# IoT: rede LoRa para envio de imagens

# *IoT: LoRa network for sending images*

Victor E. Almeida Marco A. Guerra

28 de julho de 2022

## Resumo

Redes utilizando comunicação via LoRa possibilitaram que dispositivos troquem mensagens a longas distâncias e gastando pouca energia. Tendo em vista as limitações de transferência de dados dessas redes este trabalho implementa uma rede para envio de imagens utilizando algoritmos e técnicas já consolidadas de redes de computadores para enviar as imagens em diversas partes e garantir sua integridade. Para implementação dessa rede foram utilizados dispositivos como esp32 e esp32-cam que executam o algoritmo de stop and wait. Uma vez implementada a rede pode-se observar que o tempo médio de envio de uma mensagem LoRa é de 2 segundos, sendo assim podendo enviar imagens de 6810 bytes em 1 minuto.

**Palavras-chave**: LoRa, Rede LPWAN, envio de imagens.

## Abstract

Networks using communication via LoRa made it possible for devices to exchange messages over long distances and using little energy. Observing the data transfer limitations of these networks, this work implements a network for sending images using algorithms and techniques already consolidated in computer networks to send images in several parts and guarantee their integrity. Devices such as esp32 and esp32-cam were used to implement this network, which execute the stop and wait algorithm. Once the network is implemented, it can be observed that the average time for sending a LoRa message is 2 seconds, thus being able to send images of 6810 bytes in 1 minute.

**Keywords**: LoRa, LPWAN Network, sending images.

# 1 Introdução

Redes LoRa e LoRaWAN possibilitaram a comunicação de dispositivos com baixo consumo energético a longas distâncias[1], porém a largura de banda utilizada e a taxa de transmissão ainda são limitadas, sendo assim o presente trabalho propõe a implementação de uma rede LoRa para envio de imagens.

Para atender esse objetivo, primeiramente na seção 2 buscou compreender conceitos tais como o que são Redes LPWAN seção 2.1, o que é LoRa seção 2.2, e por fim o que é IoT seção 2.3.

Uma vez visto os conceitos, na seção 3 expõe-se os Dispositivos utilizados seção 3.1, e os principais algoritmos e protocolos na seção 3.2. Em seguida apresenta-se uma Proposta de Arquitetura, e em seguida sua implementação na seção 5. Por fim é apresentado os resultados obtidos e conclusões nas seções 6 e 7 respectivamente.

# 2 Definições

## 2.1 Redes LPWAN

LPWAN, *Low Power Wide Area Network*, são redes que alcançam longas distâncias gastando pouca energia, normalmente utilizadas para enviar poucos dados[2]. Dentre as tecnologias mais utilizadas estão SigFox e LoRa.

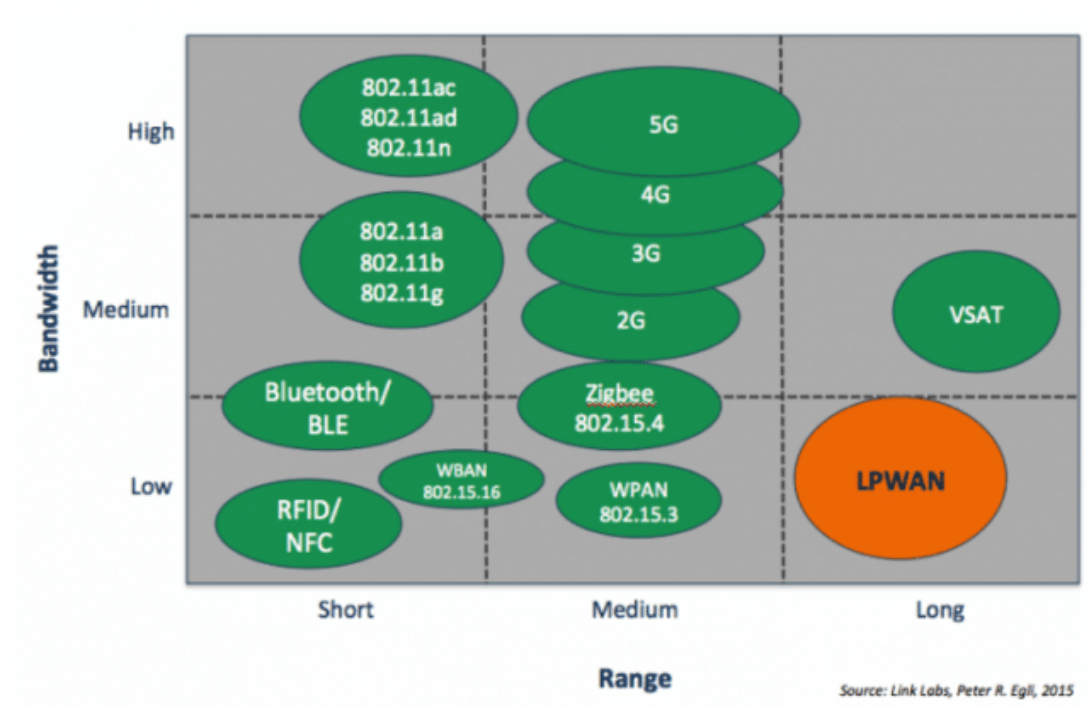


Figura 1 – Comparando Largura de Banda e distância das redes[2]

## 2.2 LoRa

LoRa, *Long Range*, é uma tecnologia que atua na camada física do modelo OSI para o envio e recebimento de dados. Criado e mantido de forma proprietária pela empresa Semtech[1] o mesmo utiliza comunicação através de ondas na frequência de radio, para codificar o envio de dados focando em abarcar longas distância a um baixo custo energético. Nesse sentido, uma rede LoRa é formada por diversas antenas[3], que se comunicam utilizando a tecnologia de *Chirp Spread Spectrum*[4] uma tecnologia de comunicação militar adaptada para uso comercial de baixo custo[3] tais antenas possuem alcance de até 15 km em áreas rurais[5], bem como uma taxa média de duração de bateria de 10 anos[5].

## 2.3 IoT

O termo *Internet of Things (IOT)*, em português internet das coisas foi elaborado pelo britânico, Kevin Ashton, em 1999[6] e se refere de forma geral a uma rede que conecta diversas "coisas" a internet, através de software, com o objetivo de trocar informações[7], tais "coisas" podem ser sensores, microcontroladores ou até mesmo objetos que nunca imaginamos tais como geladeiras, televisores, entre outros.

A estrutura do *internet of things* é baseada em três camadas principais[6]:

- **Camada de percepção**: É uma camada que envolve sensores, hardware e obtém-se dados relevantes a respeito dos fenômenos meteorológicos, biológicos ou físicos tais como temperatura e umidade do solo[8], do ar[9], índice de área foliar[10], quantidade de gás $SO_2$[11], PH do solo[11] entre muitos outros.
- **Camada de comunicação**: Responsável por enviar os dados coletados pela camada de percepção supracitada para outras camadas, quer sejam aplicações que vão analisar tais dados ou para grandes bancos de dados ou até mesmo para serviços na nuvem.
- **Camada de aplicação**: camada a qual trás sentido aos dados coletados pelos sensores, pois é nesse momento que ocorre o processamento dos dados e a apresentação dos mesmos. Nos trabalhos lidos durante a produção desta revisão bibliográfica, essa camada será responsável principalmente por mostrar ao agricultor informações relevantes de forma simples e compreensível, bem como informá-lo qual o melhor momento para plantar[12], ou em quais lugares da plantação tem doenças[13].

# 3 Materiais e métodos

## 3.1 Dispositivos


(a) ESP32 (b) LoRaMESH EndDivice (c) ESP32-CAM

Figura 2 – Dispositivos utilizados na aplicação

### 3.1.1 LoRaMESH EndDivice

Para acessar a tecnologia de camada física LoRa utilizou o Módulo LoRaMESH homologado pela ANATEL[14], o qual possui diversos parâmetros variáveis tais como: *Spreading Factor*, *Coding rate*, Largura de banda de até 500 kHz.

A configuração e utilização do dispositivo é feita através de duas interfaces de comunicação serial (UART), sendo a primeira utilizada para enviar e receber comandos de configuração e a segunda, chamada de interface transparente, envia e recebe diretamente da antena.

### 3.1.2 ESP32

Para criação do protótipo do dispositivo que recebe os dados foi utilizado um doit esp32 devkit v1[1] porém o código e a implementação devem funcionar de maneira idêntica em qualquer ESP32.

### 3.1.3 ESP32-CAM

De maneira similar para o dispositivo que envia dados foi utilizado um esp32-cam Ai-Thinker[2] porém no caso do esp32-cam para utilizar a câmera deve-se especificar quais portas estão conectadas, sendo assim para utilizar outro modelo deve definir os pinos no arquivo hardware.h para utilização do código implementado.

[1] Documentação da placa: <https://olddocs.zerynth.com/latest/official/board.zerynth.doit_esp32/docs/index.html>

[2] Documentação da placa: <https://docs.ai-thinker.com/en/esp32-cam>

## 3.2 Protocolos

Dados os requisitos do projeto a escolha dos algoritmos próprios para a tarefa a ser desempenhada se torna de extrema relevância. Nesse sentido, são abordados os diferentes algoritmos utilizados na implementação do sistema, dando especial atenção para as caraterísticas que influenciaram na sua escolha.

### 3.2.1 Detecção de erros: CRC

Visando a detecção de erros um dos algoritmos mais aconselháveis no caso de comunicação constante é o CRC devido a sua fácil implementação em hardware e baixo custo computacional para mensagens pequenas.

O Algoritmo CRC pode variar em relação: como iniciar seu valor, qual o polinômio gerador e quantos bytes gera na saída.

No caso dessa implementação em C++ o CRC inicializa com $C181_{16}$, possui 16 bits (2 bytes) e o polinômio gerador $G(X) = A001_{16}$, essas são as configurações utilizadas na fase de implementação e também são recomendadas pelos criadores do chip[14]

Algoritmo CRC 16 bits

```
static const uint16_t POLY = 0xA001;
static const uint16_t INIT = 0xC181;
uint16_t computeCRC(uint8_t* data_in, uint16_t length) {
    uint16_t i;
    uint8_t bitbang, j;
    uint16_t crc_calc = INIT;
    for(i = 0; i < length; i++) {
        crc_calc ^= (((uint16_t)data_in[i]) & 0x00FF);
        for(j = 0; j < 8; j++) {
            bitbang = crc_calc;
            crc_calc >>= 1;
            if(bitbang & 1) {
                crc_calc ^= POLY;
            }
        }
    }
    return (crc_calc & 0xFFFF);
}
```

### 3.2.2 Controle de fluxo: Stop and Wait

Pelo fato dos dispositivos LoRa possuírem canais *half duplex* (permitem envio nos 2 sentidos porém apenas um sentido por vez) inviabiliza algoritmos de janela deslizante.

Sendo assim do ponto de vista da camada de enlace de dados o algoritmo de controle de fluxo utilizado foi o **Stop and Wait**, o qual se carateriza pela sua simplicidade tanto durante a sua implementação como no seu funcionamento.

Durante a execução do protocolo o transmissor envia seu primeiro quadro e espera

que o receptor envie a resposta “ACK” do mesmo. Caso o receptor não envie sua resposta em um tempo predeterminado o transmissor reenvia seu quadro. Uma vez recebido o “ACK” o transmissor pode deletar o quadro antigo e enviar o próximo, seguindo assim o seguinte fluxo:

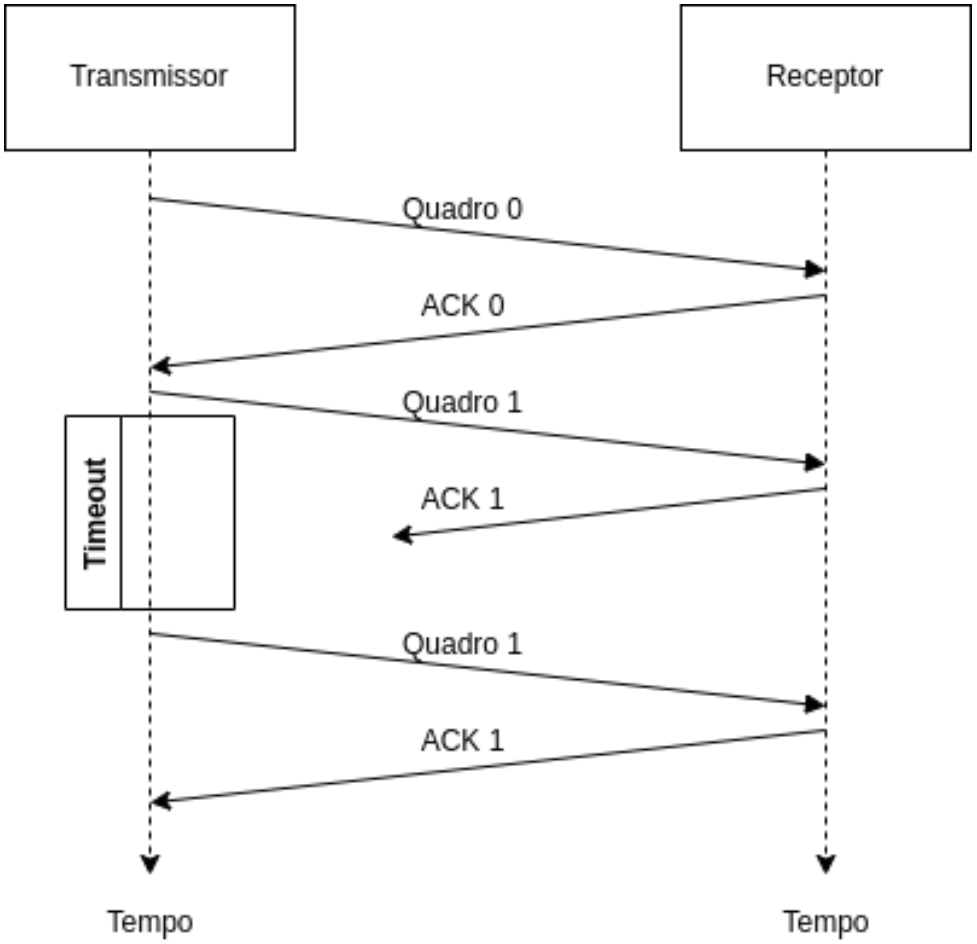


Figura 3 – Fluxo de execução no tempo do algoritmo stop and wait

# 4 Proposta de Arquitetura

Após decidir os algoritmos e dispositivos mais adequados para implementação foi idealizada a arquitetura do projeto.

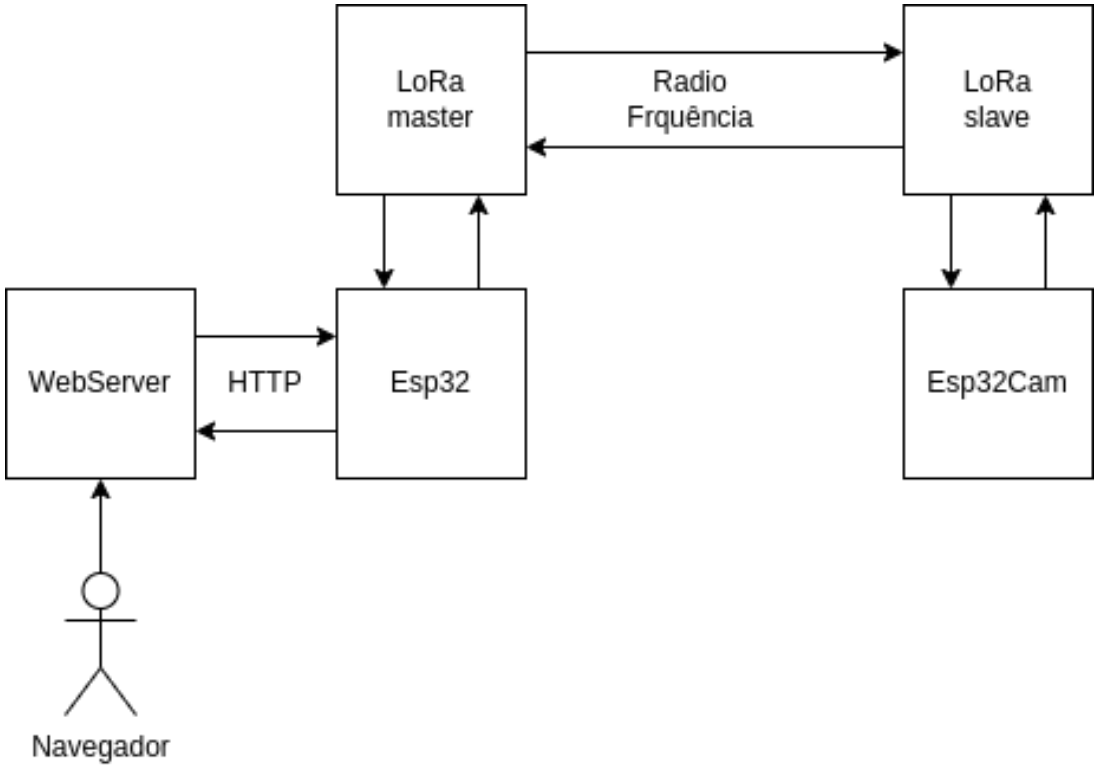


Figura 4 – Proposta de arquitetura para implementação

O foco principal da arquitetura é implementar uma rede LoRa estrela a qual possui um "mestre", chip com id 0, e diversos *slaves*, chips com id maior que 0. O chip *master* está conectado a um esp32 executando o algoritmo de receptor que será explicado na seção Implementação

Para teste foi criado 2 versões do código de dispositivos emissores, sendo eles:

1. Sempre envia, quando recebe o ack do último frame tira uma nova foto e envia a mesma.
2. Envia apenas quando solicitado, fica esperando até receber uma mensagem do *master* dizendo para tirar uma nova foto e enviar.

Para testar o recebimento e visualizar de maneira mais simples foi criado um acess point WiFi na qual um usuário final pode se conectar, também é disponibilizado os seguintes endpoints no web server:

- **/lora_img**
  - Objetivo: Mostrar no dispositivo do usuário uma imagem JPEG com a última foto salva no dispositivo;
  - Método: GET;
  - Retorna: image/jpeg
- **/req_img/{}**

- Objetivo: Fazer com que o LoRa mestre envie uma mensagem pedindo ao dispositivo que tire uma foto e envie;
- Parâmetros na url: o id do chip LoRa que vai enviar a imagem;
- Método: GET;
- Retorna: text/plain, indicando se foi possível ou não fazer a requisição.

# 5 Implementação

Visto a proposta de arquitetura supracitada, faz-se necessário implementar dois projetos sendo eles: um transmissor que envia uma imagem separada em diversos quadros utilizando o algoritmo stop and wait, e um receptor que recebe e serve essas imagens para outras aplicações.

## 5.1 Sender

Do lado do *Sender*, dispositivo que captura e envia imagens, em um primeiro momento são inicializadas a câmera e a antena LoRa. Em seguida é feita a captura de uma imagem que passa a ser dividida em partes atendendo ao tamanho máximo que pode ter o payload da mensagem que pode ser enviada pela antena LoRa. Assim, uma vez dividida a imagem inicia o seu processo de envio, no qual se envia o quadro e espera uma resposta. Caso não receba nada em determinado tempo ou receba algo que não seja o "ACK" esperado a mensagem será reenviada. Sendo assim em alto nível o algoritmo de envio:

Algoritmo Stop and Wait - Sender

```
1  void sender() {
2      InitCamera();
3      InitLoRa();
4      TakePicture(image); // tira uma foto
5      SplitImage(image); // separa image em pacotes
6      while(image.sendedParts < image.totalParts) {
7          SendImagePart(image.part()); // envia quadro contendo parte da imagem
8          ReceivePacketCommand  (buffer); // espera ate receber um quadro
9          if(buffer.messageType == ACK) {
10             image.sendedParts++; // incrementa contador de imagens enviadas
11         }
12     }
13 }
```

Dessa forma o fluxo de execução do dispositivo que envia a imagem:

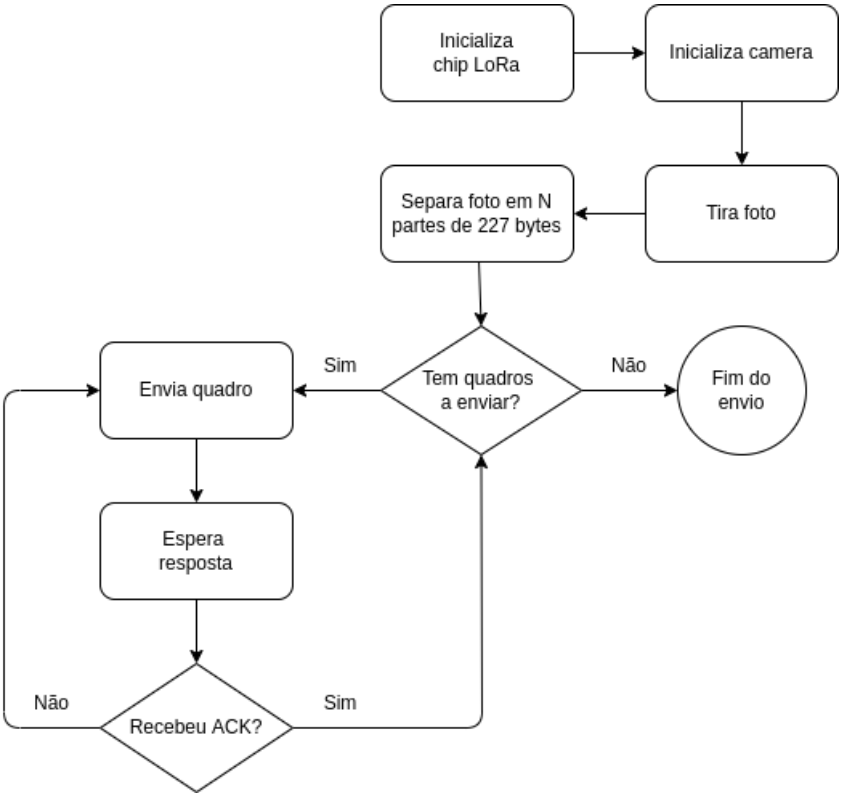


Figura 5 – Fluxograma geral do algoritmo do sender.

## 5.2 Receiver

Por outro lado no *receiver*, em um primeiro momento também são inicializadas as estruturas de controle, bem como é alocado um espaço que serve de buffer para colocar as partes da imagem que está sendo recebida. Após isso, o dispositivo fica esperando o recebimento de pacotes.

Algoritmo Stop and Wait - Receiver

```
1  void receiver() {
2      InitImage(image); // inicializa estrutura de dados das imagens
3      InitLoRa();
4      while(true) {
5          ReceivePacketCommand  (buffer); // espera ate receber um quadro
6          SaveImageBytes(buffer); // salvar bytes na estrutura da imagem
7          PrepareFrameCommand(); // prepara ACK
8          SendPacket(); // envia ACK
9          if(image.isComplete()) {
10             SaveImage(image); // salvar imagem em arquivo
11         }
12     }
13 }
```

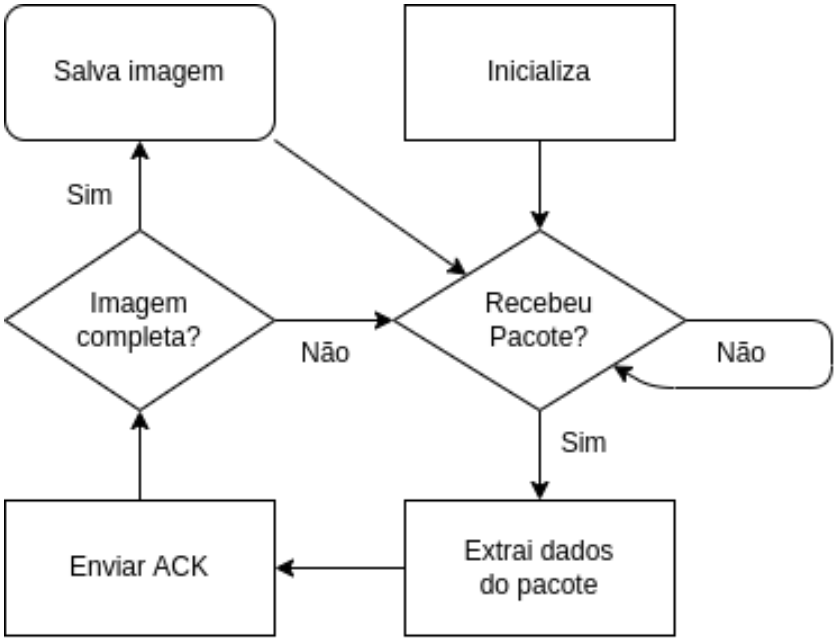


Figura 6 – Fluxograma geral do algoritmo do receiver.

## 5.3 Formato das mensagens

Para enviar mensagens utilizando o chip LoRa, é preciso seguir o formato:

Tabela 1 – Estrutura padrão de mensagens LoRa.

| ID | Command | Payload | CRC |
|---|---|---|---|
| 2 bytes | 1 byte | 1–231 bytes | 2 bytes |

Sendo assim sempre os primeiros 2 bytes de uma mensagem são o id do chip que está enviando, o terceiro byte é o comando, podendo ser para realizar alguma configuração no chip, ler algum dado específico, ou simplesmente para enviar para antena $(40_{10}, 28_{16})$, seguidos de 1 a 231 bytes de mensagem e por fim os últimos 2 bytes são o CRC calculado para a mensagem específica.

Tendo uma quantidade máxima de 231 bytes para escrever os dados da imagem, optou-se pelos seguintes campos de cabeçalho:

Tabela 2 – Estrutura definida para o payload da mensagem.

| Payload | | | | |
|---|---|---|---|---|
| Type | ID | Part | Total | Message |
| 1 byte | 1 byte | 1 byte | 1 byte | 1–227 bytes |

Sendo que:

- **Type**: campo que indica como os próximos bytes devem ser interpretados, podendo ser um ACK, ou uma parte de imagem.
- **ID**: identificador único da imagem;
- **Part**: qual a parte da imagem que está nessa mensagem;
- **Total**: a quantidade total de partes da imagem;
- **Message**: os bytes da imagem

Porém caso o 1 byte, *Type*, seja um "ACK" não terá outros campos de cabeçalho apenas um outro byte dizendo qual parte de imagem está atrelado o "ACK".

Dessa forma foi implementado a seguinte estrutura de dados para enviar as imagens:

Definição da estrutura do payload

```
1  struct _payload {
2      uint8_t byte_array[APPLICATION_MAX_PAYLOAD_SIZE];
3      uint8_t size;
4  };
5
6  struct _fields {
7      uint8_t type;
8      uint8_t id;
9      uint8_t part;
10     uint8_t last_part;
11 };
12
13 union ImagePart {
14     _fields fields;
15     _payload payload;
16 };
```

# 6 Resultados

A fim de avaliar o funcionamento da implementação foram realizados diversos testes, dando especial atenção para a forma como a resolução e a taxa de compressão da imagem influenciam no tamanho da mensagem enviada no payload. Os principais resultados são mostrados a continuação.

No primeiro grupo de testes realizado o foco foi avaliar como a resolução da imagem influencia no tamanho em bytes da mensagem enviada. Para isso seguiu-se os seguintes critérios:

- 3 fotos por resolução escolhendo sempre a mediana.
- Fotos tiradas do mesmo local na mesma posição;
- Taxa de compressão do JPEG em 0;
- Imagens em escala de cinza;

| Resolução (pixels) | Tamanho (bytes) |
|---|---|
| 640×480 | 73260 |
| 480×320 | 39139 |
| 400×296 | 35916 |
| 320×240 | 23510 |
| 240×176 | 14242 |
| 176×144 | 9147 |

Tabela 3 – Mudança de resolução afetando o tamanho da imagem

No segundo grupo de testes o foco foi avaliar a taxa de compressão do JPEG, sendo assim seguiu-se os critérios supracitados porém mantendo a resolução fixa em 480x320 e alterando a perda de qualidade da imagem.

| Qualidade (0–63) | Tamanho (bytes) |
|---|---|
| 0 | 39139 |
| 10 | 8456 |
| 20 | 6371 |
| 30 | 5613 |
| 40 | 5161 |
| 50 | 4842 |
| 60 | 4665 |
| 63 | 4616 |

Tabela 4 – Mudança de qualidade da imagem afetando o tamanho

Avaliando também a taxa de transmissão percebeu-se que utilizando o algoritmo *stop and wait* cada troca de quadros demora aproximadamente 2 segundos sendo assim pode-se calcular o tempo médio de envio de cada imagem dependendo unicamente da quantidade de quadros que a mesma ocupa.

# 7 Conclusão

Apesar de todas as limitações da tecnologia LoRa tais como velocidade de transmissão, direção do canal, ainda é possível transmitir pequenas e comprimidas imagens com alta latência.

Sendo assim possível implementar uma rede estrela para transmissão de conteúdos maiores que o suportado por uma única mensagem LoRa (231 bytes), abrindo possibilidades para envio de imagens, arquivos e grandes mensagens.

# Referências


1 BOR, M.; VIDLER, J. E.; ROEDIG, U. Lora for the internet of things. Junction Publishing, 2016. Acesso em: 25 de julho de 2022. Citado nas páginas 2 e 3.

2 MELO, P. **Introdução ao LPWAN (Low Power Wide Area Network)**. 2017. Disponível em: <https://embarcados.com.br/introducao-ao-lpwan/>. Acesso em: 26 de julho de 2022. Citado na página 2.

3 LORA-ALLIANCE. **LoRaWAN What is it?** 2015. Disponível em: <https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf>. Acesso em: 21 de fevereiro de 2021. Citado na página 3.

4 DAVCEV, D. et al. Iot agriculture system based on lorawan. In: IEEE. **2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)**. [S.l.], 2018. p. 1–4. Citado na página 3.

5 ADELANTADO, F. et al. Understanding the limits of lorawan. **IEEE Communications magazine**, IEEE, v. 55, n. 9, p. 34–40, 2017. Citado na página 3.

6 TZOUNIS, A. et al. Internet of things in agriculture, recent advances and future challenges. **Biosystems engineering**, Elsevier, v. 164, p. 31–48, 2017. Citado na página 3.

7 Chen, S. et al. A vision of iot: Applications, challenges, and opportunities with china perspective. **IEEE Internet of Things Journal**, v. 1, n. 4, p. 349–359, 2014. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6851114>. Acesso em: 16 de janeiro de 2021. Citado na página 3.

8 KIZITO, F. et al. Frequency, electrical conductivity and temperature analysis of a low-cost capacitance soil moisture sensor. **Journal of Hydrology**, Elsevier, v. 352, n. 3-4, p. 367–378, 2008. Citado na página 3.

9 MESAS-CARRASCOSA, F. et al. Open source hardware to monitor environmental parameters in precision agriculture. **Biosystems engineering**, Elsevier, v. 137, p. 73–83, 2015. Citado na página 3.

10 BAUER, J. et al. On the potential of wireless sensor networks for the in-situ assessment of crop leaf area index. **Computers and Electronics in Agriculture**, Elsevier, v. 128, p. 149–159, 2016. Citado na página 3.

11 KARIMI, N. et al. Web-based monitoring system using wireless sensor networks for traditional vineyards and grape drying buildings. **Computers and Electronics in Agriculture**, Elsevier, v. 144, p. 269–283, 2018. Citado na página 3.

12 KATH, J.; PEMBLETON, K. G. A soil temperature decision support tool for agronomic research and management under climate variability: adapting to earlier and more variable planting conditions. **Computers and Electronics in Agriculture**, Elsevier, v. 162, p. 783–792, 2019. Citado na página 3.

13 OLIVER, S. T. et al. **Development of an open sensorized platform in a smart agriculture context: a vineyard support system for monitoring mildew disease**. 2019. Citado na página 3.

14 RADIOENGE. **Módulo LoRaMESH, Manual de Utilização**. 2022. Disponível em: <https://www.radioenge.com.br/storage/2021/08/LoRaMESH_RD42C_Manual.pdf>. Acesso em: 22 de julho de 2022. Citado nas páginas 4 e 5.