2021

Image Transmission over Resource-constrained Low-Power Radio Networks

Paschal O'Connor
Technological University Dublin, paschal.oconnor@tudublin.ie

Image Transmission over Resource-constrained Low-Power Radio Networks

Paschal O'Connor

Supervised by:

Mr John Dalton; Prof. Max Ammann; Mr Joseph Kellegher

Thesis presented for the degree of

Master of Philosophy

School of Electrical and Electronic Engineering

TU Dublin

Ireland

2021

ABSTRACT

The transmission of large amounts of data over resource-constrained radio frequency (RF) networks is impacted by regulatory constraints and can affect reliability due to channel congestion. These barriers limit the use case to specific applications. This research extends the use case scenario to include the transmission of digital images over such networks which to date has not been widely documented. To achieve this, the overall data volume needs to be reduced to manageable limits. Drawing on previous theoretical work this research explored, developed and implemented novel image compression techniques suitable for use in resource-constrained RF networks.

A compression technique was developed which allows variable compression ratios to be selected dependent on the specific use case. This was implemented in an end-to-end low-power radio network operating in license-free spectrum using a customised radio frequency testbed. The robust compression scheme which was developed here enabled out-of-sequence packet reception, further increasing the reliability of the transmission.

To allow detailed viewing of a region of interest (ROI) within a large format image (quarter video graphics array) to be transmitted, a novel algorithm was designed and implemented. This enabled the transmission of a region of interest (ROI) in an uncompressed format as a stand-alone image portion, or in combination with a fully compressed image. Significantly, this yielded flexibility in the quantity of data to be transmitted which could increase the lifespan of battery powered devices. A further development allowed direct manipulation of individual image pixels. This permitted additional data, such

as battery voltage level to be directly embedded in the transmitted image data. An advantage of this innovative method was that it did not incur any extra overhead in data volume requirements.

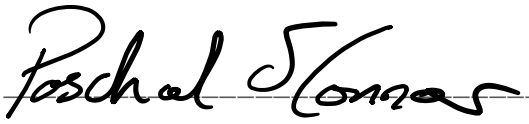
The embodied system developed is an agnostic image compression algorithm and is suitable for use with resource-constrained devices and networks. Results showed that high compression ratios (70%) with good peak signal-to-noise ratio (PSNR) of approximately 36dB was achievable for a complete end-to-end transmission system.

DECLARATION

I certify that this thesis which I now submit for examination for the award of Master of Philosophy, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work.

This thesis was prepared according to the regulations for postgraduate study by research of Technological University Dublin and has not been submitted in whole or in part for another award in any other third level institution.

The work reported on in this thesis conforms to the principles and requirements of the TU Dublin guidelines for ethics in research.

Signature  Date 10-11-21

Candidate: Paschal O'Connor

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and thanks to my project supervisor and co-supervisors Mr. John Dalton, Prof. Max Ammann and Mr. Joseph Kellegher for their advice and guidance during the course of this project. I would like to thank all the staff in the postgraduate research office for their professionalism and kind support throughout this process.

I would also like to thank my wife Maeve and family for their support and patience over the course of this project.

Table of Contents

List of Figures

List of Tables

List of Equations

List of Abbreviations.

Abbreviation	*Definition*
3GPP	Third generation Partnership Project
4G	Fourth generation
AFA	Adaptive Frequency Agility
AIoT	Artificial Intelligence of Things
API	Application Program Interface
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
CAT-M1	Category Machine
CIoT	Cellular Internet of Things
CSS	Chirp Spread Spectrum
D7A	Dash 7 alliance
dB	Decibel
DPSK	Differential Phase Shift Keying
DQPSK	Differential Quadrature Phase Shift Keying
DSSS	Direct Sequence Spread Spectrum
eNB	e Node B
ETSI	European Telecommunications Standards Institute
FSK	Frequency Shift Keying
FTDI	Future technology Devices International
GFSK	Gaussian Frequency Shift Keying
GPRS	General Packet Radio System
ICT	Information And Communications Technology
IDE	Integrated Development Environment
IoT	Internet of Things
ISI	Inter Symbol Interference
ISM	Industrial Scientific and Medical
JPEG	Joint Photographic Experts Group
LBT	Listen Before Talk
LE	Low Energy
LoRa	Long Range
LPWAN	Low Power Wireless Area network
LR	Long Range
LTE	Long Term Evolution
LTE-M	Long term Evolution Machine
MAC	Medium Access Control

Mbps	Mega-bits per second
MCU	Microcontroller unit
MSE	Mean Squared Error
MSK	Minimum Shift Keying
NB-IoT	Narrow Band Internet of Things
NSO	National Standards Organisations
OSI	Open System Interconnection Model
OTS	Off-The-Shelf
PN9	Pseudorandom Number 9 bit
PRB	Physical Resource Block
PRN	Pseudo Random Number
PSNR	Peak Signal to Noise Ratio
QoS	Quality of Service
QVGA	Quarter Video Graphics Array
RF	Radio Frequency
RLE	Run Length Encoding
ROI	Region of Interest
SF	Spreading Factor
SoCs	System on Chips
SPI	Serial Peripheral Interface
SRD	Short Range Device
TCP/IP	Transmission Control protocol / Internet Protocol
ToA	Time On Air
UART	Universal Asynchronous Receiver Transmitter
UE	User end-device
UHF	Ultra-High Frequency
Wh	Watt Hour

CHAPTER 1 INTRODUCTION

This thesis explored the specific use case of extending the normal types of data transmission in internet-of-things (IoT) type networks to include still image transmission over private IoT type networks. Image data requires a larger bandwidth or longer time-on-air to transmit relative to e.g., temperature or location data. A means to reduce this data volume by compression, and a means to select a portion of an image for transmission has been developed for use across diverse RF low power wide area networks (LPWAN).

Typical transmission data payload requirements for IoT type devices vary from 10 -100 bytes of data with a frequency of transmission of 1-24 times /Day. These figures will vary with the use case, but because of the low transmission rate requirements, battery powered devices are normally used. Information transmitted typically includes temperature/humidity, switch status, presence detection and location of device. This low volume data is suited to transmission over private IoT type networks operating in the industrial scientific and medical (ISM) frequency bands within the constraints imposed by country regulators. This research has expanded the use case scenarios for low-power WLAN such as, situational awareness, agriculture growth monitoring, forest fire detection, bridge crack monitoring, bacterial growth monitoring etc. An end-to-end RF network capable of transmitting still black & white images over a long-range using off-the-shelf (OTS)

components was implemented. The developed system allows viewing of the reconstructed image on a remotely connected TCP/IP device normally a PC.

The growth of IoT type communications systems is based on the expected transmission of small amounts of data by many interconnected devices. For RF type systems, as opposed to hardwired type networks such as IEEE 802.3 or Ethernet, Modbus etc., additional constraints are imposed on the reliability and defined response times required in defined use cases e.g., mission critical systems often found in industrial applications. The management of multiple users over an RF network requires a well thought out medium access control (MAC) layer to ensure each device can remain connected and be allowed transmit data in a controlled and fair way. If the RF system is operating in an ISM frequency band, this imposes further constraints on transmission power allowable and time-on-air (TOA) constraints. Despite these challenges, the implementation of an RF sensor type network has huge financial savings in terms of wiring costs, network expansion and flexibility of design. The availability of remote information from hard-to-reach areas makes these types of systems attractive relative to hardwired systems. The implementation of an RF sensor network requires detailed knowledge of the environment in which it is expected to operate and the gap between what is expected and what can be achieved needs to be defined at the outset. There have been several attempts by various groups to ‘standardise’ a communication type network suitable for most applications (within the IoT type definition); one such network is LoRaWAN® which allows large numbers of devices to be connected via public or private networks. The limitation of these public type

networks is the restricted amount of data transmission allowed and the fair access policy which is enforced. This has the net effect of further limiting the amount and frequency of data transmissions. Other such public networks include SigFox® which severely limit the amount and frequency of data transmission for each end device, and this has been eliminated for consideration for this use case.

1.1 Problem Definition

This research addressed the problem associated with transmission of large amounts of data (image) over RF networks designed for low-data volume transmission. Image transmission over resource-constrained RF systems present unique challenges in terms of available bandwidth, regulatory constraints, and available power supply. LPWAN operating in the ISM radio frequency bands impose further difficulties regarding channel availability and usage and can be regarded as hostile environments for ensuring reliability of transmission and guaranteed quality of service (QoS). These systems operate in ‘Lossy’ environments and some loss of data is expected, further complicating the reliable reception of large amounts of data such as image transmission. Battery powered devices used in typical IoT systems operate on resource constrained hardware and typically transmit low data volumes periodically in quantities of no more than 250 bytes per transmission. A black and white image of standard size (QVGA) will contain 76800 bytes of information and transmission of this large amount of data will be impacted by power availability and hence battery lifetime. A variable means to

compress this data (High, Medium, and Low) and preserve acceptable image quality has been proposed and implemented taking two different private LPWAN system configurations into consideration, a system capable of reception up to a range of at least 100m and one capable of reception up to 5km i.e., greater range capability than existing standard Wi-Fi or Bluetooth systems. To assess the suitability of various network types a review of fundamental RF modulation methods was undertaken. These were considered in the context of suitable low-cost, off-the-shelf (OTS) modules to construct an RF system capable of meeting the requirements of this use-case.

1.2 Solution approach

To explore this specific use case a low power wireless network was implemented using OTS modules which served as a test bed platform for the development and testing of an image transmission system. The developed image compression and manipulation algorithm is suitable for use across multiple IoT type RF networks. Emphasis was placed on simple algorithm design with acceptable performance and low memory footprint. Suitability for use at the physical layer is a primary consideration. The system topology selected was a star type network consisting of end-devices (with camera) wirelessly connecting to a central client transceiver. This was connected via TCP/IP to a remote monitoring system. A program was implemented to allow reconstruction and decompression of the received data for viewing/storage of the image received.

1.3 Thesis Outline

Chapter 2 provides a review of the relevant published work around image transmission over resource-constrained WLAN. Cellular internet of things (CIoT) and private/public RF networks operating in the IoT space are compared and evaluated here.

Chapter3 assesses the technical capability of three modulation schemes w.r.t bandwidth, bitrate, and the effect of noise in designing an RF network. A link budget analysis is carried out on each modulation type to assess its suitability for use within this use case definition.

Chapter 4 describes the rationale and design choices made in the development of a system testbed. The choice of hardware and software used during this research is described here.

Chapter 5 details the design and implementation of a suitable RF network. This describes the data compression algorithm, its development and implementation. An end-to-end system operational test is described in detail.

The final chapter, Chapter 6, discusses the rationale for the methodology, draws conclusions from the findings and outlines directions for future work.

2 CHAPTER 2 LITERATURE REVIEW

The use of low-power WLAN for image transmission (excluding video transmission) was the focus of this research. To assess the suitability of an RF system for this use case it was necessary to evaluate the physical capabilities of various modulation schemes. The impact on bandwidth, duty-cycle, expected communication range, and suitability for use in resource-constrained devices such as low-power micro-controllers was explored. Several dominant RF systems in current use are evaluated in this section. Standard image compression techniques are also discussed and assessed for suitability within the confines of this use case.

2.1 Current State of the Art in IoT networks

A wide range of RF low power network topologies exist, broadly divided into two categories, CIoT and Private/Public IoT systems operating in the ISM bands. Systems operating in the private domain tend to use proprietary protocols but not exclusively e.g., D7A [1].

2.1.1 LPWAN

For the sake of a comprehensive assessment, CIoT systems will be described, but focus will be placed on systems operating in the ISM bands allowing for unknowns regarding cost, local technical support, availability of hardware and purchase quantities, all required to operate within the cellular network structure.

2.1.2 CIoT

Traditional cellular options such as 4G and long-term evolution (LTE) networks are designed to offer high speed connectivity where power consumption is not considered a priority. In response to the expected growth rate of connected devices which typically transmit a small amount of data periodically, CIoT providers have responded with several options to enable low-power, long-range applications connect via the cellular network which are third generation partnership project (3GPP) [2] 5G standardised technologies.

2.1.3 NB-IoT

Completion of narrow band internet of things (NB-IoT) standardisation was achieved in 2016 in release 13 (LTE Advanced Pro) by 3GPP. Release 14 enhanced the specifications LTE Cat-NB2. NB-IoT can be deployed ‘in-band’ utilising resource blocks within a normal LTE carrier or used within the unused resource blocks within an LTE carrier’s guard-band. It may also be deployed in standalone mode for use in dedicated spectrum [3]. In standalone deployment, NB-IoT can occupy one global system for communication (GSM) channel (200Khz) and is intended for GSM channel refarming using existing infrastructure. When used in guard-band, or in-band, it will use one or more physical resource blocks (PRB) of LTE (180Khz). Ratasuk *et al*. [4] reported that a latency target of < 10 seconds for exception reports is achieved with a link budget for stand-alone scenario of 164dB and application layer data rate of 2.73 kbps in the downlink, and 0.31 kbps in the uplink. NB-IoT is designed to support massive numbers of low-throughput devices with a target of 20dB

extended coverage compared to legacy GPRS devices allowing communication with hard-to-reach devices e.g., underground car parks. To achieve a battery lifespan of more than 10 years, a data packet size of 200 bytes and a once-a-day transmission constraint is imposed [4]. Sun, Rongrong *et al.*, present a specific framework for the use of NB-IoT in unlicensed bands with similar results [5]; for this specific use case NB-IoT will not be considered.

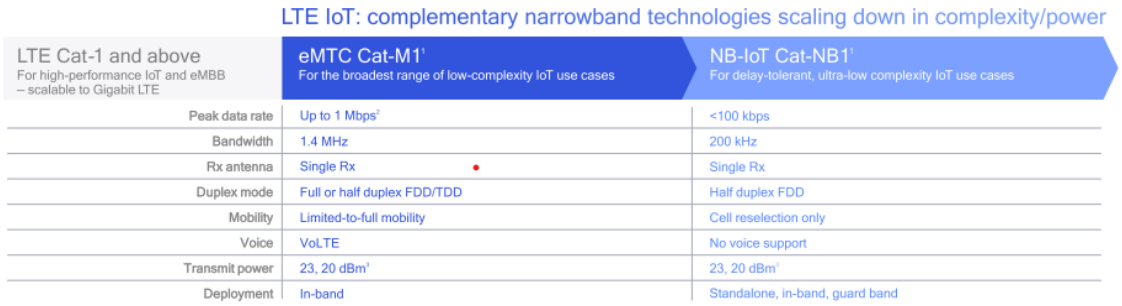
2.1.4 CAT-M1

The second LPWA technology specified in 3GPP as a 5G technology is LTE-M designed to support the Internet of Things. Rel-13 specifications were completed in 2016 with further enhancements in Rel-14 and Rel-15. To meet the 5G requirements, 3dB power spectral density boosting is used in the downlink with the addition of 4 receive antennas required at the e-nodeB (eNB) [6]. Category-M1 (CAT-M1) has an RF bandwidth of 1.4MHz and a maximum transport block size of 1000 bits typically operating in half-duplex mode with the ability to support massive number of devices with a 10-year battery life (capacity of 5 Wh) and < 10 second latency [6]. In 3GPP Rel-14 major enhancements were introduced (CAT-M2) supporting 5-MHz bandwidth and further improvements to CAT-M1 with the ability to support applications such as video and voice. Using an uplink report of size 200 bytes, the UE wakes up from power save mode and transmits an uplink report once per day and returns to power save mode, an estimated 7.6 years of battery life was reported [6]. Actual data transmission impacts on battery life determined by data size and frequency of transmission. A peak data rate of 1Mbps is achievable and can operate in full or half-duplex mode and can only operate

in-band. CAT-M technologies have not been considered for this use case as Network operators in Ireland have not deployed CAT-M as of this time.

Table 1 shows a summary of current 5G WLAN offerings [7].

Table 1: CIoT LPWLAN Summary



Private/Public ISM IoT

Given the need for remote access to data whether Industrial, commercial, or domestic, a move to RF type systems has obvious benefits including cost savings and scalability. The elimination of wiring allows rapid deployment of sensor/actuator nodes but introduces issues regarding reaction time, security, and remote power requirements if powered by battery. Maintenance and remote diagnostics capability are part of a successful RF network deployment if costly ‘truck’ rollouts are to be avoided. Additionally, bi-directional communications are desirable to enable re-designation of end-user device functionality, and to avail of latest firmware updates. This embedded functionality has associated costs in terms of available power budget, channel congestion while conforming to local regulatory restrictions. Operating in the ISM frequency bands where no license is required constitutes a noisy environment compared to operating in licensed bands. License holders are able to control their RF channel to a much greater degree and can offer a

defined QoS. Several RF Network configurations which have gained traction in recent years have been evaluated in the context of suitability for this particular use case i.e., the transmission of still images over medium and long range (100m – 5000m) in an urban environment.

2.2 LoRa™

Semtech is a leading supplier of high performance analog and mixed-signal semiconductors who manufacture a long range, low power integrated circuit suitable for use in ISM RF bands from low kHz up to 2.4GHz (LoRa™). A range of transceivers, receivers and transmitters are manufactured enabling low-power, long-range connectivity of sensors. A proprietary modulation scheme is used, chirp spread spectrum (CSS), based on spread spectrum technologies and FSK at the physical layer. The LoRa Alliance® [8] has developed a MAC layer open standard protocol (LoRaWAN®), enabling the exchange and control of multiple sensor data. There are many LoRaWAN® networks deployed throughout the world which allow large scale deployment of sensor networks in the private and public domains where UE devices meet the low data volume requirement. The LoRaWAN® MAC layer allows bi-directional communication with duty-cycle restrictions in compliance with regulatory authorities' requirements. This restricts the TOA of each transmitter, hence limiting the effectiveness of bi-directional communications. As each base-station can also be a transmitter, normally connected to hundreds of end-devices, duty-cycle limits can quickly be exceeded in this situation. Strict limitations and 'fair usage' schemes are normally employed by service providers [9] further reducing the available TOA for each transmitter. LoRa™ network topology is a star type, each user

end device (UE) is connected to a central gateway which generally has some form of internet connectivity e.g., Wi-Fi, Lte, Ethernet. Long range is achieved because of the claimed 157dB Link-Budget availability and a payload size of 255 Octets. The physical parameters satisfy some of the requirements for this use case.

2.3 ESP-NOW

Espressif Systems is a public multinational, fabless semiconductor company established in 2008 focused on developing cutting-edge Wi-Fi and Bluetooth, low-power artificial intelligence internet of things (AIoT) solutions. The ESP32 Series SoCs includes a 32-bit MCU and 2.4GHz Wi-Fi and Bluetooth/Bluetooth LE communications capability with a sleep current of less than 5µA, making it suitable for battery-powered applications [10]. ESP-NOW is a communications protocol developed by Espressif based on IEEE 802.11-2012 standard action vendor frame for information technology [11]. This enables multiple devices to communicate with one another without using Wi-Fi in the 2.4Ghz ISM band. An initial pairing of devices is needed prior to communication, after which secure, and peer-to-peer communication can continue without handshaking. This allows fast transfer of data at a rate of 1-2Mbps for up to 20 devices un-encrypted or 10 devices encrypted per network [12]. A payload of 250 bytes can be carried using direct sequence spread spectrum (DSSS) and uses baseband modulation of differential binary phase shift keying (DBPSK) and differential quadrature shift keying (DQPSK) to provide the 1Mbps and 2Mbps data rates respectively [13]. During the hand-shaking mode of system initialisation, if the long-range mode (LR) is enabled on Tx. and Rx. a bit rate of 125kbps is negotiated if 802.11B mode fails. Each

device can operate as a local device or as a peer and can be assigned different roles, a controller, slave, or combination of both. A network can operate as a simple two-device network (peer-to-peer) or in multi-peer to station (STA) mode. There are no duty-cycle restrictions applicable to this ISM band [14].

2.4 DASH7

The DASH7 Alliance was formed to foster the existence and further development of the DASH7 protocol specification (based on ISO 1800-7). It is an open standard for bi-directional sub-GHz medium range wireless communication for ultra-low-power sensor-actuator applications using private networks [1]. The specification includes OSI layers 1-7, PHYSICAL layer to APPLICATION layer, and is regarded as a mid-range (up to 500m) communications network. The communications protocol is described as a BLAST type:

Bursty: Data transfer is abrupt and does not include streaming content such as video, audio, or other isochronous forms of data.

Light: Packet sizes are limited to 256 bytes.

Asynchronous: D7A method of communication is by request-response, which requires no periodic network "hand-shaking" or synchronization between devices.

Stealth: D7A does not use discovery beacons, end nodes can choose to respond only to pre-approved devices.

Transitional: A D7A system of devices is inherently mobile or transitional. Unlike other wireless technologies D7A is upload-centric, not download-

centric. Devices do not need to be managed extensively by fixed infrastructure (i.e. base stations) to respond only to pre-approved devices.

There are three data rates available, 9.6kb/s, 55.55 kb/s and 166.667 kb/s using GFSK modulation and can operate in ISM bands 433.060 – 434.785 MHz, 863.000 – 870.000 MHz and 902.000 – 928.000 MHz using 1/2 convolutional code or 1/1 PN9, p8 [15].

2.5 Image Compression

Most image compression algorithms are based on generic standards which are suitable for video transmission of data such as JPEG or its derivatives. For the transmission of still images over lossy RF networks where the end-device is resource-constrained, and the transmission of still images is only required, the algorithm overhead can be onerous. The selection, or development of a suitable, yet simple and adaptable algorithm for this particular use case involves a trade-off between acceptable receive image quality and power consumption, while working within the constraints of RF Regulatory bodies.

Usual compression schemes such as JPEG are not suitable for use in bandlimited networks where data payload size is limited to 255 bytes (in these types of WLAN). The transmission of image data over IoT type RF networks adds considerable burden to resource-constrained devices where memory and power requirements are limited compared to ‘normal’ transmission media such as Wi-Fi or Cellular networks. The impact of on-air time for battery driven devices, typically micro-controller platforms, has a considerable effect on the expected lifetime of a battery. One way to minimise this time is to

reduce the amount of data to be transmitted i.e., data compression. The implementation of a data compression algorithm suitable for use in lossy RF networks is one objective of this use case. An outline target specification for this application is shown in Table 2.

Table 2: Outline Specification for image compression

1	Suitable for use across resource-constrained devices:	8-bit µC e.g., AVR 16-bit µC e.g., MSP 32-bit µC e.g., ESP32 Low-complexity Algorithm
2	Low memory footprint	<16k Ram, <32k Rom
3	No Hardware floating-point calculation needed.	Software floating point or Integer only. Energy Efficient computation.
4	Suitable for use in RF lossy networks	Packet size of ≤ 256Bytes, out of sequence reception tolerant.
5	Packet loss tolerant	Display of received packets, full image not required.
6	Suitable for image quality of QVGA b&w	Must be capable of compressing 320*240 bytes, 8-bit greyscale
7	Region of Interest Selectable	Transmission of high-Quality region of interest, for slower connections. Means a reduction in data to transmit.
8	Direct Pixel manipulation.	Embedding of information directly within the image or raw data manipulation.

There are several standard compression mechanisms in use for image transmission and can be divided into lossless or lossy types. If using a lossy type of compression, one measure of visual quality is the PSNR for a given compression ratio. To achieve visually lossless condition means more than 40dB reconstruction image quality.

PSNR of a transmitted image can be calculated as shown in Equation 1.

Equation 1:PSNR

$$PSNR = 10\log_{10}\frac{S^2}{MSE}$$

Image quality is assessed quantitatively and is based on the difference between the pixels of the reconstructed image after transmission and the original image. For an 8-bit image, S is 255 and MSE is the mean squared error, which is the average of the squared difference in the intensity of the pixels in the original image and the output image. It can be calculated as shown in Equation 2.

Equation 2:MSE

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=1}^{n-1}[I(i,j) - K(i,j)]^2$$

Where m and n are the respective length and width of the image matrix in pixels $I(i,j)$ and $K(i,j)$ are functions describing the intensity of individual pixels in the transmitted and received image, respectively. In 2016, C. Pham [16] proposed a remote visual surveillance system using LoRa® physical layer which transmits a compressed 128x128 8-bit grayscale image with variable quality factor selection. A limitation factor on achievable range is TOA which is constrained by duty-cycle restrictions imposed by ETSI and for

433MHz is 10% (360 sec) as shown in Table 3. C.Pham [16] has tabulated the TOA for various BW and spreading factors (SF); for payload size of 255 bytes, it ranges from 0.1009 to 9.15 seconds. For a raw uncompressed image size of 160x120 pixels (ROI for this use case), it takes (19200/255) *0.6333 = 47.68 sec using LoRa mode 7. Compressing the data image improves on this figure and hence battery life of the end device. In this instance, a compression scheme tolerant to packet loss is used based on [17] which improves on the energy consumption involved in popular algorithms such as JPEG, JPEG2000 or SPHIT. It operates on 8x8 pixel blocks using an optimised block interleaving method, and the image matrix must have the same number of rows and columns. The image is transmitted in blocks of 250 bytes which allows for out-of-sequence reconstruction and is tolerant of packet loss. The computational energy cost is well documented in [17] for this JPEG-like lossy compression scheme. For this use case, with respect to Table 2 we can satisfy requirements 1 – 5.

Leila Makkaoui et al. [17] propose a fast zonal discrete cosine transform (DCT) based image compression algorithm allowing for trade-off between energy consumption and image fidelity. Using the DCT energy compaction property allows the elimination of high frequency coefficients, hence reducing the number of DCT coefficients to be computed and thus improving energy consumption. Results were simulated based on a MSP430 16-bit microcontroller and an 802.15-compliant CC24020 radio transceiver. Their proposed compression scheme claims improvement of 14% and 18% of energy consumption compared to classical JPEG compression in the transmission of

an 128x128 pixels image. For this use case, with respect to Table 2, we can satisfy requirements 1-5.

Mookeun Ji (et al.) [18] proposed a scheme which breaks an image into small grid patches and only transmits the area of the image which has been deemed to have changed. Suitable for slowly changing scenes such as crop monitoring, an initial full image is sent as reference, and afterwards, only grid patches which have changed are sent. Each 160x160 pixel image is divided into 256 grid patches, each grid consisting of 100 bytes. Transmission of a formatted packet containing x2 patches, plus overhead, is 203 bytes in size. Transmission of the initial reference image took 127.5 seconds which appears to exceed ETSI regulations regarding duty-cycle time limitations. The end-device hardware used could not regarded as a low-power device i.e. Raspberry Pi 3 Model B + Arduino Uno + LoRa transceiver. Whilst showing promise for bandwidth-limited RF systems, the use case application is restrictive. For this use case, with respect to Table 2, we can satisfy requirements 1,2,3,4,7.

Rafeeq AL-HASHEMI et al. [19] proposes a semi-lossless image compression technique using run-length-encoding (RLE). A raw (uncompressed) image of arbitrary size is obtained, and colours are mapped to a vector of values 0 – 255, where each vector element represents a pixel value. The higher frequency values, represented by the lower 4-bits of each decimal value, is discarded and replaced by a value between 0-15. The lower nibble (4-bits) is replaced with a value which indicates the consecutive number of similar colours (upper nibble) after which run-length-encoding is performed on the image vector. The new byte now represents the pixel colour and frequency of occurrence.

The image vector length is reduced thus compressing the data package. This method is computationally simple and can be achieved using a very small algorithm footprint. Compression ratios of between 66.4% and 58.21% have been achieved with typical values for the PSNR of approx. 30dB. Acceptable values for wireless transmission quality loss are considered to be 20 dB to 25dB p.12 [19]. For this use case, with respect to Table 2, we can satisfy requirements 1-6.

3 CHAPTER 3 RF TECHNICAL REVIEW

A review of channel capacity estimation and the effects of noise, modulation type, and bitrate is described in this section. Ease of implementation, cost, and availability of suitable of-the-shelf components in discussed. In the context of assessing the suitability of the chosen private network schemes (ESPNOW, CSS, FSK) for this use case, an evaluation of the suitability of each modulation type is considered.

ETSI

The European Telecommunications Standards Institute produces globally applicable standards for information and communication technologies (ICT). Specifically, they produce harmonised standards including standards related to the use of European frequency bands which are of interest in the context of this thesis. The ultra-high frequency (UHF) band covering a frequency range of 300 MHz to 3 GHz is of particular interest within the designated ISM range, exact frequencies allowable for use with short range devices (SRD) are published [20]. ETSI works closely with national standards organisations (NSO) in each European country. In Ireland it is the Commission for Communications Regulation (ComReg) which publishes documentation pertaining to interface requirements for SRD [21] which define transmission power level restrictions, duty-cycle restrictions, and any other mitigation requirements necessary to allow compliant use of equipment within the country.

For this use case, of particular interest are the allowed transmit power levels and duty-cycle restrictions in Table 3.

Table 3: Regulatory Restrictions

Frequency Band MHz	**Maximum Permitted Radiated Power / Field Strength**	**Mitigation Requirements**
433.05 – 434.79	10 mW ERP	Duty cycle: ≤ 10%
868.0 – 868.6	25 mW ERP	Duty cycle: ≤ 1%, or LBT + AFA
2400 – 2483.5	25 mW EIRP	None

A duty-cycle restriction of 1% means a maximum on-air time of 36 Seconds in one hour. Depending on the amount of data to transmit, this will influence the frequency of transmission allowed for each end-device and base-station if used as a transmitter. Duty-cycle restrictions can be ignored if other mechanisms are implemented (within certain frequency bands) to mitigate the overuse of channel resources, such as listen-before-talk (LBT) and adaptive-frequency-agility (AFA).

- LBT – Before a device transmits it senses a channel to determine if there is activity by measuring the received signal strength (RSSI) for at least 5 msec, if the RSSI is below a set threshold the device can transmit on that channel, if not then a delay of at least 5 msec is implemented and the process is repeated. After a device has transmitted it will not transmit again on the same channel until an off time of 100 msec. has been observed. A further restriction of 100 seconds of transmission within one hour also applies on a spectrum of 200 kHz

within a European context. For equipment with LBT a duty-cycle restriction does not apply p31 [14].

- AFA – Using at least two channels allows more transmission time within a one-hour period, the more channels used the greater the transmission time allowed. An effective duty-cycle calculation based on the number of channels available can be calculated as follows:

Equation 3:Duty Cycle Estimation

$$Duty\ Cycle\ Calculation = (number\ of\ channels * 100)/3600$$

For example, using 5 channels results in an effective duty cycle of 0.138 %.

3.1 Channel Capacity

A channel can be described as the physical link between two transceivers, in this instance it is the link over an air interface. Claude Shannon [22] determined the limits of communication channels with additive white gaussian noise (AWGN). For a bandlimited channel, the maximum possible data rate that can be achieved on a given channel of bandwidth B is shown in Equation 4.

Equation 4:Maximum Data Rate

$$R = B\log_2(1 + S/N)bps$$

R = channel capacity (bits/s)

B = channel bandwidth (Hz)

S = signal strength (watts)

N = noise power (watts)

For all communication systems, channel noise is tied to bandwidth and all objects which have heat emit RF energy in the form of random (Gaussian) noise. Thermal noise can be characterised as frequency independent, noise, i.e., white noise. If the noise power in the system is greater than the power of the desired signal, then we will not be able to observe the desired signal, therefore, noise reduction in an RF system should be maximised. An example of a noise source is a resistor of value R in an environment with temperature T. Because the resistor charges are thermally excited, it is possible to measure a voltage across it. Since thermal noise is a type of white noise, the mean voltage is 0, however the root mean square value (rms) is non-zero and is given by Equation 5.

Equation 5:RMS Noise

$$V_n = \sqrt{4kTBR}$$

In a system with a resistor acting as a noise source with impedance R, the available noise power P_n delivered to a matched load can be derived with elementary circuit theory as in Equation 6. [23]

Equation 6:Noise Power

$$P_n = \left(\frac{v_n}{2}\right)^2 \frac{1}{R} = \frac{V_n^2}{4R} = kTB$$

To characterise the noise in a system, a noise figure is used which is the measurement of degradation in signal-to-noise ratio (SNR), from a point to another in a system. For example, input and output as defined in Equation 7.

Equation 7:Input/Output Noise Power

$$F = \frac{S_1/N_1}{S_2/N_2}$$ expressed in decibels as

Equation 8:Noise Power in dB

$$F(dB) = 10\log_{10}(F)$$

To calculate the amount of emitted noise radiation:

Equation 9:Noise Calculation

$$N = kTB$$

N = noise power (watts)

K = Boltzmann's constant (1.38 x 10^{-23} J/K)

T = system temperature, usually assumed to be 290K

B = channel bandwidth (Hz)

The calculated maximum channel bitrate is a theoretical limit and cannot be reached in practice but can be approached as link level design techniques improve. To analyse the expected performance of a given system this is often applied at the beginning any waveform and link budget analysis. It gives an estimation of the upper bound on the data rate achievable for a certain bandwidth and SNR. Channel impairments occur due to channel variations such as non-white noise and inter-symbol interference (ISI). Determining the capacity limits of time-varying wireless channels with multipath fading and shadowing is challenging. This is determined by the channel rate of change and the ability to track the channel variations [24]. It is clear from Equation 4, that to achieve a given performance level (as defined by bit error rate BER), RF power and bandwidth can be traded.

3.2 Modulation Techniques

The method by which analogue or digital information is converted to RF signals suitable for transmission is a key element for consideration in system design. The method chosen determines system bandwidth, power efficiency, sensitivity, and complexity [25]. Figure 1 summarises the theoretical BER for various linear modulations. [26]

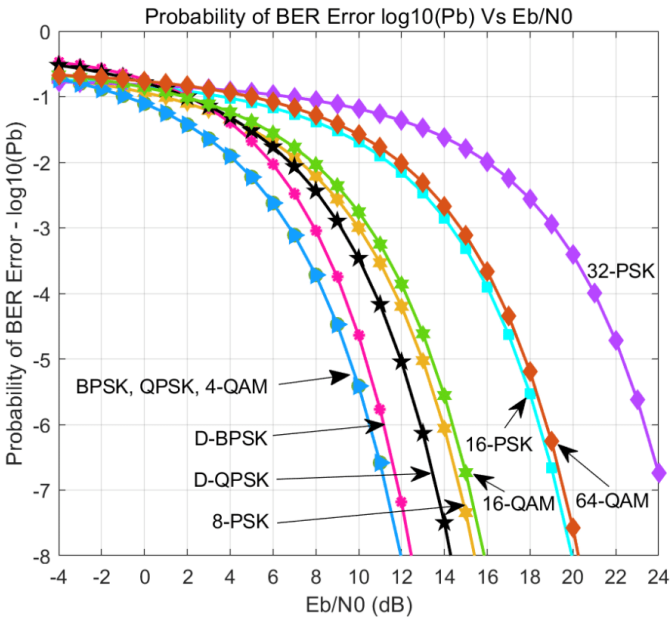


Figure 1:Probability of BER

$\frac{E_b}{N_o}$ is a measure of required energy per bit relative to the noise power and is independent of system data-rate.

To convert to SNR, the system bandwidth must be considered.

Equation 10:SNR

$$SNR = \left(\frac{E_b}{N_o}\right) * \left(\frac{R}{B_t}\right)$$

E_b = Energy required per bit of information

N_o = Thermal noise in 1Hz of bandwidth

R = System data-rate

B_t = System bandwidth

For a given BER, the more complex the modulation scheme the more energy/bit or reduction in noise level is required to achieve comparable performance levels with respect to less complex modulation schemes. Bandwidth requirements for various modulation methods are shown in Table 4.

Table 4: Bandwidth Requirements

MODULATION METHOD	***TYPICAL BANDWIDTH (NULL – TO – NULL)***
QPSK, DQPSK	1.0 x Bit Rate
MSK	1.5 x Bit Rate
BPSK, DBPSK, OFSK	2.0 x Bit Rate

DSSS

Direct sequence spread spectrum technique is one of a range of techniques in which the transmitted signal is spread over a wide frequency band, larger than necessary to transmit the information being sent. For example, a baseband signal (say a voice signal) with a bandwidth of only a few kilohertz is distributed over a band which may be several megahertz wide. This is achieved by modulating the signal to be sent with a wideband encoding (chips) signal as shown in Figure 2 [25]

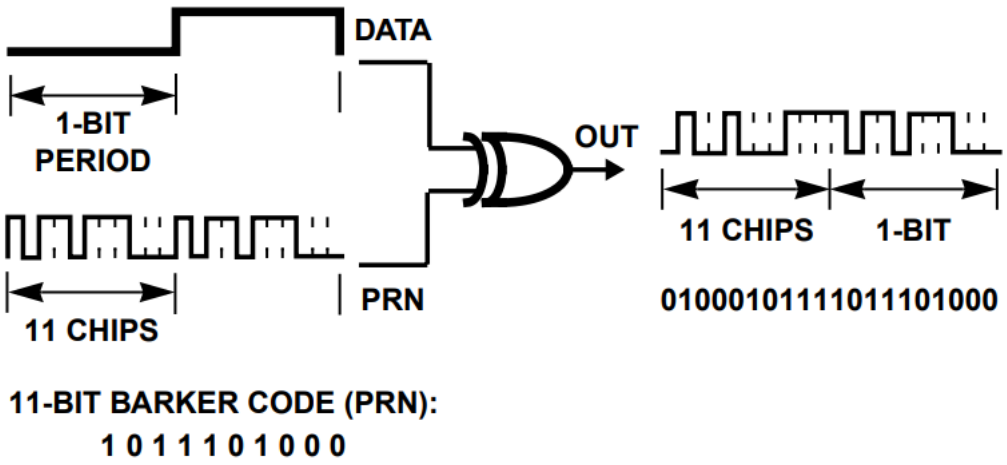


Figure 2:PRN

The pseudo random code (PRN) is a repeatable code and has predictable statistical qualities allowing recovery of the transmitted signal at the receiving end using cross correlation techniques. This allows the transmission of multiple signals using the same frequency and at the same time. Because the wideband signal spectra are generated by code modulation, the power transmitted is low in any narrow region relative to conventional signals. In these types of signals, all the transmitted power is concentrated within a band of frequencies relative to the baseband information bandwidth [27]. The de-spreading of the wideband signal at the receiver end results in a quantity called “process gain”.

In spread spectrum systems, the process gain available may be estimated by the rule of thumb: [27] Equation 11.

Equation 11:Process Gain

$$process\ gain = \ G_p = \frac{BW_{RF}}{R_{info}}$$

BW_{RF} = Bandwidth of the transmitted spread spectrum signal.

R_{info} = Data rate in the baseband information channel.

Using Equation 10 and changing bases we can see that.

$$\frac{R}{B} = 1.44 \log_e \left(1 + \frac{S}{N}\right)$$

For small $\frac{S}{N}$, (<0.1),

$$\frac{R}{B} = 1.44 \frac{S}{N}$$

From this equation we find that,

$$\frac{N}{S} = \frac{1.44B}{R} \approx \frac{B}{R} \quad \text{, and}$$

Equation 12:Spread Spectrum Bandwidth

$$B = \frac{NR}{S}$$

This shows that for any given noise-to-signal ratio, we can have a low information error rate by increasing the bandwidth used to transfer the information. For example, if we want the system to operate in a link where the interfering noise is 30 times greater than the signal strength and our information rate is 1Mbps, then we need a bandwidth of:

$$B = \frac{30\ x\ 10^6}{1.44} = 20.833\ X\ 10^6 Hz$$

CSS

A type of spread spectrum technology used by LoRa™ is a pulsed FM (CHIRP) type system called Chirp Spread Spectrum (CSS). Because a wider bandwidth is used other than the minimum required, processing gain (Equation 11) can be achieved. This modulation type is used in RADAR type applications and has recently been used in communications systems [28]. Chirp transmissions are characterised by pulsed RF signals whose frequency varies in some known way during transmission. In this instance, the frequency of the base chirps increases linearly (up chirp) Figure 3 [29] from an initial value $f_0 = -BW/2$ to a final value $f_1 = BW/2$. In the LoRa™ embodiment, the modulator is also arranged to synthesise and insert in the signal conjugate chirps. These chirps are the complex-conjugate of the base unmodulated chirp and can be regarded as down-chirps Figure 4 [29],in which the frequency falls from a value of $f_0 = +BW/2$ to $f_1 = -BW/2$.

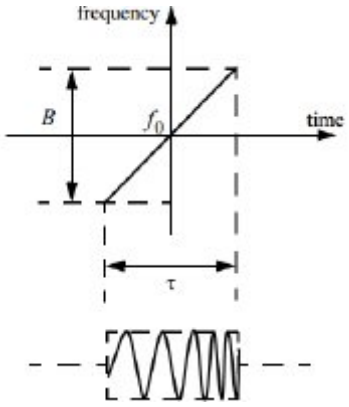


Figure 3:Up Chirp

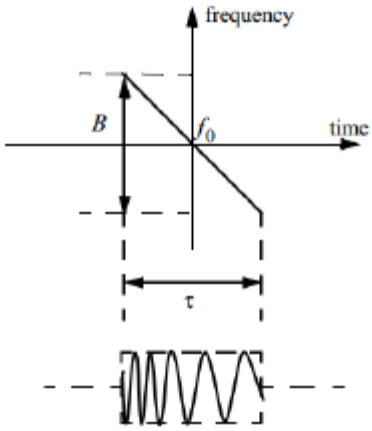


Figure 4:Down Chirp

Whatever sweep pattern is used to transmit requires an equivalent receive filter to be built. The input to the chirp filter using a linearly swept signal is shown in Equation 13.

Letting $\mu = \frac{d_\omega}{d_t}$ for the frequency sweep rate, we see that

$$F(t) = f_1 + \mu t$$

Equation 13:Linear Swept Signal

$$= A\cos(\omega_1 t + \mu t)$$

The filter at the heart of a chirp receiver is a storage and summing device that accumulates the energy received over an interval, assembles it, and releases it in one coherent burst [27]. Given the added circuit complexity required to implement a spread spectrum type network, the benefits include: [27]

1. Selective addressing capability.
2. Code division multiplexing is possible for multiple access.
3. Low-density power spectra for signal hiding
4. Message screening from eavesdroppers.

5. High-resolution ranging

6. Interference rejection.

In this use case, the added range achieved for a given transmit power satisfies one of the use case requirements.

FSK

Frequency shift keying (FSK) as specified by the DASH7 Alliance for the physical interface, is a modulation method where digital information is transmitted through the discrete frequency change of a carrier wave Figure 5. A defining parameter for FSK is the modulation index Equation 14.

Equation 14: Modulation Index

$$m = \frac{f_2 - f_1}{R}$$

Where f_2 and f_1 are the high and low frequencies on both sides of the carrier frequency and R is the maximum data rate.

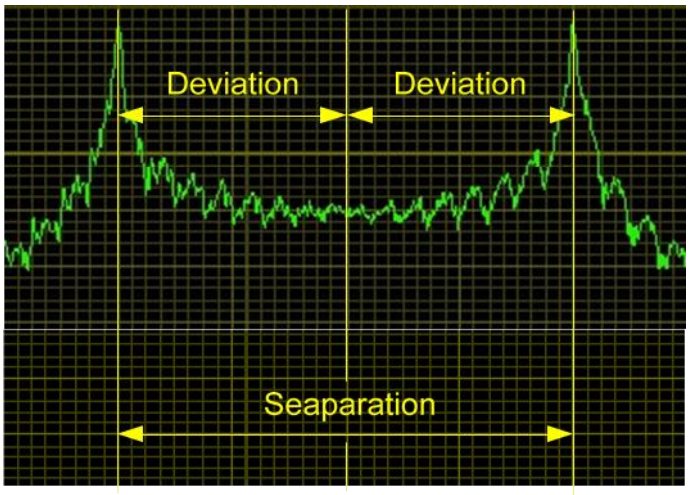


Figure 5:FSK [30]

Abrupt changes in data levels are discontinuities in a continuous signal and manifest as a widening of the spectrum larger than $f_2 - f_1$. To reduce this effect a gaussian filter can be applied in the data line before the FSK modulator in the transmitter. This type of FSK is called GFSK. The gaussian filter has the effect of limiting the rise time of the data signal and is characterised by the product of its bandwidth *B* at -3dB point times the bit period *T*, that is, *BT*. By adjusting the modulation index *m* and *BT*, it is possible to control the radiated spectrum within a defined span for a specified data rate. For a coherent detector (when there is phase synchronisation between transmitter and receiver), improved spectral efficiency is achieved when the mark and space frequencies are orthogonal to each other i.e., when *m* = 0.5. This implies that the difference between mark and space frequencies is one-half of the data rate and is called minimum shift keying (MSK). The form used in D7A is 2-(G) FSK with three different modulation indices specified for various data rates as shown in Table 5. [15]

Table 5: FSK Modulation index

Channel Class	*Channel Spacing (MHz)*	*Modulation*	*Symbol Rate (kbps)*	*Modulation Index*	*Symbol 0 kHz*	*Symbol 1 kHz*
Lo-Rate	0.025	2-(G)FSK	9.6	1	F (B, I) -4800	F (B, I) + 4800
Normal	0.200	2-(G)FSK	55.555	1.8	F (B, I) -50.000	F (B, I) +50.000
Hi-Rate	0.200	2-(G)FSK	166.667	0.5	F (B, I) -41.667	F (B, I) +41.667

FSK is a low complexity type modulation scheme enabling robust communication systems to be developed using low-cost hardware. The Semtech™ range of transceiver ICs (SX127x) can also operate in FSK mode making them suitable for dual use i.e., CSS or FSK; hence satisfying some of the parameters for this use case.

3.3 Link Budget Analysis

One of the first actions taken by an RF engineer in designing a radio system suitable for a particular application is an analysis of the environment in which the system will operate. This includes the desired data-rate, the acceptable error-rate, and the expected channel losses. The data-rate expected will be determined by the type of information needed to be delivered and the use case will place an upper bound on this requirement. For example, the transmission of large amounts of data, such a video, have different requirements compared with the transmission of a few bytes of information per hour. The limitations imposed by the selection of frequency band will also influence usable bandwidth and hence data-rate to be expected. These limitations also have an impact on power consumption of end devices. A link budget calculation allows us to estimate the reliability of a system with given assumptions. A link budget accounts for all the power gains and losses that a communication signal experiences within a telecommunication system.

It accounts for signal attenuation due to propagation, antenna and cable gains and losses as shown in Figure 6. [31].

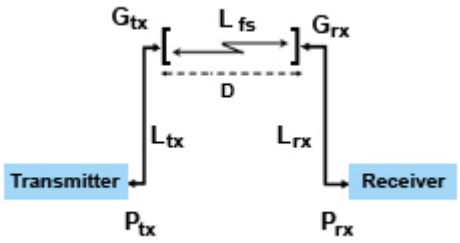


Figure 6:Link Budget

A link budget can be calculated as shown in Equation 15

Equation 15: Link Budget Calculation

$$P_{RX} = P_{TX} + G_{TX} - L_{TX} - L_{FS} - L_M + G_{RX} - L_{RX}$$

Where,

P_{RX} = Received Power (dBm)

P_{TX} = Transmitter Power Output (dBm)

G_{TX} = Transmitter Antenna Gain (dBi)

L_{TX} = Losses from Transmitter (cable, connectors etc.) (dB)

L_{FS} = Free-Space Loss (dB)

L_M = Misc. Losses (fade margin, polarization misalignment etc.) (dB)

G_{RX} = Receiver Antenna Gain (dBi)

L_{RX} = Losses from Receiver (cable, connectors etc.) (dB)

Free space loss can be calculated for each frequency of interest propagating through air as shown in Equation 16. This is the Friis [32] transmission equation for free space propagation.

Equation 16:Free Space Losses

$$L_{FS} = 20\log_{10}(d) + 20\log_{10}(f) + 20\log_{10}\left(\frac{4\pi}{c}\right) - G_{Tx} - G_{Rx}$$

Where -

d = Distance between the antennas.

f = Frequency

G (Tx) = The Gain of the Transmitting Antenna.

G (Rx) = The Gain of the Receiving Antenna.

c = Speed of light in vacuum (Meters per Second)

Calculating for this use case, where communication is required to over a distance of 100 to 5000m, we can define the required receiver sensitivity necessary for a given BER and assess the expected reliability of the network design.

2.4GHz (ESPNOW)

Assuming 0dBi gain for both transmit and receive antennas and 100m between transmitter and receiver with 0dB losses in connecting cables.

Free space losses:

$$L_{FS} = 20\log_{10}(100) + 20\log_{10}(2.4x\,10^{9}) + 20\log_{10}\left(\frac{4\pi}{c}\right) - 0 - 0$$

$$L_{FS} = 40 + 187.6 - 147.55$$

$$L_{FS} = 80.04\text{dB}$$

Power at the receiver:

$$P_{Rx} = P_{Tx} + G_{Rx} - L_{Tx} - L_{FS} - L_{M} + G_{Rx} - L_{Rx}$$

$$P_{Rx} = 14 + 0 - 0 - 80.04 - 0 + 0 - 0$$

$$P_{Rx} = -66dBm$$

If we include a further loss due to fade margin, cable losses etc. of 30dB, the required power at the receiver is -96dBm. The system sensitivity of a properly matched ESP32 operating in 802.11B 1Mbps mode is: -97 dBm [33]. So, our design parameter for range using ESPNOW is easily met. When initial communications are established, a common bitrate is negotiated [34] and will default to 1Mbps unless LR (Long Range) mode is enabled on the ESP32, and other bitrates fail to be negotiated. A link is established with a physical throughput of 125 kbps and an improved sensitivity of -105 dBm [33]. This is a proprietary communications mode and is applicable to Espressif™ only, a link range of 1km is claimed when using this mode due to a 4 dB gain in sensitivity.

433MHz (DASH7)

D7A defines several Sub-GHz ISM bands for use with this protocol [15]. Using 433 MHz band and assuming 100m between transmitter and receiver and no additional loss, the free space line of sight loss is:

$$L_{FS} = 20\log_{10}(100) + 20\log_{10}(433x\,10^{6}) + 20\log_{10}\left(\frac{4\pi}{c}\right) - 0 - 0$$

$$L_{FS} = 40 + 172.7 - 147.55$$

$$L_{FS} = 65.15\text{dB}$$

If a 3dB increase in power gives a doubling of range, we get a X5 times reduction in propagation loss using 433MHz band. Because D7A is

manufacturer agnostic for the physical layer, a typical sensitivity of -104 dBm, as a function of data rate (9.6 kbaud), with 20kHz frequency separation and BER = 10^{-3}, can be expected [35]. This is equivalent to the Lo-Rate channel class specified by D7A. The power expected at the receiver is:

$$P_{Rx} = P_{Tx} + G_{Rx} - L_{Tx} - L_{FS} - L_M + G_{Rx} - L_{Rx}$$

$$P_{Rx} = -51.5dBm$$

Including additional loss of 30 dB for cable loss, fade margin etc. we can expect:

$$P_{Rx} = -81.5dBm$$

Increasing the distance between Tx and Rx to 2000m, we get a receiver power of $P_{Rx} = -107dBm$ which is outside the stated sensitivity for the given parameters, so an expected range of 1000m – 1500m should be achievable.

868 MHz (LoRa®)

Using CSS as a modulation scheme allows robust and long-range communications systems to be realised and has been used in military and space communications systems for decades. The recent availability of low-cost hardware to implement this type of modulation scheme allows commercial development suitable for IoT type systems. The trade-off is the reduced data-rate achievable for the longest range possible. For this use case, the ISM frequency band selected is 868MHz with an allowable transmit power of +14dBm and a maximum achievable link budget of 157dB [36]. If we define the distance between the transmitter and receiver as 5000m, with no additional loss, we can calculate the expected receive power at the receiver:

$$L_{FS} = 20\log_{10}(5000) + 20\log_{10}(868x\,10^{6}) + 20\log_{10}\left(\frac{4\pi}{c}\right) - 0 - 0$$

$$L_{FS} = 73.97 + 178.77 - 147.55$$

$$L_{FS} = 105.19\text{dB}$$

Power at the receiver is

$$P_{Rx} = P_{Tx} + G_{Rx} - L_{Tx} - L_{FS} - L_{M} + G_{Rx} - L_{Rx}$$

$$P_{Rx} = -91.19dBm$$

If we include an additional 30dB of loss due to cable loss, link margin etc.

$$P_{Rx} = -121.19dBm$$

This received power is well within the maximum link budget allowable and so the range requirement for this use case is satisfied.

In summary see Table 6.

Table 6: Use Case Range Requirements.

Modulation type	DSSS(ESPNOW®)	FSK(D7A®)	CSS(LoRa®)	Range Requirement Satisfied?
Range	≤1000m	≤1200m	≤ 5000m	✓
Tx Power	14dBm	10dBm	14dBm	✓
ISM Band	2400MHz	433MHz	868MHz	✓

The other requirements necessary to achieve a reliable communications network such as collision avoidance and mitigation, error correction and security, bi-directionality, Fresnel zone radius and network topology all have an impact on the performance of a particular system architecture. Although the physical link-budget analysis has been assessed for several modulation schemes, the expected reliability and quality of a communications link will

be affected by other influences mentioned and should be allowed for in designing a system for a given expected reliability and quality of service.

3.4 RF Security

Secure transmission of data in any RF network is a broad topic and will be discussed here in the context of the three types of modulation schemes previously discussed and their inherent security capability.

ESPNOW

This is a fast connectionless communication technology using cipher block changing message authentication code protocol (CCMP) as the encryption method. This is designed for wireless LAN products and is an amendment to the original IEE802.11 standard. This provides data confidentiality, authentication and access control in conjunction with layer management. Two key types are used, private master key (PMK) and local master key (LMK). The PMK is used to encrypt the LMK with AES128 type encryption. The LMK of the paired device is used to encrypt the action frames with the CCMP method.

LoRa

Because LoRa defines the physical layer only, there are no inherent security capabilities defined within the device. Security is dealt with at the upper OSI layers where encryption and authentication are managed e.g., LoRaWAN.

DASH7

Of the three network types discussed, DASH7 is the only completely defined OSI stack consisting of a wireless air interface and system stack. Apart from

the channel coding techniques used at the physical layer (PN9,1/2 convolutional coding) network security (authentication and encryption) is defined at the network layer (NWL3) and the application layer programming interface (ALP) layers 6&7. The payload of D7ANP frames can be encrypted and authenticated using one of seven different methods as outline in the D7A specification V1.2, Table 7.4.1.1. D7ANP accepts only frames encrypted and/or authenticated in one of the active network layer security (NLS) methods defined in the device capacity file. D7A default permissions depend on where the interface ALP commands originate from. User permissions are granted to devices connected via a wired interface. Over-the-air commands have guest permissions, and the device application has root permission.

3.5 Chapter Summary

A review of currently available RF solutions operating in the IoT (Internet of Things) space was undertaken here. A focus was placed on private/public IoT type networks, although a review of CIoT network offerings was included for completeness. The decision to focus on private/public networks was made based on the availability of OTS hardware and product support. CIoT offerings are included as part of the 5G improvements as specified by 3GPP group and include NB-IoT and M1. Currently, only NB-IoT is available in Ireland. To develop a compression algorithm suitable for use across multiple RF platforms a specification was produced shown in Table 2. The range specification was set to a maximum reception range of 5km, and range calculations were assessed for three different RF network types using

different modulation schemes. The same device should be capable of using any of these schemes with the appropriate hardware attached. The three modulation schemes assessed were FSK (DASH7), CSS(LoRa) and DSSS(ESP-NOW). The commonality across these three schemes is the maximum payload size of 255Bytes/transmission. Each scheme can have different transmission bitrates and were evaluated for suitability regarding range capability and hence time-on-air which effects power consumption. A brief discussion on security within the RF space was included confined to the inherent capabilities of each type of RF network explored.

4 CHAPTER 4 SYSTEM TESTBED DESIGN

The system needed to prove the validity of the developed image compression and RF transmission technique consists of three main elements.

- Hardware
- Firmware
- Software

4.1 Hardware

There is limited information available on the transmission of still images using OTS components in RF bandlimited systems. One such implementation, which develops a system from ‘scratch’ uses low-power 8-bit Arduino type microcontrollers which is connected via a serial link to a camera and Semtech® transceiver [16]. This was the only complete end-to-end system found which addressed the regulatory constraints imposed and used LoRa® physical layer in CSS mode. A more flexible platform was needed which allowed the possibility of using all three communication types (ESPNOW, D7A, LoRa) on the one platform. An initial network topography was chosen for simplicity of implementation, performance diagnoses and power consumption. A mesh type network was considered using D7A but was found difficult to test and implement for this use case.

A star type network was eventually chosen for the same reasons, with the ability to tailor the modulation type to achieve the use case requirements as shown in Figure 7.

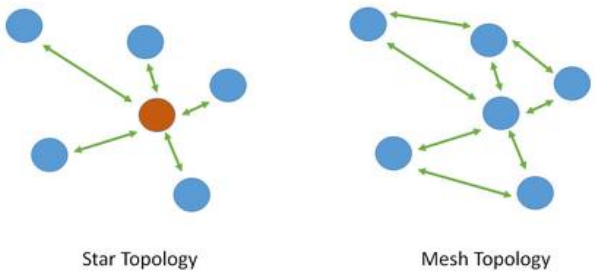


Figure 7: Network Topologies

The topography of the system is shown in **Error! Reference source not found.**.

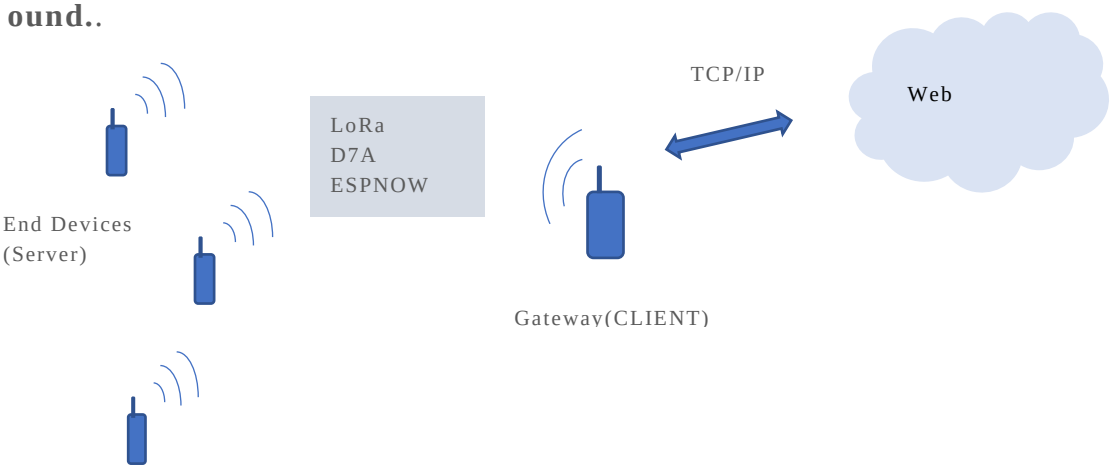


Figure 8:System Topography

The microcontroller platform selected (ESP32 DevKit_V4) by Espressif™ systems satisfied the testbed requirements for the end-node device and gateway device listed in Table 7.

Table 7: Hardware Requirements

Connectivity	UART	SPI	802.11	Bluetooth	D I/O	A I/O
ESP32 Devkit	✓	✓	✓	✓	✓	✓

This configuration of the end-node device and gateway shown below in Figure 9 allowed connection of a LoRa® transceiver via a serial peripheral interface (SPI) and a camera, via a universal asynchronous receiver transmitter (UART). This arrangement allowed selection of the preferred modulation type. The camera selected for this configuration was uCam-II (hardware rev.1.0) by 4D Systems which allowed serial connectivity and access to raw image data if required. The LoRa® transceiver used is an RFM95 module by Adafruit Industries connected via SPI.

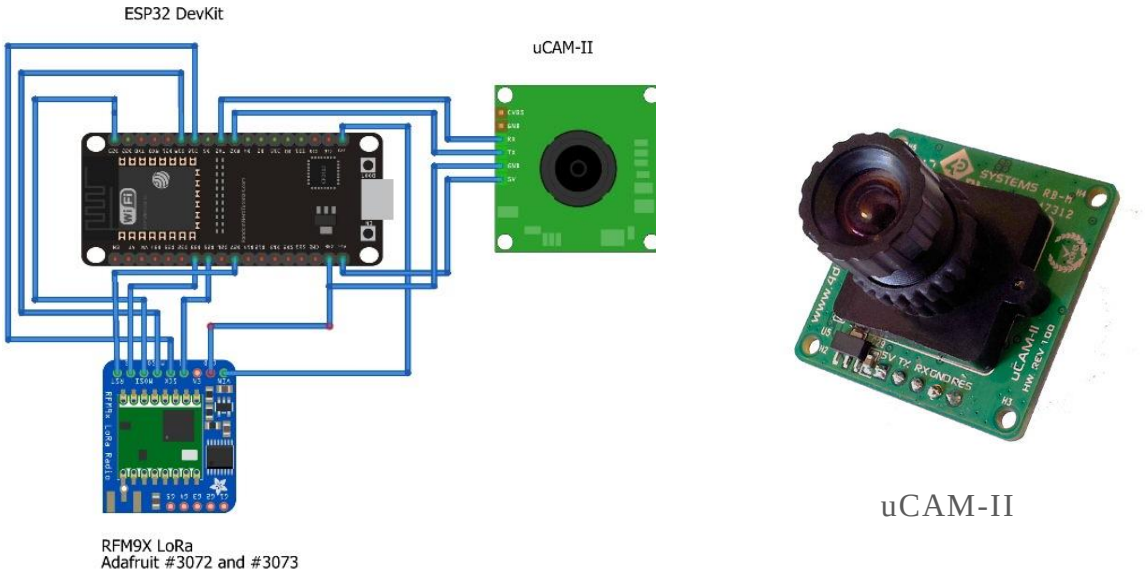


Figure 9:End Device (Server)

The end-device (Client) configuration in Figure 9 allowed selection of the modulation scheme (FSK, CSS,802.11) for various use cases. The Gateway configuration is shown here in Figure 10. Connectivity is achieved via USB from the development PC.

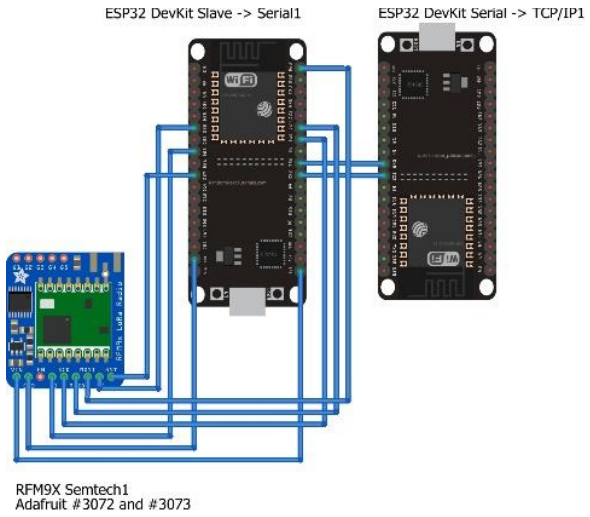


Figure 10:Gateway Hardware (Client)

The hardware chosen for development of the image compression scheme differed slightly because of an issue with unreliable communication synchronisation action on the uCam-II which caused unnecessary development delays [37]. This issue appears to have been addressed in later hardware revisions of the camera. The chosen version of the test bed platform used a different ESP32 platform for end-devices and 802.11(ESP NOW)

modulation scheme is chosen for algorithm development purposes. The testbed setup for the development work is shown in Figure 11.

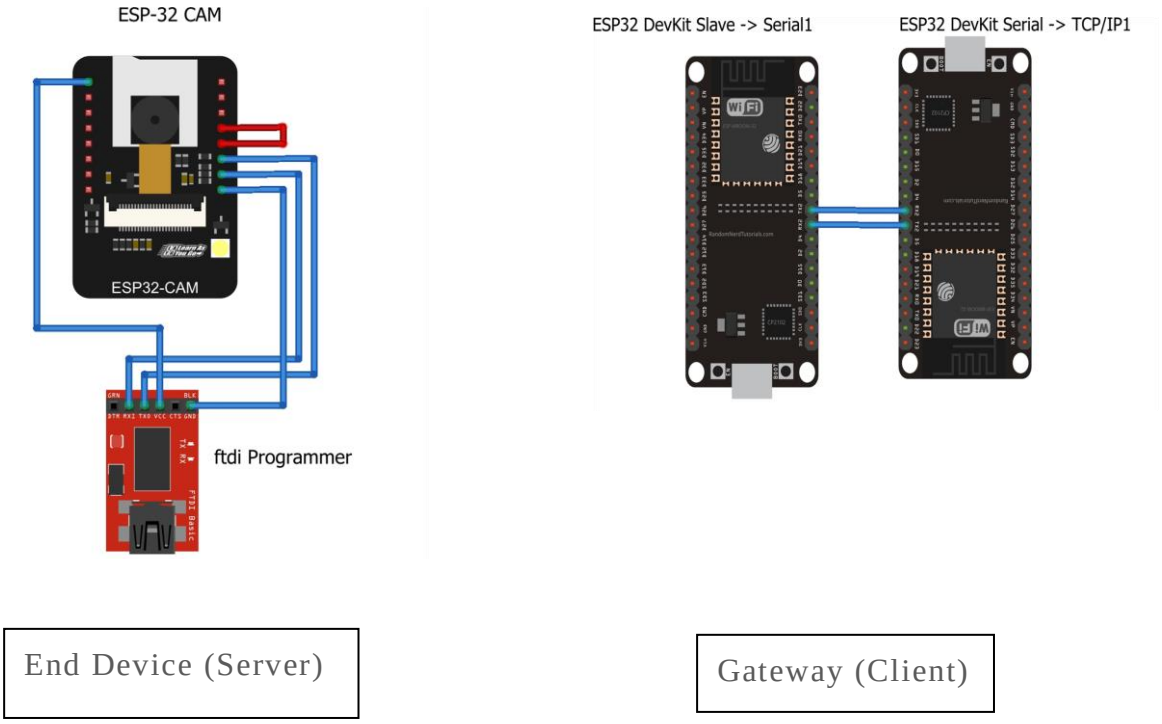


Figure 11:Test Bed Hardware

The micro-controller module chosen for the end-device (Server) is a variant of the ESP32 as shown in Figure 12.

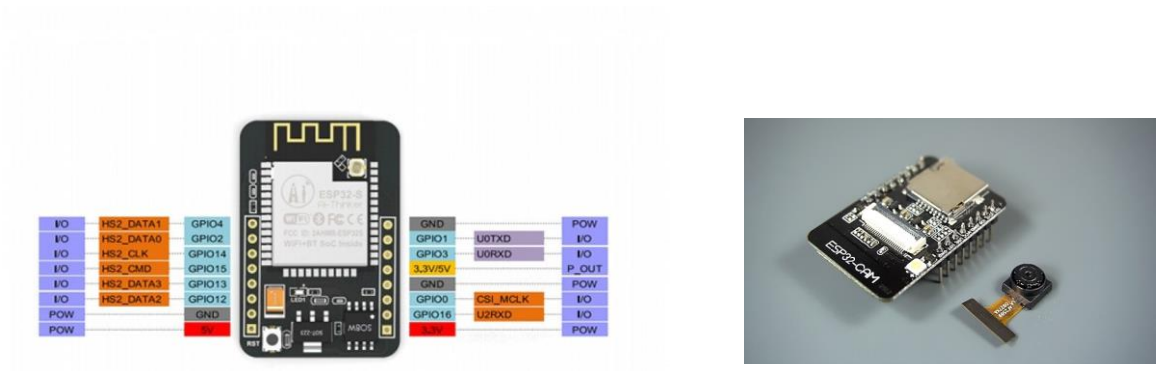


Figure 12:ESPCAM

This device has an OV2640 camera connection port included in a very small form factor.

The Camera type included with this module has a resolution of UXGA, SVGA and below, and supports JPEG compression and RAW data modes. A version of this camera is available as an SPI connected module allowing flexibility in design for use with different microcontrollers [38].

The Firmware and Software for this project was developed on a desktop PC running Linux operating system (Ubuntu 18.04.5 LTS). The ESP32 DevKit devices used for the gateway were connected to the development PC via USB cable. To allow programming of the end-devices (ESP-32 CAM), a FTDI device was connected, and a temporary link was placed between pin 10 and ground for programming. To allow monitoring and debugging of the device, this link was then removed for normal operation. The FTDI is a high-speed serial communications device which bridged connectivity between TTL serial transmissions and USB signals. Because the ESP32 DevKit has on-board USB connectivity this device was not needed.

SDR (Software Defined Radio)

To enable verification of RF Transmission below 1GHz, a SDR was used to visualise communications between end-devices and the gateway. The device used was a NooElec NESDR Mini connected via USB as shown in Figure 13. This is a modified DVB-T USB dongle tuned for SDR usage within a range of 25MHz – 1750MHz. The visualisation software used was SDR# (SDR sharp) running on Windows 10, which allowed a FFT and waterfall display. Installation instructions can be found here [39].

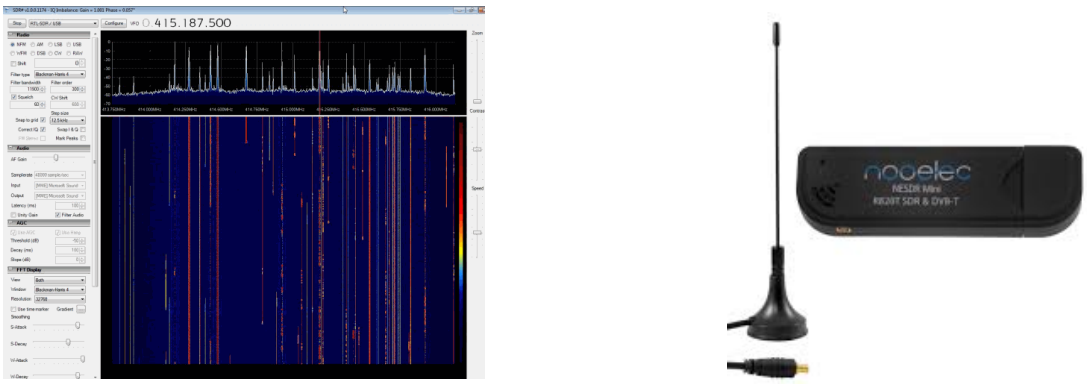


Figure 13:SDR

4.2 Firmware Development Environment

The chosen programming language to develop the necessary firmware was C++. This was selected because it allows direct access and control of each micro-controller subcomponent. Direct control of timing elements within the development of any RF system is necessary to ensure reliability and repeatability of any developed protocol. A mature eco-system exists for the development of code for micro-controllers using C++ which facilitated tweaking of compilation parameters and enabling an efficient code footprint to be developed. The Integrated development environment (IDE) chosen for this project was PlatformIO. This permitted cross-platform development of code using Microsoft Visual Studio Code. Full debugging was enabled with single-stepping and multiple breakpoints setting. Using an external high speed serial device, connected to the micro-controller under development allowed for full hardware and software debugging. Such a device is the FT2232H Mini Module available from ftdi Ltd. To integrate this device into PlatformIO, and to get it to communicate correctly proved difficult. Details of the final connectivity and setup was recorded.

FTDI

A drawback of using a high-speed serial device for debugging is the additional I/O pins required on the device under test (DUT) for connectivity. These connection pins might be required for other device connectivity such as a camera, so limited device debugging might only be achievable. Nevertheless, this is a superb low-cost solution allowing most of the functionality of a full device emulator. The debugger protocol is built on an industry standard called JTAG and the Mini Module is supported by OpenOCD [40]. This enables PlatformIO to communicate with it via a downloadable software driver. Initial debugging of this system was achieved using such a device and was connected as shown in Figure 14.

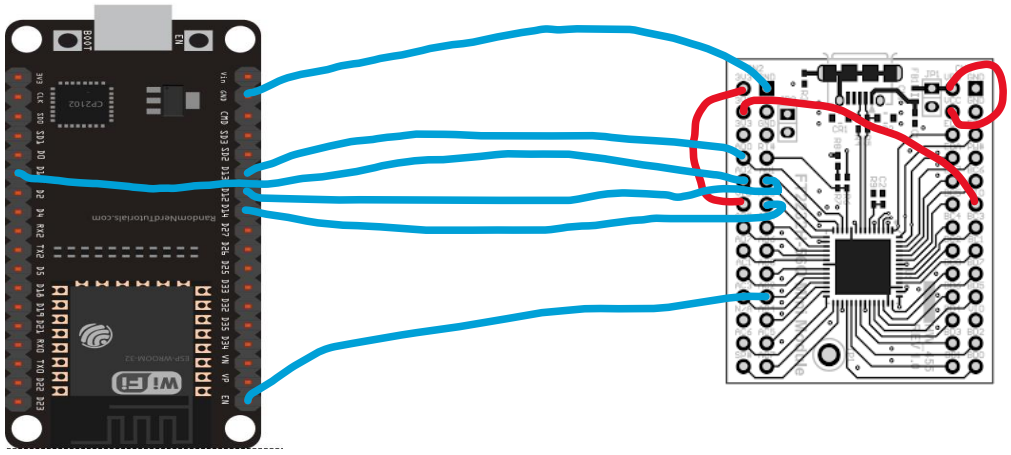


Figure 14:FTDI Connection

A full table of connections is found in Table 8

Table 8: FTDI to ESP32 Connection

FT2232H	FT2232H	ESP32 DevKit	JTAG name	FT2232H pin
CN2-7		GPIO 13	TCK	AD0
CN2-10		GPIO 12	TDI	AD1
CN2-9		GPIO15	TDO	AD2
CN2-12		GPIO 14	TMS	AD3
CN2-1	CN2-11			
CN2-3	CN3-12			
CN3-1	CN3-3			

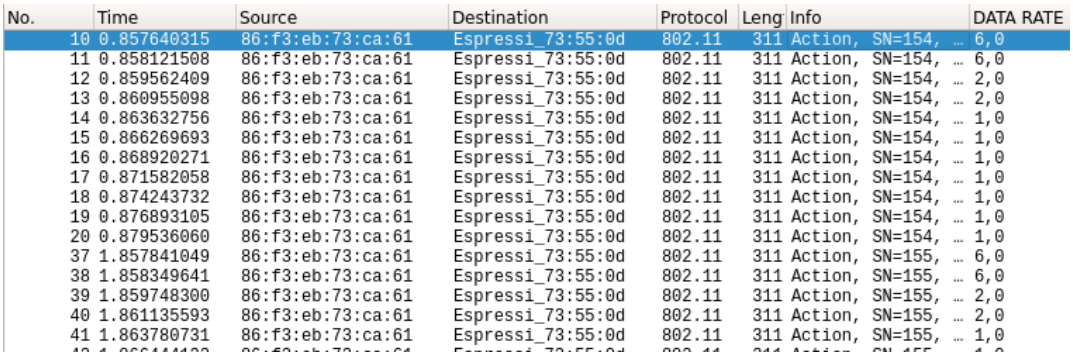
The ESP32 and the FTDI were connected to the development PC via USB where dual terminals were opened for debugging and monitoring of the application under development.

JTAG Driver:

To communicate with the FTDI, some software configuration was necessary when using Windows. However, all FTDI devices are supported in Ubuntu as standard. When using Windows, it was necessary to download a virtual COM port driver (VCP) [41] which caused the USB device to appear as an additional COM port. It was also necessary to install a USB driver configuration tool such as ‘Zadig’ [42]. There were also some configuration settings necessary in PlatformIO project configuration file (platformio.ini). To use this FTDI required setting the debug tool = minimodule [43]. This configuration permitted the setting of two hardware breakpoints and multiple software breakpoints.

4.3 Software

The software element of this research consists of the development of several Java based programs designed to run on a PC under a Linux environment. These set of programs enabled the remote connectivity of the gateway to be established using web. Sockets. The protocol used could be either TCP/IP or UDP on port 3000. Java was chosen because of its ubiquity and ease of implementation within the Processing [44] programming environment. This environment allowed easy development of visually orientated applications and was thus suited to this application. The monitoring of 802.11 traffic was achieved using WIRESHARK [45] which is a network protocol analyser. A typical packet capture is shown in Figure 15.



No.	Time	Source	Destination	Protocol	Leng	Info	DATA RATE
10	0.857640315	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	6,0
11	0.858121508	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	6,0
12	0.859562409	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	2,0
13	0.860955098	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	2,0
14	0.863632756	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
15	0.866269693	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
16	0.868920271	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
17	0.871582058	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
18	0.874243732	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
19	0.876893105	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
20	0.879536060	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=154, …	1,0
37	1.857841049	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	6,0
38	1.858349641	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	6,0
39	1.859748300	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	2,0
40	1.861135593	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	2,0
41	1.863780731	86:f3:eb:73:ca:61	Espressi_73:55:0d	802.11	311	Action, SN=155, …	1,0

Figure 15:Wireshark packet capture

The final test bed layout is shown in Figure 16.

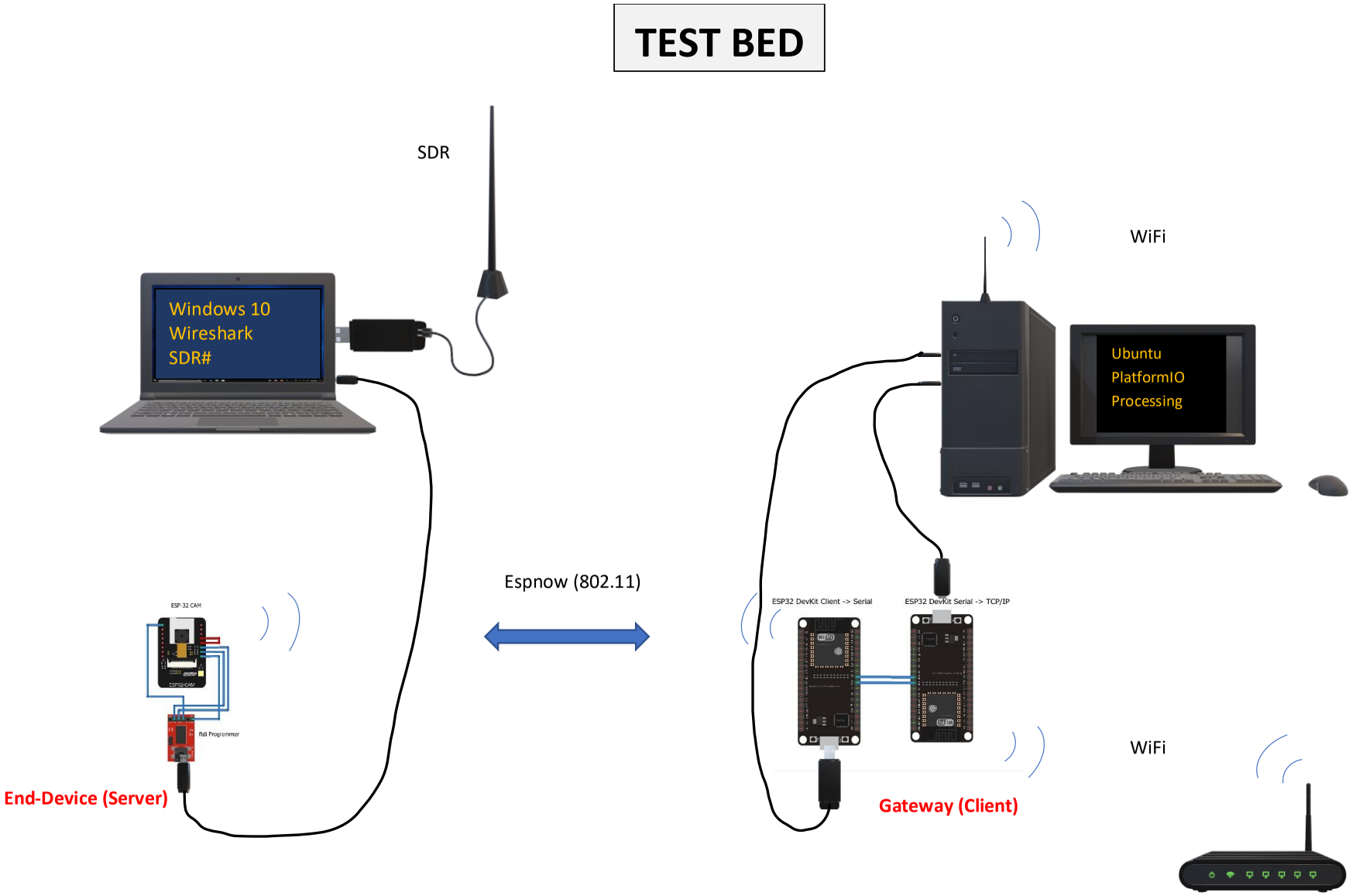


Figure 16:TEST BED Layout

4.4 Chapter Summary

This chapter described in detail the testbed designed to enable end-to-end testing and evaluation of both RF networks and the image compression algorithms developed in this research. It included a discussion of three key components: hardware, integrated development environment and the software.

The chosen hardware was selected specifically because of the compact integration of the necessary interfaces required. The hardware (ESP32/ESP32-CAM) consisted of an integrated camera, Wi-Fi, BLE and extensive I/O. The processor was a dual core, 32bit device running FreeRTOS and at low cost. These microcontroller devices were developed by Espressif™ and are the only devices capable of running ESP-NOW protocol. Although not the most low-power device available, it was selected because of the availability of on-board resources. This allowed development of suitable algorithms, and the assessment of resources needed for a basic system design. The integrated development environment (IDE) selected was suitable for both Windows and Linux operating systems and used Visual Studio for code development in C++. This environment allowed software development within the Arduino framework and was in keeping with a desire to maintain simplicity and reduce complexity, to enable cross development for lower power devices such as 8-bit AVR microcontrollers. To ‘visualise’ RF transmissions in the sub-GHz frequency bands when using LoRa or DASH7, a software defined radio (SDR) was used with AirSpy software running on a Linux (UBUNTU) machine. For real-time testing and register watching / breakpoint setting, a highspeed UART device was connected to the DUT

(Device under test) via a JTAG interface. Driver software is freely available for these devices and integrates with the PlatformIO IDE. Whilst RF networks are difficult to debug in real-time, the combination of an SDR and Hardware debugger allow most issues to be detected and resolved. Several testbed iterations were developed, and the final version is shown in Figure 16. Packet capture over the air for ESP-NOW was achieved using a freely available packet analysis software ‘Wireshark’ in association with a Wi-Fi adapter which is capable of being set to promiscuous mode.

5 CHAPTER 5 SYSTEM IMPLEMENTATION

This chapter describes the implementation of a low-power WLAN suitable for use with a developed image compression algorithm. The chosen RF topology was a star type network based on an ESP32 type microcontroller platform which is widely available. The chosen platform allowed the development of a compression technique suitable for use on a range of micro-controller platforms. This facilitated easy embedding of the algorithm within an existing program because of the minimal computational cost and memory requirements. The test bed used includes the use of ESPCAM (ESP32 Derivative) for the end-device and was chosen for its integrated camera and WiFi transceiver which required minimum system wiring. The gateway configuration chosen used two ESP32 DevKits (ESP32 Derivative) but could be replaced by any capable system which satisfied the requirements, such as a Raspberry Pi (RPi). I used two ESP32s because it allowed a single use testbed environment for both the end-device and gateway development without changing IDE. The selected modulation scheme was DSSS (ESPNOW) using the embedded WiFi capabilities of the chosen platform. ESPNOW has been previously described in section 2.3, this section describes the IDE configuration and firmware development of a working solution for both the RF network and the image compression algorithm.

5.1 WLAN implementation

This research explored various topographical type RF network structures. To arrive at a suitable network configuration, it was felt that these configurations should be explored in a practical manner to assess their suitability for the chosen use case. The two network types selected included an implementation of an image transmission system developed by C. Pham [16] but ported for use on an ESP32 type platform. This implementation uses a LoRa® connected transceiver (Semtech) and operates in the sub1GHz ISM frequency bands. The other network topology implemented was a mesh type network using the same LoRA® transceiver as used by C. Pham with the ability to switch between FSK and CSS type modulation. This implementation allowed a dual data-rate system to be selected depending on the use case requirements and was implemented on ESP32 Devkit platform. Both topologies were assessed based on the RF requirements for this application before a suitable image compression algorithm was implemented.

Star Topology

To gain experience with using and transmitting image data using resource-constrained devices, it was decided to implement a suitable existing solution. A solution developed by C. Pham [16] enabled he transmission of 128x128 byte image (Raw size of 16384bytes). The image was then compressed using a modified DCT developed in collaboration with Vincent LECUIRE (CRAN UMR 7039, Nancy-Universite, France). This method limited the image shape to the same number of Rows x Columns and did not meet the requirements for this use case. This method was ported for use on the selected hardware used

during this research. The limited image size, lack of ability to select a ROI and the use of a third-party compression technique restricted the use of this system for this application. However, the system was successfully implemented using the hardware shown in Figure 9 for the end-device and using a RPi as a gateway as shown in Figure 17.

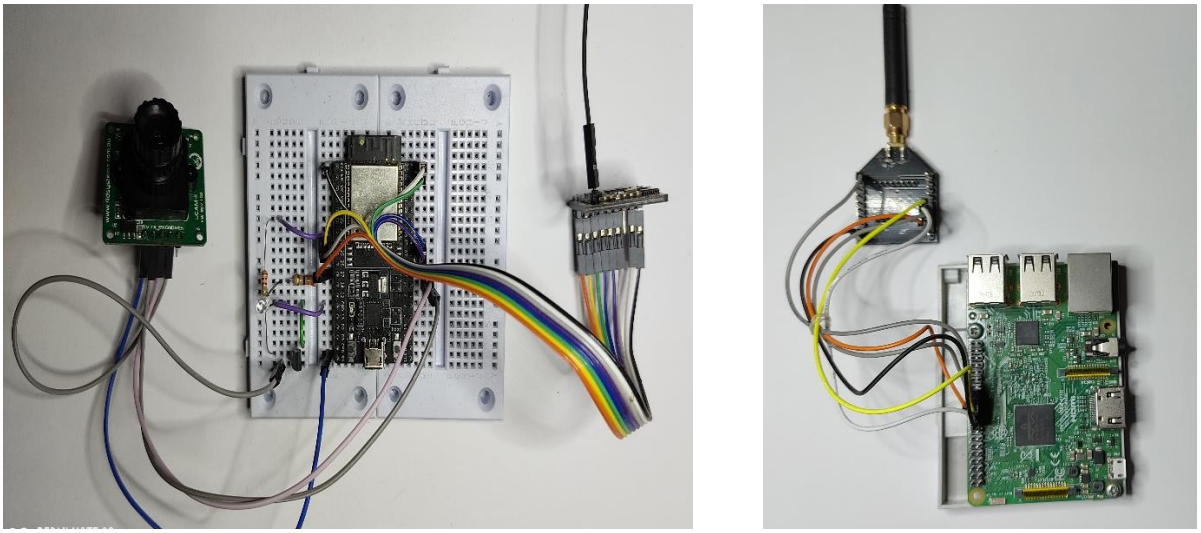


Figure 17:Camera Testing Hardware

Implementation of this star type communications network highlighted issues associated with using this revision of the serial connected camera (uCAM-II) highlighted in 4.1. An example of a captured image using this setup is shown in Figure 18 using a Quality factor of 10 and is 128x128 bytes in size.

Figure 18:128x128 Image

Mesh Topology

In the course of this research, mesh topology was implemented and tested to a limited degree. There is a brief discussion of mesh typology merits and demerits prior to a description of its implementation in this research. Working with a mesh type topology allows extended communication range between end-devices and a gateway, by routing information via a known path or by automatic route discovery. The information can be transported in a reliable way where each transmission is acknowledged by the receiver, or alternatively, in an unacknowledged way. The introduction of new end-devices can be done automatically allowing a self-healing network to be deployed, or each device can be introduced to the network in a programmatic way. The obvious benefits to this type of network include extended range because of the routing capability of each device, where a transmission is automatically (or via look-up table) routed to a destination. If the sending device is not in direct contact with the end device, the transmitted information is relayed via a device it can communicate with, to the final destination. The information is thus ‘bounced’ around the network until the delivery of the information is achieved. Routing information is constantly updated in each device which allows the introduction or withdrawal of a device from the network and allows for a changing topography within the network. This type of modular network facilitates RF access to hard-to-reach places where traditional communication type networks (e.g., Star type) have difficulty in reaching.

The issues associated with these types of networks make it difficult to implement if the end-devices are battery powered and large amounts of

information are to be transmitted such as image data. Because each device needs to be awake all the time (in listening mode) to enable them to form part of the mesh, power supply can be a major issue if these devices are positioned in a remote area where traditional power sources are absent. The maintenance and updating of routing tables incurs an overhead which can have a detrimental effect on data throughput of a system. The re-routing of data, because of a change in the network topology, or if an end-device ceases to operate, can cause a ‘flood’ of data within the network especially if the data is relayed more than once, which is highly likely. Duty-cycle restrictions can quickly become overwhelmed thus exceeding imposed regulations. The testing of such systems during development stage is a known issue, mainly due to the limited physical area within which a network can be deployed. Automatic routing algorithms are difficult to check in real-world deployment scenarios and require time to finalise design and gain operational confidence. These systems are generally simulated to allow routing checks.

Despite these drawbacks, the benefits of a mesh type network were explored within this research because of the range extension ability. The use case definitions in 1.1, require the ability to transmit Still image data over a large range (>5km). If this was achievable with a single type of network topology and meeting the other use case requirements, then a mesh type topography yields major benefits.

To alleviate the duty-cycle requirements within a mesh type network, this research explored the development of a network which allows automatic modulation type selection. This means less time-on-air if a higher throughput modulation scheme such as FSK is selected. If a device is outside the range

capabilities of such a scheme, then automatic selection of CSS type modulation scheme is adapted. This mesh topology was implemented using the same hardware as in the test bed as shown in Figure 9 and Figure 10.

Mesh Network Configuration:

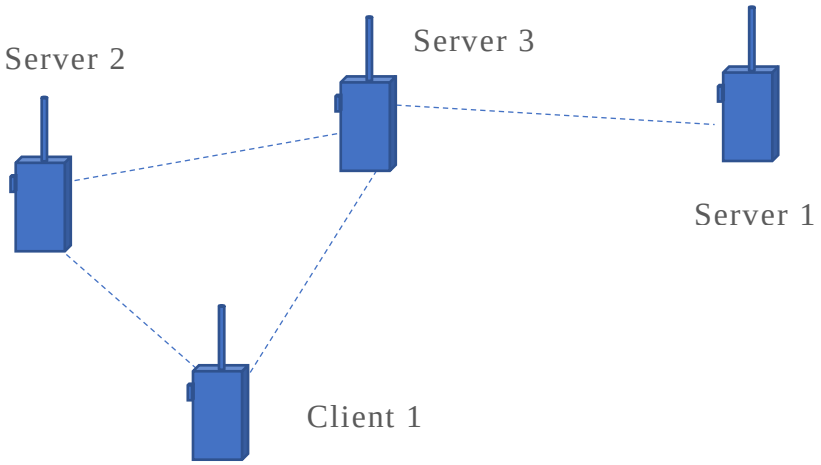


Figure 19:Mesh Network Configuration

This small network consisted of 4 devices interconnected over an RF link as shown. Server 1 can only access Client 1 via Server 3 and vice-versa. In this instance the system should automatically choose the modulation type as shown in Figure 20.

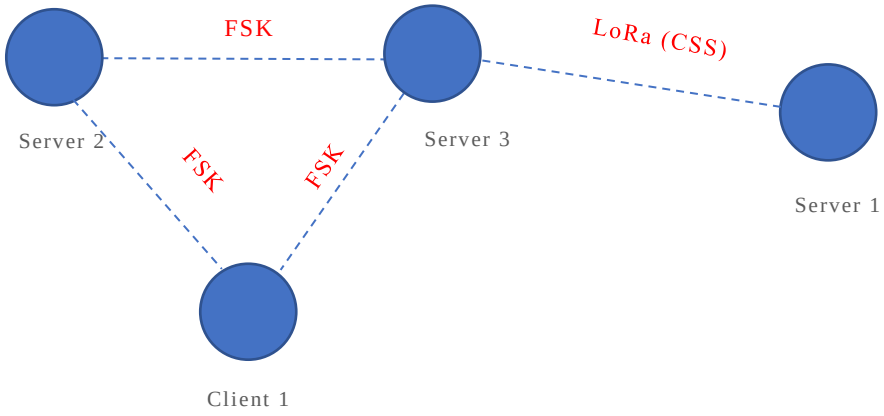


Figure 20:Mesh connection sequence

To implement the system outlined, a firmware library developed by airspayce.com [39] called RadioHead Packet Radio library for embedded microprocessors was modified for use with the hardware specified for this use case. The system firmware was developed within the PlatformIO IDE using C++ and tested for functionality in both FSK and CSS mode. The decision-making algorithm to switch between both modulation modes of operation is shown in Figure 21.

The system was successfully implemented using manual switching between modulation modes using a fixed packet size of 64bytes. However, on further analysis it was realised that to remain within the confines of duty-cycle restrictions, it was necessary to limit the spreading factor of LoRa to SF7 – SF9. This would reduce the maximum range available and negate the benefits relative to using FSK, particularly if using large amounts of data (image Transmission). This is a consequence of the increased overhead associated with maintaining the integrity of the mesh network. Due to the difficulty in real world testing of such a system, and because of time constraints, it was felt that further development was needed to fully evaluate such a concept. Further investigation would detract from the time needed to develop and

implement an image compression algorithm and this avenue of research was halted.

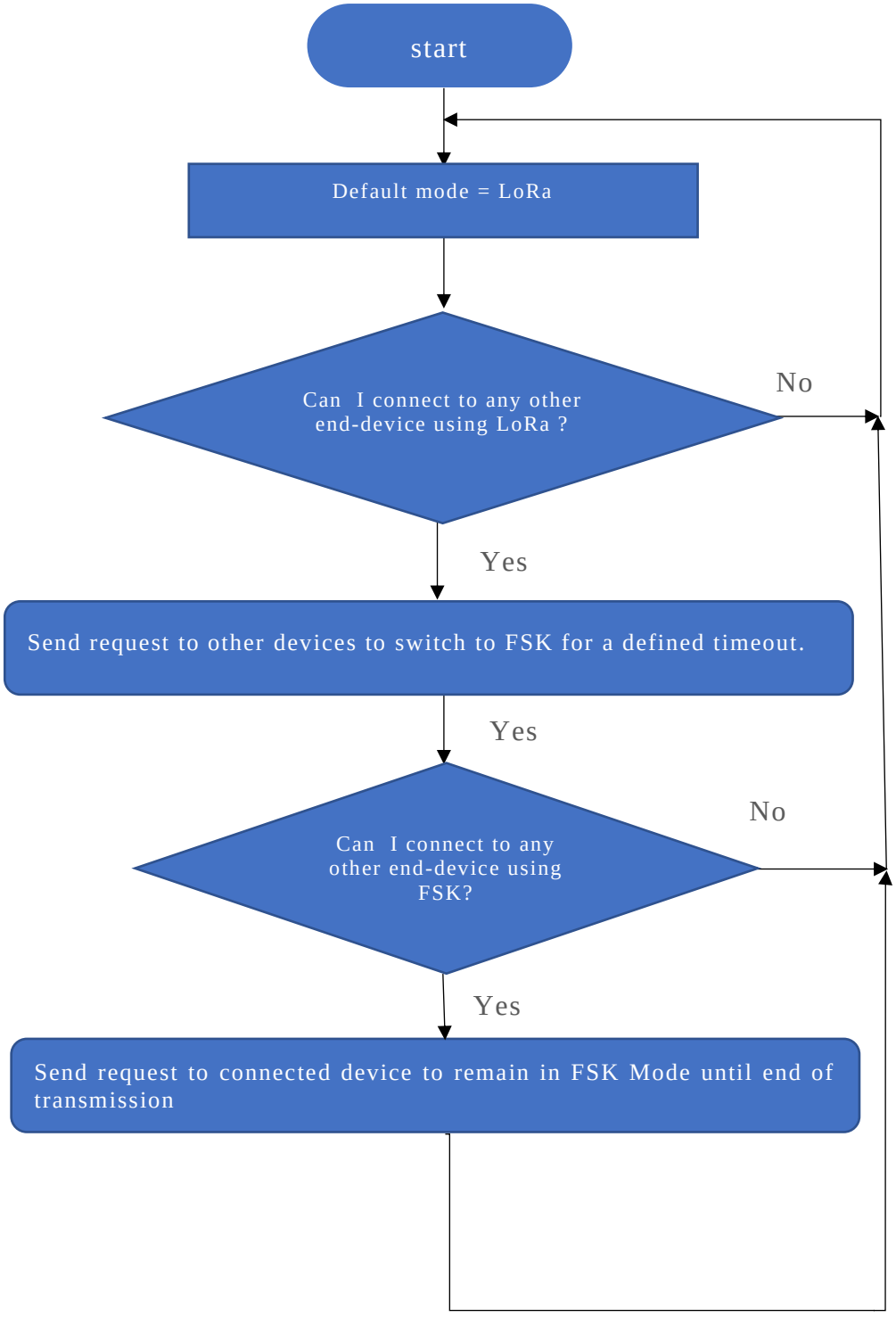


Figure 21:CSS->FSK Flowchart

Selected Network Topology

A star type network was developed using ESPCAM Platform to minimise wiring connectivity issues during development phase, and considering the issues experienced with the use of uCAM-II camera. Because all the necessary components were contained within a single module, this enabled robust hardware development. Connection via a single USB cable allowed the device to be programmed, debugged, and powered in a simple manner.

The chosen Network protocol selected for development was ESPNOW as described in 2.3. This schema (Figure 22 enabled connectivity between devices in multiple ways e.g., many-to-one, one-to-one, or one-to-many.

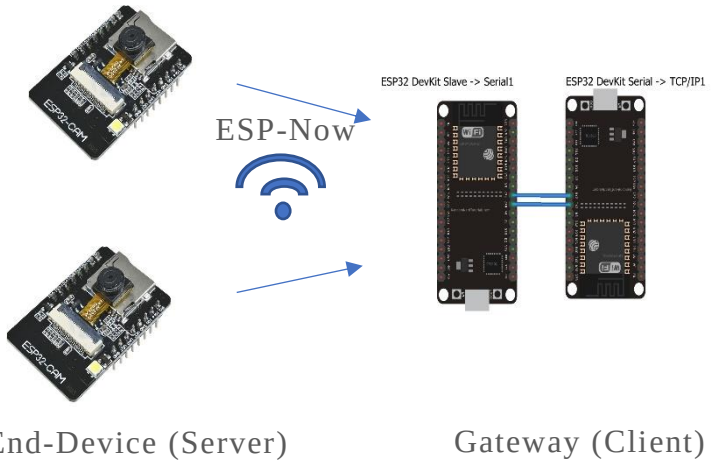


Figure 22: ESPNow Schema.

Although this modulation scheme is based on 802.11, this is a proprietary protocol and is specific to Espressif™ ESP32 devices. It is a connectionless type of scheme which allows rapid connection of one device to another which can have a positive impact on battery life.

Pairing of devices is needed prior to their communication, after pairing is done, the connection is persistent and no further handshaking is required. Common elements of programme development will be discussed in the context of server and client hardware in the following section.

Common Elements:

There are common firmware elements associated with the operation of both Server and Client hardware platforms. All necessary ESPNOW libraries are available from Espressif™ and can be freely download into common IDEs such as Arduino by adding these additional URLs to the board manager within settings:

```
https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json
```

and then under Tools>Board>Boards Manager search for ESP32 and press install button when found. This research used PlatformIO (VSCode) as the IDE but using the Arduino framework.

The terms master/controller and slave are commonly interchanged with server and client and in keeping with the nomenclature used by Espressif™, Controller and slave were used here. There is no concept of this division within the ESPNOW API as each device can act as controller or slave or both. The role of devices was defined in the configuration of each device during setup(), as SoftAP or STA mode can be selected and was selected as shown in Table 9 [12].

Table 9: ESPNow Role

Role	IDLE CONTROLLER SLAVE COMBO	The device's role. IDLE: undefined role CONTROLLER: controller SLAVE: slave COMBO: double role as controller and slave	The local device's Role will define the transmitting interface (SoftAP interface or Station interface) of ESP-NOW. IDLE: data transmission is not allowed. CONTROLLER: priority is given to Station interface SLAVE: priority is given to SoftAP interface COMBO: priority is given to SoftAP interface Station interface for Station-only mode and SoftAP interface for SoftAP-only mode. Espressif

The sequence of events needed to initialise and connect devices for both controller and slave devices are as follows:

1. Initialize the ESP-Now protocol
2. If we are developing a master or a Controller
    - Add peer (if we are developing a master or Controller)
    - Define the callback function to know if a message is sent
    - Send a message
3. If we are developing a slave
    - Add a callback function to know when a new message is arriving

To identify each device, use is made of each unit's MAC address which is unique to every device.

End-device (Controller) code to establish a basic communications network was set up as follows:

Several functions were used in this case.

- void InitESPNow()
- void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status)
- void setup()
- void deletePeer()
- bool manageSlave()

- void initBroadcastSlave()
- void ScanForSlave()
- void sendData()
- void loop()

On boot-up, void setup() is first called followed by void loop().

```
void setup()
{
Serial.begin(115200);
WiFi.mode(WIFI_STA);  //Set device in STA mode to begin with
WiFi.disconnect();
Serial.print("STA MAC: "); Serial.println(WiFi.macAddress());//
This is the mac address of the Master in Station Mode
InitESPNow();// Init ESPNow with a fallback logic

// Once ESPNow is successfully Init, we will register for Send CB to
// get the status of Trasnmitted packet
esp_now_register_send_cb(OnDataSent);
```

1. The mode of the device is set to STA with 'WiFi.mode(WIFI_STA)' because this is a controller device and sends data to a slave device.
2. Disable WiFi because this is ESPNow with WiFi.disconnect();
3. Initialise ESPNow protocol with InitESPNow();
4. Register the function to be called when data is sent with esp_now_register_send_cb(OnDataSent);

The void loop() function then runs continuously:

```
void loop() {
   // In the loop we scan for slave
  ScanForSlave();
  // If Slave is found, it would be populated in `slave` variable
  // We will check if `slave` is defined and then we proceed further
  if (slave.channel == CHANNEL) { // check if slave channel is defined
    // `slave` is defined
    // Add slave as peer if it has not been added already
    bool isPaired = manageSlave();
    if (isPaired) {
      // pair success or already paired
            // DO SOMETHING HERE
      // wait for 10 seconds to run the logic again
      delay(10000);
      }
        else
        {
      // slave pair failed
      Serial.println("Slave pair failed!");
        }



}
}
```

1. Check to see if anyone is listening with ScanForSlave();
2. Add slave address to peer list if a new device is found and continue.
3. If the device already exists within the peer list, then do something i.e., send an image or other data.
4. Wait for 10 sec and repeat (arbitrary).

The slave device consists of two ESP32 Devkits connected via UART to each other (Figure 23), designated ‘Slave-> Serial’ and ‘Serial->TCP’

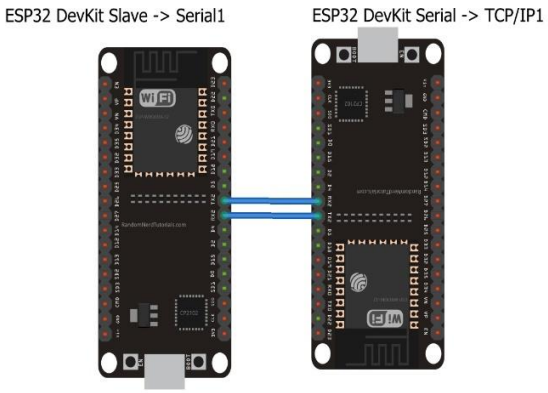


Figure 23:Serial to TCP Bridge

This configuration separates ESP-Now and normal WiFi. Although each ESP32 Devkit is Wi-Fi capable, it was not possible to use both modulation schemes at the same time without resetting the Wi-Fi driver. To solve this issue, it was decided to separate ESP-Now from Wi-Fi, and relay any data received by the slave (Slave>Serial1) to a dedicated device which was connected to a local router (Serial -> TCP). The connection between devices is over standard serial type port. The serial - > TCP device could be a more capable platform such as a Raspberry pi which would allow post-processing to be achieved locally.

Slave Device (Slave->Serial1) code:

Programming steps for a slave device is like that used by the controller.

Step 1: ESPNow Init on slave.

Step 2: Update the SSID of slave with a prefix of `slave`.

Step 3: Set slave in AP mode.

Step 4: Register the receive callback function and wait for data.

Step 5: Once data arrives, print it in the serial monitor.

Setup() runs once on bootup and then loop() continuously.

```
void setup() {

  Serial.begin(115200);
  Serial2.begin(115200,SERIAL_8N1,16,17);               // connection with socket ESP32 - pin 16 & 17

  // Set the device as a Station and Soft Access Point simultaneously
  WiFi.mode(WIFI_AP);

  // configure device AP mode
  configDeviceAP();

  // Init ESPNow with a fallback logic
  InitESPNow();

  esp_now_register_recv_cb(OnDataRecv);

    //Create ring buffer
    //RingbufHandle_t buf_handle;

  buf_handle = xRingbufferCreate(70000, RINGBUF_TYPE_BYTEBUF);
  if (buf_handle == NULL)
  {
    printf("Failed to create ring buffer\n");
    }


}
```

Some extra functions used in this instance are:

1. xRingbufferCreate(): this creates a ring buffer (circular buffer) which is needed to manage data rates.

As before, during setup(), the device is configured as an access point (configDeviceAP();) ESPNow is initialised, and a call-back function is registered to handle activity when data is received. A ring buffer is established (xRingbufferCreate(70000, RINGBUF_TYPE_BYTEBUF) of type byte, to handle the asynchronous way data is received and transmitted

over Serial1 to the Serial->TCP device. The information transmitted over ESP-Now can arrive more quickly than the program can respond and send over the physically connected serial port. This results in missing information and out-of-sequence handling of the complete system if not buffered. This allows data to be received, processed, and transmitted at two different rates.

The main programme loop():

```
void loop() {

    //Receive data from byte buffer
    size_t item_size ;
    uint8_t *item = (uint8_t *)xRingbufferReceiveUpTo(buf_handle, &item_size,
pdMS_TO_TICKS(500),1024);
    //Check received data
    if (item != NULL)
    {
   if (!sync)
     {
      for (int i = 0; i < item_size; i++)
      {
        //Serial.print(item[i]);
        //Serial.print(",");
        //Serial2.write(item[i]);
        data[i] = item[i]; // check if first three bytes in correct sequence
        vRingbufferReturnItem(buf_handle, (void *)item);
     }
     i = 0;
```

A portion of the main loop() is shown here. When data is received into the Ring buffer an event is triggered which reads the data and checks if certain criteria are met. After data is removed from the buffer, vRingbufferReturnItem(buf_handle, (void *)item); is called to reset pointers into the buffer to enable more data to be received. When certain criteria are met, data are saved to another buffer and then transmitted over the serial connection to the Serial-> TCP device.

5.2 Image Transmission

To enable the transmission of image data as specified in Table 2 a stepped approach of transmitting a full uncompressed Image was first explored. The image was a 320x240 8bit grey scale, giving a total payload size of 76800 bytes. The designed test bed was used with image data captured by the on-board OV2460 camera. This allowed testing of an end-to-end system including Camera interface, RF Interface, packetization algorithm and reconstruction of uncompressed data at the application layer interface. The packetization of the image data used by the end-device is shown below.

Packetization

The format of the vendor-specific action frame used by ESP-NOW is shown in Table 10. [46]

Table 10: Vendor-specific action frame

MAC Header	Category Code	Organization Identifier	Random Values	Vendor Specific Content	FCS
24 bytes	1 byte	3 bytes	4 bytes	7-255 bytes	4 bytes

The Field of interest is the Vendor Specific Content shown in Table 11.

Table 11:Vendor Specific Content

Element ID	Length	Organisation Identifier	Type	Version	Body
1 byte	1 byte	3 bytes	1 byte	1 byte	0~250 bytes

Specifically, the field ‘Body’ which contains the payload data is of interest. This is the location of the packetized data.

Because the maximum payload size per transmission was 255 bytes, a means to split the image data into packets was developed. The format of the packet contained enough information for the received data to be reconstructed into the original format regardless of packet reception sequence. This would be suitable for use on other microcontroller platforms and by other RF protocols discussed within this research project. The main criteria for cross-platform usage is the maximum payload size capability of 255 bytes per transmission. A packet of data was structured; this constituted the 'Body' of data within the vendor specific data field.

The developed packet structure is shown in Table 12.

Table 12: Packet Structure

Frame I.D	Frame Start	Data
1 Byte	4 Bytes	240 Bytes

Frame I. D - Specifies the type of frame to be transmitted.

Table 13:Frame I.D Specifics

<table>
<tr><th>Value (Decimal)</th><th>Function</th></tr>
<tr><td>Random 0 -10 (lower nibble)
EXOR with 144 to indicate this is an uncompressed image.
e.g.
<table>
<tr><td>Rnd.</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>144</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>Result</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
</table></td><td>A random value between 0 and 10 is used to indicate that each packet received within a defined time is from the same image. i.e. if there are multiple end-devices, a means to identify each packet stream is necessary.</td></tr>
<tr><td>127</td><td>Used to indicate end of Image data.</td></tr>
</table>

Frame Start- Specifies the location of the start of each frame. i.e. an image captured by the on-board camera is stored in a 1-D Array. To reconstruct the array faithfully, it was necessary to know the start location of each packet within the array.

Data - Image data was packed in fixed lengths of 240 Bytes.

Synchronisation of the first data packet was achieved by setting the frame I.D to a randomly generated number between 0 – 10, and Exoring the full byte with a type identifier, in this instance 144. Setting the next 4 bytes (Frame Start) of data = 0, the receiver checks for this condition and synchronises with the transmitter. Subsequent transmissions contained the same I.D and calculated frame Start values until a decimal value of 127 was identified within the Frame I.D field Table 13. This signalled the last packet within the current image data stream. The type of transmitted frame was indicated by the upper nibble of the I.D. byte. Each transmitted payload packet was 245 bytes in length, allowing a further 10 bytes for future use before the payload limit was reached.

The number of packets to transmit in non-compressed mode was image size/Data size, as shown in Equation 17.

Equation 17: No. of uncompressed Packets

$$Number\ of\ Packets = \frac{76800}{240} = 320$$

The code to achieve a full end-to-end transmission of a digital image is included in the appendix along with a detailed explanation of the main functions used for clarity.

An explanation of the code to achieve synchronisation across the RF network for image transmission is presented next.

The function ‘fullImage()’ developed to achieve this is shown below.

```
void fullImage(){
    camera_fb_t *fb = esp_camera_fb_get(); //Snap picture

    if (fb) {
       srand((unsigned) time(0));    // generate a random number 1 - 10, to use as frame I.D in payload
       int randomNumber;
       for (int index = 0; index < 1; index++) {
       randomNumber = (rand() % 10) + 1;
       }
       uint16_t num_of_frames = (fb->len / frame_length);
         // for QVGA - 320X240 Pixels, using greyscale B&W = 76800 Bytes.
         // each frame is 240 bytes long.
         // Number of frames = 76800/240 = 320 frames.

       for (cnt = 0; cnt < num_of_frames+1; cnt++) // frames+1 otherwise only 383 packets sent.
       {
         dataToSend[0] = randomNumber | 144;     // frame I.D., 144 is Raw image indicator
         dataToSend[1] = highByte(cnt);          // frame count number.
         dataToSend[2] = lowByte(cnt);
         dataToSend[3] = 0;
         dataToSend[4] = 0;

         if(frameEnd >= fb->len){
           frameEnd = fb->len;
           frame_length1 = (frameEnd - frameStart);

         }
         else{
           frame_length1 = frame_length;
         }

         while (i < frame_length1 )
         {
          dataToSend[i + 5] = fb->buf[frameStart+i];
          i++; // number of elements to copy
         }

          i = 0; // reset index for next value of cnt.
          sendData();
          frameStart = frameEnd;
          frameEnd = frameStart + (frame_length1);
    }
    esp_camera_fb_return(fb); // release camera buffer.  DO NOT RELEASE IF MULTIPLE
VARIATIONS ARE TO BE SENT FOR COMPARISION
    frameStart = 0;
    frameEnd = 240; //200
    Serial.println("Complete Frame Sent ---------------------------------------");
     }else{
          Serial.println("NO snap taken ?");
       }
   }
```

The type of image to be sent was selected in the main loop().

```
void loop() {
   // In the loop we scan for slave
  ScanForSlave();
  // If Slave is found, it would be populate in `slave` variable
  // We will check if `slave` is defined and then we proceed further
  if (slave.channel == CHANNEL) { // check if slave channel is defined
    // `slave` is defined
    // Add slave as peer if it has not been added already
    bool isPaired = manageSlave();
    if (isPaired) {
      // pair success or already paired
      fullImage();
      //delay(7000);
      //compress();   // take photo & compress & send data.
      //delay(8000);
     // ROI();    // Region of Interest - uncompressed & send data

      // wait for 10 seconds to run the logic again
      delay(10000);
      }
        else
        {
      // slave pair failed
      Serial.println("Slave pair failed!");
        }



}
}
```

The Slave-> Serial1 (Figure 23) device is the client side of the ESP-Now connection. After the device had been configured during setup, the main loop() ran continuously. Upon reception of data via ESP-Now, a conditional loop is entered, dependent on the parameters identified in the I.D. field of the payload data. For this condition (uncompressed image), if the number 144 is identified and the random number is within parameters, then the received data

is stored in a new array for processing. The raw conditional entered is shown below.

```
    //---------------------------------------------------------------------
    // Raw Data conditional
    //---------------------------------------------------------------------
     else if ((data[0]& 0xf0) == 144 && (data[0]& 0x0f) <=10) // is first Byte (I.D) < =  10 ?
and RawImage flag set ?
    {
      // Get next byte
      j = 0;
      if (data[1] == 0 && data[2] == 0) // is 1st,2nd Byte = 0 ?
      {
        sync = true;
        Raw = true;
        for (int i = 0; i < item_size && j <= 78400; i++) // Don't know how many Bytes were
buffered so get size.
        {
          Payload[j] = data[i];              // Store next Byte in correct location in Array.
          j++;

        }
        i = 0;
      }
    }
```

Once synchronisation was achieved, signalled by sync=true, the rest of the received data is ‘pulled’ from the ring Buffer until all the expected raw data is received. A FullImage flag is then set as shown below.

```
    if (sync && !sync1 && !sync2){
      for (i = 0; i < item_size  ; i++)
        {
          data[i] = item[i];            // Read in buffered data and store in next location in
'Payload' array
          Payload[j] = data[i];
          vRingbufferReturnItem(buf_handle, (void *)item);
          j++;
          //if (data[0] == 127 ){   // possibility of false trigger ?
          if (j >= 77600 ){   // possibility of false trigger ?
            FullImage = true;
          }

        }
       i = 0;
    }
```

Once FullImage and Raw has tested true then the array (Payload) is written to the connected ESP32 via Serial1.

The ESP32 Devkit, Serial->TCP device was programmed to connect to a local Wi-Fi router and is set to run as a TCP/IP connected server over port 3000 with a fixed IP address. When a remote Client connects to this server, any data received over the Serial port is written to the connected Client. The main loop() to achieve this is shown below.

```
void loop() {
  CheckForConnections();
  if (RemoteClient)
  {
  while ((capacity = Serial2.available())) {
  if (capacity > 0)
  {
    for (i = 0; i < capacity; i++)
    {
      data[i] = Serial2.read(); //
    }
     RemoteClient.write(data,capacity);
     i = 0;
     Frame = true;
  }
  }
  }
}
```

Raw image example:

An image size of 320x240, QVGA (Quarter Video Graphics Array) was transmitted over a basic network previously described using ESP-Now protocol. The image was in raw format i.e., non-compressed and consisted of an image vector of length 76800 bytes as shown in Figure 24.

Figure 24:Raw image example

The image is shown full size. A graph of brightness distribution is shown in Figure 25.

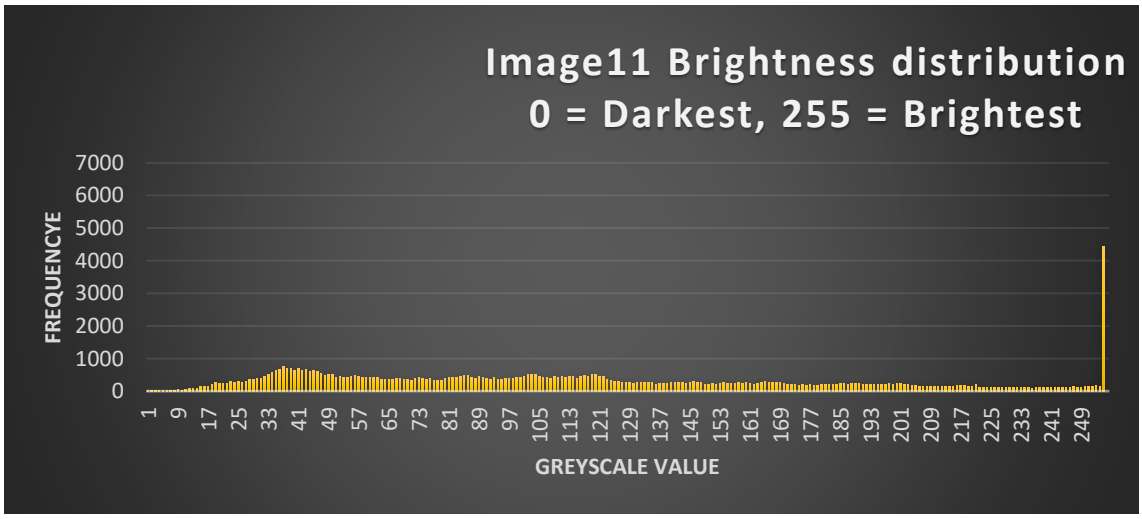


Figure 25:Brightness Distribution

The occurrence of ‘darker’ sections of the image (<127) are dominant except for a peak in maximum brightness at 255.

A further Image Figure 26 has a different distribution curve.

Figure 26: Raw Image 2

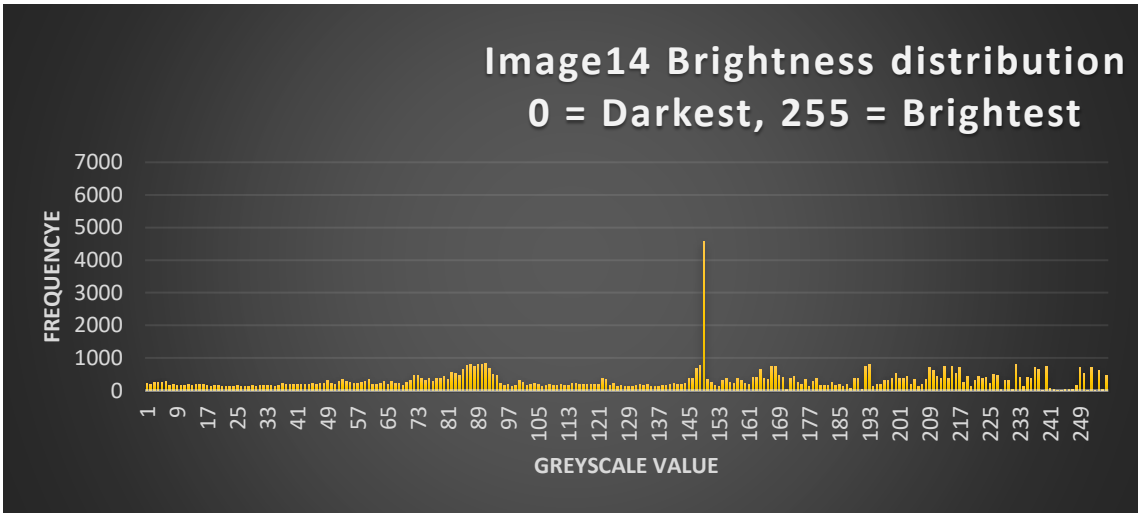


Figure 27:Brightness distribution Image 2

5.3 Image compression

This research has developed and enhanced a compression algorithm based on work done by Rafeeq AL-HASHEMI et al. [19]. The technique described is suited for use in resource-constrained devices as computation requirements

are low and allows flexibility in image size and matrix shape to be chosen. It is described as a semi-lossless compression technique using RLE (Run Length Encoding) and utilises pixel value rather than bit value. The compression algorithm described has been enhanced to allow the selection of three quality factors, QLow, QMed and QHigh. The selected quality factor remaps the original data from 0-255 values to 8,16 and 32 values as required. Two compression quality factors were implemented, and results evaluated.

Each pixel is represented by an 8-bit value, which represents the brightness of each point in the range 0-255. The selected image quality for this research was based on standard QVGA (320x240) which allowed enough detail to be observed for this use case. The image taken by the camera on the end-device was stored in the camera memory as a 1-D array and is 76800 bytes in length. The raw data was accessible via an index into this array (buffer). This process can be used on colour images if each colour is first mapped to a vector whose range varies from 0-255. In this case, we used 8-bit greyscale data.

RLE

Run length encoding is a common technique used to compress data in a lossless manner. Because a lot of consecutive data elements are the same value in image data, a sequence of data that represents an image is stored as a single data value and count, rather than the original data sequence. This method is particularly suitable for use in greyscale images where the value of each pixel carries only intensity data. If other types of data were presented where there are not many runs of similar data, it is possible that this method could double the size of the original file.

The technique used by Rafeeq AL-HASHEMI et al. firstly remaps each pixel byte value from 0-255 to 128-255. This reduces the pixel intensity range from 256 to 16. The remapping is achieved by resetting the lower 4-bits of each byte (lower nibble) to zero by logically ANDing the value with 0xf0.

Table 14:Remapping Example

Original byte value	0	0	0	0	1	0	1	0
Mask	1	1	1	1	0	0	0	0
Resultant	0	0	0	0	0	0	0	0

For example, the original value is decimal 10, this is remapped to 0. In fact, values 1-15 will be remapped to 0. This method is implemented for each pixel value and stored in a new array. The mapping table is shown in Table 15.

Table 15: Mapping Table

Original Value	Mapped code
240-255	240
224-239	224
208-223	208
192-207	192
176-191	176
160-175	160
144-159	144
128-143	128
112-127	112
96-111	96
80-95	80
64-79	64
48-63	48
32-47	32
16-31	16
1-15	0

Once the original data has been re-mapped, RLE is then used on the remapped data. The new information, the value and frequency of occurrence, is saved in a new byte of information. If the remapped decimal data value = 64, frequency of occurrence after RLE = 8, then the following values apply as shown in Table 16 .

Table 16:Remapped value + Frequency of occurrence.

Remapped data.	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
Frequency	0	0	0	0	1	0	0	0
New Byte	0	0	1	0	1	0	0	0

If the RLE occurrences exceed the 4-bit limit (15), then the algorithm will divide this ‘New Byte’ into consecutive pairs that cope with the 4-bit limitation. Because a packet size limit of 255 bytes exists, it is necessary to packetize each transmission in a structured format as shown in Table 12. Because the final length of each compressed image file is unknown until the last packet is sent (indicated by setting I.D = 127), it is necessary to monitor the position of each data element within the original uncompressed vector. To maintain a constant packet payload size during transmission for simplicity and efficiency, the RLE maintains the location of each last data byte within the original image vector. This is necessary because RLE is performed on the re-mapped image file and is normally of different lengths, determined by the similarity of consecutive bytes for each structured packet. If a standard number of re-mapped bytes were chosen (say 240), and RLE is performed on this data, then it is likely that each final data payload size will be less than

240 bytes. To ensure transmission efficiency, a standard payload size of 240 bytes is maintained. If the last packet to be sent is less than 240, then it is artificially filled to maintain the correct structure. Because the compression algorithm is a key component of this thesis a detailed explanation of the code to achieve this is presented next.

```
void compress(){
    camera_fb_t *fb = esp_camera_fb_get();    //Snap picture
    if (fb) {
     first = true;                              // first frame indicator
     Serial.println("Snap Taken");              // COMPRESSION AND RLE STARTS HERE.
     srand((unsigned) time(0));                 // generate a random number 1 - 10, to use as frame
I.D
     for (int index = 0; index < 1; index++) {
     randomNumber = (rand() % 10) + 1;
      }
       i = 0;
       while (i <= 76800)
       {
         while (j < 240 && i <= 76800)
         {
          if (Qhigh ){                          // if Qhigh is selected then
          mask = 0xf8;                           // map to 32 values
          many = 7;
          }
          else if(QMed){                        // if QMed is selected then
          mask = 0xf0;                           // map to 16 values
          many = 15; }
          k = fb->buf[i] & mask;
          while (k == (fb->buf[i+1] & mask)&& i <= 76800)  // map to Qhigh, Qmed,Qlow values
          {
           Count++;
           i++; }
          if (Count <= many && i <= 76800)
          {
           NewArray[j] = k + Count;
           j++;
           if(Count == 0){
            i++;
           }
           else{
            i++;}
          }
          else if (Count > many && i <= 76800)
          {
           uint8_t Quot = Count / many;
           uint8_t Rem = Count % many;
           while (Quot > 0)
           {
            NewArray[j] = k + many;
            Quot--;
            j++;
            if(Quot == 0 && Rem >0){
            NewArray[j] = k + Rem;
            j++;
            }
           }
           i++;}
         Count = 1; }
        j = 0;
        p = 0;
        PacketCompressed();
        sendData();}
    }
    i = 0;
    frameStart = 0;
    frameEnd = 240;
    esp_camera_fb_return(fb); // release camera buffer.}
```

if (Qhigh){ …… selects the compression ratio, here, image data is re-mapped to 32 values.

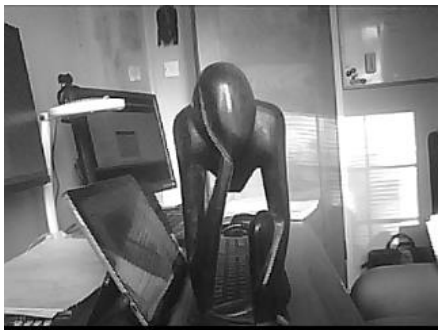
if(QMed){…….. selects the compression ratio, here, image data is re-mapped to 16 values.

NewArray[j] …. This is the array containing the re-mapped values.

PacketCompressed(); This formats each packet for transmission.

esp_camera_fb_return(fb); The camera buffer is released here for next image.

An example of transmitted compressed and raw images is shown Figure 28.



Raw Image

Compressed Image

Figure 28:Raw & Compressed Image

Debugging information at the client is shown below.

```
comp11541
Image size = 22295
ID = 0
Start = 0000
loc = 0
I.D = 166loc = 1360
I.D = 6loc = 2647
I.D = 6loc = 3784
I.D = 6loc = 4929
```

Comp11541 is the image reference. The received image size is shown, and the decoded vector location is shown for each received packet (e.g., loc = 1360) along with the I.D. of the transmission. The location data is extracted from the transmitted frame structure, i.e., data elements 1 – 4. This information is used to reconstruct the vector image array data.

If using the quality factor set to QMed (re-mapped to 16 values) a PSNR, Equation 1, of ~ 36 dB is calculated with a compression ratio of 71%.

Table 17: PSNR QMed

Image (ID)	Quality	Image Size	Compression Ratio	PSNR dB
Raw (9941)	0-255(Full)	76800	1	
Compressed(11541)	0-16 (QMed)	22295	70.97	35.96

The compression ratio is computed using Equation 18.

Equation 18: Compression Ratio

$$CR = \left(\frac{x - y}{x}\right) * 100$$

Typical values for PSNR in lossy image and video compression are between 30 and 50 dB, where higher is better. Acceptable values for wireless transmission quality loss is considered to be about 20 dB to 25 dB.

Raw image Compressed image

Figure 29:Raw & Compressed (QHigh)

If using the quality factor set to QHigh (re-mapped to 32 values) a PSNR, Equation 1, of ~ 37.4 dB is calculated with a compression ratio of 64%.

Table 18: PSNR QHigh

Image (ID)	Quality	Image Size	Compression Ratio	PSNR dB
Raw (10021)	0-255(Full)	76800	1	
Compressed(17481)	0-32 (QHigh)	27685	63.95	37.4

The compressed image shown in Figure 29, shows artefacts generated by missing information, highlighted in red. The compression algorithm is robust to transmission disturbances and allows best effort reconstruction of received data. It also allows reconstruction due to out of sequence packet reception because each packet is treated as an independent transmission and includes array location information.

ROI

The outline specification for image compression in Table 2 calls for the ability to select a region of interest for transmission. In certain use cases, one

region of an image may be of more importance than the rest, for example, identification of a car number plate, or sections of a medical image. To minimise the required bandwidth needed, one technique used was to compress all the image data except for the region of interest. The selected ROI was then transmitted alone or combined with a compressed data image. The firmware routine developed in this research allowed selection of a small region of a complete image to be transmitted, in either uncompressed or compressed format. Selection of a small subset of data and compressing it, enabled this method for use with long range systems (e.g., LoRa) where all resources were constrained. To select a subset of data, a coordinate system was developed enabling a part of an overall image to be selected and transmitted.

For this use case, the selected image size was QVGA, which is an image of size 320x240 bytes using 8-bit grey scale. The camera used for the end-device stores an image as a 1-D Array (vector) of length 76800 bytes. The image is represented as shown in Figure 30

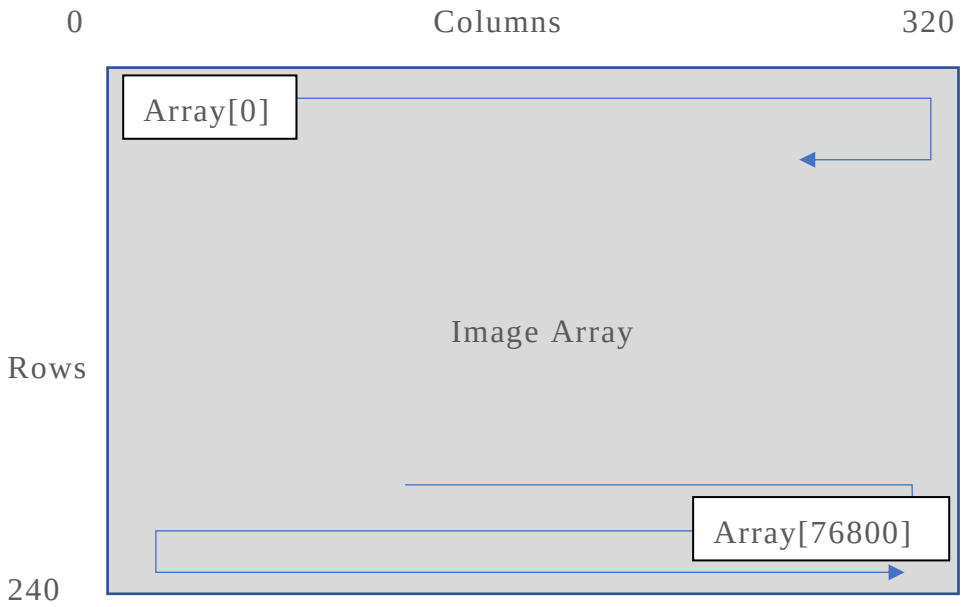


Figure 30:Image Array

Each image is arranged in columns and rows. Row 1 consists of Array [0-319] data elements, row 2 consists of Array [320-639] and so on. To select a ROI, a grid size of 80x60 bytes was overlaid on the existing image using a technique to directly manipulate any pixel within the image data array. A grid size of 80x60 was chosen but could be any grid size required. This allows an even division of columns and rows, i.e., 320/4 = 80 and 240/4 = 60. The chosen grid size will contain 80x60 = 4800 bytes of information. The upper left-hand corner of a selected grid was selected, and an algorithm was developed to translate this coordinate into an array for transmission. The ROI size was chosen to be a multiple of the allowed packet size (i.e., 240), which means the total transmission consisted of 20 packets (4800/240). The grid overlay is represented in Figure 31.

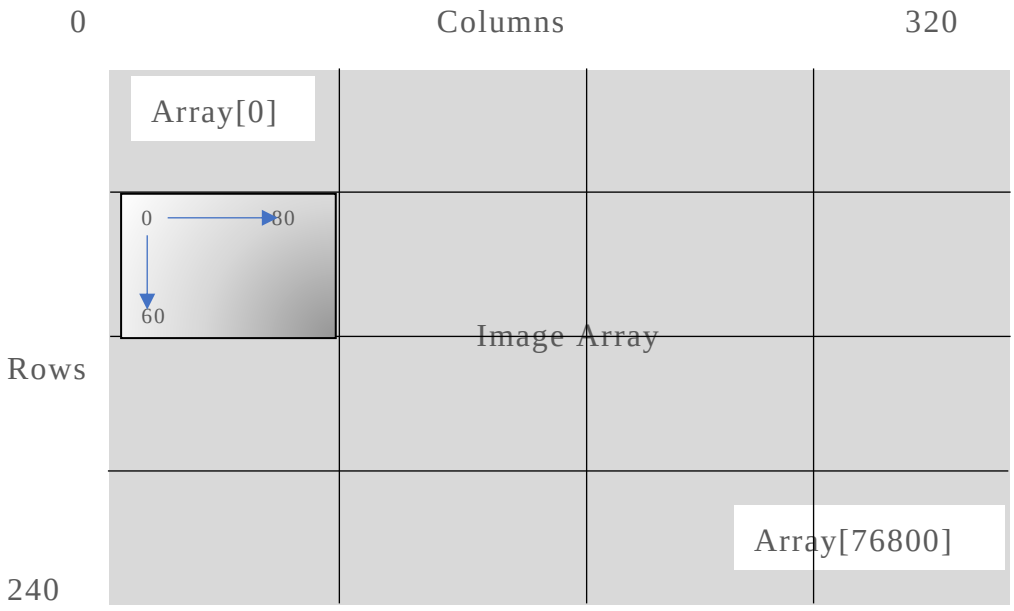


Figure 31:Grid Overlay

This is an uncompressed image size but could be further reduced in size if the developed compression algorithm is used. If using a quality factor QMed, the size could effectively be reduced by 70%, see Table 17. The overall size would then consist of 6 packets to be transmitted, a large reduction in power consumption of an end-device could then be realised. For development purposes, the grid selected is done manually during programming of the end-device. In this instance, a grid reference of column = 80 and row = 60 is chosen which determines the upper-left hand corner of the grid. This is now translated into a specific location within the full image array i.e., an index into the image vector.

To translate the grid coordinates into vector indices, several variables are first defined, and then used to calculate the array index defined in Figure 32

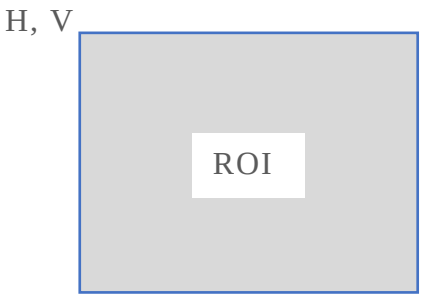


Figure 32: ROI Value calculation.

H = Horizontal Column number.

V = Vertical Row number.

W = Number of columns in complete image.

L = Number of Rows in complete image.

To calculate the top left-hand corner index, we can use Equation 19.

Equation 19:ROI Top left-hand index

$$h = (V * W) + H$$

To calculate the bottom right-hand corner index, we can use Equation 20.

Equation 20:ROI Bottom right-hand index

$$r = (h + H) + (V * W)$$

The variable h represents the index into the image array representing the top left-hand corner of the selected grid.

The variable r represents the end location of interest into the image array.

For Example.

Table 19:ROI Worked example.

H = 80	**V = 60**	**W = 320**	**L = 240**

Top left-hand corner:

$h = (V * W) + H$

h = (60x320) +80

h = 19280 ..... This is the index into the image buffer array representing the ROI grid top left corner.

Bottom right corner:

$r = (h + H) + (V * W)$

r = (19280 + 80) +(60x320)

r = 38480 ..... This is the index into the image buffer array representing the ROI grid bottom right corner.

The selection of a ROI to transmit, is achieved by selecting a combination of only two variable values (H and V) shown in Table 20.

Table 20: ROI Parameter selection

H= 0 V= 0	H= 80 V= 0	H= 160 V= 0	H= 240 V= 0
H= 0 V= 60	H= 80 V= 60	H= 160 V= 60	H= 240 V= 60
H=0 V= 120	H= 80 V= 120	H= 160 V=120	H= 240 V= 120
H= 0 V= 180	H= 80 V= 180	H= 160 V= 180	H= 240 V= 180

The code developed to produce the necessary packet structure for transmission is shown next.

```
void ROI(){
  camera_fb_t *fb = esp_camera_fb_get();      //Snap picture
  first = true;
  srand((unsigned) time(0));                  // generate a random number 1 - 10, to use as
frame I.D in payload
  for (int index = 0; index < 1; index++) {
  randomNumber = (rand() % 10) + 1;
                    }
  if (fb) {
    j = 0;
    h = (V * W) + H;
    r = (h + H) + (V * W);
    frameStart = h;
    while (h < r)  // r
    {
      while (j < 240)
      {
        for (i = h; i < h + 80; i++)  // h+80
        {
          NewArray[j] = fb->buf[i];
          j++;
        }
        h = h + W;                            // update index into array
      }
      j = 0;
      PacketROI();                            // construct transmission packet
      sendData();                             // transmit the data.
    }
      //frameStart = h + j;

  esp_camera_fb_return(fb); // release camera buffer.
  }
}
```

PacketROI(); is the function required to construct a packet for transmission. To identify that this is a ROI packet, the upper nibble of the I.D field is set = 96.

An example of a received uncompressed ROI image with uncompressed full image is shown in Figure 33.

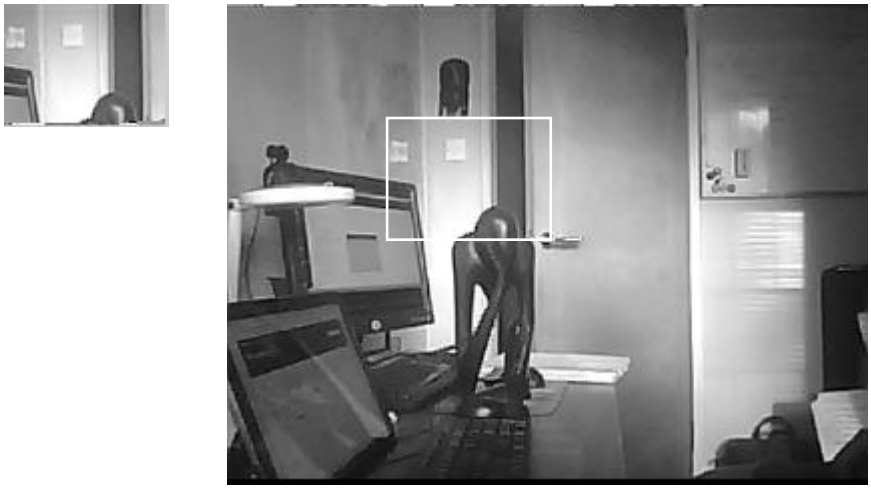


Figure 33:ROI uncompressed

The ROI is shown full size and its location is shown on the full uncompressed image. This represents a grid reference of H = 80, V = 60.

Pixel writing

The ability to read individual pixel values allows decision-making based on image content such as intrusion/movement detection and is an area of research worth exploring further, though not used in intrusion/movement detection in this use case. The ability to write to specific pixel locations also offers opportunities for specific use cases and is a specification requirement for this research as shown in Table 2.

To enable direct manipulation of any pixel value, use is made of a graphics library supplied by Adafruit.com [47]. The Adafruit_GFX library provides a common syntax and set of graphics functions for most LCD and OLED displays. It has been adapted here for use with images taken by the OV2640 which is the camera supplied with the ESP32-CAM.

An example of a full-size uncompressed image, and the same image compressed using QMed with graphics overlay is shown Figure 34.

Figure 34:Image Overlay

The compressed image has a 4x4 grid overlay and information relating to battery voltage level embedded in the image. Regardless of the changing image, the grid overlay remains in the exact location specified during programming. This facility could be used for monitoring specific areas of an image for change. For example, growth rate monitoring of bacteria in a Petrie dish or change of size of cracks in a structure. The direct manipulation of an image at transmission time does not add to the overall size of the transmitted image data. This allows for transmission of extra information without incurring a data size cost.

The code used to implement this functionality is shown next.

```
//------------------------------------------------------
// Pixel drawing stuff here.
//------------------------------------------------------
class aFrameBuffer : public Adafruit_GFX {
  public:
    uint8_t *buffer = NULL;
    uint32_t size1;
    int w;
    int h;
    aFrameBuffer(int16_t ww, int16_t hh): Adafruit_GFX(ww,hh){
      w = ww;
      h = hh;
      size1 = h * w;
    }
//------------------------------
// where to draw
//------------------------------
void setBuffer(uint8_t *b){
       buffer = b;
}

//------------------------------
// Drawing a Pixel
//------------------------------
void drawPixel(int16_t x, int16_t y, uint16_t color){
  if(x<0 || x>= w || y<0 || y>=h)
    return;
  buffer[x + y * w] = color;
}
};
aFrameBuffer OSD(320, 240);
```

This code is used to initialise the library. The code to enable image overlay with a grid and battery information as shown in Figure 34. In this instance it is used within the compress () function but could be applied to any of the image functions developed during this research.

```
void compress(){
     camera_fb_t *fb = esp_camera_fb_get();     //Snap picture
     if (fb) {
       //------------------------------------------------
       // Write to main camera buffer directly
       OSD.setBuffer(fb->buf);
       OSD.setTextSize(1);
       OSD.setTextColor(BLACK,WHITE);
       for (i = 80; i < 320;i += 80){
         OSD.drawFastVLine(i, 0, 240, WHITE);
       }
       for (i = 80; i < 240;i+= 80){
         OSD.drawFastHLine(0, i, 320, WHITE);
       }
       OSD.setCursor(20, 220);
       OSD.print("Bat. Voltage = 3.4v");
       //------------------------------------------------
```

5.4 Chapter Summary

This chapter outlined the development of a suitable data compression algorithm for use with resource-constrained devices. For testing purposes, a suitable modulation scheme (ESP-NOW) was chosen. This scheme used a vendor specific content field within the vendor specific action frame as shown in Table 10. The data contained within this field is structured in such a way as to allow packetization of a large amount of data. The packetization structure developed during this research is outlined in Table 12 and allows for a constant packet size of 245 bytes per transmission. The final compressed data file can vary in length and is dependent on the lighting conditions of the image taken by the camera. The maximum payload size allowed in the three modulation schemes discussed is 255 bytes. A stepped approach to developing a working compression scheme was used.

- Design a suitable packetization structure.
- Confirm functionality of end-to-end RF network using an uncompressed image of size 76800 Bytes (QVGA).
- Design and test a suitable compression algorithm suitable for use with all three modulation schemes.
- Confirm functionality of complete end-to-end RF system using compressed data structure.

Typical results for compression ratios and PSNR achieved are displayed below.

Image (ID)	Quality	Image Size	Compression Ratio	PSNR dB
Raw (9941)	0-255(Full)	76800	1	
Compressed (11541)	0-16 (QMed)	22295	70.97	35.96
Raw (10021)	0-255(Full)	76800	1	
Compressed (17481)	0-32 (QHigh)	27685	63.95	37.4

Table 21:Quantitative results

An approximate compression ratio of 71% was achieved with a PSNR of 36 dB using a quality factor set to QMed. This quality factor re-maps the original data from 0-255 to 0-15 levels of brightness intensity. When QHigh is selected, the original data is remapped from 0-255 to 0-31 levels of brightness intensity. The outline specification in Table 2 calls for the ability to select a region of interest (ROI) and direct manipulation of pixel data capability. This was demonstrated in section 0 and allowed transmission of smaller amounts of data of interest and direct manipulation of image data. It was possible to include extra information within the image data. This ability increased the use case scope of the current research. The small firmware footprint incurred in both the compression algorithm and image manipulation further satisfied the outlined specification.

6 CHAPTER 6 DISCUSSION AND CONCLUSIONS

This chapter discusses the process of design and implementation of the complete RF image transmission system and the boundaries which apply for system operation in detail. It then draws some conclusions from this use case design and points to possibilities for future work.

The implementation and successful testing of low-power RF networks is highly dependent on the specific use case. A one size fits all approach will fail when physical conditions change, and reliable reception of RF data is required. To design and implement a reliable RF system a detailed knowledge of the users' requirements and environment in which it is expected to operate is critical. In this research the operating range was limited to < 5km for the transmission of image data, within the constraints imposed by the regulatory bodies. A large image format (QVGA) was chosen which allowed good detailed to be discerned. The terrain chosen to operate within is loosely described as semi-urban, i.e., terrain where there is not a high density of buildings. The requirement to transmit large amounts of data (images) using IoT type systems contradicts the designed use of these systems. However, for certain use cases, these systems can be 'persuaded' to deliver the required reliability of service. This research focused on the development of private RF networks, using off-the- shelf components because of the lack of flexibility imposed by CIoT systems with respect to data volumes and time constraints.

The hardware chosen for the research allowed the selection of different modulation schemes for different applications. i.e., higher bandwidth systems

(ESP-NOW) for shorter range applications, and lower bandwidth systems such as DASH7® or LoRa™ for longer range applications. The choice of scheme was determined by the life-time requirements of a battery powered end-device and the required image detail needed. A system designed for use in disaster monitoring applications would have different battery longevity requirements than a system designed for use in applications, such as crack monitoring in buildings or bridges. The ability to read individual pixels of an image taken by the end-device suggests the possibility of further edge-device processing and reducing the amount of actual data transmitted. For example, it is entirely feasible to monitor a forest for fire outbreaks by programming the end-device to recognise a ‘centrum’ of brightness and transmit an alarm if a fire is detected.

The ability to select a ROI in an image allows detailed image information to be transmitted with a reduced payload. This might be necessary to maximise the battery lifetime of an end-device for long range transmission when using LoRa modulation scheme. At the time of writing this thesis, there has been no existing work which has used ESP-NOW protocol to transmit a digital image. The careful design of an image compression technique which has low cost in terms of required resources, and acceptable results (in terms of PSNR), has been achieved. The flexibility of use of this technique i.e., ROI selection which facilitates a large reduction in the amount of data to be transmitted, makes this technique eminently suitable for use across a variety of RF WLANs. The techniques developed during this research have advanced the capabilities of image transmission as they are system independent and can be used with a variety of system hardware configurations.

6.1 Conclusion

The main hypothesis of this research was that low-power resource-constrained devices are suitable for the transmission of image data over low-power RF networks for certain use cases. A flexible low-power RF network was successfully designed. This enabled the choice of modulation scheme to be selected for a particular use case. Moreover, a successful image compression algorithm was designed and implemented. Additionally, the ability to manipulate image data directly has been achieved. This enabled a ROI to be selected and transmitted with or without additional information included in the image data. System functionality has been successfully tested using ‘real’ hardware in an end-to-end configuration.

It can be concluded that given the correct choice of hardware and modulation scheme that it is feasible to expand the use case for such systems to include the transmission of digital imagery. Given the regulatory constraints imposed in each country for ISM bands, the longer the required transmission range, the less volume of data/unit time can be transmitted. This would limit the longer transmission range to the choice of either selecting an ROI or smaller image format.

6.2 Future work

The hardware selected for development during this research was underutilised for this application. Although not used during this research, the ESP32 is

supplied with FreeRTOS (Real Time Operating System) on-board. This can enable for more robust firmware development. Facial recognition software also exists on the ESPCAM but was not used during this research. The issue of power consumption has not been addressed in any detail but should be addressed to enable full characterisation of the image transmission system, and hence battery lifetime expectation. A suitable methodology for measuring power consumption is outlined by Congduc Pham [16] and could be used as a reference for future work. The ESP32 comes with a ULP (Ultra Low Power) microprocessor embedded as standard and shows promise for reducing power consumption to a minimum. However, this microprocessor can only be programmed using assembly language and minimum information exists regarding this.

Regarding the RF network, a collision detection and avoidance mechanism needs to be evaluated in more detail. This would allow for a more granular knowledge of the expected performance of the network to be gained. Because each protocol selected and evaluated during this research allowed bi-directional communication, a future development should explore this capability and its impact on transmission restrictions and power consumption. A detailed evaluation of antenna performance and interfacing would allow real-world testing and analysis of a complete system to be performed. Access to the correct test equipment is needed and is a requirement if theoretical evaluations are to be realised.

The software developed during this research for the reconstruction of the received image data also needs improvement to facilitate a more robust mechanism to differentiate between multiple incoming images. The current

method used to enable the selection of a ROI requires that the necessary coordinates needed to select a region must be hardcoded into each end device when the firmware is flashed to the device. This selection procedure could be enhanced to allow remote selection of a particular region when bi-directional communications with each end-device is enabled in future iterations. This capability will greatly enhance the flexibility of each end-device to minimise the volume of data to be transmitted and hence will allow tuning of the battery expected lifespan for standalone powered devices. The current software used is written in Java and has some limitations and 'quirkiness' in use for this application. A graphical user interface (GUI) written in a different programming language such as Python using QT or TKinter would further enhance the system design and user friendliness. The ability to manipulate raw image data directly allows for the expansion of use cases where evaluation of pixel brightness could be used for example to control a tracking mechanism for use in astronomical telescopes. Another possible use would be for automatic counting of bright patches within an image of interest (medical) where brightness could indicate abnormal growth. The semi-lossless compression technique used here is suitable for use where the loss of some data does not impact on our perception of the final results received and could be extended for use with the transmission of sound.

References

[1] DASH7 Alliance, “DASH7 ALLIANCE,” [Online]. Available: https://dash7-alliance.org/. [Accessed 24 May 2021].

[2] “3GPP,” [Online]. Available: https://www.3gpp.org/about-3gpp/about-3gpp. [Accessed 16 May 2021].

[3] Olof Liberg, “3GPP,” 24 October 2017. [Online]. Available: https://www.3gpp.org/news-events/3gpp-news/1906-c_iot.

[4] B. V. N. M. a. A. G. R. Ratasuk, “NB-IoT system for M2M communication,” in *2016 IEEE Wireless Communications and Networking Conference*, Doha, Qatar, 2016.

[5] R. Sun, “Design and performance of unlicensed NB-IoT,” in *Proceedings of the International Symposium on Wireless Communication Systems*, 2019.

[6] N. M. D. B. A. G. Rapeepat Ratasuk, “LTE-M Evolution Towards 5G Massive MTC,” in *[1] R. Ratasuk, N. Mangalvedhe, D. Bhatoolaul, and A. Ghosh, “LTE-M Evolution Towards 5G Massive MTC,” 2017 IEEE Globecom Work. GC Wkshps 2017 - Proc., vol. 2018-January, pp. 1–6, 2018, doi: 10.1109/GLOCOMW.2017.8269112.*, 2018.

[7] Qualcomm, July 2018. [Online]. Available: https://www.qualcomm.com/media/documents/files/whitepaper-leading-the-lte-

iot-evolution-to-connect-the-massive-internet-of-things.pdf. [Accessed 4th March 2022].

[8] [Online]. Available: https://lora-alliance.org/. [Accessed 20/5/21 May 2021].

[9] “TheThingsnetwork,” [Online]. Available: https://www.thethingsnetwork.org/docs/lorawan/. [Accessed 4th March 2022].

[10] ESPRESSIF, “Products>SoCs > all,” [Online]. Available: https://www.espressif.com/en/products/socs. [Accessed 23 May 2021].

[11] IEEE, “Part 11:Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007), 2012.

[12] ESPRESSIF, “ESP-NOW User Guide,” ESPRESSIF, 2016.07.

[13] IEEE, “IEEE: Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications,” no. Std 802.11-2012, pp. 1504 - 1535, 2012.

[14] ETSI, “ETSI EN 300 440 v2.2.1(2018-07),” ETSI, 2018.

[15] DASH7 ALLIANCE, “Wireless Sensor and Actuator Network Protocol V1.2,” DASH7 ALLIANCE, 2018.

[16] C.Pham, “Low-cost, Low-Power and Long-range Image Sensor for Visual Surveillance,” in *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, New York City, 2016.

[17] V. J. M. L.Makkaoui, “Fast zonal DCT-based image compression for wireless camera sensor networks,” in *2010 2nd International Conference on Image Processing Theory, Tools and Applications, IPTA 2010*, 2010.

[18] M. J. J. M. A. Ji, “LoRa-based Visual Monitoring Scheme for Agriculture IoT,” in *SAS 2019 - 2019 IEEE Sensors Applications Symposium, Conference Proceedings*, 2019.

[19] R. A.-D. A. F. F. M. A. AL-HASHEMI, “A Grayscale Semi-Lossless Image Compression Technique Using RLE,” *Journal of Applied Computer Science & Mathematics*, vol. January 2011, 2011.

[20] ETSI, “Organisation of European Frequency Spectrums Grouped by Standards Families,” ETSI, 2020. [Online]. Available: https://www.etsi.org/e-brochure/FrequencyChart/mobile/index.html#p=1. [Accessed 24 May 2021].

[21] COMREG, 2 Febuary 2018. [Online]. Available: file:///C:/Users/poconnor/Downloads/ComReg02_71-R11.pdf. [Accessed 24 May 2021].

[22] Wikimedia Foundation, Inc., “Claude Shannon,” 13 july 2021. [Online]. Available: https://en.wikipedia.org/wiki/Claude_Shannon. [Accessed 16 July 2021].

[23] H. J. David Frykskog, “Construction of RF-link budget template for transceiver modelling,” Linköping University Electronic Press, 2019.

[24] J. W. a. P. Varaiya, in *High-Performance Communication Networks*, Elsevier Inc., 2000, p. 7.2.7.

[25] Intersil, “Tutorial on Basic Link Budget Analysis,” Inersil, 1998.

[26] Gaussian Waves, “Performance comparison of Digital Modulation techniques,” 14 April 2010. [Online]. Available: https://www.gaussianwaves.com/2010/04/performance-comparison-of-digital-modulation-techniques-2/#comments. [Accessed May 2021].

[27] R. C. DIXON, in *Spread Spectrum Systems with Commercial applications*, New York, JOHN WILEY & SONS inc, 1994, pp. 3-17.

[28] O. B. A. S. N. Seller, “Low power long range transmitter”. European Patent EP 2 763 321 A1, 05 Febuary 2013.

[29] R. H. M. R. H. B. H. A. B. S. a. A. M. Ilfiyantri Intyas, “Improvement of Radar Performance Using LFM Pulse Compression Technique,” in *The 5th International Conference on Electrical Engineering and Informatics 2015*, Bali, 2015.

[30] Texas Instruments, “RF BASICS,” Texas Instruments.

[31] everythingRF, “Link budget calculator,” [Online]. Available: https://www.everythingrf.com/rf-calculators/link-budget-calculator. [Accessed 30 May 2021].

[32] Wikimedia Foundation, Inc., “Friis transmission equation,” 1 June 2021. [Online]. Available: https://en.wikipedia.org/wiki/Friis_transmission_equation. [Accessed 16 July 2021].

[33] ESPRESSIF, “ESP32 Specifications (WiFi Receive),” 21 September 2017. [Online]. Available: https://esp32.com/viewtopic.php?t=3096. [Accessed 4 6 2021].

[34] ESPRESSIF, “API Guides >> WiFi Driver,” ESPRESSIF, [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html#wi-fi-protocol-mode. [Accessed 5 6 2021].

[35] Texas Instruments, “Single Chip Very Low Power RF Transceiver datasheet (Rev.A),” Texas Instruments.

[36] Semtech, “SX1272/73 DATASHEET,” no. Rev.4, pp. 1-32, 2019.

[37] 4D SYSTEMS, “forum,” 4D SYSTEMS, May 2018. [Online]. Available: https://forum.4dsystems.com.au/node/63649. [Accessed 16 June 2021].

[38] ArduCam, “arducam.com,” [Online]. Available: https://www.arducam.com/product/arducam-2mp-spi-camera-b0067-arduino/. [Accessed 21 june 2021].

[39] Airspy.com, “airspy.com,” [Online]. Available: https://airspy.com/download/. [Accessed 21 june 2021].

[40] openocd.org, “openocd.org,” 7 March 2021. [Online]. Available: http://openocd.org/. [Accessed 21 June 2021].

[41] FTDI Chip, “ftdichip.com,” FTDI Chip, [Online]. Available: https://ftdichip.com/drivers/vcp-drivers/. [Accessed 21 June 2021].

[42] Zadig, “zadig.akeo.ie,” [Online]. Available: https://zadig.akeo.ie/. [Accessed 21 June 2021].

[43] PLatformIO.org, “docs.platformio.org,” [Online]. Available: https://docs.platformio.org/en/latest/tutorials/espressif32/arduino_debugging_unit_testing.html#setting-up-the-hardware. [Accessed 21 June 2021].

[44] Processing.org, “processing.org,” [Online]. Available: https://processing.org/tutorials/overview/. [Accessed 21 June 2021].

[45] wireshark.org, “wireshark.org,” [Online]. Available: https://www.wireshark.org/. [Accessed 21 june 2021].

[46] Espressif, “Frame Format,” [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_now.html?highlight=espnow. [Accessed 28 June 2021].

[47] Adafruit, “adafruit-gfx-graphics-library,” 14 July 2021. [Online]. Available: https://learn.adafruit.com/adafruit-gfx-graphics-library. [Accessed 14 July 2021].

[48] DIGI, “https://www.digi.com/resources/documentation/digidocs/pdfs/90002126.pdf,” March 2019. [Online]. Available: https://www.digi.com. [Accessed 28 May 2021].

[49] airspayce.com, “RadioHead,” [Online]. Available: http://www.airspayce.com/mikem/arduino/RadioHead/index.html. [Accessed 24 June 2021].

Appendix

C++ Code for end-device:

```
#include <Arduino.h>
#include <Adafruit_GFX.h>
#include "WiFi.h"
#include "esp_camera.h"
#include "config.h"
#include "pin_config.h"
#include <esp_wifi.h>
#include <fstream>
//#include "soc/soc.h"              // Disable Brownout problem
//#include "soc/rtc_cntl_reg.h"       // Disable Brownout problem
//-----------------------------------------------
// ESPNow stuff
//-----------------------------------------------
#include <esp_now.h>
// Global copy of slave
esp_now_peer_info_t slave;
#define CHANNEL 1
#define PRINTSCANRESULTS 0
#define DELETEBEFOREPAIR 0
#define BLACK 0x0000
#define WHITE 0xffff
uint16_t cnt = 0;
uint32_t  i = 0;
uint32_t j = 0;
uint8_t k = 0;
uint8_t k1 = 0;
uint32_t p = 0;
boolean ACKED = 0;             //Used to make sure the first frame has been acked - used for
synchronisation with slave.
uint16_t Frame = 0;            // DITTO
uint8_t CurrentID;             // DITTO

uint8_t Count = 1;
#define BUFFER_SIZE 245
uint8_t NewArray[BUFFER_SIZE * 2];
uint8_t dataBuf[BUFFER_SIZE];
uint8_t dataToSend[245];        // packet structure to transmit goes here. 245 (WAS 243 for raw image
testing)
uint32_t frameStart = 0;
uint32_t frameEnd = 240;
const uint8_t frame_length = 240; // length of payload 240
uint8_t frame_length1 = 240 ;
uint8_t randomNumber;
boolean first = false;
//---------------------------------------------------------------
//  ROI variables
uint16_t H = 80;   // Width of ROI Box
uint16_t W = 320; // Width of Image (320)
uint16_t V = 60;  // Vertical co-ord top left of box max 120.
uint32_t h = 0;   // top left hand corner of box in pixel count.
uint32_t r = 0;   // Bottom right hand corner of box in Pixel count
//---------------------------------------------------------------
//---------------------------------------------------------------
boolean QLOw = false;          // image quality setting
boolean QMed = true;
boolean Qhigh = false;
uint8_t mask;                // mask used for quality setting
uint8_t many;                 // used in compression function.
//---------------------------------------------------------------
```

```
//uint8_t Payload[BUFFER_SIZE];

using namespace std;

//ofstream outputFile;
//ofstream fs;

//std::string filename = "/home/paschal/Documents/Image.csv";


// Init ESP Now with fallback
void InitESPNow() {
  //WiFi.disconnect();
  if (esp_now_init() == ESP_OK) {
    Serial.println("ESPNow Init Success");
  }
  else {
    Serial.println("ESPNow Init Failed");
    // Retry InitESPNow, add a counte and then restart?
    // InitESPNow();
    // or Simply Restart
    ESP.restart();
  }
}

// callback when data is sent from Master to Slave
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {
  char macStr[18];
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
   mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
  Serial.print("Last Packet Sent to: "); Serial.println(macStr);
  Serial.print("Last Packet Send Status: "); Serial.println(status == ESP_NOW_SEND_SUCCESS ?
"Delivery Success" : "Delivery Fail");
}

//-------------------------------------------------------
// Pixel drawing stuff here.
//-------------------------------------------------------
class aFrameBuffer : public Adafruit_GFX {
  public:
    uint8_t *buffer = NULL;
    uint32_t size1;
    int w;
    int h;
    aFrameBuffer(int16_t ww, int16_t hh): Adafruit_GFX(ww,hh){
      w = ww;
      h = hh;
      size1 = h * w;
    }
//------------------------------
// where to draw
//------------------------------
void setBuffer(uint8_t *b){
       buffer = b;
}

//------------------------------
// Drawing a Pixel
//------------------------------
void drawPixel(int16_t x, int16_t y, uint16_t color){
  if(x<0 || x>= w || y<0 || y>=h)
    return;
  buffer[x + y * w] = color;
}
};

void setup()
{
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);  //Set device in STA mode to begin with
  WiFi.disconnect();
  Serial.print("STA MAC: "); Serial.println(WiFi.macAddress());// This is the mac address of the Master in
Station Mode
  InitESPNow();// Init ESPNow with a fallback logic
```

```
// Once ESPNow is successfully Init, we will register for Send CB to
// get the status of Trasnmitted packet
  esp_now_register_send_cb(OnDataSent);

  //------------------------------
  //  Camera
  //------------------------------
  // Tuenoff the Brownout dectector
  //WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
  camera_config_t config;
  config.pin_pwdn = PIN_PWDN;
  config.pin_reset = PIN_RESET;
  config.pin_xclk = PIN_XCLK;
  config.pin_sscb_sda = PIN_SIOD;
  config.pin_sscb_scl = PIN_SIOC;
  config.pin_d7 = PIN_D7;
  config.pin_d6 = PIN_D6;
  config.pin_d5 = PIN_D5;
  config.pin_d4 = PIN_D4;
  config.pin_d3 = PIN_D3;
  config.pin_d2 = PIN_D2;
  config.pin_d1 = PIN_D1;
  config.pin_d0 = PIN_D0;
  config.pin_vsync = PIN_VSYNC;
  config.pin_href = PIN_HREF;
  config.pin_pclk = PIN_PCLK;

  config.xclk_freq_hz = 20000000;
  config.ledc_timer = LEDC_TIMER_0;
  config.ledc_channel = LEDC_CHANNEL_0;
  //config.pixel_format = PIXFORMAT_JPEG;
  config.pixel_format = PIXFORMAT_GRAYSCALE;
  //config.frame_size = FRAMESIZE_SVGA;

  config.frame_size = FRAMESIZE_QVGA;
  config.jpeg_quality = 10;
  //config.fb_count = 2;
  config.fb_count = 1;

  esp_err_t err = esp_camera_init(&config);

  if (err != ESP_OK)
     Serial.printf("Camera init failed with error 0x%x", err);

       //---------------------------------
       // Show Camera model
       //---------------------------------
     sensor_t *s = esp_camera_sensor_get();
     switch (s->id.PID)
     {

       case OV9650_PID:
          Serial.println("OV9650 sensor found.");
          break;
       case OV7725_PID:
          Serial.println("OV7725 sensor found.");
          break;
       case OV2640_PID:
          Serial.println("OV2640 sensor found.");
          break;
       default:
          Serial.println("Unknown sensor found.");
         break;
      }
}

void deletePeer() {
  esp_err_t delStatus = esp_now_del_peer(slave.peer_addr);
  Serial.print("Slave Delete Status: ");
  if (delStatus == ESP_OK) {
    // Delete success
    Serial.println("Success");
  } else if (delStatus == ESP_ERR_ESPNOW_NOT_INIT) {
    // How did we get so far!!
    Serial.println("ESPNOW Not Init");
```

```
  } else if (delStatus == ESP_ERR_ESPNOW_ARG) {
    Serial.println("Invalid Argument");
  } else if (delStatus == ESP_ERR_ESPNOW_NOT_FOUND) {
    Serial.println("Peer not found ?.");
  } else {
    Serial.println("Not sure what happened");
  }
}

// Check if the slave is already paired with the master.
// If not, pair the slave with master
bool manageSlave() {
  if (slave.channel == CHANNEL) {
    if (DELETEBEFOREPAIR) {
      deletePeer();
    }

   // Serial.print("Slave Status: ");
    // check if the peer exists
    bool exists = esp_now_is_peer_exist(slave.peer_addr);
    if ( exists) {
      // Slave already paired.
      Serial.println("Already Paired");
      return true;
    } else {
      // Slave not paired, attempt pair
      esp_err_t addStatus = esp_now_add_peer(&slave);
      if (addStatus == ESP_OK) {
        // Pair success
        //Serial.println("Pair success");
        return true;
      } else if (addStatus == ESP_ERR_ESPNOW_NOT_INIT) {
        // How did we get so far!!
        //Serial.println("ESPNOW Not Init");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_ARG) {
        //Serial.println("Invalid Argument");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_FULL) {
       // Serial.println("Peer list full");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_NO_MEM) {
        //Serial.println("Out of memory");
        return false;
      } else if (addStatus == ESP_ERR_ESPNOW_EXIST) {
       // Serial.println("Peer Exists");
        return true;
      } else {
       // Serial.println("Not sure what happened");
        return false;
      }
    }
  } else {
    // No slave found to process
    Serial.println("No Slave found to process");
    return false;
  }
}


void initBroadcastSlave() {
          // clear slave data
          memset(&slave, 0, sizeof(slave));
          for (int ii = 0; ii < 6; ++ii) {
                    slave.peer_addr[ii] = (uint8_t)0xff;
          }
          slave.channel = CHANNEL; // pick a channel
          slave.encrypt = 0; // no encryption
          manageSlave();
}

// Scan for slaves in AP mode

void ScanForSlave() {
  int8_t scanResults = WiFi.scanNetworks();
```

```
  // reset on each scan
  bool slaveFound = 0;
  memset(&slave, 0, sizeof(slave));

  //Serial.println("");
  if (scanResults == 0) {
    Serial.println("No WiFi devices in AP Mode found");
  } else {
    Serial.print("Found "); Serial.print(scanResults); Serial.println(" devices ");
    for (int i = 0; i < scanResults; ++i) {
      // Print SSID and RSSI for each device found
      String SSID = WiFi.SSID(i);
      int32_t RSSI = WiFi.RSSI(i);
      String BSSIDstr = WiFi.BSSIDstr(i);

      if (PRINTSCANRESULTS) {

        Serial.print(i + 1);
        Serial.print(": ");
        Serial.print(SSID);
        Serial.print(" (");
        Serial.print(RSSI);
        Serial.print(")");
        Serial.println("");

      }
      delay(10);
      // Check if the current device starts with `Slave`
      if (SSID.indexOf("Slave") == 0) {
        // SSID of interest
        //Serial.println("Found a Slave.");
        //Serial.print(i + 1); Serial.print(": "); Serial.print(SSID); Serial.print(" ["); Serial.print(BSSIDstr);
Serial.print("]"); Serial.print(" ("); Serial.print(RSSI); Serial.print(")"); Serial.println("");
        // Get BSSID => Mac Address of the Slave
        int mac[6];
        if ( 6 == sscanf(BSSIDstr.c_str(), "%x:%x:%x:%x:%x:%x",  &mac[0], &mac[1], &mac[2], &mac[3],
&mac[4], &mac[5] ) ) {
          for (int ii = 0; ii < 6; ++ii ) {
            slave.peer_addr[ii] = (uint8_t) mac[ii];
          }
        }

        slave.channel = CHANNEL; // pick a channel
        slave.encrypt = 0; // no encryption

        slaveFound = 1;
        // we are planning to have only one slave in this example;
        // Hence, break after we find one, to be a bit efficient
        break;
      }
    }
  }

  if (slaveFound) {
    Serial.println("Slave Found, processing..");
  } else {
    Serial.println("Slave Not Found, trying again.");
  }

  // clean up ram
  WiFi.scanDelete();
}


void sendData() {

      const uint8_t *peer_addr = slave.peer_addr;
      esp_err_t result = esp_now_send(peer_addr, dataToSend, sizeof dataToSend);
      Serial.print("Send Status: ");
      if (result == ESP_OK) {
        if (cnt == 0){
          ACKED = true;
        }
        Serial.println("Success");
      } else if (result == ESP_ERR_ESPNOW_NOT_INIT) {
```

```
      // How did we get so far!!
      Serial.println("ESPNOW not Init.");
    } else if (result == ESP_ERR_ESPNOW_ARG) {
      Serial.println("Invalid Argument");
    } else if (result == ESP_ERR_ESPNOW_INTERNAL) {
      Serial.println("Internal Error");
    } else if (result == ESP_ERR_ESPNOW_NO_MEM) {
      Serial.println("ESP_ERR_ESPNOW_NO_MEM");
    }
    else if (result == ESP_ERR_ESPNOW_NOT_FOUND)
    {
      Serial.println("Peer not found.");
    }
    else
    {
      Serial.println("Not sure what happened");
    }

    //delay(1);
    //   else
    //       {
    //     Serial.printf("\r\nDID NOT SEND....");
    //       }
}



void Packet(){
         dataToSend[0] = randomNumber;           // frame I.D.
         dataToSend[1] = frameStart>>24;         // frame count number.
         dataToSend[2] = frameStart>>16;
         dataToSend[3] = frameStart>>8;
         dataToSend[4] = frameStart & 0Xff;
         //if (frameEnd >= 76800){
         //  dataToSend[0] = 127;                // use this as end of Image data ( I.D should be <= 10)
         //}
         if (i >= 76800){
           dataToSend[0] = 127;                // use this as end of Image data ( I.D should be <= 10)
           Serial.println("End of image");
           Serial.printf("i = %d", i);
         }
         if (first) {
         dataToSend[0] = randomNumber;           // frame I.D.
         dataToSend[1] = 0;                    // frame count number.
         dataToSend[2] = 0;
         dataToSend[3] = 0;
         dataToSend[4] = 0;
         first = false;                        // only do for first packet.
         }
/*
         Serial.printf("FrameStart =  %d", dataToSend[1]);
         Serial.printf (", %d",dataToSend[2]);
         Serial.printf(", %d", dataToSend[3]);Serial.printf (", %d", dataToSend[4]);
         Serial.println("");
         */
         //frameStart = frameEnd;
         //frameEnd = frameStart + 240;      // was +200
         //-------------------------------------------------------------------;
         //       Payload Structure     (Frame)                               ;
         //-------------------------------------------------------------------;
         //Frame I.D | frameStart  |                                         ;
         // 1 Byte   | 4 Bytes     |        240 Bytes                         ;
         //-------------------------------------------------------------------;
         p = 0;
         while (p < 240 && i <= 76800) // was 245
         {
           dataToSend[p + 5] = NewArray[p];
           p++; // number of elements to copy
           //Serial.print(dataToSend[i]);
         }
        frameStart = i;                        // index into fb->buf.
          //i = 0; // reset index for next value of cnt
        p = 0;
        /*
          if ( frameEnd > 76800){
```

```
          for (i = 0; i < 245;i++){
            Serial.print(dataToSend[i]);
            Serial.print(",");
          }
          }
          */

        //i = 0;
}


void PacketCompressed(){
          dataToSend[0] = randomNumber;           // frame I.D.
          dataToSend[1] = frameStart>>24;         // frame count number.
          dataToSend[2] = frameStart>>16;
          dataToSend[3] = frameStart>>8;
          dataToSend[4] = frameStart & 0Xff;
          //if (frameEnd >= 76800){
          //  dataToSend[0] = 127;                // use this as end of Image data ( I.D should be <= 10)
          //}
          if (i >= 76800){
            dataToSend[0] = 127;                // use this as end of Image data ( I.D should be <= 10)
            Serial.println("End of image");
            Serial.printf("i = %d", i);
          }
          if (first) {
          dataToSend[0] = randomNumber | 160;      // frame I.D.
          dataToSend[1] = 0;                       // frame count number.
          dataToSend[2] = 0;
          dataToSend[3] = 0;
          dataToSend[4] = 0;
          //Serial.printf("data ID = %d",dataToSend[0]&0xf0);
          first = false; // only do for first packet.
          }
/*
          Serial.printf("FrameStart =  %d", dataToSend[1]);
          Serial.printf (", %d",dataToSend[2]);
          Serial.printf(", %d", dataToSend[3]);Serial.printf (", %d", dataToSend[4]);
          Serial.println("");
          */
          //frameStart = frameEnd;
          //frameEnd = frameStart + 240;      // was +200
          //-------------------------------------------------------------------;
          //        Payload Structure     (Frame)                              ;
          //-------------------------------------------------------------------;
          //Frame I.D | frameStart  |                                          ;
          // 1 Byte   | 4 Bytes     |         240 Bytes                         ;
          //-------------------------------------------------------------------;
          p = 0;
          while (p < 240 && i <= 76800) // was 245
          {
            dataToSend[p + 5] = NewArray[p];
            p++; // number of elements to copy
            //Serial.print(dataToSend[i]);
          }
         frameStart = i;                          // index into fb->buf.
          //i = 0; // reset index for next value of cnt
         p = 0;
         /*
          if ( frameEnd > 76800){
          for (i = 0; i < 245;i++){
            Serial.print(dataToSend[i]);
            Serial.print(",");
          }
          }
          */

        //i = 0;
}

void PacketROI(){
          dataToSend[0] = randomNumber;           // frame I.D.
          dataToSend[1] = frameStart>>24;         // frame count number.
          dataToSend[2] = frameStart>>16;
          dataToSend[3] = frameStart>>8;
```

```
          dataToSend[4] = frameStart & 0Xff;
          //if (frameEnd >= 76800){
          //  dataToSend[0] = 127;                // use this as end of Image data ( I.D should be <= 10)
          //}
          if (h >= r){        // was i
            dataToSend[0] = 127;                // use this as end of Image data ( I.D should be <= 10)
            Serial.println("End of image");
            //Serial.printf("i = %d", i);
          }
          if (first) {
          dataToSend[0] = randomNumber | 96;      // frame I.D. -> 96 indicates a ROI frame
          dataToSend[1] = frameStart>>24;         // frame count number.
          dataToSend[2] = frameStart>>16;
          dataToSend[3] = frameStart>>8;
          dataToSend[4] = frameStart & 0Xff;
         // first = false;                        // only do for first packet.
          //Serial.printf("DataToSend[0] = %d",dataToSend[0]);
          }
/*
         Serial.printf("FrameStart =  %d", dataToSend[1]);
         Serial.printf (", %d",dataToSend[2]);
         Serial.printf(", %d", dataToSend[3]);Serial.printf (", %d", dataToSend[4]);
         Serial.println("");
         */
         //frameStart = frameEnd;
         //frameEnd = frameStart + 240;      // was +200
         //-------------------------------------------------------------------;
         //       Payload Structure     (Frame)                               ;
         //-------------------------------------------------------------------;
         //Frame I.D | frameStart  |                                          ;
         // 1 Byte   | 4 Bytes     |         240 Bytes                        ;
         //-------------------------------------------------------------------;
         p = 0;
         if(first){
         while (p < 240 && h < r) // <r
         {
           dataToSend[p + 5] = NewArray[p];
           //Serial.print(dataToSend[p]);
           p++; // number of elements to copy

           //Serial.print(dataToSend[i]);
         }
         //first = false;
         //p = 0;
         }
         else if (!first){
           p = 0;
           while (p < 240 && h < r) // < r
           {
             dataToSend[p+5] = NewArray[p];
             //Serial.print(dataToSend[p]);
             //Serial.print(",");
             p++; // number of elements to copy
             //Serial.print(dataToSend[i]);
         }
         }
         //frameStart = h + j;
         //Serial.printf("frameStart = %d", frameStart);
         //Serial.println(""); // index into fb->buf.
            //i = 0; // reset index for next value of cnt
         p = 0;
         first = false;
        // for (i = 0; i < 245; i++)
         //{
         //  Serial.print(dataToSend[i]);
         //  Serial.print(",");
          //}


        //i = 0;
}



void ROI(){
```

```
  camera_fb_t *fb = esp_camera_fb_get();     //Snap picture - DON'T TAKE SHOT IF SENDIND
MULTIPLE VERSIONS OF IMAGE - USE FULL IMAGE DATA
  first = true;
  srand((unsigned) time(0));                // generate a random number 1 - 10, to use as frame I.D in payload
    //int randomNumber;
  for (int index = 0; index < 1; index++) {
  randomNumber = (rand() % 10) + 1;
                      }
  if (fb) { //---------------------------- UNCOMMENT FOR NORMAL OPERATION
    j = 0;
    h = (V * W) + H;
    r = (h + H) + (V * W);
    frameStart = h;
    Serial.printf("frameStart = %d", h);
    Serial.println("");
    Serial.printf("corner = %d", r);
    Serial.println("");
    while (h < r)  // r
    {
      while (j < 240)
      {
        for (i = h; i < h + 80; i++)  // h+80
        {
          NewArray[j] = fb->buf[i];
          j++;
        }
        Serial.printf("h1 = %d", h);
        h = h + W;//W
        //h = h + 321;

        Serial.println("");
        //Serial.printf("j1 = %d", j);Serial.println("");
      }
      //Serial.printf("j = %d", j);Serial.println("");
      j = 0;
      //Serial.printf("h = %d", h);Serial.println("");
      PacketROI();
      sendData();
    }
      //frameStart = h + j;

  esp_camera_fb_return(fb); // release camera buffer.
  }      // ---------------------------------------------- UNCOMMENT FOR NORMAL OPERATION
  //esp_camera_fb_return(fb); // release camera buffer.
}

void compress(){
     camera_fb_t *fb = esp_camera_fb_get();     //Snap picture
     if (fb) {
       //-----------------------------------------------
       // Write to main camera buffer directly
       OSD.setBuffer(fb->buf);
       OSD.setTextSize(1);
       OSD.setTextColor(BLACK,WHITE);
       for (i = 80; i < 320;i += 80){
         OSD.drawFastVLine(i, 0, 240, WHITE);
       }
       for (i = 80; i < 240;i+= 80){
         OSD.drawFastHLine(0, i, 320, WHITE);
       }
       OSD.setCursor(20, 220);
       OSD.print("Bat. Voltage = 3.4v");
       //-----------------------------------------------

       first = true;                          // first frame indicator
       Serial.println("Snap Taken");             // COMPRESSION AND RLE STARTS HERE.
       srand((unsigned) time(0));               // generate a random number 1 - 10, to use as frame I.D in
payload
       //int randomNumber;
       for (int index = 0; index < 1; index++) {
       randomNumber = (rand() % 10) + 1;
        }
/*
     while(frameEnd <76800){
```

```
      while (j < 240){                              //
       for (i = frameStart; i < frameEnd;i++){          // free up last 4 bits of each value
         k = fb->buf[i];                  // i.e remap values.
         k = k >> 4;
         k = k << 4;
         dataBuf[p] = k;                   // HAVE 240 BYTES FROM CAMERA BUFFER
         //Serial.print(dataBuf[p]);
         p++;
         }
         */
//debug
/*
        for (i = 0; i <400; i++)
        {
        Serial.print(fb->buf[i]);
        Serial.print(",");
        }
        */
        i = 0;
        /*
        p = 0;
        i = 0;
        Count = 1;
      while(i<240  ){            // was i<200 && j<240

        while (dataBuf[i] == dataBuf[i + 1])
         {       // is current value = next value ?
         Count++; // if yes the inc. Count.
         i++;     // point to next pixel value.
         }
         if (Count <= 15 ){                 // count only has 4-bits, so max 15

           uint8_t val = dataBuf[i]; //
           val = val + Count;
           NewArray[j] = val; // upper 4-bits is greyscale val, lower 4-bits is repetition number.;
           //Serial.print(NewArray[j]);
           j++; // next loccation.
           //Serial.printf("count is less than 15, j =  %d", j);
           //Serial.printf("   ,count  =  %d", Count);
           i++;
           //Serial.printf("   ,i  =  %d", i);
           //Serial.println(" ");

         }
         else{                          // if we have more than 15 repitions then put additional values in next
location.

           uint8_t Quot = Count / 15;           // Quotient (number of repititions / 15)
           uint8_t Rem = Count % 15;            // modulus ( remainder)

           while (Quot > 0 ){                  // repeat until divisor = 0.
             uint8_t val1 = dataBuf[i];
             val1 = val1 + 15;
             NewArray[j] = val1;
             //NewArray[j] = (dataBuf[i + 15]);    // add 15 to value in NewArray
             Quot--;                  // subtract 1 from quotient until quotient = 0.
             j++ ;                          // point to next location in newArray.
             val1 = dataBuf[i];
             val1 = val1 + Rem;
             NewArray[j] = val1;
             //NewArray[j] = (dataBuf[i]) + Rem; // add remainder to current location.
             j++ ;
             i++;
 */
       //Serial.printf("count is more than 15, j =  %d", j);
       /*
            Serial.printf("   , Count  =  %d", Quot);
            Serial.printf("   , Quot is  =  %d", Quot);
            Serial.printf("  ,i  =  %d", i);
            Serial.println(" ");
            */
       /*
          }

         }
```

```
  Count = 1;

}


  frameStart = frameEnd;
  frameEnd = frameStart + 240;      // was +200

  Serial.printf("frameStart = %d", frameStart);
  Serial.println("");
  Serial.printf("frameEnd = %d", frameEnd);
  Serial.println("");

 }
 */
 /*
  Serial.println(" ");
  Serial.printf("Size of NewArray = %d", j);
  Serial.println(" ");
  Serial.printf("Index into buf = %d", i);
  Serial.println(" ");


  frameStart = (frameStart - 240) + i;      // was - 200
  frameEnd = frameStart + 240;              // was +200
  Serial.printf("So.. new frameStart = %d", frameStart);
  Serial.printf("   ,   new frameEnd = %d", frameEnd);
  Serial.println("");

  for (i = 0; i < j;i++){
   Serial.print(NewArray[i]);
   Serial.print(",");
  }
  */
 while (i <= 76800)
 {
  while (j < 240 && i <= 76800)
  {
   //p = fb->buf[i];
   //Serial.printf("buf = %d", p);Serial.print(",");
   //Serial.printf("buf+1 = %d", fb->buf[i+1]);Serial.print(",");
   if (Qhigh ){
   mask = 0xf8; //Serial.print(","); map to 32 values
   many = 7;
   }

   else if(QMed){
   mask = 0xf0; //Serial.print(","); // map to 16 values
   many = 15;
   }
   k = fb->buf[i] & mask;
   //Serial.printf("start i = %d", i);Serial.print(",");
   //k1 = fb->buf[i + 1] & 0xf0;
   //Serial.printf("k = %d", k);
   //Serial.print(",");
   //Serial.printf("k1 = %d", fb->buf[i + 1] & 0xf0);
   //while (k == k1 && i <= 76800)

   while (k == (fb->buf[i+1] & mask)&& i <= 76800)  // map to Qhigh, Qmed,Qlow values
   {
    Count++;
    i++;
    //k = fb->buf[i] & 0xf0;
    //k1 = fb->buf[i + 1] & 0xf0;
    //Serial.printf("Count = %d", Count);Serial.print(",");
   }

   if (Count <= many && i <= 76800)
   //if (Count <= 15 && i <= 76800)
   {
    NewArray[j] = k + Count;
    j++;
    //i++;
    if(Count == 0){
     i++;
```

```
            }
            else{
              i++;
              //i = i + Count;
              //i = Count + 1;
            }
            //Serial.printf("(i <= 15) = %d", i);Serial.print(",");
            //Serial.printf("Count1 = %d", Count);
          }
          else if (Count > many && i <= 76800)
          //else if (Count > 15 && i <= 76800)
          {
            uint8_t Quot = Count / many;
            uint8_t Rem = Count % many;
            //uint8_t Quot = Count / 15;
            //uint8_t Rem = Count % 15;
            while (Quot > 0)
            {

              NewArray[j] = k + many;
              //NewArray[j] = k + 15;
              Quot--;
              j++;
              if(Quot == 0 && Rem >0){
              NewArray[j] = k + Rem;
              j++;
              }
              //i++;
              //i = i + Count;
              //i = Count + 1;
              //Serial.printf("Count2 = %d", Count);
              //Serial.printf("(i >= 15) = %d", i);Serial.print(",");
            }
            //Serial.println("");
            i++;
          }

          Count = 1;
        }

        //Serial.printf(" i = %d", i);
        j = 0;
        /*
        for (p = 0; p < 100; p++)
        {
          Serial.print(fb->buf[p]);
          Serial.print(",");
        }
        p = 0;
        Serial.println("NewArray");
        for (p = 0; p < 100; p++)
        {
         Serial.print(NewArray[p]);
         Serial.print(",");
        }
        */
        p = 0;
        Serial.printf("End  i = %d", i);
        PacketCompressed();
        sendData();

    }

    }
    i = 0;
    frameStart = 0;
    frameEnd = 240;     // was = 200

    esp_camera_fb_return(fb); // release camera buffer.
}

void fullImage(){
    camera_fb_t *fb = esp_camera_fb_get(); //Snap picture
    //Serial.println(fb->width);
    //Serial.println(fb->height);
```

```
    //Serial.println(fb->format);

    if (fb) {
       //Serial.println("Snap Taken");


       srand((unsigned) time(0));                // generate a random number 1 - 10, to use as frame I.D in
payload
       int randomNumber;
       for (int index = 0; index < 1; index++) {
       randomNumber = (rand() % 10) + 1;
       }
       uint16_t num_of_frames = (fb->len / frame_length);
        // for QVGA - 320X240 Pixels, using greyscale B&W = 76800 Bytes.
        // each frame is 240 bytes long.
        // Number of frames = 76800/240 = 320 frames.

       for (cnt = 0; cnt < num_of_frames+1; cnt++) // frames+1 otherwise only 383 packets sent.
       {
        /*--------------------------  Testing only ---------------------------------
        Serial.println("   ");
        Serial.printf("    Next Frame: %d", cnt);
        Serial.print("  ");
        Serial.printf("frameStart %d", frameStart);
        Serial.printf("        :frameEnd %d", frameEnd);
        Serial.println("");
        ------------------------------------------------------------------------
        */
        dataToSend[0] = randomNumber | 144;     // frame I.D., 144 is Raw image indicator
        dataToSend[1] = highByte(cnt);          // frame count number.
        dataToSend[2] = lowByte(cnt);
        dataToSend[3] = 0;
        dataToSend[4] = 0;


        if(frameEnd >= fb->len){
          frameEnd = fb->len;
          frame_length1 = (frameEnd - frameStart);

        }
        else{
          frame_length1 = frame_length;
        }

        while (i < frame_length1 )
        {
         dataToSend[i + 5] = fb->buf[frameStart+i];
         i++; // number of elements to copy
        }

        i = 0; // reset index for next value of cnt.


        /*------------------------------- For testing only ----------------------------
        Serial.println("Compose ");
        //for (k = 0; k < sizeof(dataToSend); k++)
        for (k = 0; k < frame_length1+3; k++)
        {
          Serial.print(dataToSend[k]);
        } //  }
        Serial.printf("      Number of bytes = %d", k);
        Serial.println("          ");
        //------------------------------------------------------------------------
        */
        //************          // TEST CODE STARTS HERE

        //Frame = (dataToSend[1] << 8) + dataToSend[2];
        /*
      if (Frame == 0) // The first Frame of an Image
        {
        CurrentID = (dataToSend[0]); //Store ID of first frame

      }
        //Serial.printf("Current ID = %d", CurrentID);
        //Serial.println("");
```

```
          if (Frame <= 384)
          { // Check if it is from the same photo (i.e. same I.D)

            while (i < frame_length1 ) // -- was frameEnd .... Total data length = 203 ( image data = 200 - I.D
& Frame count)
            //while ( i < (fb->len)-(fb->len-frameStart))
            {
              Payload[j] = dataToSend[i + 3];
              i++; // number of elements to copy
              j++; // increment pointer into Payload array
            }

            i = 0; // reset index
            /*
            ------------------------------  Testing only -------------------------------------------
            Serial.println("Decompose");                      //test
            //for (k = frameStart;k < frameEnd && k <= fb->len;k++){          // test
            for (k = frameStart;k < (frameStart + frame_length1) && k <= fb->len;k++){

              Serial.print(Payload[k]);
            }
            Serial.printf("       Number of bytes = %d", frameEnd-frameStart);
            //-------------------------------------------------------------------------------------

            }


        else
        {
          //Serial.println("Received Packet from different Image.");
          //j = 0;
          //Frame = 0;
        }

        frameStart = frameEnd ;
        frameEnd = frameStart + (frame_length1);

      }

     if (Frame == 383)               // number of frames for complete image (76800 / 200) = 0 -199
      {
        j = 0; //reset index into Payload array.
        Frame = 0;
        //Serial.println("Image received");
/*
        for (k = 0; k < 204;k++){
          Serial.print(Payload[k]);

        }
        Serial.println("Buffer contents =   ");
        for (k = 0; k < 204;k++){
         Serial.print(fb->buf[k]);
        }



         // test
         while (Serial.availableForWrite())

         {

           for (k = 0; k < sizeof(Payload); k++)
           {
             //Serial.write(Payload, sizeof(Payload));
             Serial.write(Payload[k]);

           }

         }


           //client.write(buff, size);
           //client.flush();
```

```
     }


    //}

     //TEST CODE ENDS HERE
     */

   //  if (esp_now_send(peer_addr, dataToSend, frame_length1) == ESP_OK){
   //         Serial.println("Send Success");
   //       }
         sendData();

         //delay(25);
         Serial.printf("Frame Number = %d", cnt);
         frameStart = frameEnd;
         frameEnd = frameStart + (frame_length1);
        // if(!ACKED){           // if we got a response from the slave for the first frame (0) continue
        //   Serial.println("1st Frame not ACKED");
        //   break;              // if not the don't send the rest of the photo. Trying to sync the slave with
frame zero.
        // }


    }

    //ACKED = false;
    //dataToSend[5] = {I,m,a,g,e1};
    //sendData;
    esp_camera_fb_return(fb); // release camera buffer.  DO NOT RELEASE IF MULTIPLE VARIATIONS
ARE TO BE SENT FOR COMPARISION
    frameStart = 0;
    frameEnd = 240; //200

    //memset(Payload, 0, sizeof(Payload));  // flush array
    //Serial.flush();
    Serial.println("Complete Frame Sent ----------------------------------------");
     }else{
          Serial.println("NO snap taken ?");
        }
   }


void loop() {
   // In the loop we scan for slave
  ScanForSlave();
  // If Slave is found, it would be populate in `slave` variable
  // We will check if `slave` is defined and then we proceed further
  if (slave.channel == CHANNEL) { // check if slave channel is defined
    // `slave` is defined
    // Add slave as peer if it has not been added already
    bool isPaired = manageSlave();
    if (isPaired) {
      // pair success or already paired
      //fullImage();
      //delay(7000);
      compress();   // take photo & compress & send data.
      //delay(8000);
      ROI();    // Region of Interest - uncompressed & send data

      // wait for 10 seconds to run the logic again
      delay(10000);
      }
       else
       {
      // slave pair failed
      Serial.println("Slave pair failed!");
       }



}
}
```

C++ Code for ESPNOW->Serial (SLAVE):

```
/**
  ESPNOW - Basic communication - Slave
  Date: 26th September 2017
  Author: Arvind Ravulavaru <https://github.com/arvindr21>
  Purpose: ESPNow Communication between a Master ESP32 and a Slave ESP32
  Description: This sketch consists of the code for the Slave module.
  Resources: (A bit outdated)
  a. https://espressif.com/sites/default/files/documentation/esp-now_user_guide_en.pdf
  b. http://www.esploradores.com/practica-6-conexion-esp-now/

  << This Device Slave >>

  Flow: Master
  Step 1 : ESPNow Init on Master and set it in STA mode
  Step 2 : Start scanning for Slave ESP32 (we have added a prefix of `slave` to the SSID of slave for an
easy setup)
  Step 3 : Once found, add Slave as peer
  Step 4 : Register for send callback
  Step 5 : Start Transmitting data from Master to Slave

  Flow: Slave
  Step 1 : ESPNow Init on Slave
  Step 2 : Update the SSID of Slave with a prefix of `slave`
  Step 3 : Set Slave in AP mode
  Step 4 : Register for receive callback and wait for data
  Step 5 : Once data arrives, print it in the serial monitor

  Note: Master and Slave have been defined to easily understand the setup.
        Based on the ESPNOW API, there is no concept of Master and Slave.
        Any devices can act as master or salve.
*/
#include <Arduino.h>
#include <esp_now.h>
#include <WiFi.h>
#include <esp_wifi.h>


//#define BUFFER_SIZE 76800
//uint8_t buff[BUFFER_SIZE];

#define CHANNEL  1



uint32_t i = 0;
uint8_t CurrentID;
uint32_t j = 0;
uint8_t k = 0;
uint8_t l = 0;
uint16_t BytesRev;
uint16_t TotFrameNum;
uint8_t Payload[40000];//40000
uint16_t Frame = 0;
uint8_t data[250];//256
boolean sync = false;     // Raw data indicator
boolean sync1 = false;    // ROI indicator
boolean sync2 = false;    // compressed data indicator
boolean Raw = false;
boolean FullImage = false;
boolean ROI = false;              // used to indicate if received data is Region of Interest.
boolean Comp = false;

RingbufHandle_t buf_handle;
//static char tx_item[] ;



// Init ESP Now with fallback
void InitESPNow() {

  if (esp_now_init() == ESP_OK)
  {
```

```
    Serial.println("ESPNow Init Success");
  }
  else {
   // Serial.println("ESPNow Init Failed");
    // Retry InitESPNow, add a counte and then restart?
    InitESPNow();
    // or Simply Restart
    ESP.restart();
  }
}

// config AP SSID
void configDeviceAP() {
  const char *SSID = "Slave_1";
  bool result = WiFi.softAP(SSID, "Slave_1_Password", CHANNEL, 0);
  if (!result) {
    Serial.println("AP Config failed.");
  } else {
    Serial.println("AP Config Success. Broadcasting with AP: " + String(SSID));
    Serial.printf("with Channel no. = %d",CHANNEL);
    Serial.println(WiFi.macAddress());
  }
}


// callback when data is recv from Master
void OnDataRecv(const uint8_t *mac_addr,const uint8_t *data, int data_len) {

  //-------------------------------------------------------------------;
  //       Data Structure      (Frame)                                  ;
  //-------------------------------------------------------------------;
  //Frame I.D | Frame Number|                                          ;
  // 1 Byte   | 4 Bytes     |         240 Bytes                       ;
  //-------------------------------------------------------------------;
  //Serial.println(data_len);
  //uint8_t data2[200];

  //memcpy(data2,data+3,200);


  //delay(0.5);
  //Serial.println(data_len);
  //uint8_t *pa = Payload;            // pointer to array
  //uint8_t *pb = (uint8_t *)&data[3];  // pointer to received data
  //Frame = (data[1] << 8) + data[2];       //.................................
  //Serial.println(data[0]);
  //Serial.println(Frame);
  //Serial.printf("Current ID = %d", data[0]);

  //uint32_t Loc = Frame * (data_len - 3);

  //if (Frame == 0) // The first Frame of an Image   ..........................
  //{               //.....................................................
  //  CurrentID = (data[0]); //Store ID of first frame ........................
    //Serial.printf("Current ID = %d", CurrentID);
    //Serial.println("");

   // }            //.....................................................

 // if (CurrentID == data[0] && Frame <= 384) {  // Check if it is from the same photo (i.e. same I.D)
........

 // Serial.println(Frame);
    //Send an item to buffer
  UBaseType_t res = xRingbufferSend(buf_handle,data, data_len, pdMS_TO_TICKS(500));// was 100
  //Serial.printf("data_len = %d", data_len);
  if (res != pdTRUE)
  {
    printf("Failed to send item\n");
    }
  //}  //..................................................... poc 15/12/20

/*
    while (i < data_len) // Total data length = 203 ( image data = 200 - I.D & Frame count)
    {
```

```
    //Payload[j] = *pb++;
    Payload[Loc+j] = data[i + 3];          // Put packets in correct location.
     i++;                                  // number of elements to copy
     j++;                                  // increment pointer into Payload array
    }
    i = 0; // reset index

  //Serial.printf("Frame ID = %d", data[0]);

  //Serial.println(Frame);
  //Serial.printf("Data_len = %d", data_len);



  while (Serial2.availableForWrite()){
  //for (k = 3; k < data_len;k++){
  //Serial.write(data[k]);
  Serial2.write(data+3,data_len);
}

//Serial.printf("Frame = %d", Frame);
//Serial.println("");
  //}
  /*
  }
  else
  {
    //Serial.println("Received Packet from different Image.");
    j = 0;    // reset index into Payload.
    i = 0;
    //Frame = 0;
  }

  // for QVGA - 320X240 Pixels, using greyscale B&W = 76800 Bytes.
  // each frame is 200 bytes long, 200 bytes of actual data.
  // Number of frames = 76800/200 = 384 frames. - even boundaries.

}


}
*/
  }



void setup() {

  Serial.begin(115200);
  Serial2.begin(115200,SERIAL_8N1,16,17);               // connection with socket ESP32 - pin 16 & 17

  // Set the device as a Station and Soft Access Point simultaneously
  WiFi.mode(WIFI_AP);

  /* WiFi.begin(ssid, password);

  WiFi.printDiag(Serial); // Uncomment to verify channel number before
  esp_wifi_set_promiscuous(true);
  esp_wifi_set_channel(CHANNEL, WIFI_SECOND_CHAN_NONE);
  esp_wifi_set_promiscuous(false);
  WiFi.printDiag(Serial); // Uncomment to verify channel change after
*/
  //configDeviceSTA();
 /*
  WiFi.begin(ssid, password,CHANNEL);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
    Serial.println("Setting as a Wi-Fi AP..");
  }
  Serial.print("Station IP Address: ");
  Serial.println(WiFi.localIP());
  Serial.print("Wi-Fi Channel: ");
  Serial.println(WiFi.channel());
```

```
*/

  // configure device AP mode
  configDeviceAP();



  //wifiServer.begin();
  // This is the mac address of the Slave in AP Mode
  //Serial.print("AP MAC: "); Serial.println(WiFi.softAPmacAddress());
  // Init ESPNow with a fallback logic
  InitESPNow();

  esp_now_register_recv_cb(OnDataRecv);

     //Create ring buffer
   //RingbufHandle_t buf_handle;

  buf_handle = xRingbufferCreate(70000, RINGBUF_TYPE_BYTEBUF);
  if (buf_handle == NULL)
  {
   //printf("Failed to create ring buffer\n");
    }


}



void loop() {

    //Receive data from byte buffer
    size_t item_size ;
    uint8_t *item = (uint8_t *)xRingbufferReceiveUpTo(buf_handle, &item_size,
pdMS_TO_TICKS(500),1024);//was245
    //Check received data
    if (item != NULL) // was != NULL
    {
   if (!sync)
     {
      for (int i = 0; i < item_size; i++) // was < item_size
      {
        //Serial.print(item[i]);
        //Serial.print(",");
        //Serial2.write(item[i]);
        data[i] = item[i]; // check if first three bytes in correct sequence
        vRingbufferReturnItem(buf_handle, (void *)item);
     }
     i = 0;

     //---------------------------------------------------------------------------
     // Compressed Data conditional
     //---------------------------------------------------------------------------
     if ((data[0]& 0xf0) == 160 && (data[0]& 0x0f) <= 10)  // is first Byte (I.D) < =  10 ? & compressed
identifier.
     {
       // Get next byte
       if (data[1] == 0 && data[2] == 0 && data[3] == 0 && data[4] == 0) // is 1st,2nd,3rd,4th Byte = 0 ? -
- WAS DATA[1] FOR uncompressed data
       {
         //Serial.println("Compressed Sync detected");          // Yes, ID <=10 & FRame Number = 0 ( 1st
Frame)
         sync2 = true;
         sync = true;
         Comp = true;
         for (int i = 0; i < item_size; i++) // Don't know how many Bytes were buffered so get size.
         {
           Payload[j] = data[i];                // Store next Byte in correct location in Array.
           //Serial.print(Payload[j]);
           //Serial.print(",");
           j++;

         }
         i = 0;
```

```
      }
    }
    //------------------------------------------------------------------------
    // Raw Data conditional
    //------------------------------------------------------------------------
     else if ((data[0]& 0xf0) == 144 && (data[0]& 0x0f) <=10) // is first Byte (I.D) < =  10 ? and
RawImage flag set ?
    {
      // Get next byte
      j = 0;
      if (data[1] == 0 && data[2] == 0) // is 1st,2nd Byte = 0 ?
      {
        //Serial.println(" Raw Sync detected");         // Yes, ID <=10 & FRame Number = 0 ( 1st Frame)
        sync = true;
        Raw = true;
        for (int i = 0; i < item_size && j <= 78400; i++) // Don't know how many Bytes were buffered so
get size.
        {
          Payload[j] = data[i];               // Store next Byte in correct location in Array.
          //Serial.print(Payload[j]);
          //Serial.print(",");
          j++;

        }
        i = 0;

      }
    }
    //------------------------------------------------------------------------
    // ROI Data conditional
    //------------------------------------------------------------------------
    else if((data[0]& 0xf0) == 96 && (data[0]& 0x0f) <=10){
      ROI = true;
      sync1 = true;
      sync = true;
      //Serial.println("ROI Frame");
      for (int i = 0; i < item_size && j <= 4800; i++) // Don't know how many Bytes were buffered so get
size.
      {
       Payload[j] = data[i]; // Store next Byte in correct location in Array.
       //Serial.print(Payload[j]);
       //Serial.print(",");
       //Serial.println("ROI Detected");
       j++;

        }
        i = 0;
    }
    }

    if (sync2 && sync && Comp){            // compressed data indicator
      //Serial.println("in Sync2");                              // If we have already detected the 1st Frame
then ..
      for (i = 0; i < item_size  ; i++)
        {
          data[i] = item[i];             // Read in buffered data and store in next location in 'Payload' array
          Payload[j] = data[i];
          //Serial.print(Payload[j]);
          vRingbufferReturnItem(buf_handle, (void *)item);

          j++;
          if (data[0] == 127 ){   // possibility of false trigger ?
          //if (j >= 77600 ){   // possibility of false trigger ?
            FullImage = true;
          }

         }
         //vRingbufferReturnItem(buf_handle, (void *)item);
        i = 0;

     }

    if (sync1 && sync){
      //Serial.println("in Sync1");                              // If we have already detected the 1st Frame
then ..
```

```
       for (i = 0; i < item_size && j <= 4800 ; i++)
          {
           data[i] = item[i];             // Read in buffered data and store in next location in 'Payload' array
           Payload[j] = data[i];
           //Serial.print(Payload[j]);
           vRingbufferReturnItem(buf_handle, (void *)item);

           j++;
           //if (data[0] == 127 ){   // possibility of false trigger ?
           if (j >= 4800 ){   // possibility of false trigger ?
             FullImage = true;
           }

          }
          //vRingbufferReturnItem(buf_handle, (void *)item);
         i = 0;

      }
      if (sync && !sync1 && !sync2){
       //Serial.println("in Sync");                                  // If we have already detected the 1st Frame
then ..
       for (i = 0; i < item_size  ; i++)
          {
           data[i] = item[i];             // Read in buffered data and store in next location in 'Payload' array
           Payload[j] = data[i];
           //Serial.print(Payload[j]);
           vRingbufferReturnItem(buf_handle, (void *)item);

           j++;
           //if (data[0] == 127 ){   // possibility of false trigger ?
           if (j >= 77600 ){   // possibility of false trigger ?
             FullImage = true;
           }

          }
          //vRingbufferReturnItem(buf_handle, (void *)item);
         i = 0;

      }


   } else {
      //Failed to receive item
     //Serial.print("Failed to receive item\n");
   }

   //}
   if (FullImage && Comp){
     //Serial.println("");
    // Serial.println("Full Image ");
    // Serial.printf("j  = %d", j);
    // Serial.println("");
     esp_now_unregister_recv_cb();
     while(Serial.availableForWrite()){        // write Payload size to Serial2, Serial -> TCP or directly to
Serial for testing
       Serial.print("Image Size ");
       Serial.write(j & 0xff);
       Serial.write((j >> 8)  & 0xff);
       Serial.write((j >> 16) & 0xff);
       Serial.write((j >> 24) & 0xff);
       for (i = 0; i < 230;i++){            // need to send a total packet of 245 bytes to trigger a serialEvent in
processing.
        Serial.write(0);
       }
        //Serial2.println("");

        for (i = 0; i < j; i++)
        {
         Serial.write(Payload[i]);
        }

        //Serial.print(Payload[i]);
        //Serial.print(",");
       }
     FullImage = false;
```

```
      j = 0;
      sync2 = false;
      sync = false;
      Comp = false;
     }
      else if(FullImage && ROI){
       // while(Serial.availableForWrite()){
        ROI = false;
        //Serial.printf("j = %d", j);
        //Serial.println("");
        //for (i = 0; i < j; i++)
        //{
         Serial.write(Payload,j);  // was Serial2 //Payload[i]
        //}
       // }
        //Serial.flush();
        FullImage = false;
        j = 0;
        sync1 = false;
        sync = false;
      }
      else if(FullImage && Raw){
       // while(Serial.availableForWrite()){
        Raw = false;
        //Serial.printf("j = %d", j);
        //Serial.println("");
        //for (i = 0; i < j; i++)
        //{
         Serial.write(Payload,j);  // was Serial2 //Payload,j
        //}
       // }
        //Serial.flush();
        FullImage = false;
        j = 0;
        sync = false;

      }
      //Serial2.flush();
      esp_now_register_recv_cb(OnDataRecv);
      //FullImage = false;
      //j = 0;
      //sync = false;
      i = 0;
     }
     //Serial.println("");
     //esp_now_register_recv_cb(OnDataRecv);
     /*
      if (Frame == 383)               // number of frames for complete image (76800 / 200)
      {
       esp_now_unregister_recv_cb();
       j = 0; //reset index into Payload array.
       //Serial.println("Image received");
       Frame = 0;
/*
        while (Serial.availableForWrite()){
        for (k = 0; k < sizeof(Payload);k++){
        Serial.write(Payload[k]);
        }
          }

        esp_now_register_recv_cb(OnDataRecv);
       //Serial2.flush();
*/
     /*
       // read data from Payload and send to Serial2
      while ( Serial2.availableForWrite()) {
          //size = (size >= BUFFER_SIZE ? BUFFER_SIZE : size);
          //Serial.readBytes(buff, size);
          for (k = 0; k < sizeof Payload;k++){
            //Serial2.write(Payload, sizeof Payload);
            Serial2.write(Payload[k]);
          }
          //client.write(buff, size);
          //client.flush();
          //Serial2.flush();
```

```
     }
*/
    //}

     //Serial2.flush();
     // esp_now_register_recv_cb(OnDataRecv);
     //j = 0;        //reset index into Payload array.
     //for (k = 0; k<=sizeof(Payload); k++){
     //  Serial.print(Payload[k]);

     //}
     //Frame = 0;
     //Serial.println("");
     //Serial.println("Image received");
     //}
     /*
     //Serial.println("Image received");
     WiFiClient client = wifiServer.available(); // if client connected.
     if (client)
     {
       //if (client.available())
       //{
       Serial.print("Client connected with IP:");
       Serial.println(client.remoteIP());
       //for (k = 0; k<=sizeof(Payload); k++){
       // Serial.print(Payload[k]);
       client.write(Payload, sizeof(Payload));
       //client.stop();
       //}
       //delay(2);
       //}
     }
     //client.stop();
     */
    //}
//}
```

C++ Code for Serial->TCP(Slave):

```
#include "Arduino.h"
#include <WiFi.h>
// debug log, set to 1 to enable
#define ENABLE_DEBUG_LOG 1

// wifi config
const char* ssid     = "";
const char* password = "";

// ethernet config
const IPAddress local_IP(192, 168, 0, 20);
const IPAddress gateway(192, 168, 0, 1);
const IPAddress subnet(255, 255, 255, 0);
const IPAddress primaryDNS(8, 8, 8, 8);
const IPAddress secondaryDNS(8, 8, 4, 4);

// rs-server config
const int serverPort = 3000;

const int baudrate = 115200;
const int rs_config = SERIAL_8N1;
uint32_t k = 0;
uint32_t i = 0;
uint32_t j = 0;
uint16_t capacity = 0;
uint16_t Frame = 0;
uint8_t data[256];   //256                    // reading buffer config
//uint8_t Payload[77952];                     // use to actually hold complete frame structure + payload
```

```
boolean sync = false;
boolean FullImage = false;
IPAddress ipServer(192, 168, 0, 199); // Declaration of default IP for Client

hw_timer_t *timer = NULL;
volatile SemaphoreHandle_t timerSemaphore;
portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;

// global objects
WiFiServer server(serverPort);
WiFiClient RemoteClient;                        // Instantiate WiFiClient as REmoteClient


void debug_log(char* str) {
#if ENABLE_DEBUG_LOG == 1
  Serial.println(str);
#endif
}

void setup() {
  Serial.begin(baudrate, rs_config);            // Terminal on Serial1
  Serial2.begin(115200,rs_config,16,17);        // Readin on Serial2 out on Socket TCP


  // init wifi connection
  if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS))
  {
    debug_log("Failed to configure network settings");
}
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
     debug_log("connecting to WiFi network");
     delay(500);
}

  #if ENABLE_DEBUG_LOG == 1
    Serial.println("connected to WiFi");
    Serial.println("IP adddr: ");
    Serial.println(WiFi.localIP());
  #endif
  delay(1000);

//start server
  server = WiFiServer(serverPort);
  server.begin();
  delay(1000);
  debug_log("server started");

}

// On data received on Serial2 do something. Called in Main loop.
// Detect start of Image by checking if the ID is < =10, and the nect two bytes are the first frame (0,0).
// If start detected then set 'sync' and store the received data in the 'Payload' array.
// next data received should be from the same ID , so save in 'Payload'.
/*
void serialEvent2() {
while ((capacity = Serial2.available())) {
  //Serial.printf("Size = %d", size);
if(capacity >0){
    for (i = 0; i < capacity; i++)
    {
      data[i] = Serial2.read(); //
      RemoteClient.write(data[i]);
      Serial.print(data[i]);
    }
    //Serial.printf("load size = %d", size);
    //Serial.println("");
    //for (i = 0; i < capacity;i++){
        //RemoteClient.write(data[i]);
    //      Serial.print(data[i]);
     //}
     Serial.println("");
     //j = 0;                                 // Reset Index into Payload, ready for next Image.
     //i = 0;
     //Frame = false;
```

```
      i = 0;
      Frame = true;
      /*
  if (!sync)
  {
    for (i = 0; i < 245; i++)
    {
      data[i] = Serial2.read(); // check if first three bytes in correct sequence
    }
    i = 0;
    if (data[0] <= 10) // is first Byte (I.D) < =  10 ?
    {
      // Get next byte
      if (data[1] == 0 && data[2] == 0 && data[3] == 0 && data[4] == 0) // is 1st,2nd,3rd,4th Byte = 0 ? -
- WAS DATA[1] FOR uncompressed data
      {
        Serial.println("Sync detected");         // Yes, ID <=10 & FRame Number = 0 ( 1st Frame)
        sync = true;
        //Serial.printf("size = %d",size);
        for (int i = 0; i < 245; i++) // Don't know how many Bytes were buffered so get size.
        {
          Payload[j] = data[i];               // Store next Byte in correct location in Array.
          j++;
          Serial.print(data[i]);
        }
        i = 0;
      //}
    }
  }
   i = 0;
  }
 else if (sync){                              // If we have already detected the 1st Frame then ..
      for (i = 0; i < size; i++)
        {
         data[i] = Serial2.read();             // Read in buffered data and store in next location in 'Payload'
array
          Payload[j] = data[i];
          j++;
          if (data[0] == 127){
            FullImage = true;
          }
         }
        //Serial.printf("size in sync = %d",size);
        i = 0;
    }

  }
 }
}
*/
// Used to check if Client is or is trying to connect.

void CheckForConnections()
{
  if (server.hasClient())
  {
  if ( RemoteClient.connected())
  {
    Serial.println("Connection Rejected");          // If already connected then reject.
    server.available().stop();
  }
  else
  {
    Serial.println("Connected accepted");
    RemoteClient = server.available();
  }
  }

}



void loop() {
  CheckForConnections();
  if (RemoteClient)
```

```
  {
  //serialEvent2();
  while ((capacity = Serial2.available())) {
  //Serial.printf("Capacity = %d", capacity);
  //Serial.println("");
  if (capacity > 0)
  { // >0
    for (i = 0; i < capacity; i++)
    {
      data[i] = Serial2.read(); //
      //RemoteClient.write(data[i]);
      //Serial.print(data[i]);
      //Serial.print(",");
    }
    //Serial.printf("load size = %d", size);
    //Serial.println("");
    //for (i = 0; i < capacity;i++){
     RemoteClient.write(data,capacity);
        //Serial.write(data[i]);
     //}
     //Serial.println("");
     //j = 0;                                // Reset Index into Payload, ready for next Image.
     //i = 0;
     //Frame = false;
     i = 0;
     Frame = true;
  }
  }
  }
  //if (FullImage)       //j >= 77952                          // If the 'Payload' counter indicates we have
received a full image.
  //{
  //   Serial.println("Should have a full payload now");
  /*
  if (Frame){
    if (RemoteClient.connected()) {
      Serial.print("Client CAMERA connected with IP:");
      Serial.println(RemoteClient.remoteIP());
      Serial.printf("Payload size = %d", capacity);
      //RemoteClient.write(data, size); // Send Complete Image + overhead to remote Client via TCP

      for (i = 0; i < capacity;i++){
        //RemoteClient.write(data[i]);
        Serial.print(data[i]);
      }

      //j = 0;                                 // Reset Index into Payload, ready for next Image.
      i = 0;
      Frame = false;
      //FullImage = false;
      //sync = 0; // need to detect 1st Frame so reset sync.
      //Serial.println("Client Disconnected");
      }
  }
      //if (!sync){

      if (RemoteClient)
      {
        Serial.print("Client connected with IP:");
        Serial.println(RemoteClient.remoteIP());

      }
  */


}  // end loop
```

Java Image decompress Code:

```
import processing.serial.*;
Serial myPort;
Table datatable;
short portIndex = 32;      // 33 select the com port, 0 is the first port



//if (myPort.available() >0) {     //if data is available
//  val = myPort.read();          // read it and srore it in val

//int camPixels[];
int[] camPixels = new int[77800];

static int camWidth = 320;//320
static int camHeight = 240;//240

int Frame;
int i = 0;
int j = 0;
int CurrentID;
int[] data = new int[245];//203
byte [] data1 ;
int v;  // Input string from serial port
boolean Raw = false;
boolean Compressed = false;
boolean First = true;
int  x = 0;
int  y = 0;


int[] dataIn = new int[245];
int[] temp = new int[4900];
int decompress[];
//int camPixels[];
int num;
int p = 0;
;
int val = 0;
int size = 4800;
int h = 0;
int frame0;
int frame1;
int frame2;
int frame3;
boolean image = false;
boolean set = false;
boolean done = false;
int frameStart;
int frameEnd;
int ID;
int ID1;
int loc = 0;
boolean Qmed = false;
boolean Qhigh = true;
boolean info = false;
boolean Comp = false;

PrintWriter output;


void setup()
{
  size(320, 240);

  String portName = Serial.list()[portIndex];
  println(Serial.list());
  println("  Connecting to -> " + Serial.list()[portIndex]);
  myPort = new Serial(this, portName, 115200);
  myPort.buffer(245); //203
  temp = new int[4900];
  decompress = new int[80000];
  datatable = new Table();
  datatable.addColumn("Value");
```

```
  //output = createWriter("ROI_1.csv");
  //output = createWriter("Image.csv");
}

void draw()
{
  //noStroke();
  int p = 0;

  if (Frame == 315 && Raw ) {
    j = 0; //reset index into Payload array.
    println("Received an Image !");
    Raw = false;
    Frame = 0;
    First = true;
    noStroke();
    for (int y = 0; y<camHeight; y++)
      for (int x=0; x<camWidth; x++)
      {
        int v = camPixels[p++];
        //TableRow newRow = datatable.addRow();
        //newRow.setInt("Value",datatable.getRowCount() -1);
        //newRow.setInt("Value",v);
        fill(color(v, v, v));
        rect(x*1, y*1, 1, 1);
      }
    saveFrame("Raw-###1.png");
    //saveTable(datatable,"Raw-###1.csv");
  } else if (set && !Raw && !Comp ) {
    j = 0;

    noStroke();
    for (int y = 0; y<60; y++) {
      for (int x=0; x<80; x++)
      {
        int v = temp[j++];
        fill(color(v, v, v));
        rect(x*1, y*1, 1, 1);
      }
    }
    saveFrame("ROI-##1.png");
    j = 0;
    p=0;
    dataIn[0] = 0;
    set = false;
    image = false;
    First = true;
  } else if (set && Comp ) {
    i=0;
    j = 0;
    h = 0;                                  // index into decompressed file.
    while ( frameEnd <= size ) {            // is this the first frame ? - no, then....reuse j as index into
decompress - upto 76800
      for ( i=frameStart; i<frameEnd; i++) {
        if (Qhigh) {
          int col = camPixels[i]& 0xf8 ;
          int freq = camPixels[i]& 0x07 ;
          while (freq >0 ) {  // was freq >0 - poc 12/7/21
            decompress[loc+h] = col;    //if frameStart = 0 -> 0+1,0+2 etcc. until frameEnd: initially 0 ->240
            h++;
            freq--;
          }
        } else if (Qmed) {
          int col = camPixels[i]& 0xf0 ;
          int freq = camPixels[i]& 0x0f ;
          //println("freq = "+freq);
          while (freq > 0) {
            decompress[loc+h] = col;    //if frameStart = 0 -> 0+1,0+2 etcc. until frameEnd: initially 0 ->240
            h++;
            freq--;
          }
        }
      }
```

```
      h = 0;
      //print("frame Start ="+frameStart);
      //print("frame end = "+frameEnd);
      //println("");
      frameStart = frameEnd+5;
      frameEnd = frameStart + 240;//240
      int frame0 = (camPixels[frameStart -4]) <<24;
      int frame1 = (camPixels[frameStart -3]) <<16;
      int frame2 = (camPixels[frameStart -2]) <<8;
      int frame3 = (camPixels[frameStart -1]) & 0XFF;
      loc = frame0 | frame1 |frame2 | frame3;

      print("loc = "+loc);
      println("");
      print("I.D = "+ camPixels[frameStart-5]);
    }

    j = 0;
    set = false;
    image = false;
    println("set = false");
  //}

  int q = 0;
  noStroke();
  if ( frameEnd >=size) {
    //noStroke();
    for (int y = 0; y<camHeight; y++) {
      for (int x=0; x<camWidth; x++)
      {
        int v = decompress[q++];
        TableRow newRow = datatable.addRow();
        newRow.setInt("Value",datatable.getRowCount() -1);
        newRow.setInt("Value",v);
        //v&=0x000000ff;
        fill(color(v, v, v));
        rect(x*1, y*1, 1, 1);
      }


    }
  }
     saveTable(datatable,"Comp-##1.csv");
     saveFrame("Comp-##1.png");
     stop();

}

}




void serialEvent(Serial myPort) {
  if (First  ) {
    j = 0;
    for (i=0; i<245; i++) {
      dataIn[i] = myPort.read();  // check if first three bytes in correct sequence
    }
    //println(data[0]);
    if ((dataIn[0] & 0xf0) == 144 && (data[0] & 0x0f) <=10  )                    //------------ Is this a Raw
Image ?
    {
      // Get next byte
      //data[1] = myPort.read();  // check if first three bytes in correct sequence
      if (dataIn[1]  == 0)
      {
        //data[2] = myPort.read();  // check if first three bytes in correct sequence
        if (dataIn[2]  == 0)
        {
          // in sync so get last 240 bytes
          // for (int i=3; i < 245; i++) //203
          // {
```

```
        //   data[i] = myPort.read();
        // }

        while (i < 240) {  //200
          camPixels[j] = dataIn[i+5];            // disregard the first 3 bytes ( ID & Frame number).
          j ++;                              // reset index otherwise we first frame twice !
          i++;
        }
       }

       i = 0;
       Frame = (dataIn[1] << 8) + dataIn[2];  // get Frame number
       CurrentID = dataIn[0]& 0x0f;
       println("Frame1 = ", Frame, "ID =  ", CurrentID);
       Raw = true;      // we have found start of Image sequence.
       First = false;      // set to stop falling through to next if on first frame detection.
       j = 0;
      }
     }

     if ((dataIn[0] & 0xf0) == 96 && (dataIn[0] & 0x0f) <=10  ) {        //-------------------------------- Is this
a ROI Image ?
      //dataIn[0] = data[0];
      //for (int i=1; i < 245; i++)//203
      //{
      //  dataIn[i] = myPort.read();
      //}
      if ((dataIn[0] & 0xf0) == 96) {
        int frame0 = (dataIn[1]) <<24;
        println("frame0 = "+ frame0);
        int frame1 = (dataIn[2]) <<16;
        println("frame1 = "+ frame1);
        int frame2 = (dataIn[3]) <<8;
        println("frame2 = "+ frame2);
        int frame3 = (dataIn[4]) & 0XFF;
        println("frame3 = "+ frame3);
        loc = frame0 | frame1 |frame2 | frame3;
        println("dataIn[0] = ", dataIn[0]&0xf0);
        print("loc = "+loc);
        println("");
        println("I.D = "+ (dataIn[0]&0x0f));
        image = true;                  // indicate first frame.
        First = false;
        //set = true;
        i = 0;
        int start = 5;
        //int newline = 80;    // width of ROI
        // while (i<= val) {
        for ( i=start; i<245; i++) {
          temp[p] = dataIn[i];
          //print(temp[p]);
          //print(",");
          p++;
        }
       }
     }

     /*else if ((data[0] & 0xf0) == 160 && (data[0] & 0x0f) <=10  ) {       // Is this a ROI Image ?
      dataIn[0] = data[0];
      for (int i=1; i < 245; i++)//203
      {
      dataIn[i] = myPort.read();
      }
      */
     if ((dataIn[0] == 73) && (dataIn[1] == 109) && (dataIn[2] == 97) && (dataIn[3] == 103) && (dataIn[4]
== 101) && (dataIn[5] == 32) && (dataIn[6] == 83) && (dataIn[7] == 105) && (dataIn[8] == 122) &&
(dataIn[9] == 101) && (dataIn[10] == 32)) {
       int size0 = (dataIn[14])<<24;
       int size1 = (dataIn[13]) <<16;
       int size2 = (dataIn[12]) <<8;
       int size3 = (dataIn[11]) & 0XFF;
       size = size0 | size1 |size2 | size3;    // size of file to receive.
       println("");
       print("Image size = ");
       print(size);
```

```
      println("");
      ID = dataIn[15];
      print("ID = "+ ID);
      println("");
      print("Start = "+ dataIn[16] + dataIn[17] + dataIn[18] + dataIn[19]);
      println("");
      for ( i=15; i<245; i++) {
        camPixels[j] = dataIn[i];
        //print(camPixels[j]);
        // print(",");
        j++;
      }
      j = 0;
      Comp = true;                  // indicate first frame.
      First = false;
      //set = true;
      i = 0;
    }
  } else if (!First && image) {           // if we have detected the first Frame of the image already get the
rest. ROI
    for (i=0; i<245; i++) {
      dataIn[i] = myPort.read();
    }
    println("Image");
    println("p = "+p);
    //println("val = "+val);
    if (p< size) {
      for ( i=5; i<245; i++) {
        temp[p] = dataIn[i];
        p++;
      }
    } else if (p >= size) {
      dataIn[0] = 0;
      set = true;
      println("Set");
      println("Size reached");
    }
  } else if (!First && Raw) {           // if we have detected the first Frame of the image already get the rest.
    for (int i=0; i < 245; i++)//203
    {
      data[i] = myPort.read();
    }
    Frame = (data[1] << 8) + data[2];  // get Frame number
    int ID = data[0]& 0x0f;
    i=0;
    println("Frame = ", Frame, "ID =  ", ID);
    if ((data[0]&0x0f) == CurrentID  && Frame < 317) {  //384 if the frame is within the image size and it's
from the same image.(data[0]&0x0f) == CurrentID &&
      while (i < 240) {  //200
        camPixels[j] = data[i+5];             // disregard the first 3 bytes ( ID & Frame number).
        j ++;
        i++;
      };
      i = 0;
    } else {
      println("Different Image Data");
      i=0;
      j = 0;
      First = true;
      Frame = 0;
      //dataIn[0] = 0;
    }
  } else if (!First && Comp) {           // if we have detected the first Frame of the image already get the
rest.
    if (j < size)//203
    {
      for (i=0; i<245; i++) {
        camPixels[j] = myPort.read();;
        j++;
      }
    } else if (j >= size) {
      set = true;
      //camera.stop();
      //image = false;
    }
```

```
 }
 frameStart = 5;                // because data is only saved from [15] onwards.
 frameEnd = frameStart+240;//240
 loc = 0;
}




/*
 //if (camera.available()>=4800) {
 if (mySerial.available()>0) {
 //info = true;
 //initial=camera.readBytes();
 initial=mySerial.readBytes();
 int val = (initial.length);
 print("val1 = "+val);
 dataIn = new int[val];  // java converts bytes to +127 -> -128 so convert back
 for (i=0; i<val; i++) {
 dataIn[i] = int(initial[i]);
 //print(dataIn[i]);print(",");
 }
 */
//}

//println("p = "+p);
//start = newline;
//newline = start+80;
//loc = loc + 320;       //  next location of ROI i.e next line down location start
// }
//for ( i=0; i<val; i++) {
//    print(dataIn[i]);
//    print(",");
//  }
//}
//info = false;


/*
 void serialEvent(Serial mySerial) {
 if (!image) {
 for (i=0; i<245; i++) {
 dataIn[i] = mySerial.read();
 }
 //println("i = "+i);
 if ((dataIn[0] & 0xf0) == 96) {
 int frame0 = (dataIn[1]) <<24;
 println("frame0 = "+ frame0);
 int frame1 = (dataIn[2]) <<16;
 println("frame1 = "+ frame1);
 int frame2 = (dataIn[3]) <<8;
 println("frame2 = "+ frame2);
 int frame3 = (dataIn[4]) & 0XFF;
 println("frame3 = "+ frame3);
 loc = frame0 | frame1 |frame2 | frame3;
 println("dataIn[0] = ", dataIn[0]&0xf0);
 print("loc = "+loc);
 println("");
 println("I.D = "+ (dataIn[0]&0x0f));
 image = true;                  // indicate first frame.
 //set = true;
 i = 0;
 int start = 5;
 //int newline = 80;    // width of ROI
 // while (i<= val) {
 for ( i=start; i<245; i++) {
 temp[p] = dataIn[i];
 //print(temp[p]);
 //print(",");
 p++;
 }
 }
 //p=p+5;
 }
```

```
//image = true;                 // indicate first frame.
else if (image) {
for (i=0; i<245; i++) {
dataIn[i] = mySerial.read();
}
println("Image");
println("p = "+p);
//println("val = "+val);
if (p< size) {
for ( i=5; i<245; i++) {
temp[p] = dataIn[i];
p++;
}
} else if (p >= size) {
set = true;
println("Set");
println("Size reached");
//camera.stop();

//image = false;
}
//info = false;
}

//  }
//}

//frameStart = 5;                 // because data is only saved from [15] onwards.
//frameEnd = frameStart+320;//240
//loc = 0;
//}
//if (set) {
//  for (i = 0; i< size; i++) {
//    print(temp[i]);
//    print(",");
//  }
}
//image = false;
//print(i);print(",");
//}
//print(i);
//p = 0;


//p=0;
//j=0;
//}
//}

/*
if (set) {

i=0;
j = 0;
h = 0;                          // index into decompressed file.
while ( frameEnd <= size ) {            // is this the first frame ? - no, then....reuse j as index into
decompress - upto 76800
for ( i=frameStart; i<frameEnd; i++) {
if (Qhigh) {
int col = camPixels[i]& 0xf8 ;
int freq = camPixels[i]& 0x07 ;
while (freq > 0) {
decompress[loc+h] = col;    //if frameStart = 0 -> 0+1,0+2 etcc. until frameEnd: initially 0 ->240
h++;
freq--;
}
} else if (Qmed) {
int col = camPixels[i]& 0xf0 ;
int freq = camPixels[i]& 0x0f ;
//println("freq = "+freq);
while (freq > 0) {
decompress[loc+h] = col;    //if frameStart = 0 -> 0+1,0+2 etcc. until frameEnd: initially 0 ->240
h++;
freq--;
```

```
}
}
}


h = 0;
//print("frame Start ="+frameStart);
//print("frame end = "+frameEnd);
//println("");
frameStart = frameEnd+5;
frameEnd = frameStart + 240;//240
int frame0 = (camPixels[frameStart -4]) <<24;
int frame1 = (camPixels[frameStart -3]) <<16;
int frame2 = (camPixels[frameStart -2]) <<8;
int frame3 = (camPixels[frameStart -1]) & 0XFF;
loc = frame0 | frame1 |frame2 | frame3;

print("loc = "+loc);
println("");
print("I.D = "+ camPixels[frameStart-5]);
}

j = 0;
set = false;
image = false;
println("set = false");
}

int p = 0;
//noStroke();
if ( frameEnd >=size) {
noStroke();
for (int y = 0; y<camHeight; y++) {
for (int x=0; x<camWidth; x++)
{
int v = decompress[p++];
//v&=0x000000ff;
fill(color(v, v, v));
rect(x*1, y*1, 1, 1);
}
}
}
*/
```