Universidade Federal do Rio Grande do Sul
Escola de Engenharia
Departamento de Engenharia Elétrica
Trabalho de Conclusão de Curso

Projeto e Desenvolvimento de Rede de Monitoramento LoRa

Bruno Rudiger

Porto Alegre
2021

Bruno Rudiger

Projeto e Desenvolvimento de Rede de Monitoramento LoRa

Trabalho de Conclusão entregue ao professor orientador e ao regente da atividade, como parte dos requisitos necessários para a aprovação na atividade de Projeto de Diplomação - II.

Universidade Federal do Rio Grande do Sul – UFRGS

Escola de Engenharia

Departamento de Engenharia Elétrica

Projeto de Diplomação

Orientador: Prof. Dr. Paulo Francisco Butzen

Porto Alegre

2021

Bruno Rudiger
Projeto e Desenvolvimento de Rede de Monitoramento LoRa/ Bruno Rudiger. – Porto Alegre, 2021-
55 p.

Orientador: Prof. Dr. Paulo Francisco Butzen

Trabalho de Conclusão de Curso – Universidade Federal do Rio Grande do Sul – UFRGS
Escola de Engenharia
Departamento de Engenharia Elétrica
Projeto de Diplomação, 2021.

Resumo

Redes de monitoramento são fundamentais para a indústria. Elas dão aos gestores de plantas industriais um conhecimento mais profundo do que ocorre nelas. Com estas informações, é possível otimizar o funcionamento de uma empresa através da redução do consumo de energia em horários de ponta e melhor gerir a operação de máquinas, por exemplo. Este trabalho tem por objetivo desenvolver um sistema embarcado de monitoramento remoto utilizando a tecnologia LoRa, uma técnica de modulação de baixa potência e longo alcance, bem como descrever as etapas e boas práticas de desenvolvimento de um sistema embarcado para fins comerciais. O foco é o desenvolvimento de unidades remotas e de um *gateway* LoRa para aplicações industriais de monitoramento através do protocolo ModBus RTU. Além disso, é almejado o desenvolvimento do *firmware* dos equipamentos. Os resultados acerca do alcance e consumo dos dispositivos validam a solução desenvolvida.

Palavras-chave: LoRa. *gateway*. redes de monitoramento. protocolo de comunicação. *wireless*.

Lista de Abreviaturas e Siglas

- **CRC**: *Cyclic redundancy check*,
- **CSS**: *Chirp spread spectrum*.
- **FSK**: *Frequency-shift keying*.
- **GPRS**: *General Packet Radio Service*.
- **I²C**: *Inter-Integrated Circuit*.
- **IDE**: *Integrated Development Environment*.
- **IOT**: *Internet of things*.
- **MQTT**: *Message Queuing Telemetry Transport*.
- **SPIFFS**: *Serial Peripheral Interface Flash File System*.
- **RSSI**: *Received signal strength indicator*.
- **RTC**: *Real-Time clock*.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Com os avanços tecnológicos que possibilitam a criação de novos equipamentos e máquinas num ritmo extremamente acelerado em relação ao passado, surge a necessidade de monitorar seu funcionamento e condições de operação. É útil tanto para o indivíduo que quer, por exemplo, monitorar a temperatura de sua geladeira; quanto para a indústria que deseja acompanhar, com o máximo de informações possíveis, suas máquinas e equipamentos. Inserida neste contexto está a empresa em que desenvolvi meu estágioobrigatório. Ela fornece soluções de monitoramento para a indústria,especializando-se na área energética. Utilizando suas soluções, as empresas têm acesso a informações detalhadas sobre máquinas e, ou, equipamentos e seu consumo energético ao longo do dia, permitindo melhor gerir seu consumo e identificar problemas que, sem essas informações mais detalhadas, não poderiam ser identificados.

Para implementar estas soluções são utilizados *gateways*, que são módulos de processamento que captam informações dos equipamentos a ele conectados. Assim, esses dados podem ser tratados e, ou enviados para a plataforma ou servidor da empresa, operando por rede WiFi, GPRS ou via cabo Ethernet, dependendo do modelo de *gateway*. Estes métodos de comunicação cobrem boa parte das necessidades encontradas pelos clientes. Porém, há casos em que a necessidade de instalação de múltiplas unidades ou, a falta de cobertura da rede celular em uma planta industrial que cubra uma grande área, torna o uso de GPRS inviável quando não é possível o uso de WiFi ou cabo Ethernet.

Num ambiente industrial como este, o roteamento de cabos e condutos para instalação de equipamentos pode ser muito custoso e demorado. Cabos de dados e informações devem ser protegidos para não terem suas informações corrompidas por interferências causadas pela operação das máquinas ao seu redor, além de ser necessário a instalação de novos cabos e condutos toda vez que for preciso uma expansão da rede de equipamentos instalados. Estes obstáculos aumentam muito a atratividade de soluções *wireless*, onde, em geral, existe um custo de instalação inicial elevado e as instalações subsequentes têm como custo apenas o equipamento em si, além de serem de mais rápida instalação.

Com esta situação em mente, surgiu a ideia de se utilizar um número de unidades remotas portadoras do protocolo de comunicação LoRa (Long Range). Este permite que as unidades se comuniquem com um ou mais *gateways* com finalidade exclusiva de enviar os dados fornecidos pelas unidades remotas à uma plataforma *online* de monitoramento. Dentre os protocolos *wireless* mais utilizados, o LoRa se destaca por seu baixo consumo energético, longas distâncias de comunicação, como o próprio nome indica, e robustez do protocolo contra interferências. A utilização do protocolo LoRa permite a instalação de

unidades remotas que comunicam as informações medidas com um *gateway* central. A centralização do processamento dos dados diminui os custos de se expandir uma instalação e permite o uso em ambientes mais hostis para outros protocolos *wireless*, como o WiFi.

O foco deste trabalho são as redes LoRa. Serão desenvolvidos ao longo deste trabalho, o *firmware* e *hardware* para a implementação completa de uma rede de monitoramento LoRa, isto é: o desenvolvimento protótipos de placas e códigos necessários para as unidades remotas e o *gateway* central.

1.1 Objetivos

Nesta seção estão dispostos o objetivo geral e os diferentes objetivos específicos que o compõem. São eles:

- **Desenvolvimento da rede de monitoramento LoRa**: O objetivo final é o desenvolvimento de uma rede de monitoramento LoRa, estando incluso o desenvolvimento de *firmwares* e protótipo do equipamento de monitoramento, bem como do seu *gateway*. Para alcançá-lo, é necessário completar previamente certos objetivos. Eles estão dispostos a seguir em ordem cronológica.

    1. **Desenvolvimento de *firmware* para validar alcance e robustez do protocolo LoRa**: É necessário para verificar a eficácia da solução no contexto de dispositivo para uso em monitoramento industrial. Para iniciar este desenvolvimento é necessário apenas ter em mãos os kits de desenvolvimento.
    2. **Definição dos Periféricos**: Este objetivo envolve a escolha dos periféricos, RTCs ou conversores, por exemplo, necessários para a operação desejada do dispositivo.
    3. **Desenvolvimento de *firmware* para os protótipos**: Após decidir quais periféricos serão incorporados ao protótipo e os adquirir, a montagem de um protótipo em protoboard pode ser iniciada. Com o protótipo montado, seu *firmware* pode ser desenvolvido. Esta etapa permite a validação dos periféricos ligados ao protótipo e da sua integração ao microcontrolador.

Com a conclusão dos objetivos acima, pretende-se validar a viabilidade de uma implementação LoRa em ambientes industriais para a finalidade de monitorar equipamentos e dispositivos de maneira remota.

1.2 Organização do Trabalho

O presente trabalho está dividido em capítulos. Aqui serão rapidamente introduzidos seus conteúdos.

- **Introdução**: São apresentadas a motivação e o objetivo do trabalho assim como uma prévia do restante do trabalho.
- **Conceitos Preliminares**: Neste capítulo estão dispostos conceitos importantes para a compreensão do trabalho.
- **Sistema Desenvolvido**: Descrição do sistema que foi desenvolvido, razão das escolhas e possível continuação do desenvolvimento.
- **Testes Realizados**: Aqui estão descritos os testes de validação do sistema desenvolvido e comparações realizadas, bem como seus resultados.
- **Conclusão**: Por fim, aqui estão dispostas as conclusões ao fim do desenvolvimento do trabalho.
- **Apêndices**: Ao final do documento estão dispostos os apêndices, contendo um glossário de termos utilizados durante o presente estudo, bem como os códigos desenvolvidos.

2 Conceitos Preliminares

Este capítulo apresenta uma ambientação preliminar dos conceitos a serem abordados ao longo do presente trabalho. São eles: as boas práticas e preocupações no desenvolvimento deste projeto, as redes industriais de monitoramento na atualidade, os protocolos de comunicação utilizados para construir tais redes e o protocolo LoRa, um dos focos deste trabalho.

2.1 Boas práticas no desenvolvimento de *softwares* embarcados

Para um desenvolvimento organizado de um *software* embarcado, onde seja possível retomar o trabalho realizado ou que outro desenvolvedor possa ficar a parte do que foi feito, é importante que alguns pontos recebam atenção [1]. Na sequência se encontra uma breve lista de práticas sugeridas.

- Seguir um padrão de nomenclatura de variáveis e funções. Isso ajuda na identificação de funcionalidades, parâmetros e retornos de funções.
- Isolar ao máximo cada uma das funcionalidades desenvolvidas. Com isso evita-se que modificações causem um efeito cascata de mudanças indesejadas no restante do código.
- Compilar funcionalidades similares em uma biblioteca. Isso facilita a localização de funções e ajuda na portabilidade destas para outros projetos.
- Não utilizar bibliotecas de terceiros sem ter conhecimento de que e como estas operam. Isso evita incompatibilidades e comportamentos inesperados ao utilizar funções destas.
- Evitar o uso de variáveis globais. Em um sistema embarcado os recursos de memória são escassos e variáveis declaradas de maneira global alocam memória a todo momento.
- Evitar a implementação de bibliotecas desnecessárias. Em um sistema embarcado os recursos de armazenamento são escassos, logo deve-se utilizar bibliotecas que são amplamente utilizadas no código ou desenvolver as funções que se deseja.
- Desenvolver funções da maneira mais seccionada possível. Isso evita grandes linhas de código que poderiam ser representadas por uma função, mantendo o código mais estruturado.

- Manter o menor nível de complexidade necessário. Pela falta de recursos no sistema embarcado operações complexas podem demorar e afetar outras operações.
- Documentar de maneira clara e uniforme o que cada função implementada faz. Focando nos parâmetros de entrada e seu retorno, a documentação permite que o trabalho seja retomado posteriormente por outros desenvolvedores.

2.2 Redes Industriais de Monitoramento

Os avanços na miniaturização de dispositivos eletromecânicos trouxeram módulos operados a bateria capazes de sensoriamento, comunicação e de realizar processamento de dados [2]. Estes dispositivos podem ser arranjados em uma rede de monitoramento e processamento, provendo um acompanhamento contínuo em locais hostis, perigosos ou inóspitos.

Tais redes são, geralmente, divididas em duas categorias. O primeiro tipo são as redes centralizadas, onde os módulos se comunicam com um módulo central com uma maior capacidade de processamento e que conecte esta rede à outra rede, como a internet. Em contraste às redes centralizadas, as redes distribuídas são compostas de módulos capazes de operar individualmente, realizando, de maneira autônoma, a comunicação com uma ou mais redes distintas [3].

As redes centralizadas têm, na maior parte das instalações, um maior custo inicial associado a sua instalação, em comparação com uma rede distribuída. Essa diferença se dá pois seu módulo central é de maior complexidade e, consequentemente mais caro, e ele é necessário para qualquer instalação. O custo da solução se reduz conforme o aumento do número de dispositivos na rede, pois os módulos de monitoramento tem uma construção mais simples e seu custo unitário é, consequentemente, menor.

As redes distribuídas, porém, são mais confiáveis em razão de que seus módulos podem operar independentemente uns dos outros. Não dependem do funcionamento de um módulo central para que enviem dados a outra rede.

Numa rede centralizada, o módulo central é, geralmente, um *gateway*. Um *gateway* é um equipamento que permite o fluxo de dados entre redes diferentes. Isto é, coletam dados da rede de monitoramento formada pelos demais módulos e os projeta para uma rede distinta, onde estes dados serão armazenados e/ou processados. No caso de uma rede distribuída, cada equipamento tem a funcionalidade de *gateway*.

2.3 Comunicação em Redes Industriais

As redes de monitoramento industriais requerem, majoritariamente, o uso de protocolos de comunicação robustos e de longo alcance. Buscando suprir essas necessidades, foram criados, adaptados e testados diversos protocolos e normas. Destes, destacam-se alguns protocolos e normas citados abaixo.

2.3.1 Comunicação com Fio

As primeiras normas para comunicação industrial utilizavam fios e cabos para a troca de dados entre equipamentos. Alguns destes ficaram obsoletos com os avanços das tecnologias de redes cabeadas e sem fio, no entanto ainda são amplamente utilizadas.

2.3.1.1 RS485

A norma RS485 define um esquema de transmissão de dados balanceado que oferece uma solução robusta para transmissão de dados por longas distâncias em ambientes ruidosos [4]. RS485, além de especificar o enlace, não define qual o protocolo a ser utilizado para a comunicação dos dados, é uma especificação da camada física de diversos protocolos, como o ModBus, ProfiBus e DIN-Measurement-Bus, sendo este primeiro o foco das comunicações entre o equipamento e o módulo remoto LoRa descrito neste trabalho.

A norma descreve uma interface operando em linhas diferenciais e capaz de se comunicar com até 32 "unidades de carga". Cada dispositivo conectado à rede RS485 corresponde, geralmente, a uma "unidade de carga", porém existem dispositivos que consomem uma fração de "unidade de carga", possibilitando a ligação de um maior número de dispositivos. A rede é considerada *half-duplex*, isto é: em certo instante de tempo, apenas um dispositivo pode transmitir, estando desligados os transmissores dos demais dispositivos.

A norma RS485 é caracterizada pelo uso de um meio de comunicação diferencial via par trançado. Os circuitos transmissores e receptores utilizam a diferença dos níveis de tensão de cada condutor para obter a informação transmitida. O uso deste técnica resulta no cancelamento de ruídos introduzidos no meio de transmissão, pois o mesmo ruído será introduzido em ambos os condutores e a diferença entre eles o eliminará.

O limite máximo de distância de comunicação é de 1200m. A taxa de transmissão de dados utilizada é inversamente proporcional à distância de comunicação possível entre dispositivos.

2.3.1.2 ModBus

ModBus é um protocolo de comunicação para transmissão de informação entre dispositivos eletrônicos e é comumente utilizado em processos industriais. Originalmente desenvolvido tendo em vista a comunicação por linhas seriais, foi estendida para comunicação via Ethernet [5].

O protocolo serial é um protocolo mestre-escravo, onde cada rede é composta de um mestre e diversos escravos que respondem a comandos endereçados a eles. Cada rede pode conter até 247 escravos, cada um com um endereço único. Os dados dos dispositivos são armazenados em registradores de 16 bits. Cada mensagem ModBus RTU é composta de até 256 bytes, incluindo seus bytes de endereçamento, identificador de função e CRC (*circular checksum*).

2.3.1.3 Outros Protocolos

Foram também criados protocolos com intenção exclusiva de uso industrial, como é o caso do WirelessHART, que foi concebido com a intenção de migrar facilmente redes de monitoramento que utilizavam o padrão 4-20mA para uma rede sem fio [6]. É totalmente compatível com o protocolo hart, permitindo uma transição gradual para a rede sem fio. O protocolo utiliza o método TDMA (time-division multiple access), que permite que diversos equipamentos se comuniquem na mesma frequência ao dividir o canal no tempo em diferentes partes. Também é feito uso de *channel-hopping*. Estes dois fatores dão grande segurança e robustez ao final, porém aumentam o custo dos módulos que necessitam um preciso controle de tempo para se comunicar corretamente com seu módulo central. O protocolo também sofre de baixo alcance.

2.3.2 Comunicação sem fio

Com as mudanças do monitoramento com fios das redes industriais para o uso de redes sem fio, surgiu a necessidade da criação de protocolos de comunicação que fossem seguros contra tentativas de coleta de dados por terceiros, robustos para se operar em ambientes com interferência eletromagnética e de baixa potência para que os equipamentos tenham uma maior vida útil de sua bateria e menor manutenção [7]. Entretanto, não são todos os protocolos de comunicação sem fio que atendem a todas estas especificações. Dentre os protocolos de comunicação mais utilizados, os mais conhecidos são o WiFi, BlueTooth e GSM celular. Embora sejam protocolos de extensa aplicação, seu uso em ambientes industriais é reduzido. Nesta seção abordamos alguns exemplos.

2.3.2.1 WiFi

O WiFi, baseado no IEEE 802.11, permite um grande fluxo de dados e múltiplas conexões simultâneas. No entanto, por utilizar a faixa de 2.4GHz de frequência, que é de grande utilização por ser uma faixa de uso universal, para suas conexões, pode sofrer interferência de outros equipamentos que utilizam a mesma frequência. Seu sinal tem um alcance curto e seu consumo energético é alto. No entanto, redes centralizadas podem fazer uso do WiFi em seu módulo central para enviar dados coletados pelos demais dispositivos para outra rede via internet.

2.3.2.2 *BlueTooth*

O BlueTooth também sofre com as baixas distâncias de comunicação do WiFi, porém é mais robusto quanto à interferência causada por outros dispositivos que atuam na mesma faixa de frequência. Isto se dá pois o protocolo faz uso de *frequency-hopping*, onde o canal de comunicação é alterado após certo intervalo de tempo previamente definido, numa frequência de até 1600 vezes por segundo. As versões mais recentes do protocolo também evitam a utilização de canais que já estão sendo utilizados por outros equipamentos.

2.3.2.3 Redes celular

As redes de celular, como 3G e 4G, permitem a transmissão de dados a longas distâncias caso haja uma antena de alguma operadora na região. Por ser necessário o uso da infraestrutura oferecida pelas operadoras de telefonia, seu custo de operação é elevado e a quantidade de dados transmitidos é limitada. O protocolo não permite a criação de um *mesh* ou de uma rede centralizada por terceiros. Estas características favorecem seu uso em locais onde serão instalados poucos equipamentos, usualmente em locais com pouca infraestrutura para a instalação de outros tipos de rede.

2.3.3 O protocolo LoRa

Dentre os protocolos de comunicação, este trabalho se aprofunda no LoRa(Long Range). O espalhamento espectral LoRa é uma forma de modulação desenvolvida e patenteada pela Semtech, baseada no CSS [8]. O LoRa proporciona uma transmissão a longas distâncias, com baixo consumo energético, segura e baixa taxa de transmissão de dados. Também pode ser integrado às redes existentes, permitindo a implementação de soluções IOT e da Indústria 4.0, operando por baterias a um baixo custo [9].

A modulação por espalhamento espectral LoRa utiliza o CSS para codificar os dados a serem transmitidos. Cada bit é espalhado por um fator de *chirping*, onde sua frequência aumenta ou diminui com o tempo. Pequenos fatores de espalhamento aumentam a taxa de transmissão de dados, enquanto altos fatores aumentam a robustez do sinal.

A modulação LoRa é mais resistente a ruídos de fundo. Em vez de utilizar apenas as duas frequências de uma transmissão FSK(frequency shift key), ela varre todas as frequências que existem entre elas.

Estes fatores dão à modulação LoRa as seguintes características: Alta robustez, resistência a múltiplos caminhos, resistência a Doppler, longo alcance, localização, entre outros. As características de baixo consumo, baixo custo, longo alcance e robustez do sinal, tornam o protocolo LoRa útil em diversas aplicações. Entre elas: telemetria e gerenciamento energético, controle de rebanhos de gado, segurança residencial, controle de nível de caixas d'água, monitoramento de vazamentos de gás, monitoramento de temperatura, entre outros [10] [11] [12].

Um dos maiores custos associados à implementação de redes sem fio é o módulo central/*gateway*, pois este geralmente concentra todo o processamento e gerenciamento dos dados obtidos no sistema. Em decorrência de seu longo alcance, um *gateway* é capaz de cobrir uma grande área, logo são necessários menos *gateways* para gerenciar uma rede que está distribuída em uma grande área.

O LoRa opera em frequências que não necessitam de licenças para operar, em frequências abaixo de 1GHz, como 433MHz, 868MHz, 915MHz e 923MHz. O LoRa fornece uma grande cobertura, mesmo em ambientes fechados, grande vida útil de bateria e baixo custo. Estas características fazem do LoRa uma excelente escolha ao se implementar soluções IOT e para a indústria 4.0, desde que não haja a necessidade de um alto fluxo de dados ou comunicação em tempo real.

2.3.3.1 Detalhes e Preocupações

Tendo em vista o desenvolvimento de um produto que será comercializado e as limitações do protocolo LoRa na definição do meio físico como *broadcast*, deve-se atentar aos seguintes detalhes:

- Criptografia: O protocolo LoRa, em seu meio físico, não possui criptografia, apenas o protocolo LoRaWAN [13]. Como a rede LoRaWAN não será utilizada, não é possível utilizar suas capacidades de criptografar as mensagens enviadas e recebidas. Tendo em foco as questões levantadas, deve ser implementada uma criptografia a fim de evitar a interceptação dos pacotes em trânsito.
- Identificador de origem e destinatário: Como a comunicação se dará por *broadcast*, é necessário que todo pacote contenha um identificador de qual dispositivo está enviando a mensagem, bem como o destinatário da mensagem, para que o dispositivo possa tratar adequadamente as mensagens destinadas a ele.

- Limite de tamanho: Cada *payload* LoRa é limitado a 256 *bytes*, logo mensagens que ultrapassem este limite deverão ser desmembradas.
- Confirmação de recebimento: O destinatário de uma mensagem deve informar ao remetente que a mensagem foi recebida, caso contrário a mensagem será reenviada.

2.3.4 Protocolo MQTT

MQTT é um protocolo de envio e recebimento de mensagens sobre a rede TCP/IP. Ele é caracterizado por publicar as mensagens em tópicos nos quais dispositivos inscritos as recebem, em contraste com protocolos onde a mensagem é enviada diretamente para um ou mais destinatários [14]. Sua estratégia de comunicação garante um nível elevado de segurança para dispositivos, pois nenhum dispositivo tem informações sobre outros dispositivos, apenas sobre os tópicos em que publica e está inscrito.

2.4 Estado da Arte

Segundo [15], o crescimento da Indústria 4.0 e Internet das Coisas traz a necessidade de uma integração contínua do menor sensor utilizado aos sistemas de nível empresarial. Esta solução precisa ser simples, conveniente, confiável e econômica. Com estas características em foco, o LoRa se destaca, dentre outros protocolos de comunicação, como ZigBee, BlueTooth, WiFi e 3G/4G, pela confiança e robustez de suas implementações. É possível implementar um *gateway* LoRa com uma placa Raspberry PI e uma placa LoRa externa conectada a este.

Segundo [16], nos ambientes industriais, são transmitidos, em geral, pequenos pacotes de dados a longas distâncias. Por consequência a robustez, confiança, eficiência energética e latência são os principais fatores de performance a serem avaliados. O LoRaWAN, uma camada MAC sobre o nível físico LoRa, foi criado tendo em vista comunicações esporádicas entre um grande número de módulos, com ele não é possível realizar aplicações *real-time*. Inicialmente, o foco da implementação LoRa a ser desenvolvida é o de monitoramento de medidores, onde não se faz necessário o uso de aplicações *real-time*. No entanto, com a expansão de uma rede, pode-se fazer necessário o uso de alguma aplicação *real-time* para que os módulos atuem sobre o sistema ao invés de apenas coletarem dados. [16] complementa o pensamento, ao propor a criação de um MAC para LoRa que tenha capacidade de ser utilizado em aplicações *real-time*.

Segundo [17], uma rede de sensores sem fio é caracterizada por módulos de pequeno porte, baixo consumo energético e baixo custo. As redes necessitam ter uma rápida instalação, baixa manutenção e escalabilidade. Avaliando a performance de uma plataforma de monitoramento baseada em LoRa com seu *gateway* tanto ao ar livre quanto em ambientes

fechados, a eficiência do LoRa foi comprovada em seu impacto na eficiência energética e confiança da comunicação, mesmo em ambientes hostis. A implementação de uma rede baseada no protocolo LoRa, descrita por [17], em um ambiente de infraestrutura precária e de estresse nos equipamentos, mostrou como as soluções que fazem uso de LoRa são confiáveis em ambientes que estão expostos a ambientes hostis.

3 Sistema Desenvolvido

Neste capítulo estão dispostos a metodologia, as práticas adotadas e equipamentos utilizados na elaboração deste trabalho, assim como o ambiente de desenvolvimento do projeto.

3.1 O microcontrolador ESP32

No desenvolvimento do sistema, foi utilizado o microcontrolador ESP32. O ESP32 é uma família de microcontroladores com WIFI e bluetooth integrados. Está disponível em versões *single* e *dual-core*. Possui, também: chaves de antenas, amplificadores de baixo ruído, filtros e módulos de gerenciamento energético. É criado e desenvolvido pela Espressif Systems com base em Xangai.

Um microcontrolador ESP32 de uso geral possui:

- CPU XTENSA *dual-core* de 32 bits operando de 160MHz a 240MHz.
- Coprocessador *ultra low power*.
- 520KiB de SRAM, 448KiB de ROM.
- WiFi 802.11 b/g/n.
- Bluetooth 4.2.
- 34 GPIOs programáveis
- 18 entradas com ADC de 12 bits.
- 2 DACs de 8 bits.
- 10 sensores de toque (GPIOs de sensoriamento capacitivo)
- 4 SPIs.
- 2 canais I^2S.
- 2 canais I^2C.
- 3 UARTs programáveis.
- MAC Ethernet.
- Sensor Hall.

- Flash encriptado.
- Boot Seguro.
- Segurança IEEE 802.11.
- *Low-Dropout* interno.
- $5\mu A$ de corrente no modo sono profundo (*deep-sleep*).
- *Wake-up* por GPIO, *timer*, medição do ADC e sensor capacitivo.

Alguns benefícios de se utilizar o ESP32:

- Possibilidade de se trabalhar com a IDE Arduino, permitindo o uso de suas bibliotecas.
- Baixo custo.
- Grande controle sobre o dispositivo.
- Fácil integração com a internet, graças ao WiFi imbutido.
- Possibilidade de se criar um *access point* WIFI para fácil configuração de parâmetros do dispositivo.

Seu custo, facilidade de integração com a internet, possibilidades de desenvolvimento com IDE Arduino e capacidade de processamento foram os atrativos para seu uso neste projeto.

3.2 Visão Geral do Sistema

O sistema possui três hierarquias distintas durante sua operação. A primeira na comunicação por protocolo MQTT, a segunda na comunicação LoRa e a terceira na rede RS485. A estrutura adotada está representada na Figura 1 que apresenta as etapas da comunicação do *broker* até o dispositivo.

3.2.1 Hierarquia de Funcionamento MQTT

- *Slave*(*gateway*): Recebe perguntas publicadas em tópicos em que está inscrito, tratando comandos recebidos e publica dados próprios ou de equipamentos abaixo de si em tópicos adequados.
- *Master* (Serviço MQTT): Publica *payloads* em tópicos em um *broker* MQTT que serão tratados pelos *gateways* e se inscreve em tópicos para recebimento de dados publicados pelos *gateways*.

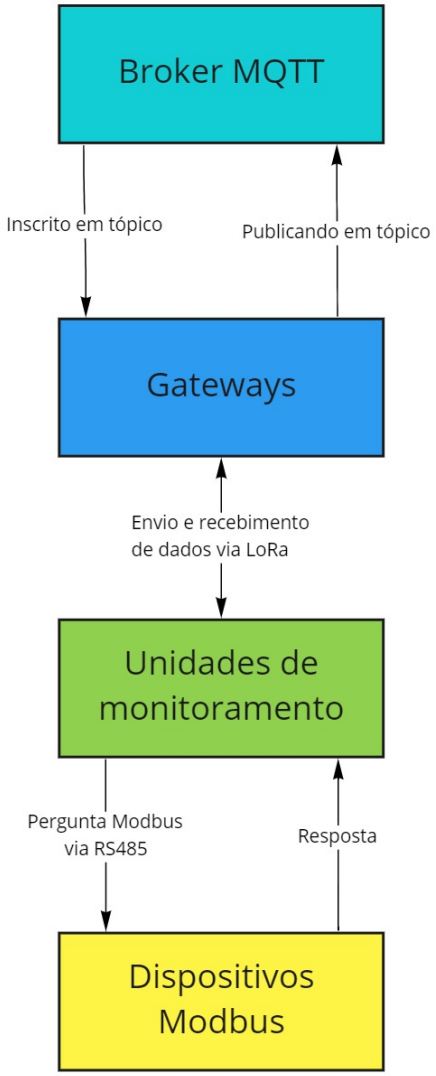
3.2.2 Hierarquia de Funcionamento LoRa

- *Slave*(unidade de monitoramento): Recebe dados *Modbus* via RS485, envia dados obtidos para seu *gateway* via LoRa a cada intervalo fixo de tempo, ao ser solicitado ou em eventos específicos. Se não receber OK do destinatário, reenvia, em nova falha salva na memória não volátil.
- *Master* (*gateway*): Recebe dados do *slave* via LoRa e responde com OK. Salva mensagem na memória e tenta publicar no *broker* MQTT, em caso de sucesso, deleta mensagem, em 'n' falhas a mantém salva até receber comando de envio. Periodicamente atualiza o RTC e publica *status* próprio.

3.2.3 Hierarquia de Funcionamento da Rede RS485

- *Slave*(equipamento dotado de comunicação *Modbus* via RS485): Responde às perguntas e comandos endereçados a ele.
- *Master* (unidade de monitoramento): Envia perguntas e comandos para um equipamento com endereço *Modbus* conhecido e trata a resposta.

Figura 1 – Visão geral da estrutura hierárquica do sistema



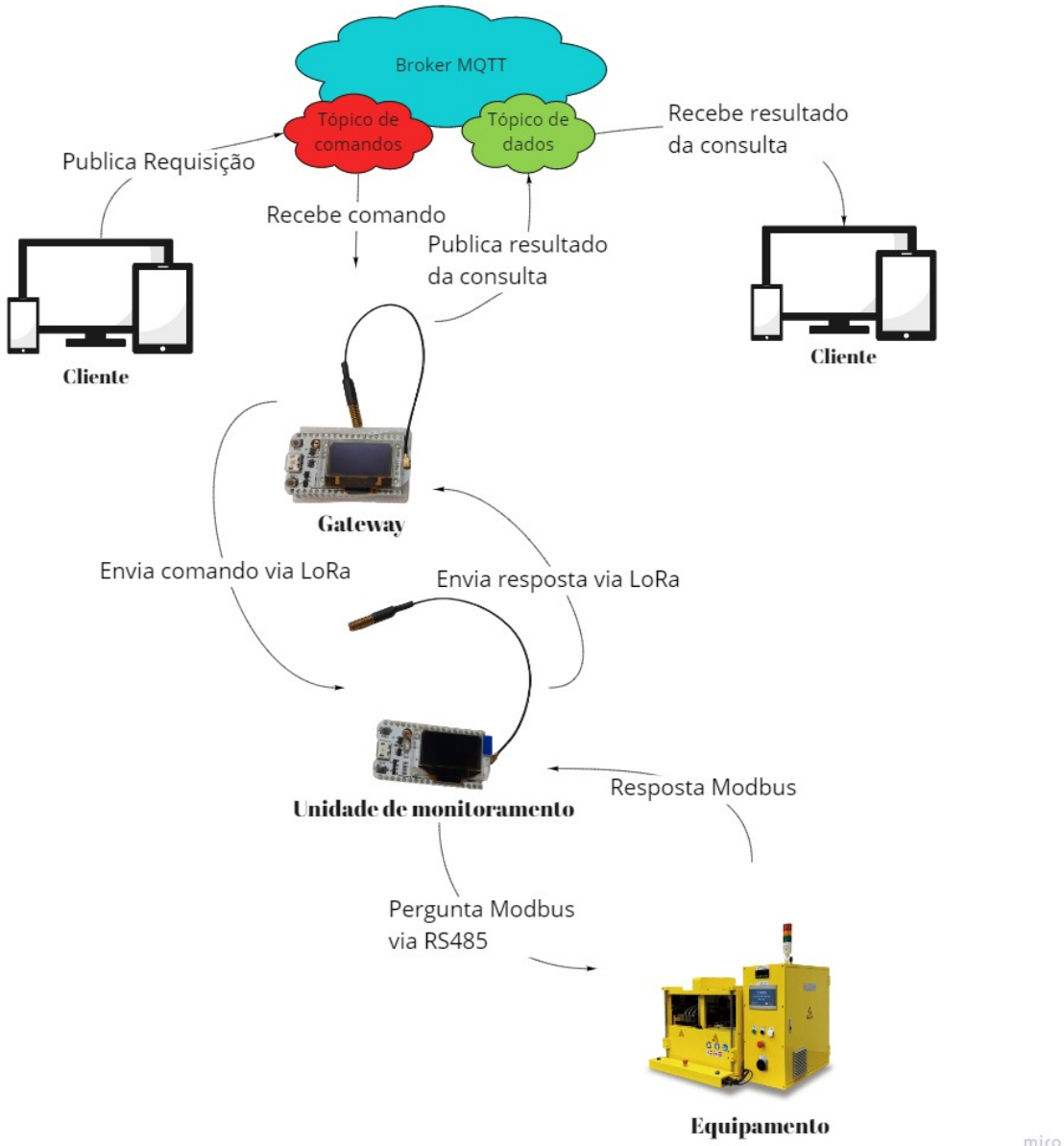
3.2.4 Integração das Etapas de Comunicação

Conforme a estrutura hierárquica disposta anteriormente, é possível descrever uma estrutura de integração que une todas elas para formar um sistema completo de aquisição de dados remotamente. Para tanto, a implementação realizada está descrita a seguir e disposta na Figura 2 que apresenta um diagrama que reflete a operação do sistema.

Um serviço remoto publica um *payload* contendo um comando de leitura de dados em um tópico específico para um dado *gateway*. Por sua vez, o *gateway* que está inscrito em seu tópico específico, recebe o *payload* e identifica uma mensagem endereçada a ele contendo um identificador único de uma unidade de monitoramento RS485, um endereço Modbus de um equipamento conectado à unidade, um registro inicial de leitura e um número de registros a serem lidos, esta mensagem é tratada e reestruturada, identificando como destinatário a unidade desejada, para enfim ser enviada via LoRa. As unidades de

monitoramento recebem a mensagem e verificam se está endereçada a ela, a unidade à qual a mensagem está endereçada monta, então, uma mensagem em formato Modbus que será enviada via RS485 com um endereço de um equipamento presente na rede e quais registros serão lidos. A resposta, ou falta de, via RS485 é tratada e enviada via LoRa para o *gateway* que, por sua vez, trata e publica os resultados da consulta em um tópico referente ao equipamento solicitado.

Figura 2 – Exemplo de integração da comunicação



3.3 Ambiente, Software e Hardware Base

Durante a elaboração do sistema descrito na seção 3.2, foram utilizados os seguintes equipamentos, *softwares* e ambiente de desenvolvimento:

HELTEC DevKit: Placa de desenvolvimento para o microcontrolador ESP32 com um CI LoRa e *display* LCD. Possui todos os componentes necessários para prototipação com ESP32 e LoRa. Contém estruturas que permitem o uso de uma interface USB para fácil gravação de *firmware*, leitura de dados via porta serial, LEDs de status e indicação de comunicação serial, conexão de antena para uso do protocolo LoRa, fonte de alimentação 3.3 Volts para alimentação do microcontrolador, *display* LCD, conexão com bateria externa para alimentação, entre outras.

Conversor serial - RS485: O módulo RS485, equipado com o CI MAX485, permite a conversão da comunicação RS485, com seus pares diferenciais, para um protocolo serial, com conectores de transmissão, recepção e terra. Este módulo é usado na comunicação entre o módulo de medição e o equipamento/simulador. O protocolo RS485 é de amplo uso em ambientes industriais. Este conversor foi escolhido por sua faixa de alimentação e fácil aquisição.

Conversor RS485 - USB: De maneira similar ao conversor serial - RS485, este conversor permite o envio e recebimento de mensagens RS485 via porta USB. Ele é de grande utilidade durante o desenvolvimento de uma solução que envolva RS485, pois com ele é possível validar os dados que estão sendo enviados pelo dispositivo embarcado, assim como é possível simular o funcionamento de um equipamento industrial a fim de se observar a operação do dispositivo em desenvolvimento. O conversor adquirido faz uso do CI CH340 para conversão dos dados.

Computador: Necessário para desenvolvimento do *software* embarcado a ser gravado nos dispositivos, realizar suas gravações, simular equipamentos e monitorar a comunicação entre dispositivos.

Visual Studio Code (VSCode): IDE da Microsoft que possui integração com GIT, permitindo um melhor controle de versionamento dos códigos produzidos. Também é possível utilizar extensões para integrar a IDE com a IDE Arduino e com a ESP-IDF desenvolvida pela Espressif Systems para desenvolvimentos em ESP32.

Arduino IDE: Utilizada para integração com VSCode e para gravação do *firmware* do equipamento, bem como aquisição de bibliotecas.

3.4 Metodologia para o Desenvolvimento de um Sistema Embarcado

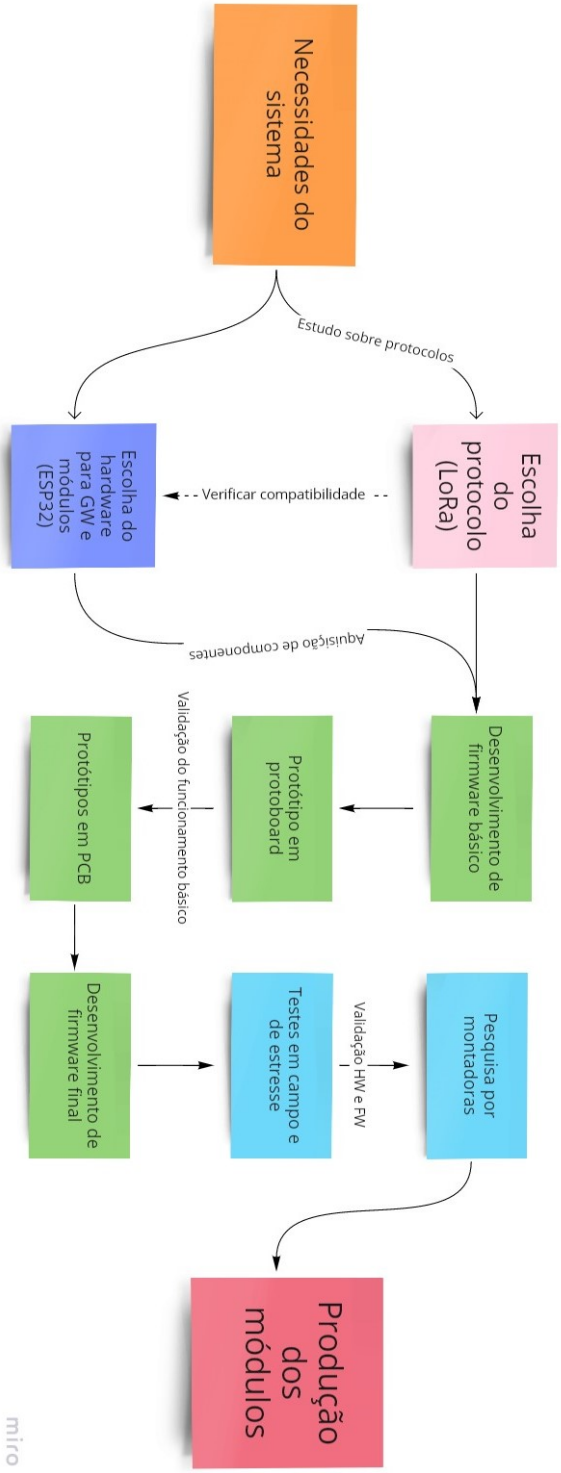
Nesta seção estão dispostos os materiais utilizados e também as etapas de desenvolvimento realizadas visando projetar, executar e produzir um sistema embarcado de monitoramento com o uso do protocolo LoRa. Ela é dividida em três etapas, sendo elas: pré-execução, execução e pós-execução.

A fase de pré-execução inclui a razão da escolha de protocolos e seleção de componentes necessários para a montagem do equipamento. A execução, por sua vez, compreende o desenvolvimento de um *firmware* base, a montagem de um protótipo em *protoboard*, criação de um *firmware* "final"[1], projeto do equipamento em placa de circuito impresso e testes funcionais e de estresse. Por fim, a pós execução engloba os passos consequentes que levam à produção de um produto comercial.

Na Figura 3 é possível visualizar o fluxo das atividades a serem, idealmente, desenvolvidas.

[1] Chamo de "final"pois o desenvolvimento do *firmware* é um processo constante, e esta denominação se dá à versão onde todas as funcionalidades previstas estão implementadas inicialmente.

Figura 3 – Fluxograma de Desenvolvimento de *Firmware* Embarcado



3.4.1 Pré-execução

Listados a seguir estão as etapas da pré-execução.

1. **Identificação da Necessidade**: O primeiro passo antes de qualquer desenvol-

vimento é a identificação da demanda suprida por ele. Assim, temos o seguinte desenvolvimento: A empresa onde o autor realizou a atividade de estágio obrigatório, a Allexo Tecnologia, fornece serviços de monitoramento e telemetria de energia e dados de equipamentos elétricos. Assim, dispõe de *gateways* que realizam estas tarefas em conjunto com sua plataforma online. A equipe de vendas da empresa constatou que certo número de possíveis clientes não realizavam a aquisição dos equipamentos ou serviços ofertados por dois motivos, sendo eles: o alto custo da compra e instalação do sistema, a dificuldade de operação dos equipamentos em locais remotos com cobertura precária dos serviços de telefonia móvel, utilizados para comunicação dos *gateways* com a plataforma de serviços.

Para resolver este problema, iniciou-se a procura por alternativas. Tal alternativa necessitaria atender a certos requisitos. São eles:

- Robustez para operar em ambientes industriais, com interferência eletromagnética;
- Utilizar uma tecnologia *wireless* para evitar custos de rotear cabos e fios de comunicação;
- Baixo custo para instalação de um grande número de equipamentos;
- Longo alcance de comunicação;

Com estes requisitos em mente, num primeiro momento foi feita a escolha de um protocolo de comunicação adequado. WiFi e GPRS já estavam inclusos na linha de equipamentos ofertada, e não atendem a essas necessidades. Como o próprio nome indica, o protocolo LoRa (Long Range) oferece um grande alcance sem fio de comunicação, e foi o primeiro protocolo analisado. Para avaliar suas capacidades de suprir os requisitos listados acima, foi adquirido um módulo de desenvolvimento da empresa Heltec com o microcontrolador ESP32 com um chip LoRa integrado. Ele foi escolhido por já possuir o chip LoRa integrado, ter uma pequena tela, e ser baseado no microcontrolador ESP32, pois este já é utilizado em outras linhas de equipamentos e o autor está familiarizado com seu uso.

Com as placas em mãos, iniciaram-se os testes de alcance da comunicação, consumo energético e robustez do sinal. Estes testes são necessários para validar a capacidade e eficiência do protocolo LoRa em operar junto ao microcontrolador ESP32. Para avaliar estes quesitos, foi elaborado um teste básico. O autor desenvolveu um par de simples *firmwares* na IDE Arduino que permitiam o equipamento gravado com um desses *firmwares* transmitir dados, enquanto outro recebia os dados enviados e os apresentava na tela integrada à placa. Em seguida, modificou-se o equipamento gravado com o *firmware* de transmissão, seccionando sua trilha de alimentação para a inserção de um resistor *shunt*. Tal modificação permite o monitoramento da

corrente consumida pelo equipamento ao medir a tensão sobre o resistor com um osciloscópio ou outro medidor de tensão e conhecendo o valor da sua resistência. O equipamento gravado com o *firmware* de recepção foi conectado a uma bateria e afastado gradualmente do transmissor. Este foi levado a distâncias superiores a 2 km, locais fechados e próximo a motores. Os testes confirmaram a capacidade do protocolo LoRa, junto ao microcontrolador ESP32, de satisfazer os requisitos de alcance, força do sinal e consumo energético necessários para a aplicação desejada. Assim, o LoRa foi escolhido como protocolo a ser adotado.

2. **Seleção dos componentes**: O próximo passo foi a seleção de componentes. Tais componentes foram utilizados na confecção de um protótipo em *protoboard* com todos os periféricos necessários para o funcionamento final do equipamento. A montagem deste protótipo foi necessária para validar o funcionamento de todos os componentes e verificar a compatibilidade entre eles. Os componentes selecionados são preferencialmente encapsulados de maneira a facilitar sua conexão com a *protoboard*.

3.4.2 Execução

Nesta seção estão dispostas as etapas necessárias para o desenvolvimento, implementação e testagem dos equipamentos.

1. **Desenvolvimento de *firmware* base**: Para validar a compatibilidade dos componentes adquiridos e seu funcionamento, foi preciso realizar testes que utilizassem todas as funcionalidades de maneira conjunta. Com este objetivo em mente, um *firmware* que faz uso de tais componentes deve ser desenvolvido. Este foi utilizado, em conjunto com o protótipo em *protoboard*, para confirmar a viabilidade da criação do produto desejado. O *firmware* foi desenvolvido utilizando a plataforma IDE Arduino, fazendo uso de bibliotecas desenvolvidas anteriormente por outros usuários dos componentes ou pelo autor.

2. **Montagem de protótipo em *protoboard***: Com os componentes adquiridos à disposição, foi iniciada a montagem do protótipo em *protoboard*. Este protótipo permitiu: avaliar o consumo energético do equipamento final, para a seleção de uma fonte de alimentação adequada e escolher a melhor maneira de conectar os componentes ao microcontrolador utilizado, neste caso o ESP32, para ser enviado ao projetista atribuído com a tarefa de criar o projeto de placa do equipamento caso este seja comercializado. Nesta etapa também foram escolhidas possíveis interfaces de informação de funcionamento para o usuário, geralmente LEDs para indicar que o equipamento está ligado, conectado à plataforma, se comunicando entre outras informações que sejam de interesse do usuário.

3. **Teste e validação do protótipo**: Preparado o protótipo e o *firmware* base, foram feitos testes para validar as escolhas de componentes e as funcionalidades básicas destes. Com estas funcionalidades validadas, o projeto de uma placa de circuito impresso (PCB, *printed circuit board*) e o desenvolvimento do *firmware* "final"do equipamento, onde todas as funcionalidades previstas são implementadas, podem ser concluídos. Estes testes avaliam a comunicação, velocidade de processamento, consumo energético, entre outros.

4. **Criação de *firmware* "final"**: Após a validação das funcionalidades básicas do equipamento, foi iniciado o desenvolvimento do *firmware* "final"do equipamento, isto é, um *firmware* onde todas as funcionalidades estão implementadas. Este processo integra todas as funcionalidades básicas testadas anteriormente criando um código que:

   - Executa a leitura de dados provenientes tanto dos equipamentos a este conectados como o *gateway* central, no caso dos módulos de monitoramento;
   - Executa a leitura de dados provenientes da plataforma online e dos módulos de monitoramento no caso do *gateway* central;
   - Trata os dados recebidos e os prepara, montando um arquivo Json por exemplo, para envio à plataforma ou ao *gateway* central;

   Para a elaboração do *firmware* destes equipamentos, é desejável desenvolver uma série de bibliotecas que facilitam o desenvolvimento e implementação do restante do firmware. Estas o farão mais compreensível e permitirão um mais rápido desenvolvimento de futuros equipamentos que utilizem a mesma linguagem de programação, neste caso, a linguagem C.

   Bibliotecas permitem uma mais simples implementação de tarefas complexas ao desmembrar um problema maior em pequenas etapas. Desta maneira, tarefas de uso rotineiro podem ser facilmente acessadas e utilizadas após serem desenvolvidas apenas uma vez.

   As bibliotecas desenvolvidas estão listadas abaixo. Todo o código das mesmas está no Apêndice B.

   - WIFI: Esta biblioteca irá conter as funções que tratam a conexão à redes WIFI e funcionalidades relacionadas, como: reconexão, desconexão, informação das redes disponíveis, intensidade do sinal, etc.
   - RS485: A biblioteca de comunicação RS485 tratará a inicialização da porta serial responsável pela comunicação via canal RS485, bem como o tratamento de mensagens recebidas por esta porta. Também serão tratados os envios de mensagens e as mudanças de padrão de comunicação serial, como a *baudrate* e paridade de bits.

- ModBus: Esta tratará as mensagens recebidas e enviadas utilizando os protocolos ModBus RTU e ModBus TCP. Podendo ser tanto via RS485 ou WiFi.
- MQTT: Esta biblioteca tratará da conexão ao *broker* MQTT utilizado no desenvolvimento desta solução, a estruturação adequada dos tópicos utilizados para a publicação dos dados obtidos e a formação dos *payloads* das mensagens publicadas.
- RTC e NTP: Esta biblioteca fará o tratamento da atualização do relógio em tempo real (RTC) e sua inicialização.
- LoRa: Responsável pelo tratamento de mensagens recebidas e enviadas via protocolo LoRa.

Com o fim desta etapa, o desenvolvimento realizado durante a elaboração deste trabalho se conclui. No restante deste capítulo serão tratadas as etapas a serem realizadas caso se deseje produzir um equipamento para fins comerciais

5. **Projeto de PCB**: Com todas as funcionalidades e componentes selecionados, o próximo passo neste desenvolvimento é o projeto da placa de circuito impresso do equipamento. Nesta etapa é feita a integração dos componentes selecionados, bem como a adição de fontes internas de alimentação, se necessário, e de interfaces para facilmente verificar o bom funcionamento do equipamento. Este projeto deve ser feito levando em conta que trilhas de potência devem ser afastadas de trilhas de comunicação, que o encapsulamento final do produto e possíveis blindagens contra radiação e interferência eletromagnética sejam incorporadas.

6. **Encomenda do protótipo em montadora**: Com o projeto e escolha do encapsulamento finalizados, é escolhida uma montadora de circuitos para confeccionar as placas. À ela é feita a encomenda de um pequeno número de peças, geralmente de 3 a 5, que serão utilizadas para a realização de testes de estresse e funcionais.

7. **Testes de estresse e funcionais**: Nesta etapa, as peças recebidas da montadora são testadas em laboratório, procurando falhas no projeto. São realizados testes de estresse, verificando os limites de operação do equipamento para: tensão de alimentação, temperatura de operação, interferência eletromagnética, humidade, entre outros. Os testes funcionais são realizados preparando os equipamentos para sua operação usual e mantendo-os operando por longos períodos de tempo e verificando anomalias. Todos os pontos de interesse são registrados para verificação posterior de erros e anomalias.

3.4.3 Pós-execução

Nesta seção estão dispostas as fases de desenvolvimento pós confecção dos módulos.

1. **Modificações**: Com todos os testes concluídos, todas as falhas e inconsistências observadas são utilizadas para modificar o projeto de placa e *firmware* do equipamento. Este é um processo contínuo de aprimoramento do projeto, a medida que o equipamento é utilizado por outros usuários e possíveis clientes, seu *feedback* é utilizado para encontrar problemas ou possíveis melhorias a serem implementadas em versões futuras do projeto.

2. **Testes em campo**: Com os testes realizados em laboratório validando o funcionamento adequado do equipamento, o próximo passo é a instalação do sistema desenvolvido em sua aplicação final. Da mesma maneira como foram realizados os testes em laboratório, os equipamentos são testados e monitorados procurando anormalidades.

3. **Criação do manual de instalação e uso**: Todos os procedimentos e especificações de instalação do sistema são registrados. Essas anotações são utilizadas na elaboração de um manual de fácil compreensão por parte do usuário, que é possivelmente leigo quanto ao funcionamento de equipamentos de monitoramento, para que este possa realizar a instalação, configuração e uso do sistema.

4. **Implementação do sistema**: O passo final é a implementação e uso do sistema por um usuário. São informados os passos e instruções necessários para sua instalação e uso. As dificuldades e problemas encontrados pelo usuário na instalação e uso do sistema são registrados para modificações futuras ao manual de usuário.

4 Testes Realizados

Para validar a implementação do sistema desenvolvido para seu propósito, o de monitorar equipamentos inseridos em um ambiente industrial, foram realizados três testes. O primeiro teve como objetivo validar a funcionalidade do sistema como um todo, isto é: garantir que é possível adquirir, remotamente, dados de dispositivos portadores de comunicação ModBus via RS485 com o uso do protocolo LoRa para transmitir os dados à um *gateway*. O segundo visava avaliar o alcance da comunicação via protocolo LoRa dos dispositivos desenvolvidos, bem como os efeitos de obstáculos sobre o alcance. Finalmente, o terceiro teste buscava avaliar o consumo dos dispositivos, mostrando como um *payload* LoRa poderia ser enviado por uma fração do consumo de uma publicação MQTT via WiFi.

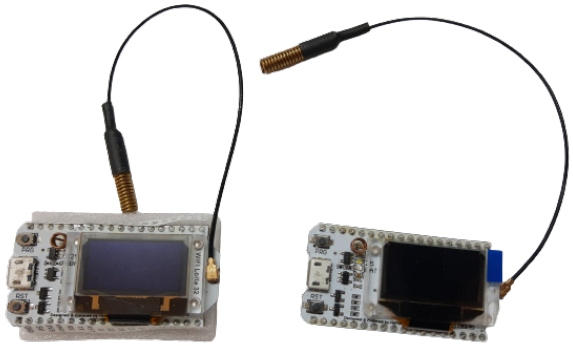
4.1 Validação do Sistema

Para validar a operação integrada do sistema como um todo, foi montado um teste assim como descrito anteriormente na seção 3.2.4. Este teste tem como objetivo demonstrar a capacidade do sistema de adquirir e enviar dados remotamente.

Para tal montagem foram utilizados os seguintes itens:

- Um computador.
- Uma *Protoboard*.
- Dois kits de desenvolvimento LoRa+ESP32 da empresa Heltec, apresentados na Figura 4.
- Duas antenas adequadas para a frequência de operação dos módulos LoRa presentes nos kits de desenvolvimento.
- Um módulo conversor serial para RS485.
- Um módulo conversor RS485 para USB.
- Dois cabos USB de alimentação para os kits de desenvolvimento.
- *Jumpers* para efetuar conexões na *protoboard*.
- Cabos para a comunicação RS485.

Figura 4 – Módulos ESP32 + LoRa utilizados para testes e prototipagem



Além destes equipamentos, foram também utilizados um *broker* MQTT gratuito, o FLESPI e os *softwares* MQTT Explorer, ModBus Slave e Putty, sendo estes três últimos um cliente MQTT utilizado para publicar e receber mensagens do sistema, um simulador de dispositivo ModBus para fornecer respostas adequadas às perguntas feitas pela unidade de monitoramento e um monitor serial para verificar o funcionamentos tanto do *gateway* como da unidade de monitoramento. Todos os desenvolvimentos de *firmware* foram realizados na IDE Visual Studio Code.

Após o desenvolvimento do *firmware* tanto para o *gateway*, quanto para a unidade de monitoramento, os equipamentos foram configurados da seguinte maneira:

1. O *gateway* recebeu as credenciais de uma rede WIFI para poder se conectar à ela.
2. As credenciais do *broker* MQTT FLESPI, um nome de tópico para publicar os dados coletados e um nome de tópico para receber comandos remotos, como apresentado na Figura 5, que apresenta a configuração do *gateway* para se conectar ao *broker*.
3. A frequência de operação da antena do módulo LoRa do kit de desenvolvimento e um número identificador único.

A unidade de monitoramento foi configurada com a mesma frequência de operação do módulo LoRa do *gateway* e um número identificador distinto.

Figura 5 – Configuração das credenciais do broker

```
/* configuração MQTT */
const int mqtt_port = 1883;
const char mqtt_broker[] = "mqtt.flespi.io";
const char mqtt_user[] = "1B23EN3r75AqW5CBVwKUr651LIX29XK1nCSRdL68EvoqqPR99pUqjNfSl2LlOla1";
const char mqtt_password[] = "broker";
const char mqtt_topic[] = "gw_lora/data";
const char mqtt_topic_sub[] = "gw_lora/commands";
```

Com os dispositivos configurados, foi feita a conexão com o *software* Modbus Slave via adaptador USB-RS485, conforme apresentado na Figura 6, que mostra a configuração da conexão serial, e este foi configurado com valores numéricos em seus registros ModBus, conforme a Figura 7, que apresenta uma série de dados simulando os disponibilizados por um dispositivo Modbus.

Figura 6 – Conexão com ModBus Slave

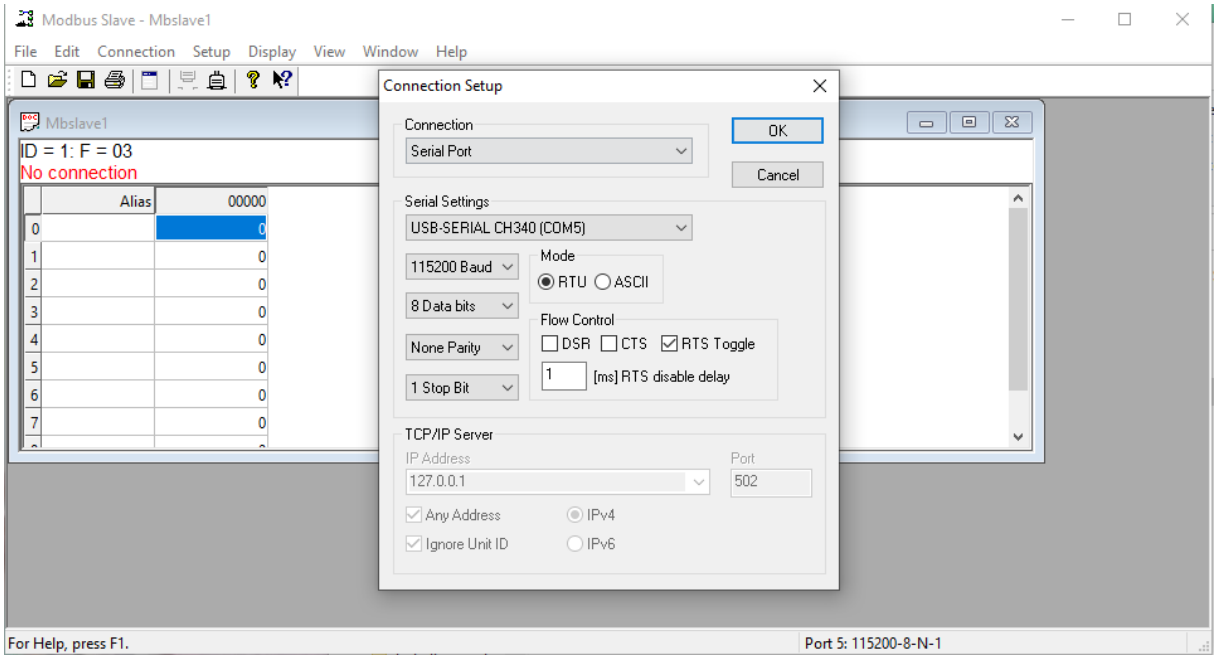
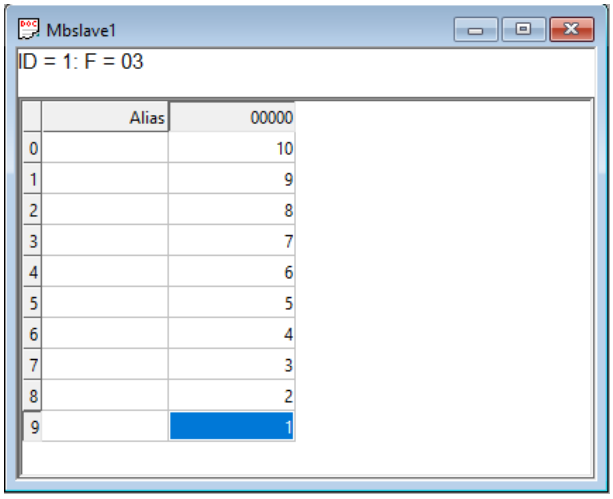
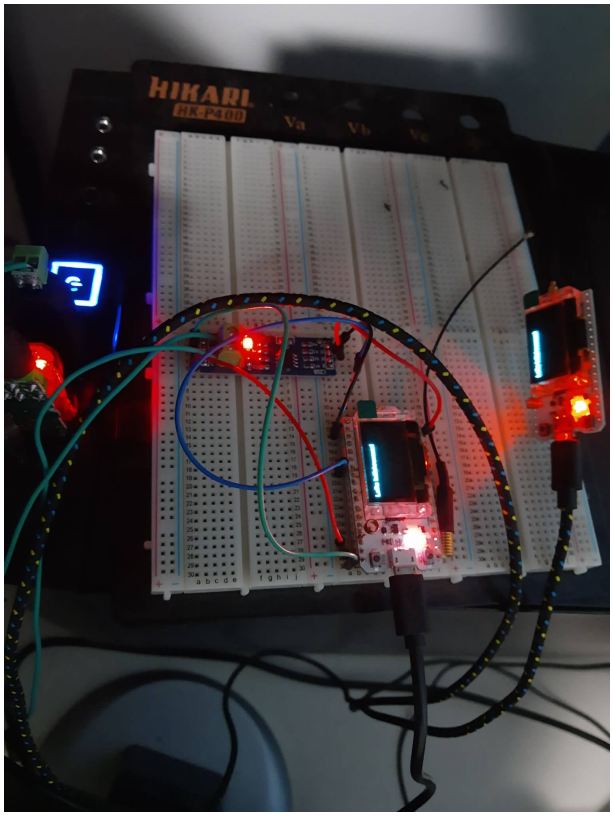


Figura 7 – Mapa de variáveis ModBus Slave



A construção do sistema descrito pode ser observada na Figura 8, que apresenta o protótipo construído para aquisição de dados via RS485, bem como o *gateway*.

Figura 8 – Sistema construído



O cliente MQTT Explorer foi utilizado para se conectar ao mesmo *broker* que o *gateway*, como se observa na Figura 9, que mostra a configuração do *software* para se conectar a um *broker*, e se inscrever nos tópicos de interesse, apresentados na Figura 10, que mostra os tópicos e dados conforme disposto pelo MQTT Explorer. Foi, então, publicado um comando no formato *Json* no tópico em que o *gateway* está inscrito contendo o identificador da unidade de monitoramento, um endereço ModBus inicial e o número de registros a serem consultados da tabela ModBus do equipamento simulado pelo *software* ModBus Slave, assim como apresentado na Figura 11, que apresenta um comando para o *gateway* no formato Json.

Figura 9 – Conexão ao *broker* via MQTT Explorer

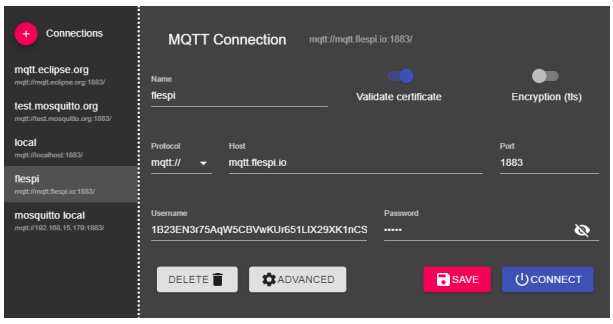


Figura 10 – Tópicos Inscritos

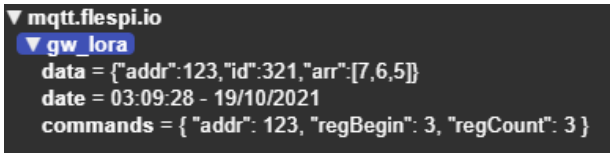
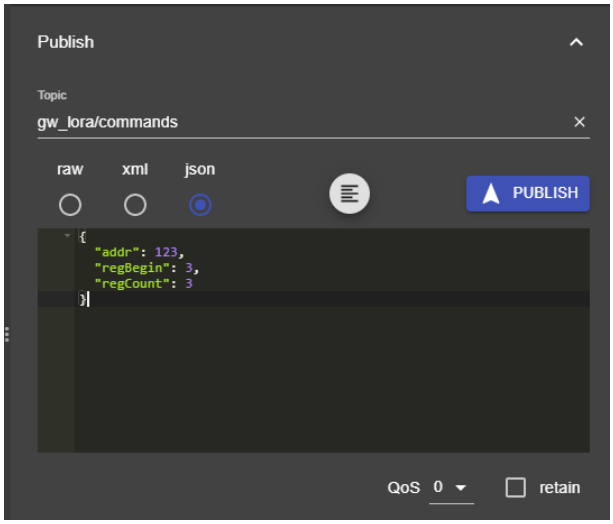


Figura 11 – Pergunta ModBus



Após alguns instantes, a mensagem publicada no tópico é recebida pelo *gateway* que, por sua vez, envia uma mensagem via LoRa contendo seu identificador único, o identificador da unidade de monitoramento a qual está sendo feita a pergunta e as informações do registro inicial e quantidade de registros a serem consultados. Esta mensagem é recebida pela unidade de monitoramento que verifica se esta é destinada a si e então realiza a pergunta via RS485 para o dispositivo simulado. A resposta, ou falta dela, é tratada adequadamente, formando um *array* de dados, caso a pergunta ao dispositivo simulado tenha sucesso, ou uma mensagem de erro no caso de falha. A resposta tratada é formatada dentro de um Json contendo o identificador de origem da pergunta formando um *payload* a ser enviado via LoRa. A formatação destes *payloads* podem ser observadas nas Figuras 12 e 13 que apresentam respostas de sucesso e de falha ao tentar ler dados do dispositivo.

Figura 12 – Payload de Sucesso

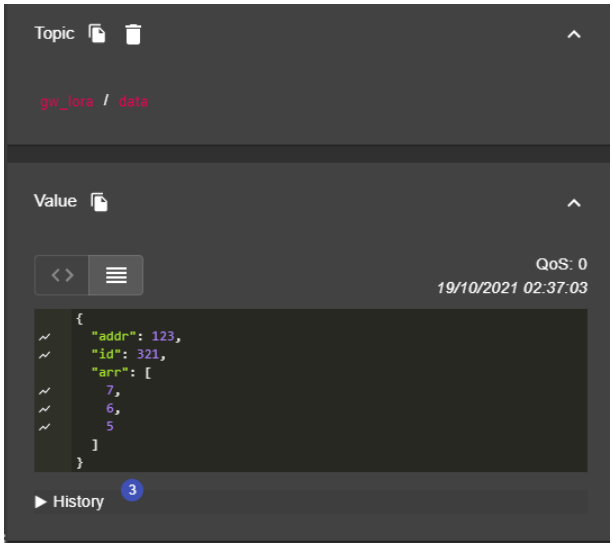
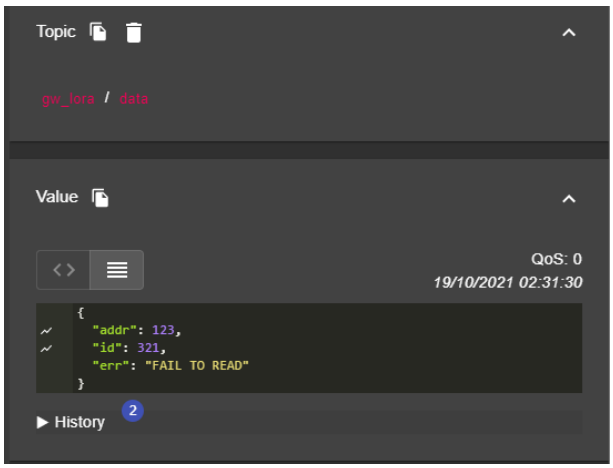


Figura 13 – Payload de Falha



Finalmente, ao receber um *payload* LoRa, o *gateway* verifica se a mensagem está endereçada a ele, conforme seu identificador, e, se a origem é de uma unidade de monitoramento, publica seu conteúdo no tópico de dados do *broker*.

Estas operações no *gateway* e na unidade de monitoramento podem ser acompanhadas via monitor serial, conforme se observa nas Figuras 14 e 15, que apresentam dados impressos nos terminais seriais do *gateway* e da unidade de monitoramento, respectivamente, após receber um comando do *broker*.

Figura 14 – Operação no *Gateway*

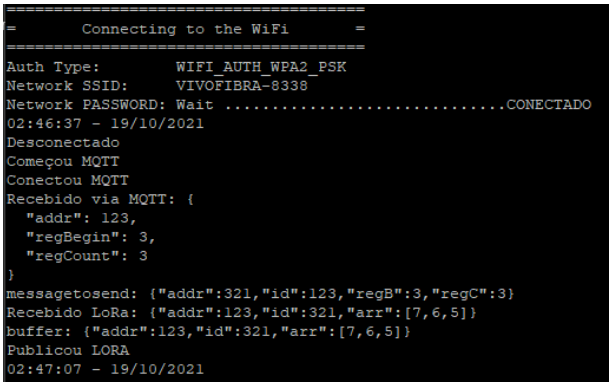
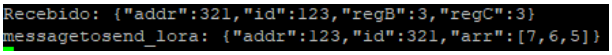


Figura 15 – Operação na unidade de monitoramento



Com estes resultados, se constatou que o sistema opera conforme desejado e entrega resultados em um período curto de tempo, mesmo se utilizando um *broker* localizado na Europa. O maior intervalo de tempo registrado durante a operação foi entre a publicação do comando no tópico e a recepção pelo *gateway*. Este pode ser reduzido ao se operar um *broker* MQTT em um servidor hospedado em local mais próximo ou ainda pode ser eliminado ao se operar a unidade de monitoramento de maneira autônoma, isto é: enviando periodicamente dados adquiridos ou, em um evento predeterminado, dados de uma variável analisada.

4.2 Alcance do Sinal

Para validar a utilização do sistema desenvolvido em locais desprovidos de infraestrutura adequada, como para a instalação e utilização de redes WiFi, via Ethernet ou via rede celular, foram realizados dois testes. O primeiro a ser realizado foi um teste de alcance, tendo como finalidade avaliar a quais distâncias os equipamentos poderiam ser instalados. O segundo teste teve como objetivo avaliar os efeitos de obstáculos na intensidade do sinal recebido.

A métrica de avaliação do sinal das mensagens recebidas foi o RSSI. O RSSI é uma métrica da qualidade relativa do sinal que representa sua potência em dBm.

Mais informações acerca da montagem e resultados obtidos para ambos os testes estão descritos mais detalhadamente nas seções 4.2.1 e 4.2.2.

Para ambos os testes foram utilizados os seguintes materiais:

- Um celular com um aplicativo cliente MQTT.

- Dois kits de desenvolvimento LoRa+ESP32 da empresa Heltec.
- Duas antenas adequadas para a frequência de operação dos módulos LoRa presentes nos kits de desenvolvimento.
- Dois cabos USB de alimentação para os kits de desenvolvimento.
- Um *power bank* para alimentar a unidade de monitoramento.

Assim como no teste descrito na seção 4.1, foi utilizado o *broker* FLESPI para a publicação de mensagens e inscrição em tópicos.

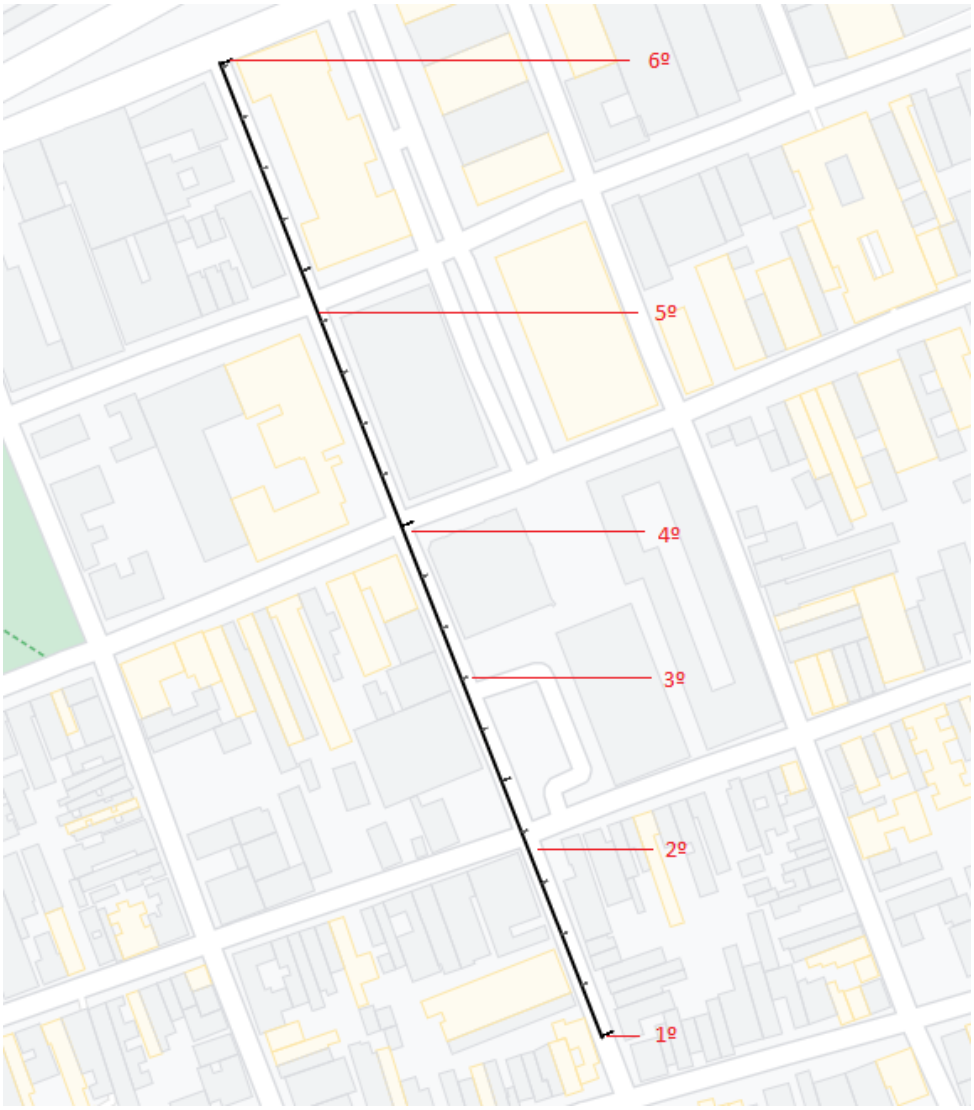
4.2.1 Teste de alcance

Para avaliar o alcance de operação do sistema, a unidade de monitoramento foi conectada a um *power bank* para fornecer a alimentação necessária para seu funcionamento e o *gateway* foi posicionado em um local com elevação de 15 metros com acesso a internet via rede WiFi. O cliente MQTT foi utilizado para enviar comandos ao tópico em que o *gateway* está inscrito para que este, por sua vez, possa enviar um *payload* LoRa para a unidade de monitoramento.

As medidas de RSSI foram tomadas em seis pontos distintos ao longo da rua General Canabarro, no centro de Porto Alegre, Rio Grande do Sul. Os locais em que foram realizadas as medidas e a ordem estão indicadas na Figura 16. A primeira medida foi realizada imediatamente abaixo do *gateway*. As distâncias entre os dois pontos foram calculadas considerando a elevação e a distância plana entre os pontos.

De acordo com a configuração descrita no parágrafo anterior, a unidade de monitoramento foi posicionada numa extremidade dos trajetos indicados enquanto o *gateway* foi posicionado na extremidade oposta. Foi avaliado o RSSI do sinal recebido pela unidade de monitoramento. O RSSI foi também medido em um ambiente fechado a uma distância de 3 e 6 metros.

Figura 16 – Localização dos pontos de medição



Após posicionar a unidade de monitoramento e o *gateway* nas diferentes configurações de distância, foram obtidos três valores de RSSI para cada posicionamento. Tais dados estão dispostos na Tabela 1.

Tabela 1 – RSSI por distância

Distância [m]	RSSI [dBm]	medida 1	medida 2	medida 3	média
3		-32	-36	-35	-34,33
6		-48	-45	-47	-46,67
16		-55	-61	-57	-57,67
79		-69	-65	-66	-66,67
142		-70	-76	-74	-73,33
203		-79	-81	-79	-79,67
287		-82	-81	-78	-80,33
383		-84	-81	-77	-80,67

Ajustando a curva dos dados das distâncias e dos RSSI médios, utilizando como base a equação $RSSI[dBm] = (10nln10(d)A)$, como descrita em [18], onde 'A' é a potência em dBm recebida do sinal à um metro de distância, 'n' é o parâmetro de perda do meio em que os dispositivos se encontram em dBm por metro e 'd' é a distância em metros entre os dispositivos, obteve-se a equação abaixo fazendo uso da ferramenta *cftool* do *software* MatLab, que permite realizar o ajuste à curva a partir de uma equação base e de uma sequência de pontos obtidos experimentalmente.

$$RSSI[dBm] = (100.9229ln10(d[m]) + 28.21[dBm]) \tag{4.1}$$

Realizando a plotagem dos pontos obtidos experimentalmente e a curva regida pela equação 4.1, tem-se o gráfico disposto na Figura 17.

Figura 17 – Pontos obtidos e curva de regressão

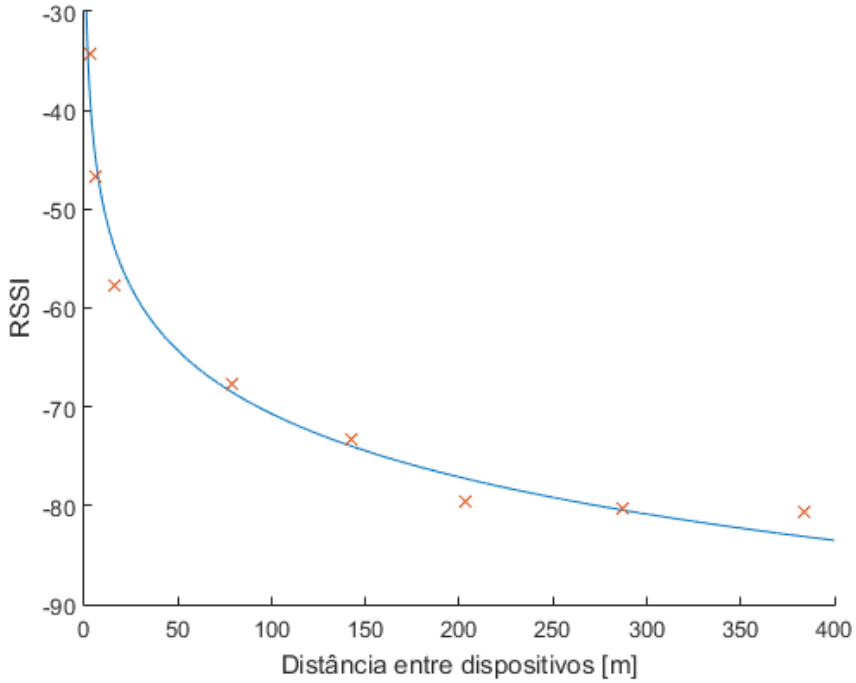


Gráfico gerado via MatLab

4.2.2 Teste interferência de obstáculos

A queda no RSSI produzido por paredes e outros obstáculos localizados entre os dois dispositivos também foi avaliada. Para tanto, foram realizadas três medidas com uma distância de 5 metros entre os dispositivos, que estavam dispostos dentro de um corredor de alvenaria de 1,48 metro de largura por 2.23 metros de altura para estabelecer uma base de valor e, então, os dispositivos foram separados à mesma distância com a inclusão de alguns obstáculos que obstruíam este corredor à 1 metro de distância do receptor. A diferença entre o RSSI obtido a partir de tais medidas e o valor de base foi registrado.

o RSSI no recebimento do sinal foi medido três vezes. Os dados obtidos estão dispostos na Tabela 2. Os obstáculos selecionados para as medidas foram: paredes de alvenaria; portas de madeira; portas compostas de duas chapas de aço de 1,4 mm de espessura afastadas de 7 cm. Sua seleção se deu por estes obstáculos serem materiais comuns em instalações e parques industriais.

Tabela 2 – Perda de RSSI por obstáculo

Obstáculo	RSSI [dBm]	medida 1	medida 2	medida 3	média
Parede de alvenaria de 20cm		-63	-63	-65	-63,67
Porta de madeira de 5cm		-54	-57	-56	-55,67
Porta de aço de 7cm		-67	-71	-66	-68,00

Considerando uma distância de referência de 6 metros com RSSI médio de -46,67, foi calculada a diferença entre os valores médios com e sem o obstáculo. As quedas de RSSI causadas pelos obstáculos foram de: 17 dBm para a parede de alvenaria, 9 dBm para a porta de madeira e 21,33 dBm para a porta de aço.

4.2.3 Conclusões

Como observado e validado pelo ajuste dos pontos, o RSSI do sinal cai conforme os equipamentos se afastam um do outro. No entanto, a taxa com a qual o RSSI cai é cada vez menor conforme a distância entre os dispositivos cresce. Extrapolando a curva obtida pelo ajuste dos pontos, pode-se calcular que o RSSI chegará a -100 dBm a uma distância próxima de 2500 metros, valores entre 0 e -100 dBm são considerados bons para o envio de mensagens LoRa. Considerando as perdas de RSSI causadas por obstáculos, pode-se ter um alcance real de cerca de 1500 metros.

4.3 Consumo de uma transmissão LoRa

Para avaliar a portabilidade de sistemas equipados com módulos LoRa se torna necessário a medição do consumo dos dispositivos que compõem o sistema. Existem diversas maneiras, tanto diretas quanto indiretas, de medir o consumo de um dispositivo.

Um destes métodos é do resistor *shunt*. Para se avaliar o consumo dos dispositivos durante a transmissão e recepção de dados, deve-se abrir o circuito de alimentação do equipamento e inserir um resistor *shunt* em série com o circuito. Com o *shunt* inserido, pode-se então avaliar a tensão sobre este e, conhecendo sua resistência, calcular a corrente consumida pelo dispositivo. Ao se avaliar o consumo via *shunt* é possível apenas saber o consumo instantâneo.

Para realizar a medição como descrita, são necessários:

- Osciloscópio.
- Ferro de solda.
- Resistor *shunt*.
- Cabo de alimentação.
- Módulo a ser avaliado.

Embora a medição via *shunt* seja totalmente válida, foi escolhida a medição pela queda da tensão de uma bateria de lítio. Esta avaliação é útil em casos em que se deseja descobrir o consumo total de um envio de mensagem, por exemplo, no entanto estes dados já são disponibilizados pelos fabricantes dos dispositivos, como disposto nas Figuras 18 a 20, que apresentam tabelas de consumo. No caso da avaliação do sistema completo, a medição da queda da tensão da bateria se torna mais prática.

Figura 18 – Consumo da unidade WiFi do devkit ESP32

Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	225	-	mA
Transmit 802.11b, CCK 11 Mbps, POUT = +18.5 dBm	-	205	-	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	-	160	-	mA
Transmit 802.11n, MCS7, POUT = +14 dBm	-	152	-	mA
Receive 802.11b, packet length = 1024 bytes, -80 dBm	-	85	-	mA
Receive 802.11g, packet length = 1024 bytes, -70 dBm	-	85	-	mA
Receive 802.11n, packet length = 1024 bytes, -65 dBm	-	80	-	mA
Receive 802.11n HT40, packet length = 1024 bytes, -65 dBm	-	80	-	mA

Fonte: Datasheet ESP32

Figura 19 – Consumo da unidade LoRa do devkit ESP32

Symbol	Description	Conditions	Min	Typ	Max	Unit
IDDSL	Supply current in Sleep mode		-	0.2	1	uA
IDDIDLE	Supply current in Idle mode	RC oscillator enabled	-	1.5	-	uA
IDDST	Supply current in Standby mode	Crystal oscillator enabled	-	1.6	1.8	mA
IDDFS	Supply current in Synthesizer mode	FSRx	-	5.8	-	mA
IDDR	Supply current in Receive mode	*LnaBoost* Off, band 1 *LnaBoost* On, band 1 Bands 2&3	- - -	10.8 11.5 12.0	- - -	mA
IDDT	Supply current in Transmit mode with impedance matching	RFOP = +20 dBm, on PA_BOOST RFOP = +17 dBm, on PA_BOOST RFOP = +13 dBm, on RFO_LF/HF pin RFOP = + 7 dBm, on RFO_LF/HF pin	- - - -	120 87 29 20	- - - -	mA mA mA mA

Fonte: Datasheet SX1278

Figura 20 – Consumo do devkit ESP32

Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 13 dBm ~ 21 dBm	160 ~ 260 mA
	Wi-Fi / BT Tx packet 0 dBm	120 mA
	Wi-Fi / BT Rx and listening	80 ~ 90 mA
	Association sleep pattern (by Light-sleep)	0.9 mA@DTIM3, 1.2 mA@DTIM1
Modem-sleep	The CPU is powered on.	Max speed: 20 mA
		Normal speed: 5 ~ 10 mA
		Slow speed: 3 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	0.15 mA
	ULP sensor-monitored pattern	25 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	2.5 μA

Fonte: Datasheet ESP32

As baterias de íons de lítio tem uma característica muito útil para medições de consumo. Ao se descarregar, a tensão da bateria cai na medida em que é consumida a carga da bateria. Esta característica permite aproximar a carga restante da bateria ao se medir a tensão desta.

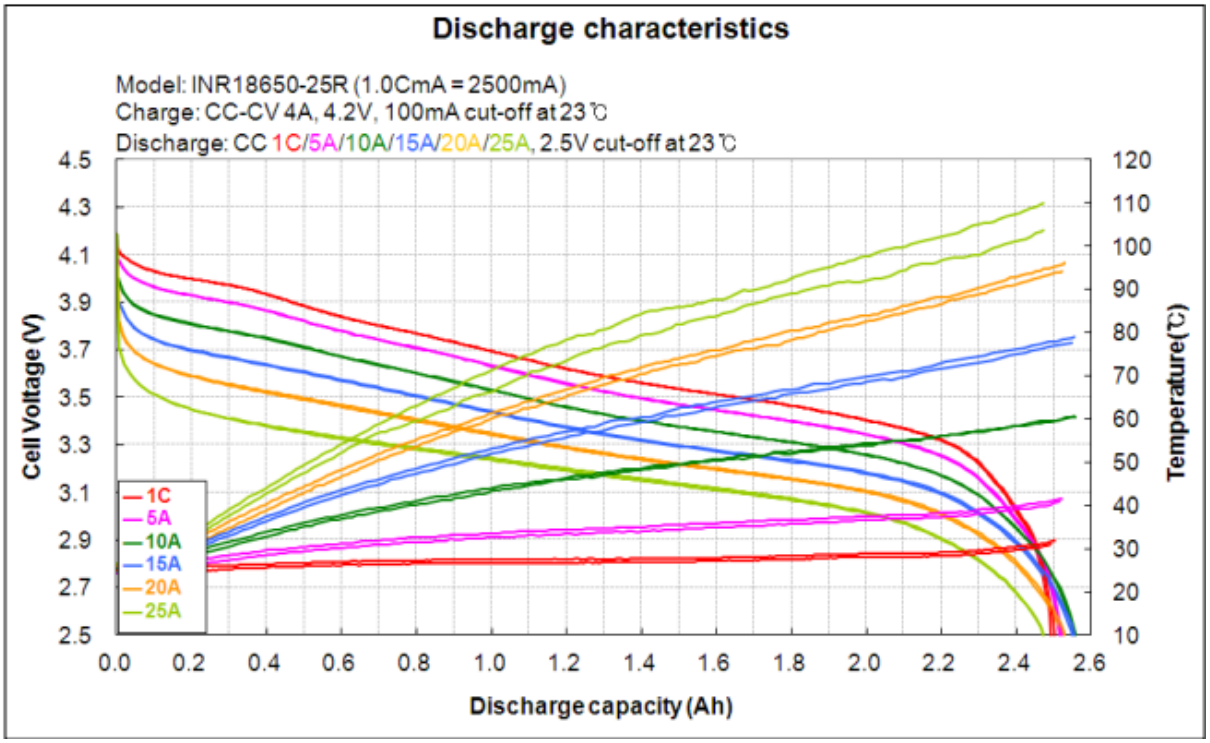
Avaliando *datasheets* de diversos fabricantes de baterias de lítio do modelo 18650, percebe-se ao analisar sua curva de descarga que em tensões acima de 3.6 Volts a queda de tensão se relaciona com a carga consumida de maneira quase que linear. A taxa de queda de tensão é de aproximadamente 0.1 volt por 250mAh de carga consumida.

A Figura 21 apresenta a curva de carga e descarga de uma bateria de íons de lítio, pode-se observar o comportamento linear da descarga para tensões acima de 3,6 Volts.

Para a medição do consumo por queda de tensão na bateria, são necessários os seguintes itens:

- Bateria para alimentar o circuito.
- Multímetro.
- Cabo de alimentação.
- Módulo a ser avaliado.

Figura 21 – Curva de carga e descarga da bateria INR18650-25R da fabricante Samsung. Dada uma temperatura e corrente constante, a tensão cai linearmente na maior parte do descarregamento.



Fonte: Datasheet INR18650-25R

Para realizar essa medição, um *powerbank* construído com uma bateria de íons de lítio modelo 18650 é carregado até que sua bateria atinja uma tensão acima de 3.6 volts. Em seguida este é conectado a um devkit Heltec ESP32 + LoRa com um *firmware* de publicação LoRa configurado para envios de *payloads* de 10 bytes a cada 1 segundo. O dispositivo é deixado em repouso por 30 minutos. Passado o tempo de repouso, é novamente medida a tensão da bateria e esta diferença é registrada. De maneira similar, é desenvolvido um *firmware* para publicação MQTT via WiFi de um *payload* de 10 bytes a cada 1 segundo. A diferença entre a tensão antes e depois do teste é registrada.

Medindo três vezes a queda de tensão na bateria em intervalos de 30 minutos para publicações MQTT via WiFi e Lora, foram obtidos os valores de queda de tensão apresentados na Tabela 3.

Tabela 3 – Queda de tensão após intervalo de 30 minutos

Protocolo	Queda de tensão [mV]	medida 1	medida 2	medida 3	média
MQTT via WiFi		41	39	34	38
LoRa		18	21	25	21

Com os dados obtidos acima e a taxa de queda de tensão apresentado no gráfico da Figura 21 de 2,5Ah/V, pode-se aplicar a equação 4.2 para determinar valores de consumo

durante o período avaliado.

$$Consumo[mAh] = 250[mAh] * 0,1[V^{-1}] * queda\ de\ tensão\ na\ bateria[V] \tag{4.2}$$

$$Corrente[mA] = Consumo[mAh]/intervalo\ de\ consumo[h] \tag{4.3}$$

Assim temos que:

Tabela 4 – Consumo e corrente medias no período avaliado

Protocolo	Consumo médio[mAh]	Corrente média[mA]
MQTT via WiFi	95,00	190,00
LoRa	53,325	106,65

Comparando os resultados aos dados apresentados pelos fabricantes e dispostos nas figuras 18 a 20, pode-se observar que o consumo do microcontrolador está de acordo com o que foi especificado nos *datasheets*. O consumo do devkit durante a transmissão MQTT via WiFi está dentro da faixa de 160mA a 260mA, assim como o consumo durante a transmissão LoRa, que consumiu 106mA.

Os módulos LoRa possuem uma eficiência energética elevada em relação ao uso de WiFi para envio de mensagens. Ainda sim é possível reduzir ainda mais o consumo de dispositivos, tanto LoRa como WiFi, ao se reduzir sua frequência de operação, utilização de modos *sleep* e *low-power*.

5 Conclusão

O presente estudo teve por objetivo validar o uso de dispositivos de baixo custo para soluções de monitoração industrial. Para o presente caso, isso se dá por meio de dispositivos embarcados que obtém dados de máquinas e equipamentos e os torna disponivel para usuários do sistema.

Durante o desenvolvimento deste estudo, o protocolo LoRa se destacou enquanto se buscava um meio de comunicação de longo alcance e robustez por seu baixo consumo energético. Com as qualidades do protocolo LoRa e o uso de um microcontrolador de baixo custo foi possível desenvolver um protótipo de um sistema de monitoração industrial.

Por meio de testes de validação (operação do sistema, alcance e consumo energético), verificou-se que a implementação desse sistema atingiu todos os pontos de maior interesse. A robustez e longo alcance validaram o sistema para o uso desejado de monitoração de equipamentos em ambientes industriais. Alcançando as metas de robustez de sinal, alcance de sinal e de eficiência energética, justificando um maior investimento de tempo e dinheiro a fim de desenvolver um produto comercializável.

O protocolo LoRa, utilizado como meio de comunicação entre os dispositivos do sistema desenvolvido, se mostrou adequado para as necessidades da indústria 4.0 e outras soluções IoT em locais de infraestrutura precária. Seu alcance e baixo consumo permitem a instalações de dispositivos operando totalmente *wireless*, isto é utilizando baterias, com uma grande autonomia.

Referências

[1] Ken Masica - Firmware Management Best Practices Guide for Energy Infrastructure Embedded Control Devices (2020)

[2] John, Tiegelkamp, Iec 61131-3 Programming Industrial Automation Systems, Springer, 2001.

[3] S.S. Iyengar - Scalable Infrastructure for Distributed Sensor Networks-Springer (2005)

[4] Texas Instruments, RS485 Reference Guide, 2014

[5] Modbus Organization, Modbus Application Protocol V1.1b3, 2012

[6] D. Chen, M. Nixon, and A. K.-L. Mok, Wirelesshart: Real-time Mesh Network for Industrial Automation. New York: Springer, 2010.

[7] Mohammad Ilyas, Imad Mahgoub - Handbook of sensor networks compact wireless and wired sensing systems-CRC Press (2004)

[8] Semtech Corporation, LoRa® and LoRaWAN®: A Technical Overview, 2019

[9] Hoang, Q. L., Jung, W. S., Yoon, T., Yoo, D., & Oh, H. (2020). A real-time LoRa protocol for industrial monitoring and control systems. IEEE Access, 8, 44727-44738.

[10] Wildan, F. M. A., Hamidi, E. A. Z., & Juhana, T. (2020, September). The Design of Application for Smart Home Base on LoRa. In 2020 6th International Conference on Wireless and Telematics (ICWT) (pp. 1-6). IEEE.

[11] Eridani, D., Widianto, E. D., & Augustinus, R. D. O. (2019, December). Monitoring system in LoRa network architecture using smart gateway in simple LoRa protocol. In 2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI) (pp. 200-204). IEEE.

[12] Opipah, S., Qodim, H., Miharja, D., Hamidi, E. A. Z., & Juhana, T. (2020, September). Prototype Design of Smart Home System Base on LoRa. In 2020 6th International Conference on Wireless and Telematics (ICWT) (pp. 1-5). IEEE.

[13] LoRa Alliance, LoRaWAN® Specification v1.1, 2017

[14] OASIS, MQTT Version 5.0, 2019.

[15] Sun, C., Zheng, F., Zhou, G., & Guo, K. (2020, July). Design and Implementation of Cloud-based Single-channel LoRa IIoT Gateway Using Raspberry Pi. In 2020 39th Chinese Control Conference (CCC) (pp. 5259-5263). IEEE.

[16] Leonardi, L., Battaglia, F., Patti, G., & Bello, L. L. (2018, October). Industrial LoRa: A novel medium access strategy for LoRa in industry 4.0 applications. In IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society (pp. 4141-4146). IEEE.

[17] Khutsoane, O., Isong, B., Gasela, N., & Abu-Mahfouz, M. (2019). Watergrid-sense: A lora-based sensor node for industrial iot applications. IEEE Sensors Journal, 20(5), 2722-2729.

[18] E. Goldoni, A. Savioli, M. Risi and P. Gamba, "Experimental analysis of RSSI-based indoor localization with IEEE 802.15.4,"2010 European Wireless Conference (EW), 2010, pp. 71-77, doi: 10.1109/EW.2010.5483396.

[19] Soto, V. S., Muller, I., Winter, J. M., Pereira, C. E., & Netto, J. C. (2014, November). Control over wirelesshart network through a host application: A wirelesshart network control proposal. In 2014 Brazilian symposium on computing systems engineering (pp. 91-96). IEEE.

[20] Lentz, J., Hill, S., Schott, B., Bal, M., & Abrishambaf, R. (2018, October). Industrial monitoring and troubleshooting based on LoRa communication technology. In IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society (pp. 3852-3857). IEEE.

[21] Zourmand, A., Hing, A. L. K., Hung, C. W., & AbdulRehman, M. (2019, June). Internet of Things (IoT) using LoRa technology. In 2019 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS) (pp. 324-330). IEEE.

[22] Rawat, A. S., Rajendran, J., Ramiah, H., & Rana, A. (2020, August). LoRa (Long Range) and LoRaWAN Technology for IoT Applications in COVID-19 Pandemic. In 2020 International Conference on Advances in Computing, Communication Materials (ICACCM) (pp. 419-422). IEEE.

[23] Tessaro, L., Raffaldi, C., Rossi, M., & Brunelli, D. (2018, June). LoRa performance in short range industrial applications. In 2018 International Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM) (pp. 1089-1094). IEEE.

[24] Ye, Y. T., & Lei, H. D. (2016, August). Wireless industrial communication system based on Profibus-DP and ZigBee. In 2016 11th International Conference on Computer Science Education (ICCSE) (pp. 666-669). IEEE.

[25] Texas Instruments, The RS-485 Design Guide, 2021

[26] (Wiley Series on Parallel and Distributed Computing) Ivan Stojmenovic - Handbook of Sensor Networks Algorithms and Architectures-Wiley-Interscience (2005)

[27] Richard L. Shell, Ernest L. Hall - Handbook of Industrial Automation-Dekker (2009)

[28] Pradeeka Seneviratne - Beginning LoRa Radio Networks with Arduino Build Long Range, Low Power Wireless IoT Networks-Apress (2019)

Apêndices

APÊNDICE A – GLOSSÁRIO

- ***Array***: Vetor de elementos de dados.
- ***Baudrate***: Taxa com que a informação é transferida em um canal de comunicação.
- ***Bluetooth***: Tecnologia *wireless* para troca de dados baseada na norma IEEE 802.15.1.
- ***Firmware*** Tipo específico de *software* para controle de um *hardware* específico.
- ***Gateway***: Dispositivo tradutor de protocolos.
- **GIT**: *Software* que registra mudanças em um conjunto de arquivos digitais.
- ***Half-duplex***: Esquema de comunicação em que ambas as partes podem se comunicar, mas não simultaneamente.
- **Indústria 4.0**: Expressão que engloba algumas tecnologias de automação e troca de dados para a eficiência dos processos fabris.
- **LoRa**: Long Range, longo alcance em português, é é uma tecnologia de rede de área ampla de baixa potência.
- **LoRaWAN**: Protocolo MAC que age sobre o protocolo físico LoRa.
- ***Microcontrolador***: Pequeno computador num único circuito integrado.
- **Modbus**: Protocolo de comunicação desenvolvido para aplicações industriais.
- ***Payload***: Parte dos dados enviados que contém a mensagem desejada.
- **Sistema Embarcado**: Sistema microprocessado onde o computador é totalmente encapsulado ou dedicado ao dispositivo que controla.
- **WiFi**: Família de protocolos *wireless* baseadas na resolução IEEE 802.11.
- ***Wireless***: Troca de informação entre dois pontos sem uso de um meio condutor.

APÊNDICE B – CÓDIGO DESENVOLVIDO

B.1 Gateway

Aqui se encontram os códigos desenvolvidos para o *gateway*.

B.1.1 Main

```
#include "ArduinoJson.h"      // Documentos Json
#include <WiFiClientSecure.h> // Client WIFI para MQTT
#include <MQTT.h>             // MQTT
#include "ModbusMaster.h"     // Modubs          - https://github.com/4-20ma/ModbusMaster

#include "wifi_lib.h"
#include "time_lib.h"
#include "wifi_lib.h"

#define SerialNumber 123 // nmero de  srie  do equipamento (identificador)
#define AddrNumber 321 // nmero de  srie  do recipiente  ( identificador )
#define BAND 433E6     // e.g. 433E6,868E6,915E6

/*
payload example:
{
  "addr": 123,
  "regBegin": 3,
  "regCount": 3
}
*/

/* variveis  para NTP e RTC*/
int segundosNTP, minutosNTPtemp, horasNTP, diasNTP, mesNTP, anoNTP; // variveis de controle de tempo

/*  configurao  MQTT */
const int mqtt_port = 1883;                                                         // porta do broker
const char mqtt_broker[] = "mqtt.flespi.io";                                        //  endereo  do broker
const char mqtt_user[] = "1B23EN3r75AqW5CBVwKUr651LIX29XK1nCSRdL68EvoqqPR99pUqjNfSl2LlOla1"; //
      usurio do broker
const char mqtt_password[] = "broker";                                              // senha do broker
const char mqtt_topic[] = "gw_lora/data";                                           //  tpico  para
        publicao
const char mqtt_topic_sub[] = "gw_lora/commands";                                   //  tpico  para

MQTTClient mqttClient(1024); //  definio  do client  MQTT
WiFiClient espClient;          // client  WIFI que  receber  MQTT

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// SETUP
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
void setup()
```

```
{
    initLora(); // inicia LORA
    LoRa.onReceive(onReceive); // tratamento de recebimento LORA

    initRTC();

    xTaskCreatePinnedToCore(voidFunc_WIFI, "WIFI", 4096, NULL, 2, NULL, tskNO_AFFINITY); // inicia TASK
        WIFI
    xTaskCreatePinnedToCore(voidFunc_MQTT, "MQTT", 4096, NULL, 2, NULL, tskNO_AFFINITY); // inicia
        TASK MQTT
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// LOOP
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
void loop()
{
    mqttClient.loop(); // mantm MQTT atualizado
    vTaskDelay(100);
}

void voidFunc_WIFI(void *) // task WIFI
{
    begin_WIFI(); // inicia WIFI
    NTPSetup(); // inicia conexo NTP
    while (1)    // loop infinito
    {
        if (WiFi.status() != WL_CONNECTED) // caso se desconecte
        {
            begin_WIFI(); // reconecta ao wifi
        }
        else // caso conectado
        {
            NTPUpdateTime(segundosNTP, minutosNTPtemp, horasNTP, diasNTP, mesNTP, anoNTP); // pega data
            char datac[30];
            sprintf(datac, "%02d:%02d:%02d - %02d/%02d/%04d", horasNTP, minutosNTPtemp, segundosNTP,
                diasNTP, mesNTP, anoNTP); // monta string da data
            String data = datac;
            Serial.println(data);        // print data
            if (mqttClient.connected()) // caso conectado ao broker mqtt
            {
                Serial.println("Publicou");                     // print data
                mqttClient.publish("gw_lora/date", data, true, 1); // publica data
            }
            else // caso desconectado do broker
            {
                Serial.println("Desconectado"); // print data
            }
        }

        vTaskDelay(30000 / portTICK_PERIOD_MS); // aguarda 30 segundos
    }
    vTaskDelete(NULL);
}

void voidFunc_MQTT(void *)
{
    while (WiFi.status() != WL_CONNECTED) // caso no esteja conectado ao WIFI
    {
        vTaskDelay(100); // aguarda
```

```
    }
    vTaskDelay(3000 / portTICK_PERIOD_MS); // aguarda 3 segundos
    Serial.println("Comeou MQTT");
    mqttClient.begin(mqtt_broker, mqtt_port, espClient);                // Configura broker
    while (!mqttClient.connect("teste_esp32", mqtt_user, mqtt_password)) // tenta conectar
    {
        Serial.print(".");
        delay(1000);
    }
    Serial.println("Conectou MQTT");
    mqttClient.subscribe(mqtt_topic_sub); // se inscreve no  tpico  para receber comandos
    mqttClient.onMessage(messageReceived); //  funo  de tratamento de mensagens recebidas
    while (1)                              // loop infinito
    {
        if (WiFi.status() == WL_CONNECTED) // verifica conexo com wifi
        {
            if (mqttClient.connected()) // se  est  conectado ao MQTT
            {
            }
            else // caso esteja desconectado
            {
                mqttconnect(); // tenta reconectar
            }
        }
        vTaskDelay(30000 / portTICK_PERIOD_MS); // aguarda 30 segundos
    }
    vTaskDelete(NULL);
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void onReceive(int packetSize) // tratamento de recebimento LoRa
{
    const char *packetBuffer = ""; // buffer da mensagem recebida
    String packet = "";            // string da mensagem recebida
    while (LoRa.available())       // enquanto  h  dados sendo recebidos
    {
        packet = LoRa.readString(); // passa dado recebido para String
    }
    // Serial.print("Received with RSSI ");
    // Serial.println(LoRa.packetRssi());
    packetBuffer = packet.c_str(); // converte para const char
    // cria task para envio de dado LoRa via MQTT
    xTaskCreate(
        sendLoraData,         /* Function that implements the task. */
        "sendLoraData",       /* Text name for the task. */
        4096,                 /* Stack size in words, not bytes. */
        (void *)packetBuffer, /* Parameter passed into the task. */
        2,                    /* Priority at which the task is created. */
        NULL);                /* Used to pass out the created task's handle. */
}

void mqttconnect() // conecta ao broker mqtt
{
    if (!mqttClient.connected()) // caso esteja desconectado
    {
        Serial.println("Tentando Reconectar");                    // print data
        mqttClient.connect("12414324sd", mqtt_user, mqtt_password); // tenta conectar
        if (mqttClient.connected())
```

```
        {
            mqttClient.subscribe(mqtt_topic_sub); // em caso de sucesso, se inscreve no  tpico
        }
    }
}

void messageReceived(String &topic, String &payload) // trata mensagem recebida MQTT
{
    if (topic == mqtt_topic_sub) // verifica se  do  tpico  de interesse
    {
        DynamicJsonDocument recebido_mqtt(128); // cria Json para tratamento de dado
        deserializeJson(recebido_mqtt, payload); // abre dado recebido como Json

        // int id = recebido_mqtt["id"];
        int addr = recebido_mqtt["addr"]; // pega endereo
        int func = recebido_mqtt["func"]; // pega cdigo da  funo
        // String time = recebido_mqtt["time"];

        if (addr == SerialNumber) // caso a mensagem seja endereada a ele
        {
            Serial.print("Recebido via MQTT: ");
            Serial.println(payload);
            switch (func) // verifica qual  funo  foi chamada
            {
            case 0: //  funo  de leitura modbus
            {
                String messagetosend = "";              // mensagem que ser enviada
                DynamicJsonDocument enviar_lora(128); // cria documento Json

                enviar_lora["addr"] = AddrNumber;               // endereo da mensagem
                enviar_lora["id"] = SerialNumber;               // identificador do remetente
                enviar_lora["regB"] = recebido_mqtt["regBegin"]; // endereo inicial
                enviar_lora["regC"] = recebido_mqtt["regCount"]; // quantidade de endereos a serem lidos

                serializeJson(enviar_lora, messagetosend); // passa json para buffer
                Serial.print("messagetosend: ");
                Serial.println(messagetosend);

                LoRa.beginPacket();                                 // inicia transmisso
                LoRa.setTxPower(5, RF_PACONFIG_PASELECT_PABOOST); // seta potncia da transmisso
                LoRa.print(messagetosend);                          // escreve pacote
                LoRa.endPacket();                                   // finaliza pacote
                LoRa.receive();                                     // retorna para modo de recepo
                break;
            }
            case 1:
            {
            }
            case 2:
            {
            }
            case 3:
            {
            }

            default:
                break;
            }
        }
    }
```

```
}

void sendLoraData(void *pvParameters) // funo para publicao de dados LoRa recebidos no Broker
{
    char *str_receivedMsg;
    str_receivedMsg = (char *)pvParameters; // mensagem a ser enviada
    Serial.print("Recebido LoRa: ");
    Serial.println(str_receivedMsg);

    DynamicJsonDocument doc(1000);      // cria documento Json
    deserializeJson(doc, str_receivedMsg); // abre json
    char buffer[256];                   // buffer da mensagem
    int addr = doc["addr"];             // pega endereo
    serializeJson(doc, buffer);         // passa json para buffer
    Serial.print("buffer: ");
    Serial.println(buffer);
    if (addr == SerialNumber) // caso seja endereado a si
    {
        while (!mqttClient.publish(mqtt_topic, buffer, true, 1)) // tenta publicar dado
        {
            vTaskDelay(1000 / portTICK_PERIOD_MS);
        }
        Serial.println("Publicou LORA"); // print data
    }
    vTaskDelete(NULL);
}
```

B.1.2 LoRa

```
#include <Arduino.h>
#include "heltec.h"             // Funcionalidades da placa heltec (lora, tela OLED, etc)

void initLora()
{
    Heltec.begin(true /*DisplayEnable Enable*/, true /*Heltec.Heltec.Heltec.LoRa Disable*/, true /*Serial Enable*/,
        true /*PABOOST Enable*/, BAND /*long BAND*/);
    LoRa.receive();             // incia modo recepo
}
```

B.1.3 WiFi

```
#include <Arduino.h>
#include <WiFi.h>               // WIFI


/* variveis para WIFI*/
const char *ssid = "VIVOFIBRA-8338_EXT"; // SSID (nome da rede)
const char *password = "77A98836AD";  // senha


void begin_WIFI() // inicia conexo wifi
{
    if (WiFi.status() != WL_CONNECTED) // caso esteja desconectado do WIFI
    {
        if (ssid != NULL && password != NULL) // caso SSID E PW no sejam nulos
```

```
        {
            WiFi.begin(ssid, password);
        }
        Serial.println("=====================================");
        Serial.println("=        Connecting to the WiFi       =");
        Serial.println("=====================================");
        Serial.println("Auth Type:         WIFI_AUTH_WPA2_PSK");
        Serial.print("Network SSID:     ");
        Serial.println(ssid);
        Serial.print("Network PASSWORD: ");
        Serial.print("Wait ");
        for (int i = 0; i < 100; i++) // por 10 segundos tenta conectar
        {
            if (WiFi.status() == WL_CONNECTED) // caso conecte
            {
                Serial.println("CONECTADO");
                break; // sai do loop
            }
            vTaskDelay(100 / portTICK_PERIOD_MS); // aguarda 100 ms
            Serial.print(".");
        }
    }
}
```

B.1.4 RTC e NTP

```
#include <Arduino.h>

#include "time.h"   //   funes   de tempo
#include <Wire.h> // I2c
#include <RTClib.h> // relgio de Tempo Real - DS3231

RTC_DS3231 RTC;
DateTime now;

void initRTC()
{
    RTC.begin();                                                                    // inicia RTC
    now = RTC.now();                                                                // pega data do rtc
    setTime(now.hour(), now.minute(), now.second(), now.day(), now.month(), now.year()); // passa data do RTC para
        o ESP
}

void NTPSetup() //  configurao  NTP
{
    const long gmtOffset_sec = -3 * 3600;                    // fuso horrio brasileiro (-3h)
    const int daylightOffset_sec = 0;                        //  horrio de vero
    const char *ntpServer = "pool.ntp.org";                  // servidor para requisio de tempo
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer); // seta servidor, fuso horrio e horrio de vero
}

bool NTPUpdateTime(int &segundos, int &minutos, int &horas, int &dias, int &mes, int &ano) // atualizao do
    horrio
{
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo)) // caso no consiga pegar hora
    {
```

```
        return false; // retorna falha
    }
    else
    {
        // atualiza   variveis
        segundos = timeinfo.tm_sec;
        minutos = timeinfo.tm_min;
        horas = timeinfo.tm_hour;
        dias = timeinfo.tm_mday;
        mes = timeinfo.tm_mon + 1;
        ano = timeinfo.tm_year + 1900;
        // atualiza RTC com hora
        RTC.adjust(timeinfo.tm_mday, timeinfo.tm_wday, timeinfo.tm_mon + 1, timeinfo.tm_year - 100,
             timeinfo.tm_hour, timeinfo.tm_min, timeinfo.tm_sec);
    }
}
```

B.2 Unidade de Monitoramento

Aqui se encontram os códigos desenvolvidos para a unidade de monitoramento.

B.2.1 Main

```
#include "heltec.h"
#include "ArduinoJson.h"
#include <WiFi.h>
#include "time.h"

#include "modbus_lib.h"

#define SerialNumber 321     // nmero de  srie   prprio
#define SerialNumberaddr 123 // endereo do  destinatrio  das mensagens
#define BAND 433E6           // 433Mhz ,e.g. 868E6,915E6

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// SETUP
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
void setup()
{
    Heltec.begin(true /*DisplayEnable Enable*/, true /*Heltec.Heltec.Heltec.LoRa Disable*/, true /*Serial Enable*/,
         true /*PABOOST Enable*/, BAND /*long BAND*/);

    LoRa.onReceive(onReceive); // indica  funo  de tratamento de recebimentos LORA
    LoRa.receive(); // modo  recepo

    initModbus(); // inicia  comunicao  modbus
}

//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
// LOOP
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
void loop()
{
    vTaskDelay(100);
}
```

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

void onReceive(int packetSize)
{
    const char *teste = ""; // buffer const char
    String packet = ""; // buffer string
    while (LoRa.available())
    {
        packet = LoRa.readString(); // passa mensagem recebida para buffer
    }
    // Serial.print("Received with RSSI ");
    // Serial.println(LoRa.packetRssi());

    teste = packet.c_str(); // converte buffer para const char
    xTaskCreate( // cria task que ir  ler os dados do dispositivo
        getModbusData, /* Function that implements the task. */
        "getModbusData", /* Text name for the task. */
        4096,            /* Stack size in words, not bytes. */
        (void *)teste,   /* Parameter passed into the task. */
        4,               /* Priority at which the task is created. */
        NULL);           /* Used to pass out the created task's handle. */
}

void getModbusData(void *pvParameters)
{
    char *str_receivedMsg;
    str_receivedMsg = (char *)pvParameters;
    Serial.print("Recebido: ");
    Serial.println(str_receivedMsg);

    DynamicJsonDocument doc(128); // cria documento Json
    deserializeJson(doc, str_receivedMsg); // abre comando recebido via LORA

    int addr = doc["addr"]; // pega endereo da mensagem
    if (addr == SerialNumber) // se  endereado a si
    {
        // pega dados do payload Lora
        int id = doc["id"];
        // String time = doc["time"];
        uint16_t dvAddr = doc["mAddr"]; // endereo do device
        uint16_t regBegin = doc["regB"]; // primeiro registro
        uint16_t regCount = doc["regC"]; // quantidade de registros

        uint8_t result_modbus = modbus.readHoldingRegisters(regBegin, regCount); // l dados do dispositivo
        char buffer[256]; // buffer para mensagem lora a ser enviada
        DynamicJsonDocument response(1000);
        if (result_modbus == modbus.ku8MBSuccess) // em caso de sucesso
        {
            response["addr"] = SerialNumberaddr; // destinatrio da mensagem
            response["id"] = SerialNumber; // id prprio
            JsonArray response_array = response.createNestedArray("arr"); // cria array a ser populado com dados
                recebidos
            for (int i = 0; i < regCount; i++)
            {
                response_array.add(modbus.getResponseBuffer(i)); // passa dados obtidos do dispositivo para array
            }
            serializeJson(response, buffer); // passa json patra buffer
        }
```

```
        else // caso de falha, monta mensagem de falha
        {
            Serial.println("FAIL TO READ");
            response["addr"] = SerialNumberaddr;
            response["id"] = SerialNumber;
            response["err"] = "FAIL TO READ";
            serializeJson(response, buffer); // passa para buffer
        }
        Serial.print("messagetosend_lora: ");
        Serial.println(buffer);

        LoRa.beginPacket(); // inicia transmisso
        LoRa.setTxPower(5, RF_PACONFIG_PASELECT_PABOOST); // seta potncia do envio
        LoRa.print(buffer); // envia payload
        LoRa.endPacket(); // finaliza mensagem
        LoRa.receive(); //volta para modo recepo
    }
    vTaskDelete(NULL);
}
```

B.2.2 ModBus

```
#include <Arduino.h>
#include "ModbusMaster.h" // Modubs — https://github.com/4-20ma/ModbusMaster

/* configurao RS485 */
#define RXcomm LOW // definio do nvel do pino de controle de fluxo para recepo
#define TXcomm HIGH // definio do nvel do pino de controle de fluxo para transmisso
#define RS232_485_RXPIN 25 // pino RX
#define RS232_485_TXPIN 13 // pino TX
#define RS485_TXRX_CONTROLPIN 17 // pino de controle de fluxo para o chip 485

ModbusMaster modbus; // instncia modbus

void preTransmission()
{
    digitalWrite(RS485_TXRX_CONTROLPIN, TXcomm); // Muda no para transmisso
}

void postTransmission()
{
    digitalWrite(RS485_TXRX_CONTROLPIN, RXcomm); // Muda pino para recebimento
}

void initModbus()
{
    pinMode(RS485_TXRX_CONTROLPIN, OUTPUT); // seta pino como output
    digitalWrite(RS485_TXRX_CONTROLPIN, RXcomm); // seta pino para recebimento de dado
    Serial2.begin(115200, 0x800001c, RS232_485_RXPIN, RS232_485_TXPIN); // inicia comunicao serial
    modbus.begin(1, Serial2); // cria instncia modbus na porta serial 2
    modbus.preTransmission(preTransmission); // callback pr transmisso
    modbus.postTransmission(postTransmission); // callback ps transmisso
}
```