task for sensor network reprogramming. Firmware images are typically in the order of dozens of kilobytes in size. This poses an even greater challenge for wireless sensor networks utilising low bandwidth LPWA technologies. Motivated by the fact that the processor consumes significantly less energy than the wireless radio, it was shown that reprogramming can be improved in terms of energy efficiency and time required by using data compression and incremental (binary difference) update techniques.

This work seeks to adopt such an incremental firmware update approach for wireless sensor nodes over the low-bandwidth, low-power, wide-area technology LoRa and its protocol LoRaWAN, with particular interest in the time, efficiency and coverage of firmware transmission.

1.2 Objectives

The objective of this work is to investigate LoRa technology and its communication protocol LoRaWAN and evaluate its ability to serve firmware updates in wireless sensor networks. This evaluation requires an analysis of several different fields of research, namely: Wireless Sensor Networks, Low-Power Wide Area Technologies, Firmware Updates in WSNs and Incremental Update Algorithms.

An advantage of LoRa and LoRaWAN as a communication protocol is its ability to serve nodes over a wide area and in remote locations. However, firmware update images are typically in the order of dozens of kilobytes and LoRaWAN is limited to data rates in the range 0.3 kbps to 50 kbps with a EU maximum duty-cycle of 1%. This poses a few challenges in addressing firmware updating in wireless sensor networks over LoRa.

The firmware update size will be the primary determinant on the number of packets required to successfully serve a node. Using the work on incremental firmware update approaches by previous researchers, the context of this work adopts such incremental firmware update approach, with firmware deltas of size 1000 to 7000 bytes considered.

Subsequently, the number of packets is the primary determinant of the time and energy cost incurred on the node during the firmware transmission process. The low data rates and duty cycle restrictions will impose a heavy out-of-operation time cost on the node, while the transmission and receival of packets using the LoRa radio will be the most energy expensive operation of the procedure.

This works designs and evaluates two firmware transmission protocols LoRa Class A devices. Accordingly, the firmware transmission protocol is designed in such a manner to keep protocol overhead (and the resulting overall number of packets and time required) to a minimum. The evaluation considers each of the six available spreading factors, which trade data rate for range and reliability.

1.3 Dissertation Structure

The remaining chapters of this dissertation are organised as follows:

Chapter 2, "State of the Art", reviews prior literature in relation to IoT, WSN, LPWA technologies, and firmware updating in WSNs.

Chapter 3, "Protocol Design", describes the two proposed firmware transmission protocols, the design considerations and the design evaluation.

Chapter 3, "Experimental Setup",

Chapter 4, "Experimental Evaluation", describes the LoRa equipped sensor node, the web service application and LoRa the infrastructure in Dublin city. It also describes the evaluation scenarios from which the results are gathered.

Chapter 5, "Results", presents the firmware transmission results for the evaluation scenarios.

Chapter 6, "Discussion", presents an analysis of the results and evaluates the designed protocols and LoRa's ability to serve firmware updates in wireless sensor networks.

Chapter 6, "Conclusion", discusses the outcomes, contributions and key findings of the work and concludes with proposals for future work.

2 State of the Art

2.1 IoT & Wireless Sensor Networks

2.1.1 Background

The term "Internet of Things", or IoT, was first coined by Kevin Ashton in 1999 [4]. The majority of the literature throughout the 2000s discusses the emerging concept of "wireless sensor networks". At the turn of the decade, the discussion evolves and the term "Internet of Things" really comes to life. Wireless sensor networks are formed from collections of sensing nodes, placed in a sensor field, that collect environmental data and report it via a gateway or base station to an application. In 2002, a survey on wireless sensor networks identified promising new applications: military, environment, health, home and other commercial areas. It is safe to say that all these predictions came through, however, the extent of the Internet of Things today is far too great as to be characterised into those five areas. Today the Internet of Things covers a vast scope of applications, from objects as small as rings with payment capabilities to as large as autonomous driving lorries. The IoT encompasses a considerable number of technologies, as such, there exists a lot of confusion over how to define the term. In [16] the authors define the term IoT: *The Internet of Things allows people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path / network and any service.* In [17] the authors explained the role of sensor nodes and sensor networks in the greater scale of IoT. Data is collected using sensors.

The data is then processed and decisions are made. Finally, actuators perform the decided actions. Sensor networks can exist without the IoT. However, the IoT cannot exist without sensor networks, because sensor networks provide the majority of the hardware infrastructure support, through providing access to sensors and actuators [17].

In 2010, the number of internet connected devices surpassed the earth's human population [18]. By 2020, there is expected to be 212 billion IoT smart objects deployed [19]. By 2025, the whole annual economic impact caused by the IoT is estimated to be in the range of $2.7 trillion to $6.2 trillion [20].

2.1.2 Characteristics

A sensor node is made up of four basic components: a sensing unit, processing unit, transceiver and a power unit [21]. The authors of [22] characterised wireless sensor networks: the sensor nodes are generally small in size with limited hardware capacity (processing power, memory); the nodes are usually battery powered. Amongst their computing, communication and sensing functions, communication consumes the largest portion of their energy. Finally, the network itself can be sparse or dense and host from tens up to hundreds or even thousands of sensor nodes. In most applications, sensor networks are deployed once and intended to operate unattended for an extended period of time. Many other works also characterise sensor nodes as low-cost [23][24], while the authors of [5] identify node failures as another characteristic.

The authors of [17] identified seven major characteristics in the IoT: intelligence, architecture, complex system, size considerations, time considerations, space considerations and everything-as-a-service.

2.1.3 Enabling Technologies

Applications based on Radio Frequency Identification (RFID) first appeared in the 1980s, with original applications in keyless logistics and supply chain management [25]. Since then RFID technology has been applied in healthcare, marketing, museums, passports and much more. Near Field Communication (NFC) is an RFID based technology that enables short-range wireless information exchange. NFC technology uses reader/writer devices and tags as in RFID technology and magnetic induction to power the tag which may not have a power source of its own [26]. Many mobile phones are now NFC enabled, which is pushing the adaption of NFC into applications past those described in RFID, such as mobile payments, ticketing systems, building management, and much more. As the name suggests, RFID technologies, and NFC in particular, are very close range, with NFC having a maximum range of centimetres. Some strains of RFID technologies have ranges spanning many metres, however with this comes greater power consumption and susceptibility to other factors such as interference.

Bluetooth Low Energy (BLE), is a wireless technology developed by the Bluetooth Special Interest Group (SIG) for short range communications. BLE has been designed as the low power edition of the Bluetooth specification, suitable for applications with low power resources. ZigBee is a competing technology that also offers low power consumption at short ranges, with a maximum range of approximately 10 m. Bluetooth and Zigbee are capable of moderate data rates, more than the LPWA technologies that will be introduced, but less than that of WiFi and cellular technologies. Hence, Bluetooth and Zigbee are suitable for applications requiring moderate data rates in relatively close proximity. Today, the technology is frequently in IoT objects in conjunction with the owner's mobile phone. The authors of [27] identify Bluetooth and WiFi as the two major technologies of the wireless scene. Bluetooth is the main technology for connecting devices as a substitute for data transfer peripherals. WiFi, on the other hand, is used for connections among computers, as a replacement for cabled LANs. WiFi is capable of a range up to 100 m with data capacity of several Mb/s. WiFi is suited to applications requiring short to medium range with high data rates, however this comes at a cost of power consumption, and may not be suited to many wireless sensor applications where power is of the essence.

Where long range communication is a sensor node requirement, cellular network technologies step in to offer long range, in the vicinity of multiple kilometres, and high data rates. Cellular networks have a world-wide established footprint and are able to deal with the challenge of ubiquitous and transparent coverage, enabling IoT nodes to be placed in the desired locations to become operational and get connected to the rest of the world [28][29]. Cellular networks, however, were not originally conceived and designed for providing machine-type services to a massive number of devices, and differing from traditional broadband services, IoT communication is expected to generate, in most cases, sporadic transmissions of short packets. Traditional cellular technologies are power hungry, and do not achieve the 10 year lifespan that many sensor node applications desire [11]. All these aspects make current cellular technologies unsuitable to support IoT sensor nodes with long range requirements but power sensitive resources.

Low-Power Wide-Area (LPWA) networks represent a novel communication paradigm, which will complement traditional cellular and short range wireless technologies in addressing the diverse requirements of IoT applications. LPWA technologies offer unique sets of features including wide-area connectivity for low power and low data rate devices, not provided by legacy wireless technologies. LPWA technologies achieve long range and low power operation at the expense of low data rate (typically in orders of tens of kilobits per seconds) and higher latency (typically in orders of seconds or minutes). With a phenomenal range of a few to tens of kilometres and battery life of ten years and beyond, LPWA technologies are promising for the Internet of low-power, low-cost, and low-throughput things [11]. Low-power wide-area technologies cover the largest range with the least power consumption, in comparison with BLE and WiFi. Figure 2.1 compares the range/power trade-off of BLE, Cellular and WiFi with that of LoRaWAN [30]. LPWA technologies are further discussed in the next section.

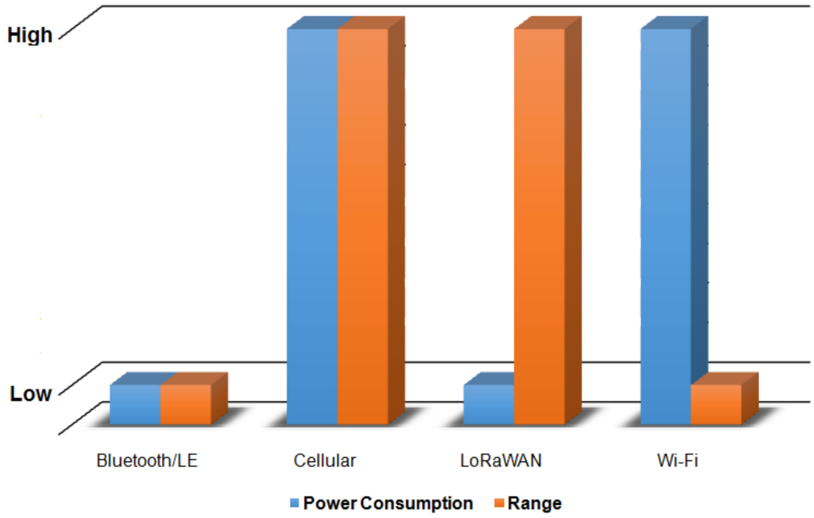


Figure 2.1: Power / Range Comparison for Different Wireless Technologies [30]

2.2 Low Power Wide Area Networks

2.2.1 Market

Many IoT applications are delay tolerant, in harsh remote environments and with low data rate requirements. These applications require low cost and minimal maintenance devices. Low-power wide-area technologies are attracting a lot of attention because of their ability to offer affordable connectivity to low-power devices distributed over very large geographical areas [11].

The commercial success of LPWA networks is tied to connecting a large number of end-devices, while keeping the cost of hardware below $5 [11][31][32][33]. Of the 30 billion IoT devices expected to be deployed by 2025, approximately one fourth are expected to be supported by LPWA technologies [13].

2.2.2 Technologies

There are many flavours of low-power wide-area technologies competing for share of what is expected to be a huge market. As it stands, LoRa and Sigfox are the two technologies at the frontier of the emerging LPWA landscape. Below, a high level overview of the current technologies on the market is provided, and the LoRa technology is given a more thorough review in the next section.

Sigfox

Sigfox was the first LPWAN technology proposed to the IoT market [28][34]. The technology employs an ultra-narrow band modulation operating in the 869 MHz and 915 MHz bands. It can achieve data rates of approximately 100 bits/s uplinks, with a maximum packet payload of 12 bytes and a restriction of 14 packets per day [12].

Weightless

Three open standards for low-power wide-area networks were developed by the Weightless Special Interest Group. The first technology, Weightless-W operates in the 470-790 MHz band, with data rates of 1 kbits/s to 1 Mbits/s, and a battery lifetime of 3-5 years [12]. The second, Weightless-N was designed to expand the range of Weightless-W and reduce the power consumption at the expense of data rate (maximum data rate of around 100 kbits/s) [12]. The third, Weightless-P is a high performance two-way communication that can operate in many frequency bands (169, 433, 470, 780, 868, 915 and 923 MHz).

Ingenu RPMA

Ingenu's RPMA technology does not operate in the sub-GHz bands like most other LPWAs. It operates in the 2.4 GHz ISM band, which in Europe and the USA has no maximum duty cycle limits, therefore enabling higher throughput and capacity than other LPWAs [11]. RPMA stands for Random Phase Multiple Access, which is the patented physical access scheme behind the technology [11]. The technology is capable of up to 624 kbits/s uplinks and 156 kbits/s downlinks, with a shorter range of around 5 to 6 kilometres [12].

IQRF

IQRF technology was developed by the IQRF Alliance based in the Czech Republic. Unlike many of the other technologies it implements a mesh topology instead of a star topology. It operates on the unlicensed frequency bands 868, 916 and 433 MHz, and uses a specific number of channels for each band. Each channel has 100 kHz bandwidth and usually operates at a data rate of 19.836 kbits/s [35][36][37].

2.3 LoRa

2.3.1 Technology

LoRa is a proprietary spread spectrum modulation scheme developed by Semtech. It is Chirp Spread Spectrum modulation (CSS) derivative and trades data rate for sensitivity within a fixed channel bandwidth. It implements variable data rates, utilizing orthogonal spreading factors, which allows the system designer to trade data rate for range or power, so as to optimize network performance in a constant bandwidth [38].

In information theory, the Shannon-Hartley theorem establishes the maximum rate at which information can be transmitted over a communication channel of a specified bandwidth in the presence of noise.

$$C = B\log_2\left(1+\frac{S}{N}\right) \quad bits/second \tag{1}$$

By rearranging from log to ln, and for small signal-to-noise (SNR) ratios such that $\frac{S}{N} << 1$, the capacity formula can be approximated to:

$$C \approx 1.433 \cdot B \cdot \frac{S}{N} \approx B \cdot \frac{S}{N} \tag{2}$$

Rearranging again gives:

$$\frac{N}{S} \approx \frac{B}{C} \tag{3}$$

From Eq. 3 it can be seen that to transmit error free information in a channel of fixed noise-to-signal ratio, only the transmitted signal bandwidth needs to be increased.

In Direct Sequence Spread Spectrum (DSSS) systems, the above process is generally achieved by multiplying the wanted data signal with a spreading code, also known as a chip sequence. The chip sequence occurs at a much faster data rate than the data signal and thus spreads the signal bandwidth beyond the original bandwidth occupied by just the original signal. At the receiver, the wanted data signal is recovered by re-multiplying with a locally generated replica of the spreading sequence. This multiplication process in the receiver effectively compresses the spread signal back to its original un-spread bandwidth.

The amount of direct sequence spreading is dependent on the ratio of chips-per-bit, i.e. the ratio of chip sequence to the wanted data rate. This is referred to as the process gain G_p, where R_c and R_b are the chip rate and bit rate respectively.

$$G_p = 10 \cdot \log_{10}\left(\frac{R_c}{R_b}\right) \quad dB \tag{4}$$

This provides inherent processing gain for the wanted signal (enabling correct recovery at the receiver even when the SNR of the channel is negative). This also reduces interfering signals as these are spread beyond the desired information bandwidth and are easily removed by filtering.

Often DSSS systems require a highly accurate and expensive reference clock source. In addition, the longer the spreading code or sequence, the longer the time required by the receiver to perform a correlation over the entire length of the code sequence. These are especially of concern for low-cost power constrained devices. Semtech's LoRa modulation addresses the above issues to provide a low-cost, low-power and robust spread-spectrum technique.

In LoRa modulation the spreading of the spectrum is achieved by generating a chirp signal that continuously varies in frequency, with the advantage that timing and frequency offsets between transmitter and receiver are equivalent, reducing the complexity of the receiver design.

The chirp frequency bandwidth is equivalent to the spectral bandwidth of the signal. The wanted data signal is chipped at a higher data rate and modulated onto the chirp signal. LoRa modulation can be expressed as follows:

$$R_b = SF \cdot \frac{1}{\frac{2^{SF}}{BW}} \quad bits/second \tag{5}$$

Where the spreading factor, SF, ranges from 7 to 12, and BW is the modulation bandwidth.

From here the symbol period (T_s), symbol rate (R_s) and chip rate (R_c) are defined.

$$T_s = \frac{2^{SF}}{BW} \quad seconds \tag{6}$$

$$R_s = \frac{1}{T_s} = \frac{BW}{2^{SF}} \quad symbols/second \tag{7}$$

$$R_c = R_s \cdot 2^{SF} \quad chips/second \tag{8}$$

Finally, the LoRa modulation also features a variable error correction scheme that improves the robustness of the transmitted signal at the expense of redundancy. The code rate (CR) of an error correction scheme is the proportion of the data-stream that is useful, i.e. non-redundant. Hence, for a code rate of $\frac{4}{5}$, there exists 4 bits of useful information, and the coder generates a total of $5 - 4 = 1$ bits which are redundant. LoRa's CR ranges from 1 to 4, and refactoring it into Eq. 5 gives the nominal bit rate of the data signal as:

$$R_b = SF \cdot \frac{\frac{4}{4+CR}}{\frac{2^{SF}}{BW}} \tag{9}$$

Defining the rate code as:

$$Rate\ Code = \frac{4}{4 + CR} \tag{10}$$

The nominal bit rate can be rewritten as:

$$R_b = SF \cdot \frac{Rate\ Code}{\frac{2^{SF}}{BW}} \quad bits/second \tag{11}$$

2.3.2 LoRaWAN Protocol

The LoRa Alliance is an open, non-profit association of members collaborating together to define and drive the success of the LoRa protocol, LoRaWAN. The LoRa Alliance was founded at the Mobile Web Congress 2015 and counts more than 300 members representing different stakeholders of the LoRa IoT ecosystem from chipsets, modules, devices, gateways to network and application servers (see Figure 2.2) [39].

End-devices, or nodes, are the low power in the field sensor devices that communicate with gateways. Gateways are the intermediate devices that forward packets coming from end-devices to a network server over an IP-based backhaul network allowing higher throughput, such as ethernet or cellular network standards. The Network Server is responsible for the management of the whole network. It receives the data from gateways, authenticates this data and forwards it to to the application server and relays any messages from the application server back to the end-device via the gateways [40].

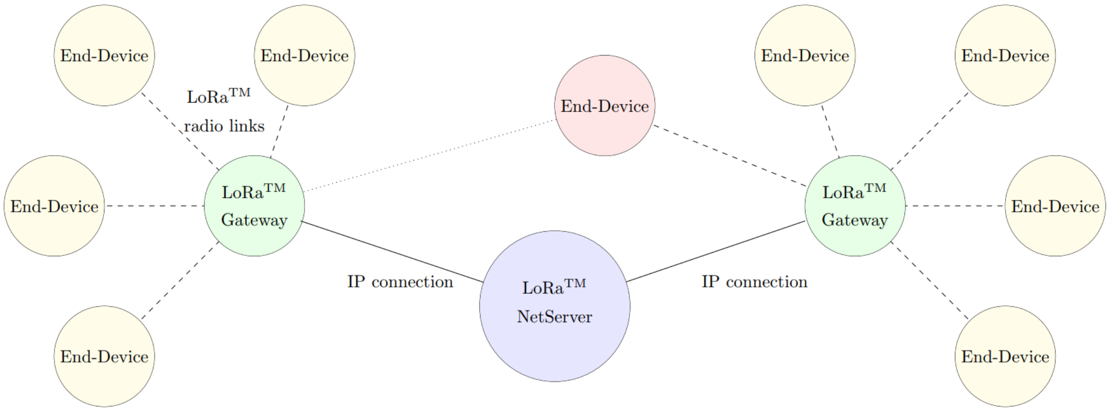


Figure 2.2: LoRa System Architecture [39]

The LoRaWAN specification describes a network consisting of three classes of end-devices, namely *Class A*, *Class B* and *Class C* [41].

Class A devices, *A* for *All*, allow for bidirectional communications whereby each end-device's uplink transmission is followed by two short downlink receive windows. The transmission slot scheduled by the end-device is based on its own communication needs with a small variation based on a random time basis (ALOHA-type of protocol). This Class A operation is the lowest power end-device system for applications that only require downlink communication from the server shortly after the end-device has sent an uplink transmission. Downlink communications from the server at any other time will have to wait until the next scheduled uplink.

Class B devices, *B* for *Beacon*, allow for more receive slots. In addition to the Class A random receive windows, Class B devices open extra receive windows at scheduled times. In order for the end-device to open its receive window at the scheduled time, it receives a time synchronized Beacon from the gateway.

Class C devices, *C* for *Continuous*, have nearly continuously open receive windows, only closed when transmitting. Class C end-devices will use more power to operate than Class A or Class B but they offer the lowest latency for server to end-device communication.

All LoRaWAN devices must implement Class A functionality as a minimum requirement. Devices may in addition implement Class B and Class C functionality, so long as they remain compatible with Class A.

In the context of this work only Class A devices are considered. Hence, only LoRaWAN specifications for Class A devices are reviewed. Further information on Class B and C device specifications is available in the LoRaWAN specifications [41]. The specifications detailed below are only relevant to Class A devices, and while there may be some overlap between the different classes, it should not be assumed that the following is also in accordance with the specifications of Class B and C devices.

Class A specification requires that the end-device opens two short receive windows after every uplink transmission (see Figure 2.3 [41]). The first receive window (RX1) uses a frequency that is a function of the uplink frequency and a data rate that is a function of the uplink data rate. As opposed to the first, the second receive window (RX2) uses a fixed configurable frequency and data rate. Both receive windows open after a delay that can be configured on the device. The length of a receive window must be at least the time required by the end-device's radio transceiver to effectively detect a downlink preamble. If a preamble is detected during one of the receive windows, the radio receiver stays active until the downlink frame is demodulated. If a frame was detected and subsequently demodulated during the first receive window and the frame was intended for this end-device after address and MIC (message integrity code) checks, the end-device must not open the second receive window. If the network intends to transmit a downlink to an end-device, it must initiate the transmission precisely at the beginning of at least one of the two receive windows [41].

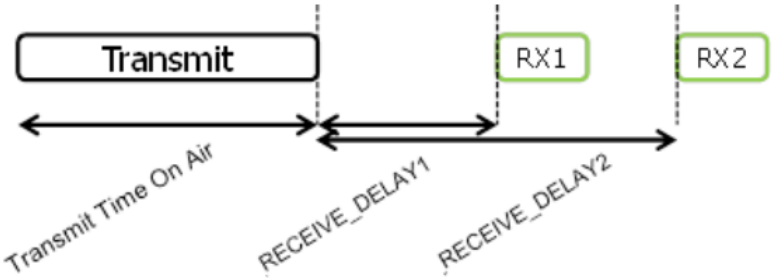


Figure 2.3: Receive Window Timing [41]

Uplink messages are sent by end-devices to the Network Server relayed by one or many gateways (see Figure 2.4 [28]).

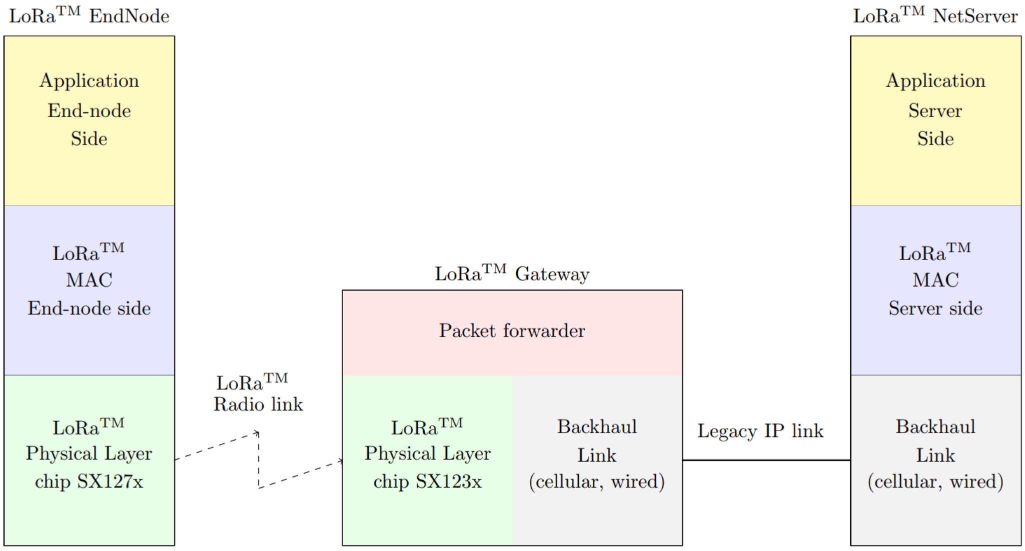


Figure 2.4: LoRa Protocol Architecture [28]

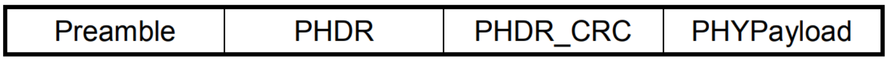
Uplink messages use the LoRa radio packet explicit mode in which the LoRa physical header (PHDR) plus a header CRC (PHDR_CRC) are included (see Figure 2.5 [41]). The integrity of the payload is protected by a CRC. The PHDR, PHDR_CRC and payload CRC fields are inserted by the radio transceiver.

Preamble	PHDR	PHDR_CRC	PHYPayload	CRC

Figure 2.5: Uplink PHY Structure [41]

All the gateways that successfully decode the message by an end-device will forward that packet to the Network Server by adding some information regarding the quality of the reception. The Network Server will hence reply to the end-device by choosing one such gateway [28].

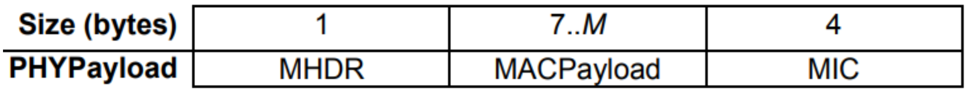
Each downlink message is sent by the Network Server to only one end-device and is relayed by a single gateway. Downlink messages use the radio packet explicit mode in which the LoRa physical header (PHDR) and a header CRC (PHDR_CRC) are included (see Figure 2.6 [41]). No payload integrity check is done at this level to keep messages as short as possible with minimum impact on any duty-cycle limitations of the ISM bands used [41].



Preamble	PHDR	PHDR_CRC	PHYPayload

Figure 2.6: Downlink PHY Structure [41]

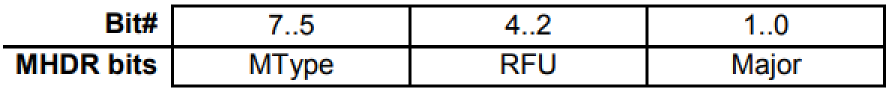
All LoRa uplink and downlink messages carry a PHY payload starting with a single-octet MAC header (MHDR), followed by a MAC payload (MACPayload), and ending with a MIC (see Figure 2.7 [41]). The maximum MACPayload size, *M*, for the varying spreading factors is detailed in Table 2.2.



Size (bytes)	1	7..*M*	4
PHYPayload	MHDR	MACPayload	MIC

Figure 2.7: PHY Payload Structure [41]

The MAC header (see Figure 2.8 [41]) specifies the message type (MType) and according to which major version of the frame format of the LoRaWAN layer specification the frame has been encoded. Bits four to two are reserved for use (RFU).

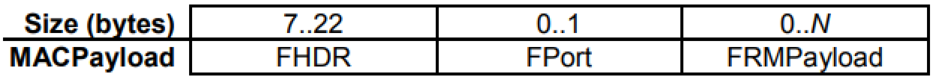


Bit#	7..5	4..2	1..0
MHDR bits	MType	RFU	Major

Figure 2.8: MAC Header (MHDR) Structure [41]

The LoRaWAN specification distinguishes between 8 different MAC message types: join-request, join-accept, unconfirmed uplink, unconfirmed downlink, confirmed uplink, confirmed downlink and proprietary protocol messages.

The MACPayload contains a frame header (FHDR) followed by an optional port field (FPort) and an optional frame payload field (FRMPayload) (see Figure 2.9 [41]). The maximum FRMPayload size, *N*, for the varying spreading factors is detailed in Table 2.2.



Size (bytes)	7..22	0..1	0..*N*
MACPayload	FHDR	FPort	FRMPayload

Figure 2.9: MACPayload Structure [41]

The FHDR contains the short device address of the end-device (DevAddr), a frame control octet (FCtrl), a 2-octets frame counter (FCnt), and up to 15 octets of frame options (FOpts) used to transport MAC commands.

The FCtrl field contains an adaptive data rate (ADR) bit. If the uplink ADR bit is set, the network will control the data rate and Tx power of the end device through the appropriate MAC commands. If the ADR bit is not set, the network will not attempt to control the data rate nor the transmit power of the end-device regardless of the received signal quality. The ADR bit may be set and unset by the end-device or the network on demand. However, whenever possible, the ADR scheme should be enabled to increase the battery life of the end-device and maximize the network capacity [41]. This work aims to evaluate firmware transmission considering each of the six available data rates, and as such, the ADR bit will not be set.

The FCtrl field contains an acknowledgement (ACK) bit. When receiving a confirmed data message, the receiver shall respond with a data frame that has this acknowledgement bit set. If the sender is an end-device, the network will try to send the acknowledgement using one of the receive windows opened by the end device after the send operation. If the sender is a gateway, the end-device transmits an acknowledgment at its own discretion. In the case of a “confirmed” downlink (gateway to end-device), if the acknowledge is not received, the application server is notified and may decide to retransmit a new “confirmed” frame. In the case of "confirmed" and "unconfirmed" uplinks, the end-device will transmit the uplink a configurable number of times or until a valid downlink is received following one of the transmissions. The end-device shall stop any further retransmission of an uplink "confirmed" frame if a corresponding downlink acknowledgement frame is received.

Note that this work only uses unacknowledged messages. This permits more accurate tracking of the number of uplinks transmitted as no uplinks are re-transmitted at the discretion of the node.

The FCnt field is used by the end-device to keep track of the number of data frames sent up from end-device to the Network Server, and sent down from the Network Server to the device.

The FOpts field transports MAC commmands that are piggybacked onto data frames. MAC commands are discussed further on in this literature review.

An FPort field value of 0 indicates that the FRMPayload field contains MAC commands only and any received frames shall be processed by the LoRaWAN implementation. FPort field values of 1 to 223 are application specific and any received frames on one of these shall be made available to the application layer by the LoRaWAN implementation.

This work utilises port 2 for firmware transmission operations to save protocol overhead.

FRMPayload, if present and provided FPort is not zero, carries application specific payloads. For each data message, the FRMPayload is encrypted using AES with a 128 bit key. FRMPayloads are encrypted under the application session key except on FPort 0 where they are encrypted under the network session key.

In this work, FRMPayload, provided FPort is 2, carries data specific to the firmware transmission protocols.

The final PHY Payload field, the message integrity code (MIC), is occupied with the message integrity code calculated over all the fields in the message.

For network administration, a set of MAC commands may be exchanged exclusively between the Network Server and the MAC layer on an end-device. MAC layer commands are never visible to the application or the application server or the application running on the end-device.

To participate in a LoRaWAN network, each end-device has to be personalised and activated. Activation of an end-device can be achieved in two ways, either via *Over-The-Air Activation* (OTAA) or via *Activation By Personalisation* (ABP). Depending on the activation method, the end-device will require certain data to join the network. For OTAA, each end-device is required to have a JoinEUI and DevEUI. The JoinEUI is a global application ID in IEEE EUI64 address space that uniquely identifies the Join Server that is able to assist in the processing of the join procedure and the session keys derivation. The DevEUI is a global end-device ID in IEEE EUI64 address space that uniquely identifies the end-device. DevEUI is the recommended unique device identifier by Network Server(s), whatever activation procedure is used, to identify a device roaming across networks. The JoinEUI and DevEUI are not required for ABP.

Each device is provisioned with two AES-128 root keys during fabrication, namely the network key (NwkKey) and the application key (AppKey).

For over-the-air activation, end-devices must follow a join procedure prior to participating in data exchanges with the Network Server. An end-device has to go through a new join procedure every time it has lost the session context information. Whenever an end-device joins a network via OTAA, the NwkKey is used to derive the forwarding network session integrity key (FNwkSIntKey), serving network session integrity key (SNwkSIntKey) and network session encryption key (NwkSEncKey), and the AppKey is used to derive the application session key (AppSKey). After activation, these sessions keys, as well as a device address (DevAddr), are stored in the end-device. The DevAddr consists of 32 bits and identifies the end-device within the current network. The DevAddr is allocated by the Network Server of the end-device.

Activation by personalization directly ties an end-device to a specific network by-passing the Join-request - Join-accept procedure. Activating an end-device by personalization means that the DevAddr and the four session keys FNwkSIntKey, SNwkSIntKey, NwkSEncKey, AppSKey are directly stored into the end-device instead of being derived from the DevEUI, JoinEUI, AppKey and NwkKey during the join procedure. Thus, the end-device is equipped with the required information for participating in a specific LoRa network as soon as it is started.

In the European Union LoRaWAN operates in the EU 838-870 MHz ISM Band [42]. The network channel be freely attributed by the network operator, however each end-device and gateway must have as a minimum three channels, with bandwidth 125 kHz, centred on the following channel frequencies: 868.10 MHz, 868.30 MHz and 868.50 MHz. Each end-device and gateway must also observe a duty-cycle restriction of less than 1% in these channels. Table 2.1 lists the encoding used for Data Rate (DR) in the 868-870 MHz band [42].

Table 2.1: Data Rate Table [42]

Data Rate	**Configuration**	**Indicative physical bit rate** $(bits/s)$
0	LoRa: SF12 / 125 kHz	250
1	LoRa: SF11 / 125 kHz	440
2	LoRa: SF10 / 125 kHz	980
3	LoRa: SF9 / 125 kHz	1760
4	LoRa: SF8 / 125 kHz	3125
5	LoRa: SF7 / 125 kHz	5470
6	LoRa: SF7 / 250 kHz	11000
7	LoRa: FSK: 50 kbps	50000
8...14	RFU	
15	Defined in [41]	

The maximum MACPayload size length, M, as discussed above, is given in Table 2.2. The maximum application length in the absence of the optional FOpt field, N, is also given. The Net Server infrastructure in this work requires the FOpt field to be present, hence the maximum application payloads are given by N.

LoRaWAN Regional Parameters for regions outside the EU are not in the scope of this work and can be found in [42].

Table 2.2: Maximum Payload Size [42]

Data Rate	**M** $(Bytes)$	**N** $(Bytes)$
0	59	51
1	59	51
2	59	51
3	123	115
4	230	222
5	230	222
6	230	222
7	230	222
8...15	Not Defined	

2.3.3 Bandwidth

Considering that each packet is prepended with a preamble to allow for frequency and timing synchronisation at the receiver, the actual data rate ranges approximately from 0.3 kbps to 11 kpbs, with $BW = 250kHz$. However, the system capacity is larger because the receiver can detect multiple simultaneous transmissions from different nodes by exploiting the orthogonality of the spreading sequences [38] [28].

In [43], the authors investigated the capacity and scalability of the LoRa network, and their work produced interesting results relating to the maximum uplink throughput for a single end-device. They considered the fact that an end-device is prohibited to transmit another uplink packet before it has either received a downlink in RX1 or RX2, or else wait until RX2 expires. Also, an end-device has a configurable time-out period after an unacknowledged uplink (default is 2 seconds). Their calculations showed that in the case where no downlink is received, the minimum packet period varied from 5.0 s at spreading factor SF 12 to 2.0 s at SF 7. They also presented findings on the maximum throughput for a single end-device under different duty-cycle restrictions. After transmitting a frame on a channel, this channel cannot be used for the next time-off (T_{off}) seconds. Time-off is given by:

$$T_{off} = ToA \cdot \left(\frac{1}{DutyCycle_{channel}} - 1 \right) \quad seconds \tag{12}$$

Where ToA is the time on air for the transmitted packet. Taking this into consideration they calculated the long-term throughput average for end-devices for each particular spreading factor. The results presented indicate a lesser maximum throughput than the indicative throughput in the LoRa specifications, with a difference of 103.9 bits/s at SF 12 and 4339.6 bits/s at SF 7.

2.3.4 Range

In [44] real-life experiments were conducted using commercially available hardware to evaluate the range of LoRa. Their experiments were conducted both over land (on a moving car) and on water (on a moving boat) in the city of Oulu, Finland. The transmitting nodes used the largest possible spreading factor, 12, as the aim of the experiment was to evaluate range. They reported a ximum range of just over 15 km on land, however with low reliability. On water a maximum range of close to 30 km was achieved, with a success rate of 62% in the 15-30 km range (results shown in Tables 2.3 and 2.4).

In [45] a comparison of LoRa modulation at operating frequencies 433 MHz and 868 MHz was carried out. All the tests were carried out at the outskirts of Zaragoza, Spain. The transmitting end-device and receiving gateway were separated by a distance of approximately 7 km and elevation 300 m. Their results showed that by decreasing the operating frequency of the transceivers it is possible to achieve a greater sensitivity which can potentially be used to achieve greater transmission range.

Table 2.3: Range Evaluation Over Land Results [44]

Range (km)	**Num. of TX Packets**	**Num. of RX Packets**	**Packet Loss Ratio** (%)
0-2	894	788	12
2-5	1215	1030	15
5-10	3898	2625	33
10-15	932	238	74
Total	6813	4506	34

Table 2.4: Range Evaluation Over Water Results [44]

Range (km)	**Num. of TX Packets**	**Num. of RX Packets**	**Packet Loss Ratio** (%)
5-15	2998	2076	31
15-30	690	430	38
Total	6813	4506	32

2.3.5 Robustness and Reliability

The range evaluation experiments in [44] also reported results on packet loss data. These results are also shown in Table 2.3 and Table 2.4. Their base station for packet reception was stationary throughout the experiments and positioned 24 m above sea level on the University of Oulu antenna tower. Over land, they found that within a 2 km range, received signal exceeds -100 dBm, however 12 % of packets were lost. Over the 2-5 km range the packet loss stayed below 15%. This increased to 33% and 74% over ranges 5-10 km and 10-15 km respectively. Experiments in this work are carried out in a city centre location with close access to gateways. It is expected that most packets will be received by gateways within a 0-2 km range.

In [46] the indoor performance of LoRa was evaluated on the campus of University of Oulu, Finland. A battery powered end-device was attached to the arm of a researcher as he went about his daily routine, moving about buildings and the campus. The base station remained stationary atop the university's antenna tower at 24 m above sea level. The end-device was configured to operate at the largest spreading factor 12 and 125 kHz. The results indicated a high packet success rate, with 96.7% of data packets being successfully received, excluding those from the anechoic chamber where no packets were successful. 95% of packets were successful as the researcher was moving between locations. The maximum distance between the end-device and the base station throughout the experiment was 420 m which would account towards the high reliability, a range quite below the maximum land range of around 15 km observed in [44].

In [47] they investigated the quality of service (QoS) that a LoRa network could provide. They designed their own protocol stack, LoRa FABIAN, and investigated a range of performance metrics, such as packet error rate (PER), received signal strength indicator (RSSI) and signal to noise (SNR). They found that the antenna location and its elevation play a major role in the network performance, and in best conditions frame losses were as low as 3%. In urban areas, packet error rate was over 40%. They also found high PERs, greater than 30%, in non-urban areas, but remarked that may be partly explained by poor elevation profiles between the LoRa station and the measurement.

The comparison between operating frequencies 433 MHz and 868 MHz performed in [45] found that at lower bandwidths there is an increased packet error rate and packet error loss rate (PELR). Bandwidths tested in their experiments ranged from 7.8 kHz to 500 kHz. At the lower bandwidths not only were there more packets lost, but the majority of received packets contained corrupted data. They reported greater error and packet loss percentage at lower bandwidths for SF 12. SF 10 configuration presented the best RSSI of all with a low PER and PELR percentage. Their poorer results at lower bandwidths conflict with theory, which assumes that by lowering the signal bandwidth it is conceivable to achieve better, or at least the same results, as higher bandwidths. They noted greater incurred channel noise at lower bandwidths and location of the receiver as possible reasons for this. In conclusion, their comparison between operating frequencies 433 MHz and 833 MHz, 433 MHz presented a better performance in all trials bar one. They found SF 10 achieves the best RSSI, however if the transceiver separation is moderate, SF 8 is also a viable option.

2.3.6 Energy

In [48] the authors presented analytical models that allow the characterisation of LoRaWAN end-device current consumption, lifetime and energy cost of data delivery. Their models were derived based on measurements taken using the commercially available MultiConnect mDot platform from Multitech used with the SX1272 transceiver. They assumed a periodic behaviour for the LoRaWAN end-device, where a time, T_{Notif}, is waited between two consecutive uplinks transmissions, and the device is in sleep mode. They presented three results on energy performance: end-device current consumption, end-device lifetime, and energy cost of data delivery. First, for unacknowledged transmissions, they found that the average current consumption decreases as T_{Notif} increases, since the sleep duration is longer between intervals. They reported that the average current consumption decreases with the data rate, since the duration of transmit and receive intervals is inversely proportional to the bit rate of the data rate used. For acknowledged transmissions, like unacknowledged, the average current consumption decreases as T_{Notif} increases. A key finding from their report is that if the acknowledgement is received in the first receive window, the current consumption may be less than that of unacknowledged packets, since the second receive window need not be opened. However, if the acknowledgement is received in the second receive window, the average energy consumption is greater than that of unacknowledged (see Figure 2.10).

Considering end-device lifetime, they showed that a maximum value of 5.96 years is obtained at DR 6 with a T_{Notif} of 1440 min. Their results also showed that lower DRs lead to lower end-device lifetime, and this effect increases as T_{Notif} decreases. For a $T_{Notif} = 5min$, end-device lifetime values fall below one year, ranging from 0.26 years to 0.83 years for DR 0 and DR 5 respectively. Considering the same DRs,

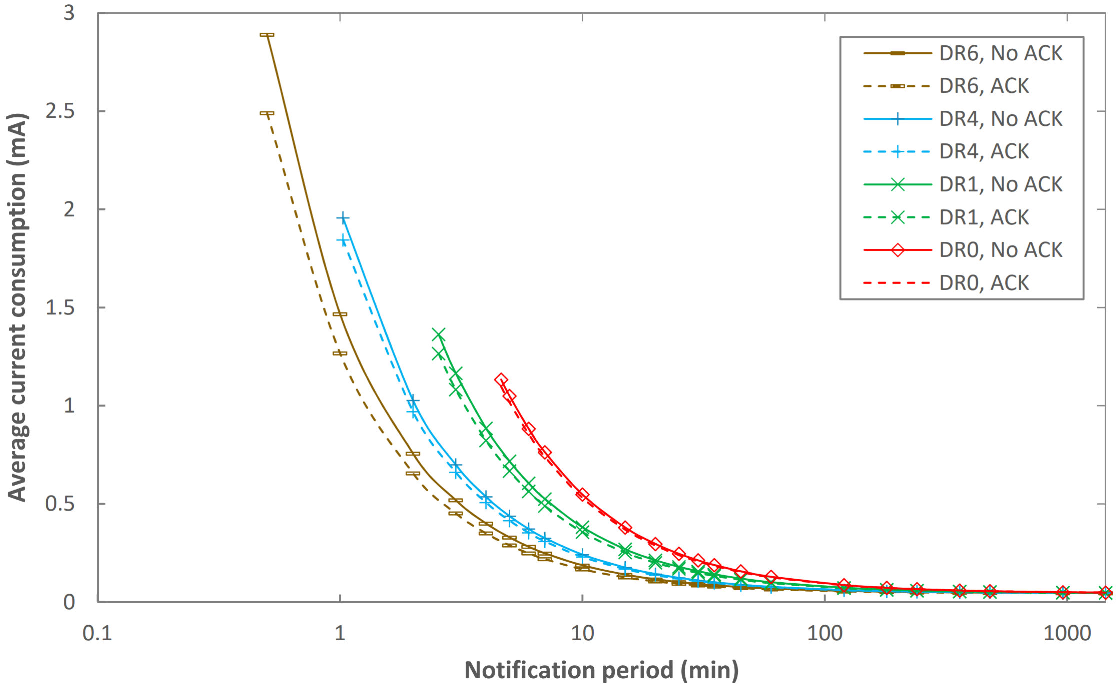


Figure 2.10: Acknowledged and Unacknowledged Energy Consumption for different T_{Notif} and DRs [48]

$T_{Notif} = 60min$ results in a lifetime of 2.13 to 3.76 years while $T_{Notif} = 360min$ results in 4.55 to 5.52 years. They found that the impact of payload size was only noticeable when T_{Notif} was low (see Figure 2.11).

In conclusion, they reported that an end-device running on a 2400 mAh battery and sending one message every 5 minutes can achieve a 1 year lifetime. Also, the energy cost per bit of maximum-sized frame payload transmission is roughly two orders of magnitude lower than the one obtained for a shorter payload of 1 byte per frame. Hence, an end-device operating as a sensor would significantly benefit from accumulating readings and sending them at the highest notification period possible.

This work, taking this into account, sends downlink carrying firmware update packets at the maximum payload size.

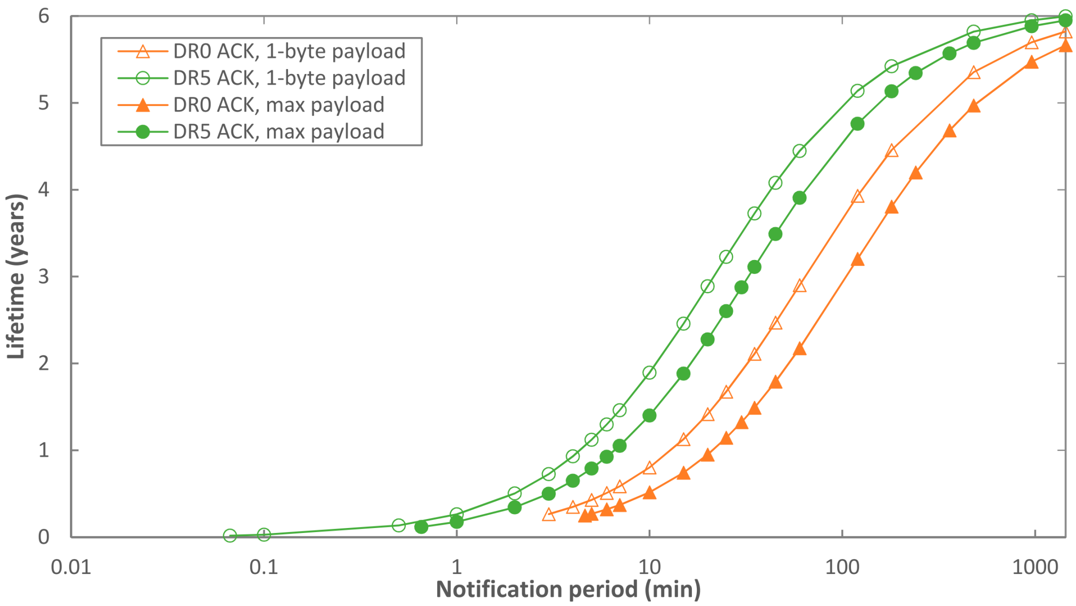


Figure 2.11: Acknowledged Energy Consumption for different T_{Notif} and DRs and Payload Size [48]

2.3.7 Scalability

LoRa networks are operated using ALOHA in a star configuration. Spreading factor programmability allows nodes closer to the base station to be operated at higher data rates to save power with shorter messages. Nodes far away from the base station can then use maximum spreading factor to maximise their range. Introducing new base stations reduces the need for range and increases the network capacity. If adaptive data rate is enabled and devices are properly managed the capacity is increased even more so [39]. A single LoRa base station, when compared against ultra-narrow band (UNB), has serious capacity limitations, however, with a sufficient number of base stations, the need for higher spreading factor is reduced and the network capacity increases, offering a potentially better capacity than UNB networks [39]. In practicality, SF7 to SF10 should be used for reaching outdoor objects, while SF11 and SF12 should be reserved to reach deep indoor objects when needed [39].

In [49] the authors investigated the capacity limits of LoRa networks. They used results from practical experimentation to build a LoRa simulator. This simulator was then used to evaluate the scalability of LoRa networks. They concluded that at configuration SF 12, BW 125 kHz, and CR 4/5, with end-devices transmitting a 20 byte packet every 16.7 mins, and requiring a data extraction rate (DER) of 90%, a single gateway network could support 120 nodes. Their evaluation of a network with dynamic parameters, where a network controller could control the parameters of individual nodes, a single gateway could support 1600 nodes under optimal conditions, the authors do note that this result relies on optimistic assumptions. Their third simulation investigated the effect of multiple sinks. The results are shown in Figure 2.12. The graph shows that increasing the number of sinks significantly increases the DER. In conclusion, they noted that in the typical scenario where a node is in range of just a single sink, this sink could handle 120 nodes in a 3.8 ha area.

In [43], similar to the work in [49], the authors provided an in-depth analysis of the potential throughput available to a single LoRa end-device and the number of devices which can be served by a single base station for a few different scenarios. They presented interesting results, where depending on the scenario and application, a single base station could serve over a million nodes. The majority of the nodes, however, must reside in the vicinity of the base station, with less than 10% residing at distances greater than 5 km. In their discussion, they do note that another factor limiting the scalability of LoRaWAN is the use of acknowledgements, given the fact that the base station must also obey duty cycle restrictions. Therefore, in reality, it cannot acknowledge each and every packet. They acknowledge that this will potentially cause poor downlink performance. This will have to be carefully considered in this work's firmware transmission protocol, as the base stations will respond to uplinks with downlinks carrying firmware segments.

2.3.8 Applications

LoRa modulation is also capable of localisation services using Time-Difference-of-Arrival (TDOA) techniques. In rural environments, deployments are leading to an accuracy of better than 40 m, while in urban areas, due to multipath

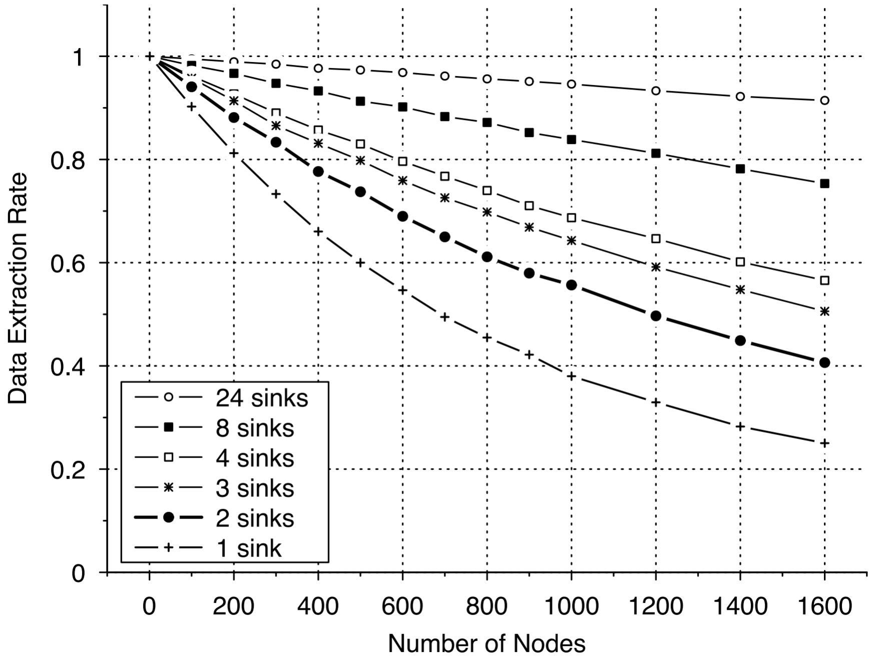


Figure 2.12: Effect of Increasing Sinks on DER [49]

issues, accuracies of around 150 m have been reported [39].

To conclude the literature reviewed on LoRa, below is some identified use cases and applications of the emerging technology. The authors of [12] considered the use of LoRa technology in five scenarios: real time monitoring, metering, smart city applications, smart transportation and logistics and video surveillance. In real time monitoring, applications can range from leak detection, environment control to industrial automation. In cases such as leak detection, with a reduced number of periodic/aperiodic messages and dispersed locations, LoRaWAN serves as a suitable technology. However, in industrial automation, where industrial control loops may require response times of 1 ms to 100 ms, LoRaWAN cannot claim to be a candidate solution. LoRaWAN has shown key success stories in smart lighting, smart parking and smart waste collection. These sorts of applications monitor events which are slow and therefore network signalling is limited to a few times a day. The authors assert that LoRaWAN is not suitable for delay intolerant smart transportation and logistics, but solutions such as fleet control and management may be supported by LoRaWAN. Finally, for video surveillance, the authors note that LoRaWAN would not be capable of supporting video surveillance, which has data rate requirements of 130 kbps for low quality to 4 Mbps for high resolution, while LoRa supports a range of 0.3 kbps to 50 kbps.

The authors of [30] discussed LoRaWAN in the context of smart city applications. On top of the applications identified above, they discussed LoRaWAN in an agricultural context, with ear tags on cattle operating as end-devices. With the appropriate use of gateways and using the localisation services discussed above, this technique could be used for cattle tracking and animal tracking on a greater scale.

2.4 Firmware Updates in WSNs

2.4.1 Reasons and Challenges

Much literature exists on the reasons why firmware may require updating in a wireless sensor network. In [50] five typical firmware update scenarios were identified: *Iterative development; Sensor network testbeds; Correction of software bugs; Application reconfiguration; Dynamic applications.* Table 2.5 compares the different scenarios and their properties [50]. *Update fraction* refers to the amount of the system that needs to be updated for every update and *program longevity* refers to how long an installed program will be expected to reside on the sensor node.

Software development is an iterative process where code is written, installed, tested

Table 2.5: Qualitative Comparison Between Different Reprogramming Scenarios [50]

Scenario	Update Frequency	Update Fraction	Program Longevity
Iterative dev.	Often	Small	Short
Testbeds	Seldom	Large	Long
Bug fixes	Seldom	Small	Long
Reconfiguration	Seldom	Small	Long
App. reconfig.	Often	Small	Long

and debugged in a cyclic fashion. Being able to dynamically reprogram parts of the sensor network system helps shorten the time of the development cycle. During the development cycle developers typically change only one part of the system, possibly only a single algorithm or a function. A sensor network used for software development may therefore see copious amounts of small changes to its code.

Sensor network testbeds are a valuable tool for development and experimentation with sensor network applications. New applications can be tested in a realistic setting and important measurements can be obtained [51]. When a new application is to be tested in a testbed, the application typically is installed in the entire network. The application is then run for a specified time, while measurements are collected both from the sensors on the sensor nodes and from network traffic.

The need for correcting software bugs in sensor networks was identified early [52]. Even after careful testing, new bugs can occur in deployed sensor networks caused by, for example, an unexpected combination of inputs or variable link connectivity that stimulate untested control paths in the communication software [53]. Software bugs can occur at any level of the system. To correct bugs, it must therefore be possible to reprogram all parts of the system.

In an already installed sensor network, the application may need to be reconfigured. This includes change of parameters, or slight changes in the application such as changing from absolute temperature readings to notification when thresholds are exceeded [54]. Even though reconfiguration does not necessarily include software updates [55], application reconfiguration can be done by reprogramming the application software. Hence software updates can be used in an application reconfiguration scenario.

There are many situations where it is useful to replace the application software of an already deployed sensor network dynamically. One example is the forest fire detection scenario presented in [56] where a sensor network is used to detect a fire. When the fire detection application has detected a fire, the fire fighters might want to run a search and rescue application as well as a fire tracking application. While it may be possible to host these particular applications on each node despite the limited memory of the sensor nodes, this approach is not scalable [17]. In this scenario, replacing the application on the sensor nodes leads to a more scalable system.

Traditional update tools are ineffective for WSNs, as they do not take into account the severe resource constraints of individual nodes, or the impact of wireless communication [15]. The resource constraints in terms of energy, memory and processing power make sensor network reprogramming a challenging task.

In [57], a number of sensor node reprogramming performance requirements were identified: for the updates to be non-intrusive; for the update protocols to have a low impact on the performance of the WSN sensing application, and to have a minimal impact on the energy resources of the nodes, and to be resource aware; the need to minimise the impact on sensor network lifetime, and to limit the use of memory resources; the need to minimise processing, limit communications in order to save energy, and only interrupt the application for a short period while updating the code; the possible need for the update to meet various real-time constraints; the need to fit within the hardware constraints of different platforms; and the need to cope with asymmetric links and intermittent disconnections, meet the conflicting needs of low overhead for update metadata overhead while providing rapid update propagation, and to be scalable to very large, high density networks. These are important considerations for the design of a firmware transmission protocol over LoRa. An update protocol should have a minimal impact on the performance of the sensing application. However, due to the low data rates of LoRa it is expected that the transmission of a large amount of data such as a firmware update will take a considerable amount of time, therefore potentially impacting the sensing application.

2.4.2 Incremental Approaches

The authors of [58] investigated the improvement of energy efficiency and delay of reprogramming using data compression and incremental update techniques. They investigated the performance of five different compression algorithms: FastLZ, LZ77, LZJB, RLE and S-LZW; and they also investigated these algorithms used in combination with three binary diff algorithms: bsdiff, rcdiff and vcdiff. The results for the compression achieved are shown in Figure 2.13 [58].

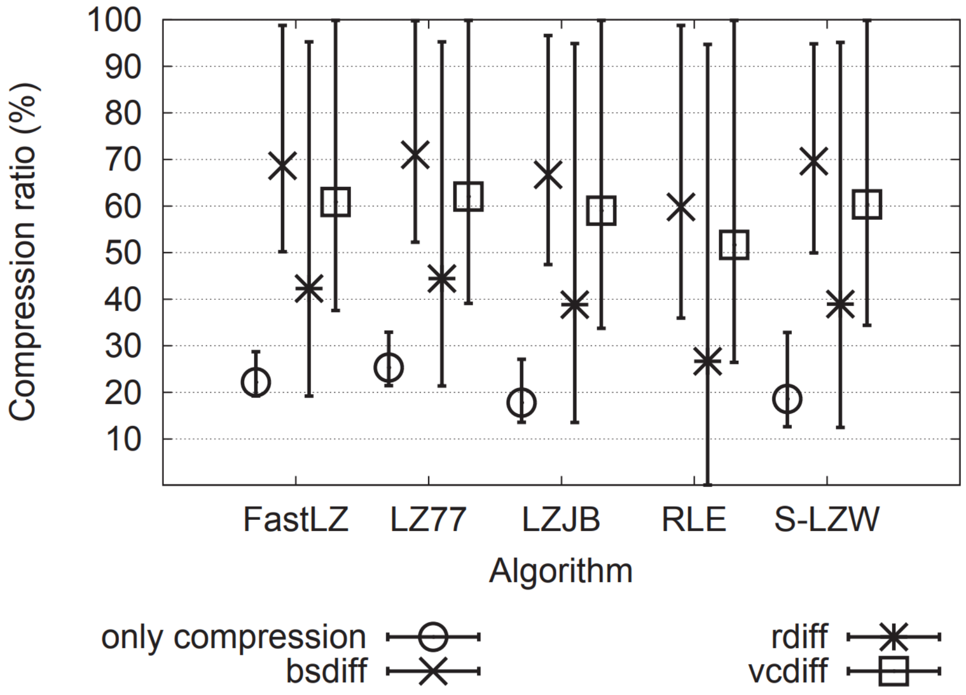


Figure 2.13: Minimum, Maximum and Average Compression Ratios [58]

They concluded that adding compression alone does not lead to lower energy performance or faster updates. However, the use of incremental updates showed solid results in all test cases with up to 95% energy savings and 95% faster updates registered. The four principal factors that influence the choice of algorithm are the following: available resources, update type, networks size, and optimisation goal (energy or delay). In the case of updates over LoRa, delay will inevitably be long due to the duty cycle limitations, however, a significant reduction of packets required would reduce the delay. In terms of energy, packet transmission and receival will be energy expensive operations. Hence, the objective is to keep the number of packets to a minimum.

Many researchers have designed and published incremental firmware update algorithms, each with their own advantages and disadvantages. Most of the research literature available describes the optimisations made by the new algorithm and compares its performance against the performance of existing algorithms for a variety of update scenarios.

The authors of [59] presented one such paper. They designed a new algorithm called Zephyr. Figure 2.14 compares the performance of their incremental firmware update approach against that of existing approaches (Deluge, Stream, Rsync, Semi Optimised Rsync and Optimised Rsync) for six software change scenarios.

1. Blink application blinking a green LED every second to every two seconds.
2. Few lines added to the Blink application.
3. Blink application to CntToLedsAndRfm application which displays the lowest three bits of the counting sequence on the LEDs as well as sending them over radio.
4. CntToLeds to CntToLedsAndRfm. CntToLeds is the same as CntToLedsAndRfm except that the counting sequence is not transmitted over radio.
5. Blink application to CntToLeds application.
6. CntToRfm to CntToLedsAndRfm, where CntToRfm is the same as CntToLedsAndRfm except that the counting sequence is not displayed on the LEDs.

As can be seen in Figure 2.14, the firmware update delta to be transmitted is less than 5000 bytes in most scenarios. In this work, deltas of 7000 bytes and less will be evaluated.

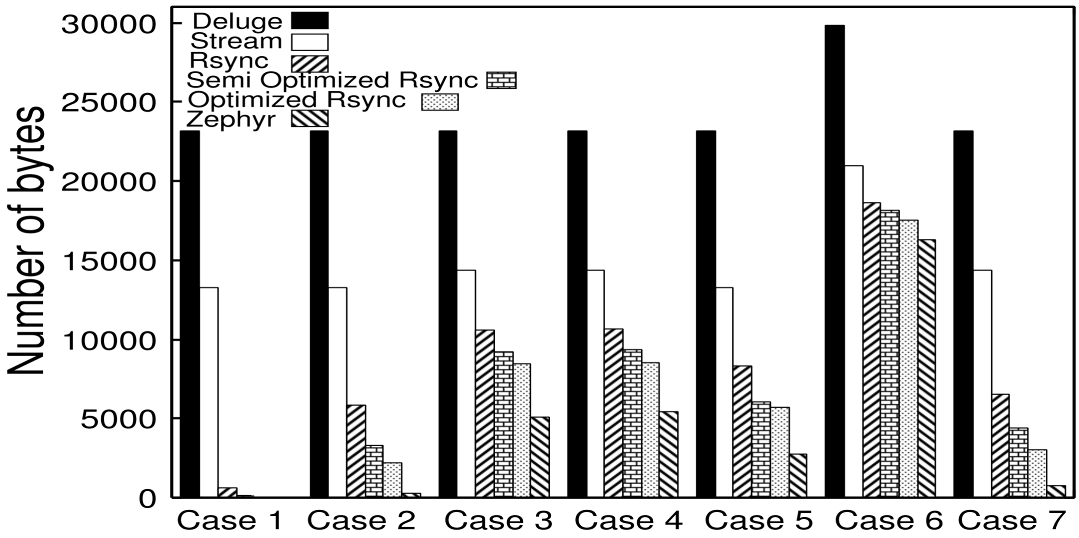


Figure 2.14: Software Change Scenarios on TinyOS [59]

3 Protocol Design

3.1 Design Considerations

The principal objective is to transmit a firmware update delta to the node. As wireless sensor nodes are typically constrained by low power resources, and wireless communication being an energy-intensive operation, transmission efficiency is of the utmost importance.

The unique bi-directional communication nature of LoRaWAN Class A devices has some implications on the design of protocol that must be considered.

3.1.1 Firmware Transmission

First, consider that a Class A device allows for bi-directional communication by opening two receive windows after every uplink (see Figure A1.3 [60]).

The uplink transmission slot is determined by the node when it has duty cycle allowance. Therefore, to transfer of a firmware update packet to the node, the firmware update packet can only be transmitted in response to a successfully received uplink and must be received in one of the two receive windows.

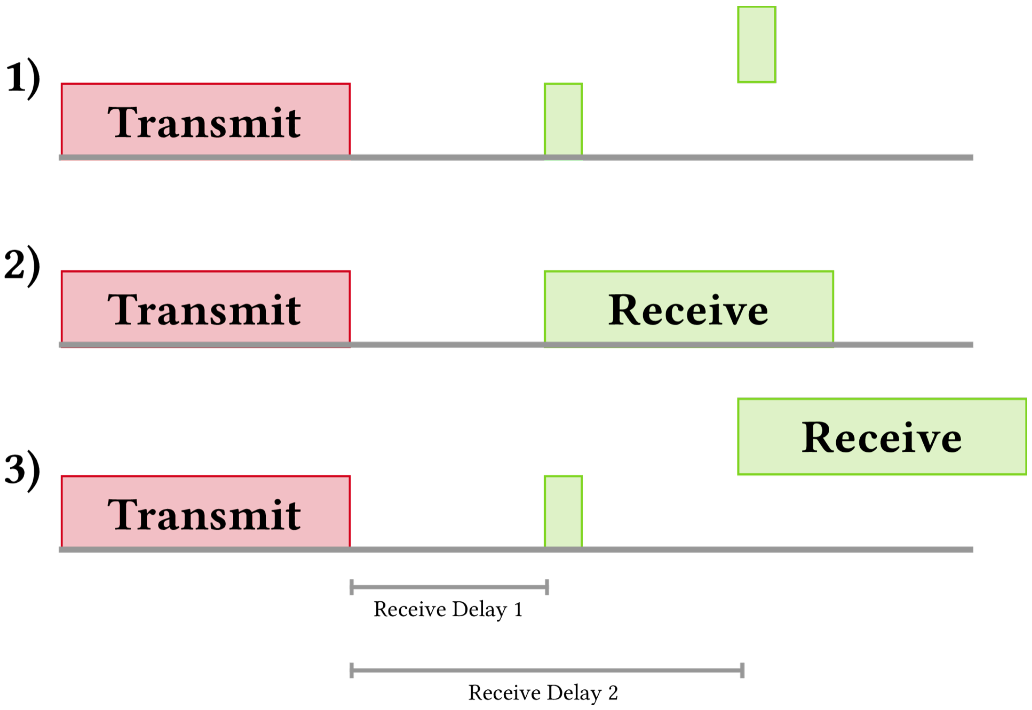


Figure 3.1: Uplink and its Receive Windows [60]

3.1.2 Downlink Scheduling

Second, considering that a downlink can only be sent in response to an uplink, the downlink must be scheduled for transmission before the arrival of the uplink. Therefore, in reality, the server can only send the response to a particular uplink in the receive windows of the next uplink. This process is illustrated in Figure 3.2. Hence, the server and node will always be one step out of sequence. This inherently leads to inefficiencies in the protocol designs as the server cannot respond immediately to the most recent information.

3.1.3 Firmware Assembly

Firmware update images can typically be in the order of tens or hundreds of kilobytes in size. Incremental firmware deltas, while much smaller than the original image, are still in the range of a few kilobytes. LoRaWAN specifications allow a maximum packet payload of 222 bytes at DR 5 (SF 7, BW 125 kHz) and 51 bytes at DR 0 (SF 12, BW 125 kHz). The LoRa infrastructure used in this work limits the downlink payload size to

```
----------------------------------------------------------------------------
                       NODE            SERVER
----------------------------------------------------------------------------
           [SEND UPLINK #N] ---------> [RECEIVE UPLINK #N]
     [NO RESPONSE RECEIVED]            [NO RESPONSE SCHEDULED]
                                       [SCHEDULE DOWNLINK RESPONSE #M] ----+
                                                                           |
                                                                           |
         [SEND UPLINK #N+1] ---------> [RECEIVE UPLINK #N+1]               |
       [RECEIVE DOWNLINK #M] <--------- [SEND DOWNLINK #M]  <---------------+
                                       [SCHEDULE DOWNLINK RESPONSE #M+1] --+
                                                                           |
                                                                           |
         [SEND UPLINK #N+2] ---------> [RECEIVE UPLINK #N+2]               |
     [RECEIVE DOWNLINK #M+1] <--------- [SEND DOWNLINK #M+1]<---------------+
                                       [SCHEDULE DOWNLINK RESPONSE #M+2] --+
                                                                           |
                                                                           |
         [SEND UPLINK #N+3] ---------> [RECEIVE UPLINK #N+3]               |
     [RECEIVE DOWNLINK #M+2] <--------- [SEND DOWNLINK #M+2]<---------------+
                                       [SCHEDULE DOWNLINK RESPONSE #M+3] --+
                                 .
                                 .
                                 .
```

Figure 3.2: Scheduling Downlinks

51 bytes, irrespective of the data rate. To be most efficient, the protocol should send the firmware update in as few packets as possible. Hence, downlink packets will be the maximum size of 51 bytes.

Naturally, the firmware will need to be split into smaller individual packets for transmission. A firmware update carrying downlink will need to contain in its payload both a firmware segment and an index identifying this segment. The index will be used by the node to correctly assemble the complete firmware update.

3.1.4 Error Control

The delivery of each firmware segment is critical. The two protocols designed in this work utilise an Automatic Repeat reQuest (ARQ) error-control method for data transmission that uses acknowledgements to achieve reliable data transmission. Upon receipt of a firmware update carrying downlink, the node will extract the firmware segment and the identifying index. The node will acknowledge to the server the receipt of the firmware segment index in the next uplink opportunity. If the server does not receive an acknowledgement of delivery it will reschedule the unacknowledged firmware segment for re-transmission.

In this work two protocols are designed and evaluated:

1. Stop and Wait ARQ based protocol.
2. Selective Repeat ARQ based protocol with Piggybacking.

3.2 Design Evaluation

The aim is to receive the full firmware update image delta as efficiently as possible. The manner of firmware version control varies from application to application. In certain applications a control centre manages the node's firmware. In these cases, the control centre has all the information to calculate incremental firmware update deltas. In other applications where the control centre doesn't have access to this level of information, the node may need to inform the control centre of its current firmware version. In some cases, the firmware transmission process may be initiated by the control centre, where in others it may be initiated by the node. As such, the evaluation presented here only considers the firmware transmission process from the first uplink request for a firmware update packet to the acknowledgement of the receipt of the last firmware update packet. All other processes, such as firmware update delta preparation, integrity checks and firmware execution are out of scope.

As discussed in the design considerations, the node can only receive firmware update packets in the opened receive windows after an uplink. Hence, to be most efficient the node should receive a unique firmware update packet in response to every uplink.

For the protocol evaluation two uplink classifications are defined: Effective Uplinks and Ineffective Uplinks.

3.2.1 Effective Uplink

An effective uplink is defined as an uplink which receives in its receive windows unique and useful data. It is classified as effective because the energy cost of the uplink transmission and receive windows is justified by the useful data received (see Figure 3.3).

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

                                 .
                                 .
                                 .
```

Figure 3.3: Effective Uplink

3.2.2 Ineffective Uplink

An ineffective uplink is therefore defined as an uplink which receives no data or non-unique data in its receive windows. It is classified as ineffective because the energy cost of the uplink transmission and receive windows is wasted as no useful data was received. Each protocol design has inherent ineffective uplinks and is outlined in the protocol description.

The following two uplink scenarios are classified as ineffective.

Ineffective Data Uplink

An uplink which receives in its response windows non-unique or non-useful data (see Figure 3.4). In the context of this work, an ineffective data uplink could be an uplink which receives a duplicate firmware update packet.

```
----------------------------------------------------------------------------
                       NODE            SERVER
----------------------------------------------------------------------------
                                 .
                                 .
                                 .
                                       [SCHEDULE DOWNLINK RESPONSE #M] ----+
                                                                           |
                                                                           |
           [SEND UPLINK #N] ---------> [RECEIVE UPLINK #N]                 |
      [RECEIVE DOWNLINK #M] <--------- [SEND DOWNLINK #M]  <---------------+
                                       [SCHEDULE DOWNLINK RESPONSE #M] ----+
                                                                           |
                                                                           |
         [SEND UPLINK #N+1] ---------> [RECEIVE UPLINK #N+1]               |
      [RECEIVE DOWNLINK #M] <--------- [SEND DOWNLINK #M]  <---------------+
    (DUPLICATE DOWNLINK DATA)
                                 .
                                 .
                                 .
```

Figure 3.4: Ineffective Data Uplink

Ineffective Response Uplink

An uplink which receives no response in its receive windows. Ineffective response uplinks can result from one of the following five scenarios.

1. Uplink is lost in transmission.
2. No downlink scheduled.
3. Downlink is lost in transmission.
4. Downlink is corrupted upon receival.
5. Gateway has no duty cycle allowance to transmit a response.

At the time of evaluation there was no definitive means to distinguish between a packet lost in transmission, corrupted or without duty cycle allowance. These three ineffective uplinks scenarios are referred to as *wasted downlinks*. Ambiguity in the RN2483 LoRa module specifications regarding the transmission response of *"mac_err"* in particular made it difficult to accurately analyse wasted downlinks. Online forum discussions suggest that any *"mac_err"* for an unacknowledged uplink should be treated as a corrupted downlink. In the upcoming "Results" chapter, *"mac_errs"* are found to significantly contribute to update transmission inefficiency and this work assumes any *mac_err* to be a corrupted downlink. Ineffective Response Uplinks are illustrated in Figures 3.5 and 3.6.

```
----------------------------------------------------------------------------
                       NODE             SERVER
----------------------------------------------------------------------------
                                 .
                                 .
                                 .

           [SEND UPLINK #N] ------>X
                     (PACKET LOST OVER THE AIR)

                                 .
                                 .
                                 .
```

Figure 3.5: Ineffective Response Lost Uplink

```
----------------------------------------------------------------------------
                       NODE            SERVER
----------------------------------------------------------------------------
                                 .
                                 .
                                 .
           [SEND UPLINK #N] ---------> [RECEIVE UPLINK #N]
     (NO RESPONSE RECEIVED)            (NO RESPONSE SCHEDULED)
                                 .
                                 .
                                 .
                                       [SCHEDULE DOWNLINK RESPONSE #M] ----+
                                                                           |
                                                                           |
         [SEND UPLINK #N+1] ---------> [RECEIVE UPLINK #N+1]               |
                               X<----- [SEND DOWNLINK #M]  <---------------+
                    (PACKET LOST OVER THE AIR)
                                 .
                                 .
                                 .
                                       [SCHEDULE DOWNLINK RESPONSE #M+1] --+
                                                                           |
                                                                           |
         [SEND UPLINK #N+2] ---------> [RECEIVE UPLINK #N+2]               |
    [RECEIVE DOWNLINK #M+1] <--------- [SEND DOWNLINK #M+1]  <-------------+
       (PACKET CORRUPTED)
                                 .
                                 .
                                 .
                                       [SCHEDULE DOWNLINK RESPONSE #M+2] --+
                                                                           |
                                                                           |
         [SEND UPLINK #N+2] ---------> [RECEIVE UPLINK #N+2]               |
                                   X-- [SEND DOWNLINK #M+2]  <-------------+
                      (NO DUTY CYCLE ALLOWANCE)
                                 .
                                 .
                                 .
```

Figure 3.6: Ineffective Response Uplinks

3.3 Protocol 1: Stop and Wait ARQ

Stop and Wait ARQ, illustrated in Figure 3.7, is a method in telecommunications to send information between two devices that ensures information is not lost due to dropped packets and that packets are received in the correct order. After sending each frame, the sender doesn't send any further frames until it receives an acknowledgement of receipt. After receiving a valid frame, the receiver sends an acknowledgement. Typically, if the acknowledgement does not reach the sender before a certain time, known as the timeout, the sender resends the same frame again.

```
------------------------------------------------------------------------------
                   RECEIVER             SENDER
------------------------------------------------------------------------------


                             <--------- [SEND FRAME #0]
              [ACK FRAME #0] --------->

                                X<----- [SEND FRAME #1]
                                              +
                                              |
                                              | (TIMEOUT)
                                              |
                                              +
                             <--------- [RESEND FRAME #1]
```

Figure 3.7: Conventional Stop and Wait ARQ Sequence Diagram

3.3.1 Implementation

Due to the nature of LoRa Class A bi-directional communication, the Stop and Wait ARQ protocol implementation presented here will vary slightly from the conventional implementation described above.

There is no timeout in this implementation. The gateway can only send downlinks in response to uplinks, as such the gateway cannot simply resend the packet after a certain amount of time has passed.

Firmware transmissions operations are limited to port 2. It is assumed that port 1 is used by an application for normal sensing duties. LoRaWAN specifies ports 1 to 223 as application specific. The use of a separate port to distinguish between node operations reduces protocol overhead.

Each uplink and downlink is transmitted with a 4 bit operational code (OPCODE) and an incrementally increasing 12 bit sequence number. The operational code is used to distinguish between different message types and the sequence number is used for tracking uplink and downlink delivery. The message types and their corresponding structures are outlined in section 3.3.2.

The server prepares a firmware update for transmission by splitting the total update into smaller segments of size 48 bytes. When the update size is not divisible by 48 without a remainder, the remainder is sent as the final firmware update packet.

Each firmware carrying downlink carries an eight bit index to identify the firmware segment and to reassemble the complete firmware image.

The node transmits a request for each firmware update segment sequentially until it has received the final firmware segment. The server, upon receipt of a request for a firmware segment, prepares the segment and schedules it for transmission in response to the next uplink. The node continues to request the firmware segment until it receives it in one of its receive windows.

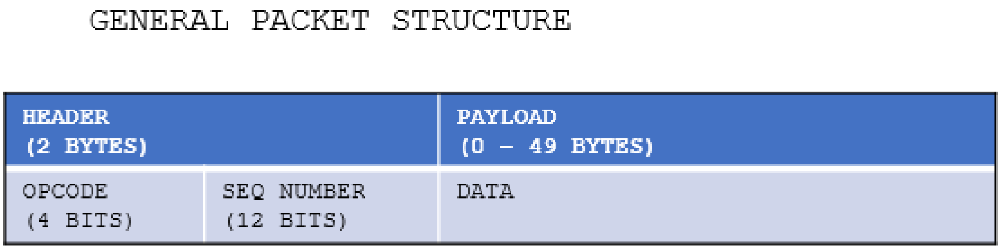
There is no explicit acknowledgement in this protocol. Upon receipt of the requested firmware update segment, the node will request the next sequential segment index. The acknowledgement is implied in the request for the next firmware segment. The server interprets the request for the next firmware segment as an acknowledgement for the previous segment and as a cue to prepare and schedule the next segment for transmission.

The last firmware update segment is recognised by the message opcode.

The process is illustrated in section 3.3.3

3.3.2 Packet Structure

Uplink and downlink payloads populate the "Frame Payload" field described in the LoRaWAN specifications. As discussed previously, the maximum permitted downlink message by the infrastructure used in this work is 51 bytes. Each message carries a 4 bit opcode to distinguish between message types. A 12 bit sequence number is included in each message to track message delivery. The opcode and sequence number together make up the packet header of size two bytes. The remaining 49 bytes are specific to the relevant uplink/downlink opcode (see Figure 3.8).



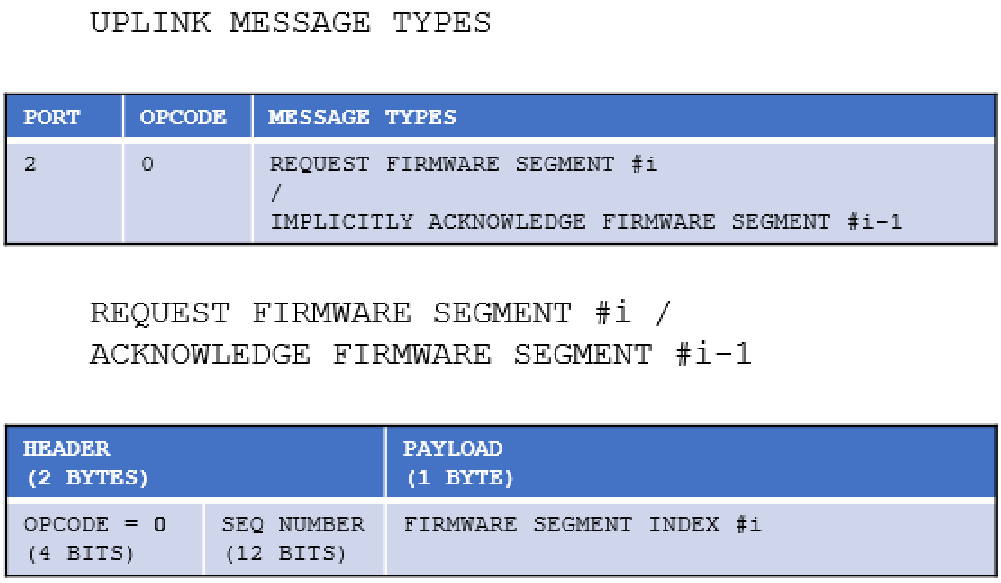
GENERAL PACKET STRUCTURE

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

Figure 3.8: Stop and Wait ARQ General Packet Structure

In this protocol only one uplink message type is required. The message, shown in Figure 3.9, is used to request a specific firmware segment index sequentially. For every update segment index after the first segment, it also implicitly implies an acknowledgement of receipt of the previous firmware segment index.

Two types of downlink messages are defined in this protocol. The first type, with an opcode of 0, is used for generic firmware update segment carrying downlinks. The second type, with an opcode of 1, is used to indicate that the firmware segment carried in this downlink is the final firmware segment to be transmitted. Both message types have in the payload an 8 bit field indicating the index of the firmware segment and a 1 to 48 byte field populated with the firmware segment.



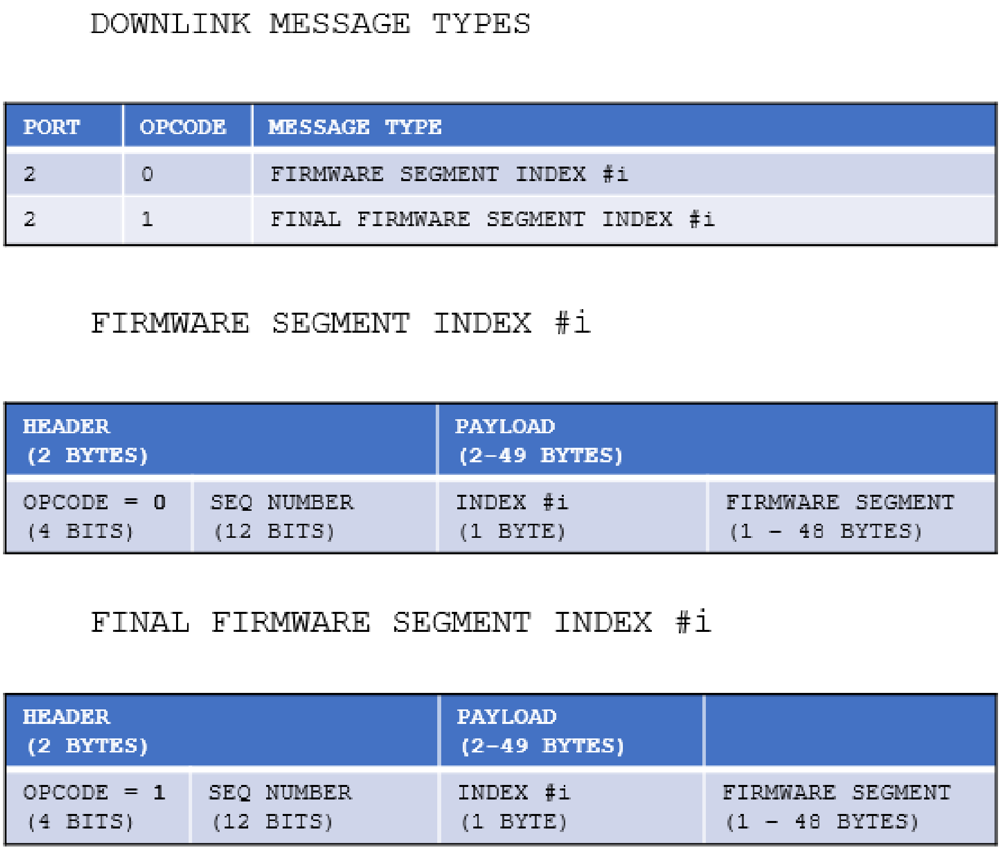
UPLINK MESSAGE TYPES

PORT	OPCODE	MESSAGE TYPES
2	0	REQUEST FIRMWARE SEGMENT #i / IMPLICITLY ACKNOWLEDGE FIRMWARE SEGMENT #i-1

REQUEST FIRMWARE SEGMENT #i /
ACKNOWLEDGE FIRMWARE SEGMENT #i-1

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

Figure 3.9: Stop and Wait ARQ Uplink Message Types and Structures



DOWNLINK MESSAGE TYPES

PORT	OPCODE	MESSAGE TYPE
2	0	FIRMWARE SEGMENT INDEX #i
2	1	FINAL FIRMWARE SEGMENT INDEX #i

FIRMWARE SEGMENT INDEX #i

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

FINAL FIRMWARE SEGMENT INDEX #i

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 3.10: Stop and Wait ARQ Downlink Message Types and Structures

3.3.3 Sequence Diagram

The firmware update transmission process for the Stop and Wait ARQ protocol is illustrated below in Figure 3.11.

```
  -----------------------------------------------------------------------------
                         NODE            SERVER
  -----------------------------------------------------------------------------
      [REQUEST FW SEGMENT #1] ---------> [RECEIVE FW SEGMENT #1 REQUEST]
       [NO RESPONSE RECEIVED]            [NO RESPONSE SCHEDULED]
   (INEFFECTIVE RESPONSE UPLINK)         [SCHEDULE FW SEGMENT #1] -----------+
                                                                             |
          (REQUEST AGAIN)                                                    |
      [REQUEST FW SEGMENT #1] ---------> [RECEIVE FW SEGMENT #1 REQUEST]      |
      [RECEIVE FW SEGMENT #1] <--------- [SEND FW SEGMENT #1] <--------------+
        (EFFECTIVE UPLINK)                   (OPCODE = 0)
                                         [SCHEDULE FW SEGMENT #1] -----------+
                                                                             |
          (REQUEST NEXT)                                                     |
      [REQUEST FW SEGMENT #2] ---------> [RECEIVE FW SEGMENT #2 REQUEST]      |
                                         [ACKNOWLEDGE #1 DELIVERY]            |
      [RECEIVE FW SEGMENT #1] <--------- [SEND FW SEGMENT #1] <--------------+
     (INEFFECTIVE DATA UPLINK)           [SCHEDULE FW SEGMENT #2] -----------+
                                                                             |
          (REQUEST AGAIN)                                                    |
      [REQUEST FW SEGMENT #2] ---------> [RECEIVE FW SEGMENT #2 REQUEST]      |
                                         [ACKNOWLEDGE #1 DELIVERY]            |
      [RECEIVE FW SEGMENT #2] <--------- [SEND FW SEGMENT #2] <--------------+
        (EFFECTIVE UPLINK)              |[SCHEDULE FW SEGMENT #1] -----------+
                                    .
                                    .
                                    .
      [REQUEST FW SEGMENT #i] ---------> [RECEIVE FW SEGMENT #i REQUEST]      |
                                         [ACKNOWLEDGE #i-1 DELIVERY]          |
       [NO RESPONSE RECEIVED]    X<----- [SEND FW SEGMENT #i] <--------------+
                      (PACKET LOST OVER THE AIR)
   (INEFFECTIVE RESPONSE UPLINK)         [SCHEDULE FW SEGMENT #i] -----------+
                                                                             |
          (REQUEST AGAIN)                                                    |
      [REQUEST FW SEGMENT #i] ---------> [RECEIVE FW SEGMENT #i REQUEST]      |
                                         [ACKNOWLEDGE #i-1 DELIVERY]          |
      [RECEIVE FW SEGMENT #i] <--------- [SEND FW SEGMENT #i] <--------------+
        (EFFECTIVE UPLINK)               [SCHEDULE FW SEGMENT #i] -----------+
                                    .
                                    .
                                    .
                                         [SCHEDULE FINAL FW SEGMENT #N] -----+
                                                                             |
      [REQUEST FW SEGMENT #N] ---------> [RECEIVE FW SEGMENT #N REQUEST]      |
                                         [ACKNOWLEDGE #N-1 DELIVERY]          |
[RECEIVE FINAL FW SEGMENT #N] <--------- [SEND FINAL FW SEGMENT #N] <--------+
        (EFFECTIVE UPLINK)                   (OPCODE = 1)
```

Figure 3.11: Stop and Wait ARQ Sequence Diagram

3.3.4 Protocol Efficiency

Considering the unique, out of step, bi-directional communication between the node and gateway, this protocol has many inherent inefficiencies. As discussed in the design considerations, the server can never reply immediately to the most recent information obtained from an uplink but can only reply in response to the next uplink.

Now, consider the sequence diagram illustrated in Figure 3.11. The first request for a firmware update will always be classified as an *Ineffective Response Uplink* as the server cannot respond immediately to the request, but will schedule the firmware segment to be sent in response to the next uplink. The node will subsequently request the first firmware segment again. If the node receives the firmware segment in its receive windows, the uplink is classified as an effective uplink.

The server, however, is unaware of whether the firmware segment successfully reached the node, and as the last uplink it received was a request for the first firmware segment, the server has scheduled the firmware segment for re-transmission. The node, now having received the first firmware segment, requests the second firmware segment. The server receives the request, acknowledges the delivery of the first segment, and schedules the second segment to be transmitted in response to the next uplink. The first firmware segment has already been queued for re-transmission and is sent in response to the request for the second segment. Assuming that the downlink reaches the node, this is classified as an *Ineffective Data Uplink* inherent to the protocol, as the first firmware segment has already been received. This process is outlined in the first three sequences of Figure 3.11.

Therefore, assuming no packets are lost in transmission or corrupted, and that the gateway's duty cycle allowance is never exceeded, for each firmware update segment there is inherently one *Ineffective Uplink* for every *Effective Uplink*. The first firmware request uplink is always an *Ineffective Response Uplink*, and each subsequent request will have at least one *Ineffective Uplink*: an *Ineffective Response Uplink* if the downlink goes missing, corrupted or the gateway has no duty cycle allowance; or an *Ineffective Data Uplink* if the response contains a firmware segment that has already been received and was scheduled for the previous uplink.

3.3.5 Node State Machine

The node state machine is outlined in Figure 3.12. In state 1 the node requests a specific firmware segment index. After transmission the node waits to receive a response in its open receive windows. If no response is received it re-transmits the request. If a response is received the node extracts the firmware segment index contained in the response payload.

If the response payload opcode is 0, the node checks the received response firmware segment index against the segment index it requested. If the requested firmware segment index is not equal to the received index, the node requests the index again.

If the requested firmware segment index is equal to the received index, the node transmits a request for the next firmware index, implicitly implying to the server an acknowledgement of the received firmware segment index.

If the firmware carrying downlink has an opcode of 1 it contains the final firmware segment and the complete firmware segment has been received.

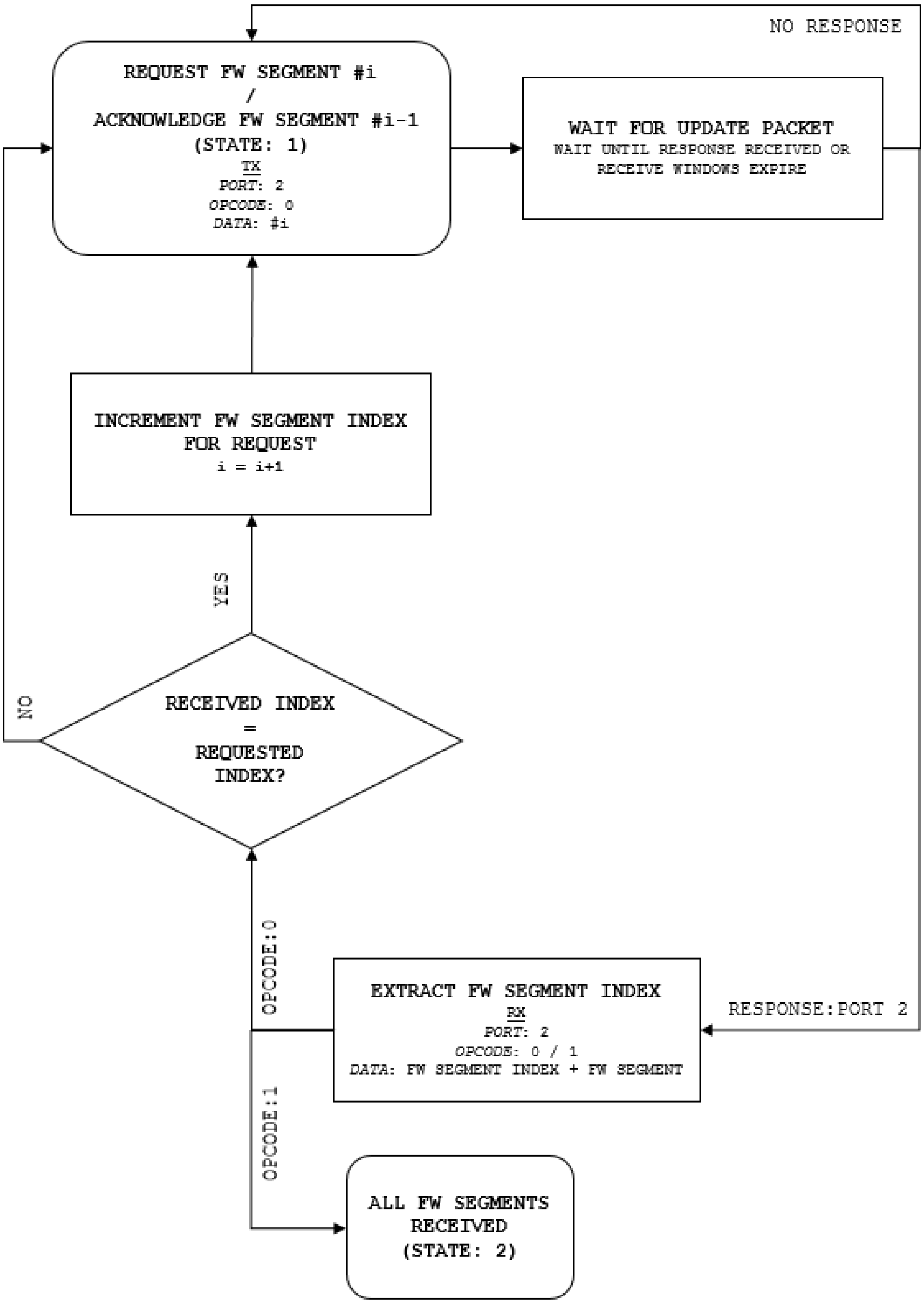


Figure 3.12: Stop and Wait ARQ Node State Machine

3.3.6 Server State Machine

The server state machine is outlined in Figure 3.13. The server state machine begins with the gateway continuously listening for uplinks from the node. When the gateway receives an uplink, it checks if it has a downlink response scheduled. If a response is scheduled it transmits the response to the node, subsequently forwarding the uplink to the server. If no response is scheduled the uplink is immediately forwarded to the server.

When the server receives the uplink it extracts the firmware segment index *i* embedded in the request. If the requested index, *i*, is any index other than the first index, the server acknowledges the successful delivery of firmware segment index *i-1*. The server then prepares firmware segment index *i*.

If index *i* is the final firmware segment to be transmitted, the server assigns the firmware carrying response an opcode of 1. If there are more firmware segments to be transmitted an opcode of 0 is assigned.

This process is repeated until all firmware segments have been transmitted.

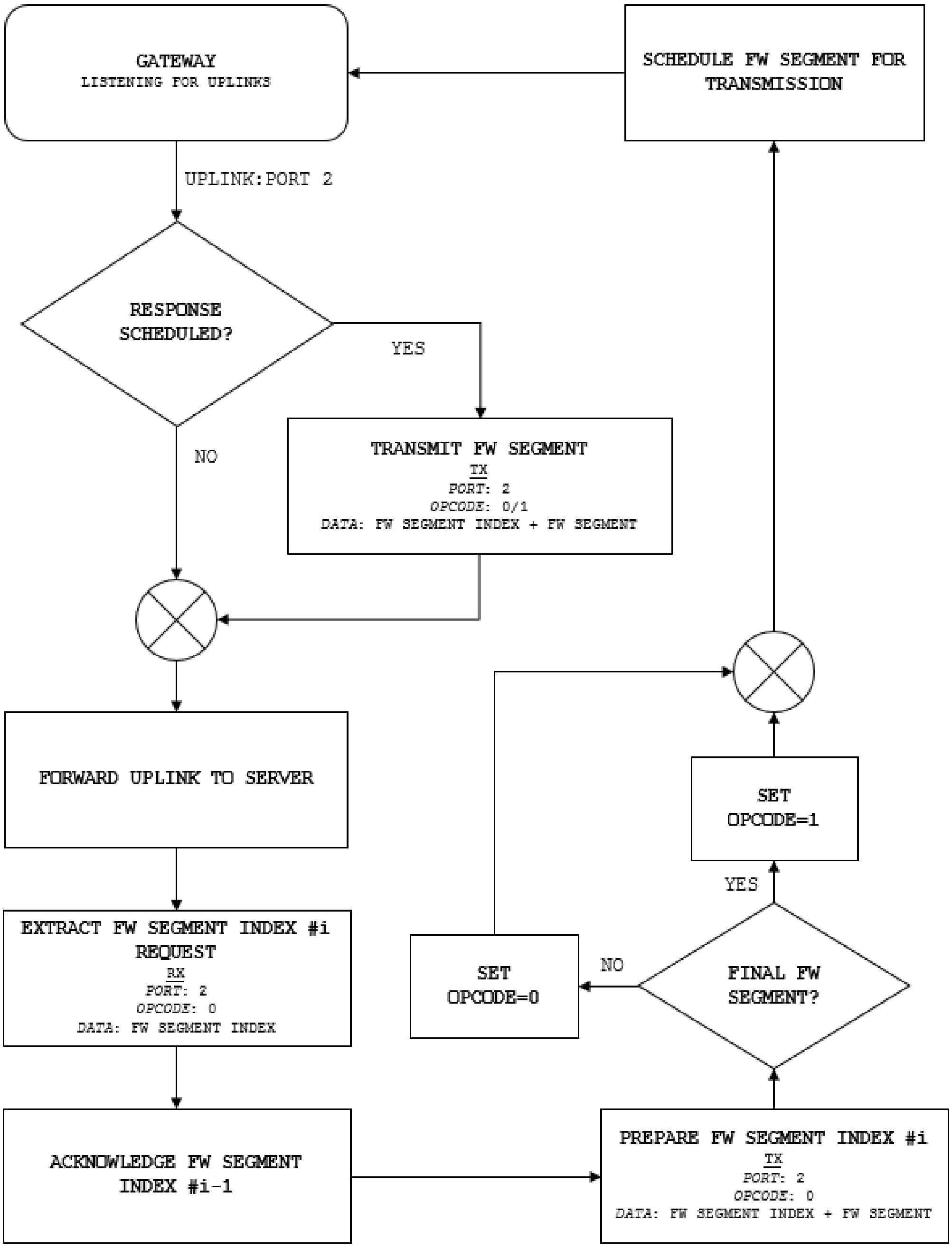


Figure 3.13: Stop and Wait ARQ Server State Machine

3.4 Protocol 2: Piggybacked Selective Repeat ARQ

Selective Repeat ARQ is another method in telecommunications to send information between two devices that ensures information is not lost due to dropped packets with disregard to the order the packets are received.

The sender sends a number of frames specified by a window size even without the need to wait for individual acknowledgements from the receiver. The receiver may selectively reject a single frame, which may be re-transmitted alone. The receiver accepts out-of-order frames and buffers them. The sender individually re-transmits frames that have timed out.

In two communications, piggybacking is the technique of temporarily delaying an acknowledgement so that it can be hooked with the next outgoing data frame.

3.4.1 Implementation

Again, due to the nature of LoRa Class A bi-directional communication, the implementation of a Selective Repeat ARQ protocol presented here will vary slightly from the conventional implementation described above.

Again, for the same reasons as the Stop and Wait ARQ protocol, there is no timeout in this implementation.

Firmware update operations are limited to port 2. It is assumed that port 1 is used by an application for normal sensing duties. LoRaWAN specifies ports 1 to 223 as application specific. The use of a separate port to distinguish between node operations reduces protocol overhead.

Each uplink and downlink is transmitted with a 4 bit operational code (OPCODE) and an incrementally increasing 12 bit sequence number. The operational code is used to distinguish between different message types and the sequence number is used for tracking uplink and downlink delivery. The message types and their corresponding structures are outlined in section 3.4.2.

The server prepares a firmware update for transmission by splitting the total update into smaller segments of size 48 bytes. When the update size is not divisible by 48 without a remainder, the remainder is sent as the final firmware update packet.

Each firmware carrying downlink carries an 8 bit index to identify the firmware segment and to reassemble the complete firmware image.

In this protocol the node transmits a generic request for any firmware update segment. If the node receives a firmware update segment in its receive windows, it piggybacks the acknowledgement of receipt on to the next firmware update segment request uplink.

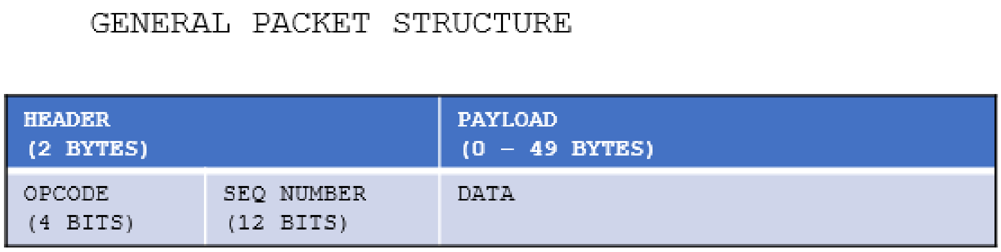
The server and node do not care in which order segments are transmitted and received. Therefore, the server upon receipt of a firmware update segment request, schedules the next unacknowledged firmware segment for transmission. After all firmware segments have been transmitted, the server re-transmits all unacknowledged segments. This process continues until all segments have been acknowledged.

The last firmware update segment is recognised by the message opcode.

The process is illustrated in section 3.4.3.

3.4.2 Packet Structure

Uplink and downlink payloads populate the "Frame Payload" field described in the LoRaWAN specifications. The maximum permitted downlink by the infrastructure used is 51 bytes. Each message carries a 4 bit opcode to distinguish between message types. A 12 bit sequence number is included in each message to track message delivery. The opcode and sequence number together make up the packet header. Therefore, the packet header is two bytes in size. The remaining 49 bytes are specific to the relevant uplink/downlink opcode (see Figure 3.14).

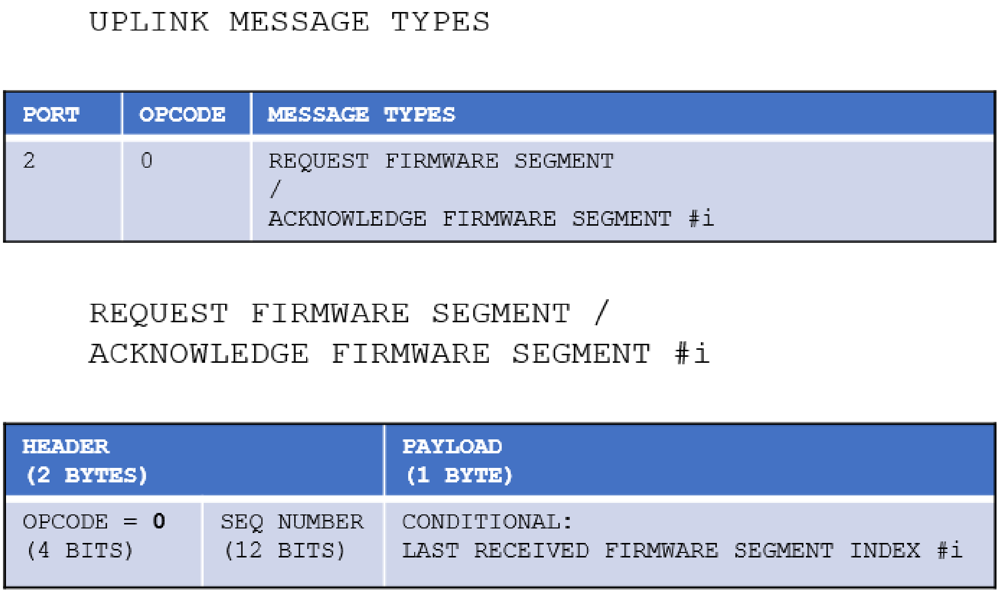


GENERAL PACKET STRUCTURE

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

Figure 3.14: Piggybacked Selective Repeat ARQ General Packet Structure

Only one uplink message type is required in this protocol. The message, shown in Figure 3.15, is used to request the next undelivered firmware update segment and acknowledge the most recent received firmware update segment.



UPLINK MESSAGE TYPES

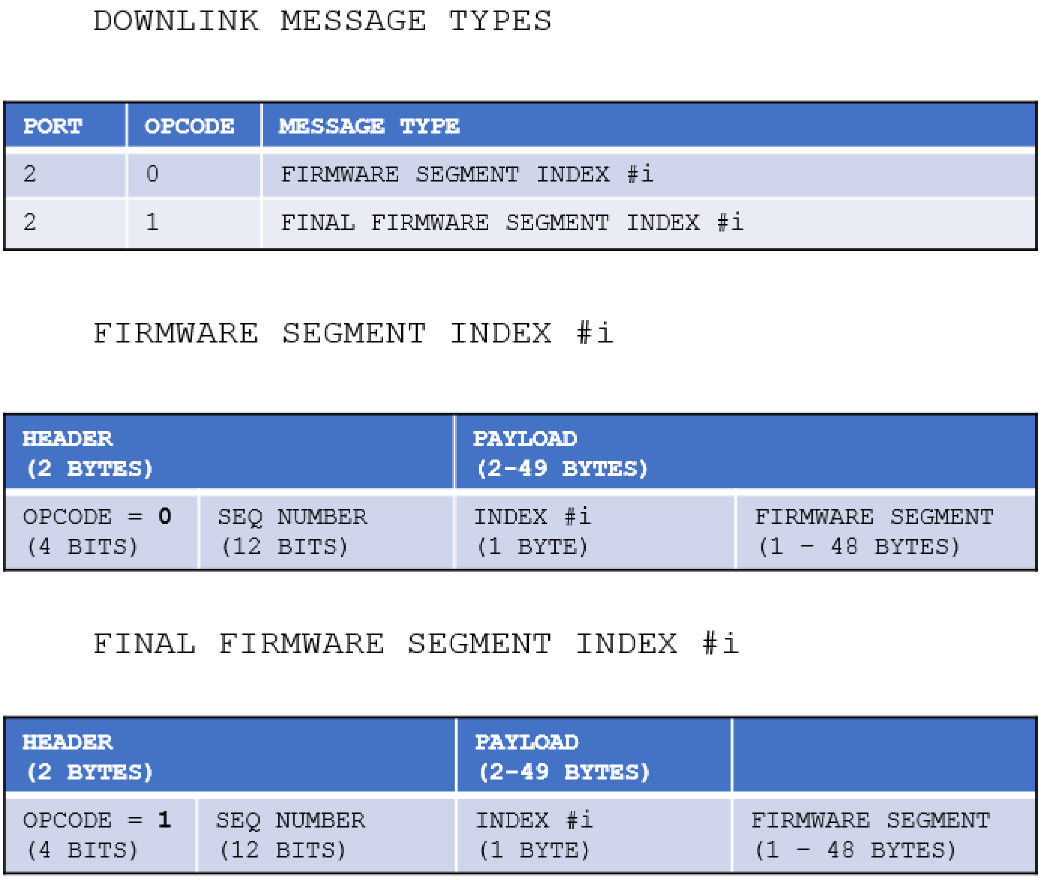
PORT	OPCODE	MESSAGE TYPES
2	0	REQUEST FIRMWARE SEGMENT / ACKNOWLEDGE FIRMWARE SEGMENT #i

REQUEST FIRMWARE SEGMENT /
ACKNOWLEDGE FIRMWARE SEGMENT #i

HEADER (2 BYTES)		PAYLOAD (1 BYTE)
OPCODE = **0** (4 BITS)	SEQ NUMBER (12 BITS)	CONDITIONAL: LAST RECEIVED FIRMWARE SEGMENT INDEX #i

Figure 3.15: Piggybacked Selective Repeat ARQ Uplink Message Types and Structures

The two downlink message types used in this protocol, shown in Figure 3.16, are identical to those defined for the Stop and Wait ARQ protocol described in section 3.3.2.



DOWNLINK MESSAGE TYPES

PORT	OPCODE	MESSAGE TYPE
2	0	FIRMWARE SEGMENT INDEX #i
2	1	FINAL FIRMWARE SEGMENT INDEX #i

FIRMWARE SEGMENT INDEX #i

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

FINAL FIRMWARE SEGMENT INDEX #i

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 3.16: Piggybacked Selective Repeat ARQ Downlink Message Types and Structures

3.4.3 Sequence Diagram

The firmware update transmission process for the Piggybacked Selective Repeat ARQ protocol is illustrated below in Figure 3.17.

```
  ----------------------------------------------------------------------------
                         NODE            SERVER
  ----------------------------------------------------------------------------
         [REQUEST FW SEGMENT] ---------> [RECEIVE FW SEGMENT REQUEST]
       [NO RESPONSE RECEIVED]            [NO RESPONSE SCHEDULED]
   (INEFFECTIVE RESPONSE UPLINK)         [SCHEDULE FW SEGMENT #1] -----------+
                                                                             |
                                                                             |
         [REQUEST FW SEGMENT] ---------> [RECEIVE FW SEGMENT REQUEST]        |
      [RECEIVE FW SEGMENT #1] <--------- [SEND FW SEGMENT #1] <--------------+
        (EFFECTIVE UPLINK)                   (OPCODE = 0)
                                         [SCHEDULE FW SEGMENT #2] -----------+
                                                                             |
                                                                             |
  [REQUEST FW SEGMENT/ACK #1] ---------> [RECEIVE FW SEGMENT REQUEST]        |
                                         [ACKNOWLEDGE #1 DELIVERY]           |
      [RECEIVE FW SEGMENT #2] <--------- [SEND FW SEGMENT #2] <--------------+
        (EFFECTIVE UPLINK)               [SCHEDULE FW SEGMENT #3] -----------+
                                                                             |
                                   .
                                   .
                                   .
  [REQUEST FW SEGMENT/ACK #i] ---------> [RECEIVE FW SEGMENT REQUEST]        |
                                         [ACKNOWLEDGE #i DELIVERY]           |
       [NO RESPONSE RECEIVED]    X<----- [SEND FW SEGMENT #j] <--------------+
                      (PACKET LOST OVER THE AIR)
   (INEFFECTIVE RESPONSE UPLINK)         [SCHEDULE FW SEGMENT #j+1] ---------+
                                   .
                                   .
                                   .
                                                                             |
      [RECEIVE FW SEGMENT #N] <--------- [SEND FW SEGMENT #N] <--------------+
        (EFFECTIVE UPLINK)               (#N = FINAL UN-TRANSMITTED SEGMENT)
                                         [RESCHEDULE UNACK FW SEGMENT #j] ---+
                                                                             |
                                                                             |
  [REQUEST FW SEGMENT/ACK #N] ---------> [RECEIVE FW SEGMENT REQUEST]        |
                                         [ACKNOWLEDGE #N DELIVERY]           |
      [RECEIVE FW SEGMENT #j] <--------- [SEND FW SEGMENT #j] <--------------+
        (EFFECTIVE UPLINK)
                                   .
                                   .
                                   .
                                         [SCHEDULE FW SEGMENT #Z] -----------+
                                                                             |
                                                                             |
  [REQUEST FW SEGMENT/ACK #Y] ---------> [RECEIVE FW SEGMENT REQUEST]        |
                                         [ACKNOWLEDGE #Y DELIVERY]           |
      [RECEIVE FW SEGMENT #Z] <--------- [SEND FW SEGMENT #Z] <--------------+
        (EFFECTIVE UPLINK)                   (OPCODE = 0)
                                         [SCHEDULE FINAL FW SEGMENT #Z] -----+
                                         (Z IS THE ONLY REMAINING            |
                                           UNACKNOWLEDGED SEGMENT)           |
                                                                             |
  [REQUEST FW SEGMENT/ACK #Z] ---------> [RECEIVE FW SEGMENT]                |
                                         [ACKNOWLEDGE #Z DELIVERY]           |
[RECEIVE FINAL FW SEGMENT #Z] <--------- [SEND FINAL FW SEGMENT #Z] <--------+
  (INEFFECTIVE DATA UPLINK)                   (OPCODE = 1)
```

Figure 3.17: Piggybacked Selective Repeat ARQ Sequence Diagram

3.4.4 Protocol Efficiency

This protocol, although still with some inherent inefficiencies, has many optimisations over the Stop and Wait ARQ protocol. The Piggybacked Selective Repeat ARQ protocol comes with just two inherent inefficiencies. Referring back to the sequence diagram shown in Figure 3.17, these inefficiencies can be seen at the very beginning and end of the sequence.

First, identical to the Stop and Wait ARQ protocol, the initial uplink is always an *Ineffective Response Uplink* as the server cannot respond immediately to the request, but schedules its response to be transmitted in response to the next uplink.

The second inherent inefficiency is always the duplicate transmission of the last firmware segment packet as the server and node are always one step out of sequence. Imagine a firmware update divided into 100 segments for transmission. Imagine that no packets have been dropped in the process and the only two remaining packets are segment indices 99 and 100. The node has received segment index 98 and requests the next firmware segment from the server whilst also acknowledging the receipt of segment index 98. The server has already scheduled segment index 99 for delivery and this is sent in response to the firmware request. The server now schedules segment index 100 to be transmitted in response to the next request.

At this stage the server does not know if node has successfully received segment index 99, therefore segment index 100 has be scheduled with an opcode of 0.

The node receives index 99 and in the subsequent uplink requests the next firmware segment and acknowledges segment index 99. The server responds with the already scheduled downlink carrying firmware segment index 100 with opcode 0. The server acknowledges the successful delivery of index 99 but is unaware if the most recent downlink carrying index 100 has reached the node and therefore reschedules firmware segment 100 again. This time, however, the server knows it is the final segment and schedules it with an opcode of 1, indicating to the node that this is the final firmware segment.

Therefore, the node always receives the final firmware segment index twice, an inherent *Ineffective Data Uplink* to the protocol.

This protocol is much more efficient than the Stop and Wait ARQ. Assuming that no packets are dropped in the process, this protocol produces just two *Ineffective Uplinks*, whereas the Stop and Wait ARQ protocol produces at least one *Ineffective Uplink* for every *Effective Uplink*. Essentially the Piggybacked Selective Repeat ARQ is $\frac{2N}{N+2}$ times more efficient than the Stop and Wait ARQ, where N is the number of downlink packets required to send the complete firmware update. This tends to twice as efficient for large values of N, and as firmware updates are typically in the order of several kilobytes, this is an apt approximation.

3.4.5 Node State Machine

The node state machine is outlined in Figure 3.18. In state 1 the node requests a firmware update segment. After transmission the node waits to receive a response in its open receive windows. If no response is received it re-transmits the request.

If a response is received, the node extracts the firmware segment index from the downlink. In the next uplink, the node will request a firmware segment index again and piggyback the last received firmware index segment as an acknowledgement of delivery.

If the firmware carrying downlink has an opcode of 1 it contains the final firmware segment and the complete firmware segment has been received.

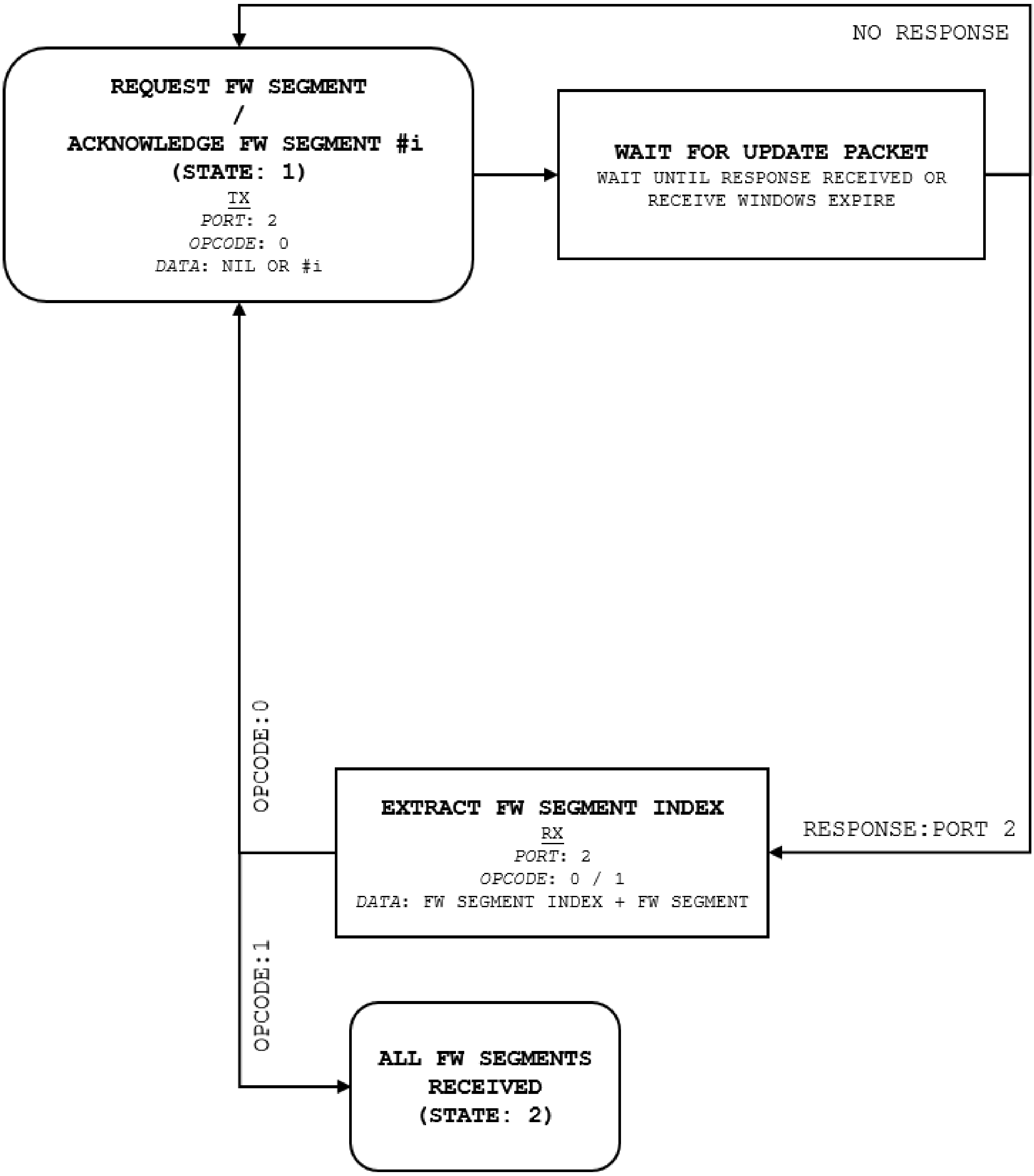


Figure 3.18: Piggybacked Selective Repeat ARQ Node State Machine

3.4.6 Server State Machine

The server state machine is outlined in Figure 3.19. The server gateway continuously listens for uplinks from the node. When the gateway receives an uplink, it checks if it has a downlink response scheduled. If a response is scheduled it transmits the response to the node and subsequently forwards the uplink to the server. If no response is scheduled the uplink is immediately forwarded to the server.

When the server receives the uplink request for a firmware segment it first checks if the request has an acknowledgement piggybacked onto it. If no, the firmware prepares the next undelivered firmware segment for transmission. If yes, the server acknowledges the successful delivery of that firmware segment index and subsequently prepares the next firmware segment for transmission.

Finally, the server checks if the firmware segment to be transmitted is the final remaining unacknowledged firmware index. If yes, the downlink is assigned an opcode of 1. If no, the downlink is assigned an opcode of 0. The server then schedules the firmware carrying downlink for transmission in response to the next uplink.

This process is repeated until all firmware segments have been acknowledged.

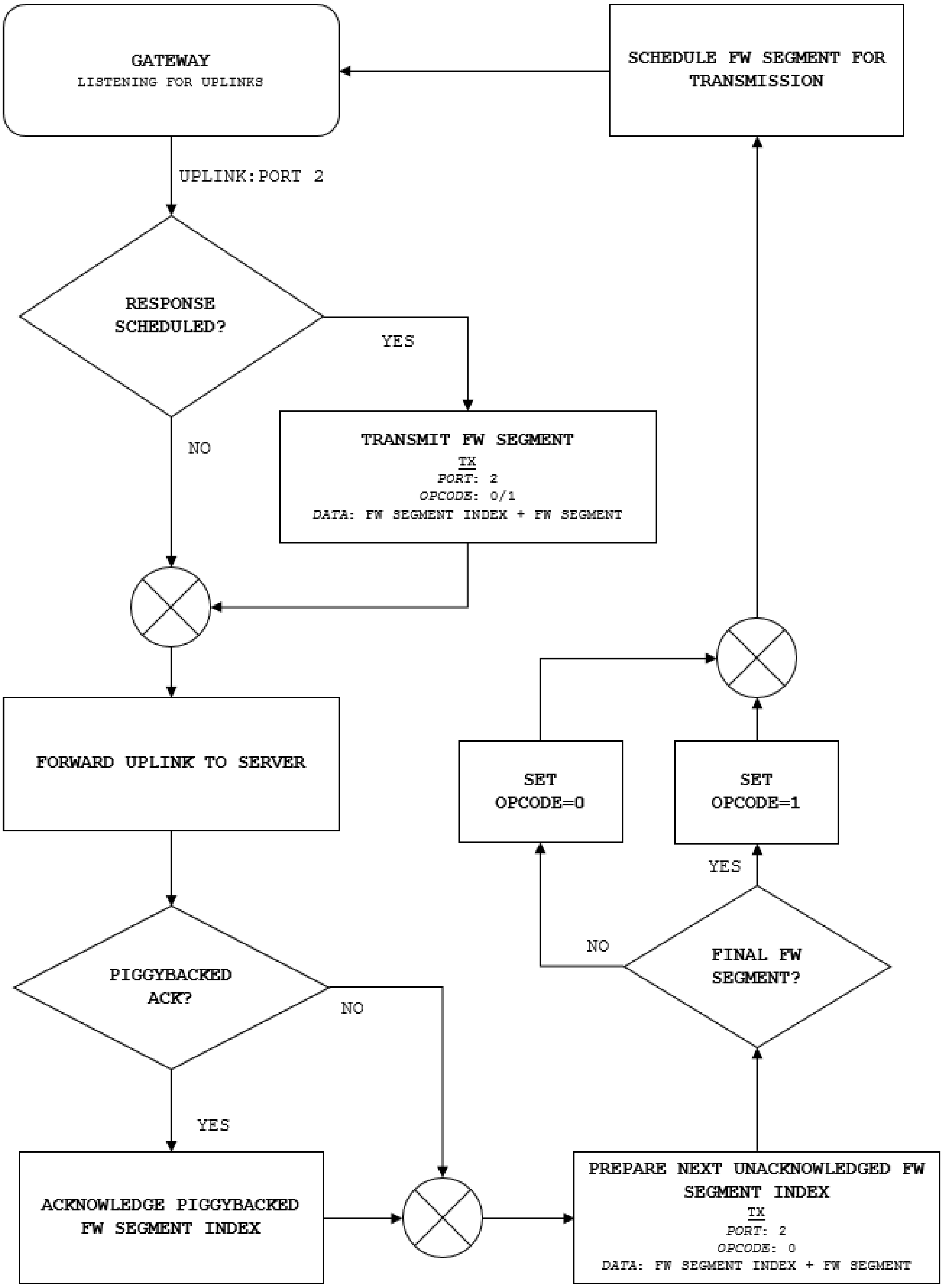


Figure 3.19: Piggybacked Selective Repeat ARQ Server State Machine

4 Experimental Evaluation

4.1 Experimental Setup

4.1.1 Sensor Node

The sensor node used in this work consisted of an Arduino Uno microcontroller connected via jumper wires to the RN2483 LoRa module from Microchip, shown in Figure 4.1. The RN2483-Arduino-Library [61], openly available on GitHub, was refactored to program the Arduino microcontroller as a Class A type device featuring this work's firmware transmission protocols. The node had three uplink channels configured centred on 868.1, 868.3 and 868.5 MHz, each with a bandwidth of 125 kHz and duty cycle allowance of 0.33%. The node joined the LoRa network using Over-The-Air activation. Upon activation the Network Server reconfigured the second receive window to spreading factor 10 on 869.525 MHz.

4.1.2 Infrastructure

Pervasive Nation infrastructure facilitated the work by providing the LoRa gateway infrastructure. Pervasive Nation is Ireland's Internet of Things testbed, operated by CONNECT, the Science Foundation Ireland Research Centre for Networks headquartered at Trinity College Dublin, the University of Dublin. The Pervasive Nation infrastructure uses the latest off-the-shelf LPWAN and software-defined radio

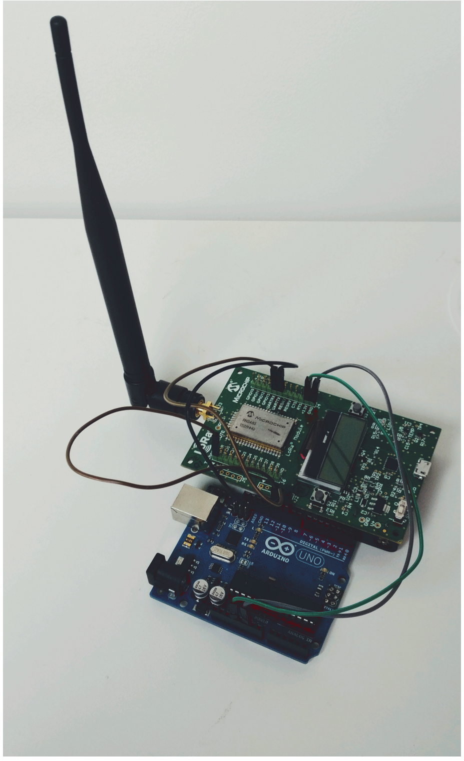


Figure 4.1: LoRa Sensor Node

technologies [62]. Pervasive Nation is a member of the LoRa Alliance and they provide all the necessary infrastructure for any LoRa equipped IoT node to join their network: gateways, network servers and application enabling platforms. Pervasive Nation has eight deployed gateways in the Dublin area, with more deployed throughout Ireland (see Figure 4.2).

Orbiwise, a Swiss-based IoT firm, supply the network server management software for Pervasive Nation [63]. The Orbiwise software interfaces between the gateways and application enabling platforms.

4.1.3 Web Service Application

The backend of this experimental setup is supported by a Node.js web service running on a Raspberry Pi 3. This web service application is responsible for the majority of the computing and processing tasks of the firmware transmission protocols. From henceforth the web service application will be referred to as the

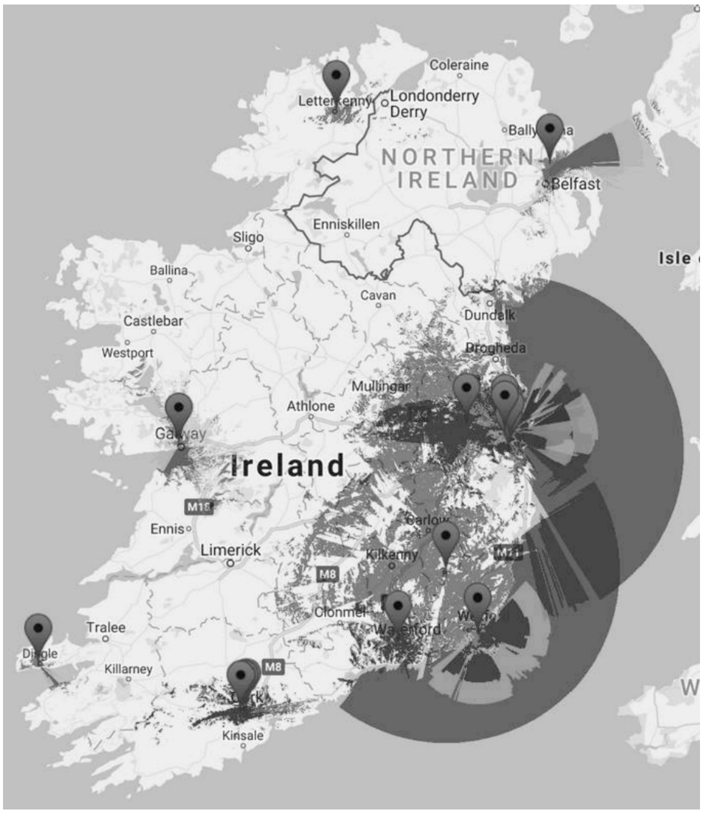


Figure 4.2: Pervasive Nation Coverage

application server. The application server interfaces with the Data Access Sub-System (DASS) of the Orbiwise network server via the Orbiwise DASS REST API. By registering the application server on the DASS, the DASS and application server can make unsolicited PUT/POST requests to each other. Figure 4.3 illustrates the Orbiwise solution interfaces.

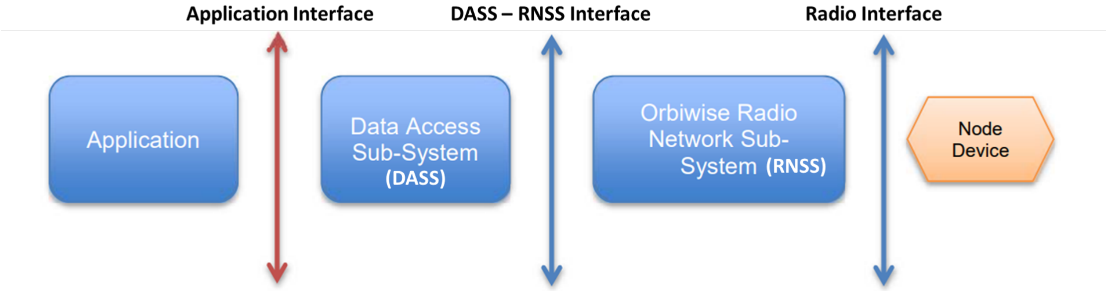


Figure 4.3: Orbiwise Solution Interfaces Overview

Upon registration and a running application, the DASS will immediately POST all new uplinks from the sensor node to the application server. The application server may also POST downlink payloads to the DASS for it to transmit them to the sensor node on the next opportunity. When a downlink transmission has completed (successfully or unsuccessfully) the DASS will inform the application server with the downlink delivery status (see Figure 4.4).

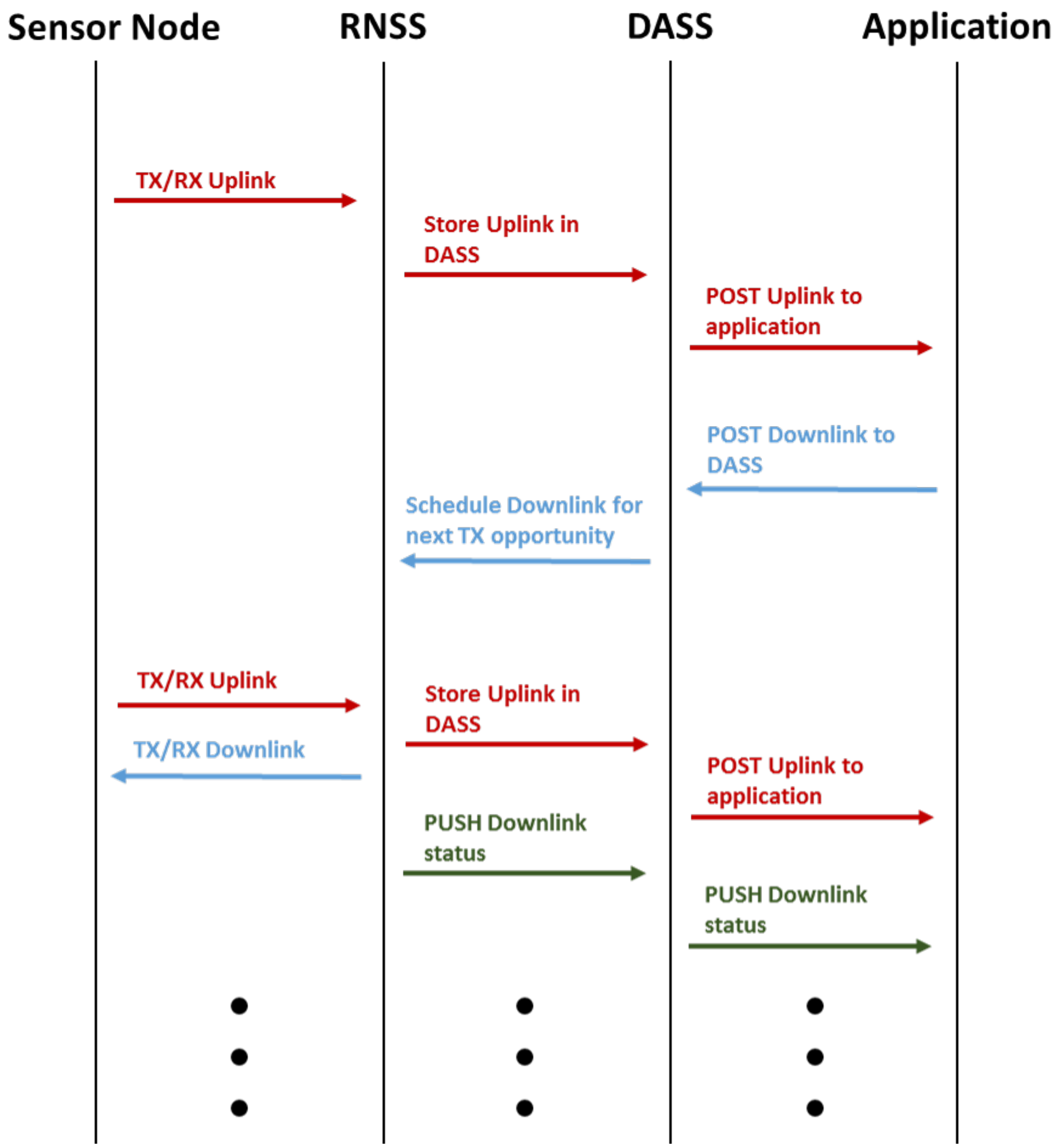


Figure 4.4: Uplink / Downlink Packet Life-Cycle

4.1.4 Experiment Environment

Figure 4.5 shows the sensor node's deployment in the Dublin City docklands area and the location of the six nearest LoRa gateways in relation to the node. The node remained stationary in an indoor location over the course of the experiments. The node was connected to a power supply and its output was monitored via the Arduino IDE Serial Monitor. This work had no control over which particular gateway received each uplink and downlink packet. Table 4.1 lists the distance of each gateway to the node.

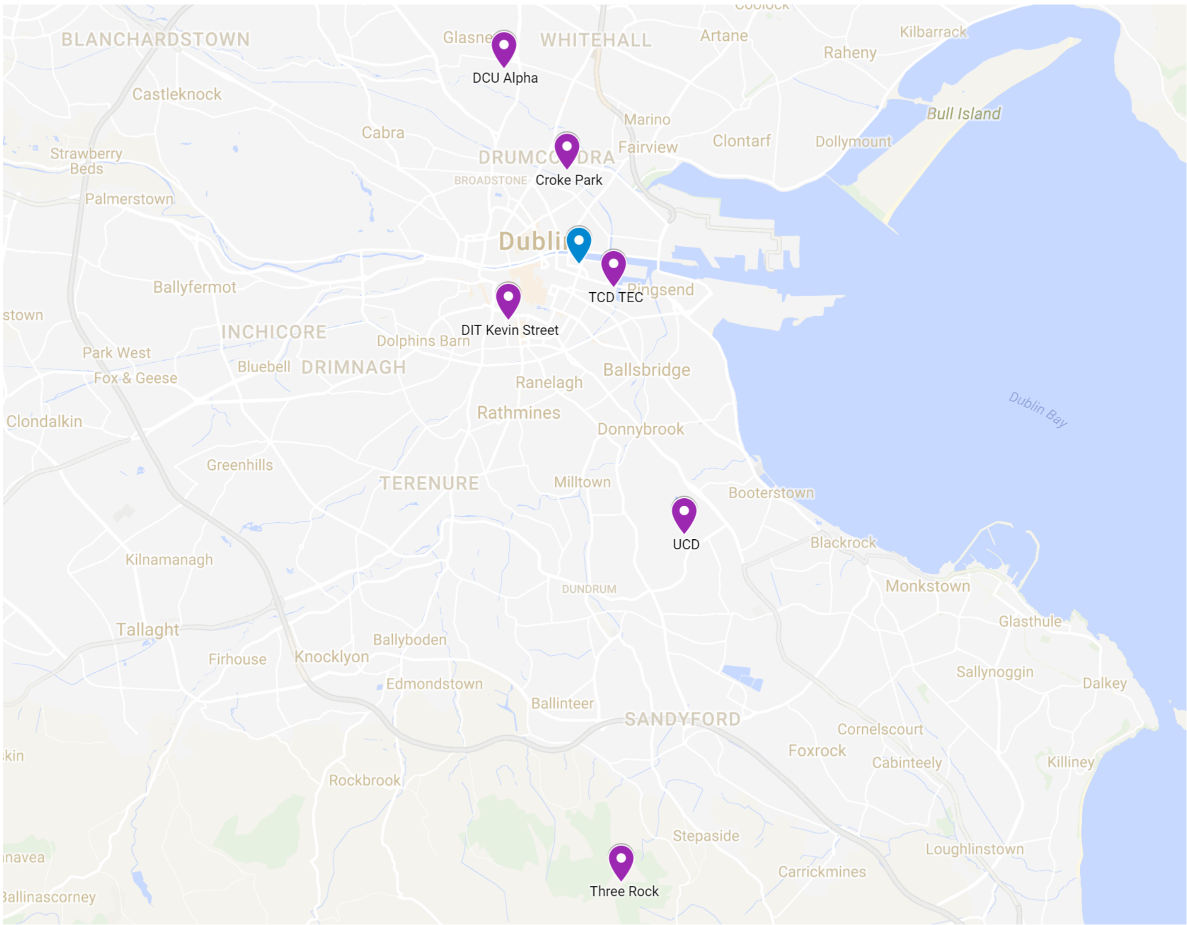


Figure 4.5: Node Deployment and Nearest Gateways

4.2 Evaluation Scenarios

4.2.1 Stop and Wait ARQ

The Stop and Wait ARQ protocol was only tested for illustrative purposes due to the high inherent inefficiencies associated with the protocol and its unsuitability for LoRa communications. Seven experiments were ran over the course of two days. Each experiment used SF7, the fastest data rate, and an update size of 5000 bytes. The node attempted to transmit an uplink every two seconds duty cycle permitting.

4.2.2 Piggybacked Selective Repeat ARQ

The Piggybacked Selective Repeat ARQ Protocol was extensively tested over the course of one week. Each spreading factor, i.e. SF7 through to SF 12, was tested 8 times with an update size of 5000 bytes. Again, the node attempted to transmit an uplink every two seconds duty cycle permitting.

Table 4.1: Gateways

Gateway	Distance from node (km)
TEC	0.75
Croke Park	1.65
DIT Kevin St.	1.65
DCU Alpha	3.77
UCD	5.25
Three Rock	11.2

5 Results

Presented below are the results of the firmware transmission experiments. Note that, unless otherwise stated, the results presented are the results for the Piggybacked Selective Repeat ARQ protocol. Hence, in the graphs and tables to follow, a result presented as "SF7" would indicate the result for the Piggybacked Selective Repeat ARQ protocol using spreading factor 7. Results for the Stop and Wait ARQ protocol would be expressed explicitly as "Stop and Wait SF7".

5.1 Time

The Piggybacked Selective Repeat ARQ protocol was tested 8 times with a 5000 byte update using spreading factors 7 to 12. The Selective Repeat ARQ protocol was tested 7 times with a 5000 byte update using just spreading factor 7. The experimental results are shown in Figure 5.1.

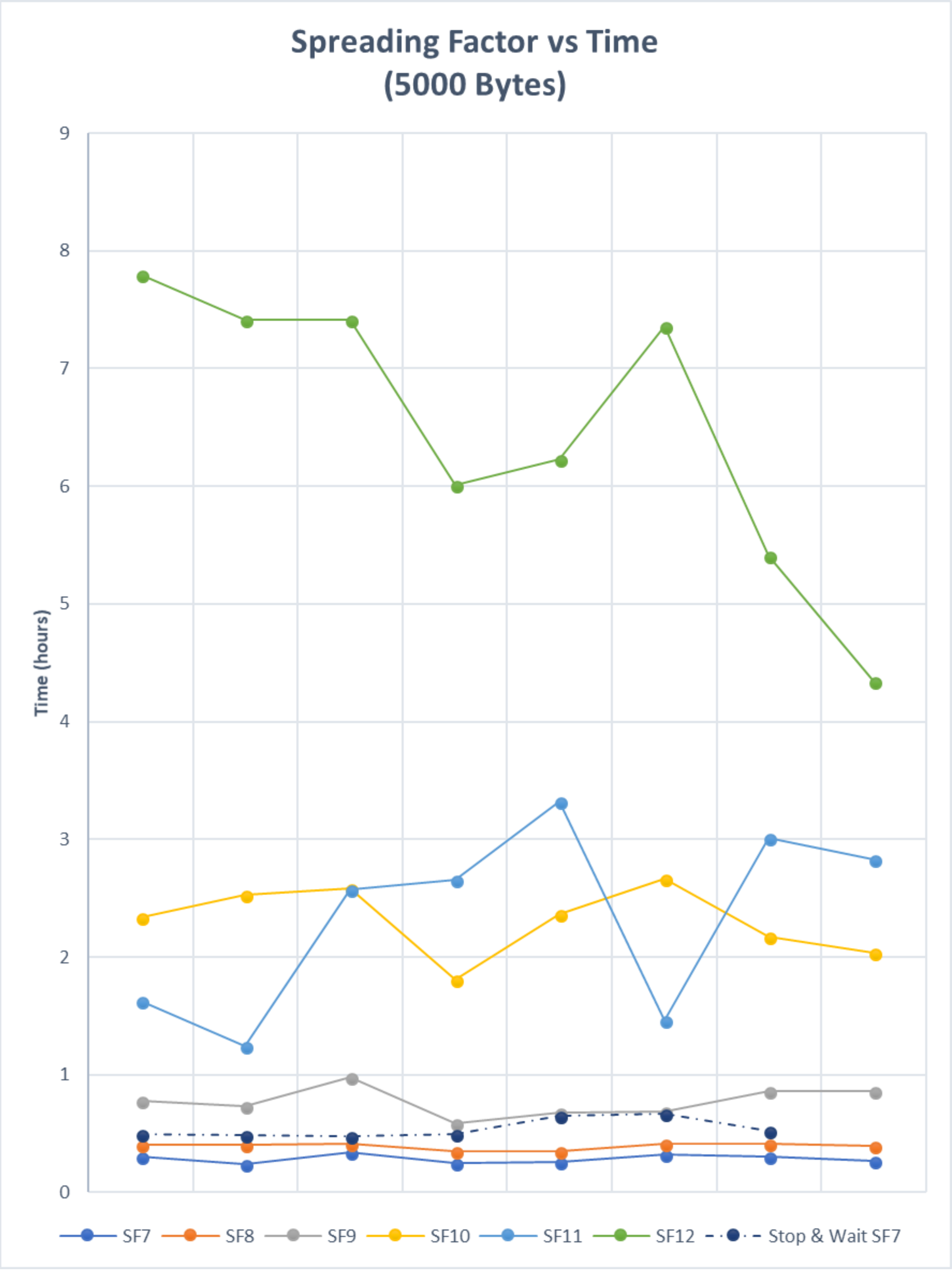


Figure 5.1: Firmware Transmission Times

There is a general increase in the firmware transmission time with increasing spreading factor. This is expected as data rate decreases with increasing spreading factor. The Stop and Wait ARQ transmission time at spreading factor 7 is close to twice the transmission time of the corresponding Piggybacked Selective Repeat ARQ. This is also expected as it was previously shown that as the number of firmware packets increases the efficiency of the Piggybacked Selective Repeat ARQ tends to twice that of Stop and Wait ARQ.

Table 5.1 presents the mean firmware transmission times. Again, the expected increase in firmware transmission time with spreading factor is found when comparing the mean times. Relatively low standard deviations for the lower spreading factors (SF 7, 8, 9, 10) indicate that the firmware transmission time remained relatively constant throughout the experiments. This is also seen in the line plots in Figure 5.1. The higher spreading factors (SF 11, 12), however, do not share this trait and this is clearly seen in Figure 5.1 where the firmware transmission time jumps between approx. 4.25 and 7.25 hours using spreading 12 and approx. 1.25 and 3.25 hours using spreading factor 11. This greater deviation from the mean time is due to the larger effect of ineffective uplinks on higher spreading factors compared to lower spreading factors because of the increased duty cycle time off air. This is elaborated on further in section 5.2.

Taking the mean values and standard deviation from Table 5.1, the mean firmware transmission times are compared against the best experimentally achieved times and analytical time estimates. The analytical time estimates were calculated using *Time on Air* estimates found with the freely available Semtech LoRa calculator. Using the *Time on Air* estimates and a duty cycle restriction of 1% the analytical time estimates were found. The comparisons are presented in Figures 5.2, 5.3 and 5.4.

The best experimental firmware transmission times again generally increased with increasing spreading factor, with the exception of SF 11 which was faster than SF 10. This was most likely due to a particularly efficient firmware transmission at SF 11, where uncontrollable external factors such as lost uplinks and lost downlinks may

Table 5.1: 5000 Byte Firmware Transmission Time

Spreading Factor	Mean Time (mins)	Std. Dev. (mins)
SF 7	17.25	2.11
Stop and Wait SF 7	32.55	4.52
SF 8	23.56	1.57
SF 9	46.28	7.11
SF 10	138.73	16.38
SF 11	140.50	43.91
SF 12	389.85	67.84

have been minimal. This particular update was even quicker than the analytical best estimate. There is a substantial difference between the analytical firmware transmission time estimates and average times at the higher spreading factors 10, 11 and 12. The increase in time can be accounted for by the inefficiencies that occurred (lost uplinks, lost downlinks, etc) during the firmware transmission. The analytical time estimate assumes that no uplinks or downlinks are lost in the process (it does however consider the inherent inefficiencies of the protocol). The effect of inefficiencies on time is elaborated on further in section 5.2. At the lower spreading factors of 7, 8 and 9, the differences between the average experimental times, best experimental times and analytical time estimates are negligible.

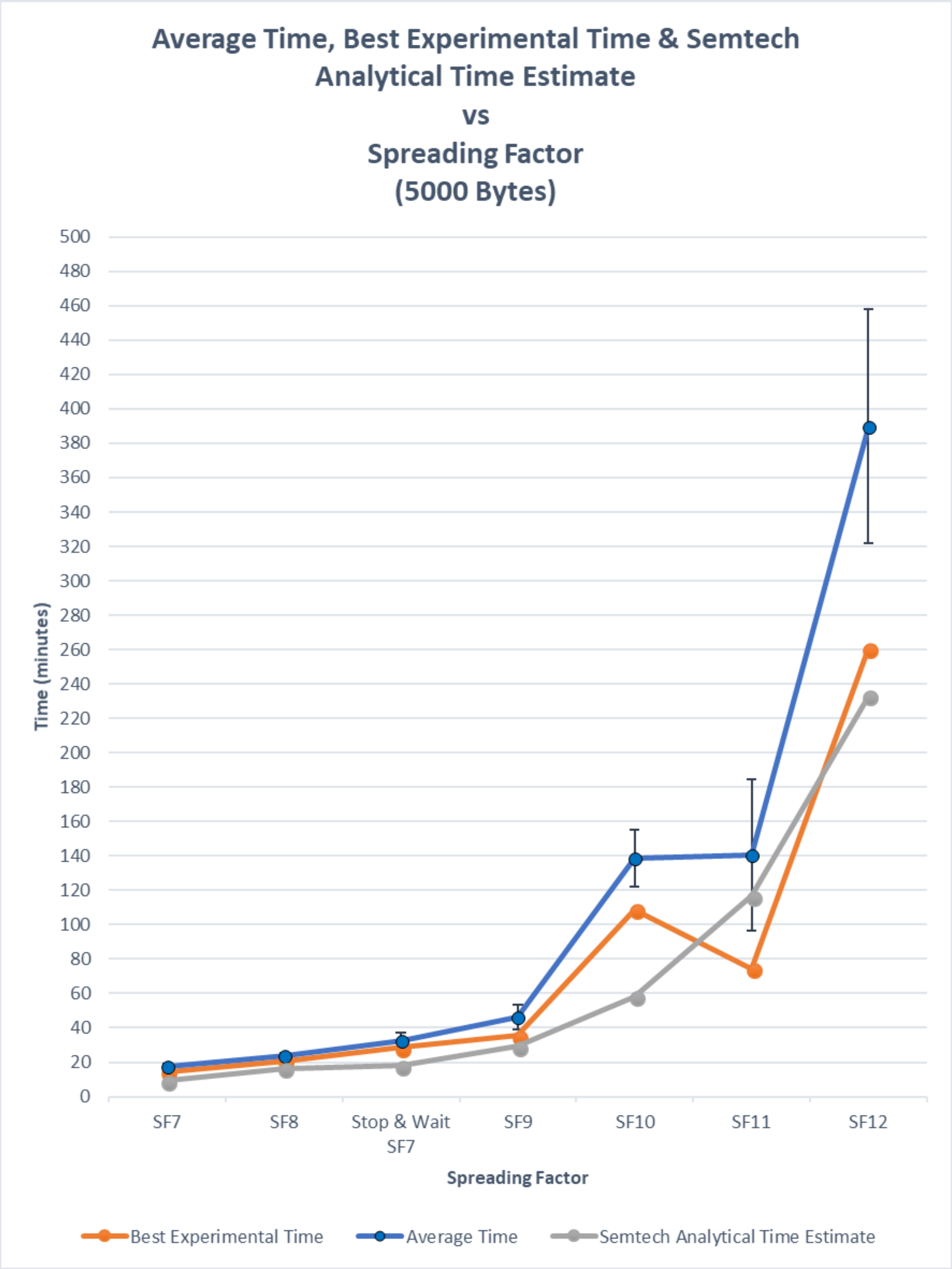


Figure 5.2: Average, Best and Analytical Firmware Transmission Times

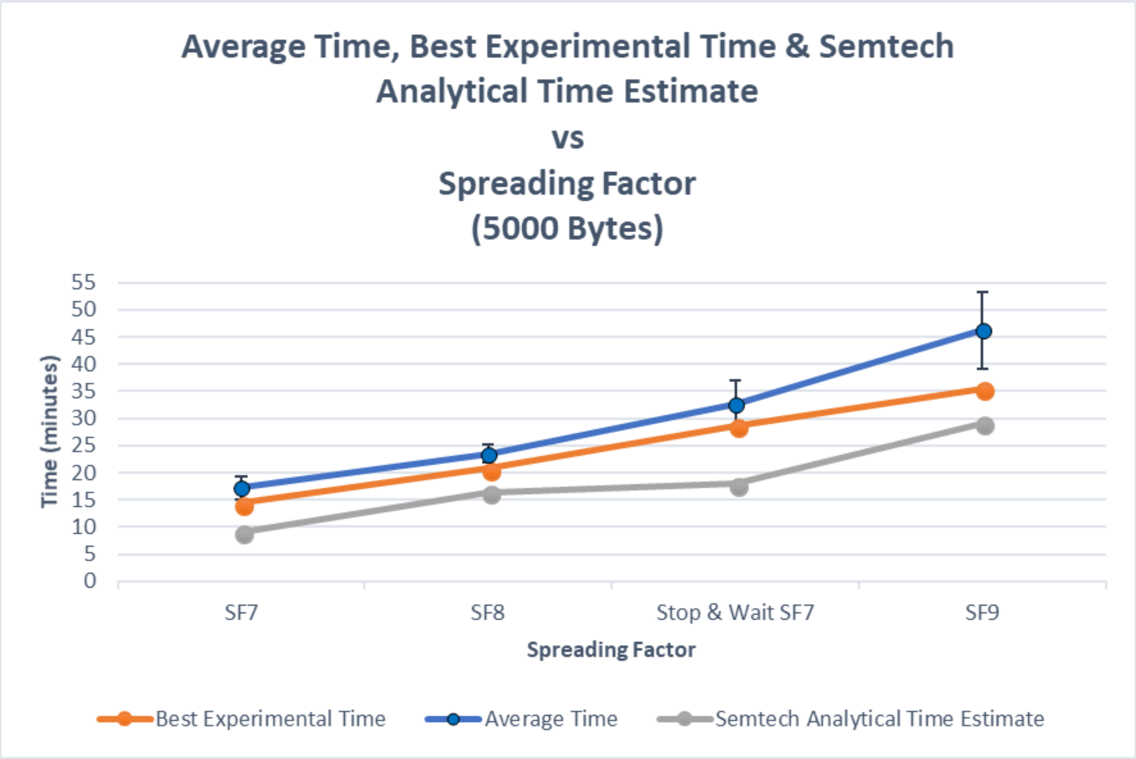


Figure 5.3: Average, Best and Analytical Firmware Transmission Times - SF 7, 8, 9

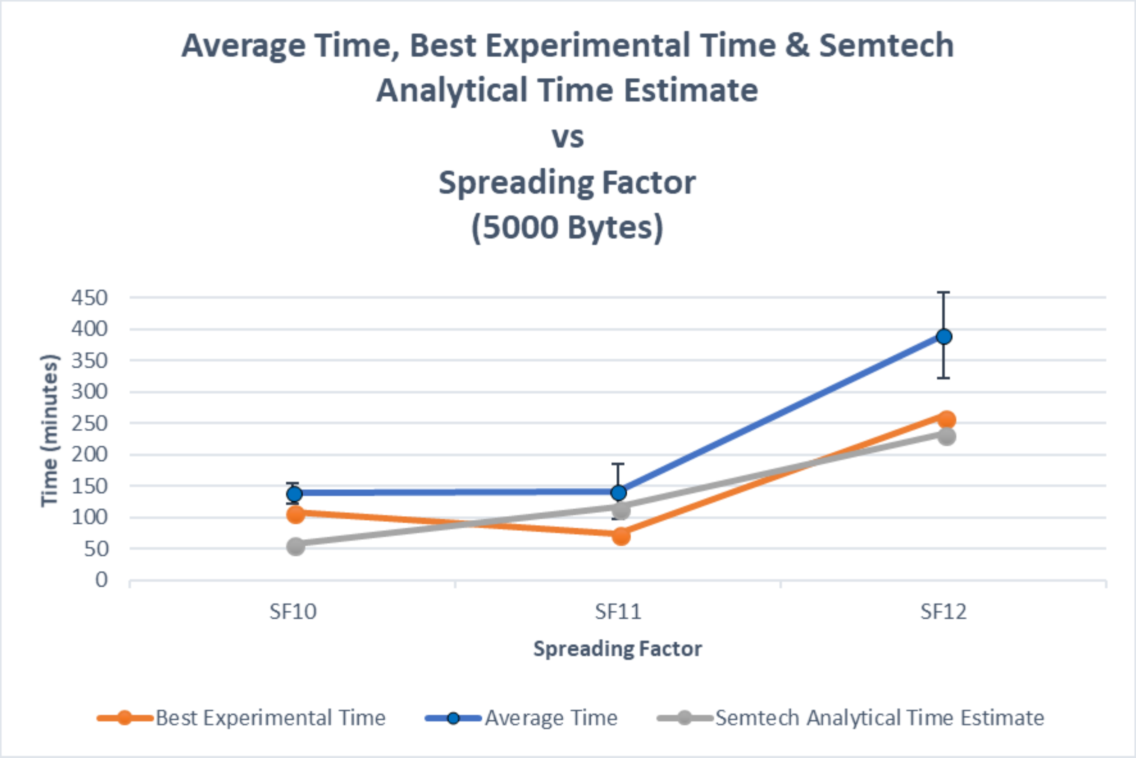


Figure 5.4: Average, Best and Analytical Firmware Transmission Times - SF 10, 11, 12

Based on the averaged experimental results for 5000 bytes in Table 5.1, Figure 5.5 presents an approximation of the required time to transmit firmware deltas of size 1000 bytes for each spreading factor, while Figures 5.6, 5.7 and 5.8 illustrate transmission time estimates for update deltas of sizes 1000 to 7000 bytes with respect to each spreading factor.

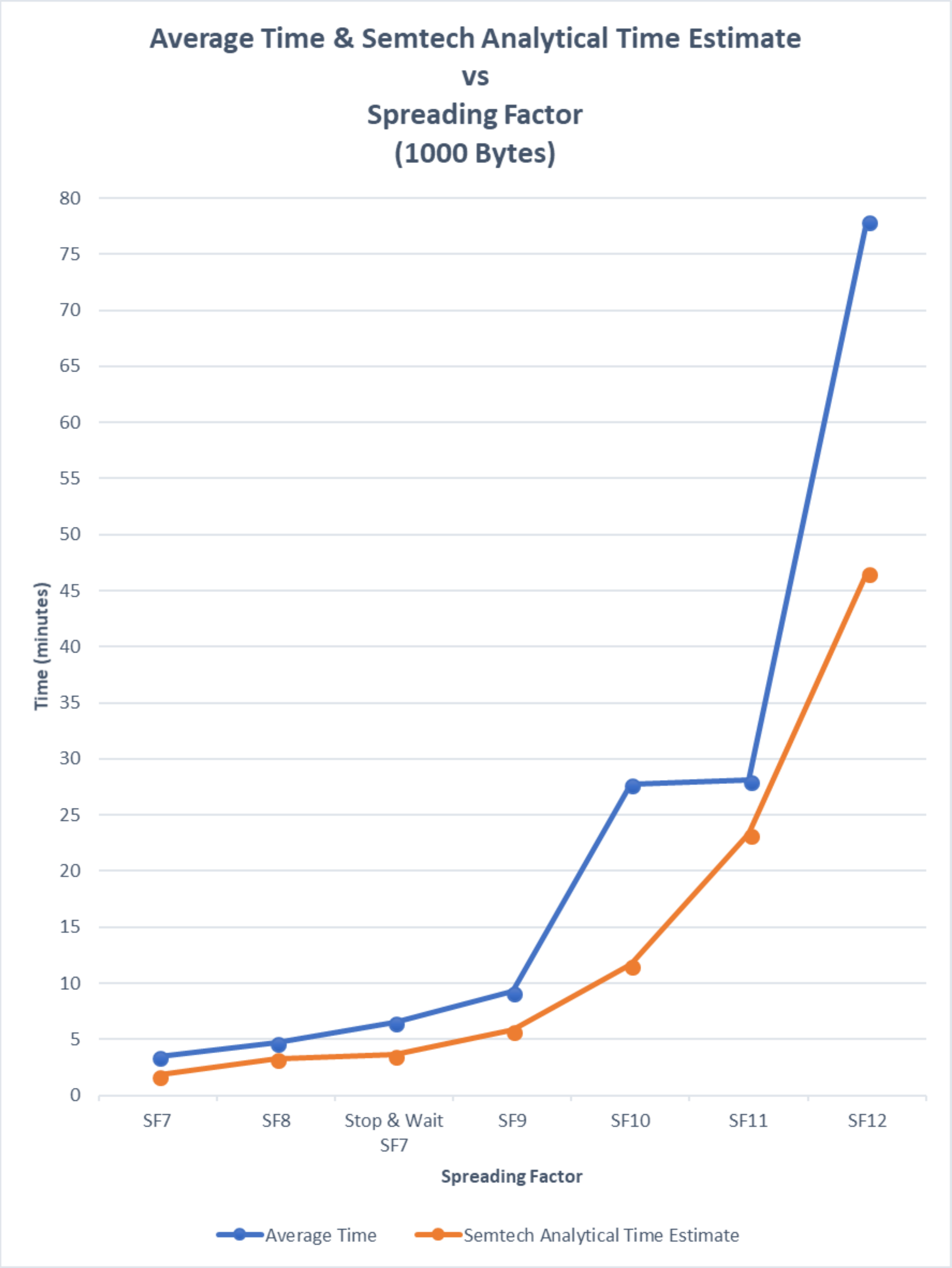


Figure 5.5: Experimental Firmware Transmission Time Approximation and Analytical Firmware Transmission Time for 1000 Bytes

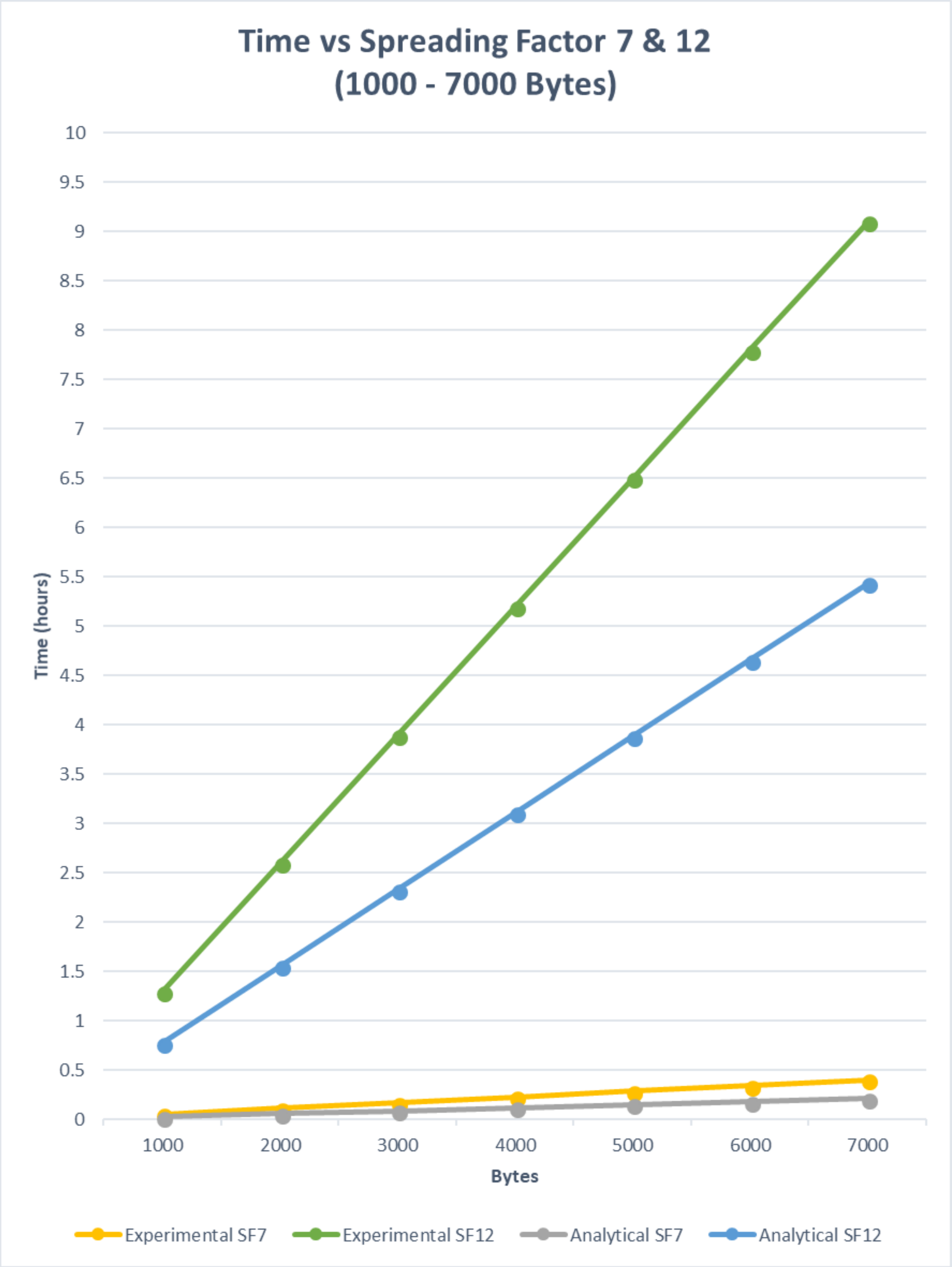


Figure 5.6: Experimental Firmware Transmission Time Approximations and Analytical Firmware Transmission Time for 1000 to 7000 Bytes - SF 7, 12

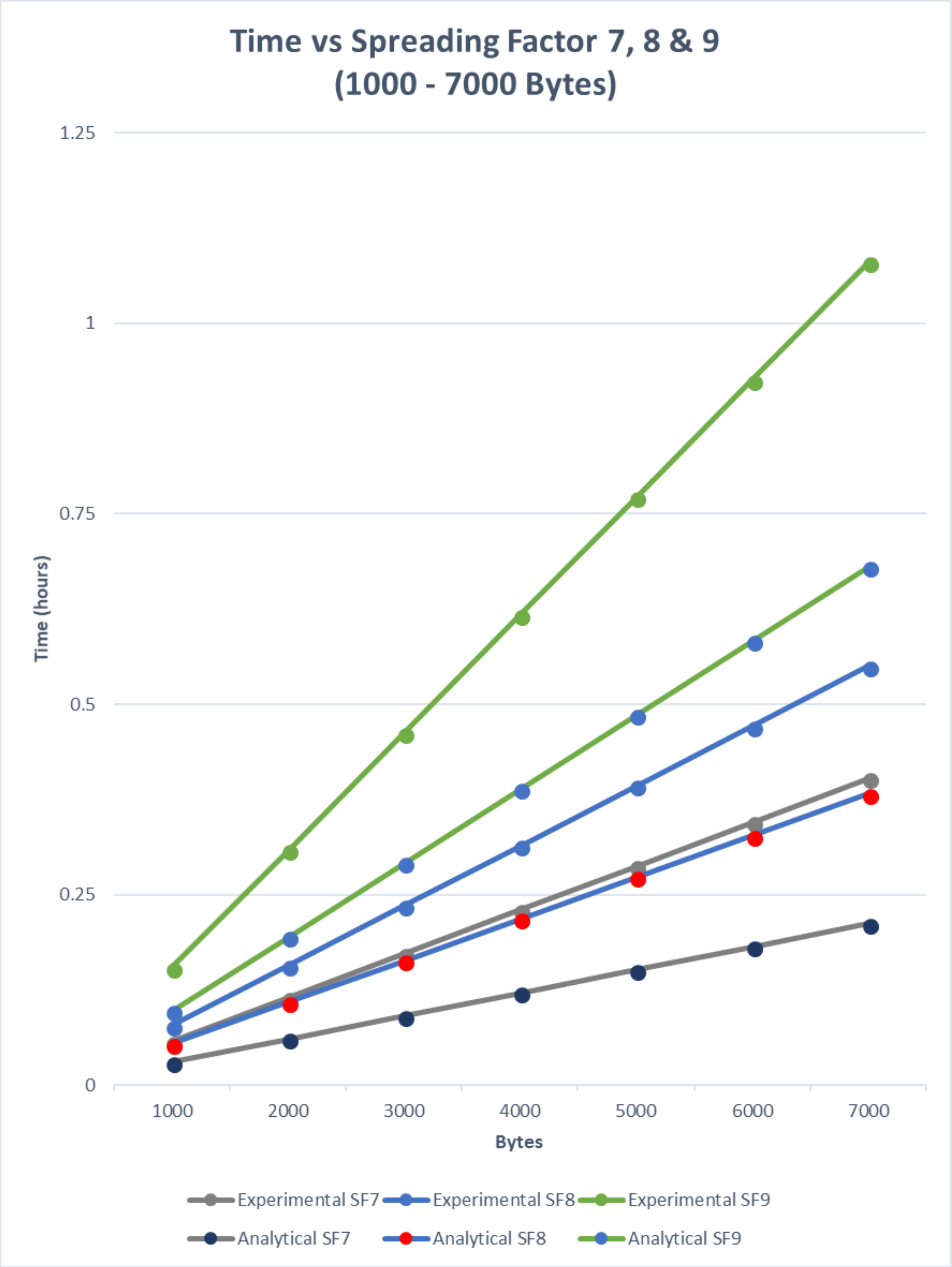


Figure 5.7: Experimental Firmware Transmission Time Approximations and Analytical Firmware Transmission Time for 1000 to 7000 Bytes - SF 7, 8, 9

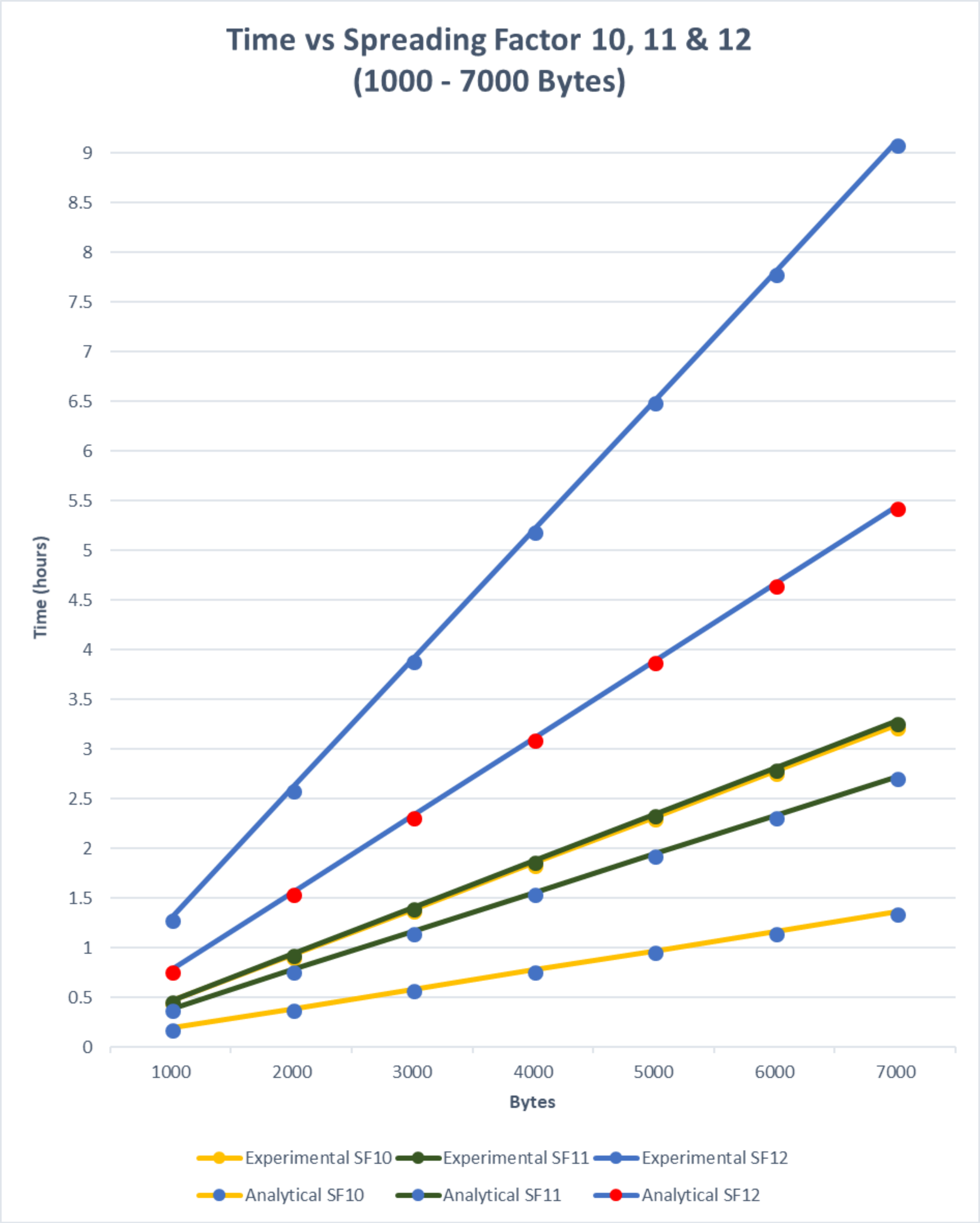


Figure 5.8: Experimental Firmware Transmission Time Approximations and Analytical Firmware Transmission Time for 1000 to 7000 Bytes - SF 10, 11, 12

5.2 Efficiency

In this evaluation, efficiency is measured as the percentage of effective uplinks to total uplinks. As the number of effective uplinks is constant for any particular update size (i.e. for a 5000 byte update it always takes 105 effective uplinks to receive 105 downlinks carrying 48 byte firmware segments), it is the number of ineffective uplinks that determines the efficiency of the transmission process. As previously described, ineffective uplinks are broken into two main types:

1. Ineffective Data Uplinks

2. Ineffective Response Uplinks

Figure 5.9 presents the averaged ineffective uplink breakdown for the Stop and Wait ARQ and Piggybacked Selective Repeat ARQ protocols at SF 7.

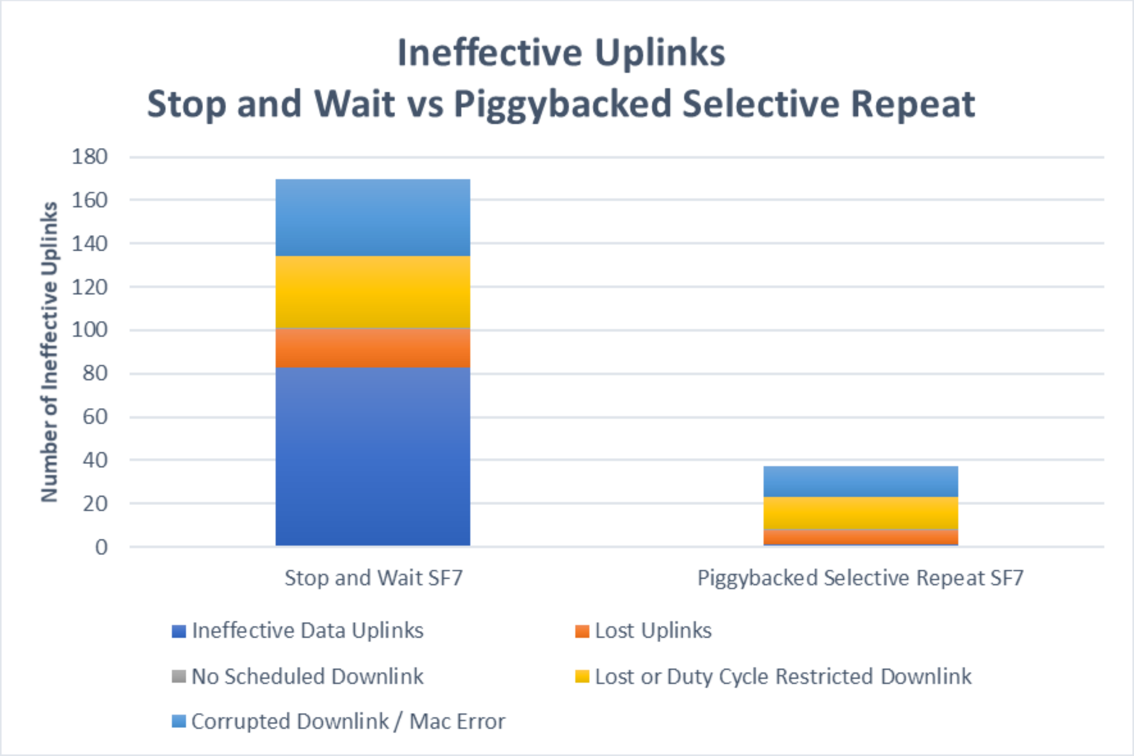


Figure 5.9: Ineffective Uplink Breakdown - Stop and Wait vs Piggybacked Selective Repeat

As expected, the Piggybacked Selective Repeat ARQ protocol is much more efficient than the Stop and Wait ARQ protocol. There are more ineffective data uplinks alone in the Stop and Wait ARQ protocol than the total number of ineffective uplinks in the Piggybacked Selective Repeat ARQ. In the Stop and Wait ARQ ineffective uplink breakdown, the biggest contributor to inefficiency is ineffective data uplinks, followed by lost/duty cycle restricted downlinks, corrupted downlinks / MAC errors and lost uplinks. The effect of the inherent no scheduled downlink is negligible.

The above is the extent to which the Stop and Wait ARQ protocol will be discussed in terms of efficiency and the remainder of the results presented here will focus solely on the Piggybacked Selective Repeat ARQ protocol.

5.2.1 Effect of Inefficiencies on Time

There is a very strong correlation between the firmware transmission inefficiency and the firmware transmission time. The strong correlation can be clearly seen in Figures 5.10, 5.11 and 5.12.

Refer back to the discussion on Figure 5.2 where there was a significant difference between the analytical and experimental times at higher spreading factors and a negligible difference at lower spreading factors. From the graphs in Figures 5.10, 5.11 and 5.12, there is a clear correlation between firmware transmission time and inefficiency. Intuitively, the effect of ineffective uplinks on time should increase with spreading factor. Imagine a firmware transmission scenario using spreading factor 7. Based on the Semtech LoRa calculator, the time off air after a 3 byte uplink and 1% duty cycle would be approximately 5 seconds. If in total there are 50 ineffective uplinks, this would increase the firmware transmission time by 250 seconds, or just over 4 minutes. Now, imagine the same scenario but at spreading factor 12. The corresponding time off air is approximately 132 seconds. If 50 uplinks are ineffective, this increases the time by 6600 seconds, or 1.8 hours. Clearly, the effect of ineffective uplinks on firmware transmission times at SF 12 is far greater than that at SF 7. This leads to the next analysis presented here, the breakdown of ineffective uplinks for the various spreading factors.

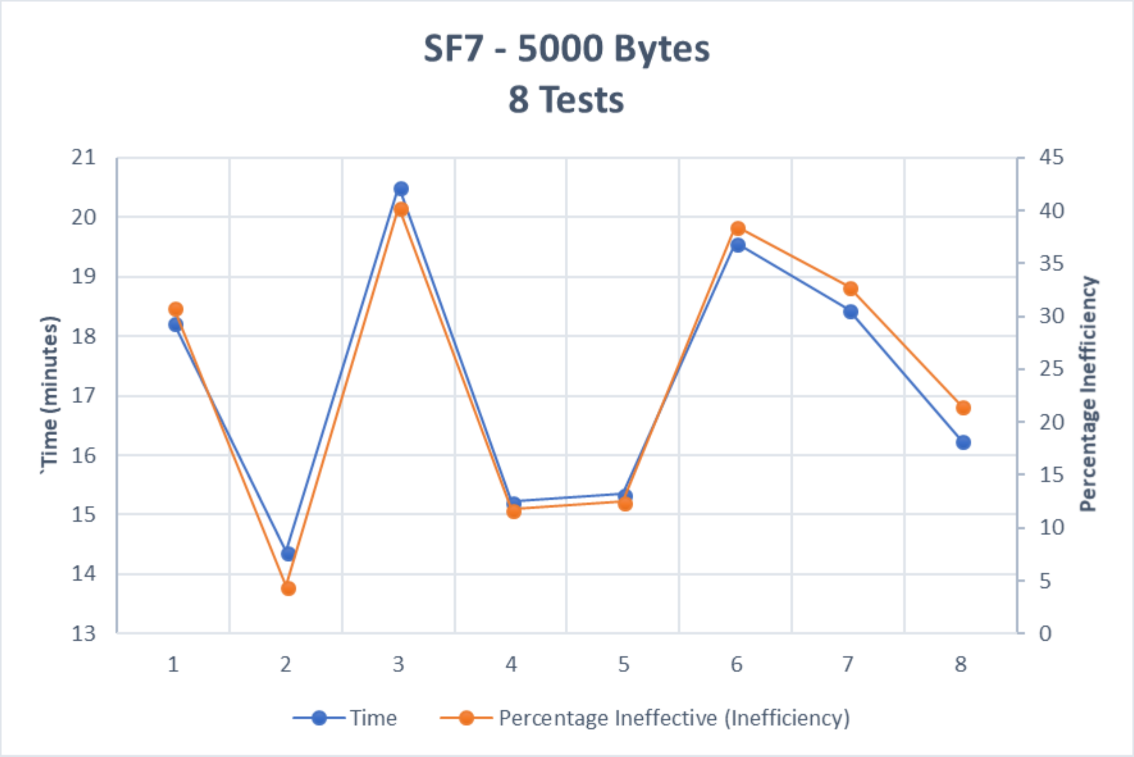


Figure 5.10: Relationship between Firmware Transmission Time and Inefficiency - SF 7

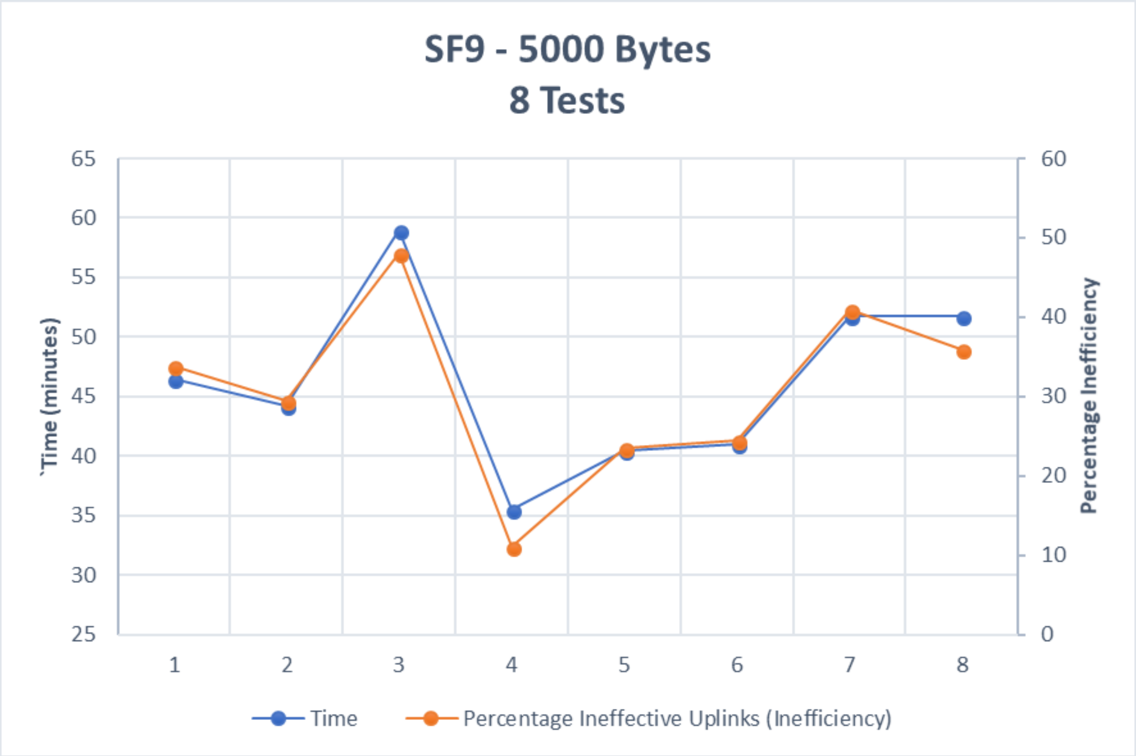


Figure 5.11: Relationship between Firmware Transmission Time and Inefficiency - SF 9

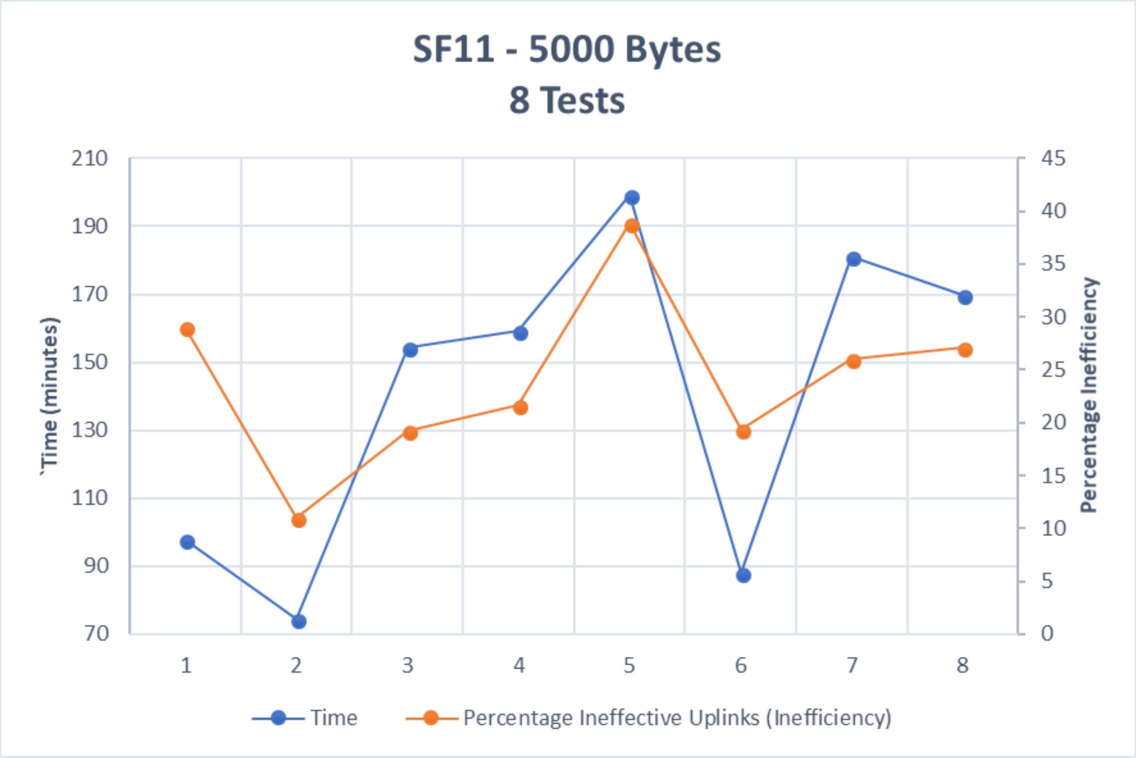


Figure 5.12: Relationship between Firmware Transmission Time and Inefficiency - SF 11

5.2.2 Inefficiency Breakdown

Figure 5.13 presents the averaged breakdown of ineffective uplinks for a 5000 byte update with respect to each spreading factor.

The inherent ineffective data uplink and no scheduled downlink, at just one each, have a negligible effect on the efficiency of the Piggybacked Selective Repeat ARQ protocol. Lost / duty cycle restricted downlinks and MAC errors are the greatest contributors to inefficiency, followed by lost uplinks. Corrupted downlinks / MAC errors are a major contributor to inefficiency across all spreading factors. While lost / duty cycle restricted downlinks are also a prominent contributing factor across all spreading factors, their role at spreading factor 10 is concerningly significant and is investigated further on.

To gain a better understanding of inefficiency breakdown with respect to each spreading factor, Figure 5.14 removes the negligible inherent inefficiencies and plots trend lines for lost uplinks, lost / duty cycle restricted downlinks and corrupted downlinks / MAC errors.

First, focusing on lost uplinks, there exists a general trend where the number of lost uplinks decreases with increasing spreading factor. As higher spreading factors trade data rate for greater distance and reliability, this trend is to be expected.

Looking at lost / duty cycle restricted downlinks, spreading factor 10 clearly stands out against the other spreading factors. If spreading factor 10 is excluded from consideration, there is a general decrease in lost / duty cycle restricted downlinks with increasing spreading factor. It should be noted that although this work could not definitively distinguish between a lost and a duty cycle restricted downlink, it is unlikely that any downlinks were duty cycle restricted as gateways are permitted to transmit on certain channels with a 10% duty cycle. Taking this into consideration and ignoring spreading factor 10, there is a general decrease in lost downlinks with increasing spreading factor. This again is to be expected as higher spreading factors trade data rate for greater distance and reliability.

Finally, looking at corrupted downlinks / MAC errors, the trend line shows an increasing number of corrupted downlinks / MAC errors with increasing spreading factor. This trend is not as expected, as one would expect the number of corrupted downlinks to decrease with increasing spreading factor. These corrupted downlinks / MAC errors are by far the largest contributor to inefficiency for spreading factors 8, 9, 11 and 12. Considering the effect of ineffective uplinks on time, especially at higher spreading factors, these MAC errors greatly contributed to the difference between the analytical firmware transmission times and the experimental times (Figure 5.2).

The high number of lost downlinks at spreading factor 10 and the high number of corrupted downlinks / MAC errors across all spreading factors was investigated further and the findings are discussed in the next section "Limiting Factors".

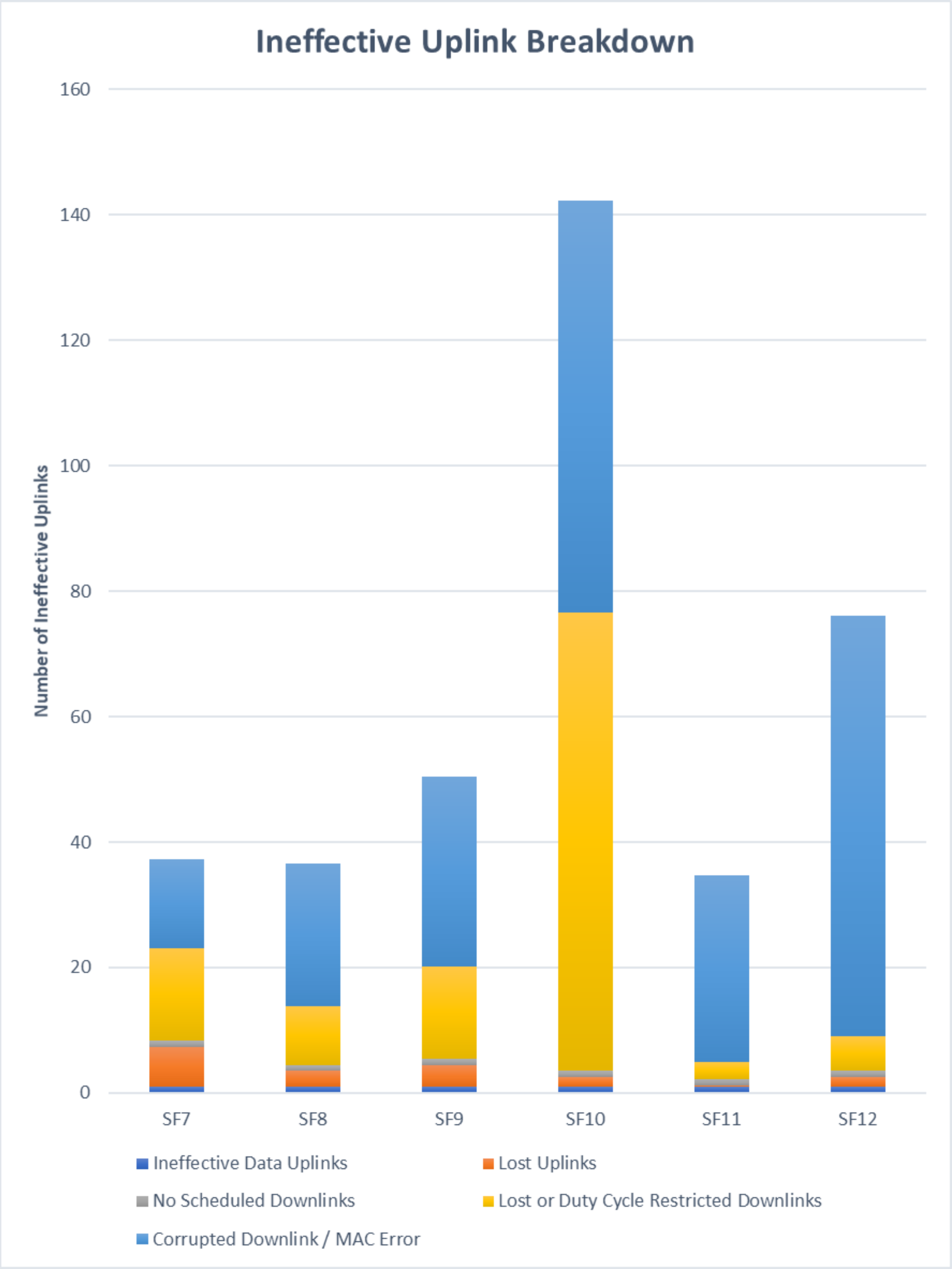


Figure 5.13: Averaged Breakdown of Ineffective Uplinks for a 5000 Byte Update

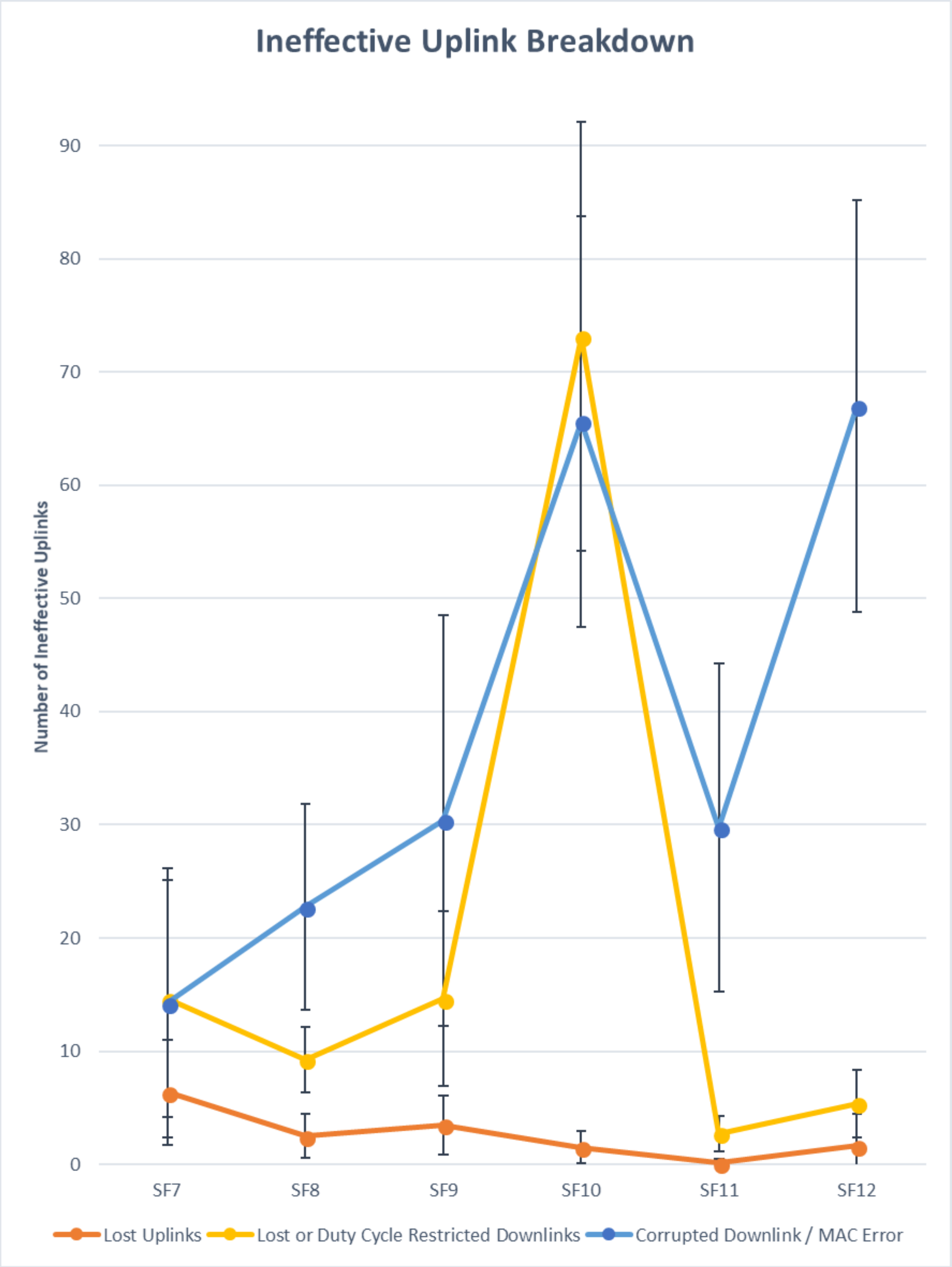


Figure 5.14: Averaged Breakdown of Ineffective Uplinks for a 5000 Byte Update - Without Inherent Inefficiencies

5.2.3 Limiting Factors

RN2483 SF 10 Receive Window Bug

Figure 5.14 presented some unexpected results regarding lost downlinks at spreading factor 10 and corrupted downlinks across all spreading factors. Further investigations into this issue highlighted a probable bug in the RN2483 LoRa module regarding downlink reception at spreading factor 10.

A spreading factor 10 receive window bug easily explains the issue regarding the high number of missing downlinks at spreading factor 10. However, its role in the high number of corrupted downlinks across all spreading factors is less intuitive and is explained in detail below.

First, just consider the high number of missing downlinks at spreading factor 10. Looking at the trendline in Figure 5.14, the number of missing downlinks at spreading 10 is far greater than the number of missing downlinks at spreading factors 7, 8 and 9. Considering that spreading factor 10 should be more reliable than 7, 8 and 9, this would suggest that there may be a bug in the RN2483 module receiving downlinks at spreading factor 10. Issues receiving downlinks at spreading factor 7 were previously reported and a subsequent update was released correcting this issue. At the time of this work, no update had been released regarding a fix for spreading factor 10 receive windows.

To understand the role this spreading factor 10 receive window bug plays in the inefficiency of other spreading factors requires a more detailed explanation on how the LoRa gateway infrastructure works.

As previously explained, Class A devices open two receive windows after each uplink during which it may receive a downlink. Nodes, in general, operate on three default channels centred on 868.1, 868.3 and 868.5 MHz with a total duty cycle allowance of 1% (or 0.33% per channel). When a node joins the network, the network may reconfigure the node's second receive window to be in line with its policy. LoRaWAN protocol suggests the reconfiguration of the second receive window to spreading factor 12 centred on 869.525 MHz. The first receive window continues to use the same parameters as the uplink. The advantage of reconfiguring the second receive window to 869.525 MHz is that EU specifications permit a 10% duty cycle allowance in the 869.4 to 869.65 MHz frequency band. As a result, when the gateway consumes its 1% duty cycle allowance responding in first receive windows (868.1, 868.3 and 868.5 MHz), the gateway can still respond to uplinks in the second receive window with a 10% duty cycle allowance.

Considering the above, it should now be mentioned that the LoRa infrastructure used in this work reconfigured the second receive window to use channel 869.525 MHz (10% duty cycle allowance) at spreading factor 10.

With this in mind, refer back to Figure 5.14 where presented is an unexpected trendline showing an increase in corrupted downlinks / MAC errors with increasing spreading factor (i.e. increasing reliability). As spreading factor increases from 7 to 12, the gateway consumes it duty cycle allowance on the first receive windows (868.1, 868.3 and 868.5 MHz at 1%) faster and faster. As a result, the gateway increasingly transmits downlinks on the 869.525 MHz channel with a duty cycle allowance of 10% and at spreading factor 10. Considering the probable bug in receiving downlinks at spreading factor 10, this would explain the increasing number of corrupted downlinks / MAC errors with increasing spreading factor.

To verify this assumption, two experiments were carried out. A 2000 byte update was transmitted to the node using spreading factors 7 and 12. The results are presented in Table 5.2.

The results show that for spreading factor 7, 6 out of the 7 downlinks sent in RX2 were received as a MAC error (86%). For spreading factor 12, the number of downlinks sent in RX2 increased to 30 (expected due to faster duty cycle consumption in RX1 with higher spreading factors) of which 21 of these were received as a MAC error

(70%).

None of the downlinks sent in RX1 at spreading factor 7 were received as a MAC error and just one of the downlinks sent in RX1 at spreading factor 12 were received as MAC error.

In total between the two experiments, of the 37 downlinks sent in RX2 with spreading factor 10, 27 of these were received with MAC error (73%). Again, this points to a bug in the RN2483 module, as spreading factor 10 should be more reliable than spreading factor 7, however 0% of downlinks transmitted at spreading factor 7 were lost or corrupted as opposed to greater than 73% of those transmitted at spreading 10.

Table 5.2: Summary of Downlinks and Receive Windows

	SF 7		**SF 12**	
	#	%	#	%
Sent on RX 1	41		35	
Received on RX 1	41	100	34	97
Lost	0	0	0	0
Corrupted / MAC Error	0	0	1	3
Sent of RX 2 (SF 10)	7		30	
Received on RX 2	1	14	8	27
Lost	0	0	1	3
Corrupted / MAC Error	6	86	21	70

51 Byte Downlink Payload Limit

It should also be noted that the LoRa infrastructure used in this work, by limiting the maximum downlink payload to 51 bytes, increases the firmware transmission time and energy at spreading factors 7, 8 and 9. The LoRaWAN specification defines a maximum payload size of 222 bytes for spreading factors 7 and 8 and a maximum payload size of 115 bytes for spreading factor 9. Hence, for an update of 5000 bytes using spreading factor 7, only 23 downlinks would be required if the maximum downlink payload was 222 bytes (carrying 219 bytes of firmware), versus 105 downlinks with the 51 byte restriction (carrying 48 bytes of firmware). This is over four times the required number of uplinks, essentially quadrupling the energy usage and the time required. Similarly, for spreading factor 9 with a maximum payload size of 115 bytes, less than half the number of packets would be required, essentially halving the time and energy cost.

5.3 Coverage

Figure 4.5 showed the locations of the six nearest gateways (purple) to the node (blue). When the node transmits an uplink it may be received by one or many of these gateways. The Network Server chooses which gateway responds to the node, as such this work had no control over which particular gateway was selected. This section presents findings on coverage using the gateway logs. The most efficient 5000 byte firmware transmission experiment for each spreading factor was chosen and the gateway logs are summarised in Tables 5.3, 5.4 and 5.5.

Percentage Uplinks Received is the percentage of transmitted uplinks that were successfully received by that gateway, while *Percentage Downlinks* is the percentage of the transmitted downlinks that were transmitted by that particular gateway as chosen by the Network Server.

Table 5.3: Gateway Logs Summary - SF 7 & 8

		SF7				**SF8**			
Gateway	**Distance (km)**	**% Uplinks Received**	**Avg. SNR (dB)**	**Avg. RSSI (dBm)**	**% Downlinks**	**% Uplinks Received**	**Avg. SNR (dB)**	**Avg. RSSI (dBm)**	**% Downlinks**
Trinty Education Centre	0.75	97.3	6.0	-97.3	97.2	93.4	3.8	-96.9	107.0
Croke Park	1.65	75.5	-3.8	-110.4	1.8	76.2	-7.1	-113.3	13.0
DIT Kevin St.	1.65	10.0	-6.1	-114.3	0.9	4.9	-12.9	-116.3	9.0
DCU Alpha	3.75	30.9	-3.9	-120.3	0.0	58.2	-3.2	-119.2	0.0
UCD	5.25	10.0	-7.5	-119.1	0.0	34.4	-10.1	-118.2	0.0
Three Rock	11.20	0.0	-	-	0.0	0.0	-	-	0.0
Node Avg. Downlink SNR (dB)		0.7				0.1			

Table 5.4: Gateway Logs Summary - SF 9 & 10

		SF9				**SF10**			
Gateway	**Distance (km)**	**% Uplinks Received**	**Avg. SNR (dB)**	**Avg. RSSI (dBm)**	**% Downlinks**	**% Uplinks Received**	**Avg. SNR (dB)**	**Avg. RSSI (dBm)**	**% Downlinks**
Trinty Education Centre	0.75	80.2	1.9	-101.2	70.1	93.6	3.2	-96.6	81.7
Croke Park	1.65	74.8	-5.9	-110.9	20.6	94.7	-9.1	-113.6	5.9
DIT Kevin St.	1.65	29.2	-9.9	-112.4	8.8	41.5	-13.0	-113.8	11.8
DCU Alpha	3.75	47.5	-6.2	-122.4	0.5	75.0	-6.6	-121.2	0.0
UCD	5.25	16.8	-11.2	-121.4	0.0	47.3	-13.0	-122.7	0.5
Three Rock	11.20	0.0	-	-	0.0	0.0	-	-	0.0
Node Avg. Downlink SNR (dB)		0.0				0.3			

Table 5.5: Gateway Logs Summary - SF 11 & 12

		SF11				**SF12**			
Gateway	**Distance (km)**	**% Uplinks Received**	**Avg. SNR (dB)**	**Avg. RSSI (dBm)**	**% Downlinks**	**% Uplinks Received**	**Avg. SNR (dB)**	**Avg. RSSI (dBm)**	**% Downlinks**
Trinty Education Centre	0.75	97.5	4.7	-95.7	91.5	95.7	-1.3	-98.5	78.7
Croke Park	1.65	81.4	-8.1	-113.9	0.8	94.9	-10.9	-113.0	5.9
DIT Kevin St.	1.65	76.3	-12.9	-116.3	7.6	72.5	-15.4	-115.4	14.0
DCU Alpha	3.75	96.6	-3.2	-119.2	0.0	79.7	-8.5	-123.6	0.7
UCD	5.25	89.0	-10.1	-118.2	0.0	73.2	-14.8	-120.0	0.7
Three Rock	11.20	0.0	-	-	0.0	9.4	-17.8	-119.2	0.0
Node Avg. Downlink SNR (dB)		1.8				0.4			

Figure 5.15 graphs the percentage of uplinks received at each gateway for each spreading factor. The nearest gateway, the TEC, received uplinks with high reliability for all spreading factors. As the gateway distance from the node increases the percentage packet reception is seen to generally decrease as expected. Distance, however, is not the only factor which affects reception. The DIT Kevin St. gateway has very poor reception for spreading factors 7, 8, 9 and 10, despite it being the same distance from the node as the Croke Park gateway. This is most likely due to the poor line of sight between the node and the DIT Kevin St. gateway. The node's indoor location has a north facing window in the general direction of Croke Park, while DIT Kevin St. is in a westerly direction.

Better line of sight probably also contributes to the higher reception of uplinks at DCU Alpha versus UCD, however the fact that DCU is approximately 1.5 km closer also plays a factor.

Only uplinks at spreading factor 12 were received by the Three Rock gateway, over 11 km away from the node location, with less than 10% of packets successfully received.

Figure 5.16 shows that the received Signal-to-Noise ratios generally decreased with increasing distance from the node. The exception being DIT Kevin St. and DCU Alpha, where the signal to noise ratios were respectively lower and higher than expected.

Figure 5.17 shows the average RSSI at each gateway. In general, RSSI decreased with increasing distance. Interestingly however, is that there is no significant difference between the RSSI values at Croke Park and DIT Kevin St., contrasting with the very significant reduction in SNR at DIT Kevin St. versus Croke Park seen in Figure 5.16. This would suggest that background noise present at DIT Kevin St. is significantly hindering packet reception.

Figure 5.18 presents the averaged noise floor at each location. The noise floor at DIT Kevin St. is noticeably higher than that at DCU Alpha and UCD. The TEC and DIT

Kevin St. gateways are both located in busy inner-city locations, while the DCU and UCD gateways are both found in relatively quiet and large university campuses, explaining the difference in noise floor. DIT Kevin St. also has a larger noise floor than Croke Park, which would contribute to the poorer packet reception.

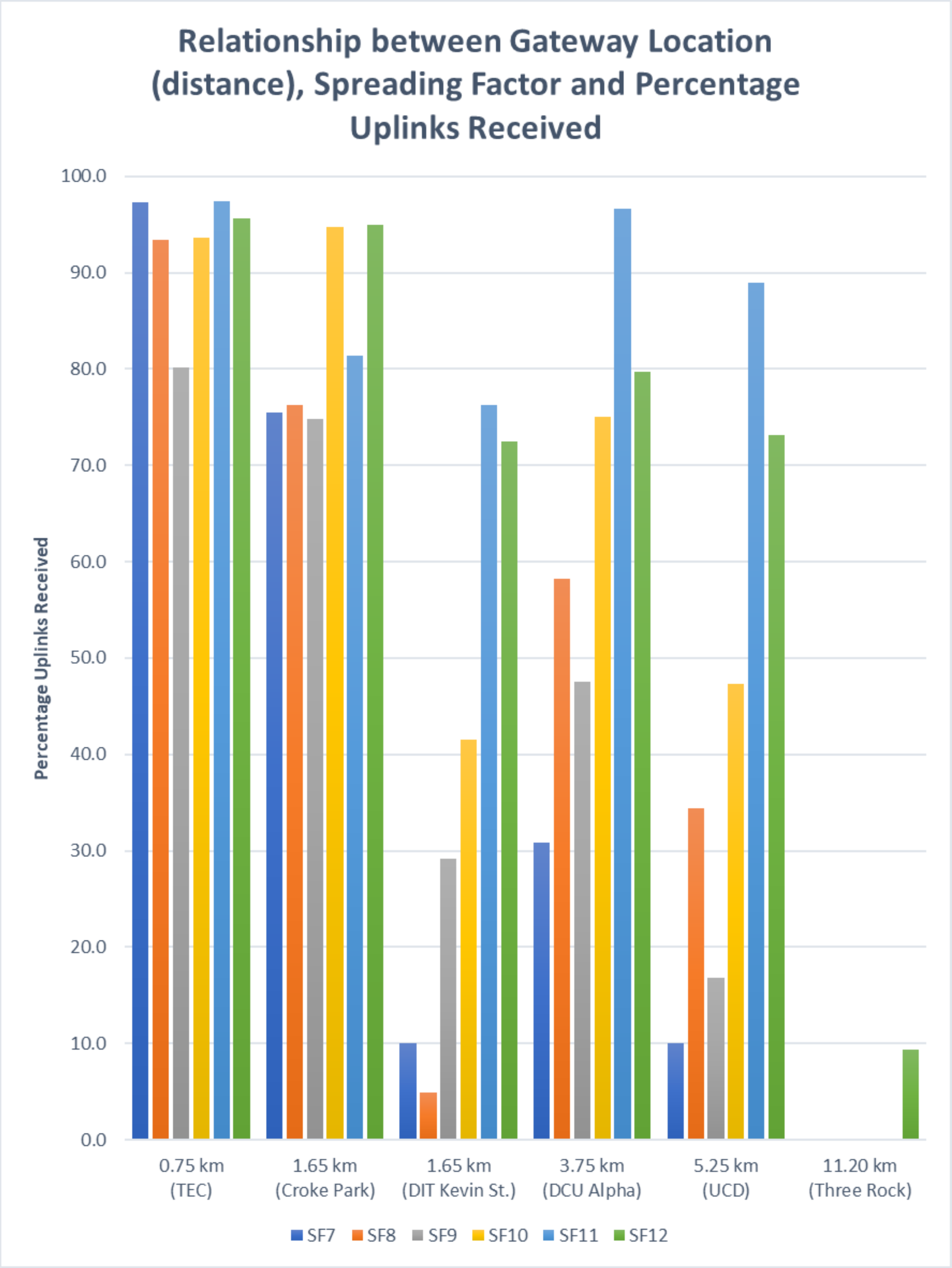


Figure 5.15: Percentage of Uplinks Received at Each Gateway for each Spreading Factor

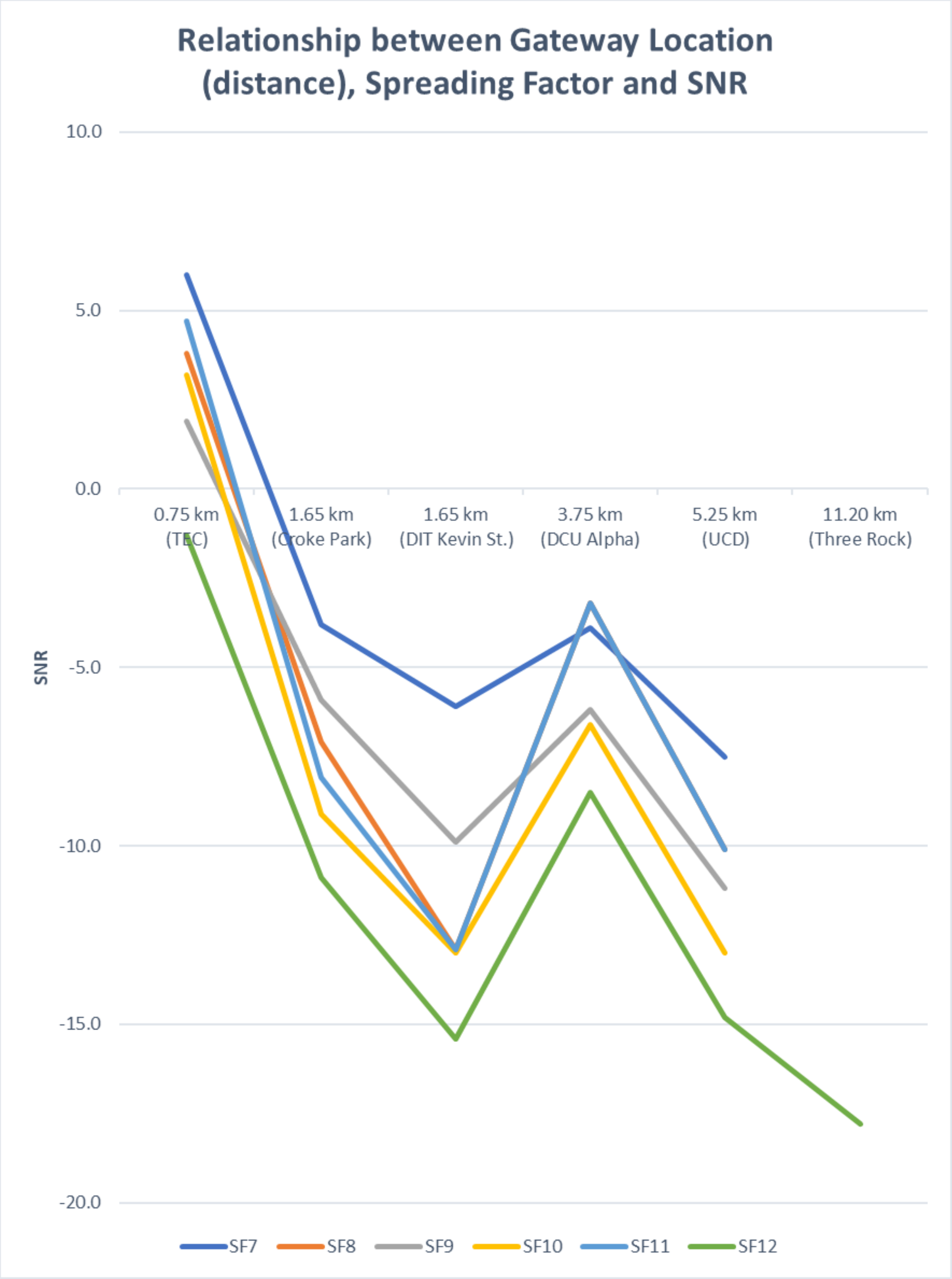


Figure 5.16: Uplink SNRs at Each Gateway for each Spreading Factor

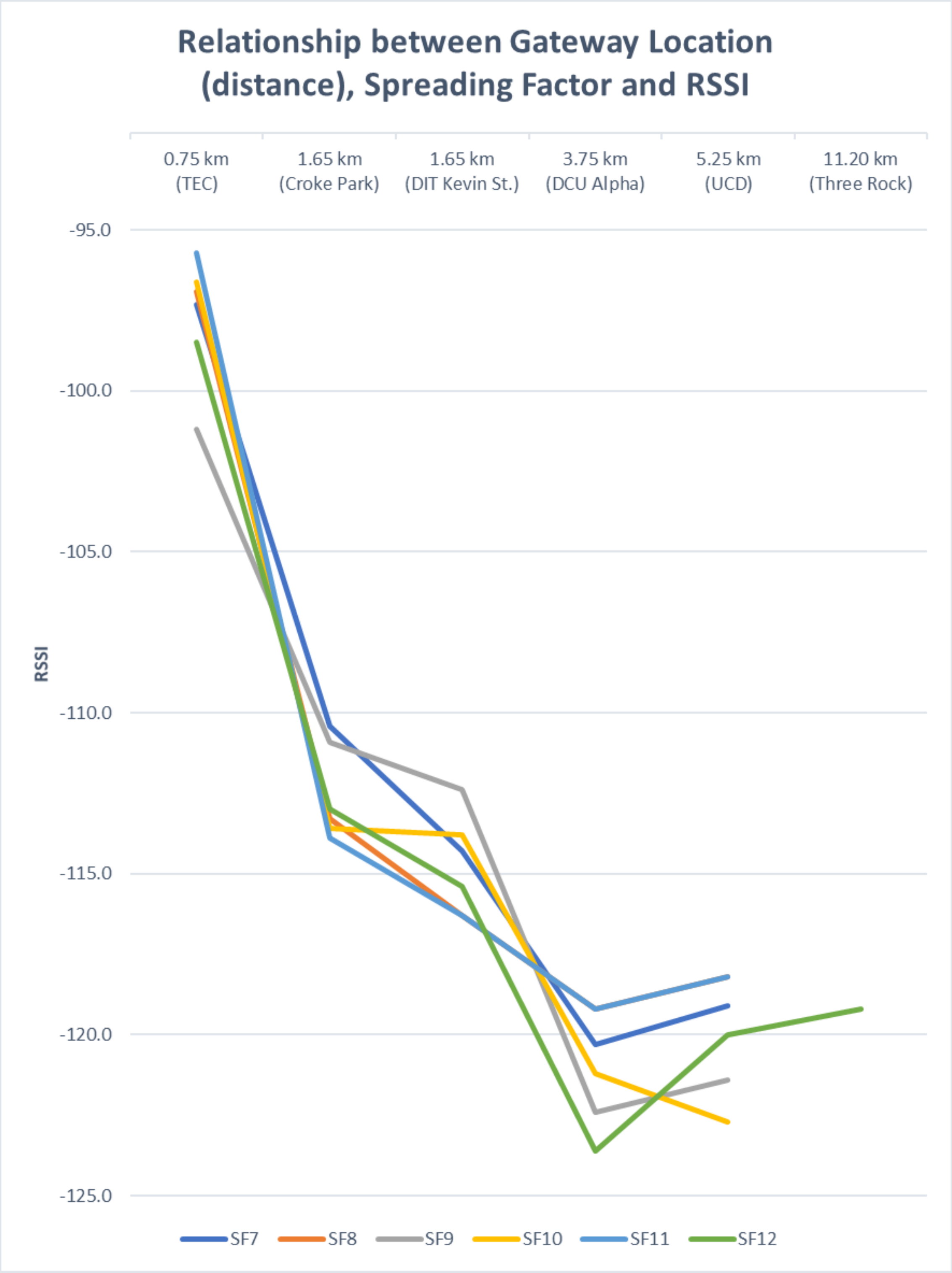


Figure 5.17: Uplink RSSI at Each Gateway for each Spreading Factor

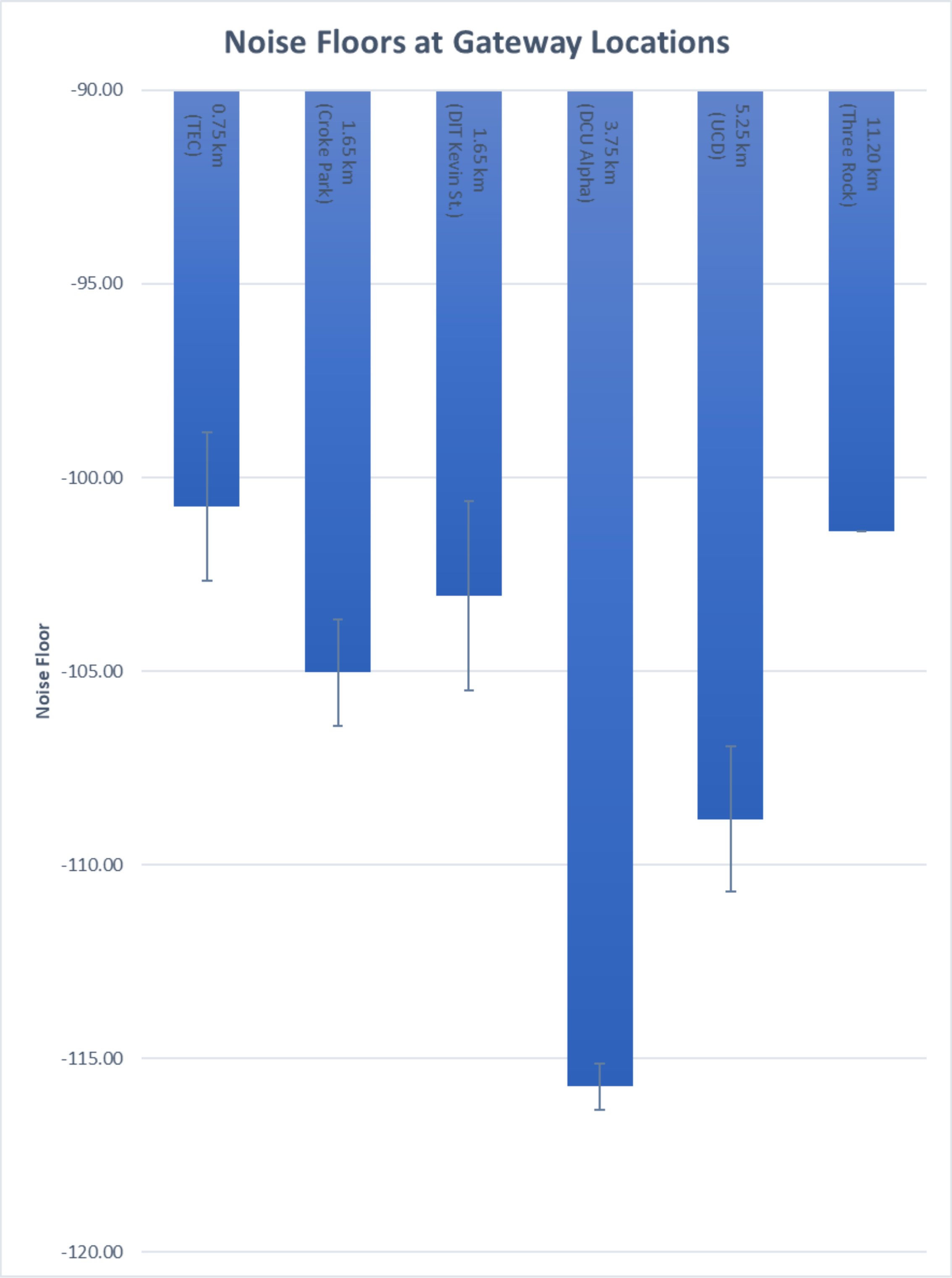


Figure 5.18: Averaged Noise Floor at Each Gateway Location

The final topic to be presented is scalability. Figures 5.19 and 5.20 illustrate how many nodes may be served by a firmware update delta by a single gateway under ideal conditions. The results are based on time on air calculations using the Semtech LoRa Calculator. Gateways have a duty cycle allowance of 10% compared to the 1% allowance of nodes. Therefore, a gateway can serve multiple nodes simultaneously. The results presented in Figure 5.20 are based on the number of 51 byte downlinks a gateway can transmit at a certain spreading obeying a duty cycle allowance of 10% within a certain time frame. If nodes are transmitting as quickly as possible, the gateway should be able to serve approximately four nodes simultaneously. As the time interval between uplink firmware requests increases, the number of possible nodes served also increases. The rate of number of nodes served with respect to the time interval decreases with increasing spreading factor.

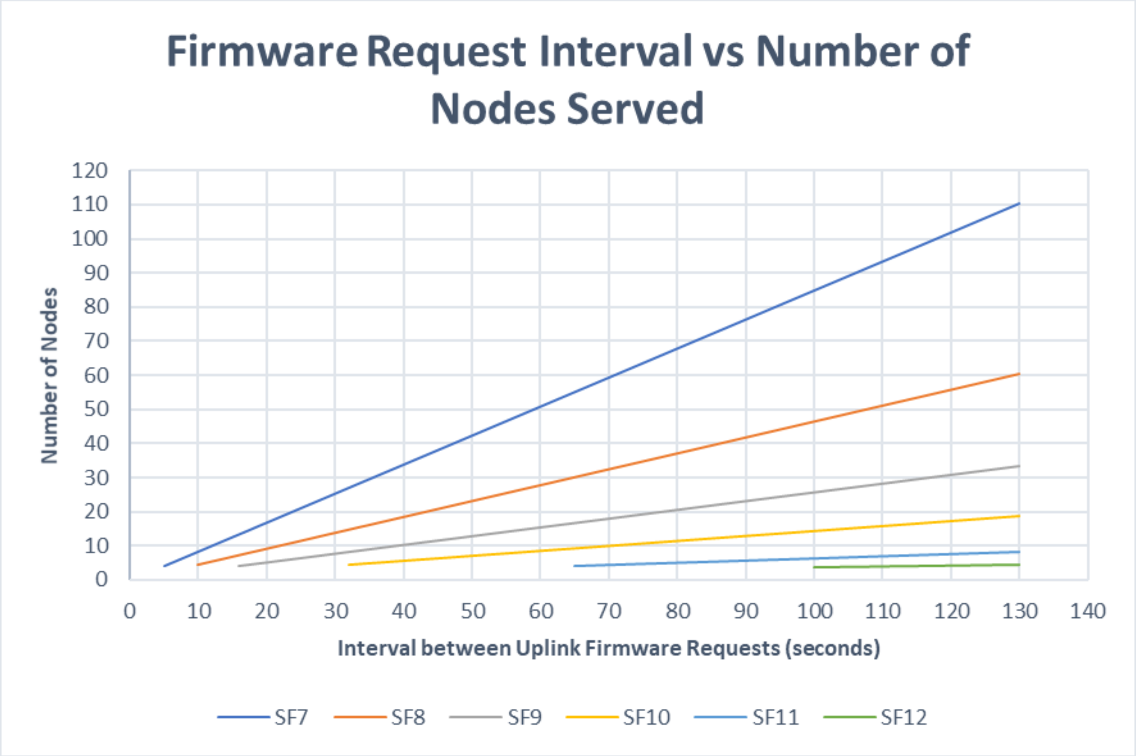


Figure 5.19: Number of Nodes Served a Firmware Update by a Single Gateway

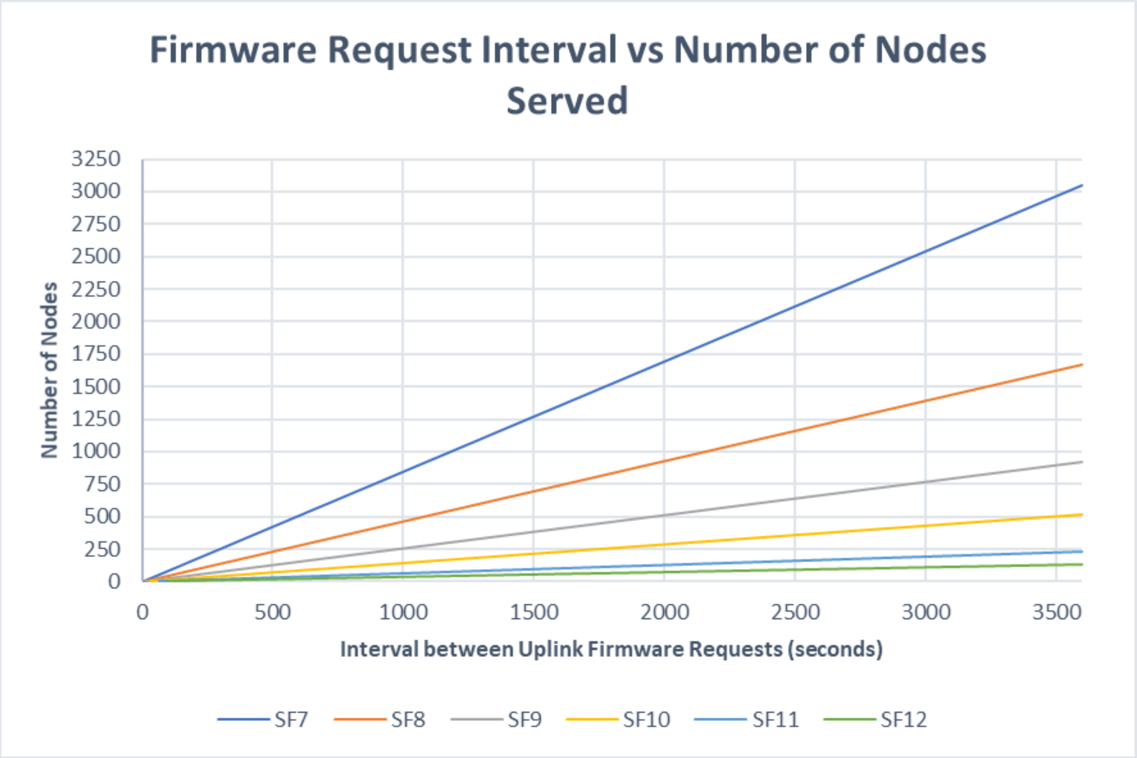


Figure 5.20: Number of Nodes Served a Firmware Update by a Single Gateway

6 Discussion

The following chapter discusses the presented results and evaluates LoRa and the designed protocols for firmware update transmission as well as addressing the contribution of this work to low-power wide-area communications and wireless firmware updating.

6.1 Results

LoRa Class A bi-directional communication underwent extensive testing over the course of this work. Most research to date has focused on uplink communication, however in this work both uplinks and downlinks received equal attention. Class A devices, in general, are designed with priority placed on uplink communications, as the devices are intended for applications which transmit sporadically to a base station. For wireless firmware updating however, priority is in the downlink, with the uplink acting as a necessary tool to bring in the downlink.

The discovered spreading factor 10 receive window bug had a negative impact on the validity of this work. This work based efficiency solely on the ability to receive a downlink after transmitting an uplink. The fact that a bug prevented downlink reception at spreading factor 10, combined with the fact that the LoRa infrastructure used in this work transmits all second receive window downlinks at spreading factor 10, understandably had a very large impact on the results. If this issue had been resolved, or if there had been time to re-evaluate using a different LoRa module, better results in terms of both time and efficiency would be expected.

6.2 Time

The Piggybacked Selective Repeat ARQ protocol presented promising firmware update transmission times for firmware deltas of 7000 bytes and less.

At the lower spreading factors of 7 and 8, 7000 byte deltas can be comfortably transmitted in less than one hour, closer to the 30 minute mark. Spreading factor 9 can achieve 7000 byte firmware transmission in just over an hour, however if the spreading factor 10 receive window issue was resolved, this time would be expected to be reduced to below 45 minutes. Considering that a node's application may require it to only transmit once every few hours, the node should be able to pull down an update image delta without impeding its application sensing function.

The firmware transmission time increases quite substantially at the higher spreading factors 10, 11 and 12. 1000 byte firmware deltas can still be transmitted in times less than and around the one hour mark, however, experimental times at 7000 bytes ranged from 3 to 9 hours at spreading factors 10 and 12 respectively. The corresponding analytical times ranged from approximately 1.5 to 5.5 hours. If the second receive window bug was resolved, the experimental firmware transmission times would be expected to be closer to the analytical times. Again, considering the typical LoRa applications, these long firmware transmission times may not be such an issue. Many LoRa applications are expected to have substantial down times, such as a water meter that transmits once a day or a parking lot meter that does not operate at night time. For these types of applications, trickling down an update over a 5 or 6 hour period should not have an effect on the primary application. Also, because of the bi-directional communication nature of LoRa, it should be noted that the firmware transmission protocol could be redesigned to piggyback on top of the node's primary application. This way, the node could continue its sensing duties and the gateway could respond to sensor reading uplinks with firmware update carrying downlinks.

6.3 Efficiency

As previously explained, efficiency in this work was measured in terms of effective and ineffective uplinks, which are in turn based on whether the energy cost of the uplink transmission was justified by a response of a useful firmware segment in one of the receive windows. The actual energy consumption was not measured in this work. As the number of packets to transmit a specific firmware delta size is constant, the primary determinant of efficiency is the number of ineffective uplinks as opposed to the number of effective uplinks.

The inefficiencies due to lost uplinks and downlinks decreased with increasing spreading factor, except at spreading factor 10 due to the receive window bug. In general, all inefficiencies would be expected to decrease with increasing spreading factor, however, the number of corrupted downlinks significantly increased with increasing spreading factor. This again, is explained by the receive window bug in the RN2483 module, as the gateway sends more downlinks in RX2 at spreading factor 10 (10% duty cycle) when it's first receive windows are operating at higher spreading factors (1% duty cycle). This is a technique used by the gateway to save overhead. If the bug was resolved, or another LoRa module used, the number of corrupted downlinks would be expected to decrease with increasing spreading factor.

The Stop and Wait ARQ protocol was shown to have at least $2N$ inherent ineffective uplinks for a firmware update requiring N downlink packets, capable of a maximum efficiency of 50%. The Piggybacked Selective Repeat ARQ protocol was shown to have only two inherent ineffective uplinks, regardless of the number of downlink packets required, capable of maximum efficiency of $\frac{N}{N+2}$. As N increases, the efficiency approaches 100%. Despite the high potential efficiency of the Piggybacked

Selective Repeat ARQ protocol, experimental inefficiencies did not reach this full potential. The greatest efficiency achieved was 95.5% (at spreading factor 7).

Table 6.1 summarises the averaged efficiencies for the different spreading factors. It should be noted however, that the spreading factor 10 receive window bug had as much of an effect on efficiency as it did on time, and if the issue was resolved, or another LoRa module used, the experimental efficiency would be expected to approach the theoretical optimum efficiency.

Table 6.1: 5000 Byte Firmware Transmission Efficiency

Spreading Factor	**Mean Efficiency (%)**	**Std. Dev. (%)**
SF 7	75.8	12.6
Stop and Wait SF 7	39.5	6.7
SF 8	74.8	7.0
SF 9	69.1	10.7
SF 10	43.0	5.7
SF 11	76.0	7.7
SF 12	58.8	7.4

6.4 Coverage

This work presented some interesting results on the importance of gateway location. The closest gateway, at approximately 0.75 km from the node, recorded the best packet reception rate. Subsequently, this gateway was also chosen by the network server to transmit the majority of downlinks.

Distance, however, was not found to be the only determinant of packet reception rate. DIT Kevin St., at just 1.65 km from the node, was in many cases found to have worse packet reception than DCU Alpha and UCD, which were 2.1 km and 3.6 km respectively further away from the node. Croke Park, at 1.65 km away from the node, had relatively good packet reception, much better than that of DIT Kevin St., particularly at the lower spreading factors, however it is likely that the superior line of sight between the node and Croke Park against that of the node and DIT Kevin St. contributed to this.

Noise floor, as well as distance, was found to hinder packet reception. This was particularly evident at DIT Kevin St. again, a busy city area. This would also have contributed to the poorer packet reception at DIT Kevin St. when compared against reception at DCU Alpha and UCD, two large and relatively quiet university campuses.

These results provide insight on how an urban LoRa infrastructure should be designed. To have a high degree of packet reception (greater than 70%) at the lower spreading factors, gateways need to be within approximately 2 km of the node. At the higher spreading factors 11 and 12, the equivalent reception can be achieved by gateways approximately 5 km from the node. Only spreading factor 12 can achieve distances greater than 11 km, however, it is achieved with poor reliability.

7 Conclusion

This work set out to evaluate LoRa low-power wide-area technology for firmware update transmission. The work was based on real-life experiments, conducted using the LoRa RN2483 module and the Pervasive Nation infrastructure in Dublin. The evaluation was to consider the time required to transmit a firmware update to a LoRa Class A device as well as considering the efficiency of the process. On top of this, results relating to coverage in an urban environment were presented.

Two firmware transmission protocols were designed and evaluated: Stop and Wait ARQ and Piggybacked Selective Repeat ARQ. The Stop and Wait ARQ protocol was identified early on to be an unsuitable protocol for the unique bi-directional communication nature of LoRa Class A devices, capable of a maximum efficiency of 50%. The Piggybacked Selective Repeat ARQ protocol showed promising potential for transmitting firmware over LoRa, capable of close to 100% efficiency when the number of packets required is large.

Each protocol was experimentally evaluated with respect to the six LoRa spreading factors. The Piggybacked Selective Repeat ARQ protocol demonstrated reliable capabilities to transmit a 7000 byte update to a node in less than thirty minutes at spreading factors 7 and 8. At higher spreading factors 10 and 12, the corresponding time rose to approximately 3 and 9 hours respectively. A bug was uncovered in the LoRa RN2483 module which contributed greatly to the time and efficiency of the firmware transmission process, particularly at the higher spreading factors. If this bug was resolved, the firmware transmission times at spreading factors 10 and 12 would be expected to decrease to approximately 1 and 5 hours respectively.

As expected, the Stop and Wait ARQ protocol proved to be much more inefficient than the Piggybacked Selective Repeat ARQ. The Piggybacked Selective Repeat ARQ protocol presented efficiencies of approximately 70% for all spreading factors except 10 and 12. The aforementioned bug however greatly contributed to the inefficiency of the transmission process for all spreading factors, and it is expected that with the resolution of this bug, the efficiency would approach the optimum efficiency.

It was found that for reliable packet reception at lower spreading factors 7, 8 and 9, gateways should be within 2 km of the node. At higher spreading factors 11 and 12 this distance can increase to 5 km. The importance of good line of sight and reasonably low noise floor were also found to contribute to successful packet reception.

In conclusion, LoRa low-power wide-area technology could be used as a means for wireless firmware transmission. Considering the nature of typical LoRa Class A devices, which may only transmit sporadically a few times a day, the impact of firmware transmission at lower spreading factors is negligible. At higher spreading factors, the impact is more considerable, however this impact may be mitigated by integrating or piggybacking firmware transmission procedures onto the sensing duties of the node.

7.1 Future Work

In this section several areas of future work are proposed and discussed based on the findings of this work.

7.1.1 Remote and Rural Areas

This work had the urban advantage of many gateways in a relatively compact area. LoRa and other LPWA technologies also target many rural applications, where gateways are not in such abundance. Hundreds of nodes may need to be served by one gateway and at great distances. A proposed area of future work is to evaluate LoRa for firmware update transmission in rural and remote environments.

7.1.2 Energy

This work did not explicitly measure energy consumption during the firmware transmission process. Typically, wireless transmission is the most energy-intensive function of a wireless sensor node. Understanding the energy consumption associated with transmitting a firmware update over LoRa is therefore a noteworthy area requiring further investigation.

7.1.3 Scalability

The maximum number of nodes a single base station can serve was briefly presented in the "Results" chapter. In this work's experiments only one node was considered, however, it is forecasted that 25% of all IoT nodes are to be enabled by LPWA technologies [10]. Therefore, a more thorough investigation into the scalability of wireless firmware updating over LoRa is suggested.

7.1.4 Firmware Application

This work only considered the transmission of a firmware image from gateway to node. The intuitive next step of research would be integrity verification of the update image and applying the new firmware. This would need to incorporate further considerations around the binary diff. algorithm used and whether the firmware update process should be single-bank or dual-bank based. A dual-bank based process would allow a node to continue performing its sensing application throughout the firmware update process, however, it comes at greater memory requirement costs, which may be constrained on many low-cost nodes.

7.1.5 LoRa Class B and Class C Devices

This work only focused on LoRa Class A devices which are intended primarily for uplink transmission. Class B and Class C devices feature more extensive downlink handling functionality. Class B devices may schedule additional receive windows, while Class C devices are in general continuously listening. In wireless firmware transmission, the downlink carrying the firmware segment is what is truly of value, and as a result, Class B and Class C devices may be capable of more efficient firmware updating than Class A devices. Evaluating LoRa firmware transmission for Class B and Class C devices is another area of proposed future work.

7.1.6 Multi-hop and Multi-cast Dissemination

LoRa is designed for single-hop communications between a node and gateway. However, potentially faster dissemination times and reduced overhead on the gateway make multi-hop and multi-cast approaches worth investigating in relation to firmware update dissemination in LoRa networks.

Bibliography

[1] Oxford Dictionaries. Telecommunication. https://en.oxforddictionaries.com/definition/telecommunication.

[2] British Express. The Spanish Armada. http://www.britainexpress.com/History/tudor/armada.htm.

[3] Aberdeen Essentials. Industry 4.0 and Industrial IoT in Manufacturing: A Sneak Peek. http://www.aberdeenessentials.com/opspro-essentials/industry-4-0-industrial-iot-manufacturing-sneak-peek/.

[4] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[5] G Jayavardhana, B Rajkumar, M Slaven, and P Marimuthu. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645 – 1660, 2013.

[6] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007(2012):1–16, 2012.

[7] James Manyika, Michael Chui, Jacques Bughin, Richard Dobbs, Peter Bisson, and Alex Marrs. *Disruptive technologies: Advances that will transform life, business, and the global economy*, volume 180. McKinsey Global Institute San Francisco, CA, 2013.

[8] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[9] Garth V Crosby and Farzam Vafa. Wireless sensor networks and lte-a network convergence. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference On*, pages 731–734. IEEE, 2013.

[10] Juha Petajajarvi, Konstantin Mikhaylov, Antti Roivainen, Tuomo Hanninen, and Marko Pettissalo. On the coverage of lpwans: range evaluation and channel attenuation model for lora technology. In *ITS Telecommunications (ITST), 2015 14th International Conference on*, pages 55–59. IEEE, 2015.

[11] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. Low power wide area networks: An overview. *IEEE Communications Surveys & Tutorials*, 19(2):855–873, 2017.

[12] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. Understanding the limits of lorawan. *IEEE Communications Magazine*, 55(9):34–40, 2017.

[13] Lte evolution for iot connectivity. Technical report, Nokia, Espoo, Finland, 2016. `https://onestore.nokia.com/asset/200178/Nokia_LTE_Evolution_for_IoT_Connectivity_White_Paper_EN.pdf`.

[14] Peter R Egli. LPWAN Technologies for Internet of Things (IoT) and M2M Scenarios. `https://www.slideshare.net/PeterREgli/lpwan/`.

[15] Stephen Brown and Cormac J Sreenan. A new model for updating software in wireless sensor networks. *IEEE Network*, 20(6), 2006.

[16] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Sergio Gusmeroli, Harald Sundmaeker, Alessandro Bassi, Ignacio Soler Jubert, Margaretha Mazura, Mark

Harrison, Markus Eisenhauer, et al. Internet of things strategic research roadmap. *Internet of Things-Global Technological and Societal Trends*, 1(2011):9–52, 2011.

[17] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, 16(1):414–454, 2014.

[18] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11, 2011.

[19] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007(2012):1–16, 2012.

[20] James Manyika, Michael Chui, Jacques Bughin, Richard Dobbs, Peter Bisson, and Alex Marrs. *Disruptive technologies: Advances that will transform life, business, and the global economy*, volume 180. McKinsey Global Institute San Francisco, CA, 2013.

[21] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[22] Qiang Wang, Yaoyao Zhu, and Liang Cheng. Reprogramming wireless sensor networks: challenges and approaches. *IEEE network*, 20(3):48–55, 2006.

[23] Bartolome Rubio, Manuel Diaz, and Jose M Troya. Programming approaches and challenges for wireless sensor networks. In *Systems and Networks Communications, 2007. ICSNC 2007. Second International Conference on*, pages 36–36. IEEE, 2007.

[24] Mei-Ling Chiang and Tsung-Lin Lu. Two-stage diff: An efficient dynamic software update mechanism for wireless sensor networks. In *Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on*, pages 294–299. IEEE, 2011.

[25] C Patauner, H Witschnig, D Rinner, A Maier, E Merlin, and E Leitgeb. High speed rfid/nfc at the frequency of 13.56 mhz. In *The First International EURASIP Workshop on RFID Technology, RFID*, pages 5–9, 2007.

[26] Antti Lahtela, Marko Hassinen, and Virpi Jylha. Rfid and nfc in healthcare: Safety of hospitals medication care. In *Pervasive Computing Technologies for Healthcare, 2008. PervasiveHealth 2008. Second International Conference on*, pages 241–244. IEEE, 2008.

[27] Erina Ferro and Francesco Potorti. Bluetooth and wi-fi wireless protocols: a survey and a comparison. *IEEE Wireless Communications*, 12(1):12–26, 2005.

[28] Lorenzo Vangelista, Andrea Zanella, and Michele Zorzi. Long-range iot technologies: The dawn of lora^{TM}. In *Future Access Enablers of Ubiquitous and Intelligent Infrastructures*, pages 51–58. Springer, 2015.

[29] Andrea Biral, Marco Centenaro, Andrea Zanella, Lorenzo Vangelista, and Michele Zorzi. The challenges of m2m massive access in wireless cellular networks. *Digital Communications and Networks*, 1(1):1–19, 2015.

[30] Jonathan de Carvalho Silva, Joel JPC Rodrigues, Antonio M Alberti, Petar Solic, and Andre LL Aquino. Lorawan—a low power wan protocol for internet of things: A review and opportunities. In *Computer and Energy Science (SpliTech), 2017 2nd International Multidisciplinary Conference on*, pages 1–6. IEEE, 2017.

[31] Nb-iot - enabling new business opportunities. Technical report, Huawei Technologies, China, 2015. `http://www.huawei.com/minisite/4-5g/img/NB-IOT.pdf`.

[32] Mobile internet of things low power wide area connectivity. Technical report, GSMA Mobile IoT, London, U.K., 2016. `https://www.gsma.com/iot/wp-content/uploads/2016/03/Mobile-IoT-Low-Power-Wide-Area-Connectivity-GSMA-Industry-Paper.pdf`.

[33] Sigfox. Sigfox's Ecosystem Delivers the World's First Ultra-Low Cost Modules to Fuel the Internet of Things Mass Market Deployment.

[34] Sigfox. Sigfox. `https://sigfox.com/`.

[35] Lukas Krupka, Lukas Vojtech, and Marek Neruda. The issue of lpwan technology coexistence in iot environment. In *Mechatronics-Mechatronika (ME), 2016 17th International Conference on*, pages 1–8. IEEE, 2016.

[36] IQRF. Technology for wireless. `https://iqrf.org/technology`.

[37] Petra Seflova, Vladimir Sulc, Jiri Pos, and Rostislav Spinar. Iqrf wireless technology utilizing iqmesh protocol. In *Telecommunications and Signal Processing (TSP), 2012 35th International Conference on*, pages 101–104. IEEE, 2012.

[38] AN Semtech. 120022, lora modulation basics, may, 2015. `https://www.semtech.com/uploads/documents/an1200.22.pdf`.

[39] Jean-Paul Bardyn, Thierry Melly, Olivier Seller, and Nicolas Sornin. Iot: The era of lpwan is starting now. In *European Solid-State Circuits Conference, ESSCIRC Conference 2016: 42nd*, pages 25–30. IEEE, 2016.

[40] Albert Pötsch and Florian Haslhofer. Practical limitations for deployment of lora gateways. In *Measurement and Networking (M&N), 2017 IEEE International Workshop on*, pages 1–6. IEEE, 2017.

[41] LoRa Alliance Technical Committee. Lorawan 1.1 specification. *LoRa Alliance*, 2017.

[42] LoRa Alliance Technical Committee. Lorawan 1.1 regional parameters. *LoRa Alliance*, 2017.

[43] Konstantin Mikhaylov, Juha Petaejaejaervi, and Tuomo Haenninen. Analysis of capacity and scalability of the lora low power wide area network technology. In *European Wireless 2016; 22th European Wireless Conference; Proceedings of*, pages 1–6. VDE, 2016.

[44] Juha Petajajarvi, Konstantin Mikhaylov, Antti Roivainen, Tuomo Hanninen, and Marko Pettissalo. On the coverage of lpwans: range evaluation and channel attenuation model for lora technology. In *ITS Telecommunications (ITST), 2015 14th International Conference on*, pages 55–59. IEEE, 2015.

[45] Carlos A Trasviña-Moreno, Rubén Blasco, Roberto Casas, and Ángel Asensio. A network performance analysis of lora modulation for lpwan sensor devices. In *Ubiquitous Computing and Ambient Intelligence*, pages 174–181. Springer, 2016.

[46] Juha Petäjäjärvi, Konstantin Mikhaylov, Matti Hämäläinen, and Jari Iinatti. Evaluation of lora lpwan technology for remote health and wellbeing monitoring. In *Medical Information and Communication Technology (ISMICT), 2016 10th International Symposium on*, pages 1–5. IEEE, 2016.

[47] Tara Petrić, Mathieu Goessens, Loutfi Nuaymi, Laurent Toutain, and Alexander Pelov. Measurements, performance and analysis of lora fabian, a real-world implementation of lpwan. In *Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium on*, pages 1–7. IEEE, 2016.

[48] Lluís Casals, Bernat Mir, Rafael Vidal, and Carles Gomez. Modeling the energy performance of lorawan. *Sensors*, 17(10):2364, 2017.

[49] Martin C Bor, Utz Roedig, Thiemo Voigt, and Juan M Alonso. Do lora low-power wide-area networks scale? In *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 59–67. ACM, 2016.

[50] Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 15–28. ACM, 2006.

[51] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. Motelab: A wireless sensor network testbed. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 68. IEEE Press, 2005.

[52] National Research Council et al. *Embedded, everywhere: A research agenda for networked systems of embedded computers*. National Academies Press, 2001.

[53] Nithya Ramanathan, Eddie Kohler, and Deborah Estrin. Towards a debugging system for sensor networks. *International Journal of Network Management*, 15(4):223–234, 2005.

[54] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97. Acm, 2002.

[55] Geoff Mainland, Laura Kang, Sebastien Lahaie, David C Parkes, and Matt Welsh. Using virtual markets to program global behavior in sensor networks. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 1. ACM, 2004.

[56] Chien-Liang Fok, G-C Roman, and Chenyang Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 653–662. IEEE, 2005.

[57] Stephen Brown and Cormac J Sreenan. Software updating in wireless sensor networks: A survey and lacunae. *Journal of Sensor and Actuator Networks*, 2(4):717–760, 2013.

[58] Milosh Stolikj, Pieter JL Cuijpers, and Johan J Lukkien. Efficient reprogramming of wireless sensor networks using incremental updates. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*, pages 584–589. IEEE, 2013.

[59] Rajesh Krishna Panta, Saurabh Bagchi, and Samuel P Midkiff. Zephyr: Efficient incremental reprogramming of sensor nodes using function call indirections and difference computation. In *Proc. of USENIX Annual Technical Conference*, 2009.

[60] The Things Network. LoRaWAN Overview. `https://www.thethingsnetwork.org/docs/lorawan/`.

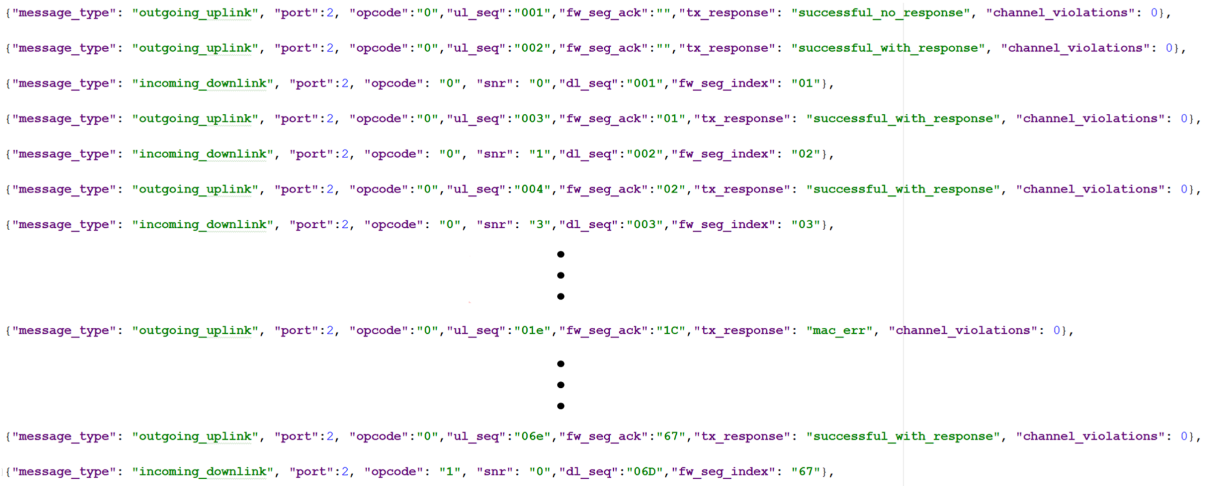
[61] jpmeijers. Rn2483-arduino-library, 2018. `https://github.com/jpmeijers/RN2483-Arduino-Library`.

[62] Pervasive Nation. Pervasive Nation - Ireland's Internet of Things Testbed. `https://connectcentre.ie/pervasive-nation/`.

[63] Connect. Connect to team with davra and orbiwise, 2018. `https://connectcentre.ie/news/connect-teams-with-davra-networks-and-orbiwise/`.

A1 Appendix

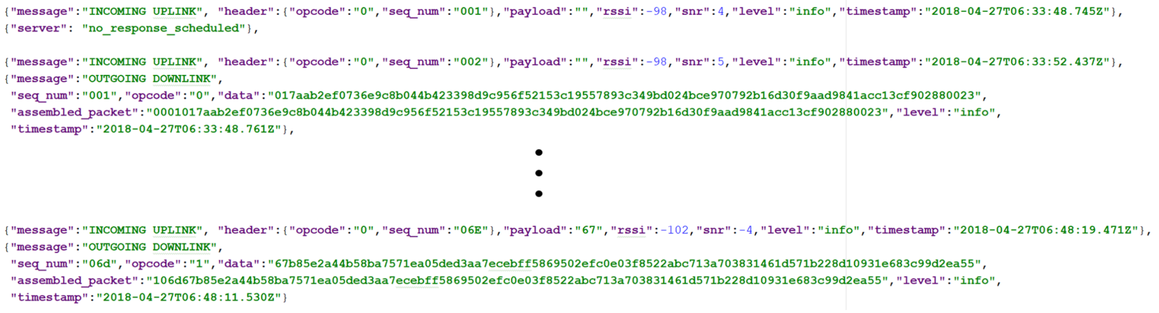
A1.1 Node Log File



```
{"message_type": "outgoing_uplink", "port":2, "opcode":"0","ul_seq":"001","fw_seg_ack":"","tx_response": "successful_no_response", "channel_violations": 0},

{"message_type": "outgoing_uplink", "port":2, "opcode":"0","ul_seq":"002","fw_seg_ack":"","tx_response": "successful_with_response", "channel_violations": 0},

{"message_type": "incoming_downlink", "port":2, "opcode": "0", "snr": "0","dl_seq":"001","fw_seg_index": "01"},

{"message_type": "outgoing_uplink", "port":2, "opcode":"0","ul_seq":"003","fw_seg_ack":"01","tx_response": "successful_with_response", "channel_violations": 0},

{"message_type": "incoming_downlink", "port":2, "opcode": "0", "snr": "1","dl_seq":"002","fw_seg_index": "02"},

{"message_type": "outgoing_uplink", "port":2, "opcode":"0","ul_seq":"004","fw_seg_ack":"02","tx_response": "successful_with_response", "channel_violations": 0},

{"message_type": "incoming_downlink", "port":2, "opcode": "0", "snr": "3","dl_seq":"003","fw_seg_index": "03"},
                                                           •
                                                           •
                                                           •
{"message_type": "outgoing_uplink", "port":2, "opcode":"0","ul_seq":"01e","fw_seg_ack":"1C","tx_response": "mac_err", "channel_violations": 0},
                                                           •
                                                           •
                                                           •
{"message_type": "outgoing_uplink", "port":2, "opcode":"0","ul_seq":"06e","fw_seg_ack":"67","tx_response": "successful_with_response", "channel_violations": 0},

{"message_type": "incoming_downlink", "port":2, "opcode": "1", "snr": "0","dl_seq":"06D","fw_seg_index": "67"},
```

Figure A1.1: Extract from Node Firmware Transmission Log File

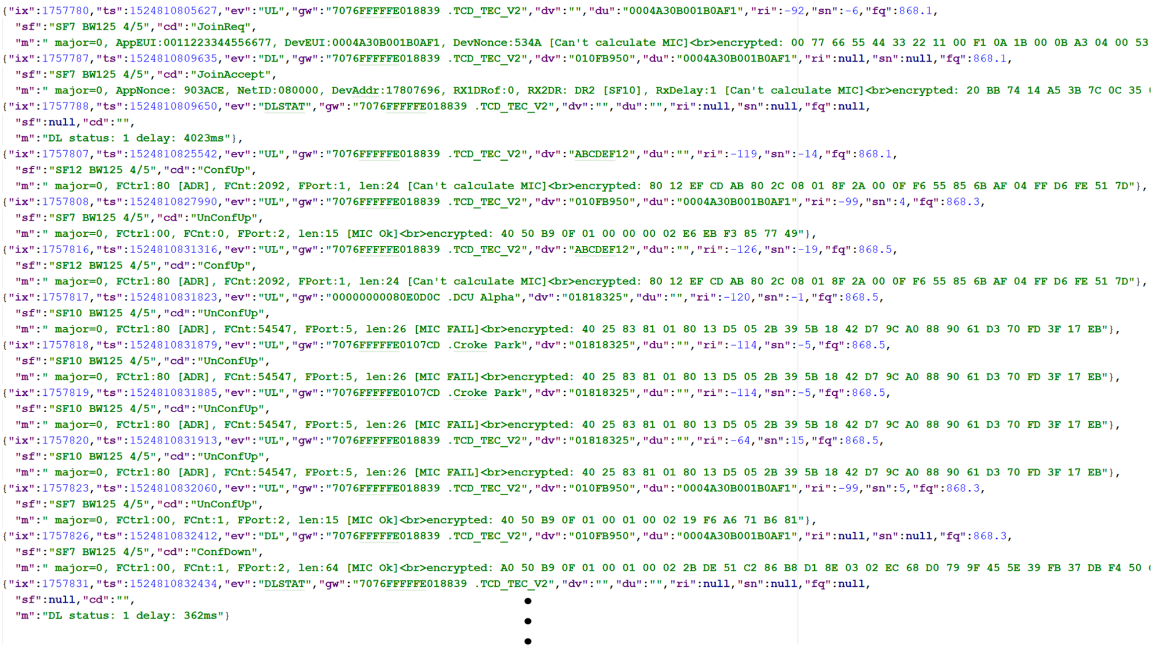
A1.2 Application Server Log



```
{"message":"INCOMING UPLINK", "header":{"opcode":"0","seq_num":"001"},"payload":"","rssi":-98,"snr":4,"level":"info","timestamp":"2018-04-27T06:33:48.745Z"},
{"server": "no_response_scheduled"},

{"message":"INCOMING UPLINK", "header":{"opcode":"0","seq_num":"002"},"payload":"","rssi":-98,"snr":5,"level":"info","timestamp":"2018-04-27T06:33:52.437Z"},
{"message":"OUTGOING DOWNLINK",
 "seq_num":"001","opcode":"0","data":"017aab2ef0736e9c8b044b423398d9c956f52153c19557893c349bd024bce970792b16d30f9aad9841acc13cf902880023",
 "assembled_packet":"0001017aab2ef0736e9c8b044b423398d9c956f52153c19557893c349bd024bce970792b16d30f9aad9841acc13cf902880023","level":"info",
 "timestamp":"2018-04-27T06:33:48.761Z"},

                                                          •
                                                          •
                                                          •

{"message":"INCOMING UPLINK", "header":{"opcode":"0","seq_num":"06E"},"payload":"67","rssi":-102,"snr":-4,"level":"info","timestamp":"2018-04-27T06:48:19.471Z"},
{"message":"OUTGOING DOWNLINK",
 "seq_num":"06d","opcode":"1","data":"67b85e2a44b58ba7571ea05ded3aa7ecebff5869502efc0e03f8522abc713a703831461d571b228d10931e683c99d2ea55",
 "assembled_packet":"106d67b85e2a44b58ba7571ea05ded3aa7ecebff5869502efc0e03f8522abc713a703831461d571b228d10931e683c99d2ea55","level":"info",
 "timestamp":"2018-04-27T06:48:11.530Z"}
```

Figure A1.2: Extract from Application Server Log File

A1.3 Gateway Log



```
{"ix":1757780,"ts":1524810805627,"ev":"UL","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"","du":"0004A30B001B0AF1","ri":-92,"sn":-6,"fq":868.1,
  "sf":"SF7 BW125 4/5","cd":"JoinReq",
  "m":" major=0, AppEUI:0011223344556677, DevEUI:0004A30B001B0AF1, DevNonce:534A [Can't calculate MIC]<br>encrypted: 00 77 66 55 44 33 22 11 00 F1 0A 1B 00 0B A3 04 00 53
{"ix":1757787,"ts":1524810809635,"ev":"DL","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"010FB950","du":"0004A30B001B0AF1","ri":null,"sn":null,"fq":868.1,
  "sf":"SF7 BW125 4/5","cd":"JoinAccept",
  "m":" major=0, AppNonce: 903ACE, NetID:080000, DevAddr:17807696, RX1DRof:0, RX2DR: DR2 [SF10], RxDelay:1 [Can't calculate MIC]<br>encrypted: 20 BB 74 14 A5 3B 7C 0C 35
{"ix":1757788,"ts":1524810809650,"ev":"DLSTAT","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"","du":"","ri":null,"sn":null,"fq":null,
  "sf":null,"cd":"",
  "m":"DL status: 1 delay: 4023ms"},
{"ix":1757807,"ts":1524810825542,"ev":"UL","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"ABCDEF12","du":"","ri":-119,"sn":-14,"fq":868.1,
  "sf":"SF12 BW125 4/5","cd":"ConfUp",
  "m":" major=0, FCtrl:80 [ADR], FCnt:2092, FPort:1, len:24 [Can't calculate MIC]<br>encrypted: 80 12 EF CD AB 80 2C 08 01 8F 2A 00 0F F6 55 85 6B AF 04 FF D6 FE 51 7D"},
{"ix":1757808,"ts":1524810827990,"ev":"UL","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"010FB950","du":"0004A30B001B0AF1","ri":-99,"sn":4,"fq":868.3,
  "sf":"SF7 BW125 4/5","cd":"UnConfUp",
  "m":" major=0, FCtrl:00, FCnt:0, FPort:2, len:15 [MIC Ok]<br>encrypted: 40 50 B9 0F 01 00 00 00 02 E6 EB F3 85 77 49"},
{"ix":1757816,"ts":1524810831316,"ev":"UL","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"ABCDEF12","du":"","ri":-126,"sn":-19,"fq":868.5,
  "sf":"SF12 BW125 4/5","cd":"ConfUp",
  "m":" major=0, FCtrl:80 [ADR], FCnt:2092, FPort:1, len:24 [Can't calculate MIC]<br>encrypted: 80 12 EF CD AB 80 2C 08 01 8F 2A 00 0F F6 55 85 6B AF 04 FF D6 FE 51 7D"},
{"ix":1757817,"ts":1524810831823,"ev":"UL","gw":"00000000080E0D0C .DCU Alpha","dv":"01818325","du":"","ri":-120,"sn":-1,"fq":868.5,
  "sf":"SF10 BW125 4/5","cd":"UnConfUp",
  "m":" major=0, FCtrl:80 [ADR], FCnt:54547, FPort:5, len:26 [MIC FAIL]<br>encrypted: 40 25 83 81 01 80 13 D5 05 2B 39 5B 18 42 D7 9C A0 88 90 61 D3 70 FD 3F 17 EB"},
{"ix":1757818,"ts":1524810831879,"ev":"UL","gw":"7076FFFFFE0107CD .Croke Park","dv":"01818325","du":"","ri":-114,"sn":-5,"fq":868.5,
  "sf":"SF10 BW125 4/5","cd":"UnConfUp",
  "m":" major=0, FCtrl:80 [ADR], FCnt:54547, FPort:5, len:26 [MIC FAIL]<br>encrypted: 40 25 83 81 01 80 13 D5 05 2B 39 5B 18 42 D7 9C A0 88 90 61 D3 70 FD 3F 17 EB"},
{"ix":1757819,"ts":1524810831885,"ev":"UL","gw":"7076FFFFFE0107CD .Croke Park","dv":"01818325","du":"","ri":-114,"sn":-5,"fq":868.5,
  "sf":"SF10 BW125 4/5","cd":"UnConfUp",
  "m":" major=0, FCtrl:80 [ADR], FCnt:54547, FPort:5, len:26 [MIC FAIL]<br>encrypted: 40 25 83 81 01 80 13 D5 05 2B 39 5B 18 42 D7 9C A0 88 90 61 D3 70 FD 3F 17 EB"},
{"ix":1757820,"ts":1524810831913,"ev":"UL","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"01818325","du":"","ri":-64,"sn":15,"fq":868.5,
  "sf":"SF10 BW125 4/5","cd":"UnConfUp",
  "m":" major=0, FCtrl:80 [ADR], FCnt:54547, FPort:5, len:26 [MIC FAIL]<br>encrypted: 40 25 83 81 01 80 13 D5 05 2B 39 5B 18 42 D7 9C A0 88 90 61 D3 70 FD 3F 17 EB"},
{"ix":1757823,"ts":1524810832060,"ev":"UL","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"010FB950","du":"0004A30B001B0AF1","ri":-99,"sn":5,"fq":868.3,
  "sf":"SF7 BW125 4/5","cd":"UnConfUp",
  "m":" major=0, FCtrl:00, FCnt:1, FPort:2, len:15 [MIC Ok]<br>encrypted: 40 50 B9 0F 01 00 01 00 02 19 F6 A6 71 B6 81"},
{"ix":1757826,"ts":1524810832412,"ev":"DL","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"010FB950","du":"0004A30B001B0AF1","ri":null,"sn":null,"fq":868.3,
  "sf":"SF7 BW125 4/5","cd":"ConfDown",
  "m":" major=0, FCtrl:00, FCnt:1, FPort:2, len:64 [MIC Ok]<br>encrypted: A0 50 B9 0F 01 00 01 00 02 2B DE 51 C2 86 B8 D1 8E 03 02 EC 68 D0 79 9F 45 5E 39 FB 37 DB F4 50
{"ix":1757831,"ts":1524810832434,"ev":"DLSTAT","gw":"7076FFFFFE018839 .TCD_TEC_V2","dv":"","du":"","ri":null,"sn":null,"fq":null,
  "sf":null,"cd":"",                                        •
  "m":"DL status: 1 delay: 362ms"}                          •
                                                            •
```

Figure A1.3: Extract from Gateway Log File