

**PERFORMANCE STUDY OF IOT CONNECTIVITY: LORA
NETWORK**

SIOW JIA YIE


**A project report submitted in partial fulfilment of the
requirements for the award of Bachelor of Engineering
(Honours) Electronic and Communications Engineering**

**Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman**

April 2020

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :  _____

Name : Siow Jia Yie _____

ID No. : 1505254 _____

Date : 15/5/2020 _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**PERFORMANCE STUDY OF IOT CONNECTIVITY: LORA NETWORK**” was prepared by **SIOW JIA YIE** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electronic and Communications Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature

:



Supervisor

:

Ir Dr. Tham Mau Luen

Date

:

17/5/2020

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2020, Siow Jia Yie. All right reserved.

ACKNOWLEDGEMENTS

I am thankful to everyone who had contributed to the successful completion of this project. First of all, I would like to express my special thanks of gratitude to my Final Year Project (FYP) supervisor Ir Dr. Tham Mau Luen for advising me with his selfless guidance and help in completing this project.

In addition, I want to thank my parents and friends who support and encourage me throughout this project. They gave me significant amount of encouragements that supported me to complete this project successfully.

ABSTRACT

The deployment of wireless sensor networks (WSNs) yields a convenient and economical solution for collecting information automatically. However, there are several limitations such as wide geographic area, bandwidth scarcity, lack of network infrastructure and shortage of power supply. Long range (LoRa) is one of the most promising low power wide area network (LPWAN) technologies that allow long-range transmissions. LoRa can be classified into two parts, namely LoRa at physical layer and Long Range Wide Area Network (LoRaWAN) at data link layer. Such architecture motivates the study of two use cases in this thesis. In the first scenario, LoRa gateway and node are implemented in order to evaluate the system performance of LoRaWAN in an indoor environment. The impacts on airtime, received signal strength indicator (RSSI) and signal-to-noise ratio (SNR) are explored under different spreading factors (SF) and transmission distances. In the second scenario, the LoRaWAN environment is shifted to Long Range Peer to Peer (LoRa P2P) image transmission that involves only the LoRa physical layer. Two nodes are set up in this scenario. Constrained by LoRa maximum transmission unit (MTU), images are segmented into multiple packets, which are transferred by either stop-and-wait protocol or LoRa Multi-Packet Transmission Protocol. As in the first scenario, similar performance metrics are assessed. Experimental results from these two studies reveal that LoRa transmissions using LoRaWAN and LoRa P2P are feasible, as long as the distance is within 35m in an indoor environment.

TABLE OF CONTENTS

DECLARATION		i
APPROVAL FOR SUBMISSION		ii
ACKNOWLEDGEMENTS		iv
ABSTRACT		v
TABLE OF CONTENTS		vi
LIST OF TABLES		ix
LIST OF FIGURES		x
LIST OF SYMBOLS / ABBREVIATIONS		xiv
LIST OF APPENDICES		xvi
CHAPTER		
1	INTRODUCTION	1
1.1	General Introduction	1
1.2	Importance of the Study	2
1.3	Problem Statement	2
1.4	Aim and Objectives	3
1.5	Scope and Limitation of the Study	3
1.6	Contribution of the Study	4
1.7	Outline of the Report	4
2	LITERATURE REVIEW	5
2.1	Introduction	5
2.2	LPWAN standards	5
2.2.1	NB-IoT	5
2.2.2	Sigfox	6
2.2.3	LoRa	6
2.3	Comparison of LPWAN standards	8
2.4	Related work	8
2.5	Summary	10
3	LORAWAN	10

3.1	Introduction	11
3.2	Selection of frequency	11
3.3	Equipment used	11
3.4	LoRa Gateway setup	16
3.5	LoRa Node setup	18
3.6	LoRa Packet Structure	27
3.7	LoRaWAN network architecture	28
3.8	Performance evaluation of LoRaWAN	29
	3.8.1 The impact on airtime under different SFs and payload lengths	29
	3.8.2 The impacts on RSSI and SNR under different transmission distances	31
4	LORA P2P	36
4.1	Introduction	36
4.2	Equipment used	36
4.3	LoRa P2P Packet Structure	38
4.4	Design of Transmission Protocol	38
	4.4.1 Stop and Wait Protocol	40
	4.4.2 LoRa Multi-Packet Transmission Protocol	42
4.5	Image Processing	47
	4.5.1 Image Compression	47
	4.5.2 Image Conversion	49
	4.5.3 Image Building	51
4.6	Design of Transmission Protocol	51
4.7	Performance evaluation of LoRaP2P	51
	4.7.1 Stop and wait transmission protocol vs. LoRa Multi-Packet Transmission Protocol	52
	4.7.2 The impacts on number of packet loss, transmission time, RSSI and SNR under different transmission distances	54
	4.7.3 The impacts on number of packet loss and transmission time when using different image qualities	59
5	CONCLUSIONS AND RECOMMENDATIONS	62

5.1	Conclusions	62
5.2	Recommendations for future work	63
REFERENCES		64
APPENDICES		67

LIST OF TABLES

Table 2.1: Three classes of LoRaWAN.	7
Table 3.1: Raspberry Pi Zero W specifications.	13
Table 3.2: RAK831 concentrator board specifications.	14
Table 3.3: RAK811 LoRa Module Specifications.	15
Table 3.4: AT command used for configuration.	25
Table 3.5: Number of DR of RAK811 LoRa module.	27
Table 3.6: LoRa configuration paramaters.	30
Table 3.7: Airtime (in milliseconds) for different SFs and payload lengths.	30
Table 3.8: Configuration parameters.	33
Table 3.9: Results of the impacts on RSSI and SNR under different transmission distances.	34
Table 4.1: Dragino LoRa module specifications.	37
Table 4.2: 5MP Camera Board for Raspberry Pi specifications.	38
Table 4.3: Results of image conversion.	50
Table 4.4: Configuration parameters.	53
Table 4.5: Average transmission time when using stop and wait protocol and LoRa Multi-Packet Transmission Protocol.	54
Table 4.6: Configuration parameters.	55
Table 4.7: Results of the impacts on number of packet loss, transmission time, RSSI and SNR under different transmission distances.	57
Table 4.8: Configuration parameters.	59
Table 4.9: Results of effect of quality of image on number of packet loss and transmission time.	60

LIST OF FIGURES

Figure 2.1: Summary of Sigfox, NB-IoT and LoRa (Mekki et al., 2019).	8
Figure 3.1: LoRaWAN frequency plan of AS920-923 (TheThingsNetwork, n. d.).	11
Figure 3.2: Raspberry Pi Zero W.	12
Figure 3.3: RAK831 concentrator board.	14
Figure 3.4: RAK811 LoRa Module.	15
Figure 3.5: LoRa gateway setup.	16
Figure 3.6: Checking of LoRa gateway's IP address.	17
Figure 3.7: LoRa gateway is accessed through SSH using PuTTY.	17
Figure 3.8: Gateway ID.	17
Figure 3.9: TTN console page after registration of gateway.	18
Figure 3.10: Boot mode for RAK811 Wisnode.	18
Figure 3.11: Laptop's USB port and RAK811 Wisnode connection.	19
Figure 3.12: RST button on RAK811 WisNode.	19
Figure 3.13: UART, port, baudrate, and parity configuration in STM32CubeProgrammer.	20
Figure 3.14: Correct log in STM32CubeProgrammer.	20
Figure 3.15: Data on RAK811 WisNode is erased and new file is opened.	20
Figure 3.16: Bootloader file (RAK811_BOOT_V3.0.2.bin) is loaded and downloaded to RAK811.	21
Figure 3.17: Message of file download complete.	21
Figure 3.18: "BOOT" pin and the "GND" pin connection.	22

Figure 3.19: Showing of BOOT MODE of RAK811 in RAK serial port tool.	22
Figure 3.20: Burning of firmware into RAK811 using RAK LoRaButton Upgrade Tool V1.0.	23
Figure 3.21: Log data when the RST button of RAK811 Wisnode is pressed.	23
Figure 3.22: Registration of application in The Thing Networks.	23
Figure 3.23: Registration of device in The Thing Networks.	24
Figure 3.24: Configuration parameters in The Thing Networks.	24
Figure 3.25: Configuration of RAK811 Wisnode by using RAK Serial Port Tool.	25
Figure 3.26: Joining LoRa network by using AT command “at+join”.	26
Figure 3.27: Sending data with AT command “at+send=lora:2:12345678”.	26
Figure 3.28: Gateway traffic shown in The Thing Networks.	27
Figure 3.29: LoRa application traffic in The Thing Networks.	27
Figure 3.30: LoRa Packet Structure.	28
Figure 3.31: LoRa Packet Structure explanations.	28
Figure 3.32: Standard network architecture of LoRaWAN.	29
Figure 3.33: Network architecture of LoRaWAN in this study.	29
Figure 3.34: Airtime against payload length with different SFs.	31
Figure 3.35: Building of Evergreen Scotpine Condominium.	32

Figure 3.36: Floor map of Evergreen Scotpine Condominium.	32
Figure 3.37: Floor map (Gateway).	34
Figure 3.38: Gateway at G floor.	34
Figure 3.39: Floor map (LoRa Wisnode).	34
Figure 3.40: Laptop and RAK811 LoRa Module at 3 rd floor.	34
Figure 4.1: Dragino LoRa module.	37
Figure 4.2: 5MP Camera Board for Raspberry Pi.	38
Figure 4.3: A single packet.	39
Figure 4.4: C language and corresponding description.	39
Figure 4.5: Adding header.	40
Figure 4.6: Stop-and-wait protocol.	40
Figure 4.7: Stop and wait protocol's transmitter coding flowchart.	41
Figure 4.8: Stop and wait protocol's receiver coding flowchart.	42
Figure 4.9: LoRa Multi-Packet Transmission Protocol.	43
Figure 4.10: LoRa Multi-Packet Transmission Protocol with timeout session.	44
Figure 4.11: LoRa Multi-Packet Protocol with resending request.	44
Figure 4.12: LoRa Multi-Packet Transmission Protocol's transmitter coding flowchart.	45
Figure 4.13: LoRa Multi-Packet Transmission Protocol's receiver flowchart.	46
Figure 4.14: 3280 x 2464 full-resolution image (ImageTx.jpg).	47
Figure 4.15: 240 x 160 pixel resized image.	48
Figure 4.16: Quality: 10, Size = 2KB, PSNR = 25.58 dB.	49

Figure 4.17: Quality: 25, Size = 4KB, PSNR = 29.75 dB.	49
Figure 4.18: Quality: 50, Size = 7KB, PSNR = 28.12 dB.	49
Figure 4.19: Quality: 95, Size = 24KB, PSNR = 32.12 dB.	49
Figure 4.20: LoRa P2P transmitter.	52
Figure 4.21: LoRa P2P receiver.	52
Figure 4.22: VNC viewer monitoring and controlling.	53
Figure 4.23: Floor map of Evergreen Scotpine Condominium (Receiver).	56
Figure 4.24: Placement of receiver at G floor.	56
Figure 4.25: Floor map of Evergreen Scotpine Condominium (Transmitter).	56
Figure 4.26: Placement of transmitter at 1 st floor.	56
Figure 4.27: Transmitter monitoring using mobile SSH through SSH connection.	56
Figure 4.28: Receiver monitoring using VNC viewer in laptop through SSH connection.	57

LIST OF SYMBOLS / ABBREVIATIONS

ABP	Authentication By Personalisation
ACK	Acknowledgement
BLE	Bluetooth Low Energy
BPSK	Binary Phase Shift Keying
CSI	Camera Serial Interface
CRC	Cyclic Redundancy Check
CSS	Chirp Spread Spectrum
EDs	End-devices
FFC	Flat Flex Connectors
FSK	Frequency shift keying
GFSK	Gaussian frequency shift keying
GSM	Global System for Mobile Communications
HDMI	High-Definition Multimedia Interface
IoT	Internet of Things
IP	Internet Protocol
ISM	Industrial, Scientific, and Medical
JPEG	Joint Photographic Experts Group
LoRa	Long range
LoRa P2P	Long range Peer to Peer
LoRaWAN	Long range wide-area network
LOS	Line of sight
LPWAN	Low Power Wide Area Network
LTE	Long-term evolution
M2M	Machine to Machine
MAC	Medium Access Control
MTU	Maximum Transmission Unit
NB-IoT	Narrowband Internet of Things
OOK	On-off keying
OS	Operating System
OTAA	Over-the-Air Activation
PC	personal computer
PIL	Python Imaging Library

PSNR	peak signal-to-noise ratio
P2P	peer-to-peer
RFID	radio-frequency identification
RPI	Raspberry Pi
RSSI	Received Signal Strength Indicator
RX	Receiver
SD	Secure Digital
SNR	Signal-to-noise ratio
SPI	Serial Peripheral Interface
SSH	Secure Shell
TTN	The Things Network
TX	Transmitter
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
Wi-Fi	Wireless Fidelity
WSNs	Wireless Sensor Networks
WUSN	Wireless Underground Sensor Networks

LIST OF APPENDICES

APPENDIX A: Raspberry Pi Zero W Datasheet	67
APPENDIX B: RAK 831 datasheet	68
APPENDIX C: RAK 811 datasheet	79
APPENDIX D: Manual to write LoRa gateway image to Micro SD	81
APPENDIX E: Registration of gateway in TTN	85
APPENDIX F: Dragino LoRa GPS HAT Single Channel LoRa & GPS modules Datasheet	88
APPENDIX G: SX1276 datasheet	94
APPENDIX H: Stop and Wait Protocol Transmitter codes	110
APPENDIX I: Stop and Wait Protocol Receiver code	125
APPENDIX J: LoRa Multi-Packet Transmission Protocol Transmitter code	141
APPENDIX K: LoRa Multi-Packet Transmission Protocol Receiver code	157
APPENDIX L: Header adding python code (addheader.py) in transmitter	174
APPENDIX M: Image compress python code (compress.py) in transmitter	176
APPENDIX N: Conversion of image to hexadecimal format python code (storebytes.py) in transmitter	177
APPENDIX O: Conversion of hexadecimal data to image and dropbox uploading python code (hextoimage.py) in receiver	178
APPENDIX P: Time printing python code (timezone.py) in both transmitter and receiver	180

CHAPTER 1

INTRODUCTION

1.1 General Introduction

The applications and interest for Internet of Things (IoT) have increased significantly in this century. Business consultancy group McKinsey and Gartner indicates that connected devices reached up to 26.7 billion nationwide in 2019 (Security, 2020). The connected devices are projected to increase by three times in 2020. A report from Strategy Analytics predicts that connected devices will increase to 38.6 billion in 2025 and grow up to 50 billion in 2030 (Help Net Security, 2019).

There are different types of wireless communication technologies such as Bluetooth, RFID, ZigBee, Wi-Fi, 2G/3G/4G, which can be used to support IoT network. However, these technologies are inadequate when taking network coverage, power consumption and implementation cost into consideration (Rubio-Aparicio et al., 2019).

Low Power Wide Area Networks (LPWANs) start to attract the attention of public with the trending of IoT as they offer functionalities to IoT devices to send packets for wide communication ranges with optimal energy consumption and appropriate for large scale deployments (Ayoub et al., 2018). LPWAN technologies utilize a relatively low frequency such as 433 MHz, 868 MHz and 915 MHz. This allows a larger reliability against unwanted signal while using lower power as compared to typical Wi-Fi which functions at 2.4 GHz frequency bands (Muaz Abdul Rahman et al., 2018). In line of sight (LOS) condition, LPWAN technologies can reach up to range of 15 km.

LPWAN technologies are classified into two groups, cellular and non-cellular. Cellular are not the main choices of the designers because they use licensed bands which have high initialisation fees and the users are restricted by operators when using it. Non-cellular technologies such as Long Range (LoRa) and Sigfox are more desirable because they use unlicensed bands which are free and the users are independent to use it (Dogan et al., 2019). Long Range Wide Area Network (LoRaWAN) caught the attention of researchers, communities and organizations among all these LPWAN technologies because LoRaWAN is

widely recognized to have the best capability for providing the LPWAN technology to various applications of IoT.

LoRa can be introduced two different parts which are the physical layer, Semtech's patented modulation technique based on the modified form of chirp spread spectrum (CSS) and LoRaWAN data link layer protocol designed and standardized by LoRa Alliance (Masadan et al., 2018). Using CSS technology, LoRa has more robustness against degradation while supporting large numbers of devices and having long communication range (Ahmad et al., 2018). Meanwhile, LoRaWAN uses LoRa modulation in physical layer. A LoRaWAN system typically made up of nodes connected to IoT system via a gateway. The gateway is used to let the nodes to connect the network server and the internet. The node acts as a transducer that will output appropriate electrical signal based on triggered events.

1.2 Importance of the Study

The performance analysis for indoor environment of LoRaWAN will be discussed in this study for possible applications and future research areas. Besides that, this study discusses the limitation in image transmission by using LoRa and how image transmission can be done in a peer-to-peer (P2P) network model.

1.3 Problem Statement

For the implementation of IoT, the most famous technologies used are Wi-Fi and cellular previously. Although the high data rate can be achieved with Wi-Fi but Wi-Fi has high power usage and only allows short range communication. On the other hand, cellular has the characteristics of high data rate and long-range communication but cellular encounter problems from high power consumption and high implementation cost (Vatcharathiansakul et al., 2017).

New explorations within the IoT requires additional communication technologies that can provide better IoT implementation in terms of cost, power, range and complexity. The power usage profile of the IoT end devices (EDs) should be carefully planned to expand the battery's lifetime as they are mostly battery-operated sensor nodes. As EDs are spread over large operation area, communication range need to be increased up to several km. Taking everything

into account, this can be only be realized by using LPWAN technologies. Although some LPWAN technologies emerge, LoRaWAN is the promising technology to overcome these issues.

1.4 Aim and Objectives

In this project, the objectives are to implement the LORA gateway, to connect the gateway to LORA nodes, and to evaluate the system performance of LoRaWAN in an indoor environment. This experiment consists of one gateway, one ED, and one network server. The gateway is implemented by connecting Raspberry Pi Zero W to LoRa gateway concentrator module RAK831 via Serial Peripheral Interface (SPI) interface. At the same time, the ED is made up of RAK811 LoRa module. The AT command is sent from the serial port tool in the laptop to configure RAK811 LoRa module. The performance evaluation will be studied in terms of airtime, signal-to-noise ratio (SNR) and received signal strength indicator (RSSI) under different spreading factors (SFs) and transmission distances.

Besides that, this project aims to build a Long Range Peer to Peer (LoRa P2P) communication system to do image transmission and to study the system performance of LoRa P2P in an indoor environment in term of transmission time, number of packet loss, RSSI and SNR.

1.5 Scope and Limitation of the Study

This is a two-part study. For Part 1 of study, the proposed prototype consists of a gateway and a node. The focus of this study is on the performance evaluation of system in terms of airtime spent when using different SFs and RSSI and SNR of signal when placing the end node RAK811 at different locations. For the second part, the investigations about the image transmission using LoRa P2P will be carried out. The prototype in Part 2 consists of a two LoRa nodes. The transmission protocol that is suitable for image transmission will be studied. The performance study in terms of number of packet loss, transmission time, RSSI and SNR for LoRa P2P image transmission also will be focused.

Due to the hardware limitations of end node RAK811, for the LoRaWAN performance evaluation, only spreading factors of SF7, SF8, SF9 and SF10 can be tested in this study. There are also some issues experienced

when testing the RAK811 with Arduino board due to immaturity of the hardware and the developer still debugs and updates the firmware for better compatibility. In addition, the performance study of LoRa at outdoor environment and long-range LoRa transmission cannot be conducted due to covid-19 lockdown during the performance testing of this study. Due to time and cost constraint, only one node and one gateway are set up for LoRaWAN, more nodes should be deployed for more variations of evaluation.

1.6 Contribution of the Study

As far as we know, the actual performance of LoRaWAN has rarely been studied in local. Most of them are carried out in simulation, which can't emulate the realistic behaviours of the actual network. The challenge is further magnified when the scenario is image transmission. This work aims to bridge this gap by investigate the real performance of both LoRaWAN and LoRa P2P image transmission. A lightweight transmission protocol is designed for reliable image transmission in order to shorten the transmission time. Overall, the findings are deemed valuable as they may serve as guidelines for local deployment.

1.7 Outline of the Report

This report includes five chapters. The introduction about this study is covered in Chapter 1. Chapter 2 presents the related work of LoRa's performance analysis and applications done by previous researchers. Chapter 3 and Chapter 4 includes the methodology, results and discussion for LoRaWAN and LoRa P2P respectively. Last but not least, the conclusion about this study and recommendation for improvement of system are included in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, famous standards of LPWAN such as Narrowband Internet of Things (NB-IoT), LoRa and Sigfox are introduced. Besides that, the related work of LoRa applications and performance evaluations are discussed.

2.2 LPWAN standards

NB-IoT, LoRa and Sigfox are three well-known technologies used in large scale connectivity of IoT devices for long-range transmissions. The technical differences of NB-IoT, LoRa and Sigfox are explained in this chapter.

2.2.1 NB-IoT

NB-IoT is a modern cellular technology introduced in June 2016 to provide wide range coverage for IoT. It was designed to offer outstanding coexistence performance with LTE (long-term evolution) and GSM (global system for mobile communications) under licensed frequency bands (Wang et al., 2017). It allows massive connectivity of IoT devices, high power efficiency, low bit rate and latency smaller than 10 seconds. NB-IoT requires 180 kHz frequency bandwidth, which is compatible to the size of one resource block in GSM and LTE. NB-IoT can be worked in three operations modes: stand-alone, guard-band, and in-band.

- Stand-alone: Utilizing the currently used GSM frequencies bands
- Guard-band: Utilizing the unused resources blocks reserved in the guard-band of LTE carrier
- In-Band: Utilizing the resources blocks reserved in the LTE carrier

NB-IoT uses frequency division multiple access (FDMA) and the orthogonal frequency division multiple access (OFDMA) in the downlink and uplink respectively. In the uplink, it uses with single carrier, and utilizes the quadrature phase shift keying modulation (QPSK) (Wang et al., 2017). For the downlink, there is 200 kbps data rate. For the uplink, there is 20 kbps data rate.

The maximum length of each payload is 1600 bytes (Mekki et al., 2019). Adhikary, Lin and Eric Wang (2017) discussed that ten years of battery lifetime can be offered for NB-IoT when transmitting average 200 bytes per day.

2.2.2 Sigfox

Sigfox is a cellular system approach which allows EDs use the Binary Phase Shift Keying (BPSK) modulation to connect to base stations (Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara, 2017). Sigfox uses unlicensed Industrial, Scientific, and Medical (ISM) bands which are 433 MHz, 868 MHz, and 915 MHz. Sigfox concentrates the radiation in an ultra-narrow band and undergoes low levels of signal interference. This leading to small power consumption, low cost design of antenna, and high sensitivity of receiver (RX) in order to achieve maximum throughput of 100 bps. The maximum uplink and downlink payload lengths are 12 bytes and 8 bytes respectively. This may causes it to face difficulty when sending and receiving large data sizes on various IoT applications. Its coverage is about 3 to 10 km in city and about 30 to 50 km in countryside. Sigfox only supported uplink communication in the beginning. After that, Sigfox developed to significant link asymmetry two-way communication technology. The downlink communication only can be done before uplink communication after each the ED must wait to obtain a response from the base station which suitable for data collection but inappropriate for controlling and commanding. (Mekki et al., 2019).

2.2.3 LoRa

LoRa is a technology which covers physical layer and offers low power wireless transmission with low data rate and long range. It operates within unlicensed band and it is a technology that performs the modulation of signals in the sub-GHz ISM band using CSS that the signal is spread with narrowband over a broader channel bandwidth. Low levels of noise levels can be achieved by the signal. It also consists of high interference resilience. Therefore, it is difficult for detecting and jamming (Reynders et al., 2016). The symbol rate and the number of bits per symbol depend on SF could change between 7 and 12 to achieve the data rate and range trade-off. The synchronous transmission utilizing different SFs is possible in a similar frequency channel because of the

orthogonal spreading codes for different SFs. Communication range of LoRa is limited for different environments. For instance, the range from 2 to 5 km in city and 45 km in countryside is possible. Depending on SF and bandwidth of channel, the achievable rate of data is between 290 bps and 50 kbps (De Carvalho Silva et al., 2017). The transmission's payload can range from 2 to 255 octets. When aggregation of channel is used, the rate of data can achieve up to 50 kbps (Widianto et al., 2019).

LoRaWAN is a data link layer protocol developed by LoRa Alliance in 2015. LoRaWAN uses LoRa modulation in the physical layer. EDs will transmit data to gateways that are connected to network server over a single wireless hop. Three classes are defined by LoRaWAN with different functionalities. Class A devices are able to do bidirectional communication, have least power consumption as they have longest sleeping time. Class A devices only can receive data after they had sent message, thus having higher latency. Class B devices are capable to do bidirectional communication, can receive data with scheduled time slot, thus increasing the downlink possibilities. Class B devices' receiving time slots also can be synchronized with beacon frames sent by gateway. Class C devices have bidirectional communication capabilities, but they have highest power consumption because they open the receiving windows all the time, thus providing lowest latency. The only time Class C devices cannot receive data is when they are transmitting information (Lavric and Popa, 2017).

Table 2.1: Three classes of LoRaWAN.

Class	Energy Consumption	Description
A	Lowest	Downlink after transmission
B	Efficient with scheduled downlink	Scheduled downlink with beacon frames or random time slots
C	Highest	Devices listen non-stop. Shortest downlink latency.

2.3 Comparison of LPWAN standards

The features of Sigfox, NB-IoT and LoRa are outlined as shown in Figure 2.1.

	Sigfox	NB-IOT	LoRaWAN
Modulation	BPSK	QPSK	CSS
Frequency	Unlicensed ISM bands	Licensed LTE frequency	Unlicensed ISM bands
Bandwidth	100 Hz	200 kHz	250kHz and 125kHz
Maximum messages/day	140 (Uplink), 4(Downlink)	Unlimited	Unlimited
Maximum payload length	12 bytes (Uplink), 8 bytes (Downlink)	1600 bytes	243 bytes
Range	10 km (urban), 40 km (rural)	1 km (urban), 10 km (rural)	5 km (urban), 20 km (rural)
Localization	Yes, Received Signal Strength Indicator (RSSI)	No (under specification)	Yes, Time Difference Of Arrival (TDOA)
Adaptive data rate	No	No	Yes
Allow private network	No	No	Yes
Interference immunity	Very high	Low	Very high
Authentication & encryption	Not supported	Yes (LTE encryption)	Yes (AES 128b)
Standardization	Sigfox company is collaborating with ETSI on the standardization of Sigfox-based network	3GPP	LoRa-Alliance

Figure 2.1: Summary of Sigfox, NB-IoT and LoRa (Mekki et al., 2019).

2.4 Related work

Lavric and Popa (2017) evaluated the LoRa technology to overcome the barriers that hamper the growth of the IoT such as connectivity, energy management, security and complexity. The authors proposed that LoRa is a modulation technique that provides the wide range information transmission and is an appropriate solution in overcoming the IoT challenges. To enhance the performance of uplink in LoRa networks, the use of spatial and time diversity through numerous receive antennas and replication of message were analysed by Hoeller *et al.* (2018), they concluded that low density networks replication and numerous receive antennas are favourable. A evaluation of performance of a LoRa network was done by Galina *et al.* (2018). The authors proposed that the good quality signal can be maintained consistently in the open space. However, the quality is significantly bad in testing in the city and forest because of the existence of dense obstacles and noise. This issue can be tackled by lifting the gateway to a higher position that free LOS. To investigate RSSI and packet loss of LoRa, the experiment was conducted by Muaz Abdul Rahman,

Hafizhelmi Kamaru Zaman and Afzal Che Abdullah (2018). The measurement of SNR between the RX and transmitter (TX) was also carried out. As a result, LoRa ED has ability to fight against multipath and signal fading with a higher SF. In several realistic scenes, the performance evaluation of LoRaWAN has been discussed by Sanchez-Iborra *et al.* (2018). The proposed work shows that there is a data rate and link robustness trade-off. Therefore, the configuration parameters of LoRaWAN need to be adjusted depending on the distance between the base station and the node and conditions of propagation. When the distance between two end points is increased, lower data rates offer more link robustness with the decreasing of transmission rate dramatically. A nomadic test was also carried out to investigate the consequence of mobility on system performance. The results of this experiment confirmed that when using low transmission data rates, LoRaWAN has less obvious vulnerability but this effect is significant when using high data rates. The authors summarised that LoRaWAN presents a high adaptability level to be used in several applications of IoT. Thus, the transmission system is capable to set up low power consumption long links with high robustness and even under conditions of mobility by selecting the most suitable configuration.

LoRaWAN analysis for Health Care Systems was addressed by Buyukkakkarlar *et al.* (2017). In this study, transmission of medical data in health care systems in a realistic hospital environment was done. The authors summarised that LoRaWAN can be a technology in delivering sensor data for long range. Besides that, Wan *et al.* (2018) carried out LoRa propagation testing in soil and Xue-Fen *et al.* (2018) presented measurement of soil propagation for wireless underground sensor networks (WUSN) by using the LoRa-based smartphone. The results show that LoRa can be potential technology for WUSN as it has the strengths of low cost, high flexibility and reliability. LPWAN vehicle diagnostic system which is LoRa based for achieving safety of driving was presented (Chou *et al.*, 2017). LoRa-based environmental sensing system for monitoring of status of a large farm in real time was discussed by Ji *et al.* (2019). LoRa module is used as transceiver system for air pollution monitoring system (Rosmiati *et al.*, 2019). From these works, LoRa technology can be applied in different areas such as smart hospitals, smart cities, smart buildings,

location tracking, industrial, agriculture, transport, and environmental protection.

For reliable image transmission using LoRa, a transmission protocol called Multi-Packet LoRa Protocol (MPLR) was proposed by Chen et al. (2019). This protocol decreases the number of acknowledgements (ACKs) and waiting time by batching data packets transmission. The practicability of real-time wide-area agricultural environments visual monitoring systems was showed by Ji et al. (2019). The reduction of bandwidth usage and performance enhancement can be achieved by sending the modified image patches. Besides that, Jebiril et al. (2018) carried out the P2P image transmission using LoRa to monitor the outdoor environment in Malaysia successfully. In this monitoring system, novel image encryption technique was used. Pham (2016) proved that image transmission with LoRa technology is possible while remaining the low power consumption for image sensors. In addition, to allow the implementation of advanced image sensor devices, an modified Carrier Sense Multiple Access (CSMA) mechanism for packet collision avoidance and mechanism of activity time for overcoming limitation of duty-cycle were proposed by Pham (2018).

2.5 Summary

The first mentioned research has presented the differences of NB-IoT, LoRa and Sigfox. Sigfox and LoRa have long range coverage cost-effective devices with long lifetime of battery. On the other hand, LoRa will provide the dependable communication which supports mobility and assist the local network deployment. Conversely, NB-IoT will provide the high expense IoT markets that offers better quality of service and low latency. This chapter also discussed the related work of LoRa's performance analysis and applications.

CHAPTER 3

LORAWAN

3.1 Introduction

The methodology and results obtained from performance study of LoRaWAN will be introduced in this chapter. The selection of frequency of this project also will be explained.

3.2 Selection of frequency

There are different operating frequencies for LoRa in different countries. LoRa utilizes unlicensed frequency bands like 433 MHz, 868 MHz and 915 MHz. There is a list of frequency plan used in The Things Network (TTN). These frequency plans are documented from LoRaWAN regional parameters (TheThingsNetwork, n. d.). In Malaysia, LoRaWAN frequency plan of AS920-923 (“AS1”) will be used as shown in Figure 3.1 (TheThingsNetwork, n. d.).



Country	Frequency Band	Regulatory Reference
Madagascar	EU863-870 EU433	CRASA follows CEPT Rec. 70-03
Malawi	EU863-870 EU433	CRASA follows CEPT Rec. 70-03
Malaysia	AS920-923 ("AS1")	

Figure 3.1: LoRaWAN frequency plan of AS920-923 (TheThingsNetwork, n. d.).

3.3 Equipment used

The equipment and components used in this study will be introduced. The hardware required are LoRa gateway concentrator module RAK831, Raspberry Pi Zero W, converter board, RAK811 Lora Module and laptop. The software required are The Things Network, Raspbian Operating System (OS), PuTTY, RAK Serial Port Tool and Advanced IP Scanner.

The concentrator module RAK831 can be used in various types of application such as IoT, Machine to Machine (M2M) and Smart Metering. By

using different SFs on multiple channels, it can receive up to 8 LoRa packets simultaneously. The concentrator module RAK831 can be integrated into a gateway as a complete RF front end of this gateway. Robust communication between a LoRa gateway and a large amount of LoRa EDs distributed over a long range is possible with the use of concentrator module RAK831. A host system is required for the RAK831 to operate properly. A personal computer (PC) or microcontroller (MCU) acts as the host processor connected to RAK831 via Universal Serial Bus (USB) or SPI.

Raspberry Pi Zero W is used as host system for RAK831 for proper operation. It is a perfect single board computer for IoT. Raspberry Pi Zero W can be controlled wirelessly. Micro secure digital (SD) card is worked together with Raspberry Pi Zero W to store the data. All the components and specifications of Raspberry Pi Zero W are shown in Figure 3.2 and Table 3.1 respectively.

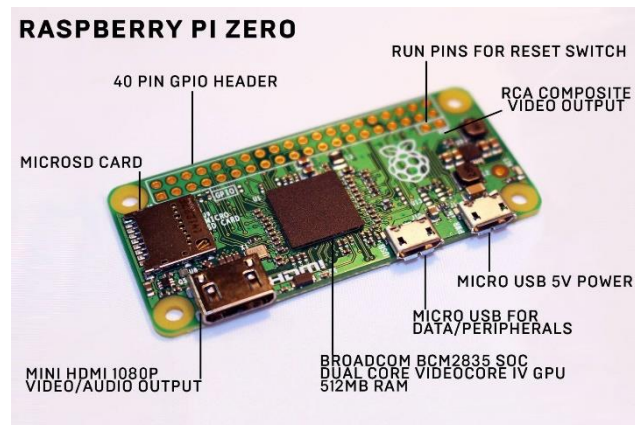


Figure 3.2: Raspberry Pi Zero W.

Table 3.1: Raspberry Pi Zero W specifications.

Operating Voltage	5 V
Operating Current	2.5 A
Processor	BCM 2835 SOC
Clock Speed	1 GHz
RAM	512 MB
Built-in Wireless	BCM43143, 802.11 b/g/n wireless LAN, Bluetooth 4.1, and Bluetooth Low Energy (BLE)
Memory	Micro-SD
Display and Audio	Mini High-Definition Multimedia Interface (HDMI)
USB port	Micro-B USB for OTG
Power input	Micro-B USB for power
GPIO	Unpopulated 40-pin GPIO connector

The RAK831 concentrator board must communicate with the host processor (Raspberry Pi Zero W board) via SPI. The connections can be made with floating wires as both boards have header connectors. However, to ease the connection, the adaptor board provided by RAK Wireless that snaps in to both boards can be used. All the components and specifications of RAK831 concentrator board is shown in Figure 3.3 and Table 3.2 respectively.



Figure 3.3: RAK831 concentrator board.

Table 3.2: RAK831 concentrator board specifications.

Supply voltage	5 V
Compact size	80.0 × 50.0 × 5.0 mm
Frequency band	433, 470, 868 and 915 MHz
Sensitivity down	Down to -142.5 dBm
Maximum link budget	162 dB
Interface	SPI
Processor	SX1301 base band
Output power level	Up to 23 dBm
Range	Up to 15 km (LOS), several km in urban environment

RAK811 LoRa Module is low-power small size solution for long range wireless communication. It supports LoRa P2P communications which can allow users to create their own long range LoRa network privately. Integration of Semtech's SX1276 and STM32L provide user a serial AT commands with Universal Asynchronous Receiver/Transmitter (UART) Interface. All the components and specifications of RAK811 LoRa Module is shown in Figure 3.4 and Table 3.3 respectively.

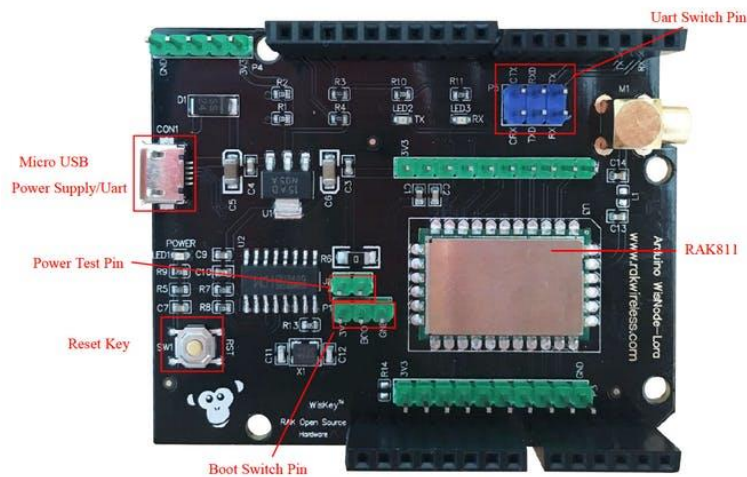


Figure 3.4: RAK811 LoRa Module.

Table 3.3: RAK811 LoRa Module Specifications.

Protocol supported	LoRaWAN
Frequency Band supported	Global license-free ISM band
Activation method	Over-the-Air Activation (OTAA) or Authentication By Personalisation (ABP)
Interface	UART
Maximum output power	100 MW (20 dBm)
High sensitivity	-148 dBm
Range	Greater than 15 km
Capacity	Up to 1 million devices
Power consumption	500 nA on standby
Modulation scheme	LoRa, frequency shift keying (FSK), Gaussian frequency shift keying (GFSK), On-off keying (OOK)
Communication way	Birectional 2-way communications
Battery life	Over 10 years

The Things Network is founded by Johan Stokking and Wienke Giezeman in 2015. It is a worldwide, open, free of charge and decentralized internet of things network. The things can be linked to the internet using low

power with the network. It usually operates with the use of the LoRaWAN technology as LoRa is a low energy and wide range wireless technology that uses an open data frequency.

Raspbian OS is the Debian-based operating system of Raspberry Pi which consists of various versions including Raspbian Stretch and Raspbian Buster.

PuTTY is a terminal emulator for platforms of Windows and Unix. Several network protocols can be supported by PuTTY, consisting Secure Shell (SSH) Protocol, rlogin and raw socket connection. Raspberry Pi Zero W can be controlled wirelessly by other devices with the set up of Wi-Fi and SSH connection.

RAK Serial Port Tool is a serial communication tool developed by RAKwireless to make the setup more compatible and convenience. However, this tool only supports Windows OS. For other OS, other serial communication tool serial can be used on user's own preference.

Advanced IP Scanner is a powerful scanner to obtain the Internet Protocol (IP) address of the devices on the network.

3.4 LoRa Gateway setup

1. The gateway is set up by using the adaptor board provided by RAK Wireless that snaps in to LoRa gateway concentrator module RAK831 and Raspberry Pi Zero W as shown in Figure 3.5.



Figure 3.5: LoRa gateway setup.

2. The latest firmware is burned into SD card as shown in APPENDIX D. After burning, the SD card is installed into Raspberry Pi Zero W.

- 3. The IP address of LoRa gateway is checked by using Advanced IP Scanner as shown in Figure 3.6.

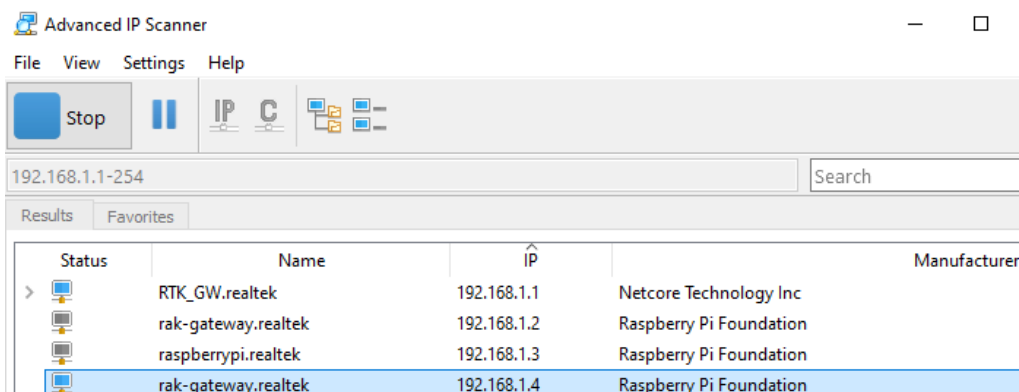


Figure 3.6: Checking of LoRa gateway’s IP address.

- 4. The LoRa gateway is accessed through SSH using PuTTY as shown in Figure 3.7. RPI’s username and password are “pi” and “raspberry” respectively.

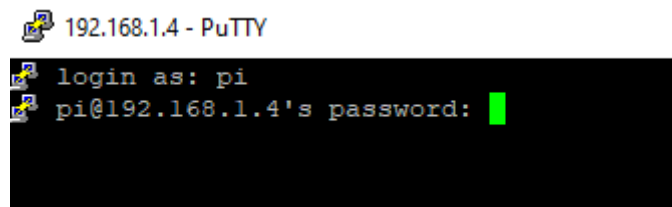


Figure 3.7: LoRa gateway is accessed through SSH using PuTTY.

- 5. A command “sudo gateway-config” is entered. The gateway ID is shown in Figure 3.8.

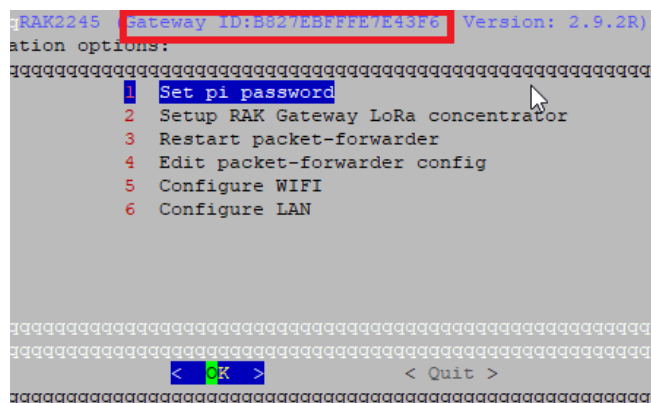


Figure 3.8: Gateway ID.

6. “Setup RAK Gateway LoRa concentrator” is chosen to set the LoRa server and frequency to TTN and AS923 respectively.
7. The gateway is registered in TTN as shown in APPENDIX E. Once the gateway is registered, the gateway overview is displayed as shown in Figure 3.9.

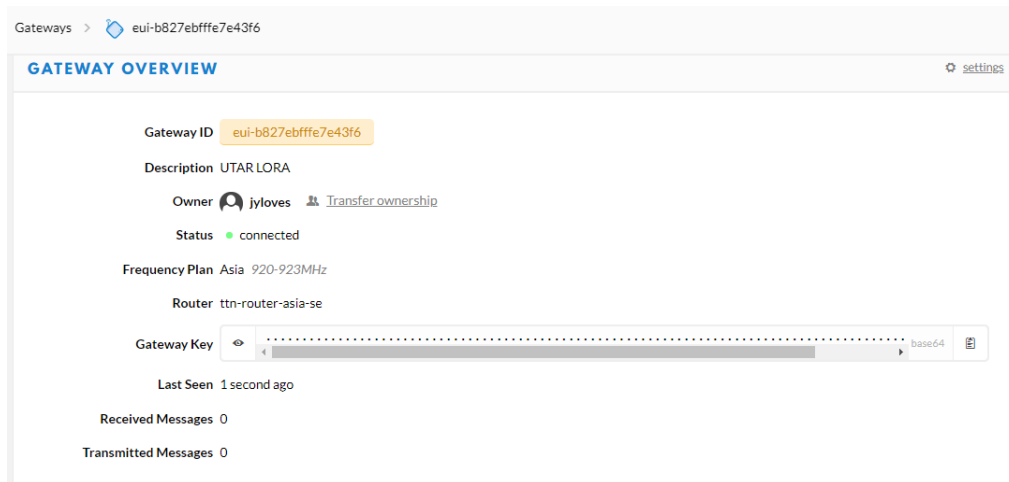


Figure 3.9: TTN console page after registration of gateway.

3.5 LoRa Node setup

1. Pre-configuration is needed to operate with RAK811. First, for boot mode, the “BOOT” pin and “3V3” pin are connected as shown in Figure 3.10.

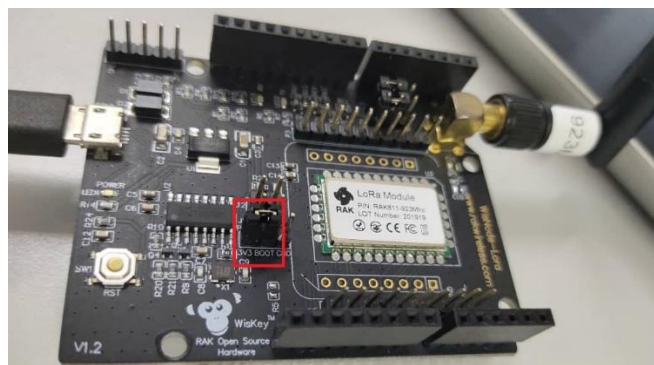


Figure 3.10: Boot mode for RAK811 Wisnode.

2. RAK811 WisNode is connected with laptop’s USB port as shown in Figure 3.11.



Figure 3.11: Laptop's USB port and RAK811 Wisnode connection.

3. The RST button on RAK811 WisNode as shown in Figure 3.12 is pressed.

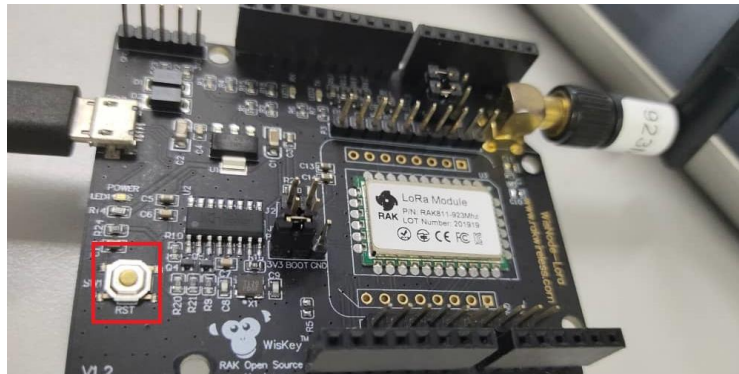


Figure 3.12: RST button on RAK811 WisNode.

4. The STM32CubeProgrammer tool is used to burn a bootloader into RAK811 WisNode. The UART type is selected, then the port, baudrate, and parity as shown in figure below:

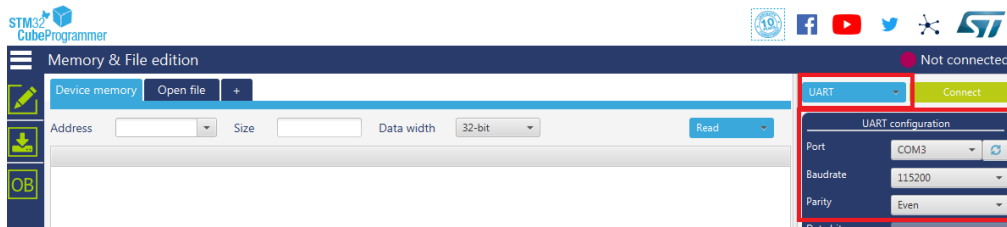


Figure 3.13: UART, port, baudrate, and parity configuration in STM32CubeProgrammer.

5. “Connect” button at the top right corner is pressed. Figure 3.14 shows the correct log.

Address	0	4	8	C	ASCII
0x08000000	20000808	08000191	08000199	08000198
0x08000010	0800019D	0800019F	080001A1	00000000
0x08000020	00000000	00000000	00000000	080001A3
0x08000030	080001A5	00000000	080001A7	08001669	Y.....\$...i...
0x08000040	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....
0x08000050	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....
0x08000060	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....
0x08000070	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....
0x08000080	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....
0x08000090	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....
0x080000A0	00000000	080001AB	080001AB	080001AB<.....<.....
0x080000B0	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....
0x080000C0	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....
0x080000D0	080001AB	080001AB	080001AB	080001AB	<.....<.....<.....

Figure 3.14: Correct log in STM32CubeProgrammer.

6. All data on RAK811 WisNode is erased and “Open file” button is pressed as presented in Figure 3.15.

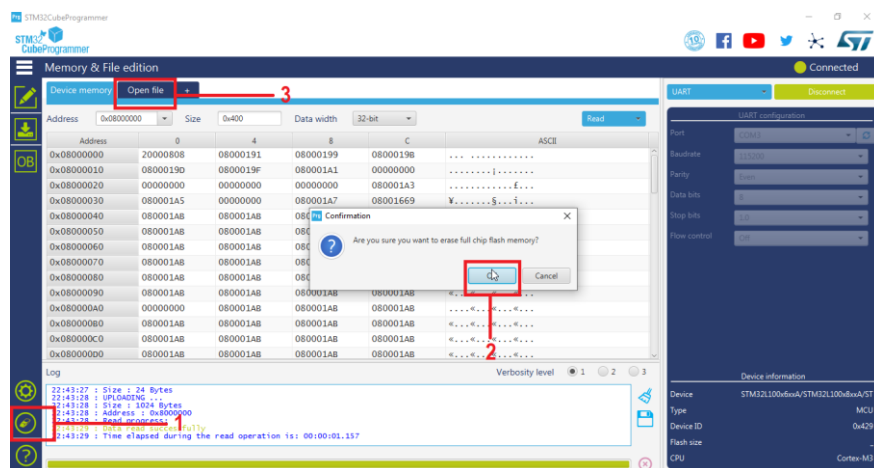


Figure 3.15: Data on RAK811 WisNode is erased and new file is opened.

- The latest bootloader file (RAK811_BOOT_V3.0.2.bin) is loaded and downloaded to RAK811 as shown in Figure 3.16.

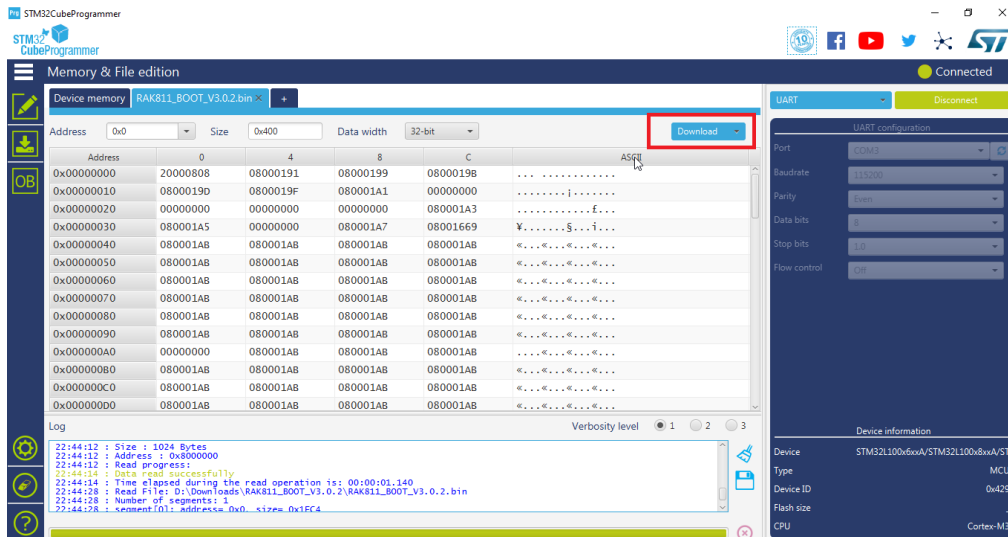


Figure 3.16: Bootloader file (RAK811_BOOT_V3.0.2.bin) is loaded and downloaded to RAK811.

- A message is pop out from the window to inform the user bootloader is burned into RAK811 WisNode successfully.

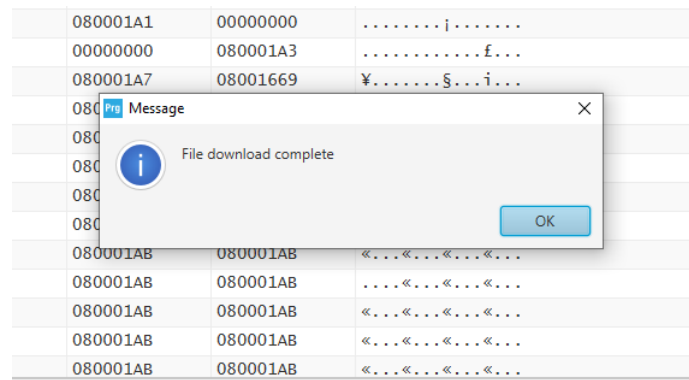


Figure 3.17: Message of file download complete.

- “Disconnect” button is pressed, the STM32CubeProgrammer tool is closed, RAK811 is powered down and the “BOOT” pin and the “GND” pin are connected as shown in Figure 3.18.



Figure 3.18: “BOOT” pin and the “GND” pin connection.

10. Then, RAK811 WisNode is connected with the laptop’s USB interface again. The RST button is pressed and BOOT MODE of RAK811 is displayed by using RAK serial port tool as shown in Figure 3.19. This means that the bootloader is burned into RAK811 WisNode successfully.

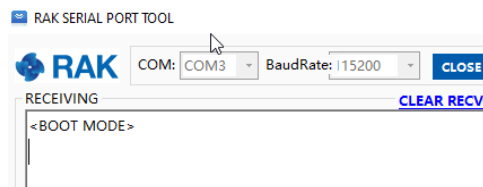


Figure 3.19: Showing of BOOT MODE of RAK811 in RAK serial port tool.

11. The latest firmware (RAK811_HF_V3.0.0.13.T3) is burned into RAK811 by using RAK LoRaButton Upgrade Tool V1.0 as shown in Figure below.

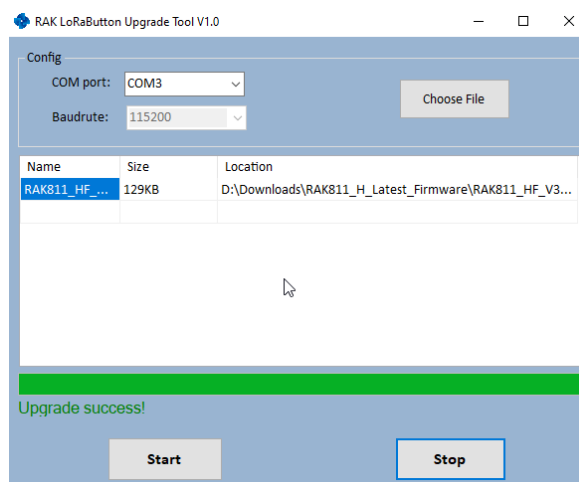


Figure 3.20: Burning of firmware into RAK811 using RAK LoRaButton Upgrade Tool V1.0.

- The correct COM port is chosen while baud rate is 115200. The log as shown in Figure 3.21 is displayed when the RST button of RAK811 Wisnode is pressed.

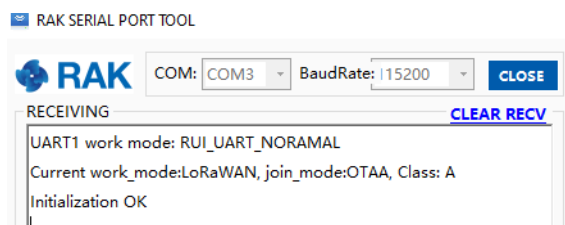


Figure 3.21: Log data when the RST button of RAK811 Wisnode is pressed.

- The application is created in the TTN.

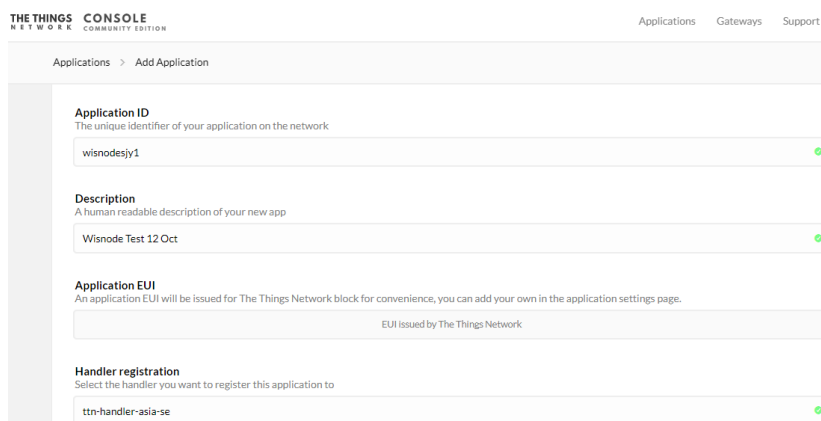


Figure 3.22: Registration of application in The Thing Networks.

14. The device is registered under the application.

Applications > wisnodesjy1 > Devices

Device ID
This is the unique identifier for the device in this app. The device ID will be immutable.
realtest

Device EUI
The device EUI is the unique identifier for this device on the network. You can change the EUI later.
this field will be generated

App Key
The App Key will be used to secure the communication between you device and the network.
this field will be generated

App EUI
70 B3 D5 7E D0 02 3A E9

Cancel Register

Figure 3.23: Registration of device in The Thing Networks.

15. The default activation method is OTAA. Device EUI, Application EUI and App Key are used on RAK811 WisNode as shown in Figure 3.24.

Applications > wisnodesjy1 > Devices > realtest

Application ID wisnodesjy1

Device ID realtest

Activation Method OTAA

Device EUI <> = 00 45 0F 42 11 B2 B6 00

Application EUI <> = 70 B3 D5 7E D0 02 3A E9

App Key <> =

Device Address <> = 26 04 24 CC

Network Session Key <> =

App Session Key <> =

Status 2 days ago

Frames up 22 [reset frame counters](#)

Frames down 0

Figure 3.24: Configuration parameters in The Thing Networks.

16. The configuration of RAK811 is done by using the AT command in Table 3.4.

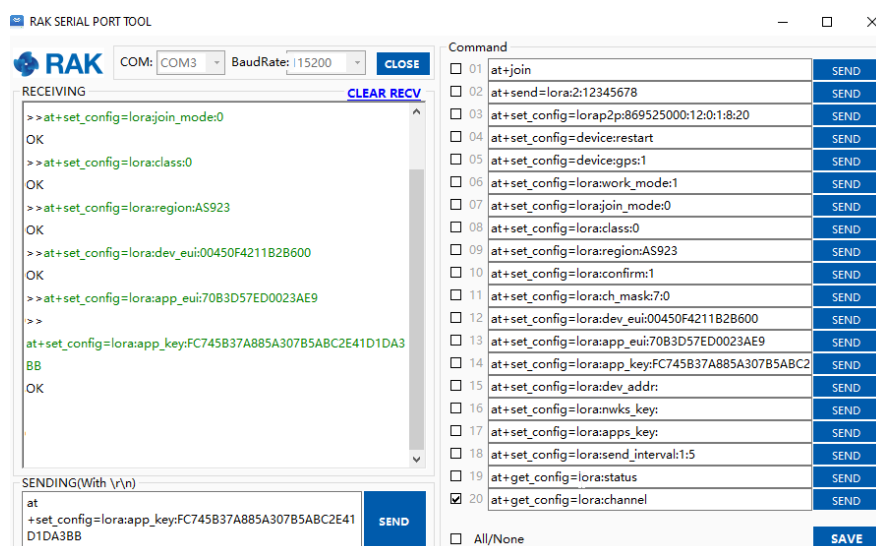


Figure 3.25: Configuration of RAK811 Wisnode by using RAK Serial Port Tool.

Table 3.4: AT command used for configuration.

AT command	Description
at+set_config=lorajoin_mode:0	Set the OTAA join mode for LoRaWAN.
at+set_config=loraclass:0	Set the class A for LoRa.
at+set_config=loraregion: AS923	Set the AS923 region for LoRa.
at+set_config=loradev_eui:00450F4211B2B600	Set the device EUI for OTAA.
at+set_config=lorapp_eui:70B3D57ED0023AE9	Set the application EUI for OTAA
at+set_config=lorapp_key:FC745B37A885A307B5ABC2E41D1DA3BB	Set the application key for OTAA.

17. OTAA mode is joined by using AT command “at+join” as shown in Figure 3.26.

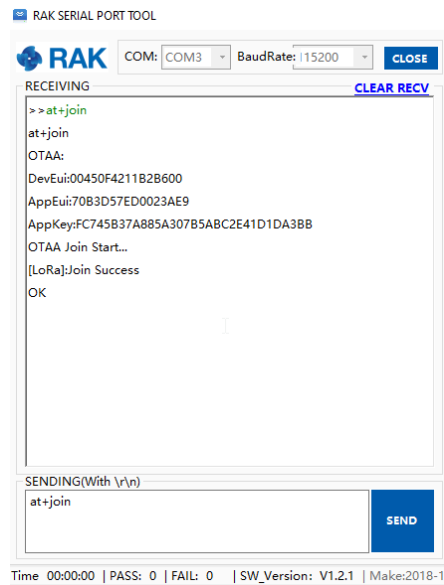


Figure 3.26: Joining LoRa network by using AT command “at+join”.

18. The payload is sent by by using AT command “at+send=lora:2:12345678” as shown in Figure 3.27.

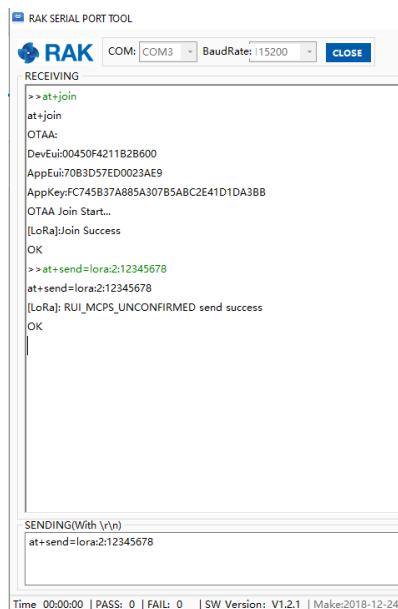


Figure 3.27: Sending data with AT command “at+send=lora:2:12345678”.

19. The traffic of gateway and node are monitored from TTN as presented in Figure 3.28 and Figure 3.29 respectively.

time	frequency	mod.	CR	data rate	airtime (ms)	cnt
16:42:16	923.6		4/5	SF 10 BW 125	329.7	
16:42:12	923.6		4/5	SF 10 BW 125	370.7	app_eui: 70 B3D5 7E D002 3A E9 dev_eui: 00 45 0F 42 11 B2 B6 00

Figure 3.28: Gateway traffic shown in The Thing Networks.

```

{
  "time": "2020-03-25T09:38:08.388475482Z",
  "frequency": 924.2,
  "modulation": "LoRa",
  "data_rate": "SF10BW125",
  "coding_rate": "4/5",
  "gateway": {
    "ntw_id": "eui-8827a0ffffe43f6",
    "timestamp": 1888427188,
    "time": "",
    "channel": 5,
    "rx1": -5,
    "snr": 18.8
  }
}

```

Figure 3.29: LoRa application traffic in The Thing Networks.

20. The SF of RAK811 LoRa module can be configured to SF7, SF8, SF9 and SF10 with different AT commands according to Table 3.5.

Table 3.5: Number of DR of RAK811 LoRa module.

Number of DR	AT command	Configuration	Indicative physical bit rate [bit/s]
2	at+set_config=lora:dr:2	LoRa: SF10/125kHz	980
3	at+set_config=lora:dr:3	LoRa: SF9/125kHz	1760
4	at+set_config=lora:dr:4	LoRa: SF8/125kHz	3125
5	at+set_config=lora:dr:5	LoRa: SF7/125kHz	5470

3.6 LoRa Packet Structure

LoRa packet is made up of 2 preambles, a header, the payload and a Cyclic Redundancy Check (CRC) as shown in Figure 3.30 and their corresponding explanations are shown in Figure 3.31. The spreading factors can be varied from

SF7 to SF12. Besides that, the coding rate of payload can be chosen from 4/8, 4/7, 4/6, and 4/5. Higher sensitivity can be obtained from higher coding rate.

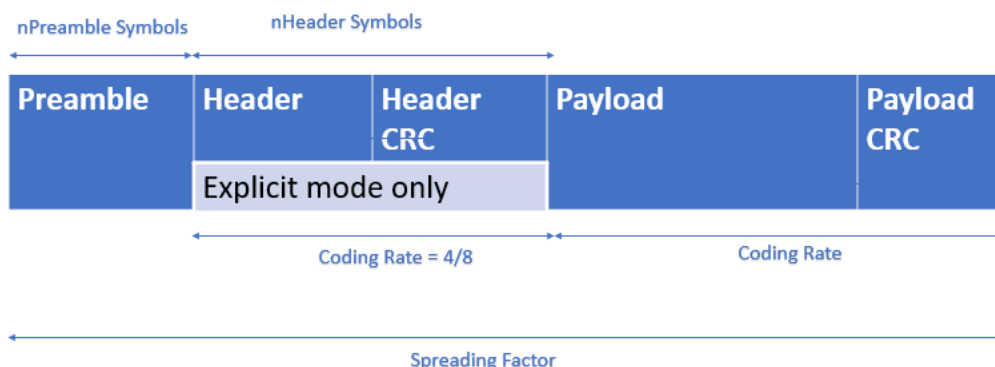


Figure 3.30: LoRa Packet Structure.

Content	Explanation
Preamble	It is used to synchronize RX with the flow of incoming data.
Header (Implicit or Explicit)	Mostly in explicit mode. Explicit mode: Header will give the information of the payload (payload length in bytes, code rate of forward error correction (FEC) and existence of CRC for the payload). Implicit mode: the header doesn't exist in the packet, all information of payload must be configured manually.
Payload	The payload is variable. For example, the sensor data can be sent as the payload.
Payload CRC	An optional CRC may be appended.

Figure 3.31: LoRa Packet Structure explanations.

3.7 LoRaWAN network architecture

The standard network architecture of LoRaWAN is shown in Figure 3.32. The EDs communicate with gateway using LoRa through a single hop with

LoRaWAN. The packets from EDs are forwarded by gateway to network server through Wi-Fi, Ethernet or satellite.

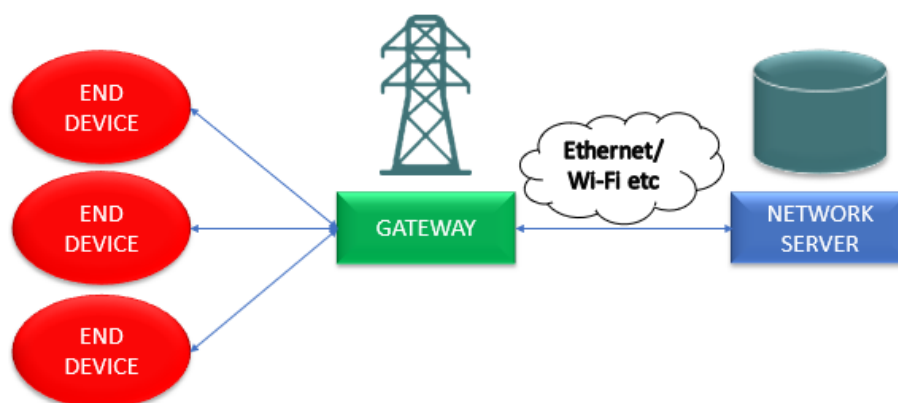


Figure 3.32: Standard network architecture of LoRaWAN.

In this study, the network architecture of LoRaWAN is shown in Figure 3.33.

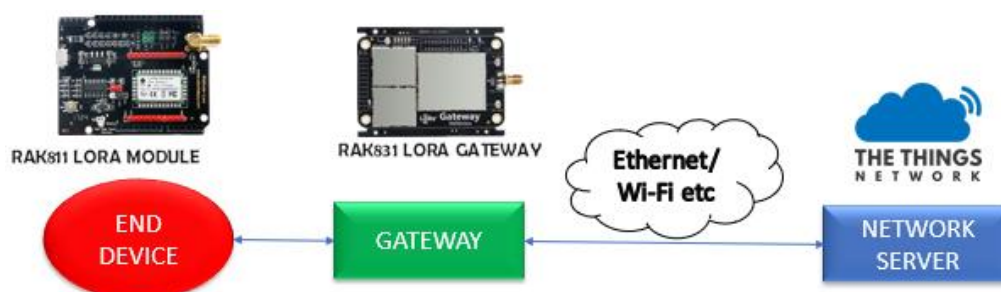


Figure 3.33: Network architecture of LoRaWAN in this study.

3.8 Performance evaluation of LoRaWAN

The LoRaWAN performance evaluation in an indoor environment is studied. The experiment is composed of one ED, one gateway and one server. The gateway is composed of RAK831, the ED is composed of RAK811 connected to laptop via the UART interface which sends data by using AT command and the server used is TTN server. The performance evaluation consists of:

1. The impact on airtime under different SFs and payload lengths.
2. The impacts on RSSI and SNR under different transmission distances.

3.8.1 The impact on airtime under different SFs and payload lengths

3.8.1.1 Experimental Setup

1. LoRa gateway concentrator module RAK831 is powered on.
2. RAK811 Wisnode is connected through an USB port and programmed using RAK Serial Port Tool.
3. The fixed parameters are configured as stated in Table 3.6.

Table 3.6: LoRa configuration paramaters.

Bandwidth	125 kHz
Coding Rate	4/5
Frequency band used	AS923
Distance between gateway and node	0 m

4. Steps 1 to 3 are repeated by varying SF7 to SF10.
5. Steps 1 to 4 are repeated by sending the payload with length of 14 bytes, 17 bytes, 29 bytes and 32 bytes.
6. The results are recorded in Table 3.7.

3.8.1.2 Result and discussion

In this study, different SFs are used for different payload lengths to investigate the impact on airtime. The airtime for different SFs and payload lengths are obtained from TTN and presented in Table 3.7 below. The corresponding chart is plotted as shown in Figure 3.34.

Table 3.7: Airtime (in milliseconds) for different SFs and payload lengths.

Payload length (bytes)	14	17	29	32
SF				
7	46.3	51.5	66.8	71.9
8	82.4	92.7	123.4	133.6
9	164.9	164.9	226.3	246.8
10	288.8	329.7	411.6	452.6

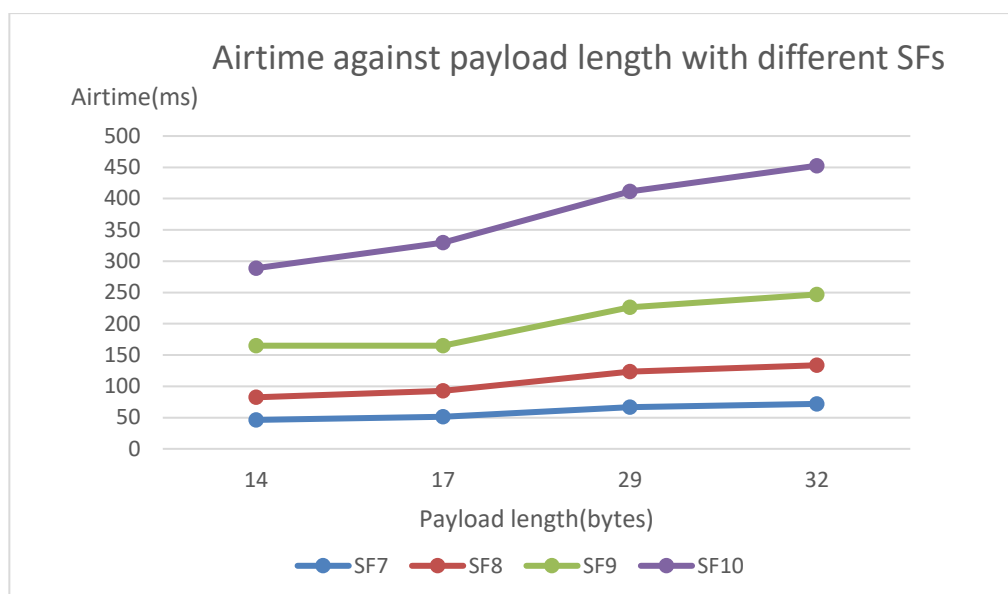


Figure 3.34: Airtime against payload length with different SFs.

The most robust transceiver setting in this study is the transceiver with spreading factor of SF10, the airtime of 288.8 ms is obtained when sending 14 bytes payload. In contrast, the transceiver with spreading factor of SF7 consumed the shortest airtime of 46.3 ms in the same scenario. The higher the spreading factor, the better the reconstitution of the signal. However, this leads to significant increment in airtime.

The results also show that the transmission time increases as the payload length increases. This is because the longer the length the payload sent, the larger the payload needed to be encoded by the ED and decoded by the gateway.

3.8.2 The impacts on RSSI and SNR under different transmission distances

3.8.2.1 Location

The impacts on RSSI and SNR under different transmission distances are studied in Evergreen Scotpine Condominium. Figure 3.35 and Figure 3.36 show the building and the floor map of Evergreen Scotpine Condominium respectively. It is 13 story building and made from thick concrete structure. The floor map is the same for each floor. The height is approximately 5m for each floor.

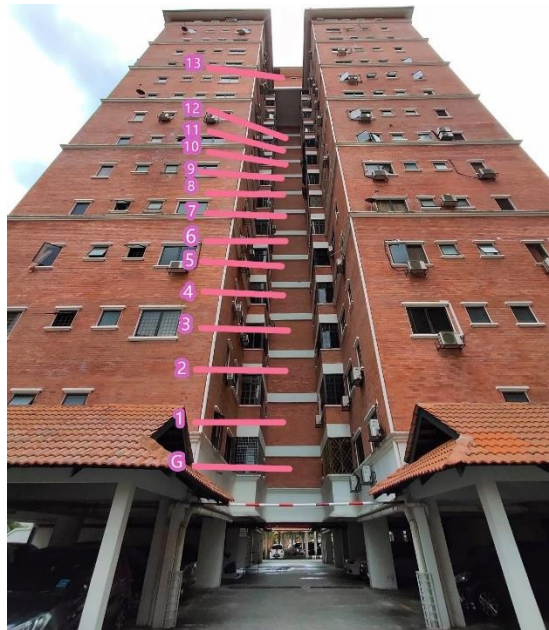


Figure 3.35: Building of Evergreen Scotpine Condominium.

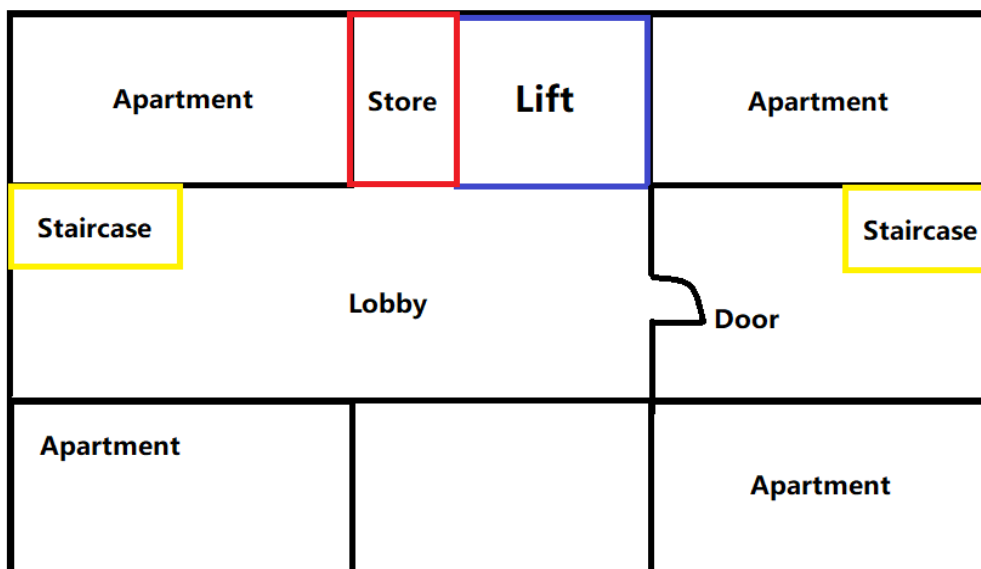


Figure 3.36: Floor map of Evergreen Scotpine Condominium.

3.8.2.2 Experimental Setup

1. LoRa gateway concentrator module RAK831 is powered on.
2. RAK811 Wisnode is connected through an USB port and programmed using RAK Serial Port Tool.
3. The fixed parameters are configured as stated in Table 3.8

Table 3.8: Configuration parameters.

Payload size (Bytes)	17
Coding Rate	4/5
SF	7
Bandwidth	125kHz
Frequency band used	AS923

4. The gateway RAK831 is located at G floor of condominium as shown in Figure 3.38 and pointed out by the blue point in Figure 3.37.
5. The RAK811 LoRa Module is placed at 1st floor as shown in Figure 3.40 and the location of RAK811 LoRa Module is represented by red point in Figure 3.39.
6. 17 bytes payload is sent from node.
7. The SNR and RSSI values of the gateway and the node are recorded.
8. The experiment is repeated by placing the RAK811 LoRa Module at G, 2nd, 3rd, 6th, 9th and 12nd floors.
9. For each variation of distance, the experiments are repeated 10 times. The SNR and RSSI values are averaged as shown in Table 3.9.

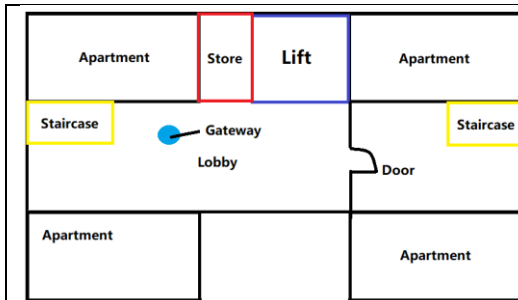


Figure 3.37: Floor map (Gateway).



Figure 3.38: Gateway at G floor.

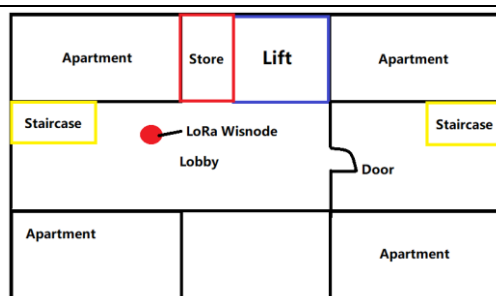
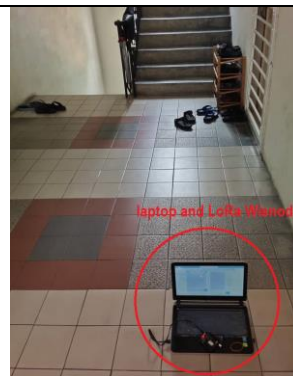


Figure 3.39: Floor map (LoRa Wisnode).

Figure 3.40: Laptop and RAK811 LoRa Module at 3rd floor.

3.8.2.3 Result and discussion

In this study, different transmission distances are used to investigate the impacts on SNR and RSSI. All the SNR and RSSI values are based on the average values of 10 runs as shown in Table 3.9.

Table 3.9: Results of the impacts on RSSI and SNR under different transmission distances.

Location of end-device	Distance (m)	Average SNR (dB)	Average RSSI (dBm)
G	0	9.5	-6.6
1st	5	8.68	-74.1
2nd	10	8.24	-88.73
3rd	15	5.87	-94.71
6th	30	2.67	-101.6

9th	60	-5.31	-106.22
12nd	90	-5.42	-106.73

From the Table 3.9, as the distance increases, the average RSSI decreases. The average RSSI when the node RAK811 is placed at G floor is -6.6 dBm but the drastic drop of average RSSI to -74.1 dBm occurred when the transmitter is placed at 1st floor. This rapidly dropping of signal strength is believed caused by the multiple signal path that presents within the building where the gateway is placed. The average RSSI gradually decrease to -88.73 dBm, -94.71 dBm, -101.6 dBm, -106.22 dBm, -106.73 dBm for the distance of 10 m, 15 m, 30 m, 60 m and 90 m respectively. The average RSSI at G floor is largest since the signals transmit without the blocking of ceilings and walls. The RSSIs become smaller when the node set farther from the gateway because the more signals are interrupted by the ceilings and walls.

SNR of the signal means the ratio of useful signal power to the noise power. There is possibility to get either positive or negative SNR. Negative SNR represents that power of signal is smaller than the power of noise. Generally, positive SNR has to be obtained during transmission. LoRa SNR values are between -20 dB and 10 dB typically. From Table 3.9, the average SNR when the node RAK811 is placed at G floor is 9.5 dB which means the received signal is less corrupted. Table 3.9 also shows that the distance increases, the average SNR decreases. It is believed that power signal suffers more attenuation and more noise power is added during transmission as the distance increases. Although there are two average SNR values are obtained below 0 dBm, LoRa modulation allows EDs to demodulate signals with the SNR value between -7.5 dB and -20 dB under the power level of noise floor due to its robustness.

CHAPTER 4

LORA P2P

4.1 Introduction

The phrase “a picture is worth a thousand words” refers to multiple information can be expressed by a single still image. In agriculture, images can provide plant breeding and yielding information. Besides that, images can help to monitor the forest, the real time situations about environment issues such as forest fire can be observed. They can also capture real time physical condition of the remote IoT system. This benefits system operation. For example, a heavy rainfall may disturb the data collection; an automated system or human being could alternate the soil moisture reading.

The LoRaWAN is not suitable to be used for image transmission. This is because there are limitations of the data each ED can send according to Fair Access Policy of TTN (TheThingsNetwork, 2020). In order to achieve long-range image transmission, LoRa P2P communication technique is used.

4.2 Equipment used

The equipment and components used in this project for LoRa P2P will be introduced. The hardware required are Dragino LoRa module, Raspberry Pi Zero W, 5MP Camera Board for Raspberry Pi and laptop. The software required are Raspbian OS, PuTTY, Advanced IP Scanner and VNC viewer.

Dragino LoRa module is designed for RPI. This transceiver module has high noise immunity and allows long-range transmission with minimum current consumption. All the components and specifications of Dragino LoRa module is shown in Figure 4.1 and Table 4.1 respectively.



Figure 4.1: Dragino LoRa module.

Table 4.1: Dragino LoRa module specifications.

Maximum link budget	168 dB
Power amplifier	+14 dBm
Programmable bit rate	300 kbps
Sensitivity	-148 dBm
RX current	10.3 mA
Maximum payload length	256 bytes
Dynamix Range RSSI	127 dB
Resolution of fully integrated synthesizer	61 Hz
Maximum output power	100 mW (20 dBm)

5MP Camera Board is specially designed for Raspberry Pi and is compatible with most of the RPI in the market. It attaches to RPI with the use of Flat Flex Connectors (FFC) cable and communicates with RPI through Camera Serial Interface (CSI).

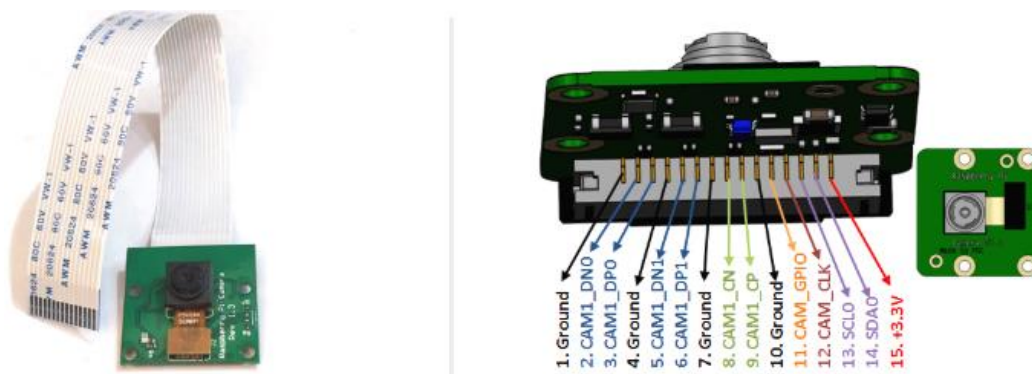


Figure 4.2: 5MP Camera Board for Raspberry Pi.

Table 4.2: 5MP Camera Board for Raspberry Pi specifications.

Compatibility	Raspberry Pi 1,2, 3 and zero series
Still picture resolution	2592 x 1944
Video supports	1080p at 30fps, 720p at 60fps and 640x480p 60/90 recording
Size	20 mm x 25 mm x 9mm
Interface	15 pin MIPI CSI
Weight	3 g
Module	5MP (megapixel) omnivision 5647 camera module

The Raspbian Operating System (OS), Advanced IP Scanner and PuTTY are the same in previous chapter. VNC viewer is a tool for controlling another computer (Raspberry Pi Zero W) by graphical desktop-sharing system.

4.3 LoRa P2P Packet Structure

The LoRa P2P packet structure is the same as in Chapter 3.6. The explanations on configuration for TX and RX are shown in APPENDIX G.

4.4 Design of Transmission Protocol

The sending and receiving of data need to be controlled. If the sending rate is higher than the receiving rate, then the data will get lost. The flow control can keep track the sending and receiving process in order to make sure the data can be transmitted successfully.

Prior to this study, some considerations must be figured out. Constrained by LoRA maximum transmission unit (MTU) of 255 bytes, the fragmentation needs to be done to segment the long image payload into multiple packets. In this study, the packet size of 128 bytes is used. It consists of 4 bytes identity number (ID) to prevent the repetition of packet and 124 bytes useful information as shown in Figure 4.3. The overhead of 4 bytes ID is only 3.125%.

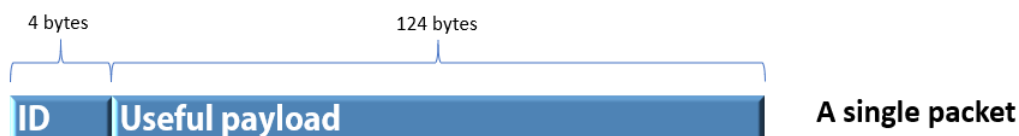


Figure 4.3: A single packet.

By using Raspberry Pi to implement Dragino LoRa module, C programming language is used as main language along with python. In order to send the image from TX to RX, the conversion of image into text is necessary. By using python, the Joint Photographic Experts Group (JPEG) image will be converted into hexadecimal format and stored in a text file (ImageTx.txt). After that, the bytes contained in the text file will be calculated using strlen function in <string.h> library and the number of packets to be sent will be evaluated using simple division and ceil function in <math.h>. For example, for a message with payload length of 130, the number of packets to be sent is 2 as shown in Figure 4.4 below.

C language	Answer
<code>numBytes = strlen(packet);</code>	<code>numBytes = 130.0</code>
<code>packet_no = ceil(numBytes/PAYLOADSIZE);</code>	<code>packet_no = ceil(130.0/124)=2</code>

Figure 4.4: C language and corresponding description.

Next, the headers will be added and stored in the text file (ImageTx_h.txt) to differentiate and give the identity to the packet segmented. The first packet with header 0000 contains the information about how many packets the RX will receive to make sure all the packets can be received correctly. This situation can be illustrated with the Figure 4.5 below.

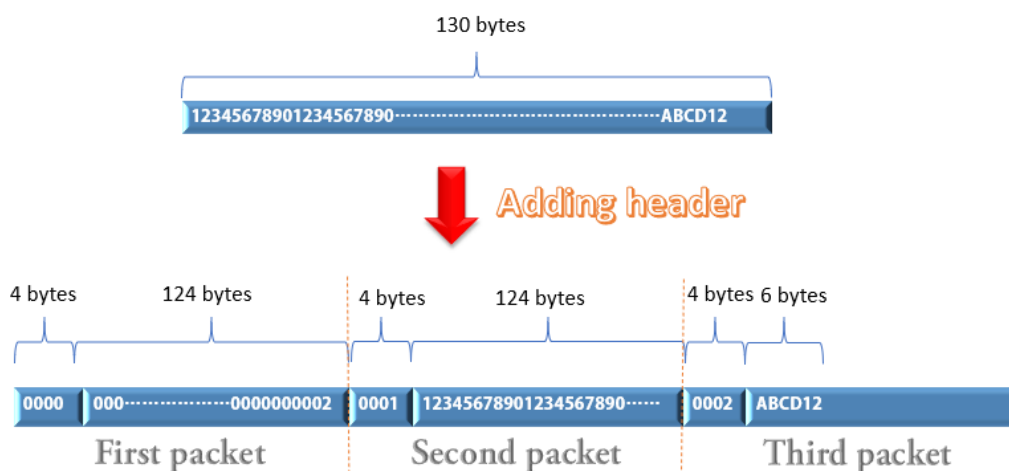


Figure 4.5: Adding header.

4.4.1 Stop and Wait Protocol

To ensure of arrival of multiple packets, reliable transport protocols are needed. In this study, two transport protocols are used. First protocol used is stop and wait protocol as shown in Figure 4.6. The sender will wait for ACK after sending out one packet and only proceed to send next packet after receiving ACK from RX. The sending and receiving process will be repeated until “DONE” ACK is received by transmitter.

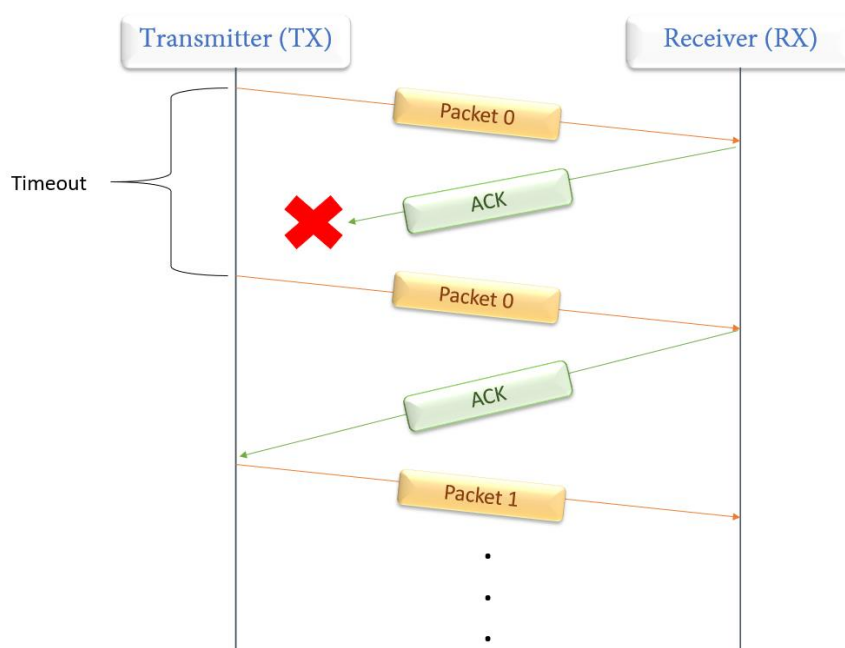


Figure 4.6: Stop-and-wait protocol.

4.4.1.1 TX

TX will start to send the packets after the hexadecimal representation of image is stored in text file (ImageTx_h.txt). After sending first packet, TX will wait to receive ACK from RX. If the ACK is 1, this indicates that the packet is received successfully by RX, then the TX will send out next packet. However, if the ACK is 0, TX will resend the packet. The sending and receiving processes will continue until the “DONE” ACK is received by TX, this ACK represents that all the packets are received by RX and the image is built successfully. The new image transmission process will start after this. Figure 4.7 shows the flowchart for writing C program of TX when using stop and wait protocol.

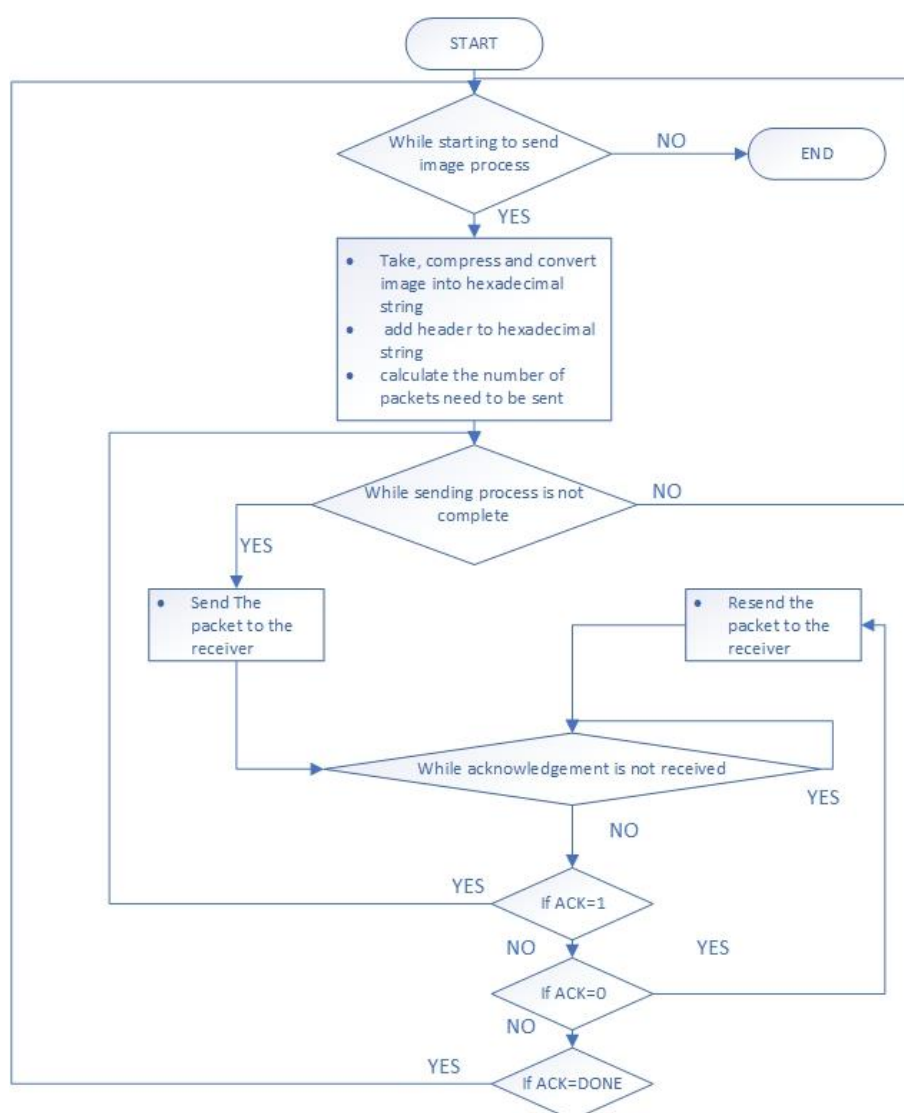


Figure 4.7: Stop and wait protocol's transmitter coding flowchart.

4.4.1.2 RX

RX will start to receive the packet sent from TX after receiving process is initiated. After receiving the packet, RX will send ACK to TX to inform the arrival of packet. The receiving and sending processes are repeated until all the packets are received successfully by receiver. After receiving all the packets, RX will start to build the image and upload the image into Dropbox. RX will send a “DONE” ACK to TX after constructing the image. New receiving process will start after this. Figure 4.8 shows the flowchart for writing C program of RX when using stop and wait protocol.

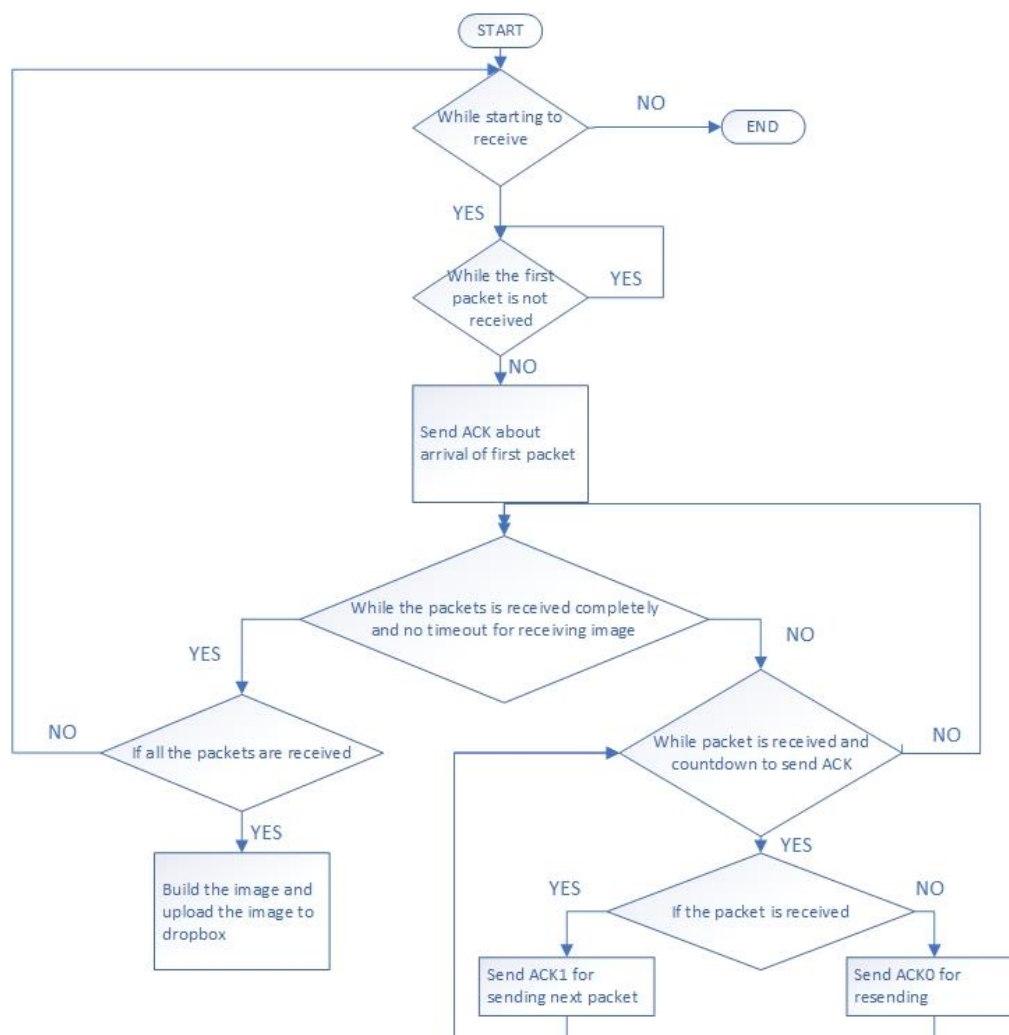


Figure 4.8: Stop and wait protocol’s receiver coding flowchart.

4.4.2 LoRa Multi-Packet Transmission Protocol

Another reliable transport protocol is designed in order to lessen the transmission time and obtain higher transmission rate. This self-defined

protocol given a name called LoRa Multi-Packet Transmission Protocol as shown in Figure 4.9. For the ideal condition, the TX will send all the data packets to the RX consecutively with interval of 5 seconds. The sending and receiving process will repeat until “DONE” ACK is received by TX.

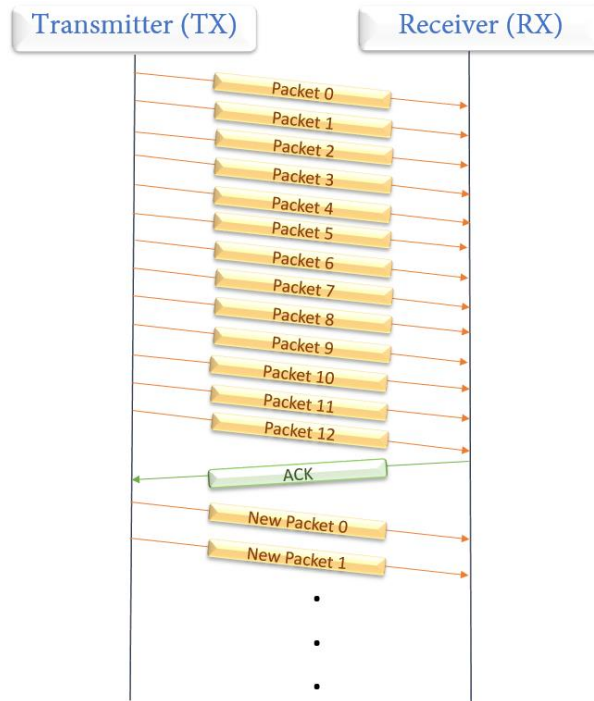


Figure 4.9: LoRa Multi-Packet Transmission Protocol.

However, the transmission loss may occur during the transmission. The “DONE” ACK sent from RX to TX may be lost. In this situation, by providing timeout which specified period of time is given to TX to wait for ACK. After timeout, new packet will be transmitted as presented in Figure 4.10.

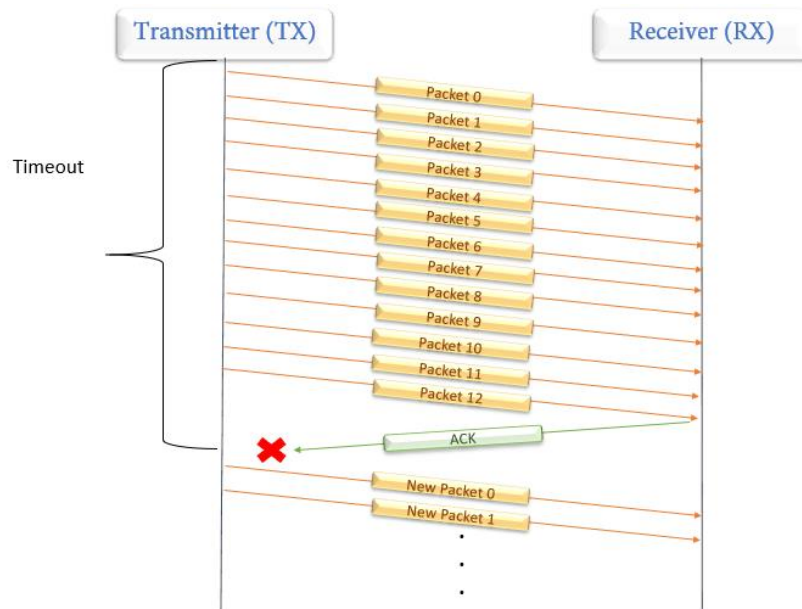


Figure 4.10: LoRa Multi-Packet Transmission Protocol with timeout session.

For the transmission loss in the sending process, RX will send ACK to TX to request for resending. After receiving ACK, sender will resend the requested packet to the RX again as shown in Figure 4.11.

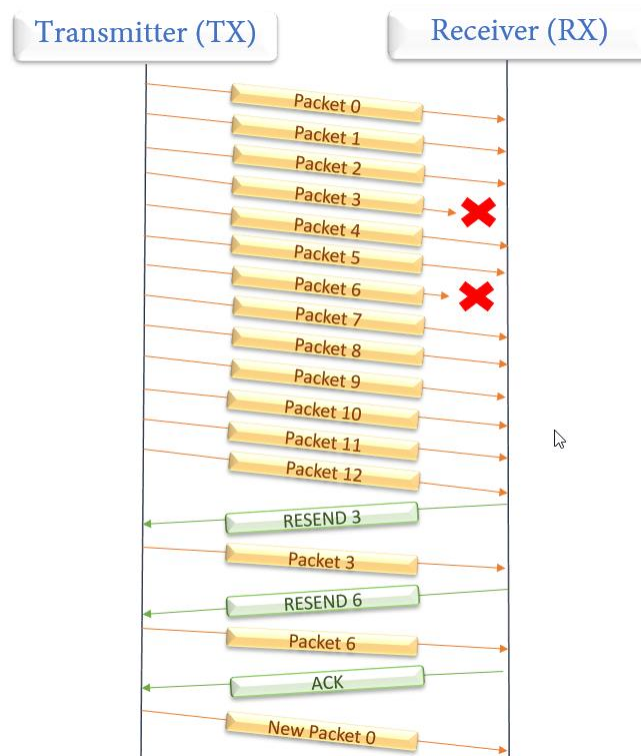


Figure 4.11: LoRa Multi-Packet Protocol with resending request.

4.4.2.1 TX

TX will begin to send the packets consecutively with interval of 5 seconds after the hexadecimal representation of image is stored in text file (ImageTx_h.txt). After sending out all the packets, TX will wait to receive ACK from RX. If the receiving ACK is integer n , this shows that the n th packet needs to be resent. TX will resend the packet and wait to receive ACK again. The receiving and resending processes will keep going until “DONE” ACK is received by RX. The new image transmission process will start after this or after all sessions are timeout. Figure 4.12 shows the flowchart for writing C program of TX when using LoRa Multi-Packet Transmission protocol.

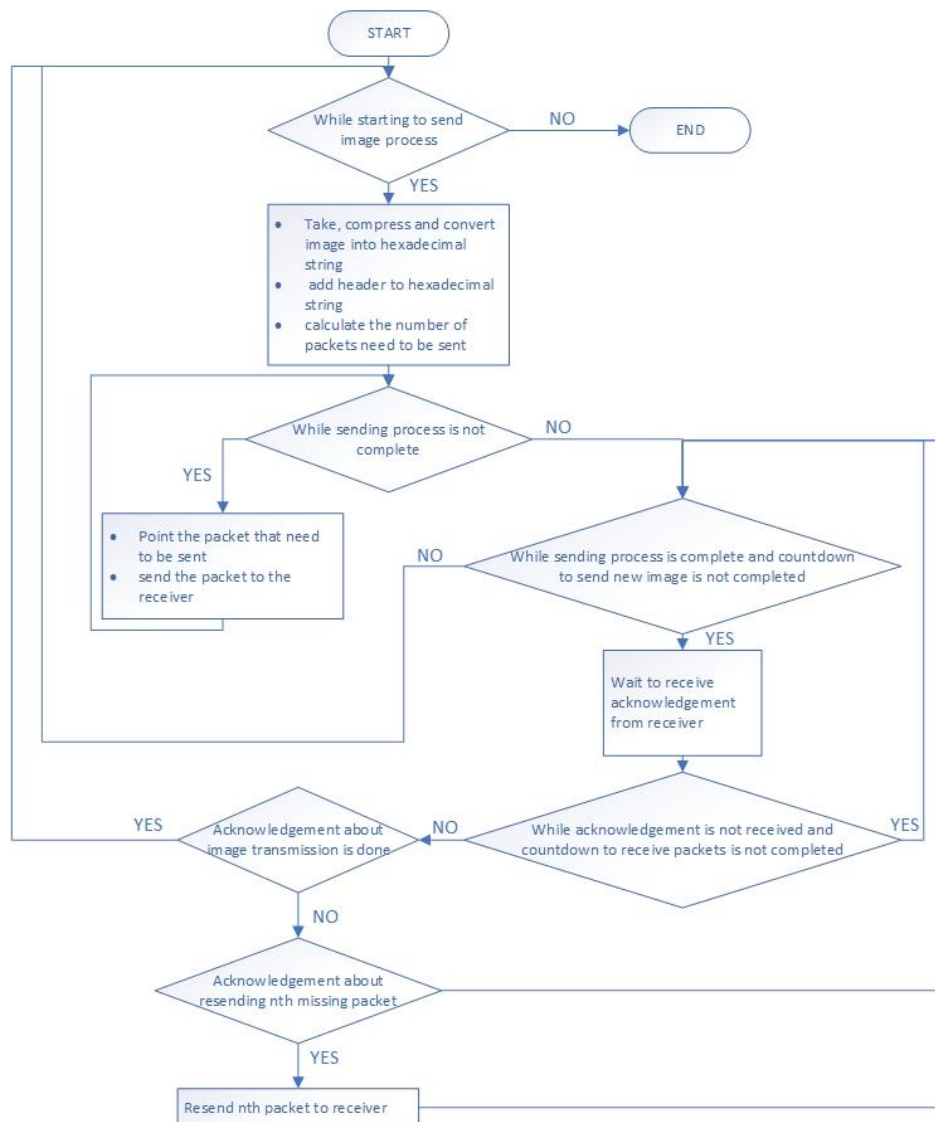


Figure 4.12: LoRa Multi-Packet Transmission Protocol's transmitter coding flowchart.

4.4.2.2 RX

RX will start to receive the packet sent from TX after receiving process is initiated. After the receiving session is expired, the receiver will check whether all the packets are received successfully or not. If there is missing packet, RX will send the integer n that represent n th missing packet to TX to request for resending. The sending and requesting processes are repeated until all the packets are received successfully by receiver. After receiving all the packets, RX will start to build the image and upload the image into Dropbox. RX will send a “DONE” ACK to TX after constructing the image. New receiving process will start after this. Figure 4.13 shows the flowchart for writing C program of RX when using LoRa Multi-Packet Transmission Protocol.

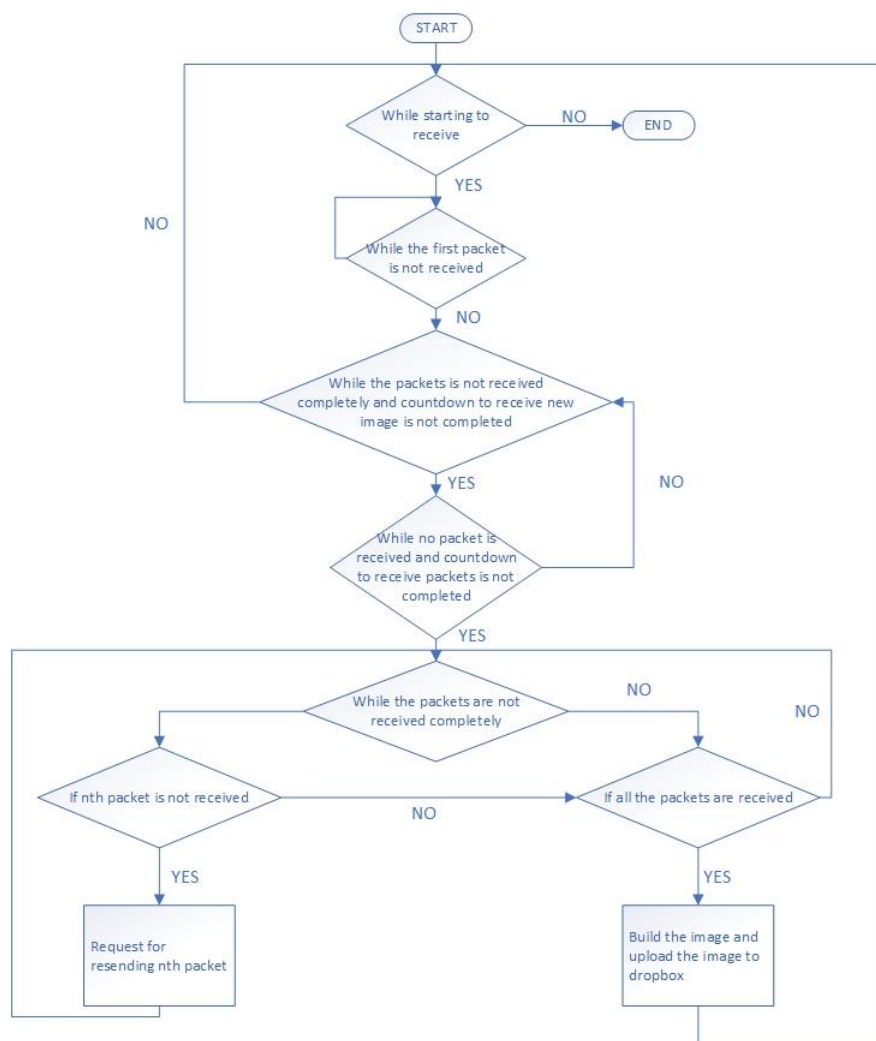


Figure 4.13: LoRa Multi-Packet Transmission Protocol’s receiver flowchart.

4.5 Image Processing

4.5.1 Image Compression

In many aspects, images can be significantly compressed without affecting their functionality. In this study, a pi camera that attached to RPI is used to capture a 3280 x 2464 full-resolution image which has size of 5.303 MB as shown in Figure 4.14. Python Imaging Library (PIL), Pillow is used to resize and compress the image. The 240 x 160 pixels downsized image in Figure 4.15 is compressed by an anti-aliasing filter which is a down sampling filter. To obtain even smaller size image, a parameter that represent the image quality which is range from 1 (most compression) to 95 (least compression) can be used to compress the image. This parameter can be adjusted according to the requirements of application. The codes shown below are used to compress and tune the quality of image.

```
from PIL import Image

# ImageTx.jpg is a 3280x2464 jpeg that is 5.303MB large
foo = Image.open("ImageTx.jpg")

# Resize the image with an ANTIALIAS filter (gives the highest quality)
foo = foo.resize((240,160),Image.ANTIALIAS)

# The saved downsized image size is 24KB
foo.save("ImageTx_compress_95.jpg",optimize=True,quality=95)
```

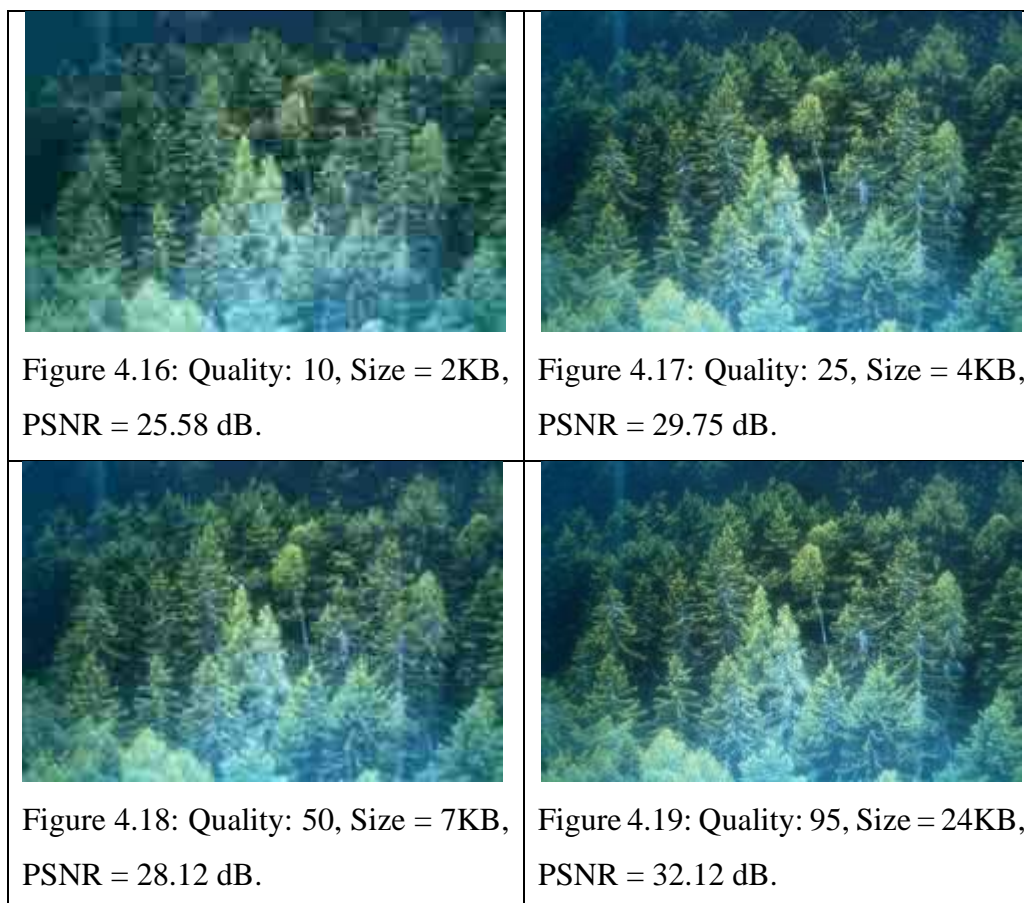


Figure 4.14: 3280 x 2464 full-resolution image (ImageTx.jpg).



Figure 4.15: 240 x 160 pixel resized image.

To show different levels of image quality compression, Figures 4.16 to 4.19 illustrate the results of compressed 240x160 pixel image after applying the algorithm of compression with quality of 10, 25, 50 and 95 respectively. The quality estimations with peak signal-to-noise ratio (PSNR) values also are conducted by using PSNR evaluation tool to compare the original image (Figure 4.15) and compressed images. The PSNR values are 25.58 dB, 28.12 dB, 29.75 dB and 32.12 dB for image quality of 10, 25, 50 and 95 respectively. The higher the PSNR, the better the quality of the compressed image. The smaller size of 7 KB and 4KB are obtained for the image quality of 50 and 25 respectively, some visible losses can be observed from the images. Besides that, there is notable image corruption can be observed from the image with quality of 10, but this quality may be tolerable in some applications that do not require clear image. Therefore, by using this compression algorithm, a 5.303 MB image can be compressed into an image of only 2KB, 4KB, 7KB and 24KB with different image quality settings. The image has more losses and smaller size with lower image quality. However, a small size image can ease the transmission with shorter transmission time and lower power consumption.



4.5.2 Image Conversion

When using the `open()` file function in python, the binary representation of image is obtained. However, the binary data is too large to be transferred using LoRa. In realization for transmission of image by using LoRa, the binary data of image is converted to hexadecimal format. `binascii.hexlify()` function is used to convert the binary data to hexadecimal representation. As a result, the 2-digit hexadecimal representation is obtained from every byte of data. Therefore, the returned bytes object is 2 times longer than the length of data. Table 4.3 shows the lengths of data after converting Figures 4.16 to 4.19 into hexadecimal representation. The hexadecimal representation of image will be stored in a text file (`ImageTx.txt`) for further usage.

```

import binascii
filename = 'ImageTx_compress_95.jpg'

#read the image file
with open(filename, 'rb') as f:
    content = f.read()

#convert binary data into hexadecimal
str_content = str(binascii.hexlify(content), 'utf-8')

#store the hexadecimal representation in text file
txtfile = 'ImageTx.txt'
with open(txtfile, 'w+') as file:
    file.write(str_content)
file.close()

```

The ID of the packets are appended to the existing hexadecimal representation stored in ImageTx.txt file using + function in python and then saved in the text file (ImageTx_h.txt) as shown in code below. Table 4.3 shows the lengths of data after appending IDs to the hexadecimal representations.

```

import binascii
import math

PAYLOADSIZE=251
HEADERSIZE=4

#non-appended file name
filename = 'ImageTx.txt'

with open(filename, 'rb') as f:
    my_str = f.read()

#calculate the number of packets
length = len(my_str)
packet_no = math.ceil(length/float(PAYLOADSIZE))
packet_no= int(packet_no)

#append the id to all the payload
for i in range (1,packet_no):
    i_str=str(i+1).zfill(HEADERSIZE)
    j=i-1
    my_str=my_str[(PAYLOADSIZE+(j*(PAYLOADSIZE+HEADERSIZE)))] + i_str + my_str[(PAYLOADSIZE+(j*(PAYLOADSIZE+HEADERSIZE)))]

#the first packet in payload that represent the number of packets to be transmitted
my_str=my_str[:0] + str(1).zfill(HEADERSIZE) + my_str[0:]
packet_no_char = str(packet_no).zfill(PAYLOADSIZE)
packet_no_char = packet_no_char[:0] + str(0).zfill(HEADERSIZE) + packet_no_char[0:]

#append the first packet to another packet
my_str = packet_no_char+ my_str

#store the appended string in to text file
txtfile = 'ImageTx_h.txt'
with open(txtfile, 'w+') as file:
    file.write(my_str)
file.close()

```

Table 4.3: Results of image conversion.

Quality of image	Size of image (bytes)	Size of image in hexadecimal representation (bytes)	Size of image in hexadecimal representation after adding header (bytes)

10	1,970	3,940	4,196
25	4,056	8,112	8,504
50	6,516	13,032	13,584
95	23,613	47,226	48,878

4.5.3 Image Building

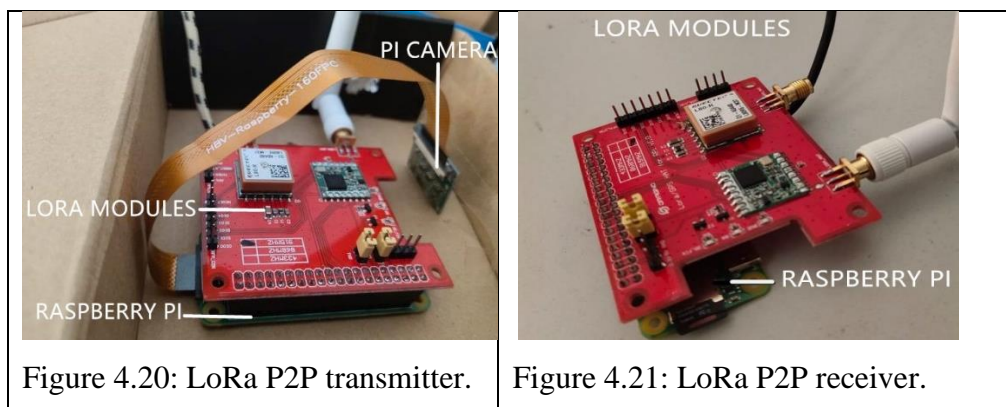
For image building in RX, RX will start to listen continuously until the arrival of the packet. In this study, each packet size is set to 128 bytes including 4 bytes of identity number (ID) and 124 bytes of useful information. All the useful information received will be stored in a text file (ImageRx.txt). If all the packets are received by RX, the image reconstruction process will start immediately. Image reconstruction is done by converting the hexadecimal representation data (useful information) to original image (ImageRx.jpg). This image (ImageRx.jpg) will be saved in the SD card and readable with image viewer. The image will be uploaded to Dropbox with the access of internet.

4.6 Design of Transmission Protocol

4.7 Performance evaluation of LoRaP2P

In this study, Dragino LoRa module that attached to Raspberry Pi Zero W and pi camera is used as TX as shown in Figure 4.20 while Dragino LoRa module that attached to Raspberry Pi Zero W is used as RX as shown in Figure 4.21. The performance evaluation consists of:

1. The transmission time when using stop and wait transmission protocol and LoRa Multi-Packet Transmission Protocol.
2. The impacts on transmission time and number of packet loss when using different image qualities.
3. The impacts on transmission time, number of packet loss, RSSI and SNR under different transmission distances.



4.7.1 Stop and wait transmission protocol vs. LoRa Multi-Packet Transmission Protocol

4.7.1.1 Experimental setup

1. TX and RX are powered on.
2. TX and RX are monitored and controlled by using VNC viewer through SSH connection in a PC as shown in Figure 4.22.
3. The fixed parameters are configured as stated in Table 4.4.
4. The stop and wait transmission protocol is used.
5. The sending and receiving processes are initiated.
6. The total transmission time is recorded.
7. Steps 1 to 6 are repeated by introducing 1 packet loss artificially.
8. Steps 1 to 7 are repeated by using the LoRa Multi-Packet Transmission Protocol instead of stop and wait transmission protocol.
9. For each protocol, the experiments are repeated 3 times, all the transmission time are averaged and presented in Table 4.5.

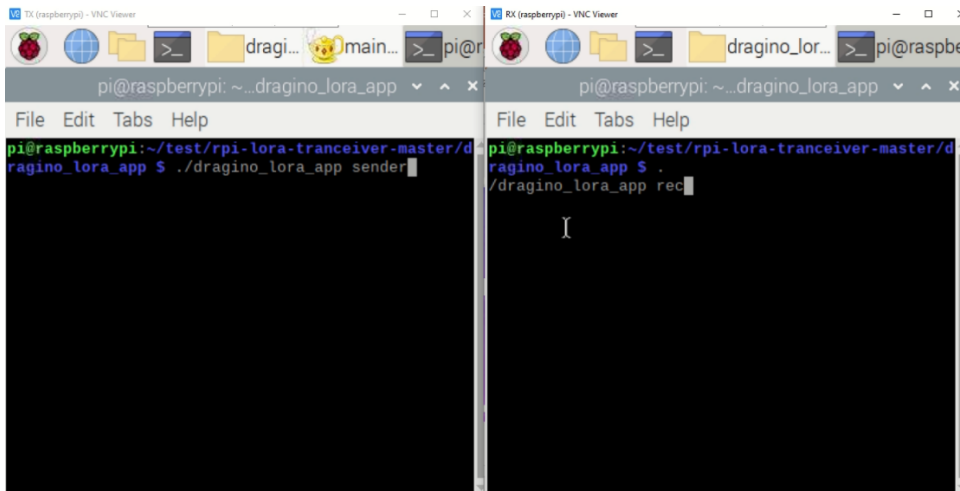



Figure 4.22: VNC viewer monitoring and controlling.

Table 4.4: Configuration parameters.

Image that needs to be transmitted	
Total Payload	1510 bytes
Payload length per packet	128 bytes
Number of packets	11
Coding Rate	4/5
Distance between TX and RX	0 m
SF	7
Bandwidth	125 kHz

4.7.1.2 Result and discussion

In order to confirm whether the proposed LoRa Multi-Packet Transmission Protocol is more beneficial than the stop and wait transmission protocol in term of time spent. The LoRa P2P image transmission experiments are conducted in an indoor environment to investigate the transmission performance of both protocols. All the total transmission time are based on the average values of 3 runs as shown in Table 4.5.

Table 4.5: Average transmission time when using stop and wait protocol and LoRa Multi-Packet Transmission Protocol.

Protocol	Stop and wait protocol	LoRa Multi-Packet Transmission Protocol
Average transmission time without packet loss	3 minutes 36 seconds	1 minutes 6 seconds
Average transmission time with 1 packet loss	4 minutes 21 seconds	2 minutes 34seconds

When using stop and wait protocol without any packet loss experienced, the transmission time is 3 minutes and 36 seconds. However, when using LoRa Multi-Packet Transmission Protocol with same scenario, the transmission time is 1 minute and 6 seconds only which are 2 minutes 30 seconds shorter than the time stop and wait protocol spent. When introducing 1 packet loss, LoRa Multi-Packet Transmission Protocol, the transmission time is almost 2x shorter than that of stop and wait protocol. A significant increment in transmission time is noticeable when using stop and wait protocol because the transmitter needs to wait for an ACK per packet to make sure the success arrival of data which causes the transmit rate of packet greatly reduced. The load of network and the receiver's required transmission rate have been increased by the ACK traffic. LoRa Multi-Packet Transmission Protocol significantly decreases the number of ACKs and reduces the time for waiting ACKs. Therefore, the data transfer efficiency is extremely increased with LoRa Multi-Packet Transmission Protocol.

4.7.2 The impacts on number of packet loss, transmission time, RSSI and SNR under different transmission distances


The experiments are carried out in the high raised building of Evergreen Scotpine Condominium as same as the location in Chapter 3.8.2.

4.7.2.1 Experimental Setup

1. TX and RX are powered on.
2. TX and RX are monitored and controlled by using mobile SSH and VNC viewer through SSH connection respectively.

3. The fixed parameters are configured as stated in Table 4.6.
4. RX is placed at G floor of the building as shown in Figure 4.24 and indicated by the blue point in Figure 4.23.
5. TX is placed at 1st floor of the building as shown in Figure 4.26 and represented by red point in Figure 4.25.
6. The sending and receiving processes are initiated.
7. The number of packet loss, transmission time, RSSI and SNR of the nodes are measured and recorded from VNC viewer and mobile SSH as shown in Figure 3.27 and Figure 3.28.
8. The experiment is repeated by placing RX at G, 2nd, 3rd, 4th, 5th, 6th, 7th and 8th floors.
10. For each variation of distance, the experiments are repeated 3 times. The results are averaged as shown in Table 4.7.

Table 4.6: Configuration parameters.

Image that needs to be transmitted	
Total Payload	1510 bytes
Payload length per packet	128 bytes
Number of packet	11
Coding Rate	4/5
SF	7
Bandwidth	125kHz

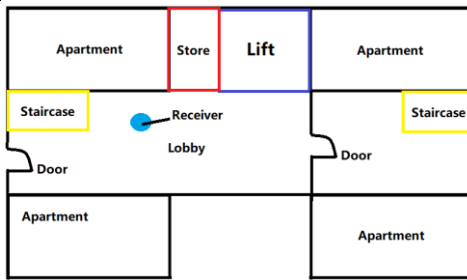


Figure 4.23: Floor map of Evergreen Scotpine Condominium (Receiver).



Figure 4.24: Placement of receiver at G floor.

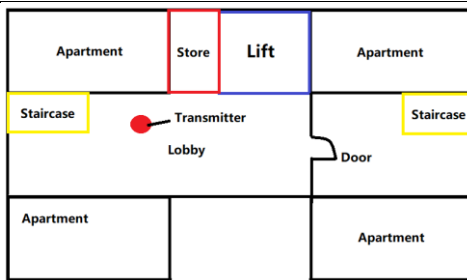


Figure 4.25: Floor map of Evergreen Scotpine Condominium (Transmitter).

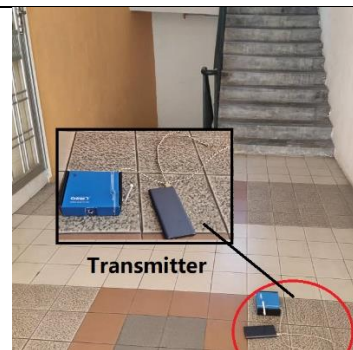


Figure 4.26: Placement of transmitter at 1st floor.

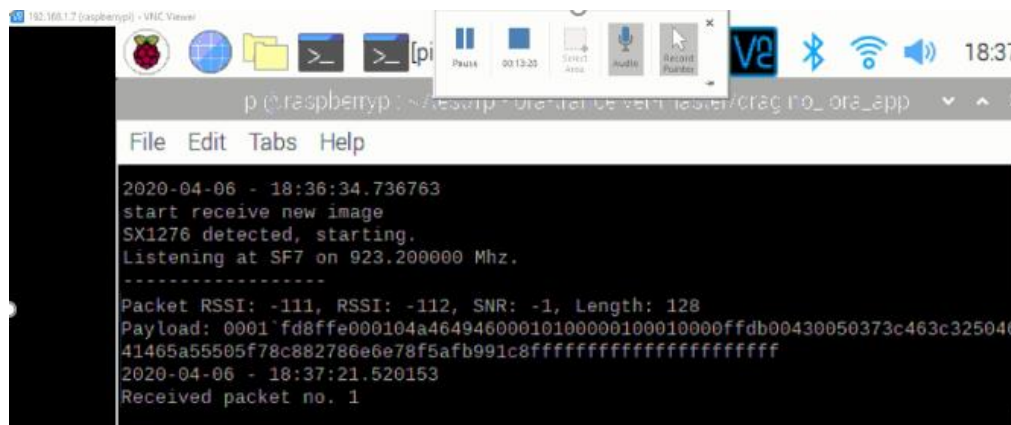
```

Mobile SSH
packet size is 12
Send packets at SF7 on 923.200000 Mhz.

Payload length is 128
2020-04-06 - 18:37:15.306353
Sending packet no. 0
send: 000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000011
Sent packet no. 0

Payload length is 128
2020-04-06 - 18:37:20.843656
Sending packet no. 1
send: 0001ffd8ffe000104a464946000101000001000100
00ffdb00430050373c463c32504641465a55505f78c882
786e6e78f5afb991c8ffffffffffffffffffff
Sent packet no. 1
  
```

Figure 4.27: Transmitter monitoring using mobile SSH through SSH connection.



```

2020-04-06 - 18:36:34.736763
start receive new image
SX1278 detected, starting.
Listening at SF7 on 923.200000 Mhz.
-----
Packet RSSI: -111, RSSI: -112, SNR: -1, Length: 128
Payload: 0001`fd8ffe000104a46494600010100000100010000ffdb00430050373c463c325048
41465a55505f78c882786e6e78f5afb991c8f
2020-04-06 - 18:37:21.520153
Received packet no. 1

```

Figure 4.28: Receiver monitoring using VNC viewer in laptop through SSH connection.

4.7.2.2 Result and discussion

The experiments are carried out to investigate the number of packet loss, transmission time, RSSI and SNR by placing the RX at different floors (distances) within Evergreen Scotpine Condominium. Table 4.7 shows the results of the experiments.

Table 4.7: Results of the impacts on number of packet loss, transmission time, RSSI and SNR under different transmission distances.

Floor	Distance (m)	Number of packet loss	Transmission time	Average RSSI (dBm)	Average SNR (dB)
G	0	0	1 minute 6 seconds	-44.3	9.0
1	5	0	1 minute 6 seconds	-90.4	8.9
2	10	1	1 minute 34 seconds	-102.7	8.3
3	15	1	1 minute 39 seconds	-103.3	9.0
4	20	2	2 minutes 17 seconds	-103.1	8.1
5	25	2	2 minutes 18 seconds	-108.2	8.2

6	30	6	5 minutes 21 seconds	-112.0	4.1
7	35	16	Expired at 11 minutes 8 seconds	-112.0	-2.0
8	40	All	-	-	-

The average RSSI when the transmitter is placed at G floor is -44.3 dBm but the drastic drop of average RSSI to -90.4 dBm occurs when the TX is placed at 1st floor. The average RSSIs are almost same for the 2nd, 3rd and 4th floors. The average RSSIs keep decreasing to -108.2 dBm at 5th floor and -112 dBm at 6th floor and 7th floor. This shows that this environment has great influences on RSSI. Besides that, the average SNR values are approximately 8 dB or 9 dB when the TX is placed at G, 1st, 2nd, 3rd, 4th and 5th floor as shown in Table 4.7. However, the average SNR values drop when the transmitter is located at the 6th floor and 7th floor which are 4.1 dB and -2.0 dB respectively. The distance between TX and RX increases, both RSSI and SNR decreases. There are many factors that can affect the propagation of transmitted signal. One of the factors is the blocking of obstacles which may absorb the signal transmitted and interfere the signal constructively or destructively. Moreover, the signal could arrive the RX by multiple paths, causing an effect called multipath propagation.

Apart from RSSI and SNR, the number of packet loss increases as the distance increases which leads to the increment in transmission time as the time-consuming retransmission is required to be done. The shortest transmission time which is 1 minute and 6 seconds and no packet is lost when the TX is placed at G and 1st floors. When the TX is placed at 6th floor, the transmission time dramatically increases to 5 minutes 21 seconds which is about 5x longer than the shortest transmission time. This is because 6 packets are lost during the transmission, thus longer time is needed to complete the transmission and retransmission of lost packet. Experimental results show that the communications can be done within 35m only. When TX is placed at 7th floor, although some packets are received by RX successfully but unfortunately the

transmission is not complete as RX are unable to receive all the packets successfully before all the sessions are expired. When the receiver is placed at 8th floor which is 40m apart from the transmitter, all the packets totally unable to be received.

4.7.3 The impacts on number of packet loss and transmission time when using different image qualities

4.7.3.1 Experimental Setup

1. TX and RX are powered on.
2. TX and RX are monitored and controlled by using VNC viewer through SSH connection in a PC.
3. The fixed parameters are configured as stated in Table 4.8.
4. The image with quality of 10 (Figure 4.16) is used as the image that needs to be transmitted.
5. The sending and receiving processes are initiated.
6. The number of packet, number of packet loss and transmission time are recorded.
7. The experiments are repeated by using the image with quality of 25, 50 and 95 (Figures 4.16 to 4.19).

Table 4.8: Configuration parameters.

Protocol used	LoRa Multi-Packet Transmission Protocol
Payload length per packet	128 bytes
Coding Rate	4/5
Distance between transmitter and receiver	0 m
SF	7
Bandwidth	125kHz

4.7.3.2 Result and discussion

To evaluate the performance to transmit different quality images using LoRa P2P, the experiments are conducted. Table 4.9 gives the results of effect of quality of image on number of packet loss and transmission time.

Table 4.9: Results of effect of quality of image on number of packet loss and transmission time.

Quality of image	Payload size (bytes)	Number of packet	Number of packet loss	Transmission time
10	4196	33	0	3 minutes 1 second
25	8504	67	2	12 minutes 32 seconds
50	13584	107	1	24 minutes 40 seconds
95	48878	382	9	42 minutes 54 seconds

For image quality of 10, 25, 50 and 95, the payload length are 4196 bytes, 8504 bytes, 13584 bytes and 48878 bytes respectively. With image quality of 10, no packet is lost when transmitting all the 33 packets and the transmission time is 3 minutes 1 second. The number of packets with image quality of 25 is almost double as compared to quality of 10 and its transmission time is almost 4x longer than that of image quality of 10. With image quality of 50, only 1 packet is lost and transmission time is 24 minutes and 40 seconds. When using the highest quality of image which is 95, the payload length is 48878 bytes which is very large as compared to other quality. Not surprisingly, the number of packet loss is highest among all qualities tested which is 9 and it took 42 minutes and 54 seconds to do all the transmission.

The higher the quality of image, the higher the number of packet. This causes higher possibility of packet loss. In addition, each packet loss leads to retransmission and this will increase the transmission time. Therefore, trade-off must be made. Although there is detectable image corruption when the quality

of image is 10 but the transmission time is preferable and this may be tolerable in some applications.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

There are many LPWAN technologies exist in the world, however, LoRaWAN is getting popularity in applications of IoT because of its unique characteristic and operates on unlicensed spectrum that are free to be used. In this project, the indoor performance of LoRaWAN is studied by variety of experiments. The results show that the higher the spreading factor, the better the reconstitution of the signal but the longer the airtime. Besides that, the longer the length the payload sent, the longer the airtime required as larger payload needed to be encoded by the end-device and decoded by the gateway. The results obtained also show that the LoRaWAN is suitable to be used for indoor applications. This is because the whole building can be covered even though there is only one gateway used. However, there are some researchers pointed out that LoRaWAN networks should be optimized for different use cases.

For LoRa P2P image transmission, the image processing is first conducted in order to compress the image to smaller size. The compressed image is converted into hexadecimal format to allow it to be transmitted using LoRa. The protocols stop and wait transmission protocol and LoRa Multi-Packet Transmission Protocol are designed and implemented for LoRa P2P image transmission. Both protocols are implemented and their performance evaluations in term of transmission time are carried out. The results show that time spent of LoRa Multi-Packet Transmission Protocol for 11 packets image transmission is shorter than stop and wait transmission protocol by 69.44 %. When 1 packet is lost, LoRa Multi-Packet Transmission Protocol also saves about 50 % of time for transmission. Besides that, for higher quality of image transmitted, more time is needed, thus the quality of image needs to be chosen wisely depends on the requirement of the applications. When the LoRa P2P image transmission being tested in an indoor environment, the image can be received as long as the distance is within 35 m.

5.2 Recommendations for future work

The power consumption of devices cannot be recorded in the experiment due to limitations of equipment. Therefore, the power consumption of the devices cannot be identified. It is important to know about power consumption because it is one of the main factors that regulate wireless sensor networks' performance and maintain the network lifetime. The low power consumption of LoRa needs to be further investigated.

Besides that, the experiments are carried out at an indoor environment. More real-world experiments need to be conducted to explore the scalability and reliability of LoRa to work for real deployment environments.

In addition, to reduce the number of bits that need to be sent, there are several image compression algorithms available. JPEG 2000 is one of the image compression standards that has can produce better quality of image with better performance of compression to reduce the packet size significantly.

Last but not least, the security is not the main focus in this project but it is very important in real-work implementation. Without security measures, network security threats will be faced by the WSN. Thus, security issue needs to be taken into account and proper security mechanism such as authentication and authorization need to be implemented into protocol to protect the WSN from the threats.

REFERENCES

- Adhikary, A., Lin, X. and Eric Wang, Y.P., 2017. Performance evaluation of NB-IoT coverage. *IEEE Vehicular Technology Conference*, pp.1–5.
- Ayoub, W. et al., 2018. Internet of Mobile Things : Overview of LoRaWAN , DASH7 , and NB-IoT in LPWANs standards and Supported Mobility To cite this version : HAL Id : hal-01901612 Internet of Mobile Things : Overview of LoRaWAN , DASH7 , and NB-IoT in LPWANs standards and Suppo.
- Buyukakkaslar, M.T., Erturk, M.A., Aydin, M.A. and Vollero, L., 2017. LoRaWAN as an e-Health Communication Technology. *Proceedings - International Computer Software and Applications Conference*, 2, pp.310–313.
- De Carvalho Silva, J. et al., 2017. LoRaWAN - A low power WAN protocol for Internet of Things: A review and opportunities. *2017 2nd International Multidisciplinary Conference on Computer and Energy Science, SpliTech 2017*, (August).
- Chen, T., Eager, D. and Makaroff, D., 2019. Efficient image transmission using lora technology in agricultural monitoring iot systems. *Proceedings - 2019 IEEE International Congress on Cybermatics: 12th IEEE International Conference on Internet of Things, 15th IEEE International Conference on Green Computing and Communications, 12th IEEE International Conference on Cyber, Physical and So*, pp.937–944.
- Chou, Y. et al., 2017. i-Car System A LoRa-based Low Power Wide Area Networks Vehicle Diagnostic. , pp.789–791.
- Dogan, G., Yildirim, G. and Tatar, Y., 2019. Empirical Observations on LoRa Performance for Different Environments. *Proceedings - 2019 3rd International Conference on Applied Automation and Industrial Diagnostics, ICAAID 2019*, 1(September), pp.1–6.
- Ertürk, M.A., 2017. Lorawan indoor performance analysis. *International Research Journal of Computer Science (IRJCS)*, 4(10), pp.23–29.
- Galinina, O., Andreev, S., Balandin, S. and Koucheryavy, Y., 2018. *Correction to: Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, Springer International Publishing.
- Help Net Security, 2019, *Number of connected devices reached 22 billion, where is the revenue? - Help Net Security* [Online]. Available at: <https://www.helpnetsecurity.com/2019/05/23/connected-devices-growth/>.
- Hoeller, A. et al., 2018. Analysis and Performance Optimization of LoRa Networks with Time and Antenna Diversity. *IEEE Access*, 6, pp.32820–32829.

Jebril, A.H., Sali, A., Ismail, A. and Rasid, M.F.A., 2018. Overcoming limitations of LoRa physical layer in image transmission. *Sensors (Switzerland)*, 18(10).

Ji, M. et al., 2019. LoRa-based Visual Monitoring Scheme for Agriculture IoT. *SAS 2019 - 2019 IEEE Sensors Applications Symposium, Conference Proceedings*. 2019

Lavric, A. and Popa, V., 2017. Internet of things and LoRaTM low-power wide-area networks challenges. *Proceedings of the 9th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2017*, 2017-Janua, pp.1–4.

Masadan, N.A.B., Habaebi, M.H. and Yusoff, S.H., 2018. LoRa LPWAN Propagation Channel Modelling in IIUM Campus. *Proceedings of the 2018 7th International Conference on Computer and Communication Engineering, ICCCE 2018*, pp.14–19.

MCMC, 2019. Class-Assignment-No-2-of-2019.pdf. , p.18,19.

Mekki, K., Bajic, E., Chaxel, F. and Meyer, F., 2019. A comparative study of LPWAN technologies for large-scale IoT deployment. *ICT Express*, 5(1), pp.1–7. Available at: <https://doi.org/10.1016/j.icte.2017.12.005>.

Muaz Abdul Rahman, A., Hafizhelmi Kamaru Zaman, F. and Afzal Che Abdullah, S., 2018. Performance Analysis of LPWAN Using LoRa Technology for IoT Application. *International Journal of Engineering & Technology*, 7(4.11), p.252.

Pham, C., 2018. Enabling and deploying long-range IoT image sensors with LoRa technology. *2018 IEEE Middle East and North Africa Communications Conference, MENACOMM 2018*, pp.1–6.

Pham, C., 2016. Low-cost, low-power and long-range image sensor for visual surveillance. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 03-07-Octo, pp.35–40.

Python, *binascii — Convert between binary and ASCII — Python 3.8.2 documentation* [Online]. Available at: <https://docs.python.org/3/library/binascii.html> [Accessed: 3 April 2020].

Reynders, B., Meert, W. and Pollin, S., 2016. Range and coexistence analysis of long range unlicensed communication. *2016 23rd International Conference on Telecommunications, ICT 2016*, (c), pp.1–6.

Rosmiati, M., Rizal, M.F., Susanti, F. and Alfisyahrin, G.F., 2019. Air pollution monitoring system using LoRa modul as transceiver system. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 17(2), p.586.

Rubio-Aparicio, J., Cerdan-Cartagena, F., Suardiaz-Muro, J. and Ybarra-Moreno, J., 2019. Design and implementation of a mixed IoT LPWAN network architecture. *Sensors (Switzerland)*, 19(3).

Sanchez-Iborra, R. et al., 2018. Performance evaluation of lora considering scenario conditions. *Sensors (Switzerland)*, 18(3).

Security, S., 2020, *Incorporating The IoT To Improve Connectivity To Your Customers* [Online]. Available at: <https://www.gosolis.com/blog/incorporating-the-iot-to-improve-connectivity-to-your-customers/>.

TheThingsNetwork, 2020, *Limitations: data rate, packet size, 30 seconds uplink and 10 messages downlink per day Fair Access Policy [guidelines] - End Devices (Nodes) - The Things Network* [Online]. Available at: <https://www.thethingsnetwork.org/forum/t/limitations-data-rate-packet-size-30-seconds-uplink-and-10-messages-downlink-per-day-fair-access-policy-guidelines/1300>.

TheThingsNetwork, *LoRaWAN Frequencies Overview* [Online]. Available at: <https://www.thethingsnetwork.org/docs/lorawan/frequency-plans.html>.

TheThingsNetwork, *LoRaWAN Frequency Plans and Regulations by Country* [Online]. Available at: <https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country.html>.

Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara, 2017. Low Power, Wide Area Networks networks (LPWANs). , 19(2), pp.855–873.

Vatcharatiansakul, N., Tuwanut, P. and Pornavalai, C., 2017. Experimental performance evaluation of LoRaWAN: A case study in Bangkok. *Proceedings of the 2017 14th International Joint Conference on Computer Science and Software Engineering, JCSSE 2017*.

Wan, X.F., Yang, Y., Cui, J. and Sardar, M.S., 2018. Lora propagation testing in soil for wireless underground sensor networks. *2017 IEEE 6th Asia-Pacific Conference on Antennas and Propagation, APCAP 2017 - Proceeding*, pp.1–3.

Wang, Y.P.E. et al., 2017. A Primer on 3GPP Narrowband Internet of Things. *IEEE Communications Magazine*, 55(3), pp.117–123.

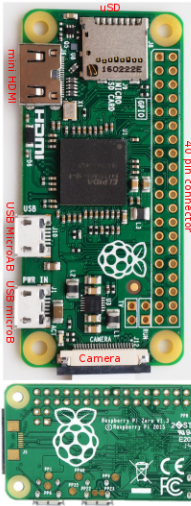
Widianto, E.D., Pakpahan, M.S.M., Faizal, A.A. and Septiana, R., 2019. LoRa QoS Performance Analysis on Various Spreading Factor in Indonesia. *ISESD 2018 - International Symposium on Electronics and Smart Devices: Smart Devices for Big Data Analytic and Machine Learning*.

Xue-Fen, W. et al., 2018. Smartphone based LoRa in-soil propagation measurement for wireless underground sensor networks. *2017 IEEE Conference on Antenna Measurements and Applications, CAMA 2017*, 2018-Janua, pp.114–117.

APPENDICES

APPENDIX A: Raspberry Pi Zero W Datasheet

Raspberry Pi Zero v1.3

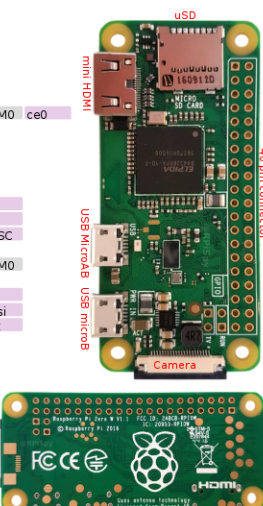


Position	Power	Ground	Control	GPIO
	Wiring	BCM	Serial	PWM Misc

Different places use different pin numbers
GPIO, Wiring, and BCM have been included.

3.3V	1	2	5V	
SDA	8	2	3	4
SCL	9	3	5	6
GPIOK0	4	7	4	7
		GND	9	10
ce1	17	0	17	11
	27	2	27	13
	22	3	22	15
		3.3V	17	16
MOSI	12	10	19	20
MISO	13	9	21	22
SCLK	14	11	23	24
		GND	25	26
ID_SD	30	0	DNC	27
GPIOK1	5	21	5	29
GPIOK2	6	22	6	31
PWM1	13	23	13	33
	19	24	19	35
	26	25	26	37
		GND	39	40
		TV +	TV	Run
		TV -	TV	Run

Raspberry Pi Zero W v1.1



GPIO 0 and 1 are reserved - Do Not Connect
 PAL or NTSC via composite video on TV pads
 Run - temporarily connect pins to reset chip (or start chip after a shutdown)
 Camera Connector (not on Zero 1.1 or 1.2) - 22pin, 0.5mm
 Board Dimensions - 65mm x 30mm x 0.2mm,
 mounting holes M2.5

Processor - BCM2835
 ARM v7
 Single Core
 1GHz
 (same as B+ and A+)

Memory
 512MB RAM
 uSD slot to run OS

Video
 mini HDMI
 PAL or NTSC via pads
 HDMI capable of 1080p

USB
 microB for power
 microAB for OTG

Audio
 from HDMI port only

Wireless
 2.4GHz
 802.11n
 Bluetooth 4.1/BLE

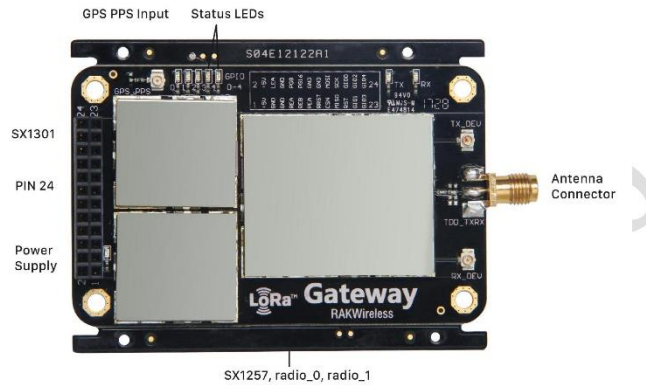
APPENDIX B: RAK 831 datasheet



RAK831 Datasheet

2.Module Package

In the following the RAK831 module package is described. This description includes the RAK831 pinout as well as the modules dimensions.



2.1 Pinout Description

The RAK831 provides headers at the bottom side, which have a pitch of 2.54 mm. The description of the pins is given by below Table .

+5V	1	+5V	2
GND	3	LNA_EN_A	4
GND	5	GND	6
RADIO_EN_A	7	PA_G8	8
RADIO_EN_B	9	PA_G16	10
PA_EN_A	11	GND	12
RADIO_RST	13	GND	14
CSN	15	MOSI	16
MISO	17	SCK	18
RESET	19	GPIO0	20
GPIO1	21	GPIO2	22
GPIO3	23	GPIO4	24

Pin	Name	Type	Description
1	+5V	POWER	+5V Supply Voltage
2			
3	GND	GND	GND
4	LNA_EN_A	Input	SX1301 Radio C Sample Valid
5	GND	GND	GPS Module LDO:Enable Pin

5

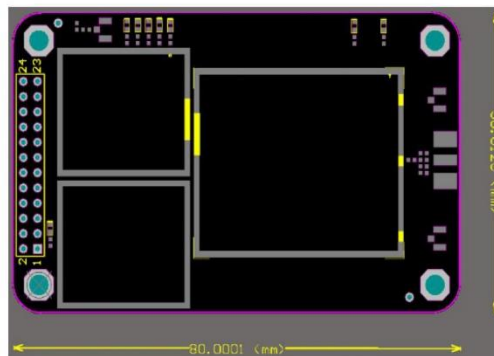
COPYRIGHT ©
SHENZHEN RAKWIRELESS TECHNOLOGY CO., LTD



6	GND	GND	GND
7	RADIO_EN_A	Input	SX1257_A_EN
8	PA_G8	Input	PA GAIN 0
9	RADIO_EN_B	Input	SX1257_B_EN
10	PA_G16	Input	PA GAIN 1
11	PA_EN_A	Input	PA EN
12	GND	GND	GND
13	RADIO_RST	RST	SX1257_A_B RESET
14	GND	GND	GND
15	CSN	SPI	SX1301 SPI_NSS
16	MOSI	SPI	SX1301 SPI_MOSI
17	MISO	SPI	SX1301 SPI_MISO
18	SCK	SPI	SX1301 SPI_CLK
19	RESET	RST	SX1301 RESET
20	GPIO0	GPIO	SX1301 GPIO
21	GPIO1	GPIO	SX1301 GPIO
22	GPIO2	GPIO	SX1301 GPIO
23	GPIO3	GPIO	SX1301 GPIO
24	GPIO4	GPIO	SX1301 GPIO

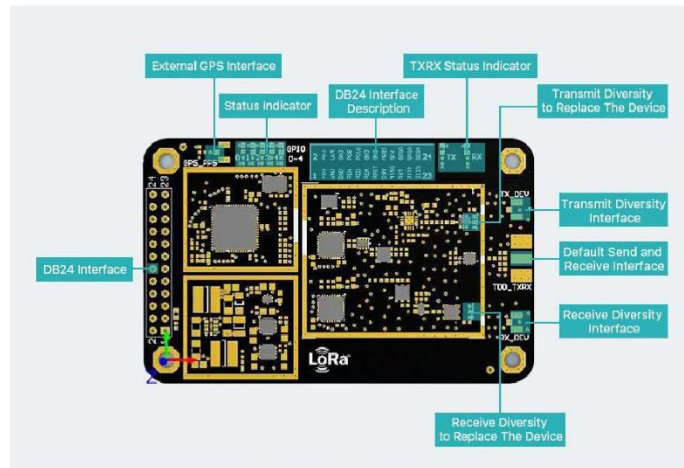
2.2 Module Dimensions

The outer dimensions of the RAK831 are given by 80.0 x 50.0 mm \pm 0.2 mm. The RAK831 provide four drills for screwing the PCB to another unit each with a drill diameter of 3 mm.



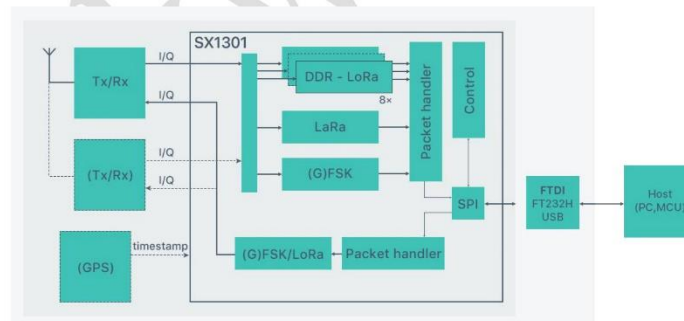
3.Module Overview

The Concentrator Module is currently available in one versions with SPI interface.



3.1 SX1301

The RAK831 includes Semtech's SX1301 which is a digital baseband chip including a massive digital signal processing engine specifically designed to offer breakthrough gateway capabilities in the ISM bands worldwide. It integrates the LoRa concentrator IP.



The SX1301 is a smart baseband processor for long range ISM communication. In the receiver part, it receives I and Q digitized bit stream for one or two receivers (SX1257), demodulates these signals using several demodulators, adapting the demodulators settings to the received signal and stores the received demodulated packets in a FIFO to be retrieved from a host system (PC, MCU). In the transmitter part, the packets are modulated using a programmable (G)FSK/LoRa modulator and sent to one transmitter (SX1257). Received packets can be time-stamped using a GPS PPS

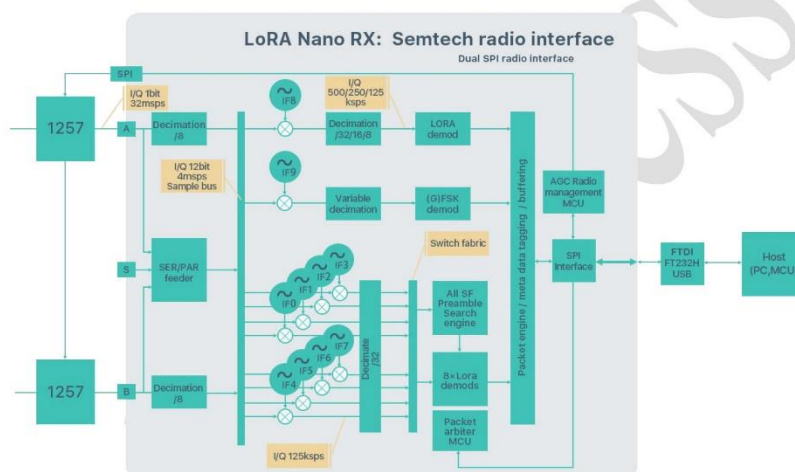
input.

The SX1301 has an internal control block that receives microcode from the host system (PC, MCU). The microcode is provided by Semtech as a binary file to load into the SX1301 at power-on (see Semtech application support for more information).

The control of the SX1301 by the host system (PC, MCU) is made using a Hardware Abstraction Layer (HAL). The Hardware Abstraction Layer source code is provided by Semtech and can be adapted by the host system developers.

It is highly recommended to fully re-use the latest HAL as provided by Semtech on <https://github.com/Lora-net>.

3.1.1 Block Diagram



The SX1301 digital baseband chip contains 10 programmable reception paths. Those paths have differentiated levels of programmability and allow different use cases. It is important to understand the differences between those demodulation paths to make the best possible use from the system.

3.1.2 IF8 LORA channel

This channel is connected to one SX1257 using any arbitrary intermediate frequency within the allowed range. This channel is LoRa only. The demodulation bandwidth can be configured to be 125, 250 or 500 kHz. The data rate can be configured to any of the LoRa available data rates (SF7 to SF12) but, as opposed to IF0 to IF7, only the configured data rate will be demodulated. This channel is intended to serve as a high speed backhaul link to other gateways or infrastructure equipment. This demodulation path is compatible with the signal transmitted by the SX1272 and SX1276 chip family.

3.1.3 IF9 (G) FSK channel

The IF9 channel is connected to a GFSK demodulator. The channel bandwidth and bit rate can be adjusted. This demodulator offers a very high level of configurability, going well beyond the scope of this document. The demodulator characteristics are essentially the same than the GFSK demodulator implemented on the SX1232 and SX1272 Semtech chips. This demodulation path can demodulate any legacy FSK or GFSK formatted signal.

3.1.4 IF0 to IF7 LORA channels

Those channels are connected to one SX1257. The channel bandwidth is 125 kHz and cannot be modified or configured. Each channel IF frequency can be individually configured. On each of those channels any data rate can be received without prior configuration.

Several packets using different data rates (different spreading factors) may be demodulated simultaneously even on the same channel. Those channels are intended to be used for a massive asynchronous star network of 10000's of sensor nodes. Each sensor may use a random channel (amongst IF0 to IF7) and a different data rate for any transmission.

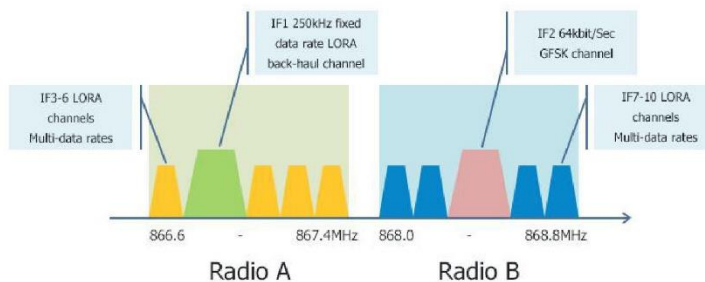
Sensors located near the gateway will typically use the highest possible data rate in the fixed 125 kHz channel bandwidth (e.g. 6 kbit/s) while sensors located far away will use a lower data rate down to 300 bit/s (minimum LoRa data rate in a 125 kHz channel).

The SX1301 digital baseband chip scans the 8 channels (IF0 to IF7) for preambles of all data rates at all times.

The chip is able to demodulate simultaneously up to 8 packets. Any combination of spreading factor and intermediate frequency for up to 8 packets is possible (e.g. one SF7 packet on IF0, one SF12 packet on IF7 and one SF9 packet on IF1 simultaneously).

The SX1301 can detect simultaneously preambles corresponding to all data rates on all IF0 to IF7 channels. However, it cannot demodulate more than 8 packets simultaneously. This is because the SX1301 architecture separates the preamble detection and signal acquisition task from the demodulation process. The number of simultaneously demodulated packets (in this case 8) is an arbitrary system parameter and may be set to other values for a customer specific circuit.

The unique multi data-rate multi-channel demodulation capacity SF7 to SF12 and of channels IF0 to IF7 allows innovative network architectures to be implemented.



3.3 External Module Connector

3.3.1 SPI

The connector on the bottom side provides an SPI connection, which allows direct access to the Sx1301 SPI interface. This gives the target system the possibility to use existing SPI interfaces to communicate.

After powering up RAK831, it is required to reset SX1301 via PIN 13. If the Hal driver from Github is used this functionality is already implemented.

3.3.2 GPS PPS

In case of available PPS signals in the target system, it is possible to connect this available signal to the appropriate pin at the connector.

3.3.3 Digital IOs

There are five GPIOs of the Sx1301 available, which gives the user some possibilities to get information about the system status. These pins are the same, as they are used for the LEDs on the RAK831.

As default setting the LEDs :

- 1) Backhaul packet
- 2) TX packet
- 3) RX Sensor packet
- 4) RX FSK packet
- 5) RX buffer not empty
- 6) Power

4.LoRa Systems, Network Approach

The use of LoRa technology can be distinguished in “Public” and “Private” networks. In both cases the usage of a concentrator module can be reasonable. Public networks are operator (e.g. telecom) managed networks whereas private networks are individually managed networks.

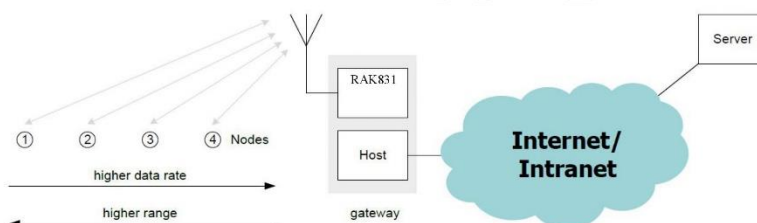
LoRa networks are typically star or multiple star networks where a gateway relays the packets between the end-nodes and a central network server. For private network approaches the server can also be implemented on the gateway host.

Due to the possible high range the connection between end-nodes and the concentrator RAK831 is always a direct link. There are no repeaters or routers within a LoRa network.

Depending on the used spreading factor and signal bandwidth different data rates¹ (0.3 kbps to ~22 kbps) and sensitivities down to -142.5 dBm are possible. Spreading factor and signal bandwidth are a trade-off between data rate and communication range.

4.1 Overview

The RAK831 is able to receive on different frequency channels at the same time and is able to demodulate the LoRa signal without knowledge of the used spreading factor of the sending node.



Due to the fact that the combination of spreading factors and signal bandwidths results in different data rates the use of “Dynamic Data-Rate Adaption” becomes possible. That means that LoRa nodes with high distances from the RAK831 must use higher spreading factors and therefore have a lower data rate. LoRa nodes which are closer to the concentrator can use lower spreading factors and therefore can increase their data rate.

Due to the fact that spreading factors are orthogonal and RAK831 supports up to 10 demodulations paths the channel capacity of a LoRa cell can be increased using RAK831 compared to conventional modulation techniques.

4.2 Firmware

The LoRa MAC specification is currently driven by the companies Semtech, IBM and Actility. Currently all available software, firmware and documentation can be found and downloaded from the open source project LoRa-net hosted on <https://github.com/Lora-net>

This project considers all parts that are needed to run a network based on LoRa technology. It includes the node firmware (several hardware platforms are supported), the gateway host software (HAL driver for SX1301, packet forwarder) and a server implementation.

It is highly recommended to fully re-use the latest HAL as provided by Semtech.

5. Electrical Characteristics & Timing specifications

In the following different electrical characteristics of the RAK831 are listed. Furthermore details and other parameter ranges are available on request.

Note: Stress exceeding of one or more of the limiting values listed under "Absolute Maximum Ratings" may cause permanent damage to the radio module.

5.1 Absolute Maximum Ratings

Parameter	Condition	Min	Typ.	Max	Unit
Supply Voltage(VDD)		-0.3	5.0	5.5	V
Operating Temperature		-40		+85	°C
RF Input Power				-15	dBm
Note:					

Note: With RF output power level above +15 dBm a minimum distance to a transmitter should be 1 m for avoiding too large input level.

5.2 Global Electrical Characteristics

Parameter	Condition	Min	Typ.	Max	Unit
Supply Voltage(VDD)		4.8	5.0	5.2	V
Current Consumption	RX Current		100		mA
	TX Current		80		
Note:					

T=25°C, VDD=5V(Typ.) if nothing else stated

Parameter	Condition	Min	Typ.	Max	Unit
Logic low input threshold(VIL)	"0"logic input			0.4	V
Logic high input threshold(VIH)	"1"logic input	2.9		3.3	V
Logic low output level(VOL)	"0"logic output,2mA sink			0.4	V
Logic high output level(VOH)	"1"logic output,2mA source	2.9		3.3	V
Note:					

5.3 SPI Interface Characteristics

T=25°C, VDD=5V(Typ.) if nothing else stated

Parameter	Condition	Min	Typ.	Max	Unit
SCK frequency				10	MHz
SCK high time		50			ns
SCK low time		50			ns
SCK rise time			5		ns
SCK fall time			5		ns
MOSI setup time	From MOSI change to SCK rising edge	10			ns
MOSI hold time	From SCK rising edge to MOSI change	20			ns
NSS setup time	From NSS falling edge to SCK rising edge	40			ns
NSS hold time	From SCK falling edge to NSS rising edge	40			ns
NSS high time between SPI accesses		40			ns
Note:					

5.4 RF Characteristics

5.4.1 Transmitter RF Characteristics

The RAK831 has an excellent transmitter performance. It is highly recommended, to use an optimized configuration for the power level configuration, which is part of the HAL. This results in a mean RF output power level and current consumption.

PA Control	DAC Control	MIX Control	DIG Gain	Nominal RF Power Level [dBm]
0	3	8	0	-5
0	3	9	0	-3
0	3	11	0	0
0	3	15	0	3
1	3	9	0	6
1	3	11	0	10
1	3	12	0	11
2	3	8	0	12
2	3	9	0	13
1	3	15	0	14
2	3	10	0	15
2	3	11	0	16
2	3	11	0	17
2	3	12	0	18
2	3	13	0	19
2	3	14	0	20

T=25°C, VDD=5V(Typ.) if nothing else stated

Parameter	Condition	Min	Typ.	Max	Unit
Frequency Range		863		870	MHz
Modulation Techniques	FSK/LoRa™				
TX Frequency Variation vs. Temperature	Power Level Setting:20	-3		+3	KHz
TX Power Variation vs. Temperature		-5		+5	dB
TX Power Variation		-1.5		+1.5	dB

Note: Also support 433,470,915 Frequency Range.

5.4.2 Receiver RF Characteristics

It is highly recommended, to use optimized RSSI calibration values, which is part of the HAL v3.1. For both, Radio 1 and 2, the RSSI-Offset should be set -169.0.

The following table gives typically sensitivity level of the RAK831 :

Signal Bandwidth/[KHz]	Spreading Factor	Sensitivity/[dBm]
125	12	-137
125	7	-126
250	12	-136
250	7	-123
500	12	-134
500	7	-120

5.5. RF Key Components

This section introduces the key components in RAK831 and help the developer to utilize the system to realize own system level design.

1) LDO

The system power supply is provided by the external 5V DC power supply. SX1301 and related clock crystal is powered by Dual output LDO transformer outputs 1.8V and 3.3V in order to meet the normal working condition of SX1301. Other key components are powered by LDO transformer output 3.3V. To be aware of the system design of LDO's power supply enable is provided by the output GPIO of SX1301 as default. The connection method of pin enable should be kept same as Semtech official code. At the same time, System design also need to keep flexibility and all LDO enable should be connect to pin DB24. For this case, user can run the official reference code in this board, and also can change all external enable clock as they need for achieve the flexibility debugging.

2) Power amplifier

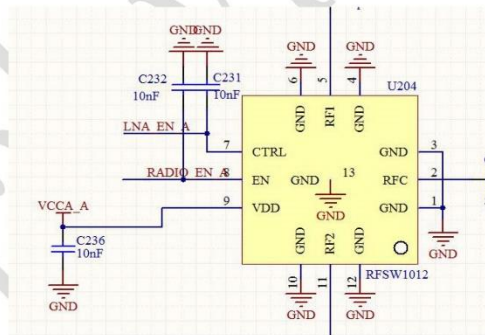
The Power amplifier chooses RFMD LF Power Amplifier and built in two steps gain. It realize the Max. 0.5w output power.The frequency range can cover from 380MHZ~960MHZ. The two steps gain control table as:

Parameter	Specification			Unit	Condition
	Min.	Typ.	Max.		
Overall					T=25 °C, V _{CC} =3.6V, V _{PD} =V _{BIAS} =3.0V, P _{IN} =0dBm, Freq=915MHz
CW Output Power		27.5		dBm	V _{CC} =3.6V
CW Output Power		30		dBm	V _{CC} =5V
Small Signal Gain		32		dB	P _{IN} =-10dBm
Second Harmonic		23		dBc	Without external second harmonic trap
Third Harmonic		45		dBc	
CW Efficiency	55	63		%	G16="high", G8="high", P _{IN} =0dBm
Power Down "ON"		3.0		V	Voltage supplied to the input
Power Down "OFF"	0	0.5	0.8	V	Voltage supplied to the input
VPD Input Current		6		mA	Only in "ON" state
G16, G8 "ON"	1.7		3.0	V	Voltage supplied to the input
G16, G8 "OFF"	0		0.7	V	Voltage supplied to the input
G16, G8 Input Current		1.0		mA	Only in "ON" state
Output Power	26.5	27.5	29	dBm	G16="high", G8="high", P _{IN} =0dBm
	21	23	25	dBm	G16="high", G8="low", P _{IN} =0dBm
	14	16	18	dBm	G16="low", G8="high", P _{IN} =0dBm
	3	5	8	dBm	G16="low", G8="low", P _{IN} =0dBm
Turn On/Off Time		200		ns	

3) RF switch

The RF switch choose RFSW1012 which has advantage of high Isolation and low insertion loss. This chip handling the switch between Tx and Rx. The Control logic as below image. Specially need highlight that the pin of CTRL was controlled by SX1301's GPIO through output signal of LNA_EN_A, the Pin of EN was controlled by SX1301's GPIO through output signal of RADIO_EN_A. Simultaneously, it also can be controlled by external input signal through DB24.

State	V _{DD}	CTRL	EN	RF Path
1	2.7V to 4.6V	V _{HIGH}	V _{HIGH}	ANT-RF2
2	2.7V to 4.6V	V _{LOW}	V _{HIGH}	ANT-RF1
Shutdown	2.7V to 4.6V	Don't Care	V _{LOW}	Shutdown



5.6. RF antenna interface

RAK831 provide three types of RF interface like SMA and other two IPEX connector. See the image as below for TDD_TXRX · TX_DEV · RX_DEV. Consider the developer may require supporting Tx/Rx simultaneously, therefore to make the compatible design. The Tx_DEV is the Tx channel, need change the C224 to NC and C216 with CAP(56pf/0402) or 0ohm resistance when using as standalone channel. RX_DEV is the Rx channel, need change C240 to NC and C244 with CAP(56pF/0402) or

APPENDIX C: RAK 811 datasheet

6. General Specification

6.1 General specification

Model Name	RAK811
Dimension	L x W x H: 22 x 14 x 1.7 mm
Interface	UART1, GPIOs
Operating temperature	-40°C to 85°C
Storage temperature	-40°C to 85°C

6.2 Recommended Operating Rating

	Min.	Typ.	Max.	Unit
Operating Temperature	-40	25	85	deg.C
VCC	3.15	3.3	3.45	V

6.3 Specification

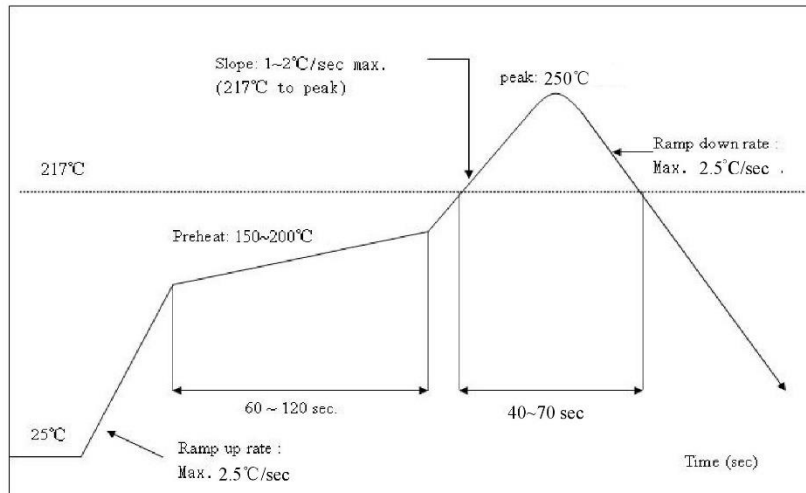
Feature	Description				
<i>General Specification</i>					
Frequency Band	RAK811-LF: EU433, CN470 RAK811-HF: EU868, US915, AU915, KR920, AS923, IN865				
Host Interface	UART				
Characteristics	Condition	Min	TYP	MAX	UNIT
Transmit	TX Power		14	20	dBm
RX Sensitivity	RSSI	-130	-		dBm
	SNR	-15			dB
Current Consumption	TX mode	30 (14dBm)			mA
	RX mode	5.5			mA
	Sleep mode	7.2			uA

8. Recommended Reflow Profile

Referred to IPC/JEDEC standard.

Peak Temperature : <math> < 250^{\circ} \text{C}</math>

Number of Times : ≤ 2 times



RAK

APPENDIX D: Manual to write LoRa gateway image to Micro SD



Who will need to write the Image

Case 1 : If you get a Micro SD card and get the following info,



that mean you must write the Image by yourself . The Micro SD have nothing in .

Case 2 : If must update the Image , you can try by yourself . The Micro SD must be formated firstly and then following the steps .

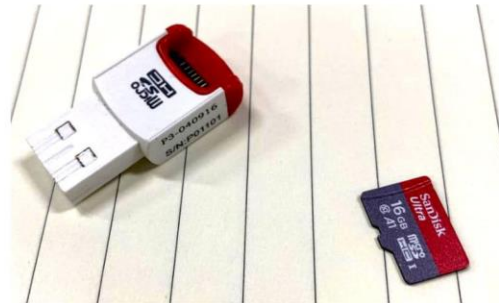
Note : This image must work with Raspberry Pi 3B(+).

How to write the Image

1. Prepare the Cards.

- 1) Card Reader for Micro SD
- 2) Micro SD for Raspberry.

Note : Please make sure the Micro SD can work with raspberry.

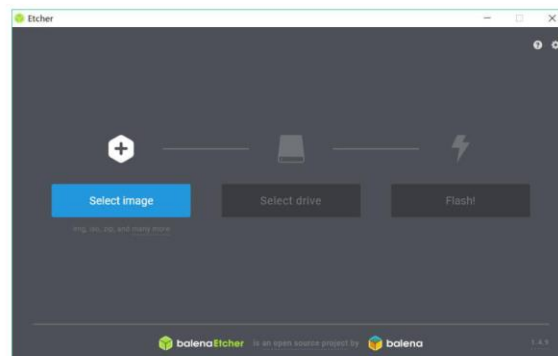


2. Download The Tools for Windows.

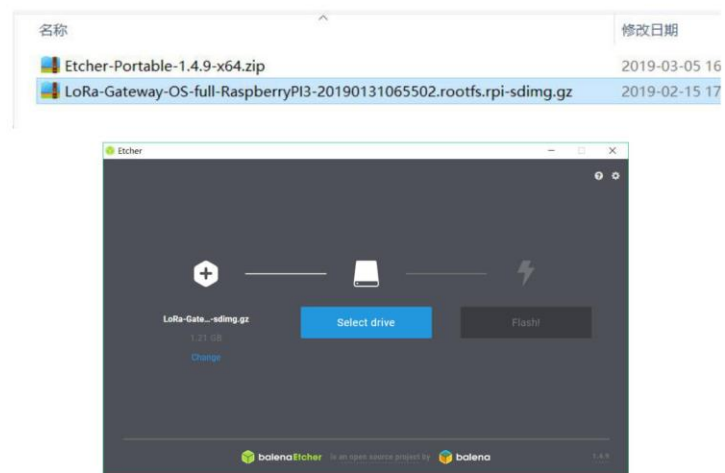
Software: [Etcher-Portable-1.4.9-x64](#)

3. Write the Image.

3.1 Open the Etcher .

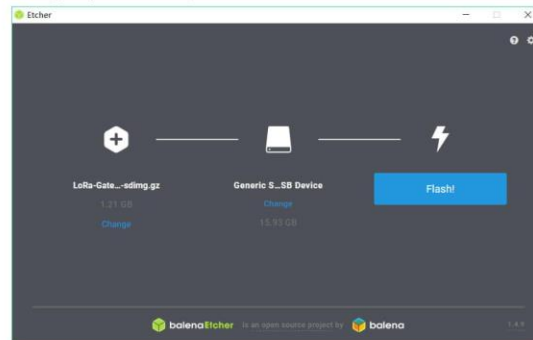


3.2 Select Image .

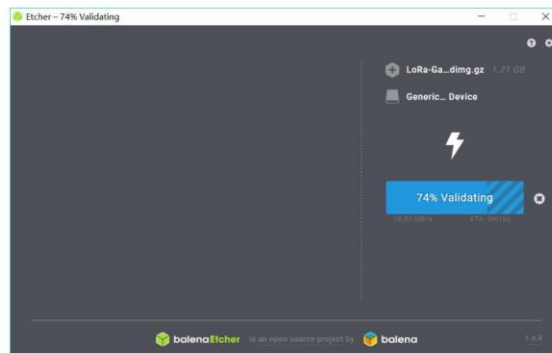
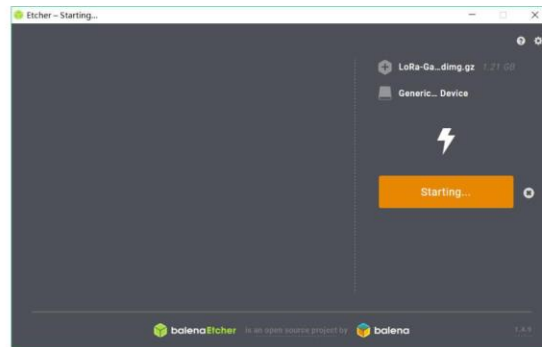


3.3 Plug in the USB Card Reader With Micro SD .

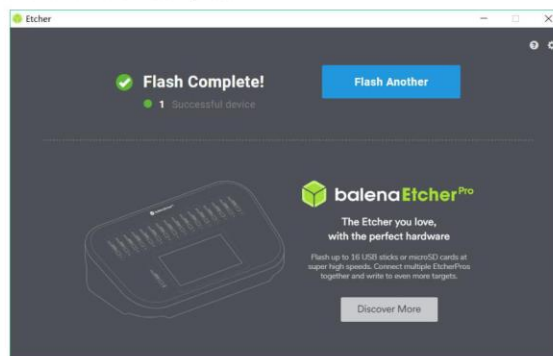
If you only have 1 storage device in your PC , Etcher will detect automatically . if you want to chagne ,please change and select the new one .



3.4 Flash Image.

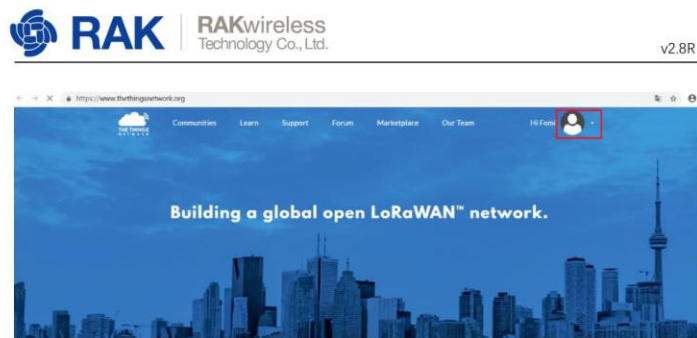


3.5 Wait for the Flash Complete.

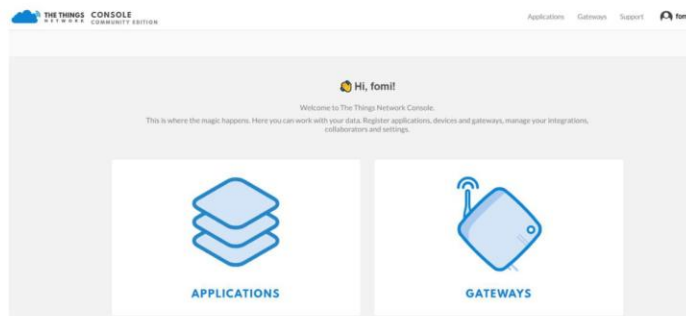


3.6 Plug in the Micro SD to your Raspberry Pi and Enjoy it.

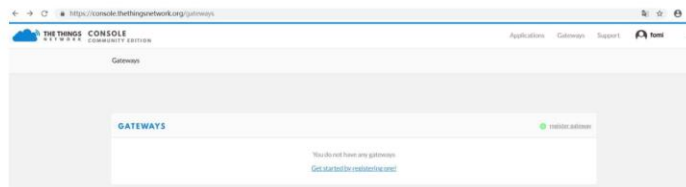
APPENDIX E: Registration of gateway in TTN



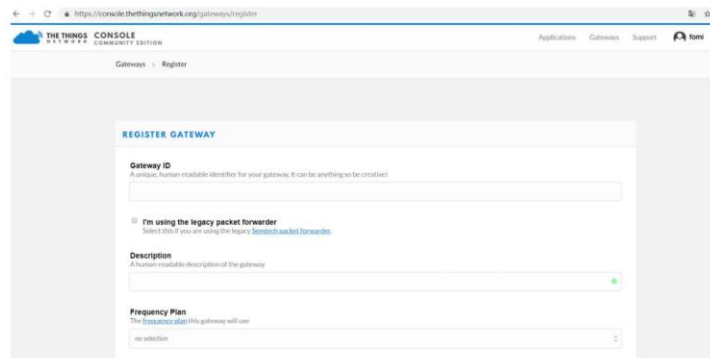
Can you see this page?




Click "GATEWAYS":



Click "register gateway"



THE THINGS CONSULE
COMMUNITY EDITION

Applications Gateways Support  **form**

Gateways > Register

REGISTER GATEWAY

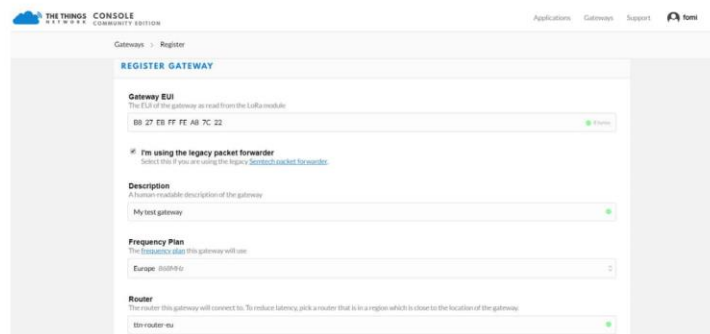
Gateway ID
A unique, human-readable identifier for your gateway. It can be anything so be creative!

I'm using the legacy packet forwarder
Select this if you are using the legacy [Semtech packet forwarder](#).


Description
A human-readable description of the gateway

Frequency Plan
The frequency plan this gateway will use

Fill in them one by one:



THE THINGS CONSULE
COMMUNITY EDITION

Applications Gateways Support  **form**

Gateways > Register

REGISTER GATEWAY

Gateway EUI
The EUI of the gateway as read from the LoRa module

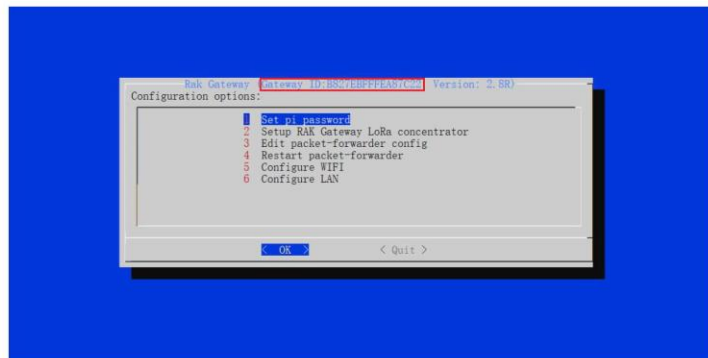
I'm using the legacy packet forwarder
Select this if you are using the legacy [Semtech packet forwarder](#).

Description
A human-readable description of the gateway

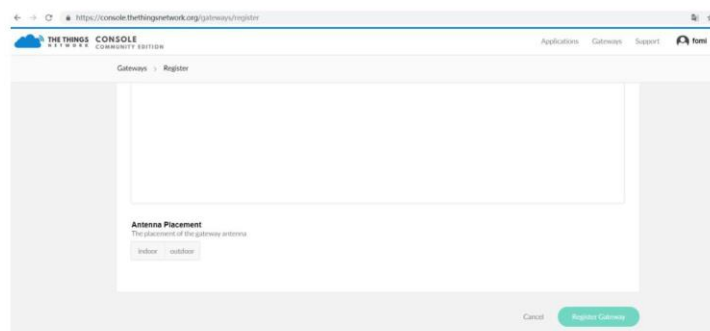
Frequency Plan
The frequency plan this gateway will use

Router
The router this gateway will connect to. To reduce latency, pick a router that is in a region which is close to the location of the gateway

Note: Please notice that the "Frequency Plan" is the frequency you want to use, and it must be same with LoRa gateway and LoRa node. The "Gateway EUI" is the one you have ever met in the following page, it must be same with the LoRa gateway's true "Gateway ID", otherwise, you will fail to register your LoRa gateway on TTN:



OK, click "Register Gateway" to register.



Do you see the page like as the following picture?

APPENDIX F: Dragino LoRa GPS HAT Single Channel LoRa & GPS modules Datasheet



www.dragino.com

1. Introduction

1.1 What is LoRa GPS HAT

LoRa GPS HAT is an expansion module for LoRaWAN and GPS for use with the **Raspberry Pi**. This product is intended for those interested in developing LoRaWAN solutions.

LoRa GPS HAT is **based on the SX1276/SX1278** transceiver. The add on **L80 GPS (Base on MTK MT3339)** is designed for applications that use a GPS connected via the serial ports to the Raspberry Pi such as timing applications or general applications that require GPS information.

The transceivers of the HAT feature the **LoRa™ long range modem** that provides ultra-long range spread spectrum communication and high interference immunity whilst minimizing current consumption. The LoRa/GPS HAT can achieve a sensitivity of over -148dBm using a low cost crystal and bill of materials. The high sensitivity combined with the integrated +20 dBm power amplifier yields industry leading link budget making it optimal for any application requiring range or robustness. LoRa™ also provides significant advantages in both blocking and selectivity over conventional modulation techniques, solving the traditional design compromise between range, interference immunity and energy consumption.

The L80 GPS module can calculate and predict orbits automatically using the ephemeris data (up to 3 days) stored in internal flash memory, so the HAT can fix position quickly even at indoor signal levels with low power consumption. With AlwaysLocate™ technology, the Lora/GPS HAT can adaptively adjust the on/off time to achieve balance between positioning accuracy and power consumption according to the environmental and motion conditions. The GPS also supports **automatic antenna switching function**. It can achieve the switching between **internal patch antenna and external active antenna**. Moreover, it keeps positioning during the switching process.

1.2 Specifications

LoRa Spec

- 168 dB maximum link budget.
- +20 dBm - 100 mW constant RF output vs.
- +14 dBm high efficiency PA.
- Programmable bit rate up to 300 kbps.
- High sensitivity: down to -148 dBm.
- Bullet-proof front end: IIP3 = -12.5 dBm.
- Excellent blocking immunity.
- Low RX current of 10.3 mA, 200 nA register retention.
- Fully integrated synthesizer with a resolution of 61 Hz.
- FSK, GFSK, MSK, GMSK, LoRaTM and OOK modulation.
- Built-in bit synchronizer for clock recovery.
- Preamble detection.
- 127 dB Dynamic Range RSSI.
- Automatic RF Sense and CAD with ultra-fast AFC.
- Packet engine up to 256 bytes with CRC.
- Built-in temperature sensor and low battery indicator.

GPS Spec

- Based on MT3339.
- Power Acquisition:25mA,Power Tracking:20mA.
- Compliant with GPS, SBAS.
- Programmable bit rate up to 300 kbps.
- Serial Interfaces UART: Adjustable 4800~115200 bps,Default: 9600bps.
- Update rate:1Hz (Default), up to10Hz.
- I/O Voltage:2.7V ~ 2.9V.
- Protocols:NMEA 0183,PMTK.
- Horizontal Position Accuracy:Autonomous <2.5 m CEP.
- TTFF@-130dBm with EASY™:Cold Start <15s,Warm Start <5s,Hot start <1s;TTFF@-130dBm.without EASY™:Cold Start <35s,Warm Start <30s,Hot Start <1s.
- Timing Accuracy:1PPS out 10ns, Reacquisition Time <1s.
- Velocity Accuracy Without aid <0.1m/s,Acceleration Accuracy Without aid 0.1m/s².
- Sensitivity Acquisition -148dBm, Tracking -165dBm, Reacquisition -160dBm.
- Environmental:Operating Temperature -40°C to 85°C,Storage Temperature -45°C to 125°C.
- Dynamic Performance Altitude Max.18000m, Maximum Velocity Max.515m/s, Maximum Acceleration 4G.
- L1 Band Receiver(1575.42MHz) Channel 22 (Tracking) /66 (Acquisition).

1.3 Features

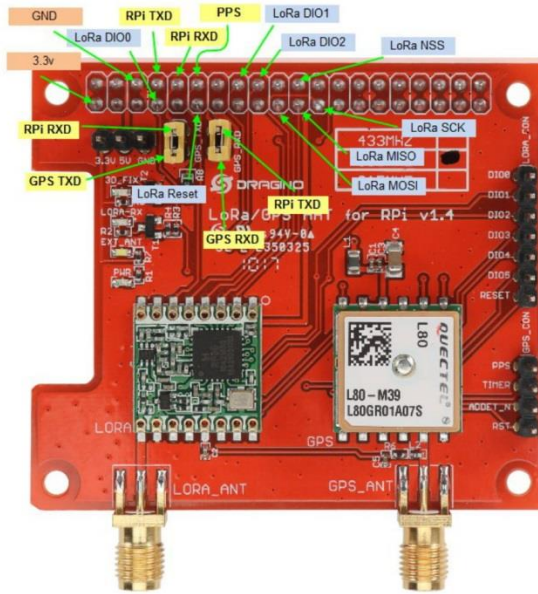
- ✓ Frequency Band: 868 MHz/433 MHz/915 MHz(Pre-configure in factory)
- ✓ Low power consumption
- ✓ Compatible with Raspberry Pi 2 Model B/Raspberry Pi 3 model B/B+
- ✓ LoRa™ Modem
- ✓ FSK, GFSK, MSK, GMSK, LoRa™and OOK modulation
- ✓ Preamble detection
- ✓ Baud rate configurable
- ✓ Built-in temperature sensor and low battery indicator
- ✓ Excellent blocking immunity
- ✓ Automatic RF Sense and CAD with ultra-fast AFC
- ✓ Support DGPS, SBAS(WAAS/EGNOS/MSAS/GAGAN)
- ✓ GPS automatic switching between internal patch antenna and external active antenna
- ✓ PPS VS. NMEA can be used in time service
- ✓ Support SDK command
- ✓ Built-in LNA for better sensitivity
- ✓ EASY™, advanced AGPS technology without external memory
- ✓ AlwaysLocate™, an intelligent controller of periodic mode
- ✓ GPS FLP mode, about 50% power consumption of normal mode
- ✓ GPS support short circuit protection and antenna detection

1.4 Applications

- ✓ Smart Buildings & Home Automation
- ✓ Logistics and Supply Chain Management
- ✓ Smart Metering
- ✓ Smart Agriculture
- ✓ Smart Cities
- ✓ Smart Factory

1.5 Pin Definition

Pin Illustration:



Pin Mapping

LoRa GPS HAT	RaspberryPi Wiring PI IO
3.3v	3.3v
5v	5v
GND	GND
DIO0	GPIO7
GPS_RX	GPIO15/TX
GPS_TX	GPIO16/RX
RESET	GPIO0
LoRa_NSS	GPIO6
LoRa_MISO	GPIO13/MISO
LoRa_MOSI	GPIO12/MOSI



www.dragino.com

SCK	GPIO14/SCLK
DIO1	GPIO4
DIO2	GPIO5
1PPS	GPIO1

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
		3.3v			1	2		5v		
2	8	SDA.1	ALTO	1	3	4		5v		
3	9	SCL.1	ALTO	1	5	6		0v		
4	7	GPIO. 7	IN	1	7	8	0	TXD	15	14
		0v			9	10	1	RXD	16	15
17	0	GPIO. 0	IN	0	11	12	0	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13	14		0v		
22	3	GPIO. 3	IN	0	15	16	0	GPIO. 4	4	23
		3.3v			17	18	1	GPIO. 5	5	24
10	12	MOSI	ALTO	0	19	20		0v		
9	13	MISO	ALTO	1	21	22	1	GPIO. 6	6	25
11	14	SCLK	ALTO	1	23	24	1	CE0	10	8
		0v			25	26	1	CE1	11	7
0	30	SDA.0	ALTO	1	27	28	1	SCL.0	31	1
5	21	GPIO.21	IN	1	29	30		0v		
6	22	GPIO.22	IN	1	31	32	0	GPIO.26	26	12
13	23	GPIO.23	IN	0	33	34		0v		
19	24	GPIO.24	IN	0	35	36	0	GPIO.27	27	16
26	25	GPIO.25	IN	0	37	38	0	GPIO.28	28	20
		0v			39	40	0	GPIO.29	29	21

1.6 Hardware Change log

- LoRa/GPS_HAT v1.0: The first hardware release for the LoRa/GPS_HAT.
- LoRa/GPS_HAT v1.3:
 - ✓ Add a trace from LoRa DIO1 to RPi GPIO4(wiringPi definition).
 - ✓ Add a trace from Lora DIO2 to RPi GPIO5(wiringPi definition). They are required by LMIC library in RPi.
- LoRa/GPS HAT v1.4:
 - ✓ Change SMA connector to support active antenna
 - ✓ Add AADET_N LED to show if external antenna is active.
 - ✓ Connect GPS PPS pin to RPi BCM pin 18
 - ✓ Modify Silkscreen for GPS TXD/RXD

1.7 LEDs

- ✓ **PWR:** Power Indicate LED. Turns on once there is power.
- ✓ **LoRa-RX:** Indicate there is a wireless packet received in the LoRa module.
- ✓ **3D_FIX:** The led blink every 100ms after the GPS fixing position.
- ✓ **EXT_ANT:** Indicate there is an external GPS antenna connected.

1.8 Dimension & Weight

- ✓ **Size:** 60mm*53mm*25mm
- ✓ **Net weight:** 30g.
- ✓ **Package Size:** 98mm x 81mm x 32mm

APPENDIX G: SX1276 datasheet



SX1276/77/78/79

WIRELESS, SENSING & TIMING

DATASHEET

4. SX1276/77/78/79 Digital Electronics

4.1. The LoRa™ Modem

The LoRa™ modem uses spread spectrum modulation and forward error correction techniques to increase the range and robustness of radio communication links compared to traditional FSK or OOK based modulation. Examples of the performance improvement possible, for several possible settings, are summarised in the table below. Here the spreading factor and error correction rate are design variables that allow the designer to optimise the trade-off between occupied bandwidth, data rate, link budget improvement and immunity to interference.

Table 12 Example LoRa™ Modem Performances, 868MHz Band

Bandwidth (kHz)	Spreading Factor	Coding rate	Nominal Rb (bps)	Sensitivity indication (dBm)	Frequency Reference
10.4	6	4/5	782	-131	TCXO
	12	4/5	24	-147	
20.8	6	4/5	1562	-128	
	12	4/5	49	-144	
62.5	6	4/5	4688	-121	XTAL
	12	4/5	146	-139	
125	6	4/5	9380	-118	
	12	4/5	293	-136	

Notes - for all bandwidths lower than 62.5 kHz, it is advised to use a TCXO as a frequency reference. This is required to meet the frequency error tolerance specifications given in the Electrical Specification

- Higher spreading factors and longer transmission times impose more stringent constraints on the short term frequency stability of the reference. Please get in touch with a Semtech representative to implement extremely low sensitivity products.

For European operation the range of crystal tolerances acceptable for each sub-band (of the ERC 70-03) is given in the specifications table. For US based operation a frequency hopping mode is available that automates both the LoRa™ spread spectrum and frequency hopping spread spectrum processes.

Another important facet of the LoRa™ modem is its increased immunity to interference. The LoRa™ modem is capable of co-channel GMSK rejection of up to 20 dB. This immunity to interference permits the simple coexistence of LoRa™ modulated systems either in bands of heavy spectral usage or in hybrid communication networks that use LoRa™ to extend range when legacy modulation schemes fail.

4.1.1. Link Design Using the LoRa™ Modem

4.1.1.1. Overview

The LoRa™ modem is setup as shown in the following figure. This configuration permits the simple replacement of the FSK modem with the LoRa™ modem via the configuration register setting *RegOpMode*. This change can be performed on the fly (in Sleep operating mode) thus permitting the use of both standard FSK or OOK in conjunction with the long range capability. The LoRa™ modulation and demodulation process is proprietary, it uses a form of spread spectrum modulation combined with cyclic error correction coding. The combined influence of these two factors is an increase in link budget and enhanced immunity to interference.

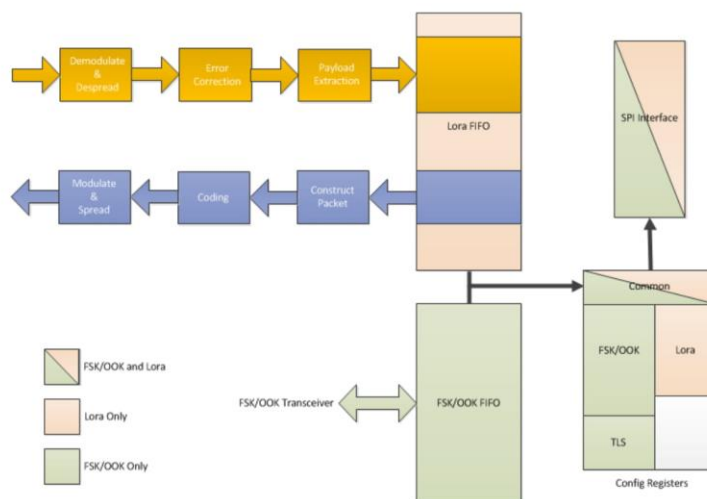


Figure 5. LoRa™ Modem Connectivity

A simplified outline of the transmit and receive processes is also shown above. Here we see that the LoRa™ modem has an independent dual port data buffer FIFO that is accessed through an SPI interface common to all modes. Upon selection of LoRa™ mode, the configuration register mapping of the SX1276/77/78/79 changes. For full details of this change please consult the register description of Section 6.

So that it is possible to optimise the LoRa™ modulation for a given application, access is given to the designer to three critical design parameters. Each one permitting a trade off between link budget, immunity to interference, spectral occupancy and nominal data rate. These parameters are spreading factor, modulation bandwidth and error coding rate.

4.1.1.2. Spreading Factor

The spread spectrum LoRa™ modulation is performed by representing each bit of payload information by multiple chips of information. The rate at which the spread information is sent is referred to as the symbol rate (Rs), the ratio between the nominal symbol rate and chip rate is the spreading factor and represents the number of symbols sent per bit of information. The range of values accessible with the LoRa™ modem are shown in the following table.

Table 13 Range of Spreading Factors

SpreadingFactor (RegModulationCfg)	Spreading Factor (Chips / symbol)	LoRa Demodulator SNR
6	64	-5 dB
7	128	-7.5 dB
8	256	-10 dB
9	512	-12.5 dB
10	1024	-15 dB
11	2048	-17.5 dB
12	4096	-20 dB

Note that the spreading factor, *SpreadingFactor*, must be known in advance on both transmit and receive sides of the link as different spreading factors are orthogonal to each other. Note also the resulting signal to noise ratio (SNR) required at the receiver input. It is the capability to receive signals with negative SNR that increases the sensitivity, so link budget and range, of the LoRa receiver.

Spreading Factor 6

SF = 6 is a special use case for the highest data rate transmission possible with the LoRa modem. To this end several settings must be activated in the SX1276/77/78/79 registers when it is in use. These settings are only valid for SF6 and should be set back to their default values for other spreading factors:

- ◆ Set *SpreadingFactor* = 6 in *RegModemConfig2*
- ◆ The header must be set to Implicit mode.
- ◆ Set the bit field *DetectionOptimize* of register *RegLoRaDetectOptimize* to value "0b101".
- ◆ Write 0x0C in the register *RegDetectionThreshold*.

4.1.1.3. Coding Rate

To further improve the robustness of the link the LoRa™ modem employs cyclic error coding to perform forward error detection and correction. Such error coding incurs a transmission overhead - the resultant additional data overhead per transmission is shown in the table below.

Table 14 Cyclic Coding Overhead

CodingRate (RegTxCfg1)	Cyclic Coding Rate	Overhead Ratio
1	4/5	1.25
2	4/6	1.5
3	4/7	1.75
4	4/8	2

Forward error correction is particularly efficient in improving the reliability of the link in the presence of interference. So that the coding rate (and so robustness to interference) can be changed in response to channel conditions - the coding rate can optionally be included in the packet header for use by the receiver. Please consult Section 4.1.1.6 for more information on the LoRa™ packet and header.

4.1.1.4. Signal Bandwidth

An increase in signal bandwidth permits the use of a higher effective data rate, thus reducing transmission time at the expense of reduced sensitivity improvement. There are of course regulatory constraints in most countries on the permissible occupied bandwidth. Contrary to the FSK modem which is described in terms of the single sideband bandwidth, the LoRa™ modem bandwidth refers to the double sideband bandwidth (or total channel bandwidth). The range of bandwidths relevant to most regulatory situations is given in the LoRa™ modem specifications table (see Section 2.5.5).

Table 15 LoRa Bandwidth Options

Bandwidth (kHz)	Spreading Factor	Coding rate	Nominal Rb (bps)
7.8	12	4/5	18
10.4	12	4/5	24
15.6	12	4/5	37
20.8	12	4/5	49
31.2	12	4/5	73
41.7	12	4/5	98
62.5	12	4/5	146
125	12	4/5	293
250	12	4/5	586
500	12	4/5	1172

Note In the lower band (169 MHz), the 250 kHz and 500 kHz bandwidths are not supported.

4.1.1.5. LoRa™ Transmission Parameter Relationship

With a knowledge of the key parameters that can be controlled by the user we define the LoRa™ symbol rate as:

$$R_s = \frac{BW}{2^{SF}}$$

where BW is the programmed bandwidth and SF is the spreading factor. The transmitted signal is a constant envelope signal. Equivalently, one chip is sent per second per Hz of bandwidth.

4.1.1.6. LoRa™ Packet Structure

The LoRa™ modem employs two types of packet format, explicit and implicit. The explicit packet includes a short header that contains information about the number of bytes, coding rate and whether a CRC is used in the packet. The packet format is shown in the following figure.

The LoRa™ packet comprises three elements:

- ◆ A preamble.
- ◆ An optional header.
- ◆ The data payload.

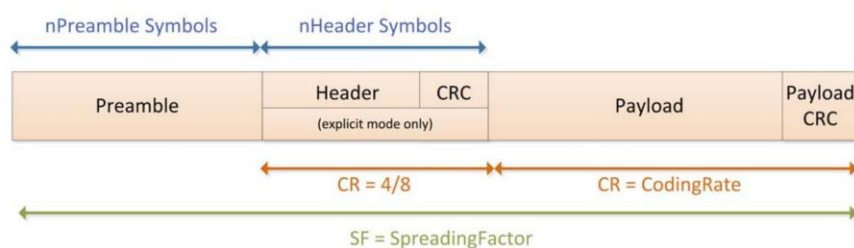


Figure 6. LoRa™ Packet Structure

Preamble

The preamble is used to synchronize receiver with the incoming data flow. By default the packet is configured with a 12 symbol long sequence. This is a programmable variable so the preamble length may be extended, for example in the interest of reducing to receiver duty cycle in receive intensive applications. However, the minimum length suffices for all communication. The transmitted preamble length may be changed by setting the register *PreambleLength* from 6 to 65535, yielding total preamble lengths of 6+4 to 65535+4 symbols, once the fixed overhead of the preamble data is considered. This permits the transmission of a near arbitrarily long preamble sequence.

The receiver undertakes a preamble detection process that periodically restarts. For this reason the preamble length should be configured identical to the transmitter preamble length. Where the preamble length is not known, or can vary, the maximum preamble length should be programmed on the receiver side.

Header

Depending upon the chosen mode of operation two types of header are available. The header type is selected by the *ImplicitHeaderModeOn* bit found within the *RegModemConfig1* register.

Explicit Header Mode

This is the default mode of operation. Here the header provides information on the payload, namely:

- ◆ The payload length in bytes.
- ◆ The forward error correction code rate
- ◆ The presence of an optional 16-bits CRC for the payload.



The header is transmitted with maximum error correction code (4/8). It also has its own CRC to allow the receiver to discard invalid headers.

Implicit Header Mode

In certain scenarios, where the payload, coding rate and CRC presence are fixed or known in advance, it may be advantageous to reduce transmission time by invoking implicit header mode. In this mode the header is removed from the packet. In this case the payload length, error coding rate and presence of the payload CRC must be manually configured on both sides of the radio link.

Note With $SF = 6$ selected, implicit header mode is the only mode of operation possible.

Explicit Header Mode:

In Explicit Header Mode, the presence of the CRC at the end of the payload is selected only on the transmitter side through the bit *RxPayloadCrcOn* in the register *RegModemConfig1*.

On the receiver side, the bit *RxPayloadCrcOn* in the register *RegModemConfig1* is not used and once the payload has been received, the user should check the bit *CrcOnPayload* in the register *RegHopChannel*. If the bit *CrcOnPayload* is at '1', the user should then check the Irq Flag *PayloadCrcError* to make sure the CRC is valid.

If the bit *CrcOnPayload* is at '0', it means there was no CRC on the payload and thus the IRQ Flag *PayloadCrcError* will not be triggered even if the payload has errors.

Explicit Header	Transmitter	Receiver	CRC Status
Value of the bit <i>RxPayloadCrcOn</i>	0	0	CRC is not checked
	0	1	CRC is not checked
	1	0	CRC is checked
	1	1	CRC is checked

Implicit Header Mode;

In Implicit Header Mode, it is necessary to set the bit *RxPayloadCrcOn* in the register *RegModemConfig1* on both sides (TX and RX)

Implicit Header	Transmitter	Receiver	CRC Status
Value of the bit <i>RxPayloadCrcOn</i>	0	0	CRC is not checked
	0	1	CRC is always wrong
	1	0	CRC is not checked
	1	1	CRC is checked



Low Data Rate Optimization

Given the potentially long duration of the packet at high spreading factors the option is given to improve the robustness of the transmission to variations in frequency over the duration of the packet transmission and reception. The bit *LowDataRateOptimize* increases the robustness of the LoRa link at these low effective data rates. Its use is mandated when the symbol duration exceeds 16ms. Note that both the transmitter and the receiver must have the same setting for *LowDataRateOptimize*.

Payload

The packet payload is a variable-length field that contains the actual data coded at the error rate either as specified in the header in explicit mode or in the register settings in implicit mode. An optional CRC may be appended. For more information on the payload and how it is loaded from the data buffer FIFO please see Section 4.1.2.3.

4.1.1.7. Time on air

For a given combination of spreading factor (SF), coding rate (CR) and signal bandwidth (BW) the total on-the-air transmission time of a LoRa™ packet can be calculated as follows. From the definition of the symbol rate it is convenient to define the symbol rate:

$$T_s = \frac{1}{R_s}$$

The LoRa packet duration is the sum of the duration of the preamble and the transmitted packet. The preamble length is calculated as follows:

$$T_{preamble} = (n_{preamble} + 4.25)T_{sym}$$

where $n_{preamble}$ is the programmed preamble length, taken from the registers *RegPreambleMsb* and *RegPreambleLsb*. The payload duration depends upon the header mode that is enabled. The following formula gives the number of payload symbols.

$$n_{payload} = 8 + \max\left(\text{ceil}\left[\frac{(8PL - 4SF + 28 + 16CRC - 20IH)}{4(SF - 2DE)}\right], (CR + 4), 0\right)$$

With the following dependencies:

- ◆ PL is the number of Payload bytes (1 to 255)
- ◆ SF is the spreading factor (6 to 12)
- ◆ IH=0 when the header is enabled, IH=1 when no header is present
- ◆ DE=1 when *LowDataRateOptimize*=1, DE=0 otherwise
- ◆ CR is the coding rate (1 corresponding to 4/5, 4 to 4/8)

The Payload duration is then the symbol period multiplied by the number of Payload symbols

$$T_{payload} = n_{payload} \times T_s$$

The time on air, or packet duration, is simply then the sum of the preamble and payload duration.

$$T_{packet} = T_{preamble} + T_{payload}$$



6.4. LoRa™ Mode Register Map

This details the SX1276/77/78/79 register mapping and the precise contents of each register in LoRa™ mode.

It is essential to understand that the LoRa™ modem is controlled independently of the FSK modem. Therefore, care should be taken when accessing the registers, especially as some register may have the same name in LoRa™ or FSK mode.

The LoRa registers are only accessible when the device is set in Lora mode (and, in the same way, the FSK register are only accessible in FSK mode). However, in some cases, it may be necessary to access some of the FSK register while in LoRa mode. To this aim, the *AccessSharedReg* bit was created in the *RegOpMode* register. This bit, when set to '1', will grant access to the FSK register 0x0D up to the register 0x3F. Once the setup has been done, it is strongly recommended to clear this bit so that LoRa register can be accessed normally.

Convention: r: read, w: write, c: set to clear and t: trigger.

Name (Address)	Bits	Variable Name	Mode	Reset	LoRa™ Description
RegFifo (0x00)	7-0	Fifo	rw	0x00	LoRa™ base-band FIFO data input/output. FIFO is cleared and not accessible when device is in SLEEP mode
Common Register Settings					
RegOpMode (0x01)	7	LongRangeMode	rw	0x0	0 → FSK/OOK Mode 1 → LoRa™ Mode This bit can be modified only in Sleep mode. A write operation on other device modes is ignored.
	6	AccessSharedReg	rw	0x0	This bit operates when device is in Lora mode; if set it allows access to FSK registers page located in address space (0x0D:0x3F) while in LoRa mode 0 → Access LoRa registers page 0x0D: 0x3F 1 → Access FSK registers page (in mode LoRa) 0x0D: 0x3F
	5-4	reserved	r	0x00	reserved
	3	LowFrequencyModeOn	rw	0x01	Access Low Frequency Mode registers 0 → High Frequency Mode (access to HF test registers) 1 → Low Frequency Mode (access to LF test registers)
	2-0	Mode	rwt	0x01	Device modes 000 → SLEEP 001 → STDBY 010 → Frequency synthesis TX (FSTX) 011 → Transmit (TX) 100 → Frequency synthesis RX (FSRX) 101 → Receive continuous (RXCONTINUOUS) 110 → receive single (RXSINGLE) 111 → Channel activity detection (CAD)
(0x02)	7-0	reserved	r	0x00	-
(0x03)	7-0	reserved	r	0x00	-
(0x04)	7-0	reserved	rw	0x00	-
(0x05)	7-0	reserved	r	0x00	-
RegFRmsb (0x06)	7-0	Fr(23:16)	rw	0x6c	MSB of RF carrier frequency



SX1276/77/78/79

WIRELESS, SENSING & TIMING

DATASHEET

Name (Address)	Bits	Variable Name	Mode	Reset	LoRa™ Description
RegFrMid (0x07)	7-0	Fr(15:8)	rw	0x80	MSB of RF carrier frequency
RegFrLsb (0x08)	7-0	Fr(7:0)	rwt	0x00	LSB of RF carrier frequency $f_{RF} = \frac{F(XOSC) \cdot Frf}{2^{19}}$ Resolution is 61.035 Hz if F(XOSC) = 32 MHz. Default value is 0x6c8000 = 434 MHz. Register values must be modified only when device is in SLEEP or STAND-BY mode.
Registers for RF blocks					
RegPaConfig (0x09)	7	PaSelect	rw	0x00	Selects PA output pin 0 → RFO pin. Output power is limited to +14 dBm. 1 → PA_BOOST pin. Output power is limited to +20 dBm
	6-4	MaxPower	rw	0x04	Select max output power: Pmax=10.8+0.6*MaxPower [dBm]
	3-0	OutputPower	rw	0x0f	Pout=Pmax-(15-OutputPower) if PaSelect = 0 (RFO pin) Pout=17-(15-OutputPower) if PaSelect = 1 (PA_BOOST pin)
RegPaRamp (0x0A)	7-5	unused	r	-	unused
	4	reserved	rw	0x00	reserved
	3-0	PaRamp(3:0)	rw	0x09	Rise/Fall time of ramp up/down in FSK 0000 → 3.4 ms 0001 → 2 ms 0010 → 1 ms 0011 → 500 us 0100 → 250 us 0101 → 125 us 0110 → 100 us 0111 → 62 us 1000 → 50 us 1001 → 40 us 1010 → 31 us 1011 → 25 us 1100 → 20 us 1101 → 15 us 1110 → 12 us 1111 → 10 us
RegOcp (0x0B)	7-6	unused	r	0x00	unused
	5	OcpOn	rw	0x01	Enables overload current protection (OCP) for PA: 0 → OCP disabled 1 → OCP enabled
	4-0	OcpTrim	rw	0x0b	Trimming of OCP current: Imax = 45+5*OcpTrim [mA] if OcpTrim <= 15 (120 mA) / Imax = -30+10*OcpTrim [mA] if 15 < OcpTrim <= 27 (130 to 240 mA) Imax = 240mA for higher settings Default Imax = 100mA



Name (Address)	Bits	Variable Name	Mode	Reset	LoRa™ Description
RegLna (0x0C)	7-5	LnaGain	rw	0x01	LNA gain setting: 000 → not used 001 → G1 = maximum gain 010 → G2 011 → G3 100 → G4 101 → G5 110 → G6 = minimum gain 111 → not used
	4-3	LnaBoostLf	rw	0x00	Low Frequency (RFI_LF) LNA current adjustment 00 → Default LNA current Other → Reserved
	2	reserved	rw	0x00	reserved
	1-0	LnaBoostHf	rw	0x00	High Frequency (RFI_HF) LNA current adjustment 00 → Default LNA current 11 → Boost on, 150% LNA current
Lora page registers					
RegFifoAddrPtr (0x0D)	7-0	FifoAddrPtr	rw	0x00	SPI interface address pointer in FIFO data buffer.
RegFifoTxBaseAddr (0x0E)	7-0	FifoTxBaseAddr	rw	0x80	write base address in FIFO data buffer for TX modulator
RegFifoRxBaseAddr (0x0F)	7-0	FifoRxBaseAddr	rw	0x00	read base address in FIFO data buffer for RX demodulator
RegFifoRxCurrentAddr (0x10)	7-0	FifoRxCurrentAddr	r	n/a	Start address (in data buffer) of last packet received
RegIrqFlags Mask (0x11)	7	RxTimeoutMask	rw	0x00	Timeout interrupt mask: setting this bit masks the corresponding IRQ in RegIrqFlags
	6	RxDoneMask	rw	0x00	Packet reception complete interrupt mask: setting this bit masks the corresponding IRQ in RegIrqFlags
	5	PayloadCrcErrorMask	rw	0x00	Payload CRC error interrupt mask: setting this bit masks the corresponding IRQ in RegIrqFlags
	4	ValidHeaderMask	rw	0x00	Valid header received in Rx mask: setting this bit masks the corresponding IRQ in RegIrqFlags
	3	TxDoneMask	rw	0x00	FIFO Payload transmission complete interrupt mask: setting this bit masks the corresponding IRQ in RegIrqFlags
	2	CadDoneMask	rw	0x00	CAD complete interrupt mask: setting this bit masks the corresponding IRQ in RegIrqFlags
	1	FhssChangeChannelMask	rw	0x00	FHSS change channel interrupt mask: setting this bit masks the corresponding IRQ in RegIrqFlags
	0	CadDetectedMask	rw	0x00	Cad Detected Interrupt Mask: setting this bit masks the corresponding IRQ in RegIrqFlags



Name (Address)	Bits	Variable Name	Mode	Reset	LoRa™ Description
RegIrqFlags (0x12)	7	RxTimeout	rc	0x00	Timeout interrupt: writing a 1 clears the IRQ
	6	RxDone	rc	0x00	Packet reception complete interrupt: writing a 1 clears the IRQ
	5	PayloadCrcError	rc	0x00	Payload CRC error interrupt: writing a 1 clears the IRQ
	4	ValidHeader	rc	0x00	Valid header received in Rx: writing a 1 clears the IRQ
	3	TxDone	rc	0x00	FIFO Payload transmission complete interrupt: writing a 1 clears the IRQ
	2	CadDone	rc	0x00	CAD complete: write to clear: writing a 1 clears the IRQ
	1	FhssChangeChannel	rc	0x00	FHSS change channel interrupt: writing a 1 clears the IRQ
	0	CadDetected	rc	0x00	Valid Lora signal detected during CAD operation: writing a 1 clears the IRQ
	RegRxBytes (0x13)	7-0	FifoRxBytesNb	r	n/a
RegRxHeaderCntValueMsb (0x14)	7-0	ValidHeaderCntMsb(15:8)	r	n/a	Number of valid headers received since last transition into Rx mode, MSB(15:8). Header and packet counters are reseted in Sleep mode.
RegRxHeaderCntValueLsb (0x15)	7-0	ValidHeaderCntLsb(7:0)	r	n/a	Number of valid headers received since last transition into Rx mode, LSB(7:0). Header and packet counters are reseted in Sleep mode.
RegRxPacketCntValueMsb (0x16)	7-0	ValidPacketCntMsb(15:8)	rc	n/a	Number of valid packets received since last transition into Rx mode, MSB(15:8). Header and packet counters are reseted in Sleep mode.
RegRxPacketCntValueLsb (0x17)	7-0	ValidPacketCntLsb(7:0)	r	n/a	Number of valid packets received since last transition into Rx mode, LSB(7:0). Header and packet counters are reseted in Sleep mode.
RegModemStatus (0x18)	7-5	RxCodingRate	r	n/a	Coding rate of last header received
	4	ModemStatus	r	'1'	Modem clear
	3		r	'0'	Header info valid
	2		r	'0'	RX on-going
	1		r	'0'	Signal synchronized
	0		r	'0'	Signal detected



SX1276/77/78/79

WIRELESS, SENSING & TIMING

DATASHEET

Name (Address)	Bits	Variable Name	Mode	Reset	LoRa™ Description
RegPktSnrValue (0x19)	7-0	PacketSnr	r	n/a	Estimation of SNR on last packet received. In two's complement format multiplied by 4. $SNR[dB] = \frac{PacketSnr(two's\ complement)}{4}$
RegPktRssiValue (0x1A)	7-0	PacketRssi	r	n/a	RSSI of the latest packet received (dBm): RSSI[dBm] = -157 + Rssi (using HF output port, SNR >= 0) or RSSI[dBm] = -164 + Rssi (using LF output port, SNR >= 0) (see section 5.5.5 for details)
RegRssiValue (0x1B)	7-0	Rssi	r	n/a	Current RSSI value (dBm) RSSI[dBm] = -157 + Rssi (using HF output port) or RSSI[dBm] = -164 + Rssi (using LF output port) (see section 5.5.5 for details)
RegHopChannel (0x1C)	7	PllTimeout	r	n/a	PLL failed to lock while attempting a TX/RX/CAD operation 1 → PLL did not lock 0 → PLL did lock
	6	CrcOnPayload	r	n/a	CRC Information extracted from the received packet header (Explicit header mode only) 0 → Header indicates CRC off 1 → Header indicates CRC on
	5-0	FhssPresentChannel	r	n/a	Current value of frequency hopping channel in use.



SX1276/77/78/79

WIRELESS, SENSING & TIMING

DATASHEET

Name (Address)	Bits	Variable Name	Mode	Reset	LoRa™ Description
RegModemC onfig1 (0x1D)	7-4	Bw	rw	0x07	Signal bandwidth: 0000 → 7.8 kHz 0001 → 10.4 kHz 0010 → 15.6 kHz 0011 → 20.8kHz 0100 → 31.25 kHz 0101 → 41.7 kHz 0110 → 62.5 kHz 0111 → 125 kHz 1000 → 250 kHz 1001 → 500 kHz other values → reserved In the lower band (169MHz), signal bandwidths 8&9 are not supported)
	3-1	CodingRate	rw	'001'	Error coding rate 001 → 4/5 010 → 4/6 011 → 4/7 100 → 4/8 All other values → reserved In implicit header mode should be set on receiver to determine expected coding rate. See 4.1.1.3
	0	ImplicitHeaderModeOn	rw	0x0	0 → Explicit Header mode 1 → Implicit Header mode
RegModemC onfig2 (0x1E)	7-4	SpreadingFactor	rw	0x07	SF rate (expressed as a base-2 logarithm) 6 → 64 chips / symbol 7 → 128 chips / symbol 8 → 256 chips / symbol 9 → 512 chips / symbol 10 → 1024 chips / symbol 11 → 2048 chips / symbol 12 → 4096 chips / symbol other values reserved.
	3	TxContinuousMode	rw	0	0 → normal mode, a single packet is sent 1 → continuous mode, send multiple packets across the FIFO (used for spectral analysis)
	2	RxPayloadCrcOn	rw	0x00	Enable CRC generation and check on payload: 0 → CRC disable 1 → CRC enable If CRC is needed, RxPayloadCrcOn should be set: - in Implicit header mode: on Tx and Rx side - in Explicit header mode: on the Tx side alone (recovered from the header in Rx side)
	1-0	SymbTimeout(9:8)	rw	0x00	RX Time-Out MSB
RegSymbTim eoutLsb (0x1F)	7-0	SymbTimeout(7:0)	rw	0x64	RX Time-Out LSB RX operation time-out value expressed as number of symbols: $TimeOut = SymbTimeout \cdot Ts$



SX1276/77/78/79

WIRELESS, SENSING & TIMING

DATASHEET

Name (Address)	Bits	Variable Name	Mode	Reset	LoRa™ Description
RegPreambleMsb (0x20)	7-0	PreambleLength(15:8)	rw	0x0	Preamble length MSB, = PreambleLength + 4.25 Symbols See 4.1.1 for more details.
RegPreambleLsb (0x21)	7-0	PreambleLength(7:0)	rw	0x8	Preamble Length LSB
RegPayloadLength (0x22)	7-0	PayloadLength(7:0)	rw	0x1	Payload length in bytes. The register needs to be set in implicit header mode for the expected packet length. A 0 value is not permitted
RegMaxPayloadLength (0x23)	7-0	PayloadMaxLength(7:0)	rw	0xff	Maximum payload length; if header payload length exceeds value a header CRC error is generated. Allows filtering of packet with a bad size.
RegHopPeriod (0x24)	7-0	FreqHoppingPeriod(7:0)	rw	0x0	Symbol periods between frequency hops. (0 = disabled). 1st hop always happen after the 1st header symbol
RegFifoRxByteAddr (0x25)	7-0	FifoRxByteAddrPtr	r	n/a	Current value of RX databuffer pointer (address of last byte written by Lora receiver)
RegModemConfig3 (0x26)	7-4	Unused	r	0x00	
	3	LowDataRateOptimize	rw	0x00	0 → Disabled 1 → Enabled; mandated for when the symbol length exceeds 16ms
	2	AgcAutoOn	rw	0x00	0 → LNA gain set by register LnaGain 1 → LNA gain set by the internal AGC loop
	1-0	Reserved	rw	0x00	Reserved
(0x27)	7-0	PpmCorrection	rw	0x00	Data rate offset value, used in conjunction with AFC
	7-4	Reserved	r	n/a	Reserved
RegFeilMsb (0x28)	3-0	FreqError(19:16)	r	0x0	Estimated frequency error from modem MSB of RF Frequency Error $F_{Error} = \frac{FreqError \times 2^{24}}{F_{xstal}} \times \frac{BW[kHz]}{500}$
RegFeilMid (0x29)	7-0	FreqError(15:8)	r	0x0	Middle byte of RF Frequency Error
RegFeilLsb (0x2A)	7-0	FreqError(7:0)	r	0x0	LSB of RF Frequency Error
(0x2B)	-	Reserved	r	n/a	Reserved
RegRssiWideband (0x2C)	7-0	RssiWideband(7:0)	r	n/a	Wideband RSSI measurement used to locally generate a random number
(0x2D) - (0x30)	-	Reserved	r	n/a	Reserved



SX1276/77/78/79

WIRELESS, SENSING & TIMING
DATASHEET

Name (Address)	Bits	Variable Name	Mode	Reset	LoRa™ Description
RegDetectOptimize (0x31)	7-3	Reserved	r	0xC0	Reserved
	2-0	DetectionOptimize	rw	0x03	LoRa Detection Optimize 0x03 → SF7 to SF12 0x05 → SF6
(0x32)	-	Reserved	r	n/a	Reserved
RegInvertIQ (0x33)	7	Reserved	rw	0x0	Reserved
	6	InvertIQ	rw	0x0	Invert the LoRa I and Q signals 0 → normal mode 1 → I and Q signals are inverted
	5-0	Reserved	rw	0x27	Reserved
(0x34) - (0x36)	7-0	Reserved	r	n/a	Reserved
RegDetectionThreshold (0x37)	7-0	DetectionThreshold	rw	0x0A	LoRa detection threshold 0x0A → SF7 to SF12 0x0C → SF6
(0x38)	-	Reserved	r	n/a	Reserved
RegSyncWord (0x39)	7-0	SyncWord	rw	0x12	LoRa Sync Word Value 0x34 is reserved for LoRaWAN networks
(0x3A) - (0x3F)	-	Reserved	r	n/a	Reserved



7.5. Example CRC Calculation

The following routine(s) may be implemented to mimic the CRC calculation of the SX1276/77/78/79:

```

1 // CRC type
2 #define CRC_TYPE_CCITT 0
3 #define CRC_TYPE_IBM 1
4
5 // Polynomial = X^16 + X^12 + X^5 + 1
6 #define POLYNOMIAL_CCITT 0x1021
7 // Polynomial = X^16 + X^15 + X^2 + 1
8 #define POLYNOMIAL_IBM 0x8005
9
10 // Seeds
11 #define CRC_IBM_SEED 0xFFFF
12 #define CRC_CCITT_SEED 0x1D0F
13
14 /*
15  * CRC algorithm implementation
16  *
17  * \param[in] crc Previous CRC value
18  * \param[in] data New data to be added to the CRC
19  * \param[in] polynomial CRC polynomial selection [CRC_TYPE_CCITT, CRC_TYPE_IBM]
20  *
21  * \retval crc New computed CRC
22  */
23 U16 ComputeCrc( U16 crc, U8 data, U16 polynomial )
24 {
25     U8 i;
26     for( i = 0; i < 8; i++ )
27     {
28         if( ( ( crc & 0x8000 ) >> 8 ) * ( data & 0x80 ) != 0 )
29         {
30             crc <<= 1; // shift left once
31             crc ^= polynomial; // XOR with polynomial
32         }
33         else
34         {
35             crc <<= 1; // shift left once
36         }
37         data <<= 1; // Next data bit
38     }
39     return crc;
40 }
41
42 /*
43  * CRC algorithm implementation
44  *
45  * \param[in] buffer Array containing the data
46  * \param[in] bufferlength Buffer length
47  * \param[in] crcType Selects the CRC polynomial [CRC_TYPE_CCITT, CRC_TYPE_IBM]
48  *
49  * \retval crc Buffer computed CRC
50  */
51 U16 RadioPacketComputeCrc( U8 *buffer, U8 bufferlength, U8 crcType )
52 {
53     U8 i;
54     U16 crc;
55     U16 polynomial;
56
57     polynomial = ( crcType == CRC_TYPE_IBM ) ? POLYNOMIAL_IBM : POLYNOMIAL_CCITT;
58     crc = ( crcType == CRC_TYPE_IBM ) ? CRC_IBM_SEED : CRC_CCITT_SEED;
59
60     for( i = 0; i < bufferlength; i++ )
61     {
62         crc = ComputeCrc( crc, buffer[i], polynomial );
63     }
64
65     if( crcType == CRC_TYPE_IBM )
66     {
67         return crc;
68     }
69     else
70     {
71         return ( U16 )( ~crc );
72     }
73 }

```

Figure 54. Example CRC Code

APPENDIX H: Stop and Wait Protocol Transmitter codes

```

/*****
*****
*
* Copyright (c) 2018 Dragino
*
* http://www.dragino.com
*
*****
*****/
#include <string.h>
#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/time.h>
#include <signal.h>
#include <stdlib.h>

#include <sys/ioctl.h>

#include <wiringPi.h>
#include <wiringPiSPI.h>

#include <math.h>

// #####
// #####

#define REG_FIFO          0x00
#define REG_OPMODE        0x01
#define REG_FIFO_ADDR_PTR 0x0D
#define REG_FIFO_TX_BASE_AD 0x0E
#define REG_FIFO_RX_BASE_AD 0x0F
#define REG_RX_NB_BYTES    0x13
#define REG_FIFO_RX_CURRENT_ADDR 0x10
#define REG_IRQ_FLAGS      0x12
#define REG_DIO_MAPPING_1  0x40
#define REG_DIO_MAPPING_2  0x41
#define REG_MODEM_CONFIG    0x1D
#define REG_MODEM_CONFIG2   0x1E
#define REG_MODEM_CONFIG3   0x26
#define REG_SYMB_TIMEOUT_LSB 0x1F
#define REG_PKT_SNR_VALUE 0x19
#define REG_PAYLOAD_LENGTH 0x22

```

```

#define REG_IRQ_FLAGS_MASK      0x11
#define REG_MAX_PAYLOAD_LENGTH 0x23
#define REG_HOP_PERIOD          0x24
#define REG_SYNC_WORD          0x39
#define REG_VERSION             0x42

#define PAYLOAD_LENGTH          0x40

// LOW NOISE AMPLIFIER
#define REG_LNA                  0x0C
#define LNA_MAX_GAIN            0x23
#define LNA_OFF_GAIN            0x00
#define LNA_LOW_GAIN            0x20

#define RegDioMapping1          0x40 // common
#define RegDioMapping2          0x41 // common

#define RegPaConfig              0x09 // common
#define RegPaRamp                0x0A // common
#define RegPaDac                 0x5A // common

#define SX72_MC2_FSK             0x00
#define SX72_MC2_SF7             0x70
#define SX72_MC2_SF8             0x80
#define SX72_MC2_SF9             0x90
#define SX72_MC2_SF10            0xA0
#define SX72_MC2_SF11            0xB0
#define SX72_MC2_SF12            0xC0

#define SX72_MC1_LOW_DATA_RATE_OPTIMIZE 0x01 // mandated for
SF11 and SF12

// sx1276 RegModemConfig1
#define SX1276_MC1_BW_125        0x70
#define SX1276_MC1_BW_250        0x80
#define SX1276_MC1_BW_500        0x90
#define SX1276_MC1_CR_4_5        0x02
#define SX1276_MC1_CR_4_6        0x04
#define SX1276_MC1_CR_4_7        0x06
#define SX1276_MC1_CR_4_8        0x08

#define SX1276_MC1_IMPLICIT_HEADER_MODE_ON 0x01

// sx1276 RegModemConfig2
#define SX1276_MC2_RX_PAYLOAD_CRCON 0x04

// sx1276 RegModemConfig3
#define SX1276_MC3_LOW_DATA_RATE_OPTIMIZE 0x08
#define SX1276_MC3_AGCAUTO        0x04

```



```

// preamble for lora networks (nibbles swapped)
#define LORA_MAC_PREAMBLE          0x34

#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG1 0x0A
#ifdef LMIC_SX1276
#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG2 0x70
#elif LMIC_SX1272
#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG2 0x74
#endif

// FRF
#define REG_FRF_MSB      0x06
#define REG_FRF_MID     0x07
#define REG_FRF_LSB     0x08

#define FRF_MSB         0xD9 // 868.1 Mhz
#define FRF_MID         0x06
#define FRF_LSB         0x66

// -----
// Constants for radio registers
#define OPMODE_LORA     0x80
#define OPMODE_MASK    0x07
#define OPMODE_SLEEP   0x00
#define OPMODE_STANDBY 0x01
#define OPMODE_FSTX    0x02
#define OPMODE_TX      0x03
#define OPMODE_FSRX    0x04
#define OPMODE_RX      0x05
#define OPMODE_RX_SINGLE 0x06
#define OPMODE_CAD     0x07

// -----
// Bits masking the corresponding IRQs from the radio
#define IRQ_LORA_RXTOUT_MASK 0x80
#define IRQ_LORA_RXDONE_MASK 0x40
#define IRQ_LORA_CRCERR_MASK 0x20
#define IRQ_LORA_HEADER_MASK 0x10
#define IRQ_LORA_TXDONE_MASK 0x08
#define IRQ_LORA_CDDONE_MASK 0x04
#define IRQ_LORA_FHSSCH_MASK 0x02
#define IRQ_LORA_CDDETD_MASK 0x01

// DIO function mappings          D0D1D2D3
#define MAP_DIO0_LORA_RXDONE 0x00 // 00-----
#define MAP_DIO0_LORA_TXDONE 0x40 // 01-----
#define MAP_DIO1_LORA_RXTOUT 0x00 // --00----
#define MAP_DIO1_LORA_NOP    0x30 // --11----
#define MAP_DIO2_LORA_NOP    0xC0 // ----11--

```

```

// size of radio tx buffer
#define PAYLOADSIZE 124
#define SEQUENCE_SIZE 4
#define TXBUFFERSIZE 128
#define ackTime 1

// #####
// #####
//
typedef bool boolean;
typedef unsigned char byte;

static const int CHANNEL = 0;

char message[TXBUFFERSIZE+1];

bool sx1272 = true;

byte receivedbytes;

enum sf_t { SF7=7, SF8, SF9, SF10, SF11, SF12 };

/*****
*****
*
* Configure these values!
*
*****
*****/

// SX1272 - Raspberry connections
int ssPin = 6;
int dio0 = 7;
int RST = 0;

// Set spreading factor (SF7 - SF12)
sf_t sf = SF7;

// Set center frequency
uint32_t freq=923200000; // in Mhz! (868.1)

unsigned char hello[32]="HELLO";
char packet[100000];
char txBuffer[TXBUFFERSIZE+1];
float numBytes;
int count=0;
int packet_no;

```

```
int resendindex= 999999999;

bool transmitmode = false;
bool receivemode = false;

bool receiveimage = false;
bool startprocess = true;
bool resend = false;
int timecount;
int timecount1;

struct content
{
    int index;
    char payload[TXBUFFERSIZE+1];
};

struct content packetinc[10000];

void die(const char *s)
{
    perror(s);
    exit(1);
}

void selectreceiver()
{
    digitalWrite(ssPin, LOW);
}

void unselectreceiver()
{
    digitalWrite(ssPin, HIGH);
}

byte readReg(byte addr)
{
    unsigned char spibuf[2];

    selectreceiver();
    spibuf[0] = addr & 0x7F;
    spibuf[1] = 0x00;
    wiringPiSPIDataRW(CHANNEL, spibuf, 2);
    unselectreceiver();

    return spibuf[1];
}
```

```

void writeReg(byte addr, byte value)
{
    unsigned char spibuf[2];

    spibuf[0] = addr | 0x80;
    spibuf[1] = value;
    selectreceiver();
    wiringPiSPIDataRW(CHANNEL, spibuf, 2);

    unselectreceiver();
}

static void opmode (uint8_t mode) {
    writeReg(REG_OPMODE, (readReg(REG_OPMODE) &
~OPMODE_MASK) | mode);
}

static void opmodeLora() {
    uint8_t u = OPMODE_LORA;
    if (sx1272 == false)
        u |= 0x8; // TBD: sx1276 high freq
    writeReg(REG_OPMODE, u);
}

void SetupLoRa()
{
    digitalWrite(RST, HIGH);
    delay(100);
    digitalWrite(RST, LOW);
    delay(100);

    byte version = readReg(REG_VERSION);

    if (version == 0x22) {
        // sx1272
        printf("SX1272 detected, starting.\n");
        sx1272 = true;
    } else {
        // sx1276?
        digitalWrite(RST, LOW);
        delay(100);
        digitalWrite(RST, HIGH);
        delay(100);
        version = readReg(REG_VERSION);
        if (version == 0x12) {
            // sx1276
            printf("SX1276 detected, starting.\n");
            sx1272 = false;
        }
    }
}

```

```

    } else {
        printf("Unrecognized transceiver.\n");
        //printf("Version: 0x%x\n",version);
        exit(1);
    }
}

opmode(OPMODE_SLEEP);

// set frequency
uint64_t frf = ((uint64_t)freq << 19) / 32000000;
writeReg(REG_FRF_MSB, (uint8_t)(frf>>16) );
writeReg(REG_FRF_MID, (uint8_t)(frf>> 8) );
writeReg(REG_FRF_LSB, (uint8_t)(frf>> 0) );

writeReg(REG_SYNC_WORD, 0x34); // LoRaWAN public sync word

if (sx1272) {
    if (sf == SF11 || sf == SF12) {
        writeReg(REG_MODEM_CONFIG,0x0B);
    } else {
        writeReg(REG_MODEM_CONFIG,0x0A);
    }
    writeReg(REG_MODEM_CONFIG2,(sf<<4) | 0x04);
} else {
    if (sf == SF11 || sf == SF12) {
        writeReg(REG_MODEM_CONFIG3,0x0C);
    } else {
        writeReg(REG_MODEM_CONFIG3,0x04);
    }
    writeReg(REG_MODEM_CONFIG,0x72);
    writeReg(REG_MODEM_CONFIG2,(sf<<4) | 0x04);
}

if (sf == SF10 || sf == SF11 || sf == SF12) {
    writeReg(REG_SYMB_TIMEOUT_LSB,0x05);
} else {
    writeReg(REG_SYMB_TIMEOUT_LSB,0x08);
}

writeReg(REG_MAX_PAYLOAD_LENGTH,0x80);
writeReg(REG_PAYLOAD_LENGTH,PAYLOAD_LENGTH);
writeReg(REG_HOP_PERIOD,0xFF);
writeReg(REG_FIFO_ADDR_PTR, readReg(REG_FIFO_RX_BASE_AD));

writeReg(REG_LNA, LNA_MAX_GAIN);

}

boolean receive(char *payload) {
    // clear rxDone

```

```

writeReg(REG_IRQ_FLAGS, 0x40);

int irqflags = readReg(REG_IRQ_FLAGS);

// payload crc: 0x20
if((irqflags & 0x20) == 0x20)
{
    printf("CRC error\n");
    writeReg(REG_IRQ_FLAGS, 0x20);
    return false;
} else {

    byte currentAddr = readReg(REG_FIFO_RX_CURRENT_ADDR);
    byte receivedCount = readReg(REG_RX_NB_BYTES);
    receivedbytes = receivedCount;

    writeReg(REG_FIFO_ADDR_PTR, currentAddr);

    for(int i = 0; i < receivedCount; i++)
    {
        payload[i] = (char)readReg(REG_FIFO);
    }
}
return true;
}

boolean receiveack() {

    long int SNR;
    int rssicorr;

    if(digitalRead(dio0) == 1)
    {

//clear message before next message received
memset(message, 0, TXBUFFERSIZE+1);

        if(receive(message)) {
            byte value = readReg(REG_PKT_SNR_VALUE);
            if( value & 0x80 ) // The SNR sign bit is 1
            {
                // Invert and divide by 4
                value = ( (~value + 1 ) & 0xFF ) >> 2;
                SNR = -value;
            }
            else
            {
                // Divide by 4
                SNR = ( value & 0xFF ) >> 2;
            }
        }
    }
}

```

```

    if (sx1272) {
        rssidcorr = 139;
    } else {
        rssidcorr = 157;
    }

    printf("Packet RSSI: %d, ", readReg(0x1A)-rssidcorr);
    printf("RSSI: %d, ", readReg(0x1B)-rssidcorr);
    printf("SNR: %li, ", SNR);
    printf("Length: %i", (int)receivedbytes);
    printf("\n");
    printf("Payload: %s\n", message);

if(!strcmp(message,"DONE")){
receiveimage = true;
system("python3 timeprint.py");
printf("Received DONE ACK\n\n");

}
else if(!strcmp(message,"0"))
{

system("python3 timeprint.py");
printf("received ACK 0, send again\n\n");
resend=true;

}else if (!strcmp(message,"1")){

    system("python3 timeprint.py");
    printf("received ACK 1, sent success\n\n");
    count++;
}

timecount1 = 0;
return true;

    } // received a message

    } // dio0=1

return false;
}

static void configPower (int8_t pw) {
    if (sx1272 == false) {
        // no boost used for now
        if(pw >= 17) {

```

```

        pw = 15;
    } else if(pw < 2) {
        pw = 2;
    }
    // check board type for BOOST pin
    writeReg(RegPaConfig, (uint8_t)(0x80|(pw&0xf)));
    writeReg(RegPaDac, readReg(RegPaDac)|0x4);

} else {
    // set PA config (2-17 dBm using PA_BOOST)
    if(pw > 17) {
        pw = 17;
    } else if(pw < 2) {
        pw = 2;
    }
    writeReg(RegPaConfig, (uint8_t)(0x80|(pw-2)));
}
}

static void writeBuf(byte addr, byte *value, byte len) {
    unsigned char spibuf[256];
    spibuf[0] = addr | 0x80;
    for (int i = 0; i < len; i++) {
        spibuf[i + 1] = value[i];
    }
    selectreceiver();
    wiringPiSPIDataRW(CHANNEL, spibuf, len + 1);
    unselectreceiver();
}

void txlora(byte *frame, byte datalen) {

    // set the IRQ mapping DIO0=TxDone DIO1=NOP DIO2=NOP
    writeReg(RegDioMapping1,
MAP_DIO0_LORA_TXDONE|MAP_DIO1_LORA_NOP|MAP_DIO2_LOR
A_NOP);
    // clear all radio IRQ flags
    writeReg(REG_IRQ_FLAGS, 0xFF);
    // mask all IRQs but TxDone
    writeReg(REG_IRQ_FLAGS_MASK, ~IRQ_LORA_TXDONE_MASK);

    // initialize the payload size and address pointers
    writeReg(REG_FIFO_TX_BASE_AD, 0x00);
    writeReg(REG_FIFO_ADDR_PTR, 0x00);
    writeReg(REG_PAYLOAD_LENGTH, datalen);

    // download buffer to the radio FIFO
    writeBuf(REG_FIFO, frame, datalen);
    // now we actually start the transmission

```



```

    opmode(OPMODE_TX);

    printf("send: %s\n", frame);
}

char *readfile(){

static char c[1000];
    FILE *fptr;
    if ((fptr = fopen("test.txt", "r")) == NULL) {
        printf("Error! opening file");
        // Program exits if file pointer returns NULL.
        exit(1);
    }

    // reads text until newline is encountered
    fscanf(fptr, "%[^\n]", c);
    //printf("Data from the file:\n%s", c);
    fclose(fptr);

    return c;
}

int main (int argc, char *argv[]) {

    if (argc < 2) {
        printf ("Usage: argv[0] sender|rec [message]\n");
        exit(1);
    }

    wiringPiSetup () ;
    pinMode(ssPin, OUTPUT);
    pinMode(dio0, INPUT);
    pinMode(RST, OUTPUT);

    wiringPiSPISetup(CHANNEL, 500000);

    SetupLoRa();

    if (!strcmp("sender", argv[1])) {

while(startprocess){

startprocess =false;
    count = 0;
receiveimage = false;

    printf("Start to take the picture\n");

```

```

//system time for taking picture
system("python3 timeprint.py");

    // Take a picture with the raspicam
    system("raspistill -o ImageTx.jpg -hf -vf -w 100 -h 100");

    //compress imagetx.jpg to image_compressed
    system("python3 compress.py");

    printf("Picture taken and compress\n");

    //read file store to ImageTx.txt
    system("python3 storebytes.py");

//python to generate appended file
system("python addheader.py");

    //scan appended header file for transmit
    FILE *fptr;
    if ((fptr = fopen("ImageTx_h.txt", "r")) == NULL)
    {
        printf("Error! opening file");
        // Program exits if file pointer returns NULL.
        exit(1);
    }

    // reads text until newline is encountered
    fscanf(fptr, "%[^\n]", packet);
    fclose(fptr);

//print out appended header text file
    printf("Data from the file:\n%s", packet);

    //no of packet in floating point
    numBytes = strlen(packet);
    printf("\nTotal payload length is %.2f\n",numBytes);

    //round to higher integer of no of packet
    packet_no = ceil(numBytes/TXBUFFERSIZE);
    printf("Total packet to be sent is %d\n\n",packet_no);

//count=0
//if str=12345,txbuffer size = 2,raw packet_no = 3,appended packet_no=4 true
while count<4, go while loop 4 times

while(count<packet_no) {

```

```

receiveimage = false;

    //sender mode
    opmodeLora();
// enter standby mode (required for FIFO loading)
opmode(OPMODE_STANDBY);

writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) | 0x08); // set PA ramp-
up time 50 uSec

configPower(23);

    delay(5000);

//split the string into different packet
strncpy(txBuffer,packet+(count*TXBUFFERSIZE),TXBUFFERSIZE);

printf("Payload length is %d\n",strlen(txBuffer));

    //system time for sending
system("python3 timeprint.py");
printf("Sending packet no. %d\n", count);

txlora((byte*)txBuffer, strlen((char *)txBuffer));

printf("Sent packet no. %d\n", count);
printf("\n");

delay(1000);

//receive setup-----
wiringPiSetup () ;
pinMode(ssPin, OUTPUT);
pinMode(dio0, INPUT);
pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

    SetupLoRa();

//receivemode
    opmodeLora();
    opmode(OPMODE_STANDBY);
    opmode(OPMODE_RX);
system("python3 timeprint.py");

```

```

        printf("Listening ACK at SF%i on %.6lf Mhz. for 20s\n",
sf,(double)freq/1000000);
        printf("-----\n");

```

```

delay(1000);

```

```

        if(count<(packet_no-1)){
            //keep listening
            while(!receiveack() {

                }//while(!receivepacket())
        }

```

```

timecount1=0;

```

```

//final look
while(count==(packet_no-1)&&timecount1 < ackTime){

```

```

timecount=0;
    //keep listening
    printf("Listening final ACK for 100s.....\n");

```

```

resend=false;

```

```

        while(!receiveack()&&timecount<500000000) {
            timecount++;
        }//while(!receivepacket()&&timecount<100000000)

```

```

if(receiveimage){
startprocess = true;
packet_no = 0;
timecount1 = ackTime;
system("python3 timeprint.py");
printf("Sent all successfully\n");
printf("Starting new process in 120s.....\n\n");
delay(120000);
}

```

```

timecount1++;

```

```

} //while(count==(packet_no-1)&&timecount1 < ackTime)

```

```

//countdown to exit while loop and take new pic

```

```

if(timecount1 == ackTime&&!resend){
    startprocess = true;
count=packet_no;
    system("python3 timeprint.py");
    printf("All session timeout\n");
    printf("Starting new process in 120s.....\n\n");
delay(120000);

}

delay(1);

//send setup-----
wiringPiSetup () ;
pinMode(ssPin, OUTPUT);
pinMode(dio0, INPUT);
pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

SetupLoRa();

} //while(count<packet_no)

} //while (startprocess)
} //if sender
else {

    // radio init
    opmodeLora();
    opmode(OPMODE_STANDBY);
    opmode(OPMODE_RX);
    printf("Listening at SF%i on %.6lf Mhz.\n", sf,(double)freq/1000000);
    printf("-----\n");
    while(1) {
        //receivepacket();
        delay(1);
    }

}

return (0);}

```

APPENDIX I: Stop and Wait Protocol Receiver code

```

/*****
*****
*
* Copyright (c) 2018 Dragino
*
* http://www.dragino.com
*
*****
*****/

#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/time.h>
#include <signal.h>
#include <stdlib.h>

#include <sys/ioctl.h>

#include <wiringPi.h>
#include <wiringPiSPI.h>

// #####
// #####

#define REG_FIFO          0x00
#define REG_OPMODE        0x01
#define REG_FIFO_ADDR_PTR 0x0D
#define REG_FIFO_TX_BASE_AD 0x0E
#define REG_FIFO_RX_BASE_AD 0x0F
#define REG_RX_NB_BYTES   0x13
#define REG_FIFO_RX_CURRENT_ADDR 0x10
#define REG_IRQ_FLAGS     0x12
#define REG_DIO_MAPPING_1 0x40
#define REG_DIO_MAPPING_2 0x41
#define REG_MODEM_CONFIG  0x1D
#define REG_MODEM_CONFIG2 0x1E
#define REG_MODEM_CONFIG3 0x26
#define REG_SYMB_TIMEOUT_LSB 0x1F
#define REG_PKT_SNR_VALUE 0x19
#define REG_PAYLOAD_LENGTH 0x22
#define REG_IRQ_FLAGS_MASK 0x11
#define REG_MAX_PAYLOAD_LENGTH 0x23

```

```

#define REG_HOP_PERIOD          0x24
#define REG_SYNC_WORD          0x39
#define REG_VERSION            0x42

#define PAYLOAD_LENGTH          0x40

// LOW NOISE AMPLIFIER
#define REG_LNA                 0x0C
#define LNA_MAX_GAIN           0x23
#define LNA_OFF_GAIN           0x00
#define LNA_LOW_GAIN          0x20

#define RegDioMapping1         0x40 // common
#define RegDioMapping2         0x41 // common

#define RegPaConfig            0x09 // common
#define RegPaRamp              0x0A // common
#define RegPaDac               0x5A // common

#define SX72_MC2_FSK           0x00
#define SX72_MC2_SF7           0x70
#define SX72_MC2_SF8           0x80
#define SX72_MC2_SF9           0x90
#define SX72_MC2_SF10          0xA0
#define SX72_MC2_SF11          0xB0
#define SX72_MC2_SF12          0xC0

#define SX72_MC1_LOW_DATA_RATE_OPTIMIZE 0x01 // mandated
for SF11 and SF12

// sx1276 RegModemConfig1
#define SX1276_MC1_BW_125      0x70
#define SX1276_MC1_BW_250      0x80
#define SX1276_MC1_BW_500      0x90
#define SX1276_MC1_CR_4_5       0x02
#define SX1276_MC1_CR_4_6       0x04
#define SX1276_MC1_CR_4_7       0x06
#define SX1276_MC1_CR_4_8       0x08

#define SX1276_MC1_IMPLICIT_HEADER_MODE_ON 0x01

// sx1276 RegModemConfig2
#define SX1276_MC2_RX_PAYLOAD_CRC_ON 0x04

// sx1276 RegModemConfig3
#define SX1276_MC3_LOW_DATA_RATE_OPTIMIZE 0x08
#define SX1276_MC3_AGCAUTO      0x04

// preamble for lora networks (nibbles swapped)
#define LORA_MAC_PREAMBLE       0x34

```

```

#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG1 0x0A
#ifndef LMIC_SX1276
#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG2 0x70
#elif LMIC_SX1272
#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG2 0x74
#endif

// FRF
#define REG_FRF_MSB      0x06
#define REG_FRF_MID      0x07
#define REG_FRF_LSB      0x08

#define FRF_MSB          0xD9 // 868.1 Mhz
#define FRF_MID          0x06
#define FRF_LSB          0x66

// -----
// Constants for radio registers
#define OPMODE_LORA      0x80
#define OPMODE_MASK     0x07
#define OPMODE_SLEEP    0x00
#define OPMODE_STANDBY  0x01
#define OPMODE_FSTX     0x02
#define OPMODE_TX       0x03
#define OPMODE_FSRX     0x04
#define OPMODE_RX       0x05
#define OPMODE_RX_SINGLE 0x06
#define OPMODE_CAD      0x07

// -----
// Bits masking the corresponding IRQs from the radio
#define IRQ_LORA_RXTOUT_MASK 0x80
#define IRQ_LORA_RXDONE_MASK 0x40
#define IRQ_LORA_CRCERR_MASK 0x20
#define IRQ_LORA_HEADER_MASK 0x10
#define IRQ_LORA_TXDONE_MASK 0x08
#define IRQ_LORA_CDDONE_MASK 0x04
#define IRQ_LORA_FHSSCH_MASK 0x02
#define IRQ_LORA_CDDETD_MASK 0x01

// DIO function mappings          D0D1D2D3
#define MAP_DIO0_LORA_RXDONE 0x00 // 00-----
#define MAP_DIO0_LORA_TXDONE 0x40 // 01-----
#define MAP_DIO1_LORA_RXTOUT 0x00 // --00----
#define MAP_DIO1_LORA_NOP    0x30 // --11----
#define MAP_DIO2_LORA_NOP    0xC0 // ----11--

// size of radio rxbuffer
#define PAYLOADSIZE 124

```



```

#define SEQUENCE_SIZE 4
#define TXBUFFER_SIZE 128

struct content
{
    char rxindexstr[SEQUENCE_SIZE+1];
    int rxindex;
    char rxpayload[PAYLOAD_SIZE+1];
};

// #####
// #####
//
typedef bool boolean;
typedef unsigned char byte;

static const int CHANNEL = 0;

char message[TXBUFFER_SIZE+1];

bool sx1272 = true;

byte receivedbytes;

enum sf_t { SF7=7, SF8, SF9, SF10, SF11, SF12 };

/*****
*****
*
* Configure these values!
*
*****
*****/

// SX1272 - Raspberry connections
int ssPin = 6;
int dio0 = 7;
int RST = 0;

// Set spreading factor (SF7 - SF12)
sf_t sf = SF7;

// Set center frequency
uint32_t freq = 923200000; // in Mhz! (868.1)

char rxbuffer[SEQUENCE_SIZE+1];
char ack1[2]="1";
char ack0[2]="0";

```

```

int i=1;
int no_of_packet=10000;
int count=0;

struct content packet[10000];
int timecount=0;
int timecount1=0;
int timecount2=0;
bool receiveimage;
bool receivemessage;

void die(const char *s)
{
    perror(s);
    exit(1);
}

void selectreceiver()
{
    digitalWrite(ssPin, LOW);
}

void unselectreceiver()
{
    digitalWrite(ssPin, HIGH);
}

byte readReg(byte addr)
{
    unsigned char spibuf[2];

    selectreceiver();
    spibuf[0] = addr & 0x7F;
    spibuf[1] = 0x00;
    wiringPiSPIDataRW(CHANNEL, spibuf, 2);
    unselectreceiver();

    return spibuf[1];
}

void writeReg(byte addr, byte value)
{
    unsigned char spibuf[2];

    spibuf[0] = addr | 0x80;
    spibuf[1] = value;
    selectreceiver();
    wiringPiSPIDataRW(CHANNEL, spibuf, 2);

    unselectreceiver();
}

```

```

    }

    static void opmode (uint8_t mode) {
        writeReg(REG_OPMODE, (readReg(REG_OPMODE) &
~OPMODE_MASK) | mode);
    }

    static void opmodeLora() {
        uint8_t u = OPMODE_LORA;
        if (sx1272 == false)
            u |= 0x8; // TBD: sx1276 high freq
        writeReg(REG_OPMODE, u);
    }

    void SetupLoRa()
    {

        digitalWrite(RST, HIGH);
        delay(100);
        digitalWrite(RST, LOW);
        delay(100);

        byte version = readReg(REG_VERSION);

        if (version == 0x22) {
            // sx1272
            printf("SX1272 detected, starting.\n");
            sx1272 = true;
        } else {
            // sx1276?
            digitalWrite(RST, LOW);
            delay(100);
            digitalWrite(RST, HIGH);
            delay(100);
            version = readReg(REG_VERSION);
            if (version == 0x12) {
                // sx1276
                printf("SX1276 detected, starting.\n");
                sx1272 = false;
            } else {
                printf("Unrecognized transceiver.\n");
                //printf("Version: 0x%x\n",version);
                exit(1);
            }
        }
    }

    opmode(OPMODE_SLEEP);

    // set frequency

```

```

uint64_t frf = ((uint64_t)freq << 19) / 32000000;
writeReg(REG_FRF_MSB, (uint8_t)(frf>>16) );
writeReg(REG_FRF_MID, (uint8_t)(frf>> 8) );
writeReg(REG_FRF_LSB, (uint8_t)(frf>> 0) );

writeReg(REG_SYNC_WORD, 0x34); // LoRaWAN public sync word

if (sx1272) {
    if (sf == SF11 || sf == SF12) {
        writeReg(REG_MODEM_CONFIG,0x0B);
    } else {
        writeReg(REG_MODEM_CONFIG,0x0A);
    }
    writeReg(REG_MODEM_CONFIG2,(sf<<4) | 0x04);
} else {
    if (sf == SF11 || sf == SF12) {
        writeReg(REG_MODEM_CONFIG3,0x0C);
    } else {
        writeReg(REG_MODEM_CONFIG3,0x04);
    }
    writeReg(REG_MODEM_CONFIG,0x72);
    writeReg(REG_MODEM_CONFIG2,(sf<<4) | 0x04);
}

if (sf == SF10 || sf == SF11 || sf == SF12) {
    writeReg(REG_SYMB_TIMEOUT_LSB,0x05);
} else {
    writeReg(REG_SYMB_TIMEOUT_LSB,0x08);
}
writeReg(REG_MAX_PAYLOAD_LENGTH,0x80);
writeReg(REG_PAYLOAD_LENGTH,PAYLOAD_LENGTH);
writeReg(REG_HOP_PERIOD,0xFF);
writeReg(REG_FIFO_ADDR_PTR,
readReg(REG_FIFO_RX_BASE_AD));

writeReg(REG_LNA, LNA_MAX_GAIN);

}

boolean receive(char *payload) {
    // clear rxDone
    writeReg(REG_IRQ_FLAGS, 0x40);

    int irqflags = readReg(REG_IRQ_FLAGS);

    // payload crc: 0x20
    if((irqflags & 0x20) == 0x20)

```

```

    {
        printf("CRC error\n");
        writeReg(REG_IRQ_FLAGS, 0x20);
        return false;
    } else {

        byte currentAddr = readReg(REG_FIFO_RX_CURRENT_ADDR);
        byte receivedCount = readReg(REG_RX_NB_BYTES);
        receivedbytes = receivedCount;

        writeReg(REG_FIFO_ADDR_PTR, currentAddr);

        for(int i = 0; i < receivedCount; i++)
        {
            payload[i] = (char)readReg(REG_FIFO);
        }
    }
    return true;
}

bool receivepacket() {

    long int SNR;
    int rssiCorr;

    if(digitalRead(dio0) == 1)
    {
        //clear message before next message received
        memset(message, 0, TXBUFFERSIZE+1);

        if(receive(message)) {

            byte value = readReg(REG_PKT_SNR_VALUE);
            if( value & 0x80 ) // The SNR sign bit is 1
            {
                // Invert and divide by 4
                value = ( ( ~value + 1 ) & 0xFF ) >> 2;
                SNR = -value;
            }
            else
            {
                // Divide by 4
                SNR = ( value & 0xFF ) >> 2;
            }

            if (sx1272) {
                rssiCorr = 139;
            } else {
                rssiCorr = 157;
            }
        }
    }
}

```

```

    printf("Packet RSSI: %d, ", readReg(0x1A)-rssicorr);
    printf("RSSI: %d, ", readReg(0x1B)-rssicorr);
    printf("SNR: %li, ", SNR);
    printf("Length: %i", (int)receivedbytes);
    printf("\n");
    printf("Payload: %s\n", message);

    //store in variable for used later
    strncpy(packet[i].rxindexstr,message+(0),SEQUENCESIZE);

    if((atoi(packet[i].rxindexstr)) == 0){
        packet[atoi(packet[i].rxindexstr)].rxindex = atoi(packet[i].rxindexstr);

    strncpy((packet[atoi(packet[i].rxindexstr)].rxpayload),message+(SEQUENCE
    SIZE),PAYLOADSIZE);

    system("python3 timeprint.py");
    printf("Received packet no. %d\n\n",packet[i].rxindex);

        no_of_packet = atoi((packet[atoi(packet[i].rxindexstr)].rxpayload));
        printf("no of packet to be received is %d\n\n",no_of_packet);

    }else if(atoi(packet[i].rxindexstr) !=
    packet[atoi(packet[i].rxindexstr)].rxindex){

        packet[atoi(packet[i].rxindexstr)].rxindex =
        atoi(packet[i].rxindexstr);
        //printf("index is %d\n",packet[i].rxindex);

    strncpy((packet[atoi(packet[i].rxindexstr)].rxpayload),message+(SEQUENCE
    SIZE),PAYLOADSIZE);

    system("python3 timeprint.py");
    printf("Received packet no. %d\n\n",packet[i].rxindex);

        count++;
        i=atoi(packet[i].rxindexstr)+1;
        //printf("count is %d\n",count);

    }
    timecount1 = 0;
    receivemessage=true;
    return true;
} // received a message

} // dio0=1

```

```

return false;

}

static void configPower (int8_t pw) {
    if (sx1272 == false) {
        // no boost used for now
        if(pw >= 17) {
            pw = 15;
        } else if(pw < 2) {
            pw = 2;
        }
        // check board type for BOOST pin
        writeReg(RegPaConfig, (uint8_t)(0x80|(pw&0xf)));
        writeReg(RegPaDac, readReg(RegPaDac)|0x4);

    } else {
        // set PA config (2-17 dBm using PA_BOOST)
        if(pw > 17) {
            pw = 17;
        } else if(pw < 2) {
            pw = 2;
        }
        writeReg(RegPaConfig, (uint8_t)(0x80|(pw-2)));
    }
}

static void writeBuf(byte addr, byte *value, byte len) {
    unsigned char spibuf[256];
    spibuf[0] = addr | 0x80;
    for (int i = 0; i < len; i++) {
        spibuf[i + 1] = value[i];
    }
    selectreceiver();
    wiringPiSPIDataRW(CHANNEL, spibuf, len + 1);
    unselectreceiver();
}

void txlora(byte *frame, byte datalen) {

    // set the IRQ mapping DIO0=TxDone DIO1=NOP DIO2=NOP
    writeReg(RegDioMapping1,
MAP_DIO0_LORA_TXDONE|MAP_DIO1_LORA_NOP|MAP_DIO2_LOR
A_NOP);
    // clear all radio IRQ flags
    writeReg(REG_IRQ_FLAGS, 0xFF);
    // mask all IRQs but TxDone
    writeReg(REG_IRQ_FLAGS_MASK, ~IRQ_LORA_TXDONE_MASK);
}

```

```

// initialize the payload size and address pointers
writeReg(REG_FIFO_TX_BASE_AD, 0x00);
writeReg(REG_FIFO_ADDR_PTR, 0x00);
writeReg(REG_PAYLOAD_LENGTH, datalen);

// download buffer to the radio FIFO
writeBuf(REG_FIFO, frame, datalen);
// now we actually start the transmission
opmode(OPMODE_TX);

printf("send: %s\n", frame);
}

int main (int argc, char *argv[]) {

    if (argc < 2) {
        printf ("Usage: argv[0] sender|rec [message]\n");
        exit(1);
    }

    if (!strcmp("sender", argv[1])) {
        opmodeLora();
        // enter standby mode (required for FIFO loading)
        opmode(OPMODE_STANDBY);

        writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) | 0x08); // set
PA ramp-up time 50 uSec

        configPower(23);

        printf("Send packets at SF%i on %.6lf Mhz.\n",
sf,(double)freq/1000000);
        printf("-----\n");

        if (argc > 2)
            //strncpy((char *)hello, argv[2], sizeof(hello));

        while(1) {
            //txlora(hello, strlen((char *)hello));
            delay(5000);
        }
    } else
    {

receiveimage = true;

while(receiveimage){

```



```

//reset all variable
receivemessage=false;
receiveimage = false;
count = 0;

i=1;
no_of_packet=10000;
count=0;

timecount=0;
timecount1=0;

//clear rx in c
for (int j=1;j<(no_of_packet+1);j++){
packet[j].rxindex = 0;
memset(packet[j].rxpayload, 0, PAYLOADSIZE+1);
}

printf("*****Start receive new image*****\n");

//receive setup-----

-

wiringPiSetup () ;
pinMode(ssPin, OUTPUT);
pinMode(dio0, INPUT);
pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

SetupLoRa();

//receivemode
opmodeLora();
opmode(OPMODE_STANDBY);
opmode(OPMODE_RX);
printf("Listening at SF%i on %.6lf Mhz.\n", sf,(double)freq/1000000);
printf("-----\n");

//keep listening first packet as receiver
while(!receivepacket() {
}

while(count<(no_of_packet)&&timecount1<5){

delay(2000);

```

```

//send setup-----
    wiringPiSetup () ;
    pinMode(ssPin, OUTPUT);
    pinMode(dio0, INPUT);
    pinMode(RST, OUTPUT);

    wiringPiSPISetup(CHANNEL, 500000);

    SetupLoRa();

    //sendermode
    opmodeLora();
    // enter standby mode (required for FIFO loading)
    opmode(OPMODE_STANDBY);

    writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) | 0x08);
// set PA ramp-up time 50 uSec

    configPower(23);

    delay(5000);

    if(receivemessage){
system("python3 timeprint.py");
        printf("Sending ACK1\n");
        txlora((byte*)ack1, strlen((char *)ack1));
        printf("\n");
    }
    else{
system("python3 timeprint.py");
        printf("Sending ACK0\n");
        txlora((byte*)ack0, strlen((char *)ack0));
printf("\n");
    }

    receivemessage=false;

    delay(1);
//receive setup-----
-----

    wiringPiSetup () ;
    pinMode(ssPin, OUTPUT);
    pinMode(dio0, INPUT);
    pinMode(RST, OUTPUT);

    wiringPiSPISetup(CHANNEL, 500000);

    SetupLoRa();

```

```

        //receivemode
        opmodeLora();
        opmode(OPMODE_STANDBY);
        opmode(OPMODE_RX);
        system("python3 timeprint.py");
        printf("Listening at SF%i on %.6lf Mhz. for 20s\n",
sf,(double)freq/1000000 );
        printf("-----\n");

        timecount = 0;

        //keep listening
        while(!receivepacket()&&timecount<100000000) {
            timecount++;

        }
        delay(1);

        if(timecount==99999999){
            printf("Timeout\n");
            system("python3 timeprint.py");
            printf("\n\n");
        }

        timecount1++;
        }/(count<(no_of_packet)&&timecount1<2)

        if(no_of_packet==count){

        system("python3 timeprint.py");
        printf("Start building image\n");

        /* File pointer to hold reference of input file */
        FILE *fPtr;

        if ((fPtr = fopen("ImageRx.txt", "w")) == NULL) {
            printf("Error! opening file");
            // Program exits if file pointer returns NULL.
            exit(1);
        }
    
```

```

//print all variable to file

    for (int j=1;j<(no_of_packet+1);j++){
        fputs(packet[j].rxpayload,fPtr);
    }

    /* Done with file, hence close file. */
fclose(fPtr);

printf("finally done\n");

    system("python hextoimage.py");
system("python3 timeprint.py");
    printf("Image is print to file\n");

//send done to sender
//clear rxbuffer before store
memset(rxbuffer, 0, SEQUENCESIZE+1);

//store rx buffer
strncpy(rxbuffer,"DONE",SEQUENCESIZE+1);

//send setup-----
-----
wiringPiSetup () ;
pinMode(ssPin, OUTPUT);
pinMode(dio0, INPUT);
pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

SetupLoRa();

//sendermode
opmodeLora();
// enter standby mode (required for FIFO loading)
opmode(OPMODE_STANDBY);

writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) | 0x08);
// set PA ramp-up time 50 uSec

configPower(23);

delay(6000);

system("python3 timeprint.py");
printf("Sending DONE ACK at SF%i on %.6lf Mhz.\n",
sf,(double)freq/1000000);

```

```
printf("-----\n");

txlora((byte*)rxbuffer, strlen((char *)rxbuffer));
printf("\n\n");

delay(1);

receiveimage = true;

} //if(no_of_packet==count) all file received

} //while(receiveimage)
} //else receiver

return (0);
}
```

APPENDIX J: LoRa Multi-Packet Transmission Protocol Transmitter code

```

/*****
*****
*
* Copyright (c) 2018 Dragino
*
* http://www.dragino.com
*

*****/
*****/
#include <string.h>
#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/time.h>
#include <signal.h>
#include <stdlib.h>

#include <sys/ioctl.h>

#include <wiringPi.h>
#include <wiringPiSPI.h>

#include <math.h>

// #####
// #####

#define REG_FIFO          0x00
#define REG_OPMODE        0x01
#define REG_FIFO_ADDR_PTR 0x0D
#define REG_FIFO_TX_BASE_AD 0x0E
#define REG_FIFO_RX_BASE_AD 0x0F
#define REG_RX_NB_BYTES    0x13
#define REG_FIFO_RX_CURRENT_ADDR 0x10
#define REG_IRQ_FLAGS      0x12
#define REG_DIO_MAPPING_1  0x40
#define REG_DIO_MAPPING_2  0x41
#define REG_MODEM_CONFIG    0x1D
#define REG_MODEM_CONFIG2   0x1E
#define REG_MODEM_CONFIG3   0x26
#define REG_SYMB_TIMEOUT_LSB 0x1F
#define REG_PKT_SNR_VALUE 0x19
#define REG_PAYLOAD_LENGTH 0x22
#define REG_IRQ_FLAGS_MASK 0x11

```

```

#define REG_MAX_PAYLOAD_LENGTH 0x23
#define REG_HOP_PERIOD          0x24
#define REG_SYNC_WORD0x39
#define REG_VERSION 0x42

#define PAYLOAD_LENGTH          0x40

// LOW NOISE AMPLIFIER
#define REG_LNA                  0x0C
#define LNA_MAX_GAIN            0x23
#define LNA_OFF_GAIN            0x00
#define LNA_LOW_GAIN            0x20

#define RegDioMapping1          0x40 // common
#define RegDioMapping2          0x41 // common

#define RegPaConfig             0x09 // common
#define RegPaRamp                0x0A // common
#define RegPaDac                 0x5A // common

#define SX72_MC2_FSK            0x00
#define SX72_MC2_SF7            0x70
#define SX72_MC2_SF8            0x80
#define SX72_MC2_SF9            0x90
#define SX72_MC2_SF10           0xA0
#define SX72_MC2_SF11           0xB0
#define SX72_MC2_SF12           0xC0

#define SX72_MC1_LOW_DATA_RATE_OPTIMIZE 0x01 // mandated
for SF11 and SF12

// sx1276 RegModemConfig1
#define SX1276_MC1_BW_125        0x70
#define SX1276_MC1_BW_250        0x80
#define SX1276_MC1_BW_500        0x90
#define SX1276_MC1_CR_4_5        0x02
#define SX1276_MC1_CR_4_6        0x04
#define SX1276_MC1_CR_4_7        0x06
#define SX1276_MC1_CR_4_8        0x08

#define SX1276_MC1_IMPLICIT_HEADER_MODE_ON 0x01

// sx1276 RegModemConfig2
#define SX1276_MC2_RX_PAYLOAD_CRCON 0x04

// sx1276 RegModemConfig3
#define SX1276_MC3_LOW_DATA_RATE_OPTIMIZE 0x08
#define SX1276_MC3_AGCAUTO        0x04

// preamble for lora networks (nibbles swapped)

```

```

#define LORA_MAC_PREAMBLE          0x34

#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG1 0x0A
#ifdef LMIC_SX1276
#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG2 0x70
#elif LMIC_SX1272
#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG2 0x74
#endif

// FRF
#define REG_FRF_MSB      0x06
#define REG_FRF_MID     0x07
#define REG_FRF_LSB     0x08

#define FRF_MSB          0xD9 // 868.1 Mhz
#define FRF_MID          0x06
#define FRF_LSB          0x66

// -----
// Constants for radio registers
#define OPMODE_LORA     0x80
#define OPMODE_MASK    0x07
#define OPMODE_SLEEP   0x00
#define OPMODE_STANDBY 0x01
#define OPMODE_FSTX    0x02
#define OPMODE_TX      0x03
#define OPMODE_FSRX    0x04
#define OPMODE_RX      0x05
#define OPMODE_RX_SINGLE 0x06
#define OPMODE_CAD     0x07

// -----
// Bits masking the corresponding IRQs from the radio
#define IRQ_LORA_RXTOUT_MASK 0x80
#define IRQ_LORA_RXDONE_MASK 0x40
#define IRQ_LORA_CRCERR_MASK 0x20
#define IRQ_LORA_HEADER_MASK 0x10
#define IRQ_LORA_TXDONE_MASK 0x08
#define IRQ_LORA_CDDONE_MASK 0x04
#define IRQ_LORA_FHSSCH_MASK 0x02
#define IRQ_LORA_CDDETD_MASK 0x01

// DIO function mappings          D0D1D2D3
#define MAP_DIO0_LORA_RXDONE 0x00 // 00-----
#define MAP_DIO0_LORA_TXDONE 0x40 // 01-----
#define MAP_DIO1_LORA_RXTOUT 0x00 // --00----
#define MAP_DIO1_LORA_NOP    0x30 // --11----
#define MAP_DIO2_LORA_NOP    0xC0 // ----11--

// size of radio tx buffer

```



```

#define PAYLOADSIZE 124
#define SEQUENCE_SIZE 4
#define TXBUFFERSIZE 128
#define ackTime 10

// #####
// #####
//
typedef bool boolean;
typedef unsigned char byte;

static const int CHANNEL = 0;

char message[TXBUFFERSIZE+1];

bool sx1272 = true;

byte receivedbytes;

enum sf_t { SF7=7, SF8, SF9, SF10, SF11, SF12 };

/*****
*****
*
* Configure these values!
*
*****
*****/

// SX1272 - Raspberry connections
int ssPin = 6;
int dio0 = 7;
int RST = 0;

// Set spreading factor (SF7 - SF12)
sf_t sf = SF7;

// Set center frequency
uint32_t freq=923200000; // in Mhz! (868.1)

unsigned char hello[32]="HELLO";
char packet[100000];
char txBuffer[TXBUFFERSIZE+1];
float numBytes;
int count=0;
int packet_no;
int resendindex= 999999999;

```

```
bool transmitmode = false;
bool receivemode = false;

bool receiveimage = false;
bool startprocess = true;
bool resend = false;
int timecount;
int timecount1;

struct content
{
    int index;
    char payload[TXBUFFERSIZE+1];
};

struct content packetinc[10000];

void die(const char *s)
{
    perror(s);
    exit(1);
}

void selectreceiver()
{
    digitalWrite(ssPin, LOW);
}

void unselectreceiver()
{
    digitalWrite(ssPin, HIGH);
}

byte readReg(byte addr)
{
    unsigned char spibuf[2];

    selectreceiver();
    spibuf[0] = addr & 0x7F;
    spibuf[1] = 0x00;
    wiringPiSPIDataRW(CHANNEL, spibuf, 2);
    unselectreceiver();

    return spibuf[1];
}

void writeReg(byte addr, byte value)
{
```

```

unsigned char spibuf[2];

spibuf[0] = addr | 0x80;
spibuf[1] = value;
selectreceiver();
wiringPiSPIDataRW(CHANNEL, spibuf, 2);

unselectreceiver();
}

static void opmode (uint8_t mode) {
    writeReg(REG_OPMODE, (readReg(REG_OPMODE) &
~OPMODE_MASK) | mode);
}

static void opmodeLora() {
    uint8_t u = OPMODE_LORA;
    if (sx1272 == false)
        u |= 0x8; // TBD: sx1276 high freq
    writeReg(REG_OPMODE, u);
}

void SetupLoRa()
{

    digitalWrite(RST, HIGH);
    delay(100);
    digitalWrite(RST, LOW);
    delay(100);

    byte version = readReg(REG_VERSION);

    if (version == 0x22) {
        // sx1272
        printf("SX1272 detected, starting.\n");
        sx1272 = true;
    } else {
        // sx1276?
        digitalWrite(RST, LOW);
        delay(100);
        digitalWrite(RST, HIGH);
        delay(100);
        version = readReg(REG_VERSION);
        if (version == 0x12) {
            // sx1276
            printf("SX1276 detected, starting.\n");
            sx1272 = false;
        } else {
            printf("Unrecognized transceiver.\n");

```

```

        //printf("Version: 0x%x\n",version);
        exit(1);
    }
}

opmode(OPMODE_SLEEP);

// set frequency
uint64_t frf = ((uint64_t)freq << 19) / 32000000;
writeReg(REG_FRF_MSB, (uint8_t)(frf>>16) );
writeReg(REG_FRF_MID, (uint8_t)(frf>> 8) );
writeReg(REG_FRF_LSB, (uint8_t)(frf>> 0) );

writeReg(REG_SYNC_WORD, 0x34); // LoRaWAN public sync word

if (sx1272) {
    if (sf == SF11 || sf == SF12) {
        writeReg(REG_MODEM_CONFIG,0x0B);
    } else {
        writeReg(REG_MODEM_CONFIG,0x0A);
    }
    writeReg(REG_MODEM_CONFIG2,(sf<<4) | 0x04);
} else {
    if (sf == SF11 || sf == SF12) {
        writeReg(REG_MODEM_CONFIG3,0x0C);
    } else {
        writeReg(REG_MODEM_CONFIG3,0x04);
    }
    writeReg(REG_MODEM_CONFIG,0x72);
    writeReg(REG_MODEM_CONFIG2,(sf<<4) | 0x04);
}

if (sf == SF10 || sf == SF11 || sf == SF12) {
    writeReg(REG_SYMB_TIMEOUT_LSB,0x05);
} else {
    writeReg(REG_SYMB_TIMEOUT_LSB,0x08);
}
writeReg(REG_MAX_PAYLOAD_LENGTH,0x80);
writeReg(REG_PAYLOAD_LENGTH,PAYLOAD_LENGTH);
writeReg(REG_HOP_PERIOD,0xFF);
writeReg(REG_FIFO_ADDR_PTR,
readReg(REG_FIFO_RX_BASE_AD));

writeReg(REG_LNA, LNA_MAX_GAIN);

}

boolean receive(char *payload) {
    // clear rxDone
    writeReg(REG_IRQ_FLAGS, 0x40);

```

```

int irqflags = readReg(REG_IRQ_FLAGS);

// payload crc: 0x20
if((irqflags & 0x20) == 0x20)
{
    printf("CRC error\n");
    writeReg(REG_IRQ_FLAGS, 0x20);
    return false;
} else {

    byte currentAddr = readReg(REG_FIFO_RX_CURRENT_ADDR);
    byte receivedCount = readReg(REG_RX_NB_BYTES);
    receivedbytes = receivedCount;

    writeReg(REG_FIFO_ADDR_PTR, currentAddr);

    for(int i = 0; i < receivedCount; i++)
    {
        payload[i] = (char)readReg(REG_FIFO);
    }
}
return true;
}

boolean receiveack() {

    long int SNR;
    int rssicorr;

    if(digitalRead(dio0) == 1)
    {

//clear message before next message received
memset(message, 0, TXBUFFERSIZE+1);

        if(receive(message)) {
            byte value = readReg(REG_PKT_SNR_VALUE);
            if( value & 0x80 ) // The SNR sign bit is 1
            {
                // Invert and divide by 4
                value = ( ( ~value + 1 ) & 0xFF ) >> 2;
                SNR = -value;
            }
            else
            {
                // Divide by 4
                SNR = ( value & 0xFF ) >> 2;
            }
        }
    }
}

```

```

    if (sx1272) {
        rssiCorr = 139;
    } else {
        rssiCorr = 157;
    }

    printf("Packet RSSI: %d, ", readReg(0x1A)-rssiCorr);
    printf("RSSI: %d, ", readReg(0x1B)-rssiCorr);
    printf("SNR: %li, ", SNR);
    printf("Length: %i", (int)receivedbytes);
    printf("\n");
    printf("Payload: %s\n", message);

if(!strcmp(message,"DONE")){
receiveimage = true;
system("python3 timeprint.py");
printf("Received DONE\n");
}
else if(strlen(message)!=0)
{
resendindex = atoi(message);
resend = true;
system("python3 timeprint.py");
printf("Received resend request ACK\n");
}

timecount1=0;

return true;

    } // received a message

    } // dio0=1

return false;
}

static void configPower (int8_t pw) {
    if (sx1272 == false) {
        // no boost used for now
        if(pw >= 17) {
            pw = 15;
        } else if(pw < 2) {
            pw = 2;
        }
        // check board type for BOOST pin
        writeReg(RegPaConfig, (uint8_t)(0x80|(pw&0xf)));
        writeReg(RegPaDac, readReg(RegPaDac)|0x4);
    }
}

```

```

    } else {
        // set PA config (2-17 dBm using PA_BOOST)
        if(pw > 17) {
            pw = 17;
        } else if(pw < 2) {
            pw = 2;
        }
        writeReg(RegPaConfig, (uint8_t)(0x80|(pw-2)));
    }
}

static void writeBuf(byte addr, byte *value, byte len) {
    unsigned char spibuf[256];
    spibuf[0] = addr | 0x80;
    for (int i = 0; i < len; i++) {
        spibuf[i + 1] = value[i];
    }
    selectreceiver();
    wiringPiSPIDataRW(CHANNEL, spibuf, len + 1);
    unselectreceiver();
}

void txlora(byte *frame, byte datalen) {

    // set the IRQ mapping DIO0=TxDone DIO1=NOP DIO2=NOP
    writeReg(RegDioMapping1,
MAP_DIO0_LORA_TXDONE|MAP_DIO1_LORA_NOP|MAP_DIO2_LOR
A_NOP);
    // clear all radio IRQ flags
    writeReg(REG_IRQ_FLAGS, 0xFF);
    // mask all IRQs but TxDone
    writeReg(REG_IRQ_FLAGS_MASK, ~IRQ_LORA_TXDONE_MASK);

    // initialize the payload size and address pointers
    writeReg(REG_FIFO_TX_BASE_AD, 0x00);
    writeReg(REG_FIFO_ADDR_PTR, 0x00);
    writeReg(REG_PAYLOAD_LENGTH, datalen);

    // download buffer to the radio FIFO
    writeBuf(REG_FIFO, frame, datalen);
    // now we actually start the transmission
    opmode(OPMODE_TX);

    printf("send: %s\n", frame);
}

char *readfile(){

static char c[1000];

```

```

FILE *fptr;
if ((fptr = fopen("test.txt", "r")) == NULL) {
    printf("Error! opening file");
    // Program exits if file pointer returns NULL.
    exit(1);
}

// reads text until newline is encountered
fscanf(fptr, "%[^\n]", c);
//printf("Data from the file:\n%s", c);
fclose(fptr);

return c;
}

int main (int argc, char *argv[]) {

    if (argc < 2) {
        printf ("Usage: argv[0] sender|rec [message]\n");
        exit(1);
    }

    wiringPiSetup () ;
    pinMode(ssPin, OUTPUT);
    pinMode(dio0, INPUT);
    pinMode(RST, OUTPUT);

    wiringPiSPISetup(CHANNEL, 500000);

    SetupLoRa();

    if (!strcmp("sender", argv[1])) {

while(startprocess){

startprocess =false;
    count = 0;
receiveimage = false;

system("python3 timeprint.py");
    printf("Start to take the picture\n");
//system time for taking picture

    // Take a picture with the raspicam
system("raspistill -o ImageTx.jpg -hf -vf -w 100 -h 100");

    //compress imagetx.jpg to image_compressed
system("python3 compress.py");

```



```

printf("Picture taken and compress\n");

//read file store to ImageTx.txt
system("python3 storebytes.py");

//python to generate appended file
system("python addheader.py");

//scan appended header file for transmit
FILE *fptr;
if ((fptr = fopen("ImageTx_h.txt", "r")) == NULL)
{
    printf("Error! opening file");
    // Program exits if file pointer returns NULL.
    exit(1);
}

// reads text until newline is encountered
fscanf(fptr, "%[^\n]", packet);
fclose(fptr);

//print out appended header text file
printf("Data from the file:\n%s", packet);

//no of packet in floating point
numBytes = strlen(packet);
printf("\nTotal payload length is %.2f\n",numBytes);

//round to higher integer of no of packet
packet_no = ceil(numBytes/TXBUFFERSIZE);
printf("packet size is %d\n",packet_no);

//sender mode
opmodeLora();
// enter standby mode (required for FIFO loading)
opmode(OPMODE_STANDBY);

writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) | 0x08); // set PA
ramp-up time 50 uSec

configPower(23);
printf("Send packets at SF%i on %.6lf Mhz.\n", sf,(double)freq/1000000);
printf("-----\n");

//count=0
//if str=12345,txbuffer size = 2,raw packet_no = 3,appended packet_no=4
true while count<4, go while loop 4 times

```

```

while(count<packet_no) {

    delay(5000);

    //split the string into different packet
    strncpy(txBuffer,packet+(count*TXBUFFERSIZE),TXBUFFERSIZE);

    printf("Payload length is %d\n",strlen(txBuffer));

    //system time for sending
    system("python3 timeprint.py");
    printf("Sending packet no. %d\n", count);

    txlorax((byte*)txBuffer, strlen((char *)txBuffer));
    //delay(5000);

    printf("Sent packet no. %d\n", count);

    printf("\n\n");

    count++;
} //while(count<packet_no) send all the packet

timecount1 = 0;

//before entering while loop
system("python3 timeprint.py");
printf("Waiting to receive ack\n");

delay(1000);
//receive setup-----
wiringPiSetup ();
pinMode(ssPin, OUTPUT);
pinMode(dio0, INPUT);
pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

SetupLoRa();

//receivemode
opmodeLora();
opmode(OPMODE_STANDBY);
opmode(OPMODE_RX);
printf("Listening at SF%i on %.6lf Mhz.\n", sf,(double)freq/1000000);
printf("-----\n");

```

```

while(count==packet_no && timecount1 < ackTime){
resendindex =9999999;
resend = false;
receiveimage = false;

timecount=0;
//before entering while loop
system("python3 timeprint.py");
printf("Listening ack for 60s\n");

//keep listening
    while(!receiveack()&&timecount<301450000) {
        timecount++;
    }

delay(1);
//after exiting while loop
if(timecount==301449999){

system("python3 timeprint.py");
printf("Timeout\n");

}

if(receiveimage){
startprocess = true;
packet_no = 0;
}

//delay(5000);

if(resend){

delay(2000);
wiringPiSetup () ;
pinMode(ssPin, OUTPUT);
pinMode(dio0, INPUT);
pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

SetupLoRa();

//sender mode
opmodeLora();
// enter standby mode (required for FIFO loading)
opmode(OPMODE_STANDBY);

```

```
writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) | 0x08); // set PA
ramp-up time 50 uSec
```

```
configPower(23);
```

```
//resend mode-----
```

```
//send missing packet
```

```
strncpy(txBuffer,packet+(resendindex*TXBUFFERSIZE),TXBUFFERSIZE);
```

```
printf("Payload length is %d\n",strlen(txBuffer));
```

```
delay(5000);
```

```
//system time for sending
```

```
system("python3 timeprint.py");
```

```
printf("Resending packet no. %d\n", resendindex);
```

```
txlora((byte*)txBuffer, strlen((char *)txBuffer));
```

```
printf("Resent packet no. %d\n\n ", resendindex);
```

```
delay(1000);
```

```
//receivesetup-----
```

```
wiringPiSetup () ;
```

```
pinMode(ssPin, OUTPUT);
```

```
pinMode(dio0, INPUT);
```

```
pinMode(RST, OUTPUT);
```

```
wiringPiSPISetup(CHANNEL, 500000);
```

```
SetupLoRa();
```

```
//receivemode
```

```
opmodeLora();
```

```
opmode(OPMODE_STANDBY);
```

```
opmode(OPMODE_RX);
```

```
printf("Listening at SF%i on %.6lf Mhz.\n", sf,(double)freq/1000000);
```

```
printf("-----\n");
```

```
delay(1000);
```

```
}//if(resend)
```

```
timecount1++;
```

```

printf("Leaving about %d seconds to receive ack\n\n",(10-timecount1)*60);
//countdown to exit while loop and take new pic
if(timecount1 == ackTime){
    startprocess = true;
    system("python3 timeprint.py");
    printf("All session timeout\n");
    printf("Starting new process.....\n");
}

//count=0;
} //while(count==packet_no)

} //while (startprocess)
} //if sender
else {

    // radio init
    opmodeLora();
    opmode(OPMODE_STANDBY);
    opmode(OPMODE_RX);
    printf("Listening at SF%i on %.6lf Mhz.\n", sf,(double)freq/1000000);
    printf("-----\n");
    while(1) {
        //receivepacket();
        delay(1);
    }

}

return (0);
}

```

APPENDIX K: LoRa Multi-Packet Transmission Protocol Receiver code

```

/*****
*****
*
* Copyright (c) 2018 Dragino
*
* http://www.dragino.com
*

*****/

#include <string>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <string.h>
#include <sys/time.h>
#include <signal.h>
#include <stdlib.h>

#include <sys/ioctl.h>

#include <wiringPi.h>
#include <wiringPiSPI.h>

// #####
// #####

#define REG_FIFO          0x00
#define REG_OPMODE        0x01
#define REG_FIFO_ADDR_PTR 0x0D
#define REG_FIFO_TX_BASE_AD 0x0E
#define REG_FIFO_RX_BASE_AD 0x0F
#define REG_RX_NB_BYTES   0x13
#define REG_FIFO_RX_CURRENT_ADDR 0x10
#define REG_IRQ_FLAGS     0x12
#define REG_DIO_MAPPING_1 0x40
#define REG_DIO_MAPPING_2 0x41
#define REG_MODEM_CONFIG  0x1D
#define REG_MODEM_CONFIG2 0x1E
#define REG_MODEM_CONFIG3 0x26
#define REG_SYMB_TIMEOUT_LSB 0x1F
#define REG_PKT_SNR_VALUE 0x19
#define REG_PAYLOAD_LENGTH 0x22
#define REG_IRQ_FLAGS_MASK 0x11

```

```

#define REG_MAX_PAYLOAD_LENGTH 0x23
#define REG_HOP_PERIOD          0x24
#define REG_SYNC_WORD          0x39
#define REG_VERSION            0x42

#define PAYLOAD_LENGTH          0x40

// LOW NOISE AMPLIFIER
#define REG_LNA                  0x0C
#define LNA_MAX_GAIN            0x23
#define LNA_OFF_GAIN            0x00
#define LNA_LOW_GAIN            0x20

#define RegDioMapping1           0x40 // common
#define RegDioMapping2           0x41 // common

#define RegPaConfig              0x09 // common
#define RegPaRamp                 0x0A // common
#define RegPaDac                  0x5A // common

#define SX72_MC2_FSK              0x00
#define SX72_MC2_SF7              0x70
#define SX72_MC2_SF8              0x80
#define SX72_MC2_SF9              0x90
#define SX72_MC2_SF10             0xA0
#define SX72_MC2_SF11             0xB0
#define SX72_MC2_SF12             0xC0

#define SX72_MC1_LOW_DATA_RATE_OPTIMIZE 0x01 // mandated
for SF11 and SF12

// sx1276 RegModemConfig1
#define SX1276_MC1_BW_125          0x70
#define SX1276_MC1_BW_250          0x80
#define SX1276_MC1_BW_500          0x90
#define SX1276_MC1_CR_4_5           0x02
#define SX1276_MC1_CR_4_6           0x04
#define SX1276_MC1_CR_4_7           0x06
#define SX1276_MC1_CR_4_8           0x08

#define SX1276_MC1_IMPLICIT_HEADER_MODE_ON 0x01

// sx1276 RegModemConfig2
#define SX1276_MC2_RX_PAYLOAD_CRCON 0x04

// sx1276 RegModemConfig3
#define SX1276_MC3_LOW_DATA_RATE_OPTIMIZE 0x08
#define SX1276_MC3_AGCAUTO          0x04

// preamble for lora networks (nibbles swapped)

```

```

#define LORA_MAC_PREAMBLE          0x34

#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG1 0x0A
#ifdef LMIC_SX1276
#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG2 0x70
#elif LMIC_SX1272
#define RXLORA_RXMODE_RSSI_REG_MODEM_CONFIG2 0x74
#endif

// FRF
#define REG_FRF_MSB      0x06
#define REG_FRF_MID     0x07
#define REG_FRF_LSB     0x08

#define FRF_MSB          0xD9 // 868.1 Mhz
#define FRF_MID          0x06
#define FRF_LSB          0x66

// -----
// Constants for radio registers
#define OPMODE_LORA      0x80
#define OPMODE_MASK     0x07
#define OPMODE_SLEEP    0x00
#define OPMODE_STANDBY  0x01
#define OPMODE_FSTX     0x02
#define OPMODE_TX       0x03
#define OPMODE_FSRX     0x04
#define OPMODE_RX       0x05
#define OPMODE_RX_SINGLE 0x06
#define OPMODE_CAD      0x07

// -----
// Bits masking the corresponding IRQs from the radio
#define IRQ_LORA_RXTOUT_MASK 0x80
#define IRQ_LORA_RXDONE_MASK 0x40
#define IRQ_LORA_CRCERR_MASK 0x20
#define IRQ_LORA_HEADER_MASK 0x10
#define IRQ_LORA_TXDONE_MASK 0x08
#define IRQ_LORA_CDDONE_MASK 0x04
#define IRQ_LORA_FHSSCH_MASK 0x02
#define IRQ_LORA_CDDETD_MASK 0x01

// DIO function mappings          D0D1D2D3
#define MAP_DIO0_LORA_RXDONE 0x00 // 00-----
#define MAP_DIO0_LORA_TXDONE 0x40 // 01-----
#define MAP_DIO1_LORA_RXTOUT 0x00 // --00----
#define MAP_DIO1_LORA_NOP    0x30 // --11----
#define MAP_DIO2_LORA_NOP    0xC0 // ----11--

// size of radio rxbuffer

```



```

#define PAYLOADSIZE 124
#define SEQUENCE_SIZE 4
#define TXBUFFER_SIZE 128

struct content
{
    char rxindexstr[SEQUENCE_SIZE+1];
    int rxindex;
    char rxpayload[PAYLOADSIZE+1];
};

// #####
// #####
//
typedef bool boolean;
typedef unsigned char byte;

static const int CHANNEL = 0;

char message[TXBUFFER_SIZE+1];

bool sx1272 = true;

byte receivedbytes;

enum sf_t { SF7=7, SF8, SF9, SF10, SF11, SF12 };

/*****
*****
*
* Configure these values!
*
*****
*****/

// SX1272 - Raspberry connections
int ssPin = 6;
int dio0 = 7;
int RST = 0;

// Set spreading factor (SF7 - SF12)
sf_t sf = SF7;

// Set center frequency
uint32_t freq = 923200000; // in Mhz! (868.1)

char rxbuffer[SEQUENCE_SIZE+1];

```

```

int i=1;
int no_of_packet=10000;
int count=0;

struct content packet[10000];
int timecount=0;
int timecount1=0;
int timecount2=0;
bool receiveimage;

void die(const char *s)
{
    perror(s);
    exit(1);
}

void selectreceiver()
{
    digitalWrite(ssPin, LOW);
}

void unselectreceiver()
{
    digitalWrite(ssPin, HIGH);
}

byte readReg(byte addr)
{
    unsigned char spibuf[2];

    selectreceiver();
    spibuf[0] = addr & 0x7F;
    spibuf[1] = 0x00;
    wiringPiSPIDataRW(CHANNEL, spibuf, 2);
    unselectreceiver();

    return spibuf[1];
}

void writeReg(byte addr, byte value)
{
    unsigned char spibuf[2];

    spibuf[0] = addr | 0x80;
    spibuf[1] = value;
    selectreceiver();
    wiringPiSPIDataRW(CHANNEL, spibuf, 2);

    unselectreceiver();
}

```

```

static void opmode (uint8_t mode) {
    writeReg(REG_OPMODE, (readReg(REG_OPMODE) &
~OPMODE_MASK) | mode);
}

static void opmodeLora() {
    uint8_t u = OPMODE_LORA;
    if (sx1272 == false)
        u |= 0x8; // TBD: sx1276 high freq
    writeReg(REG_OPMODE, u);
}

void SetupLoRa()
{

    digitalWrite(RST, HIGH);
    delay(100);
    digitalWrite(RST, LOW);
    delay(100);

    byte version = readReg(REG_VERSION);

    if (version == 0x22) {
        // sx1272
        printf("SX1272 detected, starting.\n");
        sx1272 = true;
    } else {
        // sx1276?
        digitalWrite(RST, LOW);
        delay(100);
        digitalWrite(RST, HIGH);
        delay(100);
        version = readReg(REG_VERSION);
        if (version == 0x12) {
            // sx1276
            printf("SX1276 detected, starting.\n");
            sx1272 = false;
        } else {
            printf("Unrecognized transceiver.\n");
            //printf("Version: 0x%x\n",version);
            exit(1);
        }
    }
}

opmode(OPMODE_SLEEP);

// set frequency
uint64_t frf = ((uint64_t)freq << 19) / 32000000;

```

```

writeReg(REG_FRF_MSB, (uint8_t)(frf>>16) );
writeReg(REG_FRF_MID, (uint8_t)(frf>> 8) );
writeReg(REG_FRF_LSB, (uint8_t)(frf>> 0) );

writeReg(REG_SYNC_WORD, 0x34); // LoRaWAN public sync word

if (sx1272) {
    if (sf == SF11 || sf == SF12) {
        writeReg(REG_MODEM_CONFIG,0x0B);
    } else {
        writeReg(REG_MODEM_CONFIG,0x0A);
    }
    writeReg(REG_MODEM_CONFIG2,(sf<<4) | 0x04);
} else {
    if (sf == SF11 || sf == SF12) {
        writeReg(REG_MODEM_CONFIG3,0x0C);
    } else {
        writeReg(REG_MODEM_CONFIG3,0x04);
    }
    writeReg(REG_MODEM_CONFIG,0x72);
    writeReg(REG_MODEM_CONFIG2,(sf<<4) | 0x04);
}

if (sf == SF10 || sf == SF11 || sf == SF12) {
    writeReg(REG_SYMB_TIMEOUT_LSB,0x05);
} else {
    writeReg(REG_SYMB_TIMEOUT_LSB,0x08);
}
writeReg(REG_MAX_PAYLOAD_LENGTH,0x80);
writeReg(REG_PAYLOAD_LENGTH,PAYLOAD_LENGTH);
writeReg(REG_HOP_PERIOD,0xFF);
writeReg(REG_FIFO_ADDR_PTR,
readReg(REG_FIFO_RX_BASE_AD));

writeReg(REG_LNA, LNA_MAX_GAIN);

}

```

```

boolean receive(char *payload) {
    // clear rxDone
    writeReg(REG_IRQ_FLAGS, 0x40);

    int irqflags = readReg(REG_IRQ_FLAGS);

    // payload crc: 0x20
    if((irqflags & 0x20) == 0x20)
    {

```

```

printf("CRC error\n");
writeReg(REG_IRQ_FLAGS, 0x20);
return false;
} else {

byte currentAddr = readReg(REG_FIFO_RX_CURRENT_ADDR);
byte receivedCount = readReg(REG_RX_NB_BYTES);
receivedbytes = receivedCount;

writeReg(REG_FIFO_ADDR_PTR, currentAddr);

for(int i = 0; i < receivedCount; i++)
{
payload[i] = (char)readReg(REG_FIFO);
}
}
return true;
}

bool receivepacket() {

long int SNR;
int rssicorr;

if(digitalRead(dio0) == 1)
{
//clear message before next message received
memset(message, 0, TXBUFFERSIZE+1);

if(receive(message)) {

byte value = readReg(REG_PKT_SNR_VALUE);
if( value & 0x80 ) // The SNR sign bit is 1
{
// Invert and divide by 4
value = ( ( ~value + 1 ) & 0xFF ) >> 2;
SNR = -value;
}
else
{
// Divide by 4
SNR = ( value & 0xFF ) >> 2;
}

if (sx1272) {
rssicorr = 139;
} else {
rssicorr = 157;
}
}
}

```

```

    printf("Packet RSSI: %d, ", readReg(0x1A)-rssi corr);
    printf("RSSI: %d, ", readReg(0x1B)-rssi corr);
    printf("SNR: %li, ", SNR);
    printf("Length: %i", (int)receivedbytes);
    printf("\n");
    printf("Payload: %s\n", message);

    //store in variable for used later
    strncpy(packet[i].rxindexstr,message+(0),SEQUENCESIZE);

    if((atoi(packet[i].rxindexstr)) == 0){
        packet[atoi(packet[i].rxindexstr)].rxindex = atoi(packet[i].rxindexstr);

    strncpy((packet[atoi(packet[i].rxindexstr)].rxpayload),message+(SEQUENCE
    SIZE),PAYLOADSIZE);

    system("python3 timeprint.py");
    printf("Received packet no. %d\n",packet[i].rxindex);

        no_of_packet = atoi((packet[atoi(packet[i].rxindexstr)].rxpayload));
        printf("no of packet to be received is %d\n\n",no_of_packet);

    }else if(atoi(packet[i].rxindexstr) !=
    packet[atoi(packet[i].rxindexstr)].rxindex){

        packet[atoi(packet[i].rxindexstr)].rxindex =
    atoi(packet[i].rxindexstr);
        //printf("index is %d\n",packet[i].rxindex);

    strncpy((packet[atoi(packet[i].rxindexstr)].rxpayload),message+(SEQUENCE
    SIZE),PAYLOADSIZE);

    system("python3 timeprint.py");
    printf("Received packet no. %d\n",packet[i].rxindex);

    printf("\n\n");

        count++;
        i=atoi(packet[i].rxindexstr)+1;

    timecount1 = 0;
    timecount2 = 0;

    return true;
    }
    }// received a message

    }// dio0=1

```

```

return false;

}

static void configPower (int8_t pw) {
    if (sx1272 == false) {
        // no boost used for now
        if(pw >= 17) {
            pw = 15;
        } else if(pw < 2) {
            pw = 2;
        }
        // check board type for BOOST pin
        writeReg(RegPaConfig, (uint8_t)(0x80|(pw&0xf)));
        writeReg(RegPaDac, readReg(RegPaDac)|0x4);

    } else {
        // set PA config (2-17 dBm using PA_BOOST)
        if(pw > 17) {
            pw = 17;
        } else if(pw < 2) {
            pw = 2;
        }
        writeReg(RegPaConfig, (uint8_t)(0x80|(pw-2)));
    }
}

static void writeBuf(byte addr, byte *value, byte len) {
    unsigned char spibuf[256];
    spibuf[0] = addr | 0x80;
    for (int i = 0; i < len; i++) {
        spibuf[i + 1] = value[i];
    }
    selectreceiver();
    wiringPiSPIDataRW(CHANNEL, spibuf, len + 1);
    unselectreceiver();
}

void txlora(byte *frame, byte datalen) {

    // set the IRQ mapping DIO0=TxDone DIO1=NOP DIO2=NOP
    writeReg(RegDioMapping1,
MAP_DIO0_LORA_TXDONE|MAP_DIO1_LORA_NOP|MAP_DIO2_LOR
A_NOP);
    // clear all radio IRQ flags
    writeReg(REG_IRQ_FLAGS, 0xFF);
    // mask all IRQs but TxDone
    writeReg(REG_IRQ_FLAGS_MASK, ~IRQ_LORA_TXDONE_MASK);
}

```

```

// initialize the payload size and address pointers
writeReg(REG_FIFO_TX_BASE_AD, 0x00);
writeReg(REG_FIFO_ADDR_PTR, 0x00);
writeReg(REG_PAYLOAD_LENGTH, datalen);

// download buffer to the radio FIFO
writeBuf(REG_FIFO, frame, datalen);
// now we actually start the transmission
opmode(OPMODE_TX);

printf("send: %s\n", frame);
}

int main (int argc, char *argv[]) {

    if (argc < 2) {
        printf ("Usage: argv[0] sender|rec [message]\n");
        exit(1);
    }

    if (!strcmp("sender", argv[1])) {
        opmodeLora();
        // enter standby mode (required for FIFO loading)
        opmode(OPMODE_STANDBY);

        writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) | 0x08); // set
        PA ramp-up time 50 uSec

        configPower(23);

        printf("Send packets at SF%i on %.6lf Mhz.\n",
sf,(double)freq/1000000);
        printf("-----\n");

        if (argc > 2)
            //strcpy((char *)hello, argv[2], sizeof(hello));

        while(1) {
            //txlora(hello, strlen((char *)hello));
            delay(5000);
        }
    } else
    {

        receiveimage = true;

        while(receiveimage){

```



```

//reset all variable
receiveimage = false;
count = 0;

i=1;
no_of_packet=10000;
count=0;

timecount=0;
timecount1=0;

//clear rx in c
for (int j=1;j<(no_of_packet+1);j++){
packet[j].rxindex = 0;
memset(packet[j].rxpayload, 0, PAYLOADSIZE+1);
}

printf("start receive new image\n");

//receive setup-----
-

wiringPiSetup () ;
pinMode(ssPin, OUTPUT);
pinMode(dio0, INPUT);
pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

SetupLoRa();

//receivemode
opmodeLora();
opmode(OPMODE_STANDBY);
opmode(OPMODE_RX);
printf("Listening at SF%i on %.6lf Mhz.\n", sf,(double)freq/1000000);
printf("-----\n");

//keep listening first packet as receiver
while(!receivepacket()) {
}

bool noreceive = true;
timecount2 = 0;

while(noreceive&& timecount2 < 5){

//re-receive //dunno no_of_packet
while(count<(no_of_packet)&&timecount1<2) {

```

```

        timecount = 0;
        system("python3 timeprint.py");
        printf("Listening at SF%i on %.6lf Mhz. for 20s\n",
sf,(double)freq/1000000);
        printf("-----\n");

        //keep listening
        while(!receivepacket()&&timecount<100000000) {
            timecount++;
        }//while(!receivepacket()&&timecount<100000000)

        if(timecount==99999999){
            printf("Timeout\n");
            system("python3 timeprint.py");
            printf("\n\n");
        }

        timecount1++;
    }//while(count<(no_of_packet))

    timecount1=0;

    int j=0;
    bool inloop = true;

    //check whether missing packet
    while(j<(no_of_packet+1)&&inloop){

        if(strlen(packet[j].rxpayload)==0){

            system("python3 timeprint.py");
            printf("Packet no. %d is lost, request to resend\n",j);

        }

        printf("\n");

        //clear rxbuffer before store
        memset(rxbuffer, 0, SEQUENCESIZE+1);

        //store rx buffer
        sprintf(rxbuffer, "%d", j);
        delay(2000);
        //send setup-----
    }

    -----

    wiringPiSetup () ;
    pinMode(ssPin, OUTPUT);

```

```

    pinMode(dio0, INPUT);
    pinMode(RST, OUTPUT);

    wiringPiSPISetup(CHANNEL, 500000);

    SetupLoRa();

    //sendermode
    opmodeLora();
    // enter standby mode (required for FIFO loading)
    opmode(OPMODE_STANDBY);

    writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) |
0x08); // set PA ramp-up time 50 uSec

    configPower(23);

    printf("Send packets at SF%i on %.6lf Mhz.\n",
sf,(double)freq/1000000);
    printf("-----\n");

    delay(5000);
    system("python3 timeprint.py");
    printf("Sending resending request\n");
    txlora((byte*)rxbuffer, strlen((char *)rxbuffer));
    printf("Sent resending request\n");

    printf("\n");
    delay(1000);
    inloop = false;

//receive setup-----
wiringPiSetup ();
pinMode(ssPin, OUTPUT);
pinMode(dio0, INPUT);
pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

SetupLoRa();

//receivemode
opmodeLora();
opmode(OPMODE_STANDBY);
opmode(OPMODE_RX);
printf("Listening at SF%i on %.6lf Mhz.\n",
sf,(double)freq/1000000);
printf("-----\n");

```

```

        }//if(strlen(packet[j].rxpayload)==0)

        j++;

    }//while(j<(no_of_packet))

    if(no_of_packet==count){

system("python3 timeprint.py");
printf("Start building image\n");

        /* File pointer to hold reference of input file */
        FILE *fPtr;

        if ((fPtr = fopen("ImageRx.txt", "w")) == NULL) {
            printf("Error! opening file");
            // Program exits if file pointer returns NULL.
            exit(1);
        }

        //print all variable to file

        for (int j=1;j<(no_of_packet+1);j++){
            fputs(packet[j].rxpayload,fPtr);
        }

        /* Done with file, hence close file. */
        fclose(fPtr);

        printf("finally done\n");

        system("python hextoimage.py");
        printf("Image is print to file\n");
system("python3 timeprint.py");
        printf("\n");

        //send done to sender
        //clear rxbuffer before store
        memset(rxbuffer, 0, SEQUENCESIZE+1);

        //store rx buffer
        strncpy(rxbuffer,"DONE",SEQUENCESIZE+1);
        delay(2000);
        //send setup-----
-----

        wiringPiSetup ();
        pinMode(ssPin, OUTPUT);
    }
}

```

```

        pinMode(dio0, INPUT);
        pinMode(RST, OUTPUT);

wiringPiSPISetup(CHANNEL, 500000);

SetupLoRa();

//sendermode
opmodeLora();
// enter standby mode (required for FIFO loading)
opmode(OPMODE_STANDBY);

writeReg(RegPaRamp, (readReg(RegPaRamp) & 0xF0) | 0x08);
// set PA ramp-up time 50 uSec

configPower(23);

printf("Send packets at SF%i on %.6lf Mhz.\n",
sf,(double)freq/1000000);
printf("-----\n");

delay(5000);
txlora((byte*)rxbuffer, strlen((char *)rxbuffer));

delay(1000);

receiveimage = true;
noreceive = false;

} //if(no_of_packet==count) all file received

timecount2++;
if(no_of_packet!=count){
printf("Try to request for resending for %d times\n",timecount2);}

//countdown to exit while loop and take new pic
if(timecount2 == 3){
receiveimage = true;
printf("receiveimage is true");
}

} //while(noreceive)

if(no_of_packet!=count){
printf("Countdown for waiting reached\n");
system("python3 timeprint.py");}

```

```
}//while(receiveimage)
}//else receiver

return (0);
}
```

APPENDIX L: Header adding python code (addheader.py) in transmitter

```

import binascii
import math

PAYLOADSIZE=226
HEADERSIZE=4

#non-appended file name
filename = 'ImageTx.txt'

with open(filename, 'rb') as f:

    my_str = f.read()

#my_str = "1234567890123456789012345678901234567"

length = len(my_str)
packet_no = math.ceil(length/float(PAYLOADSIZE))
packet_no= int(packet_no)

#print(my_str)
#print(length)
#print(packet_no)

for i in range (1,packet_no):
    #print(i)
    i_str=str(i+1).zfill(HEADERSIZE)
    j=i-1

my_str=my_str[:((PAYLOADSIZE+(j*(PAYLOADSIZE+HEADERSIZE))))] +
i_str + my_str[(PAYLOADSIZE+(j*(PAYLOADSIZE+HEADERSIZE))):]

my_str=my_str[:0] + str(1).zfill(HEADERSIZE) + my_str[0:]
#print(my_str)

packet_no_char = str(packet_no).zfill(PAYLOADSIZE)
#print(packet_no_char)

packet_no_char = packet_no_char[:0] + str(0).zfill(HEADERSIZE) +
packet_no_char[0:]
#print(packet_no_char)

my_str = packet_no_char+ my_str
#print(my_str)

#appended file name

```

```
txtfile = 'ImageTx_h.txt'  
  
with open(txtfile, 'w+') as file:  
    file.write(my_str)  
  
file.close()
```


APPENDIX M: Image compress python code (compress.py) in transmitter

```
from PIL import Image
# image
foo = Image.open("ImageTx.jpg")
# I downsize the image with an ANTIALIAS filter (gives the highest quality)
foo = foo.resize((300,300),Image.ANTIALIAS)
foo.save("ImageTx_compress_95.jpg",optimize=True,quality=95)
```

APPENDIX N: Conversion of image to hexadecimal format python code
(storebytes.py) in transmitter

```
import binascii

filename = 'ImageTx_compress_95.jpg'

with open(filename, 'rb') as f:

    content = f.read()

#print(content)
#print(binascii.hexlify(content))

str_content = str(binascii.hexlify(content), 'utf-8')
#print(str_content)

txtfile = 'ImageTx.txt'

with open(txtfile, 'w+') as file:

    #for i in range(10):
    file.write(str_content)

file.close()
```

APPENDIX O: Conversion of hexadecimal data to image and dropbox
uploading python code (hextoimage.py) in receiver

```

import binascii
import time

#received file name
import dropbox
from dropbox.exceptions import ApiError, AuthError
import time
import datetime
import picamera
import sys, os

#convert received data to image
filename = 'ImageRx.txt'

with open(filename, 'rb') as f:

    data = f.read()

data=data.strip()
data=data.replace(' ', '')
data=data.replace('\n', '')
data = binascii.a2b_hex(data)

timestr = time.strftime("%Y%m%d-%H%M%S")

with open('hextoimage-'+timestr+'.jpg', 'wb') as image_file:
    image_file.write(data)

#upload image to dropbox
# Authorisation token
TOKEN = 'q2Cu3yRj3-
AAAAAAAAAAHxmYLIDeJOZN2V58j2SAvjDOleu9q3pxpHwv48PvakLZU'

# Upload localfile to Dropbox
def uploadFile(localfile):

    # Check that access token added
    if (len(TOKEN) == 0):
        sys.exit("ERROR: Missing access token. "
                "try re-generating an access token from the app console at
dropbox.com.")

    # Create instance of a Dropbox class, which can make requests to API
    print("Creating a Dropbox object...")
    dbx = dropbox.Dropbox(TOKEN)

    # Check that the access token is valid

```

```

try:
    dbx.users_get_current_account()
except AuthError as err:
    sys.exit("ERROR: Invalid access token; try re-generating an "
            "access token from the app console at dropbox.com.")

# Specify upload path
uploadPath = '/' + localfile

# Read in file and upload
with open(localfile, 'rb') as f:
    print("Uploading " + localfile + " to Dropbox as " + uploadPath + "...")

    try:
        dbx.files_upload(f.read(), uploadPath)
    except ApiError as err:
        # Check user has enough Dropbox space quota
        if (err.error.is_path() and
            err.error.get_path().error.is_insufficient_space()):
            sys.exit("ERROR: Cannot upload; insufficient space.")
        elif err.user_message_text:
            print(err.user_message_text)
            sys.exit()
        else:
            print(err)
            sys.exit()

# Delete file
#def deleteLocal(file):
#    os.system("rm " + file)
#    print("File: " + file + " deleted ...")

def main():

    # image file name
    file = 'hextoimage-'+timestr+'.jpg'

    # Upload file
    uploadFile(file)

    # Delete local file
    #deleteLocal(file)

    print("Done")

if __name__ == '__main__':
    main()

```

APPENDIX P: Time printing python code (timezone.py) in both transmitter
and receiver

```
from datetime import datetime
from datetime import timezone

current_GMT_timestamp = datetime.utcnow()
print (current_GMT_timestamp)
```