MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
Instituto de Ciências Exatas e Aplicadas
Graduação em Engenharia Elétrica

UNIVERSIDADE FEDERAL DE OURO PRETO

INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Implementação de uma rede híbrida LoRaWAN/LoRaMESH no Instituto de Ciências Exatas e Aplicadas da UFOP

RICARDO MATIAS SILVINO

TRABALHO DE CONCLUSÃO DE CURSO
JOÃO MONLEVADE 2024

RICARDO MATIAS SILVINO

IMPLEMENTAÇÃO DE UMA REDE LORAWAN/LORAMESH NO INSTITUTO DE CIÊNCIAS EXATAS E APLICADAS DA UFOP

Trabalho de Conclusão de Curso apresentado como requisito para obter o título de Bacharel em Engenharia Elétrica pela Universidade Federal de Ouro preto.

Orientadora: Profª. Drª. Aline Rocha de Assis

JOÃO MONLEVADE

2024

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S587i Silvino, Ricardo Matias.
Implementação de uma rede híbrida LoRaWAN/LoRaMESH no Instituto de Ciências Exatas e Aplicadas da UFOP. [manuscrito] / Ricardo Matias Silvino. - 2024.
79 f.: il.: color., tab., mapa.

Orientadora: Profa. Dra. Aline Rocha de Assis.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia Elétrica .

1. Comunicação e tecnologia. 2. Internet das coisas. 3. Redes de computadores. 4. Sistemas de comunicação sem fio. 5. Telecomunicações. 6. Topologias de redes. I. Assis, Aline Rocha de. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.7

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE ENGENHARIA ELETRICA

FOLHA DE APROVAÇÃO

Ricardo Matias Silvino

Implementação de uma rede híbrida LoRaWAN/LoRaMESH no Instituto de Ciências Exatas e Aplicadas da UFOP

Monografia apresentada ao Curso de Engenharia Elétrica da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Engenheiro Eletricista

Aprovada em 17 de Outubro de 2024

Membros da banca

Drª Aline Rocha de Assis - Orientadora - Universidade Federal de Uberlândia
Dr. Marcelo Moreira Tiago - Universidade Federal de Ouro Preto
Dr. Theo Silva Lins - Universidade Federal de Ouro Preto

Aline Rocha de Assis, orientadora do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 07/11/2024

Documento assinado eletronicamente por **Renan Fernandes Bastos**, **COORDENADOR(A) DO CURSO DE ENGENHARIA ELÉTRICA**, em 07/11/2024, às 17:01, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0807635** e o código CRC **33D3DA18**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0818 - www.ufop.br

Dedico este trabalho, primeiramente, à minha família, pelo apoio incondicional e por sempre acreditarem em mim, nos momentos de dificuldades e nos momentos de conquista.

Também dedico este projeto a todos os amigos, professores e colegas que, de alguma forma, contribuíram com sugestões, críticas e ideias, ajudando a enriquecer o conteúdo e a tornar este trabalho possível.

AGRADECIMENTOS

A conclusão deste trabalho só foi possível graças ao apoio e à colaboração de muitas pessoas que contribuíram, direta ou indiretamente, para cada etapa desse processo. Assim, gostaria de expressar minha sincera gratidão.

Aos meus familiares, pelo amor, compreensão e apoio incondicional, que foram fundamentais ao longo de toda a minha trajetória. Obrigado por estarem ao meu lado em todos os momentos, especialmente nos desafios, e por serem minha fonte constante de motivação.

Aos professores e a orientadora, por compartilharem seu conhecimento, por cada ensinamento e orientação que me ajudaram a construir este projeto. Suas orientações foram cruciais para o desenvolvimento do trabalho e para meu crescimento profissional e pessoal.

Aos colegas e amigos, pela ajuda, conselhos e incentivo durante a elaboração deste trabalho. Cada colaboração, cada palavra de incentivo e cada troca de ideias enriqueceram este projeto e fizeram a diferença no meu caminho.

A todos que, de alguma forma, contribuíram para esta realização, meu mais profundo agradecimento. Sem o apoio e a participação de cada um de vocês, este trabalho não teria sido possível.

A todos vocês, meu sincero agradecimento e eterna gratidão, sem cada um de vocês, este trabalho não seria o que é. Muito obrigado!

“Não existem métodos fáceis para resolver problemas difíceis.”
(RENÉ DESCARTES)

RESUMO

A rede LoRa (*Long Range*) é uma tecnologia de comunicação sem fio, não licenciada, de baixa potência e longo alcance projetada para suportar aplicações de Internet das Coisas (IoT). Essa rede oferece uma solução eficiente e econômica para conectar dispositivos inteligentes, sensores e atuadores em diversas áreas, como agricultura, monitoramento ambiental, cidades inteligentes e indústria. Seu protocolo base é o LoRaWAN, no qual se baseia na topologia estrela que consiste na comunicação de um dispositivo final com o *gateway* sem nenhum intermediário. Entretanto, esta topologia apresenta uma limitação significativa em relação a distância de comunicação, já que se trata de apenas um dispositivo se comunicando com outro o que fica restrito à capacidade dos equipamentos envolvidos e às interferências externas. Este trabalho visa ampliar a comunicação da rede LoRaWAN através da aplicação de uma topologia *mesh* agregada a ela, formando assim uma rede LoRa híbrida. Desta forma um dispositivo final poderá receber informação de um outro e repassar as mensagens entre nós intermediários até atingir o ponto alvo. A metodologia utilizada compreende a implementação de uma ponte entre as redes LoRaWAN e LoRaMESH, utilizando módulos da Radioenge® e dispositivos como o ESP 8266, afim de testar a transmissão em diferentes condições de distância. Os resultados obtidos evidenciaram que a adição da topologia *mesh* aumentou o alcance da rede em duas vezes nos casos testados, permitindo uma cobertura ampliada em áreas inacessíveis pela topologia estrela. A aplicação de uma topologia híbrida demonstrou-se eficaz para ampliar o alcance de transmissão, os resultados evidenciam que áreas mais distantes do *gateway* que antes não recebiam cobertura do sinal, após os testes passaram a receber, o que mantem a qualidade da comunicação atendendo de forma satisfatória o objetivo proposto.

Palavras-chave: LoRa, topologia *mesh*, topologia estrela, LoRaWAN

ABSTRACT

The LoRa (Long Range) Network is an unlicensed, low-power, long-range wireless communication technology designed to support Internet of Things (IoT) applications. This network offers an efficient and cost-effective solution to connect smart devices, sensors and actuators in diverse areas such as agriculture, environmental monitoring, smart cities and industry. Its base protocol is LoRaWAN, which is based on the star topology that consists of the communication of an end device with the gateway without any intermediary. However, this topology presents a significant limitation in relation to the communication distance, since there is only one device communicating with another, which is restricted to the capacity of the equipment involved and to external interference. This work aims to expand the communication of the LoRaWAN network through the application of a mesh topology added to it, thus forming a hybrid LoRa network. In this way, an end device can receive information from another and pass the messages between intermediate nodes until reaching the target point. The methodology used comprises the implementation of a bridge between the LoRaWAN and LoRaMESH networks, using Radioenge® modules and devices such as the ESP 8266, in order to test transmission under different distance conditions. The results obtained showed that the addition of the mesh topology increased the network's reach by two times in the cases tested, allowing expanded coverage in areas inaccessible by the star topology. The application of a hybrid topology proved to be effective in expanding the transmission range. The results show that areas further away from the gateway that previously did not receive signal coverage, after the tests began to receive it, which maintains the quality of communication, providing the proposed objective satisfactorily.

Key words: LoRa, mesh topology, star topology, LoRaWAN

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Com o crescente avanço da Internet das Coisas (IoT) e a demanda por soluções de conectividade de longo alcance e baixo consumo de energia, a tecnologia LoRa (*Long Range*) tem emergido como uma das principais alternativas para atender a essas necessidades. A conectividade robusta, a ampla cobertura geográfica e a eficiência energética proporcionadas por este sistema de comunicação a tornam uma escolha promissora para uma variedade de aplicações, desde monitoramento ambiental até cidades inteligentes e agricultura de precisão.

No contexto atual, onde dispositivos inteligentes estão sendo implantados em diversos setores para coletar e transmitir dados relevantes, a capacidade de se comunicar de maneira confiável, mesmo em áreas remotas, torna-se uma necessidade crítica. No entanto, muitas tecnologias de comunicação existentes apresentam limitações em termos de alcance e consumo de energia, o que pode ser um obstáculo para o desenvolvimento de soluções eficazes e escaláveis.

A tecnologia LoRa, baseada em modulação de espectro espalhado, oferece uma abordagem para superar esses desafios. Ao explorar a largura de banda não licenciada disponível, permite comunicações de longo alcance e baixa potência, viabilizando aplicações que anteriormente eram inviáveis devido às restrições tecnológicas. Essa capacidade de estender a conectividade a áreas distantes e de difícil acesso abre novas possibilidades para a aquisição de dados em tempo real e a tomada de decisões (LoRa® Alliance Technical Marketing Workgroup, 2015, p. 4).

Nos últimos anos, houve avanços significativos em termos de modulação, alcance e eficiência de energia na tecnologia LoRa. Novas técnicas de modulação foram introduzidas para melhorar ainda mais a capacidade de transmissão de dados em condições adversas. Além disso, o desenvolvimento de circuitos integrados e módulos mais eficientes permitiram a criação de dispositivos ainda mais compactos e de baixo consumo de energia (Semtech, 2019, p. 1).

Em relação aos padrões de comunicação, a LoRa Alliance, uma organização de colaboração entre empresas líderes no setor, tem desempenhado um papel fundamental no estabelecimento de padrões e especificações abertas para garantir a interoperabilidade entre dispositivos LoRa de diferentes fabricantes. Isso tem

impulsionado o crescimento do ecossistema e facilitado a adoção em larga escala (LoRa® Alliance Technical Marketing Workgroup, 2015, p. 4).

As aplicações da rede têm se expandido para diversos setores. Na agricultura, por exemplo, a tecnologia LoRa é usada para monitorar condições de solo, clima e rebanhos, permitindo a implementação de práticas de agricultura de precisão. Em cidades inteligentes, a rede LoRa é empregada para gerenciar a iluminação pública, coletar dados de sensores ambientais e monitorar o tráfego. Além disso, a saúde também está se beneficiando da conectividade, com dispositivos de monitoramento remoto e rastreamento de ativos em ambientes hospitalares (Salgado, 2019, p. 15).

Apesar dos avanços, a tecnologia também enfrenta desafios. A gestão da interferência espectral em áreas urbanas densas e a otimização da eficiência de energia para prolongar a vida útil da bateria de dispositivos são considerações críticas. Além disso, a segurança e a privacidade dos dados transmitidos por meio da rede LoRa continuam sendo áreas de foco para pesquisa e desenvolvimento.

O uso da topologia estrela adotada no protocolo LoRaWAN *(LoRa Wide Area Network)* acaba limitando a performance da transmissão dos dispositivos, uma vez que se tem apenas a comunicação entre dois dispositivos, de modo que, a presença de um obstáculo qualquer, ou até mesmo a distância entre eles acaba limitando a criação de algumas aplicações.

Nesse contexto, este trabalho se propõe a explorar a tecnologia LoRa de forma abrangente, investigando seus princípios de funcionamento, suas aplicações em diferentes cenários e as vantagens que oferece em relação a outras tecnologias de comunicação. Além disso, será proposta uma fusão entre as topologias *mesh* e estrela, a fim de se criar pontos de repetição do sinal, permitindo um maior alcance da rede LoRaWAN.

Em resumo, a tecnologia LoRa representa uma abordagem promissora para atender à crescente demanda por conectividade de longo alcance e baixo consumo de energia na era da IoT. Sua capacidade de superar as limitações das tecnologias de comunicação convencionais a torna uma escolha valiosa para uma ampla gama de aplicações, com potencial para transformar a maneira como interagimos com o mundo ao nosso redor.

1.1 MOTIVAÇÃO E JUSTIFICATIVAS

A cobertura da rede LoRaWAN é fortemente influenciada pela topologia do terreno e pela disposição dos dispositivos, principalmente em ambientes urbanos onde a cobertura sofre uma assimetria ainda maior (Ballart, 2018). Diferentes ambientes geográficos apresentam desafios únicos que afetam a propagação do sinal, como obstáculos físicos, relevo e interferências eletromagnéticas. Nesse contexto, soluções como instalação de novos *gateways* seria inviável tendo em vista seu consumo energético e alto custo de aquisição. A principal motivação deste trabalho é preencher as deficiências de cobertura da rede LoRaWAN com um custo reduzido.

Para isso se faz necessário utilizar módulos trabalhando em topologia *mesh*, com intuito de servir como repetidor de sinal para os *end devices* da topologia LoRaWAN. Comparado com um *gateway*, um *end device* LoRaMESH tem um custo bem menor tanto na aquisição, quanto na questão de consumo energético, sem contar na questão da instalação que acaba sendo bem mais simplificada.

Algumas propostas de solução para o problema de cobertura da rede LoRaWAN são apresentadas em (Maziero, 2020, p. 40), onde o autor apresenta duas abordagens que visam minimizar o problema. A primeira delas propõe o uso de topologia *mesh* onde a rota de comunicação se baseia na intensidade do sinal recebido (RSSI) para encontrar o caminho ideal até o dispositivo concentrador (Kirichek, Vishnevsky, Pham, & Koucheryavy, 2020). A segunda é baseada na adoção de um nó aprimorado, que atuaria como um extensor de alcance, sua estrutura consiste em receber uma rede diferente em sua porta de entrada, desta forma esse nó atuaria como uma espécie de *gateway* para redes herdadas e como um repetidor (Sisinni, et al., 2018, p. 2).

1.2 OBJETIVOS

O trabalho tem como objetivo geral o desenvolvimento de uma rede LoRa híbrida para aumentar o alcance de cobertura, proporcionada pelo protocolo LoRaWAN.

Compõe os objetivos específicos deste trabalho:

- Implementação de uma rede LoRaWAN experimental;
- Implementação de uma rede LoRaMESH experimental;
- Implementação de uma rede LoRa híbrida LoRaWAN/LoRaMESH;
- Realizar testes de cobertura do sinal no entorno do *gateway* instalado na cidade de João Monlevade;
- Comparar a cobertura obtida com a rede LoRaWAN e a cobertura obtida a partir da rede LoRa híbrida.

1.3 ORGANIZAÇÃO DO TRABALHO

O presente trabalho está estruturado da seguinte forma: no capítulo 2 é apresentada uma revisão bibliográfica detalhada sobre os conceitos-chave relacionados à tecnologia LoRa, suas aplicações e especificações, também será feita uma breve abordagem sobre Internet das Coisas. No capítulo 3 são discutidos os aspectos técnicos na implementação da rede LoRaWAN, e no capítulo 4 são discutidos os aspectos técnicos da implementação da rede LoRaMESH, seguidos por um desenvolvimento de protótipos usando LoRaWAN e LoRaMESH, são apresentados os resultados preliminares, destacando os dados obtidos durantes os testes dos protótipos implementados. O capítulo 5 apresenta a implementação da rede LoRa híbrida. Por fim, são apresentadas as conclusões obtidas durante o trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTERNET DAS COISAS E SUAS APLICAÇÕES

A Internet das Coisas (IoT, do inglês *Internet of things*) tem sido amplamente reconhecida como uma das tecnologias mais disruptivas e promissoras do século XXI. Ela se refere à interconexão de objetos físicos, como dispositivos eletrônicos, sensores, veículos e eletrodomésticos, à internet, permitindo a coleta, troca e análise de dados de maneira inteligente e eficiente. A IoT transcende as fronteiras tradicionais dos sistemas de computadores e expande a conectividade para além dos dispositivos convencionais, estendendo-se a itens cotidianos que compõem o ambiente ao nosso redor.

A proliferação da IoT se deve ao crescente interesse em melhorar a eficiência, a automação e a qualidade de vida, tanto no âmbito doméstico quanto em diversos setores industriais. Na agricultura, por exemplo, a IoT possibilita a criação de fazendas inteligentes, onde sensores coletam informações sobre solo, umidade, temperatura e outras variáveis para otimizar o uso de recursos e maximizar a produtividade.

2.1.1 Tecnologias de conectividade para a IOT: uma visão geral

Para viabilizar a Internet das Coisas, diversas tecnologias de conectividade foram desenvolvidas, cada uma com suas próprias características e adequações a diferentes cenários de aplicação. Dentre elas, destacam-se a rede LoRa (*Long Range*), a Sigfox e a NB-IoT (*Narrowband* IoT).

A comparação entre a tecnologia LoRa com outras tecnologias de conectividade, como a NB-IoT, que utiliza infraestrutura das operadoras de telefonia móvel, ou a Sigfox, que opera em frequências livres, a principal vantagem é o baixo custo de implementação e operação. Na Tabela 1 pode-se notar algumas características das tecnologias de conexão.

Tabela 1 - Comparativo entre as tecnologias de transmissão usadas para IOT.

Característica	**LoRaWAN**	**NB-IoT**	**SigFox**
Modulação	LoRa CSS	OFDMA	UNB
Largura de banda *downlink*	125kHz/500kHz	180kHz	100Hz
Largura de banda *uplink*	125kHz	180kHz	100Hz
Largura de banda total	8MHz	180kHz	200kHz
Taxa de transmissão	0,625kbps-100kbps	20kbps	100bps
Limite de mensagem	Não	Não	140msg/dia
Máximo *Payload* (bytes)	<1000	>10.000	12
Método de acesso ao canal	ALOHA	TDMA/FDMA	ALOHA
***Firmware* OTA**	Não	Sim	Não
Frequência	ISM	Licenciadas	ISM
Modelo de negócio	Rede privada ou pública	Rede pública	Rede pública
Principais vantagens	Custo baixo, alta imunidade a ruídos, rápida implementação	Cobertura	Cobertura e consumo
Principais desvantagens	Suporte a sincronismo	Custo	Custo

Fonte: Adaptado de Carloto, 2020

A tecnologia LoRa apresenta vantagens claras sobre NB-IoT e SigFox em várias características essenciais para aplicações de IoT. Na modulação, utiliza o método CSS (*Chirp Spread Spectrum*), que garante excelente imunidade a interferências, possibilitando um alcance de transmissão maior em ambientes

ruidosos e com obstáculos físicos, superando SigFox e NB-IoT em termos de estabilidade da comunicação. Em relação à largura de banda, oferece uma faixa mais ampla no *downlink* (125 kHz à 500 kHz) e *uplink* (125 kHz), permitindo a transmissão de mais dados por canal, superando as demais, que operam com uma largura muito limitada. Isso proporciona maior flexibilidade e capacidade de transmissão para diferentes aplicações. No que diz respeito à largura de banda total, com 8 MHz, suporta um maior número de dispositivos e dados simultâneos, superando as faixas mais restritas de SigFox (200 kHz) e NB-IoT (180 kHz). Além disso, a taxa de transmissão, que varia de 0,625 kbps até 100 kbps, oferece maior flexibilidade para ajustar a velocidade conforme as necessidades da aplicação.

A tecnologia LoRa não impõe limites de mensagens diários, ao contrário de SigFox, que restringe a comunicação a 140 mensagens por dia, o que permite seu uso em aplicações que exigem envio frequente de dados sem restrições. No quesito máximo *payload*, a tecnologia pode transmitir mensagens com até 1000 bytes, sendo muito superior aos 12 bytes permitidos pela SigFox, permitindo o uso em aplicações que requerem o envio de dados mais complexos ou em maior volume.

No método de acesso ao canal, tanto LoRa quanto SigFox utilizam o protocolo ALOHA, mas LoRa se destaca pelo custo operacional mais baixo e maior imunidade a interferências, garantindo uma comunicação mais eficiente. Outro ponto forte é a possibilidade de operar em redes privadas e públicas, o que oferece mais flexibilidade nos modelos de negócios, permitindo que empresas implementem suas próprias redes, ao contrário de NB-IoT e SigFox, que dependem de redes públicas, elevando os custos.

Em síntese, as principais vantagens da rede LoRa são seu custo baixo, a alta imunidade a ruídos e a rápida implementação, o que a torna uma escolha atrativa para quem busca uma solução acessível e eficiente, adequada a uma ampla gama de aplicações IoT.

2.2 SURGIMENTO DA REDE LORA: CONTEXTO HISTÓRICO

O surgimento da rede LoRa remonta ao início dos anos 2000, quando uma equipe de pesquisadores da empresa Cycleo, com sede em Grenoble, França,

começou a explorar novas tecnologias de comunicação sem fio, com foco em aplicações de Internet das Coisas. Essa equipe, liderada por Michel Fallah, tinha como objetivo desenvolver uma tecnologia de comunicação eficiente e de longo alcance que pudesse atender às crescentes demandas de conectividade para dispositivos de baixa potência e baixo custo.

Em 2009, a Cycleo lançou o sistema de modulação *Chirp Spread Spectrum* (CSS), que se tornou a base da tecnologia LoRa. A modulação CSS apresentava a capacidade de transmitir dados em longas distâncias, superando obstáculos físicos e interferências, ao mesmo tempo em que mantinha um consumo extremamente baixo de energia. Essa combinação única de longo alcance e baixo consumo tornou o CSS altamente adequado para aplicações IoT, que frequentemente envolvem dispositivos alimentados por baterias com vida útil prolongada.

Em 2012, a Cycleo chamou a atenção da empresa norte-americana Semtech Corporation, especializada em semicondutores e soluções de comunicação. Impressionada com o potencial da tecnologia CSS, a Semtech adquiriu a Cycleo, dando início a uma nova fase de desenvolvimento e expansão da tecnologia LoRa.

Com o apoio da Semtech, a tecnologia LoRa evoluiu significativamente e foi adaptada para se tornar o padrão LoRaWAN. O LoRaWAN é um protocolo de comunicação que define a arquitetura e a especificação para a operação da rede LoRa. Ele estabelece a estrutura de como os dispositivos finais, ou end devices, se comunicam com os *gateways*, que são os pontos de acesso à internet e a nuvem.

Uma das principais estratégias para promover a adoção global da tecnologia LoRa foi a criação da LoRa Alliance em 2015. A LoRa Alliance é uma associação aberta e sem fins lucrativos que reúne empresas líderes, provedores de serviços, fabricantes de chips e outros atores do ecossistema IoT interessados em padronizar e promover o uso do LoRaWAN. A colaboração entre os membros da LoRa Alliance permitiu a expansão da infraestrutura LoRa em várias partes do mundo, facilitando a interoperabilidade entre dispositivos e promovendo uma abordagem colaborativa para o desenvolvimento de soluções IoT baseadas em LoRa.

Nos anos seguintes, a rede LoRa ganhou força globalmente, conquistando aplicativos em diversas indústrias e setores, desde cidades inteligentes até agricultura, monitoramento ambiental, logística, saúde e muito mais. Sua implantação e adoção crescente evidenciam a importância e o potencial da rede LoRa na construção de uma sociedade mais conectada, eficiente e inteligente.

Em resumo, a fundamentação teórica da IoT e das tecnologias de conectividade, como a rede LoRa, é de vital importância para compreender as possibilidades e os desafios associados à criação de sistemas inteligentes e conectados. O próximo tópico abordará detalhadamente os princípios e características da rede LoRa, oferecendo uma compreensão mais aprofundada dessa tecnologia promissora.

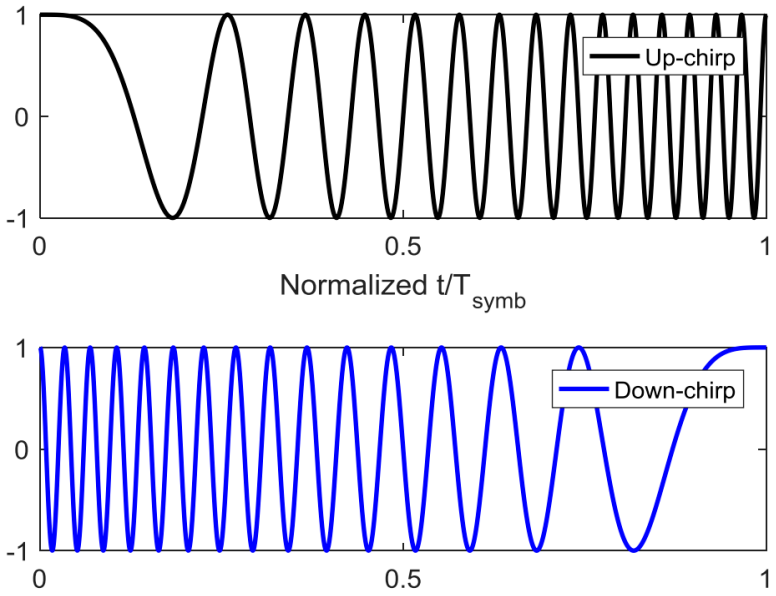
2.3 PRINCÍPIOS E CARACTERÍSTICAS DA REDE LORA

2.3.1 Modulação chirp

A tecnologia LoRa se destaca por sua capacidade de fornecer conectividade de longo alcance com baixo consumo de energia, tornando-a ideal para aplicações IoT em áreas extensas e de difícil acesso. Baseada em uma tecnologia de modulação especial, conhecida como CSS, a LoRa permite a transmissão de dados em distâncias que podem chegar a vários quilômetros em áreas urbanas e até mesmo dezenas de quilômetros em ambientes rurais.

Ainda se tratando da modulação CSS, na Figura 1 temos o comportamento da frequência do *up chirp* no gráfico de cor preta e do *down chirp* no gráfico de cor azul claro “percebe-se que para cada símbolo transmitido há um incremento ou decremento na frequência de transmissão, sendo a banda desta variação fixa. Se há um incremento, é definido como *Up Chirp*, ou decremento como *Down Chirp*.” (Carloto, 2020, p. 51).

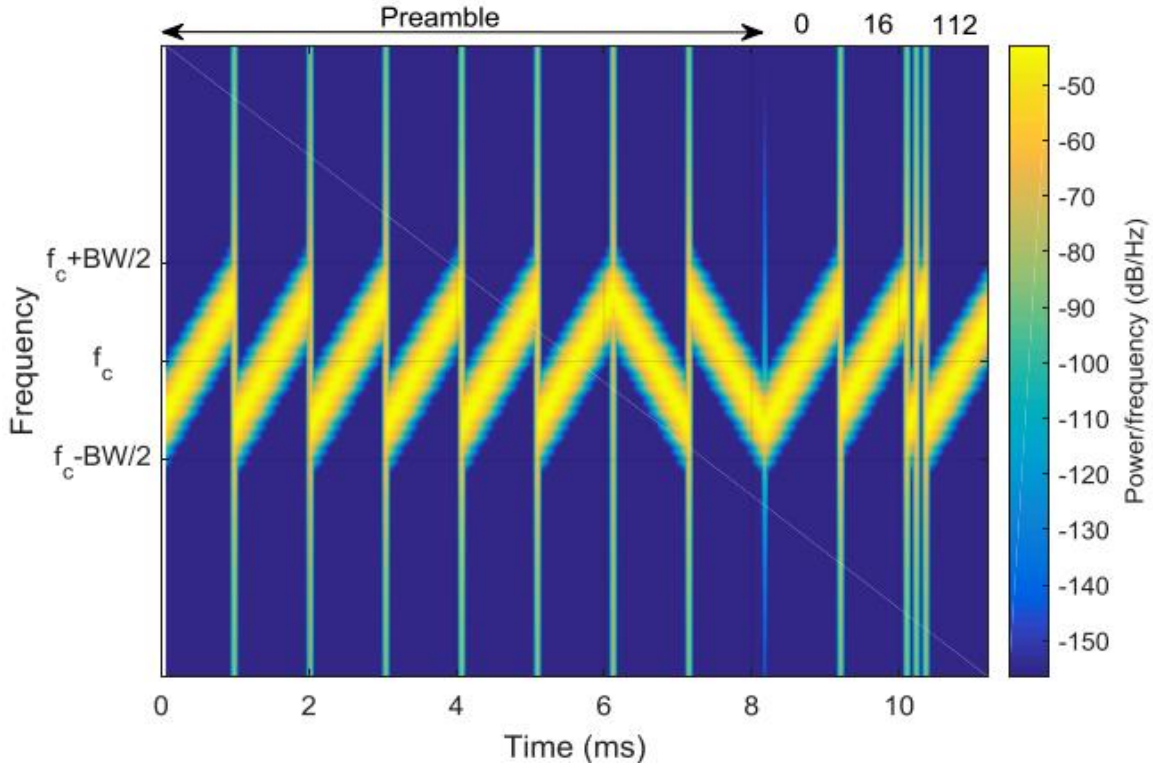
Figura 1 - Sinal *up* e *down chirp* na transmissão LoRa.



Fonte: (Fialho & Azevedo, 2018)

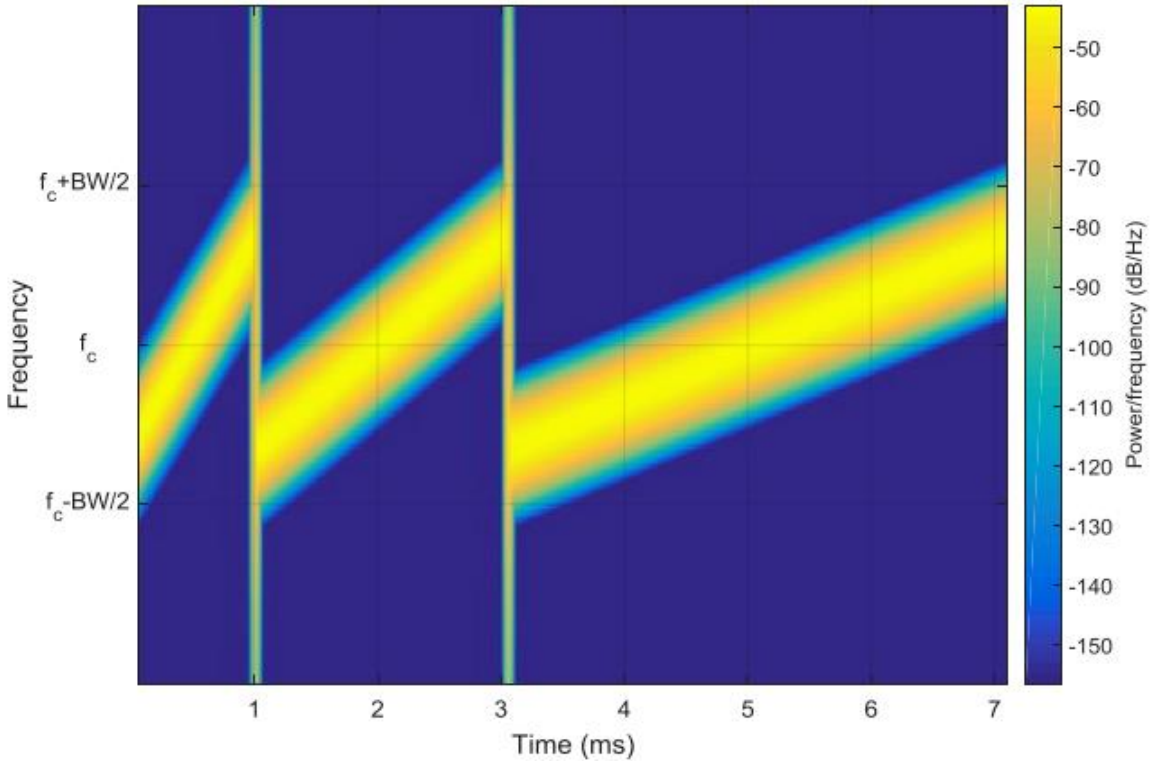
O *Spreading fator* (SF) determina o tempo de variação da frequência em cada *chirp* ou largura de banda. “Para largura de banda podemos ter variações entre 125 kHz, 250kHz e 500kHz, e o SF varia de 6 a 12, que é diretamente proporcional ao tempo para o pacote ser transmitido.” (Semtech, 2019, p. 23). A Figura 2 é um espectrograma cujo objetivo é mostrar o comportamento do sinal dentro da transmissão, nela se tem a representação de seis *up-chirps* e dois *down-chirps* dentro de um espaço de tempo de transmissão de 10ms, no eixo Y é mostrado a faixa de frequência do sinal centrada na frequência central (f_c), e variando entre f_c.-BW/2 e f_c.+BW/2, onde BW é a largura de banda do sinal. O preâmbulo indicado serve para sincronia do receptor com o transmissor antes da transmissão de dados. As cores representam a potência do sinal, onde as mais próximas do amarelo indicam maior potência enquanto as mais próximas do azul escuro representam menor potência. Pode-se notar na Figura 3 “para um SF de 7, 8 e 9 o tempo de transmissão aumenta diretamente com o aumento do SF.” (Carloto, 2020, p. 51).

Figura 2 - Espectrograma da comunicação LoRa.



Fonte: (Fialho & Azevedo, 2018)

Figura 3 - Tempo de ar para diferentes SF.

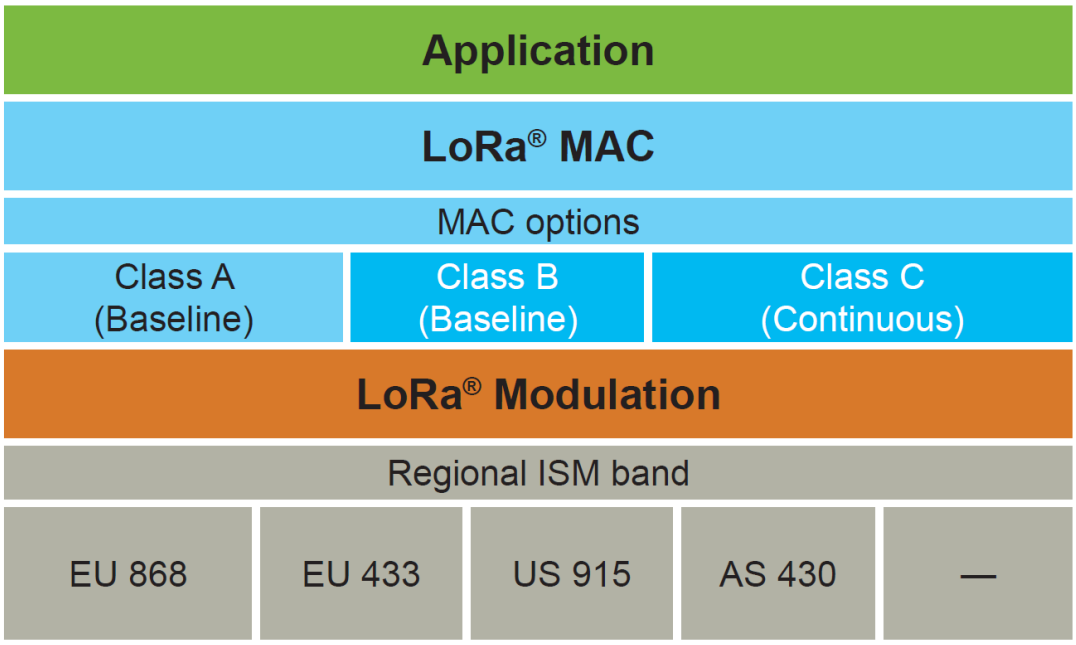


Fonte: (Fialho & Azevedo, 2018)

2.3.2 Arquitetura da rede LoRaWAN

A arquitetura da rede LoRaWAN, que padroniza a tecnologia LoRa, divide-se em três camadas conforme Figura 4: a camada de percepção *(end devices)*, a camada de rede (*gateways*) e a camada de aplicação. Os dispositivos finais, também chamados de *end nodes* ou sensores LoRa, coletam dados e os enviam para os *gateways* através da modulação LoRa. Os *gateways*, por sua vez, encaminham esses dados para um servidor de rede onde podem ser processados, analisados e enviados para uma determinada aplicação que se encontra no servidor de aplicação.

Figura 4 - Arquitetura da rede LoRaWAN.



Fonte: (LoRa® Alliance Technical Marketing Workgroup, 2015, p. 7)

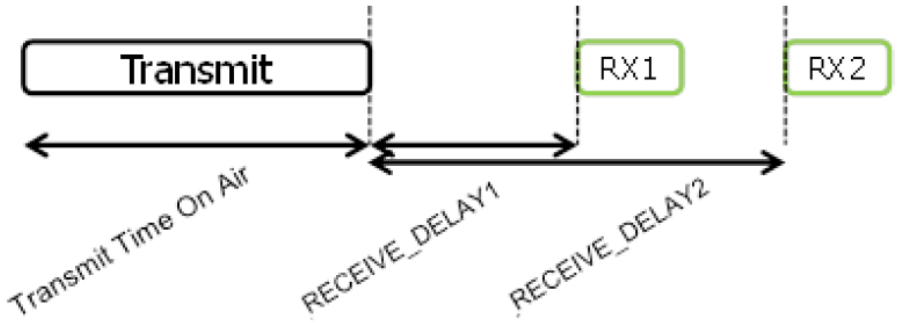
2.3.3 Dispositivos finais

O protocolo LoraWAN define 3 classes de dispositivos, sendo representados pelas letras A, B e C. Isso se dá pelo fato deles não serem capazes de transmitir e receber dados de forma simultânea, desta forma alguns padrões foram criados para

que janelas de recepção e transmissão fossem abertos dentro de alguns conjuntos de regras criadas dentro de cada classe.

Nos dispositivos de classe A, o rádio opera em mais baixo consumo energético. O dispositivo pode aleatoriamente enviar mensagens para o servidor. O dispositivo abre então duas janelas de recepção em momentos especificados após uma transmissão conforme mostrado na Figura 5. Se o servidor não responder em nenhuma dessas janelas de recepção ele terá uma próxima oportunidade após a próxima transmissão de link ascendente do dispositivo. O servidor pode então emitir uma resposta na primeira janela de recebimento ou na segunda janela de recebimento, mas nunca nas duas janelas. (Committee, 2017, p.14). Após o término da transmissão o dispositivo pode entrar em uma condição de sono profundo para poupar energia.

Figura 5 - Janelas de transmissão e recepção para dispositivos de classe A.



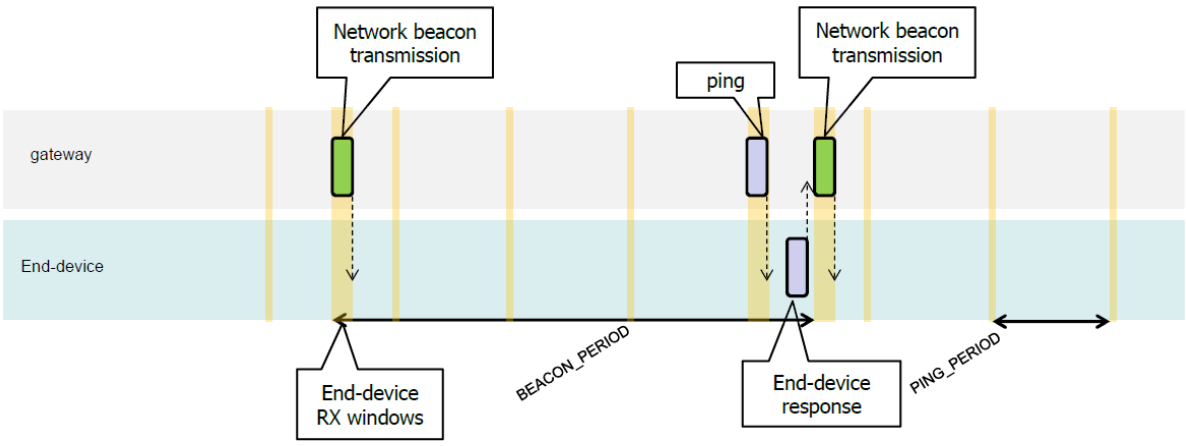
Fonte: (Committee, 2017, p.13)

Nos dispositivos de classe B, o servidor de rede envia pacotes de sincronismo *(beacons)* para o *gateway* distribuir entre os dispositivos finais, adicionando janelas de recebimento programadas para mensagens de *downlink* do servidor. Desta forma o servidor tem o controle do momento que o dispositivo está apto para receber uma mensagem e faz a transmissão no momento adequado.

Na Figura 6, considerando que o período do beacon é de 128 segundos, o dispositivo final também abre uma janela de recepção de ping a cada 32 segundos. Normalmente, essa janela de recepção não é utilizada pelo servidor, o que faz com que o dispositivo feche rapidamente a janela assim que o transceptor de rádio verifica que não há um preâmbulo (sinal de início de transmissão) no canal. Caso o preâmbulo seja detectado, o transceptor mantém o rádio ativo até que o quadro de *downlink* seja

recebido e demodulado. Após isso, a camada MAC (responsável pela comunicação) processa o quadro, confirma se o endereço do destinatário corresponde ao do dispositivo final e verifica a integridade da mensagem antes de encaminhá-la para a camada de aplicação, que tratará os dados recebidos.

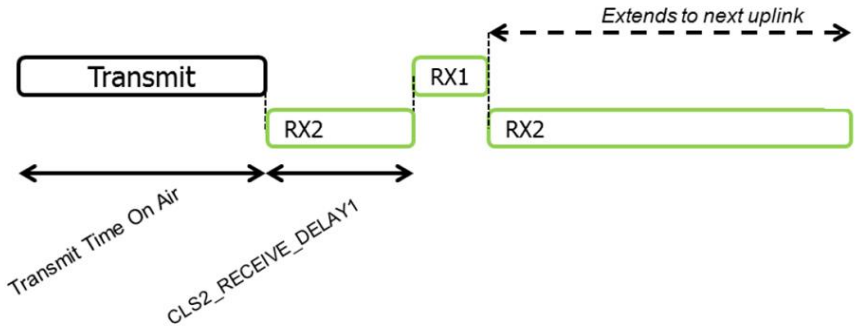
Figura 6 - Janela de transmissão e recepção para dispositivos classe B.



Fonte: (Committee, 2017, p.70)

Dispositivos classe C: não foram desenvolvidos para baixo consumo de energia, pois nesta classe as janelas de recepção ficam a todo momento abertas, exceto nos momentos onde o dispositivo está fazendo alguma transmissão conforme mostrado na Figura 7 “Com isto tem-se uma comunicação de baixa latência, e um elevado consumo energético comparado aos dispositivos Classe A.” (Radioenge, s.d., p. 10).

Figura 7 - Janela de transmissão e recepção para dispositivos classe C.



Fonte: (Committee, 2017, p.87)

2.3.4 Bandas de frequência por região

A rede LoRa *(Long Range)* opera em diferentes frequências em todo o mundo, a fim de cumprir com as regulamentações e restrições de espectro de cada região. A Tabela 2 apresenta algumas das frequências comuns e as regiões em que são utilizadas para a rede LoRa:

Tabela 2 - Frequências de operação da rede LoRa por região.

Região	**Frequência (MHz)**
América do Norte	902-928
Europa	863-870
China	470 – 510 e 778-787
Índia	865 - 867
Austrália e Nova Zelândia	915-928
América do Sul	915-928
Ásia (regiões não especificadas)	923 - 925

Fonte: Adaptado pelo autor de (The Things Network, s.d.)

É importante observar que as frequências podem variar dentro de cada região e existem outras faixas de frequências designadas para LoRa em diferentes partes do mundo. Além disso, é fundamental verificar as regulamentações específicas e as faixas de frequências permitidas em cada país ou localidade antes de implantar uma rede LoRa para garantir conformidade com as normas locais.

2.3.5 Topologia e componentes da rede LoRa

O conceito de topologia está diretamente ligado a forma onde os dispositivos que compõe a rede estão interligados, tendo como foco o nó central ou *gateway* e a posição que ele assume nessa rede. Para a rede LoRa há a topologia estrela e a topologia *mesh*.

Na topologia estrela (Figura 8), todos os dispositivos IoT se conectam diretamente a um ponto central, como um *gateway* LoRa, que atua como um concentrador central para coletar e encaminhar os dados para a nuvem ou para uma rede local. Nesse cenário, os dispositivos IoT não se comunicam diretamente entre si, mas apenas com o *gateway*.

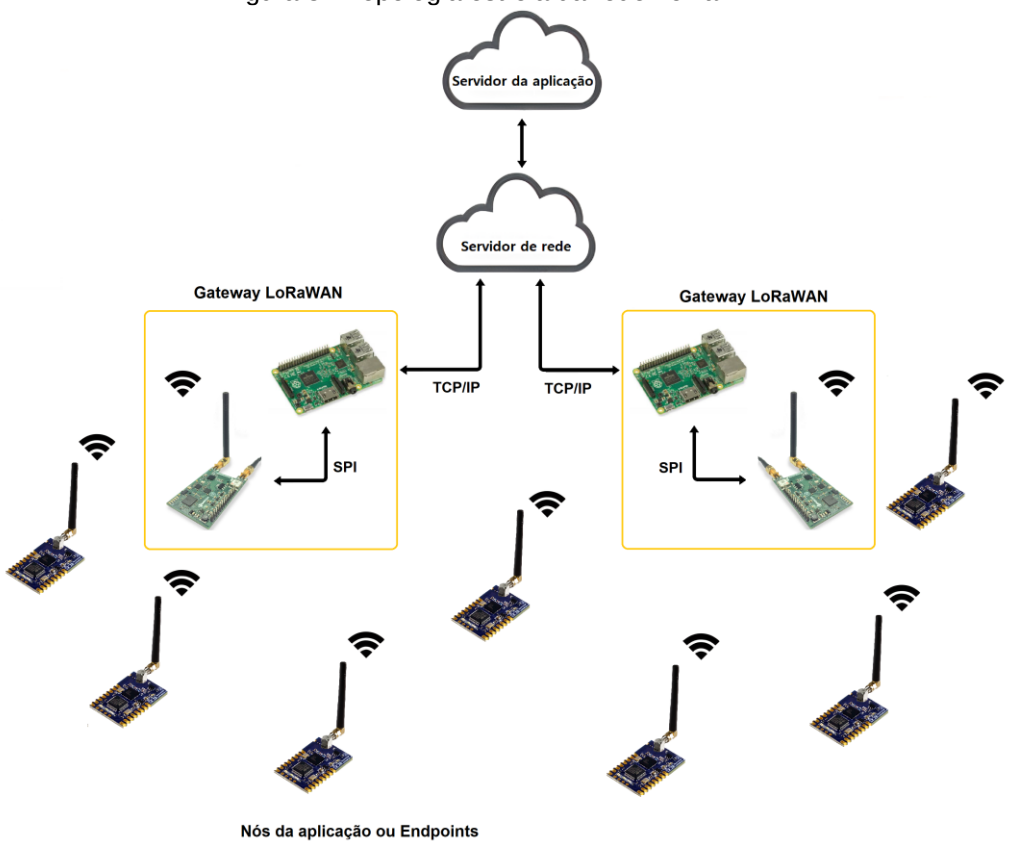
Vantagens:

- Simplicidade de implementação e gerenciamento, pois todos os dispositivos convergem para um único ponto central.
- Baixo consumo de energia nos dispositivos IoT, já que eles não precisam retransmitir dados.
- Menor latência, pois não há roteamento complexo envolvido.

Desafios:

- Dependência total do *gateway*; se ele falhar, toda a rede pode ficar inacessível.
- Restrições na escalabilidade, pois o *gateway* pode ficar sobrecarregado ao lidar com um grande número de dispositivos.

Figura 8 - Topologia estrela da rede LoRaWAN.



Fonte: (Manual de operação end device LoRaWAN, 2023, p. 3)

Na topologia *mesh* (Figura 9*)*, todos os dispositivos IoT são capazes de se comunicar diretamente entre si, formando uma rede auto organizável e descentralizada. Cada dispositivo pode atuar como um nó de roteamento, encaminhando os dados para outros dispositivos até que eles alcancem o destino final.

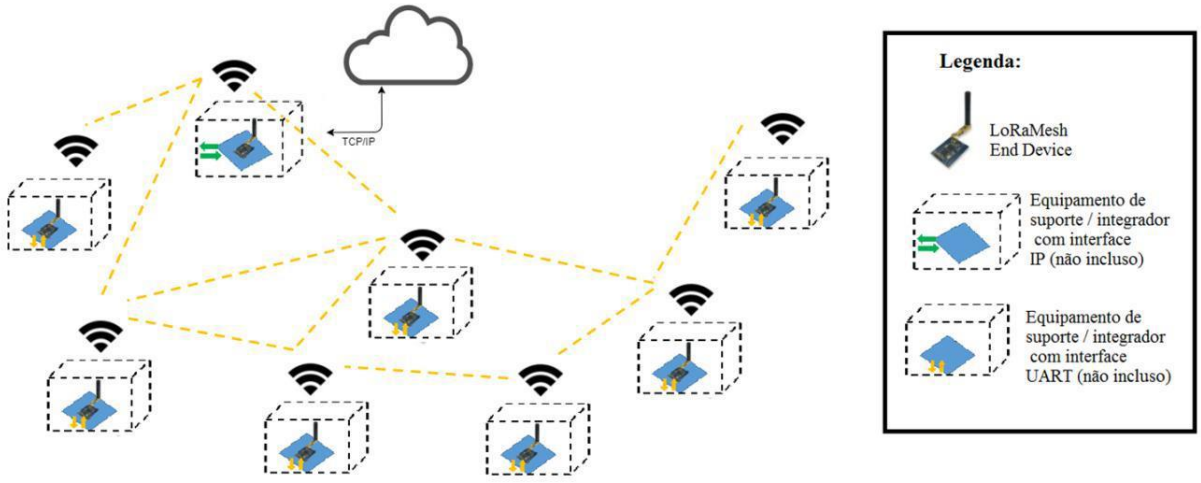
Vantagens:

- Alta redundância e resiliência, pois não há ponto único de falha; os dados podem ser roteados por diferentes caminhos.
- Melhor escalabilidade, pois novos dispositivos podem ser facilmente adicionados, ampliando a cobertura da rede.
- Maior autonomia, pois o roteamento adaptativo pode economizar energia, encaminhando dados por caminhos mais eficientes.

Desafios:

- Complexidade de roteamento, que pode exigir algoritmos de roteamento avançados para garantir a eficiência da rede.
- Consumo de energia mais elevado em dispositivos atuando como nós de roteamento, especialmente em cenários de tráfego intenso.

Figura 9 - Diagrama de uma rede *mesh*.



Fonte: (Manual de utilização módulo LoRaMESH, 2022, p. 3)

3 DESENVOLVIMENTO DA REDE LORAWAN

3.1 SOBRE OS HARDWARES UTILIZADOS

No desenvolvimento deste trabalho, diversos dispositivos de *hardware* foram utilizados, cujas especificações e características técnicas são detalhadas a seguir.

3.1.1 Gateway Radioenge

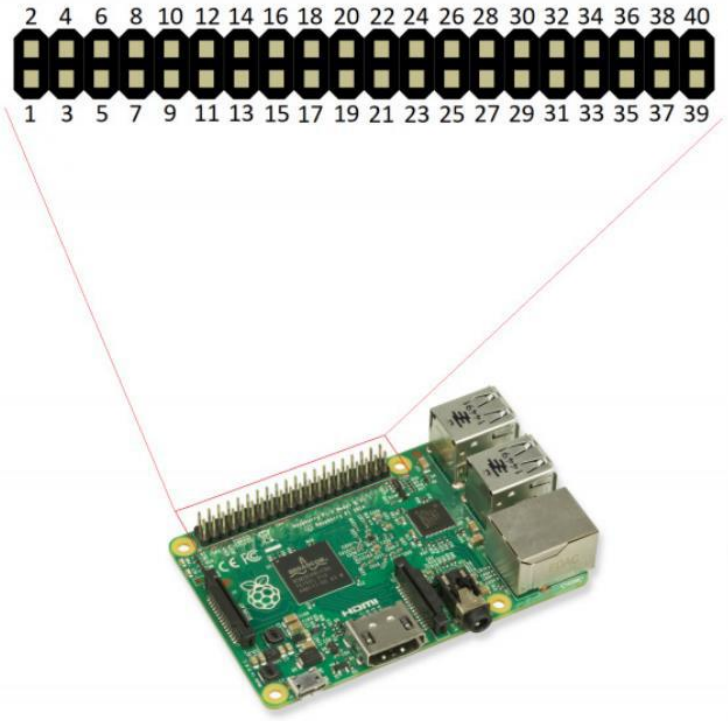
O *gateway* é um dispositivo usado para intermediar a comunicação entre os *end devices* e o servidor de rede na rede LoRaWAN, sua topologia é a estrela, onde apenas o dispositivo final se comunica com ele. Sua estrutura é um *shield* que se acopla a um Raspberry Pi 3, responsável por gerenciar as funcionalidades do rádio por meio de uma interface SPI (*Serial Peripheral Interface*).

Nas Figuras 10 e 12 se têm, respectivamente a estrutura do gateway conforme fornecido pelo fabricante, e sua montagem na placa Raspberry Pi 3 que tem sua estrutura representada na Figura 11 com destaque para os 40 pinos de conexão. Na Figura 13 tem-se a estrutura completa instalada no laboratório, já com a estrutura de rede e de antenas. Foram utilizadas duas antenas, a primeira é uma omnidirecional com ganho de 9dbi responsável pela transmissão LoRa e a segunda é uma antena para captação do sinal de GPS, com finalidade de determinar a localização do conjunto.

Figura 10 - *Gateway* LoRaWAN da Radioenge.

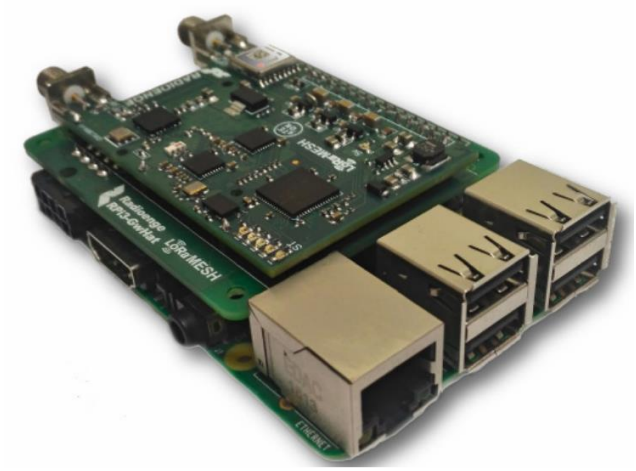
Fonte: (Manual Técnico Gateway LoRaWAN, 2022, p. 9)

Figura 11 – Raspberry Pi 3 e sua pinagem.



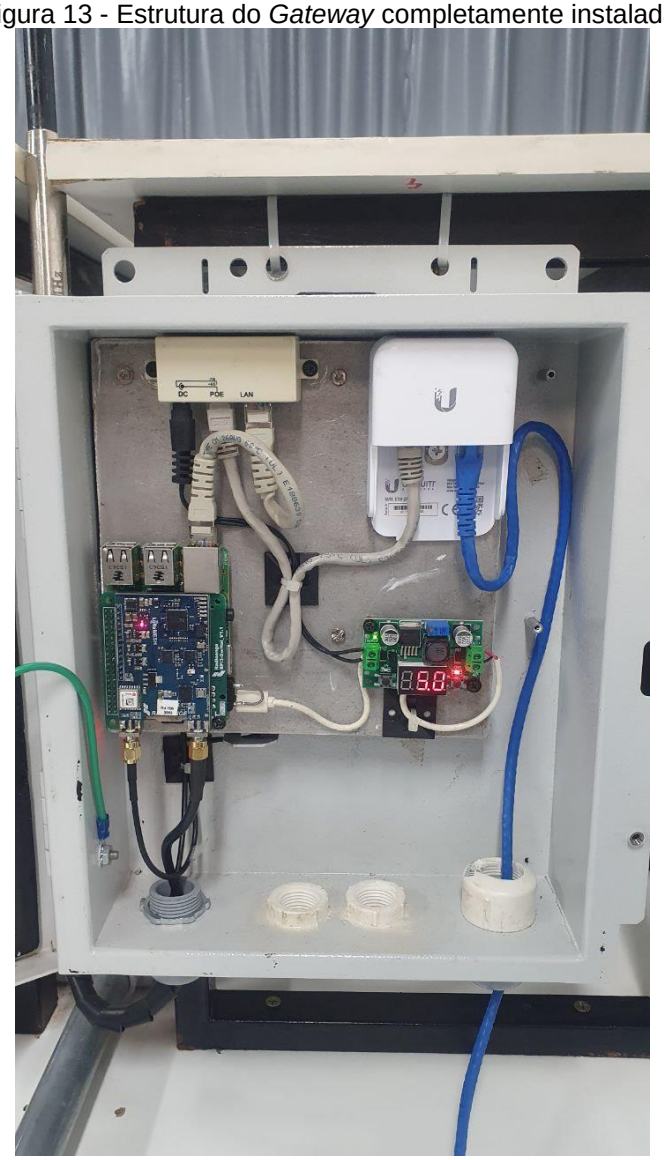
Fonte: (Manual Técnico Gateway LoRaWAN, 2022, p. 8)

Figura 12 - *Gateway* LoRa acoplado ao Raspberry Pi 3.



Fonte: (Manual Técnico Gateway LoRaWAN, 2022, p. 9)

Figura 13 - Estrutura do *Gateway* completamente instalada.

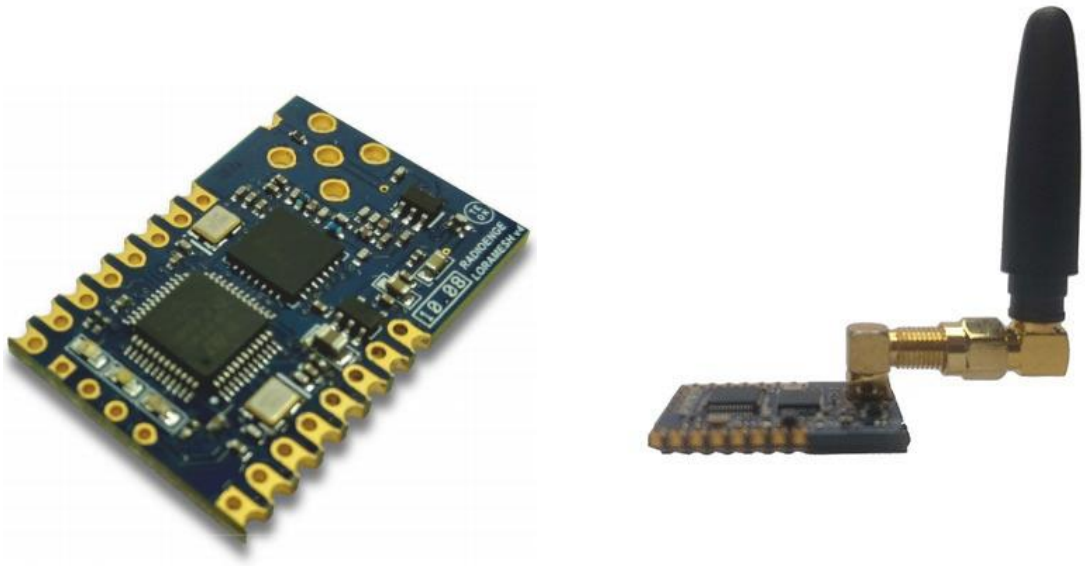


Fonte: Do autor

3.1.2 Dispositivos finais ou *end devices* Radioenge

São dispositivos instalados em campo, geralmente ligados a sensores que coletam dados e enviam por meio da rede LoRa. Sua construção conta com um microprocessador ARM CortexM0+ de 32-bits responsável por gerenciar as GPIO's e um circuito de modulação CSS ou FSK, dependendo das configurações feitas via *software* pelo usuário. O aspecto construtivo dos *end devices* LoRaWAN e LoRaMESH podem ser visualizados na Figura 14, são idênticos, neste caso se deve fazer o download do *firmware* adequado e instalar de acordo com a topologia desejada, ou adquirir com o *firmware* desejado já instalado de fábrica. O dispositivo conta com uma antena que opera na faixa de 902 a 928MHz, com ganho de 2,15dbi.

Figura 14 - Módulo final (*end device*) da Radioenge.



Fonte: (Manual de operação end device LoRaWAN, 2023)

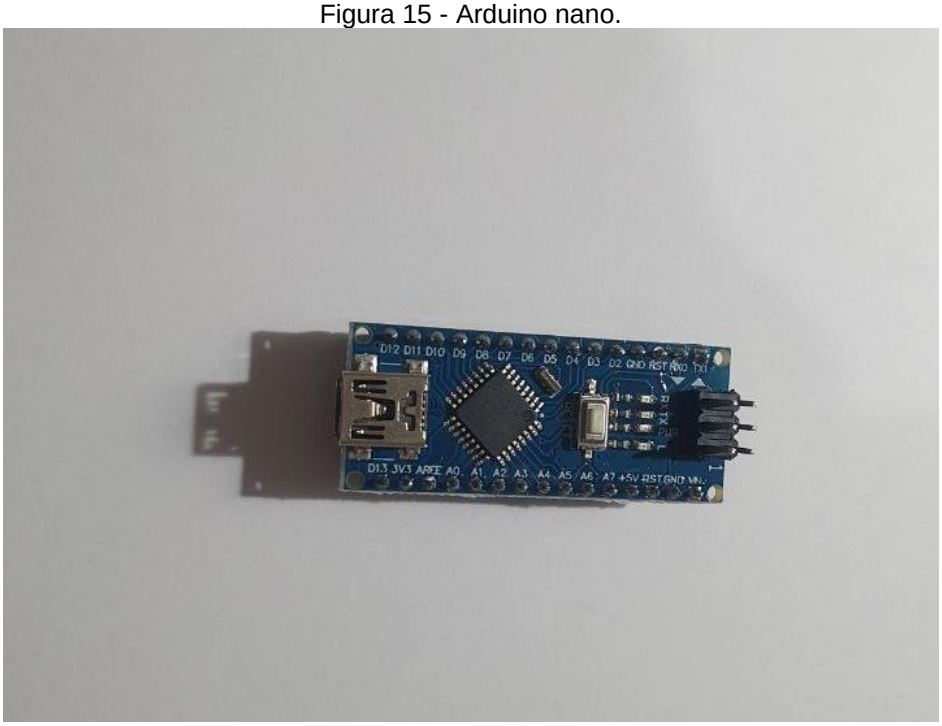
3.1.3 Hardwares adicionais

Para efetuar testes nos módulos fez-se necessário utilizar alguns dispositivos dotados de microcontroladores, tais dispositivos tem como função gerar sinais para

teste de alcance da rede LoRaWAN, e também fornecer informações da rede *mesh* através do monitor serial, e controlar os dispositivos escravos.

O primeiro deles o Arduino® Nano apresentado na Figura 15, foi utilizado como disparador de sinais para o módulo LoRaWAN. A placa de desenvolvimento Arduino® Nano é projetada para facilitar a prototipagem rápida. Com dimensões reduzidas e um conjunto de interfaces adequadas, é suficiente para várias aplicações. Equipada com o microcontrolador ATmega328, que opera a um clock de 16 MHz, oferece 22 pinos digitais, 8 pinos analógicos e uma porta mini-USB. (Arduino.cc, 2023, p. 1).

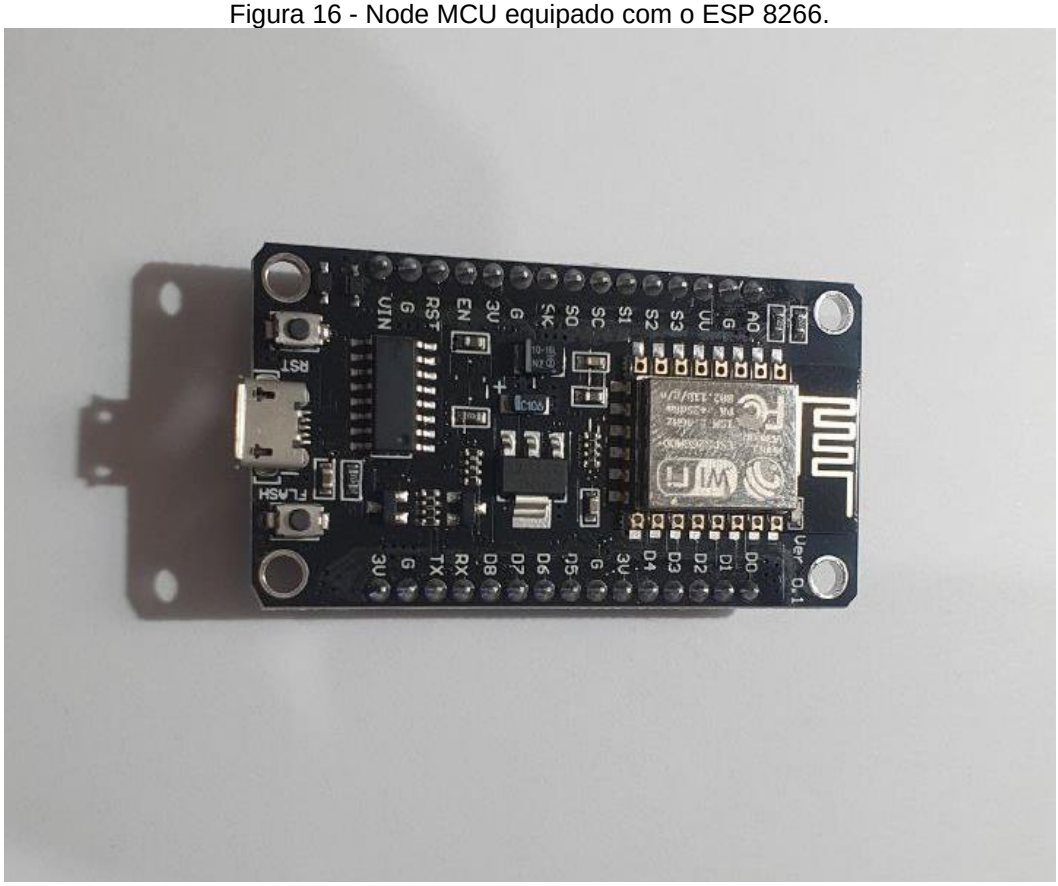
Figura 15 - Arduino nano.



Fonte: Do autor

Este trabalho também utilizou um NodeMCU ESP 8266, que é uma placa de prototipagem voltada a IoT, sua grande quantidade de pinos e alta capacidade de processamento a torna ideal para o desenvolvimento de projetos, além disto, sua programação pode ser feita via IDE do Arduino, o que facilita bastante a utilização, pode-se ver sua estrutura física na Figura 16. O objetivo é utilizá-lo como ponte de comunicação entre os *end devices* LoRaWAN e LoRaMESH.

Figura 16 - Node MCU equipado com o ESP 8266.

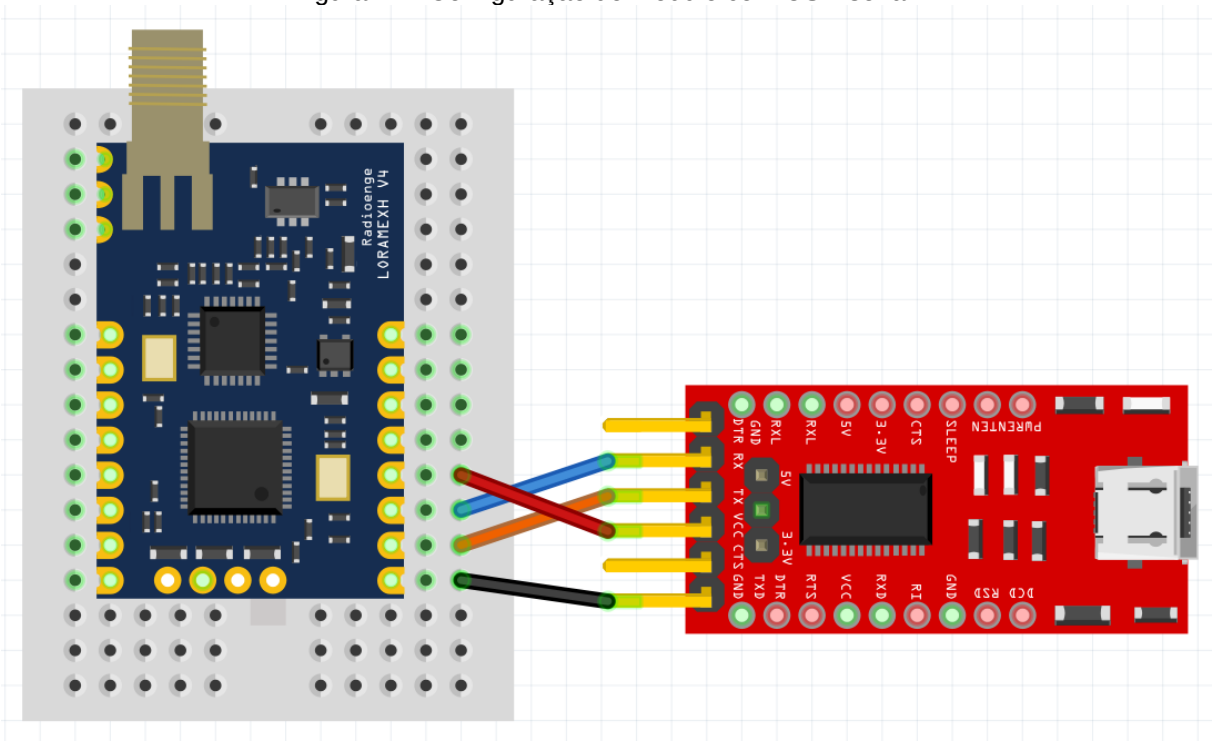


Fonte: Do autor

3.2 IMPLEMENTAÇÃO DA REDE LORAWAN

Para desenvolver a rede LoRaWAN, primeiramente foi necessário configurar o módulo com uso do *software* MOTE, que é utilizado para a configuração dos módulos disponibilizados pela Radioenge. Para conectar o módulo ao computador, basta usar uma interface USB-Serial conforme Figura 17, em seguida realizar as configurações necessárias no módulo através do *software*.

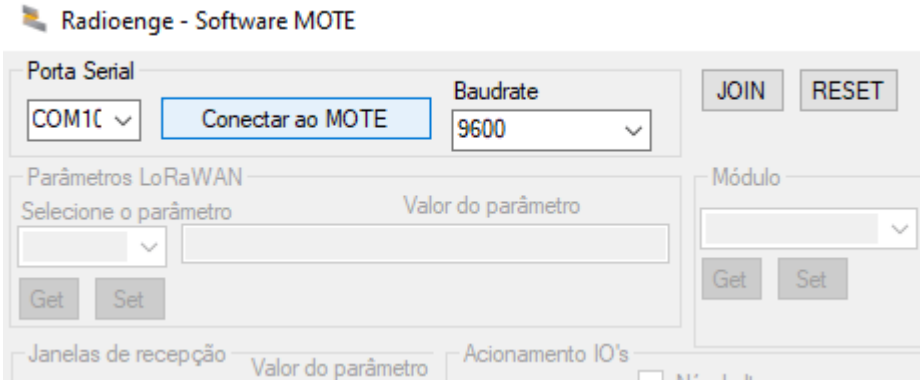
Figura 17 - Configuração do módulo com USB-serial.



Fonte: Do autor

Em seguida deve-se selecionar a porta COM relacionada ao dispositivo, ajustar o baudrate para 9600 e clicar em conectar ao MOTE conforme mostrado na Figura 18, feito isso os demais parâmetros são liberados na tela para configuração.

Figura 18 - Configuração da porta COM e baudrate a partir do *software* MOTE.



Fonte: Do autor

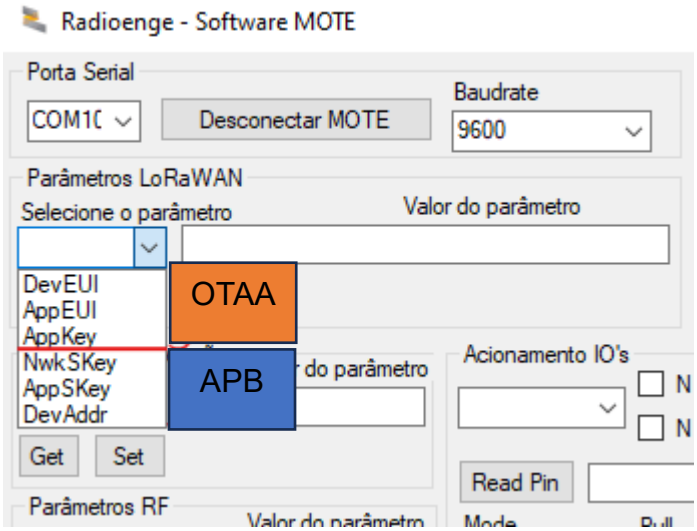
Na seção de parâmetros LoRaWAN, é preciso registrar os valores de quatro parâmetros para inserção no site onde a aplicação será criada. Esses parâmetros são:

- DevEUI
- Device Address

- AppSKey
- NwkSKey

Na seção contém seis parâmetros distintos que são destinados a dois métodos de conexão com o servidor de rede, o primeiro é o OTAA e o segundo é o APB, destacados na Figura 19.

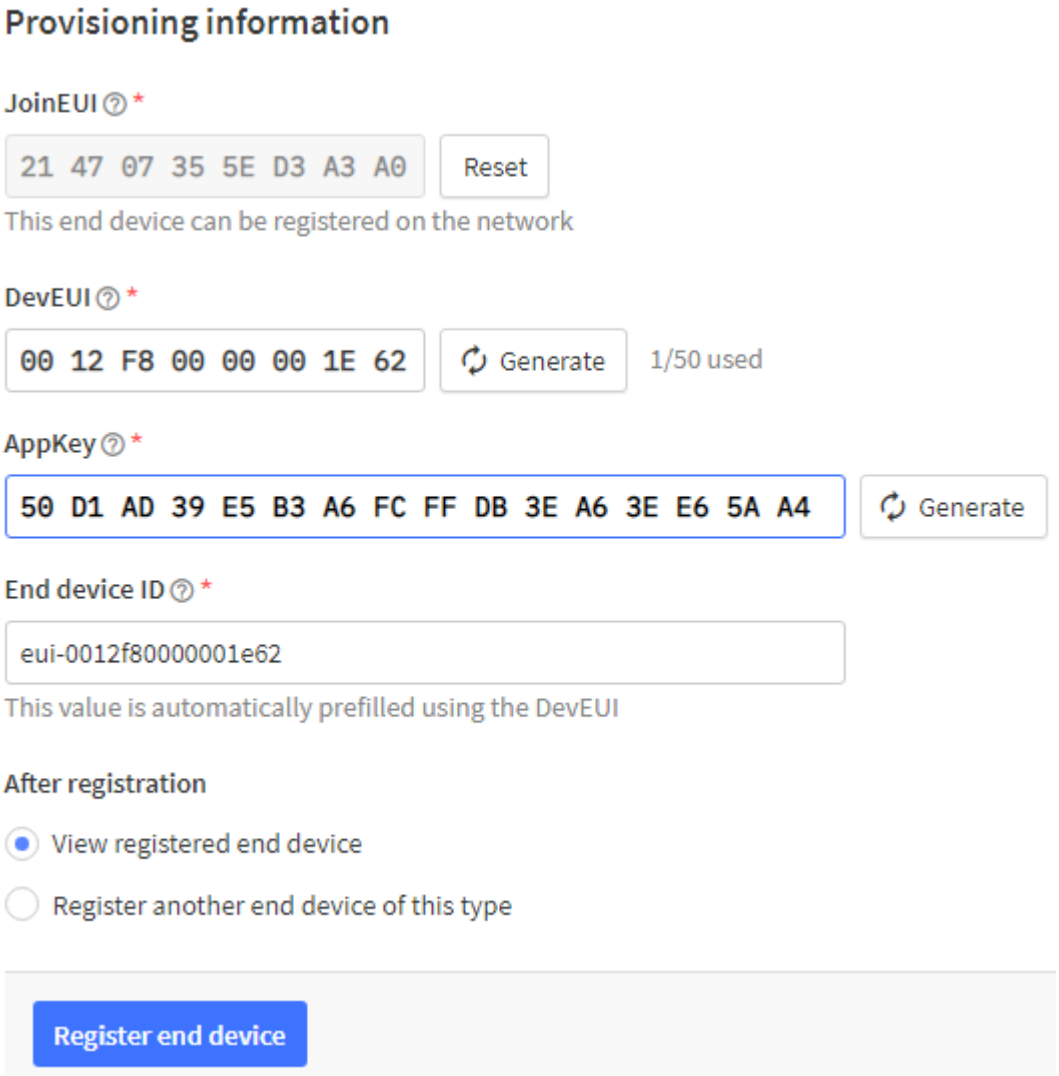
Figura 19 - Seleção de parâmetros LoRaWAN a partir do *software* MOTE.



Fonte: Do autor

Com os dados coletados, basta abrir uma conta no site do *The Things Network* (TTN), criar uma aplicação, ir em *register end device*, ajustar os parâmetros de frequência de acordo com a região, em seguida entrar com os parâmetros coletados via *software* anteriormente, e inserir esses dados na janela que vai surgir, conforme mostrado na Figura 20.

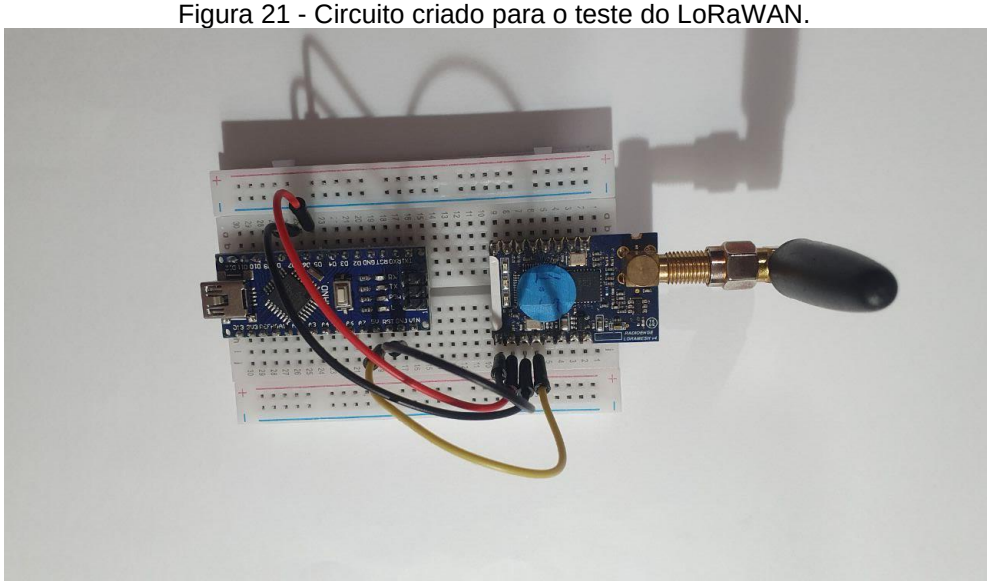
Figura 20 - Entrada de parâmetros a partir da tela do TTN.



Fonte: Do autor

Após configurado o TTN com o dispositivo, foi gerado um código para Arduino, o qual seria responsável para servir como um disparador de sinais no módulo. A partir dele, pode-se ver a conexão na tela do TTN feita a partir do *gateway* cadastrado na plataforma. O circuito gerado para teste do código se encontra na Figura 21. O código se encontra no ANEXO A – e sua explicação sucinta pode-se ver logo em seguida.

Figura 21 - Circuito criado para o teste do LoRaWAN.



Fonte: Do autor

Este código configura um dispositivo para se comunicar em uma rede LoRaWAN usando o método de autenticação ABP. A cada 15 segundos, o dispositivo envia uma *string* contendo um contador que é incrementado em cada ciclo, e é exibido no monitor serial.

- **Passo 1 - Inclusão de bibliotecas**

  O Primeiro passo do código é incluir bibliotecas, que são conjuntos de ferramentas com comandos e funções prontas para serem utilizadas:
  LoRaWAN: Esta biblioteca contém funções e definições necessárias para implementar a comunicação LoRaWAN.
  SoftwareSerial: Esta biblioteca permite a comunicação serial em pinos digitais que não são usados pela porta serial padrão, permitindo a conexão de mais dispositivos.
  stdint: Essa biblioteca fornece definições para tipos de dados inteiros de tamanhos específicos, como int8_t, int16_t, etc.

- **Passo 2 – Declaração de variáveis**

  Aqui, várias variáveis são definidas:

hSerialCommands: Um ponteiro para uma instância da classe SoftwareSerial, que será usado para comunicação em série.
DADDR: Um array de caracteres que contém o endereço do dispositivo na rede LoRaWAN.
APPSKEY: Uma chave de aplicação (Application Key) necessária para a autenticação na rede LoRaWAN.
NWKSKEY: Uma chave de rede (Network Session Key) usada para autenticação e segurança da rede.
devAddr: Uma string que contém o endereço do dispositivo.
str_counter: Um array de caracteres para armazenar uma string que conterá um contador.
counter: Um contador que será incrementado e enviado.

- **Passo 3 - Configuração inicial setup**

Serial.begin(9600): Inicializa a comunicação serial com uma taxa de 9600 bps (bits por segundo), permitindo a comunicação com o monitor serial.
Serial.println("Initializing..."): Envia uma mensagem ao monitor serial informando que a inicialização está em andamento.
delay(5000): Aguarda 5 segundos, permitindo que o sistema se estabilize antes de prosseguir.
hSerialCommands = SerialCommandsInit(7, 6, 9600): Inicializa a comunicação serial em pinos digitais 7 (RX) e 6 (TX) com a mesma taxa de 9600 bps.
SendAtCommand(AT_NJM, AtSet, "0"): Envia um comando AT para configurar o dispositivo para usar o método de autenticação ABP (Activation By Personalization).

- **Passo 4 – Função principal loop**

sprintf(str_counter, "Counter: %d\r\n\0", counter++): Formata a string com o valor atual do contador e a armazena em str_counter. O contador é incrementado após a formatação.
Serial.println(str_counter): Envia a string formatada para o monitor serial, permitindo que o usuário veja o valor do contador.

SendString(str_counter, 2): Envia a string str_counter via LoRaWAN para o dispositivo no endereço 2. Este comando assume que a função SendString já está definida em outro lugar do código.
delay(900): Aguarda 900 milissegundos (0,9 segundos) antes de repetir o loop. Isso significa que o contador será enviado aproximadamente a cada 15 segundos, considerando o tempo total do loop.

3.3 TESTES E RESULTADOS

Para validação da conexão do *end device* com o *gateway*, foi coletado os dados mostrados na Figura 22, ela é um *log* de comunicação LoRaWAN, com registros de mensagens enviadas e recebidas. Na coluna *Time*, é mostrado o horário de ocorrência dos eventos, na coluna *Type* pode se acompanhar quando o *gateway* está enviando ou recebendo mensagens. Em *Data preview* são mostrados alguns parâmetros de conexão sendo eles:

- **Receive *gateway* status:** Mostra o status, com métricas como *ackr, txok, txin*, entre outros, que indicam a confirmação e a transmissão de pacotes;
- **Tx Power:** Potência de transmissão utilizada no *downlink;*
- **Data rate:** A taxa de transmissão de dados, como SF12BW500, que indica o fator de espalhamento (SF) e a largura de banda (BW);
- **DevAddr:** Endereço do dispositivo que enviou a mensagem para o *gateway*;
- **FPort:** Porta do frame, variando de 1 a 223 é utilizado para definir qual aplicação ou serviço específico que deve processar a mensagem;
- **SNR:** É a relação sinal ruído, que indicam a qualidade do sinal recebido entre a comunicação do dispositivo e o *gateway*.

A comunicação se dá como bem sucedida, a partir do momento onde se aparece *receive uplink mesage* e *send downlink message*, a primeira está diretamente relacionada ao *end device*, onde consta o endereço cadastrado no dispositivo e o parâmetro FCnt que é um contador de quadros que ajuda o servidor de rede a manter um controle de todas as mensagens enviadas e recebidas assegurando que não houve perda de nenhuma delas.

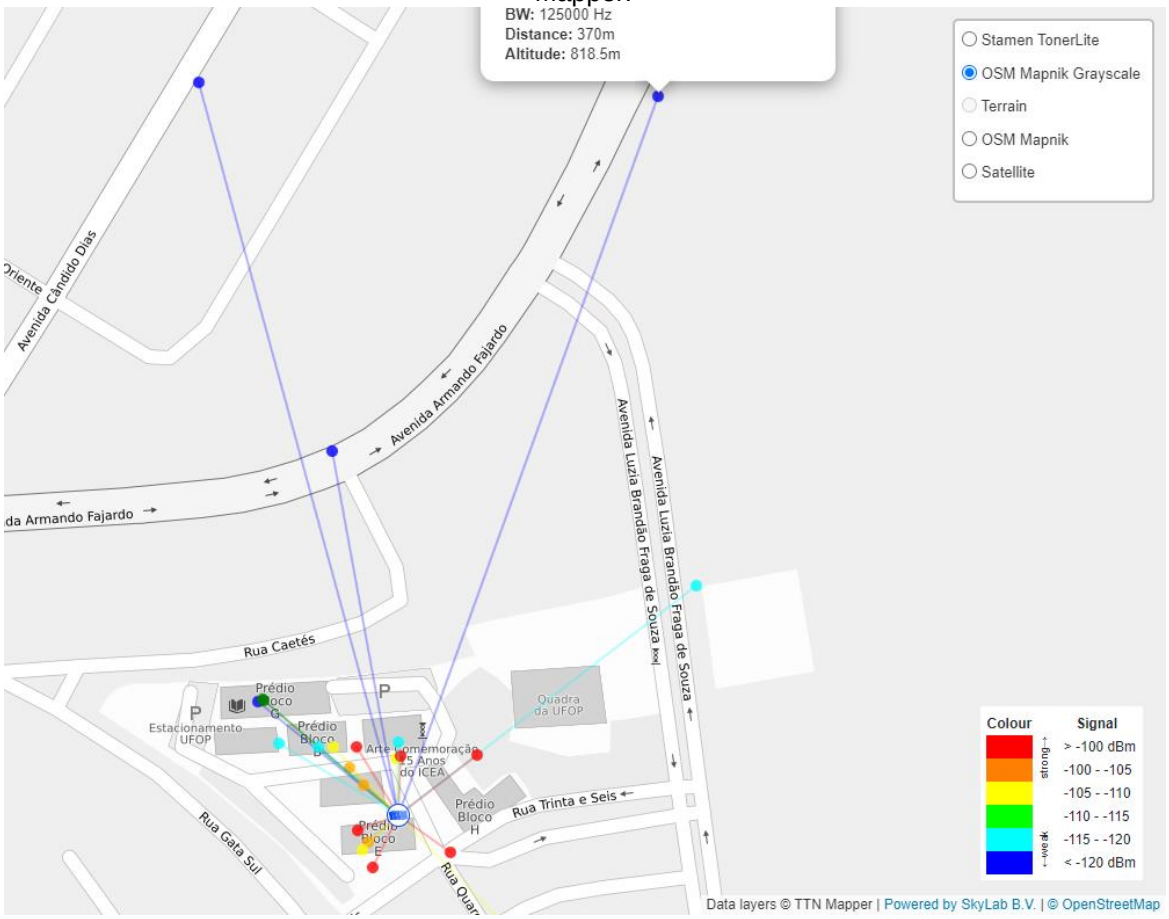
Figura 22 - Dados gerados da conexão do módulo com o *gateway* a partir da tela do TTN.

Fonte: Do autor

Para testar o alcance do módulo LoRaWAN fez-se o uso de um aplicativo para *smartphone* chamado TTN Mapper, para realizar esses testes, uma série de percursos foi realizado no entorno da localidade onde se encontra o *gateway*, para isto levou-se o circuito contendo o dispositivo alimentado através de um *power bank* e através da localização do GPS do *smartphone* gerou-se pontos no mapa relativos ao ponto onde se houve uma comunicação entre o dispositivo e o *gateway*.

Na Figura 23, os pontos estão indicando a potência do sinal em dBm, apresentada em cores conforme a legenda, nela é mostrado que sinais mais fortes estão na escala acima de -100 dbm (pontos em vermelho), havendo perda de potência do sinal a medida que se distancia o módulo do *gateway* chegando no limiar de transmissão por volta de valores próximos aos -120dbm (pontos em azul) onde foram definidos os limites de comunicação durante os testes, o ponto mais distante alcançado em testes, foi de aproximadamente 370m do *gateway*, mas a tecnologia LoRa permite comunicações a vários quilômetros de distância, tudo depende da qualidade das antenas dos dispositivos, das interferências externas, e do posicionamento onde os dispositivos estão instalados. Na ocasião apresentada se tem um *gateway* instalado dentro de uma edificação, onde sua antena se encontra instalada no interior do edifício. O entorno também apresenta bastante obstáculos, como presença de diversas residências e prédios, sem contar o relevo acidentado que intensifica ainda mais a ação destes obstáculos na obstrução do sinal.

Figura 23 - Pontos de localização do *end device* durante a comunicação a partir da tela do TTN. Mapper.



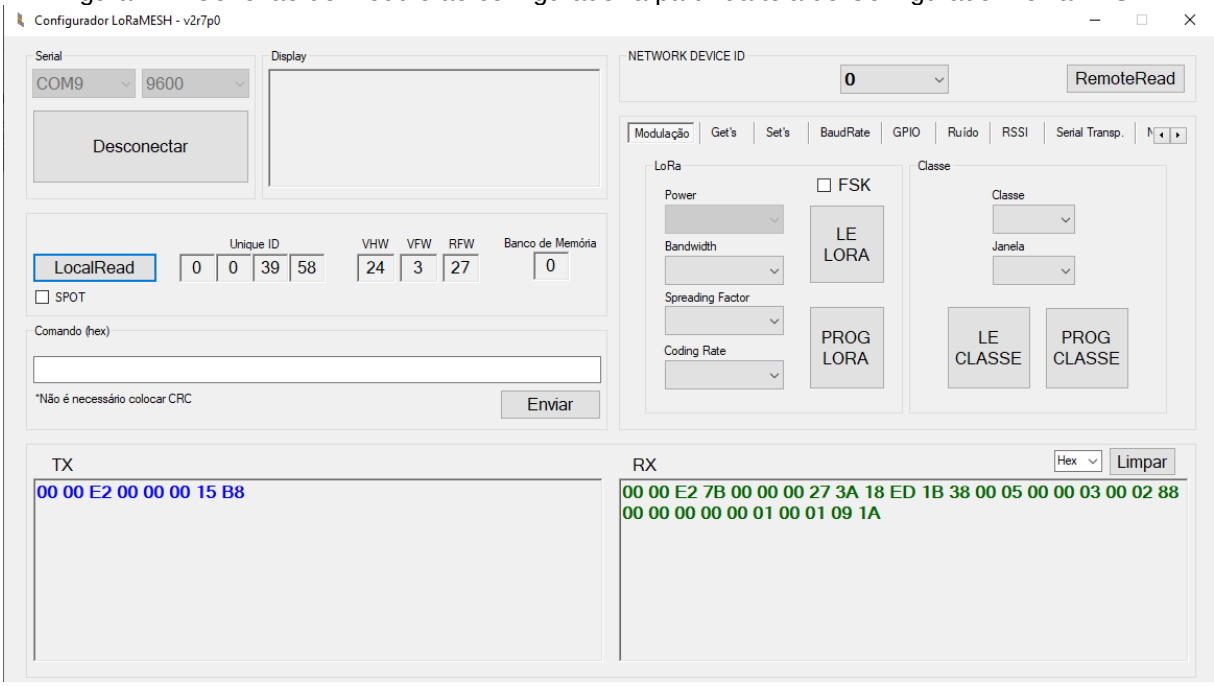
Fonte: Do autor

4 DESENVOLVIMENTO DA REDE LORAMESH

4.1 IMPLEMENTAÇÃO DA REDE LORAMESH

Para criar a rede *mesh*, foi necessário configurar os dois módulos, o mestre e o escravo. O mestre é o responsável pelo gerenciamento da rede, e é capaz de se comunicar com vários escravos, utilizando técnicas como o *polling,* por exemplo, que faz a consulta dos dispositivos um por vez a partir de um tempo configurado. Conforme a Figura 17, o módulo escolhido para atuar como mestre deve ser conectado a um computador rodando o *software* configurador da Radioenge desenvolvido para LoRaMESH, em seguida deve ser escolhida a porta COM correspondente ao dispositivo e clicado em conectar. A conexão será bem sucedida quando os parâmetros do dispositivo forem exibidos na tela conforme Figura 24.

Figura 24 - Conexão do módulo ao configurador a partir da tela do Configurador LoRaMESH.



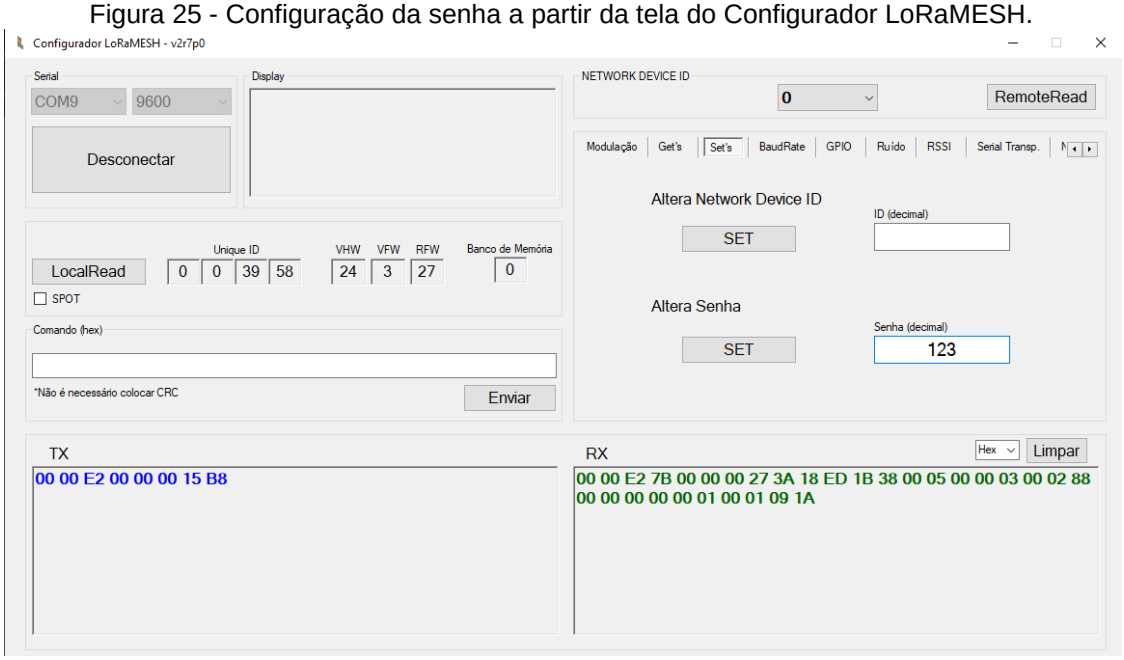
Fonte: Do autor

O *Unique* ID, corresponde ao ID de fábrica do dispositivo, cada módulo tem o seu único identificador que é a identidade do dispositivo gravada no microcontrolador, os parâmetros VHW, VFW e RFW correspondem as versões do *hardware* e do

firmware. Cada configuração feita, gera um código correspondente em hexadecimal nas abas correspondentes a comunicação serial, pode-se ver isso nas abas TX e RX do *software*.

Em seguida, deve-se configurar a senha e o *network device* ID na janela mostrada na Figura 25, a senha deve ser igual entre todos os módulos que compõe a rede, já para o *device* ID, o zero é reservado para o dispositivo mestre, somente ele pode assumir este valor, sendo que os escravos devem assumir valores a partir do um.

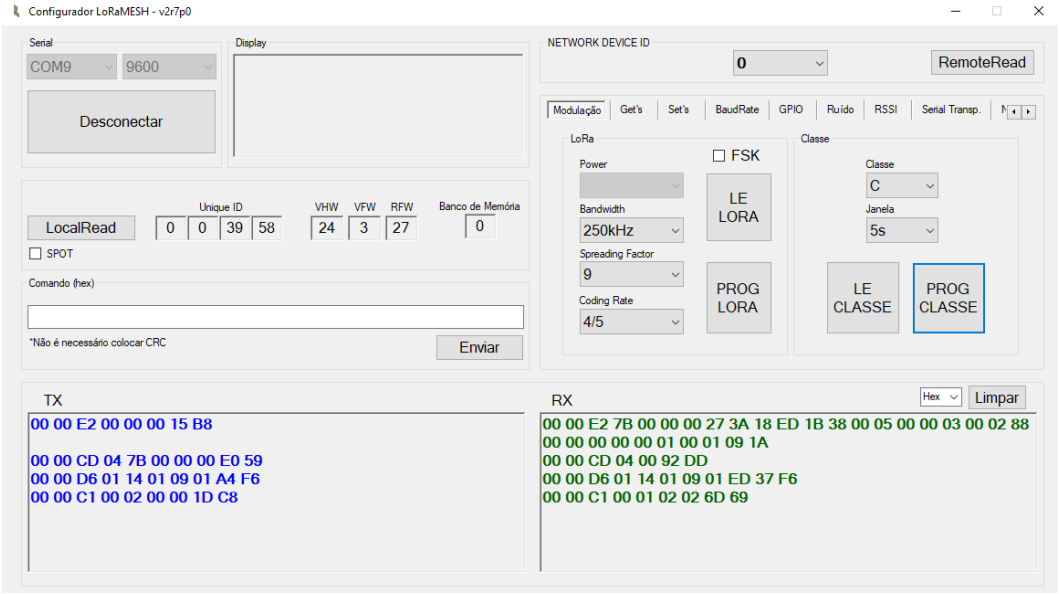
Figura 25 - Configuração da senha a partir da tela do Configurador LoRaMESH.



Fonte: Do autor

Depois da senha configurada, temos que configurar a rede LoRa, para isso a opção FSK deve estar desmarcada por se tratar de um outro tipo de modulação, logo após os parâmetros da rede devem ser selecionados de acordo com a necessidade de aplicação, e também os parâmetros de classe do dispositivo, conforme a Figura 26. Os parâmetros definidos anteriormente não devem ser esquecidos pois serão iguais em todos dispositivos participantes da rede.

Figura 26 - Configuração dos parâmetros LoRa a partir da tela do configurador LoRaMESH.

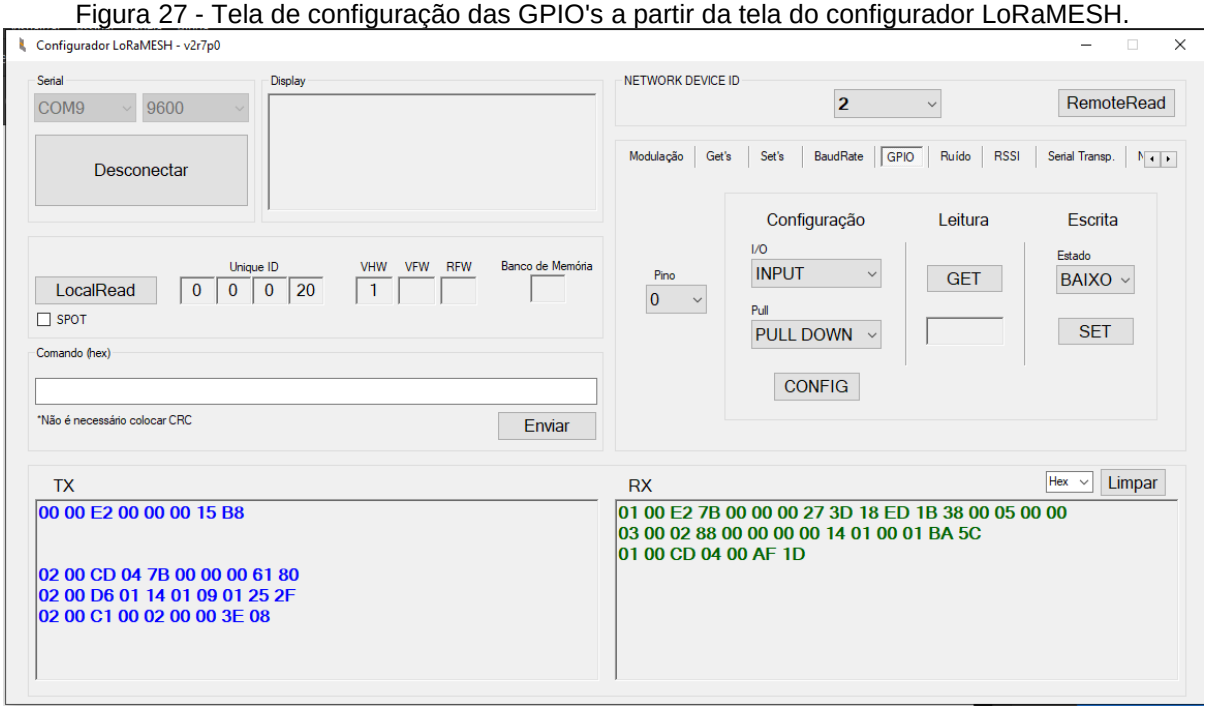


Fonte: Do autor

A configuração do rádio escravo não é muito diferente que a do mestre, basta apenas modificar o *network device ID*, como mencionado anteriormente, devendo assumir um valor diferente do zero.

Os módulos escravos serão os responsáveis por receberem os sensores e atuadores presentes em campo, no entanto, será necessário configurar suas GPIO's de acordo com o tipo de dispositivo que será conectado em suas portas, sendo entrada ou saída, analógico ou digital, e se vai ser ou não utilizado *pull down* ou *pull up* pode-se acompanhar esta configuração na Figura 27.

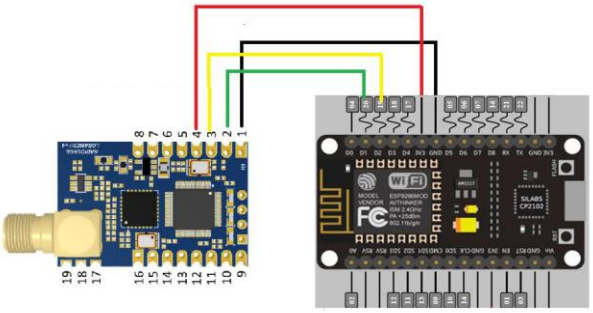
Figura 27 - Tela de configuração das GPIO's a partir da tela do configurador LoRaMESH.



Fonte: Do autor

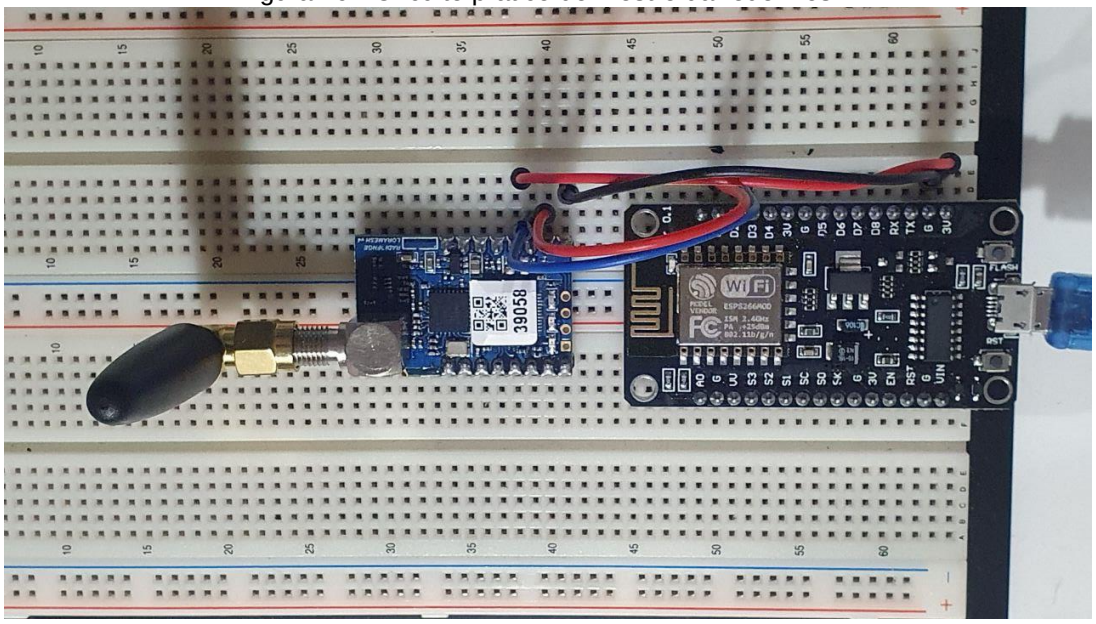
Com os módulos já configurados, foi utilizado um NodeMCU conectado ao módulo configurado como mestre, para fazer a comunicação com os escravos e gerenciar a rede, na Figura 28 têm-se o circuito e na Figura 29 é mostrado o protótipo gerado para teste, o código gravado no NodeMCU se encontra no ANEXO B - Código utilizado na criação do dispositivo mestre da rede mesh. Este programa é projetado para enviar e receber dados de um dispositivo mestre em uma rede LoRaMESH, por meio da comunicação com módulos escravos.

Figura 28 - Conexão do módulo ao Node MCU.



Fonte: (LoRaMESH & NodeMCU, s.d., p. 3)

Figura 29 - Circuito prático do mestre da rede *mesh*.



Fonte: Do autor

- **Passo 1 - Inclusão de bibliotecas**

  O código inclui as seguintes bibliotecas:
  Ticker: Essa biblioteca é utilizada para criar temporizadores (timers) de forma simples, permitindo a execução de funções em intervalos de tempo específicos.
  SoftwareSerial: Essa biblioteca permite a criação de uma porta serial em pinos digitais, deixando livre a porta serial padrão para comunicação com o monitor serial.

- **Passo 2 – Declaração de variáveis**

  As variáveis declaradas têm as seguintes finalidades:
  SWSerial: Cria uma instância de SoftwareSerial para comunicação com dispositivos escravos.
  count: Um contador que é incrementado para controle de tempo e eventos.
  inData: Um buffer de caracteres para armazenar dados recebidos.
  inChar: Variável que armazena um caractere lido da porta serial.
  indice: Um índice que permite percorrer o buffer de dados.
  listaEndDevices: Um array que contém os IDs dos dispositivos finais que podem ser acessados.

polingID: Utilizado para iterar pelos IDs no array listaEndDevices.
flag: Usada para sinalizar a recepção de dados.
comandos: Controla o ciclo dos comandos enviados.
listaGPIO: Contém os números das GPIOs a serem lidas.

- **Passo 3 – Função de interrupção do timer (onTimerISR)**

  Esta função é chamada quando o temporizador "estoura". A função alterna o estado do LED embutido (liga ou desliga). Define o temporizador para ser acionado novamente em 600 microsegundos. Incrementa o contador que será usado em outras partes do código.

- **Passo 4 – Configuração inicial setup**

  Nesta função, as configurações iniciais do microcontrolador são realizadas. Inicializa a comunicação serial padrão e por *software* a 9600 bps. Configura o pino do LED embutido como saída. Inicializa e configura o temporizador que será usado para alternar o LED e incrementar o contador.

- **Passo 5 – Função principal loop**

  O loop principal é executado continuamente. A cada 30 segundos (250 iterações de 120ms), executa um comando:
  Envia um comando para ler o nível de sinal RSSI do dispositivo; lê o estado das GPIOs; move para o próximo ID após enviar comandos para o dispositivo atual; verifica se há novos dados recebidos via comunicação serial e se dados foram recebidos, chama a função para tratar os dados.

- **Passo 6 – Leitura das portas seriais**

  A função verifica se há dados disponíveis na porta serial por *software*. Continua lendo enquanto houver dados disponíveis. Os dados lidos são armazenados no buffer inData. A função retorna 1 se dados foram recebidos e 0 caso contrário.

- **Passo 7 – Processamento de dados recebidos**

  A função exibe os dados recebidos no console e verifica se eles são válidos. Converte os dois primeiros bytes do buffer em um ID. Dependendo do comando recebido, processa os dados e exibe informações relevantes, como o nível de sinal.

- **Passo 8 – Leitura e escrita de GPIO**

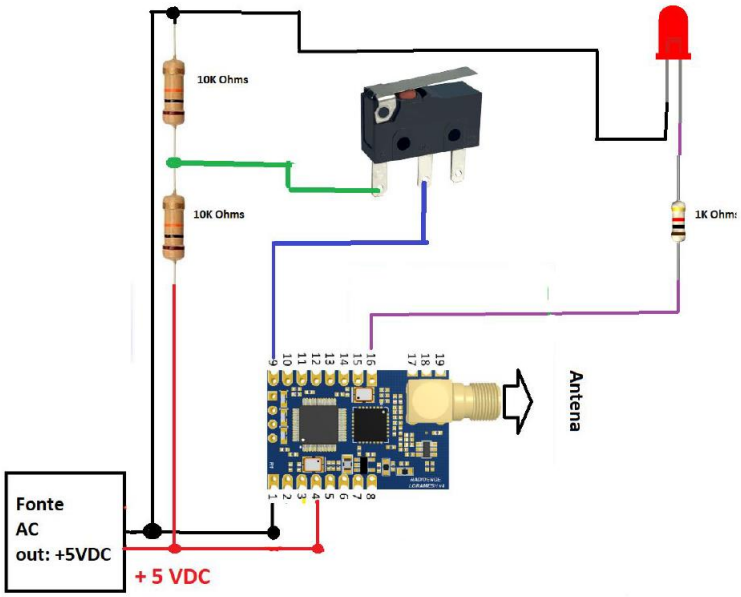
  As funções LerGPIO e SetGPIO são responsáveis por enviar comandos para ler e controlar as GPIOs do dispositivo. Ambas as funções seguem uma lógica semelhante, separando o ID em bytes e montando um pacote com o comando e parâmetros necessários antes de enviar via SWSerial.

- **Passo 9- Cálculo de RSSI**

  O cálculo da intensidade do sinal recebido RSSI é tratado dentro da função LerNivelRx, que envolve o envio de um comando ao dispositivo e o processamento da resposta.

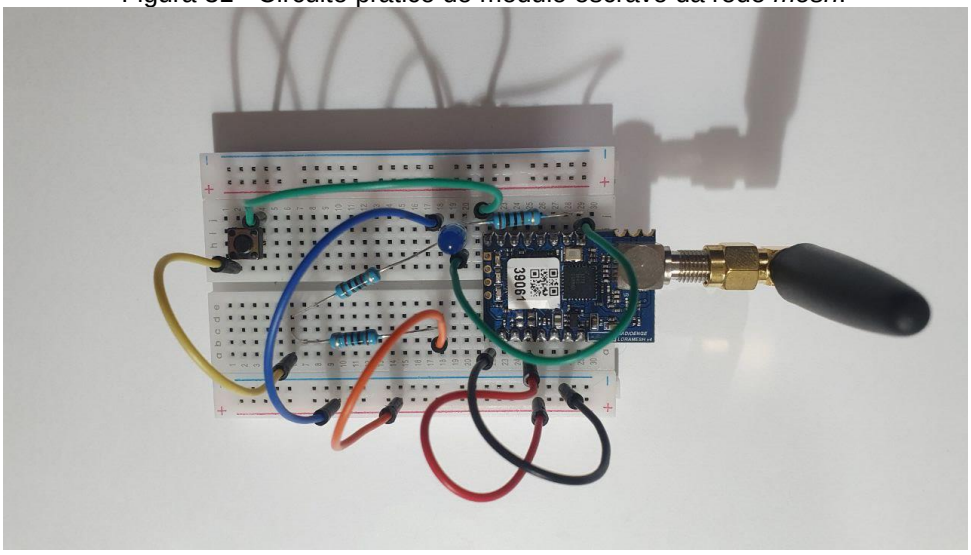
Um segundo módulo foi ligado a um led e um botão para testar a comunicação da rede *mesh* tanto na transmissão como na recepção, o circuito é mostrado na Figura 30 e na Figura 31 consta o protótipo gerado para testes.

Figura 30 - Ligação do módulo escravo.



Fonte: Adaptado de (LoRaMESH & NodeMCU, s.d., p. 5)

Figura 31 - Circuito prático do módulo escravo da rede *mesh*.



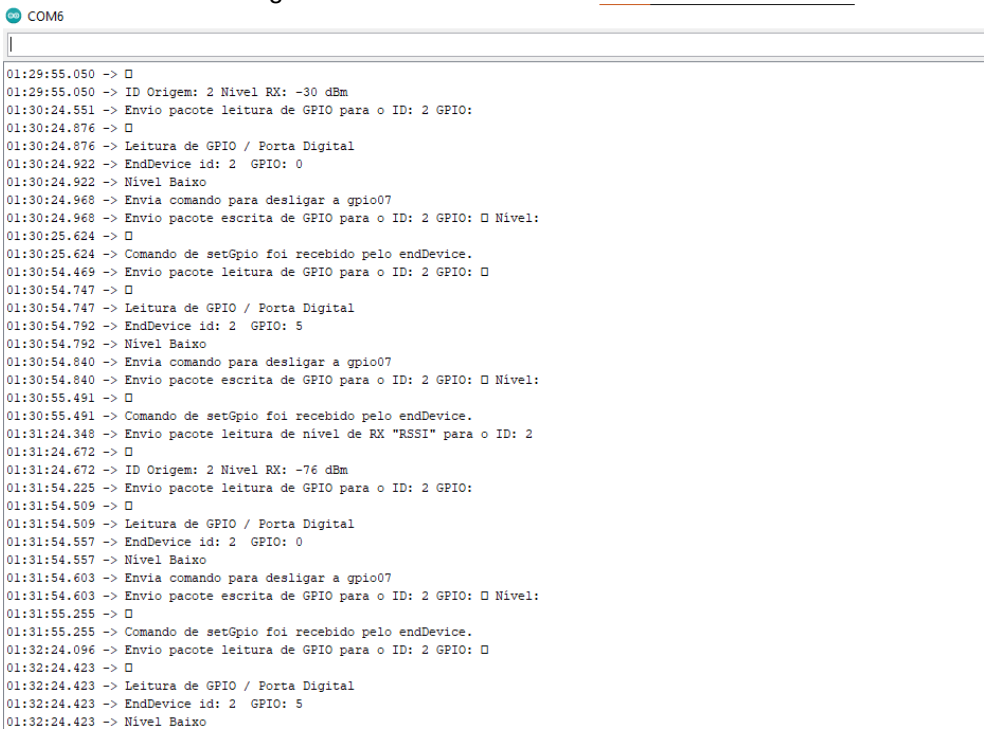
Fonte: Do autor

4.2 TESTES E RESULTADOS

O teste de funcionamento da rede *mesh* experimental se dá pelo acionamento do botão e pela verificação do led. Através do monitor serial do Arduino IDE mostrado na Figura 32, pode-se ver que o acionamento do botão gera um nível alto na GPIO 5,

já o led é aceso ou apagado quando é dado o comando via código de ligar ou desligar a GPIO 7. Quanto a intensidade do sinal é possível acompanhar através do nível de RX dado em dBm conforme pode-se notar no monitor serial obteve-se valores de -30 dBM e -76 dBm durante os testes realizados, o primeiro valor os módulos estavam mais próximos entre si cerca de 1m de distância, já para o segundo valor foi realizado um distanciamento dos mesmos usando outro cômodo mais distante em um apartamento de forma que as paredes comportassem como obstáculos apenas para verificar que a medição de sinal estava se comportando de acordo com a realidade.

Figura 32 - Monitor serial do Arduino IDE.



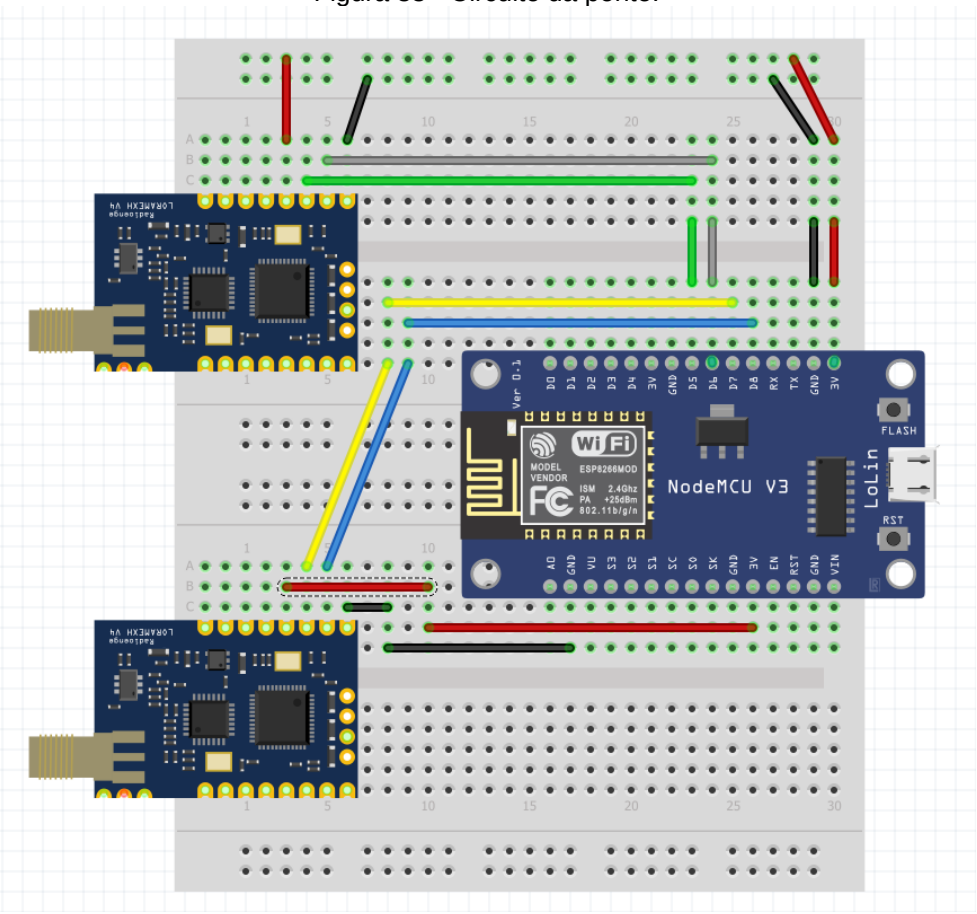
Fonte: Do autor

5 DESENVOLVIMENTO DA REDE HÍBRIDA LORAWAN/LORAMESH

5.1 MONTAGEM DO PROTÓTIPO

A confecção do protótipo se deu pela união de dois módulos Radioenge e um Esp8266 servindo de ponte entre eles. A Figura 33 mostra o esquema de montagem da ponte, onde o módulo na parte superior é o LoRaWAN e o da parte inferior é o LoRaMESH. Devido a impossibilidade de conectar um dispositivo do tipo *mesh* diretamente ao *gateway*, optou-se por estabelecer uma conexão LoRaWAN convencional, e interligado a ela como uma alternativa para contornar o problema. Esse protótipo serviu para coletar as informações vindas do circuito terminal via rede *mesh* e as encaminhar para o *gateway* utilizado no trabalho.

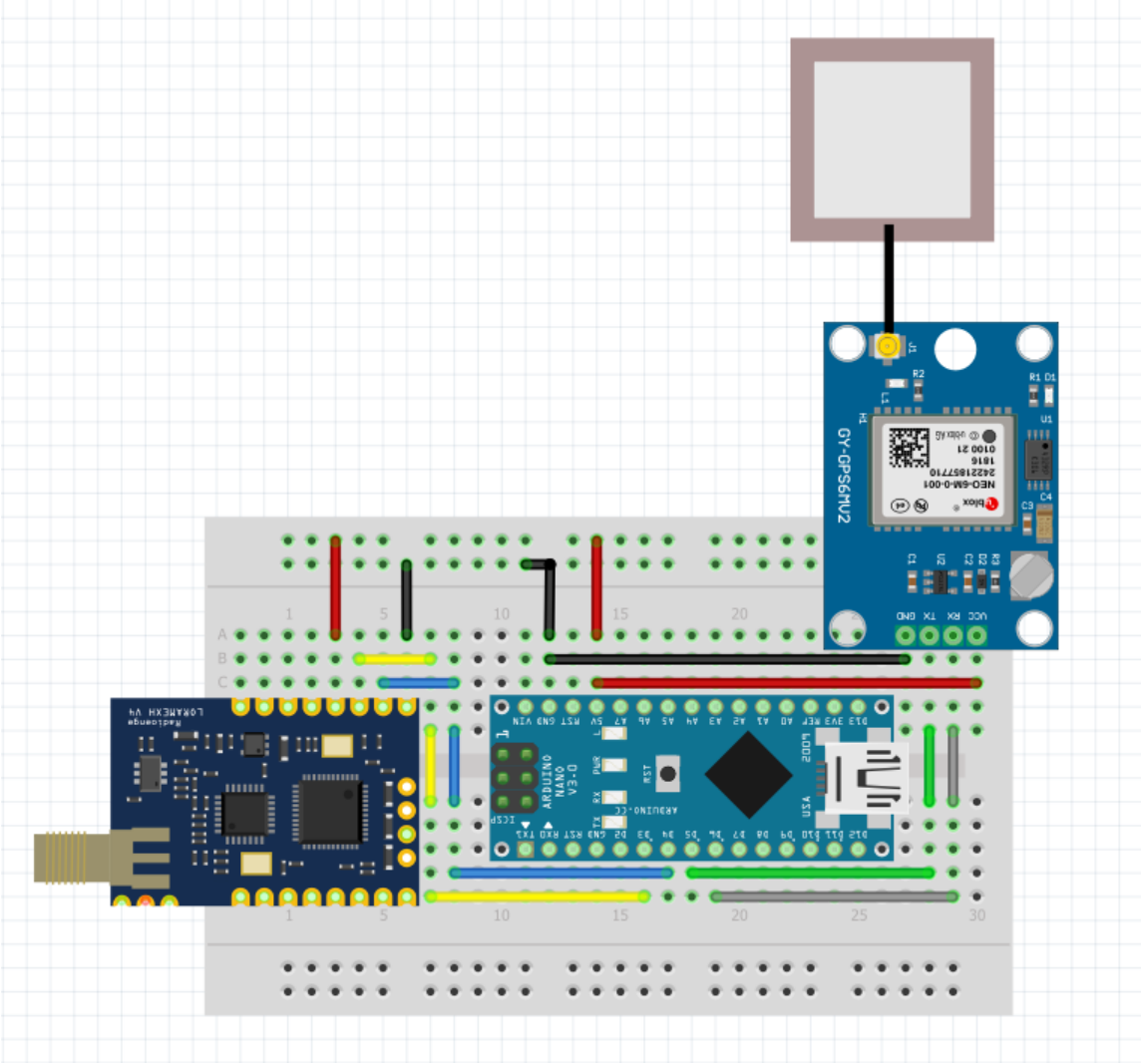
Figura 33 - Circuito da ponte.



Fonte: Do autor

Do outro lado foi criado um circuito contendo um arduíno nano, um módulo LoRaMESH e um módulo GPS como mostra a Figura 34. O objetivo deste circuito foi de coletar as coordenadas de posição do local onde ele estava localizado, para fazer o teste de alcance do sinal real do projeto. Neste caso o Arduino faz o tratamento dos dados e os lança na rede *mesh*, podendo assim se obter uma comunicação entre este circuito e o da Figura 33.

Figura 34 - Circuito terminal LoRaMESH.



Fonte: Do autor

5.2 FUNCIONAMENTO DO CÓDIGO

5.2.1 Circuito ponte LoRaWAN LoRaMESH

O código foi criado para ser executado em um ESP32 ou em um ESP8266, com intuito de fazer a ponte entre as redes LoRaWAN e LoRaMESH. Este código estabelece comunicação entre dispositivos em uma rede LoRaMESH, recebendo coordenadas geográficas e enviando-as para a rede LoRaWAN. A seguir tem-se a explicação de forma detalhada. Já o código se encontra no ANEXO C - Código utilizado no circuito ponte LoRaWAN/LoRaMESH.

- **Passo 1 - Inclusão de bibliotecas**

  O Primeiro passo do código é incluir bibliotecas, que são conjuntos de ferramentas com comandos e funções prontas para serem utilizadas:
  LoRaMESH: Fornece uma interface para comunicação em uma rede MESH usando LoRa.
  LoRaWAN_Radioenge: Permite comunicação via LoRaWAN.
  SoftwareSerial: Cria portas seriais adicionais em microcontroladores que possuem portas seriais limitadas.

- **Passo 2 - Configuração de depuração**

  Nesta etapa, é definido um modo para acompanhar a execução do programa, o DEBUG serve para indicar que o código deve mostrar mensagens detalhadas sobre o que está acontecendo, o que facilita encontrar possíveis erros.

- **Passo 3 - Configuração das conexões seriais**

  Aqui o código define duas conexões seriais, uma para o módulo LoRaWAN e outra para o módulo LoRaMESH.

- **Passo 4 - Inicialização dos módulos LoRaMESH e LoRaWAN**

  Dependendo do microcontrolador utilizado o código se adapta automaticamente. Sendo que será definida uma porta de comunicação serial independente para cada módulo LoRa conectado tanto no ESP 8266, como no ESP 32.

- **Passo 5 - Configuração inicial setup()**

  A função setup é executada uma única vez quando o dispositivo é ligado, realizando a comunicação com o computador para monitoramento do funcionamento do dispositivo.

- **Passo 6 - Inicialização da rede LoRaMESH**

  O código espera 10 segundos para garantir que todos os dispositivos estejam prontos, depois ele tenta inicializar a rede até obter um identificador válido para o dispositivo, no caso 0 (zero) por ser um dispositivo mestre. Em seguida o código executa a configuração dos demais parâmetros de rede como taxa de transmissão, classe do dispositivo e a senha da rede.

- **Passo 7 - Inicialização da rede LoRaWAN**

  O dispositivo é configurado para se conectar a uma rede LoRaWAN, inicializando a rede LoRaWAN, imprimindo seus parâmetros e se conectando usando o método ABP ao servidor do TTN (*The Things Network*).

- **Passo 8 – Coleta de dados de latitude e longitude**

  O dispositivo entra em um loop e fica aguardando o recebimento de coordenadas de latitude e longitude, armazenando-as nas variáveis slat e slng. Caso as coordenadas sejam recebidas de forma completa latitude (0x11) e longitude (0x13), os valores são extraídos e convertidos para string.

- **Passo 10 – Envio dos dados via LoRaWAN**

  Após recebimento, as coordenadas são concatenadas em uma única mensagem e convertidas em um formato compatível com o envio, para então serem enviadas através da rede LoRaWAN.

5.2.2 Circuito rastreador LoRaMESH

O código visa realizar a comunicação LoRaMesh para enviar as coordenadas geográficas (latitude e longitude) a partir de um GPS, convertendo os valores de ponto flutuante para bytes e enviando-os em quadros de comando via uma rede LoRa. Abaixo o código é explicado em detalhes, e o mesmo se encontra no ANEXO D - Código utilizado no circuito rastreador LoRaMESH.

- **Passo 1 - Inclusão de bibliotecas**

  O primeiro passo envolve a inclusão de bibliotecas que são necessárias para a funcionalidade do sistema:
  LoRaMESH: Biblioteca para gerenciar a comunicação via rede LoRaMESH.
  SoftwareSerial: Permite a criação de portas seriais adicionais.
  GPS: Biblioteca para trabalhar com dados de GPS.

- **Passo 2 - Definição da configuração e depuração de componentes**

  É realizado a configuração de depuração, e inicialização das portas de comunicação e a declaração de objetos necessários:
  DEBUG: Modo de depuração, usado para imprimir informações via Serial.
  gps: Instância da classe GPS, usada para lidar com os dados de localização.
  SerialCommand: Interface serial para comunicação com o módulo LoRaMESH.
  SerialGPS: Interface serial para o GPS.

LoRa: Inicializa a comunicação com o módulo LoRaMESH, usando a porta SerialComand
ID: Identificador único para o dispositivo na rede LoRa. Como se trata de um nó escravo, é configurado com valores diferente de zero.

- **Passo 3 - Função de depuração**

  É criada uma função para imprimir mensagens no monitor serial se o modo de debug estiver ativado, isto permite monitorar e diagnosticar a execução do programa.

- **Passo 4 - Configuração inicial setup()**

  A função setup realiza todas as configurações iniciais do dispositivo, iniciando a comunicação com o computador para depuração, e inicia a comunicação com o módulo LoRaMESH através dos pinos especificados.

- **Passo 5 - Inicialização e configuração do módulo LoRaMESH**

  O código aguarda por 10 segundos e depois tenta continuamente inicializar a comunicação MESH até obter um identificador único válido. Se o identificador do módulo for diferente de 1, é realizado sua reconfiguração, em seguida é ajustado os parâmetros de comunicação, a classe de operação e a senha.

- **Passo 6 – Configuração do GPS e envio de coordenadas**

  Inicia-se a comunicação com o módulo GPS e configuração da porta serial correta, em seguida, são iniciadas as funções de coleta de dados de localização, se foi coletado dados de latitude e longitude corretamente esses dados passam por uma conversão de float para uma sequência de bytes, desta forma é realizado o envio de um pacote com os comandos 0x11 e 0x13 para o dispositivo mestre da rede.

5.3 TESTES E RESULTADOS

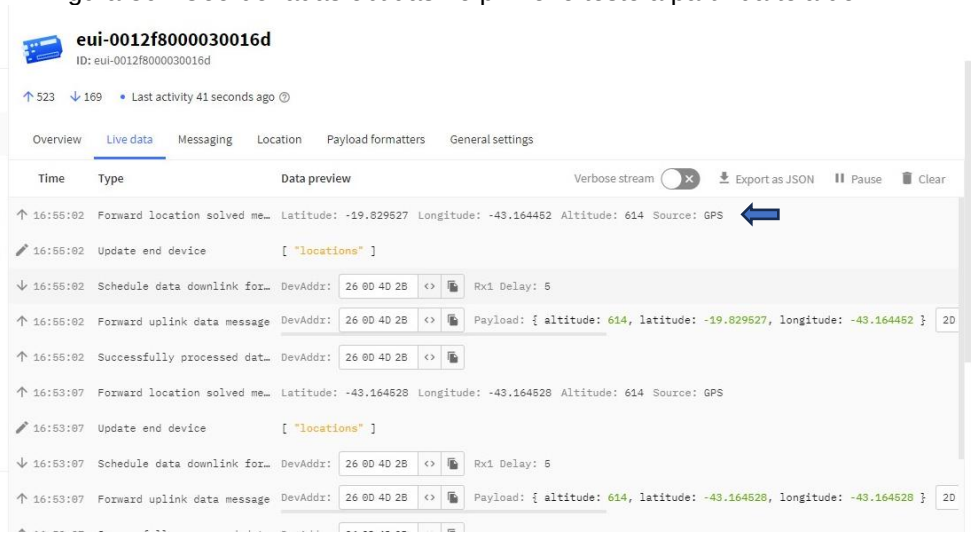
Para os testes primeiramente foi posicionado o circuito da ponte mostrado na Figura 33 nos dois locais de maior alcance coletados na tela do TTN Mapper mostrado na Figura 23 mantendo o circuito estático e verificando a conexão bem sucedida na tela do TTN, esse procedimento foi realizado em um ponto de cada vez. Em seguida se distanciou o circuito terminal deste ponto até se obter uma queda na conexão, a partir da queda se retornou ao ponto onde se obteve a última coordenada e manteve o dispositivo estático por ali durante um período afim de se coletar os dados de qualidade de sinal, como mostra a Figura 35, pode se verificar que para este módulo valores de RSSI próximos a -120dbm significa que se está próximo aos limites de comunicação entre os dispositivos.

Figura 35 - Parâmetros de conexão para o primeiro ponto testado.

Fonte: Do autor

A Figura 36 mostra as coordenadas obtidas no primeiro teste, a partir dela e do uso do Google Maps se obteve conforme mostrado na Figura 37 a posição da distância máxima obtida entre o circuito terminal e o *gateway.*

Figura 36 - Coordenadas obtidas no primeiro teste a partir da tela do TTN.



Fonte: Do autor

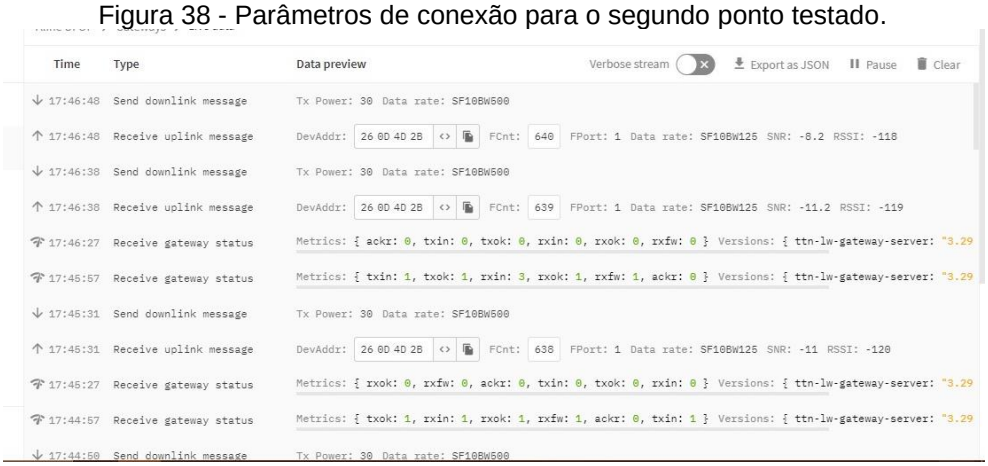
Figura 37 - Posição obtida durante o primeiro teste vista no Google Maps.

Fonte: Adaptado de João Monlevade (2024)

No segundo passo foram replicados os mesmos testes realizados na primeira parte, mudando apenas o local de referência para o segundo ponto mais distante

obtido na tela do TTN Mapper. Pode-se verificar na Figura 38 o valor de RSSI que também oscilou em valores próximos a -120dbm, indicando limite da qualidade de sinal.

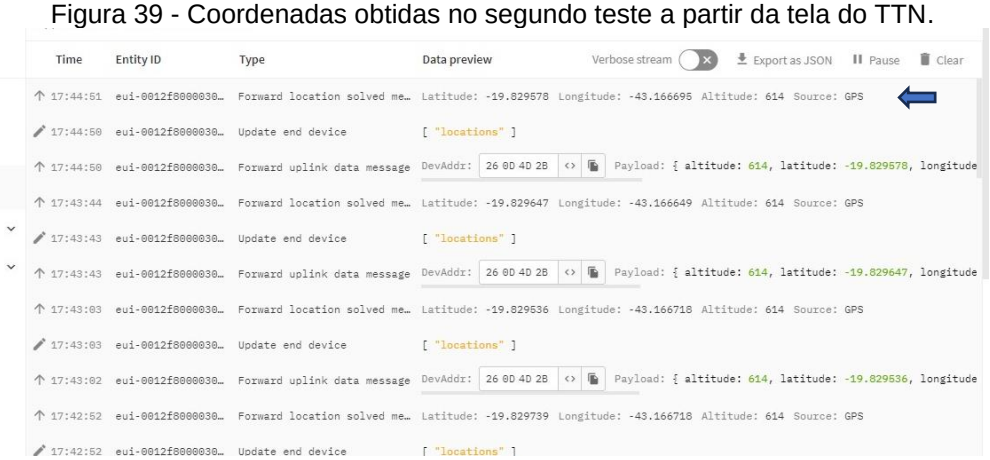
Figura 38 - Parâmetros de conexão para o segundo ponto testado.



Fonte: Do autor

A partir das coordenadas obtidas na Figura 39 se obteve o mapa de localização do ponto mais distante do *gateway* para a ocasião, através do Google Maps conforme mostrado na Figura 40.

Figura 39 - Coordenadas obtidas no segundo teste a partir da tela do TTN.



Fonte: Do autor

Figura 40 - Posição obtida durante o segundo teste vista no Google Maps.

Fonte: Adaptado de João Monlevade (2024)

6 CONCLUSÕES FINAIS

Através deste trabalho pode-se perceber que a distância, a presença de obstáculos, o tempo, e as interferências eletromagnéticas, interferem em muito a comunicação LoRa, esse tipo de fenômeno já era de se esperar devido à natureza de todo tipo de comunicação de radiofrequência.

É notória a redução da potência do sinal quando se afasta um dispositivo do outro, a colocação de barreiras entre os dispositivos também contribui para essa ocorrência. Nos testes realizados com a rede *mesh* experimental, pode-se ver uma alteração no valor do nível de RX quando o dispositivo foi levado para longe do mestre.

Nos testes realizados com a rede híbrida, pode-se notar um aumento significativo do alcance de transmissão, os dados obtidos através da análise dos mapas mostram que a relação de distância dobrou após a integração da rede tipo *mesh* trabalhando em conjunto com a rede LoRa. Este tipo de aplicação seria muito útil para corrigir a deficiência da topologia estrela do LoRaWAN, uma vez que em uma aplicação urbana por exemplo, teremos casos em que sensores deverão ser posicionados em locais obstruídos por vários obstáculos físicos, que impedem a boa qualidade de transmissão.

6.1 CONTRIBUIÇÕES E LIMITAÇÕES DO ESTUDO

Este estudo proporcionou uma análise do comportamento da transmissão LoRa, destacando uma de suas deficiências, a perda de comunicação ou enfraquecimento dela à medida que se aumenta a distância, no entanto a topologia estrela adotada no protocolo LoRaWAN seria ineficaz em algumas aplicações.

O trabalho contribuiu de forma significativa para cobrir sombras de sinais do LoRaWAN, sejam elas geradas por presença de obstáculos ou até mesmo pelas distâncias excessivas, proporcionando uma cobertura ampla sem demandar investimentos em múltiplos gateways, mantendo a eficiência energética e reduzindo os custos operacionais da rede. No campo acadêmico, o estudo contribui com dados

empíricos sobre as vantagens e desafios dessa arquitetura, ampliando a literatura sobre redes de baixo consumo.

O fabricante dos dispositivos utilizados neste trabalho afirma ter conseguido alcances de transmissão de até 20 Km, na aplicação se obteve distâncias relativamente menores, mas os dispositivos estavam localizados em locais com extrema presença de interferências, e não se sabe ao certo o tipo de antena que o fabricante utilizou durante os seus testes.

Na rede LoRaWAN, o TTN Mapper foi uma ferramenta de fundamental importância justamente por apresentar uma interface bem intuitiva com mapa e posicionamento dos dispositivos sobre ele. Para a rede *mesh* a aplicação não tem compatibilidade, no entanto, ficou faltando alguma ferramenta para medição de alcance de transmissão dos dispositivos, o único parâmetro que servia de referência é o nível do sinal em dBm apresentado no monitor serial do Arduino IDE, a partir dele e pela distância do dispositivo tem-se uma noção do alcance máximo do dispositivo em determinadas condições.

Como a rede *mesh* não se integra totalmente com o ambiente LoRaWAN, os mapas apresentados no TTN não foram possíveis visualizar a posição do circuito rastreador, neste caso para contornar o problema utilizou-se um módulo GPS o que mostrou com maior veracidade a posição do dispositivo terminal. Transmissões LoRa foram feitas para longas distâncias, como os testes foram realizados com os dispositivos em alturas relativamente baixas, não se pode extrair o máximo da qualidade de transmissão, pois a partir do momento que os dispositivos foram movimentados por uma pessoa de altura relativamente mediana, até mesmo os veículos e residências do local se tornaram obstáculos para a transmissão.

Como sugestões para trabalhos futuros, recomenda-se a inclusão de mais dispositivos na rede *mesh*, o que resultaria em uma malha ampliada. Além disso, seria pertinente desenvolver um servidor de aplicação que possibilitasse a apresentação dos resultados de maneira mais intuitiva na interface do usuário. Também é aconselhável a instalação do *gateway* em uma área externa, a fim de garantir uma comunicação sem interferências.

REFERÊNCIAS

ARDUINO.CC. Datasheet Arduino Nano. **Arduino Nano**, 11 Agosto 2023. Disponível em: https://docs.arduino.cc/static/c1c35001f493b10b99d0d1f5b1a3bb6b/A000005-datasheet.pdf. Acesso em: 13 ago. 2023.

BALLART, Daniel A. J. ESTUDIO DE LA COBERTURA DE LA MODULACIÓN LORA EN LA BANDA DE 915[MHz] en un ambiente urbano, Santiago de Chile, 2018. Disponível em: https://repositorio.uchile.cl/handle/2250/168163. Acesso em: 20 ago. 2023.

CARLOTO, Filipe G. Sistema de telegestão para iluminação pública usando comunicação LoraWAN, Santa Maria, 2020. 116. Disponível em: http://repositorio.ufsm.br/handle/1/22437. Acesso em: 17 jul. 2023.

FIALHO, Vitor M.; AZEVEDO, Fernando. Wireless Communication Based on Chirp Signals for LoRa IoT Devices, 4, Dezembro 2018. Disponível em: https://www.researchgate.net/publication/329913331_Wireless_Communication_Based_on_Chirp_Signals_for_LoRa_IoT_Devices. Acesso em: 20 jul. 2023.

JOÃO MONLEVADE. In: GOOGLE MAPS. Mountain View. 2024. Disponível em: https://www.google.com/maps/place/19%C2%B049'46.3%22S+43%C2%B009'52.0%22W/@-19.8332193,-43.1623424,16z/data=!4m4!3m3!8m2!3d-19.8295278!4d-43.1644444?entry=ttu&g_ep=EgoyMDI0MTAyOS4wIKXMDSoASAFQAw%3D%3D. . Acesso em: 11 set. 2024.

JOÃO MONLEVADE. In: GOOGLE MAPS. Mountain View. 2024. Disponível em: https://www.google.com/maps/place/19%C2%B049'46.5%22S+43%C2%B010'00.1%22W/@-19.8321418,-43.1650255,17.25z/data=!4m4!3m3!8m2!3d-19.8295833!4d-43.1666944?entry=ttu&g_ep=EgoyMDI0MTAyOS4wIKXMDSoASAFQAw%3D%3D. Acesso em: 11 set. 2024.

KIRICHEK, Ruslan et al. Analytic model of a mesh topology based on lora technology. 22nd International Conference on Advanced Communication Technology (ICACT), 2020. 251-255.

LORA® ALLIANCE TECHNICAL COMMITTEE. LoRaWAN® 1.1 Specification. **Lora Alliance®**, Outubro 2017. Disponível em: https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1 Acesso em: 08 out. 2024.

LORA® ALLIANCE TECHNICAL MARKETING WORKGROUP. What is LoRaWAN®. **Lora Alliance®**, Novembro 2015. Disponível em: https://hz1.37b.myftpupload.com/resource_hub/what-is-lorawan/. Acesso em: 20 jul. 2023.

MAZIERO, Lucas. Projeto de um nó LoRaWAN híbrido com repetidor embutido de baixo custo e baixo consumo de energia, Santa Maria, 23 Setembro 2020. Disponível em: http://repositorio.ufsm.br/handle/1/22635. Acesso em: 20 ago. 2023.

RADIOENGE. LoRaMESH. Repositório GitHub, 2023. Disponível em: https://github.com/Radioenge/LoRaMESH. Acesso em: 15 jul. 2023.

RADIOENGE. LoRaWAN. Repositório GitHub, 2023. Disponível em: https://github.com/Radioenge/LoRaWAN. Acesso em: 15 jul. 2023.

RADIOENGE. Manual de utilização módulo LoraMESH. Radioenge, Junho 2022. Disponível em: https://www.radioenge.com.br/storage/2021/08/Manual_LoRaMESH_Jun2022.pdf. Acesso em: 15 jul. 2023.

RADIOENGE. Manual Técnico Gateway LoRaWAN, Maio 2022. Disponível em: https://www.radioenge.com.br/storage/2021/08/Manual_GW_LoRaWAN_Maio2022-1.pdf. Acesso em: 10 ago. 2023.

RADIOENGE. Manual de operação end device LoraWAN. **Radioenge**, Janeiro 2023. Disponível em: https://www.radioenge.com.br/storage/2021/08/Manual_LoRaWAN_Jan2023.pdf. Acesso em: 15 jul. 2023.

RADIOENGE. Gateway Lorawan. **Radioenge**. Disponível em: https://www.radioenge.com.br/storage/downloads-produtos/gateway-lorawan/tutorial-ttn.pdf. Acesso em: 20 jul. 2023.

RADIOENGE. LoRaMESH & NodeMCU. Disponível em: https://www.radioenge.com.br/storage/downloads-produtos/modulo-loramesh/loramesh-esp8266.zip. Acesso em: 10 ago. 2023.

SALGADO, Nuno C. Sistema de controle de iluminação pública utilizando comunicações LoRaWAN, Lisboa, 10 Dezembro 2019. Disponível em: https://repositorio.iscte-iul.pt/handle/10071/20240. Acesso em: 14 ago. 2023.

SEMTECH. Datasheet SX1272/73. **Semtech Corporation**, Janeiro 2019. Disponível em: https://www.semtech.com/products/wireless-rf/lora-connect/sx1272#documentation. Acesso em: 17 jul. 2023.

SISINNI, Emiliano *et al.* Enhanced flexible LoRaWAN node for industrial IoT. **2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)**, 2018. 1-4. THE THINGS NETWORK. LoraWAN®. Disponível em: https://www.thethingsnetwork.org/. Acesso em: 21 jul. 2023.

ANEXO A – Código do disparador de sinais para arduino

```
#include <LoRaWAN.h>

/* Includes ---------------------- */
#include <SoftwareSerial.h>
#include <stdint.h>

/* SoftwareSerial handles */
SoftwareSerial* hSerialCommands = NULL;
char DADDR[] = "01:C6:13:43";
char APPSKEY[] = "54:91:8A:8F:B0:BD:45:9B:CB:A8:C1:67:A8:55:FD:F3";
char NWKSKEY[] = "C0:F9:3F:A0:ED:B6:4B:BC:D8:31:16:97:85:6E:A2:34";
const char *devAddr = "01:C6:13:43";

char str_counter[32];
int counter = 0;

void setup() {
  Serial.begin(9600); /* Initialize monitor serial */
  Serial.println("Initializing...");
  delay(5000);

  /* Initialize SoftwareSerial */
  //portas do arduino para comunic e tempo de delay
  hSerialCommands = SerialCommandsInit(7, 6, 9600);

  SendAtCommand(AT_NJM, AtSet, "0");// comando para alterar a autenticação
para ABP
  /*SendAtCommand(AT_APPSKEY, AtSet, APPSKEY); envia comando para registra
appkey
  SendAtCommand(AT_NWKSKEY, AtSet, NWKSKEY);  envia comando para registra
nwskey
  SendAtCommand(AT_DEVaDDR, AtSet, DADDR); */
}
void loop() {
  /* Sends a string containing a counter every 15s */
  sprintf(str_counter, "Counter: %d\r\n\0", counter++);
  Serial.println(str_counter);
  SendString(str_counter, 2);
  delay(900);
  }
```

ANEXO B - Código utilizado na criação do dispositivo mestre da rede mesh

```
// Este programa visa coletar dados vindo de um lora mesh mestre, através de sua comunicação com um módulo escravo
#include <Ticker.h>
#include<SoftwareSerial.h>
//========SoftwareSerial=================
SoftwareSerial SWSerial(4, 5);  //D2, D1 : Será usada a serial por software para deixar livre a serial do monitor serial
//===============Timer====================
int count = 0;
char inData[255]; // Buffer com tamanho suficiente para receber qualquer mensagem
char inChar=-1; // char para receber um caractere lido
byte indice = 0; // índice para percorrer o vetor de char = Buffer
int listaEndDevices[5] = {2,2,2,2,2};//pode aumentar o tamanho do arry e incluir um novo ID. coloquei 1,2,1,2,1 ... porque no teste só temos os IDs 1 e 2 ... em uma rede maior ... deve-se cololcar os IDs existentes 1,2,3,50,517,714,... qualquer inteiro de 1 até 1023.
int polingID=0;//usado para escolher qual ID do vetor acima será usado.
char flag= false; //flag usada para avisar que a serial recebeu dados
char comandos = 0;//usado para escolher qual comando enviar.
char listaGPIO[]={0,5}; // usado para ler as GPIOs que estão neste vetor ... pode-se ler da 0 até 7

//=========================================================================
//Timer : Quando este timer "estourar" ... o led mudará seu estado e será incrementado o tempo usado no polling.
void ICACHE_RAM_ATTR onTimerISR(){
    digitalWrite(LED_BUILTIN,!(digitalRead(LED_BUILTIN)));  //Toggle LED Pin
    timer1_write(600000);//12us
    count++;
}

//=========================================================================
// ====================== protótipos das funções  ================================
char lerSerial();
char TrataRX(byte indice);
void escreve(int contador);
char LerGPIO(int id, char gpio);
char SetGPIO(int id, char gpio, char nivel);
uint16_t CalculaCRC(char* data_in, uint32_t length);
```

```
//=============================================================================
====

void setup() {

  Serial.begin(9600);
  SWSerial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT); // Initialize the LED_BUILTIN pin as an output
  //Initialize Ticker every 0.5s
  timer1_attachInterrupt(onTimerISR);
  timer1_enable(TIM_DIV16, TIM_EDGE, TIM_SINGLE);
  timer1_write(600000); //120 ms altera o tempo do timer para 120ms

}

// the loop function runs over and over again forever
void loop() {
  if(count % 250 == 0)//30 : Se o contador que é incrementado no timer chegar
a 30 segundos (120ms * 250=30) ... envia um comando para um escravo. o comando
será RSSI, depois Leitura da porta analógica 5 e por últio
  {
        count++;
        if((comandos%3) ==0)// %3 porque são 3 comandos diferente para cada
EndDevice / Radio. Podem ser incluídos novos comandos. o Primeiro é a medida
de nível de sinal de recepção
        {
          LerNivelRx(listaEndDevices[polingID%5]);//%5 porque o array com IDs
de EndDevices a ser percorrido só tem 5 endereços... pode-se diminuir ou
aumentar ... depende do número de IDs/EndDevices existentes na rede

        }
        else if((comandos%3) == 1) // %3 porque são 3 comandos diferente para
cada EndDevice / Radio. Podem ser incluídos novos comandos
        {
          LerGPIO(listaEndDevices[polingID%5], listaGPIO[0]);//%5 porque o
array com IDs de EndDevices a ser percorrido só tem 5 endereços

        }
        else if((comandos%3) == 2) //podem ser incluídos novos comandos
        {
         LerGPIO(listaEndDevices[polingID%5], listaGPIO[1]);//%5 porque o
array com IDs de EndDevices a ser percorrido só tem 5 endereços ... se mudar o
array listaEndDevices[] também mude o valor desta divisão por módulo.
          polingID++;//Enviou os 3 comandos diferentes para um ID... então ...
vai para o próximo ID/Rádio/EndDevice
        }
        comandos++;//próximo comando
  }
```

```
    flag= LerSerial(); // verifica se a serial por software recebeu alguma
informação. Ser quiser usar a serial nativa para receber dados...  tem que
implementar uma nova função.
    if (flag)//pode-se colocar o LerSerial() dentro do if, mas fica mais claro
de mostrar usando a flag.
    {
      TrataRX(indice);//chama a função que vai interpretar o que foi recebido
na serial por software
    }

}

//========================== função para vericar se recebeu alguma informação
na serial por software pinos D1 e D2 ==================
char LerSerial( )
{
    indice=0;
    while (SWSerial.available() > 0) // Don't read unless// there you know
there is data
    {
        if(indice < 254) // One less than the size of the array
        {
            inChar = SWSerial.read(); // Read a character
            inData[indice] = inChar; // Store it
            delay (10);
            indice++; // Increment where to write next
            inData[indice] = '\0'; // Null terminate the string
        }
    }
    if(indice>0)
    {
        return(1); //retorno da função avisando que recebeu dados na serial
    }
    else
    {
      return (0);
    }

}
//-------------------------------------------------------------------------
-//

//================== tratamento do que recebeu na serial
========================
char TrataRX(byte indice) //IMPORATANTE: O terceiro byte [2] é sempre o
comando... sabendo o comando sabemos como tratar o restante do pacote
{
      Serial.println(inData); //mostra no console serial
```

```
      if(indice>5)//verifica se recebeu uma reposta de comando maior que 5
bytes para não ler dados inválidos
      {
        int id=inData[0]+inData[1]*256;//id msb*256+id lsb converte os dois
primeiros bytes em um int que é o id do EndDevice de origem
        if(inData[2] == 0xD5)//0xD5 ou 213 decimal é o comando de leitura de
nível de RX byte3=máximo byte 4 = médio byte 5=nmínimo
        {
          char sinal = inData[6]; //valor médio do nível de RX do EndDevice no
sétimo byte do array
          char buf[35];//buffer para montar uma string a ser envida na serial
do monitor
          sprintf(buf,"ID Origem: %d Nivel RX: -%d dBm",id, sinal);  //format
two ints into character array
          Serial.println(buf);
        }
        if(inData[2] == 0xC2 )//resposta de comando na GPIO
        {
          if(inData[4] == 0x00 && inData[3] == 0x00)//[4]== 0: resposta ok.
sem erro! [3] == 0: resposta de leitura
          {

            if(inData[6] & 0x80)//verifica se o primeiro bit do byte 6 é 1...
se for 1.. o pacote contém a resposta de uma porta digitial então o próximo
byte pode ser zero ou um
            {
              Serial.println("Leitura de GPIO / Porta Digital"); //imprime na
serial do monitor ... somente para acompanhar o que está acontecendo.
              Serial.print("EndDevice id: ");//imprime na serial do monitor .
              Serial.print(id);//imprime na serial do monitor .
              Serial.print("  GPIO: ");//imprime na serial do monitor .
              Serial.println((int)inData[5]);//qual GPIO de 0 a 7//imprime na
serial do monitor .
              char infoTTS[15] = {0};
              char valor = '0';
              if(inData[7] == 1)
              {
                Serial.println("Nível Alto");//imprime na serial do monitor
                valor = '1';
                Serial.println("Envia comando para ligar a gpio07");//imprime
na serial do monitor
                SetGPIO(id, 0x07, 0x01); //IMPORTANTE: Aqui temos uma ação...
Se a entrada digital estiver em nível alto então... é chamada a função SetGPIO
que por sua vez envia o comando para ligar a o gpio 7 e acinar o Relé ...
              }
              else
              {
                Serial.println("Nível Baixo");
                valor = '0';
```

```
                Serial.println("Envia comando para desligar a gpio07");
                SetGPIO(id, 0x07, 0x00); //IMPORTANTE: Aqui temos uma ação...
Se a entrada digital estiver em baixo alto então... é chamada a função SetGPIO
que por sua vez envia o comando para desligar a o gpio 7 e acinar o Relé ...
              }

                if(id == 2)
                {
                  sprintf(infoTTS,"field4=%c", valor );
                }
            }
            else //resposta de leitura de uma porta analógica
            {
              Serial.println("Leitura de GPIO / Porta Analógica");//imprime na
serial do monitor
              Serial.print("EndDevice id: ");
              Serial.print(id);
              Serial.print("  GPIO: ");
              Serial.print((int)inData[5]);//qual GPIO de 0 a 7
              int valorAD=inData[6]*256+inData[7];
              Serial.print("Leitura em BITs: ");
              Serial.println(valorAD);
              Serial.print("Leitura em Volts: ");
              Serial.println((3.3/4096)*valorAD);
              char infoTTS[15] = {0};
              if(id == 1)
                {
                  sprintf(infoTTS,"field1=%.2f", ((3.3/4096)*valorAD) );
                }
              else if(id == 2)
                {
                  sprintf(infoTTS,"field3=%.2f", ((3.3/4096)*valorAD) );
                }
            }
          }
          else if(inData[4] == 0x01)
          {
            Serial.println("Erro na leitura ou tentativa de ler uma porta
configurada como saída.");
          }
          else if(inData[3] == 0x01)
          {
            Serial.println("Comando de setGpio foi recebido pelo endDevice.");
          }

        }
      }
return indice;
}
```

```
//=====================================================
void escreve(int contador)
{
   Serial.println(count);
}
//=====================================================

//===================cálculo do CRC16==================
/**
*@brief Calcula CRC16.
*@param data_in: Ponteiro para o buffer contendo os dados.
*@param length: Tamanho do buffer
*@retval Valor de 16 bits representando o CRC16 do buffer
fornecido. */
#define CRC_POLY (0xA001)
uint16_t CalculaCRC(char* data_in, uint32_t length)
{
uint32_t i;
uint8_t bitbang, j;
uint16_t crc_calc;
crc_calc = 0xC181;
for(i=0; i<length; i++)
{
crc_calc ^= ((uint16_t)data_in[i]) & 0x00FF;
for(j=0; j<8; j++)
{
bitbang = crc_calc;
crc_calc >>= 1;
if(bitbang & 1)
{
crc_calc ^= CRC_POLY;
}
}
}
return crc_calc;
}
//=====================================================================

//===================  função para leitura no nível de RX  =================
char LerNivelRx(int id)
{
  char id_lsb = id&0xFF; //separando o ind em dois bytes para enviar no
pacote; &0xFF pega somente os 8 bits menos significativos
  char id_msb = (id>>8)&0xFF;//bitwise desloca os 8 bits mais signficativos e
pega somente a parte msb do int
  char comando = 0xD5;//comando de leitura do nível de recepção
  char pacote[]={id_lsb, id_msb, comando,0x00, 0x00, 0x00, 0x00,0x00};
  int crc = CalculaCRC(pacote, 6);
```

```
  pacote[6] = crc&0xFF;
  pacote[7]= ((crc>>8)&0xFF);
  SWSerial.write(pacote,8);//para enviar qualquer pacote para o EndDevice use
a função write!
  Serial.print("Envio pacote leitura de nível de RX \"RSSI\" para o ID:
");//somente para aparecer no monitor serial do Arduino
  Serial.println(id);
  return(1);
}

//=============================================================================

//======================  função para leitura das GPIO  ====================
char LerGPIO(int id, char gpio)
{
  char id_lsb = id&0xFF; //separando o ind em dois bytes para enviar no
pacote; &0xFF pega somente os 8 bits menos significativos
  char id_msb = (id>>8)&0xFF;//bitwise desloca os 8 bits mais signficativos e
pega somente a parte msb do int
  char comando = 0xC2; //comando referente a GPIO
  char sub_cmd_leitura_gpio = 0x00; // subcomando de leitura da GPIO
  char pacote[]={id_lsb, id_msb, comando, sub_cmd_leitura_gpio, gpio, 0x00,
0x00,0x00};
  int crc = CalculaCRC(pacote, 6);
  pacote[6] = crc&0xFF;
  pacote[7]= ((crc>>8)&0xFF);
  SWSerial.write(pacote,8);
  Serial.print("Envio pacote leitura de GPIO para o ID: ");//somente para
aparecer no monitor serial do Arduino
  Serial.print(id);
  Serial.print(" GPIO: ");
  Serial.println(gpio);
  return(1);
}

//=============================================================================

//======================  função para Acionamento da
GPIO  ====================
char SetGPIO(int id, char gpio, char nivel) //02 00 C2 01 07 01 00 6A 65  //02
00 C2 01 07 00 00 6B F5
{
  char id_lsb = id&0xFF; //separando o ind em dois bytes para enviar no
pacote; &0xFF pega somente os 8 bits menos significativos
  char id_msb = (id>>8)&0xFF;//bitwise desloca os 8 bits mais signficativos e
pega somente a parte msb do int
  char comando = 0xC2; // comando referente a GPIO
  char sub_cmd_escrita_gpio = 0x01; //comando de escrita/acionamento da GPIO
```

```
  char pacote[]={id_lsb, id_msb, comando, sub_cmd_escrita_gpio, gpio, nivel,
0x00,0x00};
  int crc = CalculaCRC(pacote, 6);
  pacote[6] = crc&0xFF;
  pacote[7]= ((crc>>8)&0xFF);
  SWSerial.write(pacote,8);
  Serial.print("Envio pacote escrita de GPIO para o ID: ");//somente para
aparecer no monitor serial do Arduino
  Serial.print(id);
  Serial.print(" GPIO: ");
  Serial.print(gpio);
  Serial.print(" Nível: ");
  Serial.println(nivel);
  return(1);
}
```

ANEXO C - Código utilizado no circuito ponte LoRaWAN/LoRaMESH

```
#include "LoRaMESH.h"
#include "LoRaWAN_Radioenge.h"
#include <SoftwareSerial.h>

#define DEBUG true

EspSoftwareSerial::UART SerialLoRaWAN;
EspSoftwareSerial::UART SerialLoRaMesh;

#ifdef ESP32
LoRaMESH LoRaMesh(&Serial2);
#elif ESP8266
LoRaMESH LoRaMesh(&SerialLoRaMesh);
#endif
LoRaWAN_Radioenge LoRaWAN(&SerialLoRaWAN);

uint8_t ID = 0;  // Como vai receber do slave, esse tem que ser master

void debug(String val) {
#ifdef DEBUG
  Serial.println(val);
#endif
}

void setup() {
  Serial.begin(9600);

#ifdef ESP32
  Serial2.begin(9600);
  SerialLoRaWAN.begin(9600, SWSERIAL_8N1, 22, 23, false);
#elif ESP8266
  SerialLoRaMesh.begin(9600, SWSERIAL_8N1, 13, 15, false);
  SerialLoRaWAN.begin(9600, SWSERIAL_8N1, 14, 12, false);
#endif

  delay(10000);
  for (;;) {
    LoRaMesh.begin(DEBUG);
    if (LoRaMesh.localUniqueId != 0)
      break;
    delay(random(1000, 5000));
  }

  if (LoRaMesh.localId != ID) {
```

```
    if (!LoRaMesh.setNetworkID(ID)) {
      debug("Erro ao definir o novo ID");
      while (1)
        ;
    }

    debug("ID configurado com sucesso!");

    if (!LoRaMesh.setBPS(BW500, SF7, CR4_5)) {
      debug("Erro ao configurar bps");
      while (1)
        ;
    }

    debug("Parametros LoRaMesh configurados com sucesso!");

    if (!LoRaMesh.setClass(CLASS_C, WINDOW_15s)) {
      debug("Erro ao configurar a classe");
      while (1)
        ;
    }

    debug("Modo de operacao configurado com sucesso!");

    if (!LoRaMesh.setPassword(123)) {
      debug("Erro ao gravar a senha ou a senha gravada não condiz com a senha
definida");
      while (1)
        ;
    }

    debug("Senha configurada com sucesso!");
  }

  debug("LocalID: " + String(LoRaMesh.localId));
  debug("UniqueID: " + String(LoRaMesh.localUniqueId));
  debug("Pass <= 65535: " + String(LoRaMesh.registered_password));

  LoRaWAN.begin(DEBUG);
  LoRaWAN.printParameters();
  LoRaWAN.JoinNetwork(ABP, TTN, true, false, "", "", "", "26:0D:4D:2B");
}

void loop() {
  String slat = "", slng = "";
  while (slat == "" || slng == "") {  // Fica no loop caso algum das duas
variáveis sejam nulas
    // Aguarda receber valor de lng caso não tenha recebido ainda
```

```
    if (LoRaMesh.receivePacketCommand(&LoRaMesh.localId, &LoRaMesh.command,
LoRaMesh.bufferPayload, &LoRaMesh.payloadSize, 1000)) {
      if (LoRaMesh.command == 0x11) {
        uint8_t byteData[4];                          // Cria buffer array de
tamanho 4
        byteData[0] = LoRaMesh.bufferPayload[0];
        byteData[1] = LoRaMesh.bufferPayload[1];
        byteData[2] = LoRaMesh.bufferPayload[2];
        byteData[3] = LoRaMesh.bufferPayload[3];

        float lat;
        memcpy(&lat, byteData, sizeof(float));        // Pega a posição da
memória de lat e adiciona todo o pacote do array na posição de memória lat
        slat = String(lat, 6);                        // Converte float em
String com 6 casas decimais

        debug("lat: " + String(slat));
      } else if (LoRaMesh.command == 0x13) {
        uint8_t byteData[4];
        byteData[0] = LoRaMesh.bufferPayload[0];
        byteData[1] = LoRaMesh.bufferPayload[1];
        byteData[2] = LoRaMesh.bufferPayload[2];
        byteData[3] = LoRaMesh.bufferPayload[3];

        float lng;
        memcpy(&lng, byteData, sizeof(float));
        slng = String(lng, 6);

        debug("lng: " + String(slng));
      }
    }
  }

  // Converte String para char array
  String packet = slat + ";" + slng;
  char charPacket[packet.length() + 1];
  packet.toCharArray(charPacket, packet.length() + 1);

  // Envia charPacket via LoRaWAN na porta 1
  LoRaWAN.SendString(charPacket, 1);
}
```

ANEXO D - Código utilizado no circuito rastreador LoRaMESH

```
#include "LoRaMESH.h"
#include <SoftwareSerial.h>
#include "GPS.h"

#define DEBUG true

GPS gps;
SoftwareSerial SerialCommand(3, 4);  //rx e tx
SoftwareSerial SerialGPS(5, 6);
LoRaMESH LoRa(&SerialCommand);

uint8_t ID = 1;  // Como esse é slave, tem que ser um número diferente de 0

void debug(String val) {
#ifdef DEBUG
  Serial.println(val);
#endif
}

void setup() {
#ifdef DEBUG
  Serial.begin(9600);
#endif
  SerialCommand.begin(9600);

  delay(10000);
  for (;;) {
    LoRa.begin(DEBUG);
    if(LoRa.localUniqueId != 0)
      break;
     delay(random(1000, 5000));
  }


  if (LoRa.localId != ID) {
    if (!LoRa.setNetworkID(ID)) {
      debug("Erro ao definir o novo ID");
      while (1)
        ;
    }

    debug("ID configurado com sucesso!");

    if (!LoRa.setBPS(BW500, SF7, CR4_5)) {
```

```
      debug("Erro ao configurar bps");
      while (1)
        ;
    }

    debug("Parametros LoRa configurados com sucesso!");

    if (!LoRa.setClass(CLASS_C, WINDOW_15s)) {
      debug("Erro ao configurar a classe");
      while (1)
        ;
    }

    debug("Modo de operacao configurado com sucesso!");

    if (!LoRa.setPassword(123)) {
      debug("Erro ao gravar a senha ou a senha gravada não condiz com a senha
definida");
      while (1)
        ;
    }

    debug("Senha configurada com sucesso!");
  }

  debug("LocalID: " + String(LoRa.localId));
  debug("UniqueID: " + String(LoRa.localUniqueId));
  debug("Pass <= 65535: " + String(LoRa.registered_password));

  SerialGPS.begin(9600);
  gps.begin(&SerialGPS, DEBUG);
}

void loop() {
  gps.getLocation();


  if (gps.lat() != 0 && gps.lng() != 0) {  // Entra na condicional apenas
quando o GPS pegar localização
    union {
      float f;
      uint8_t bytes[4];
    } lat;

    lat.f = gps.lat();                              // Joga float para union
e automaticamente converte para byte array
    LoRa.bufferPayload[0] = lat.bytes[0];                    // Joga
lat.bytes[0] para o buffer do LoRaMesh
    LoRa.bufferPayload[1] = lat.bytes[1];
```

```
    LoRa.bufferPayload[2] = lat.bytes[2];
    LoRa.bufferPayload[3] = lat.bytes[3];

    LoRa.prepareFrameCommand(0, 0x11, LoRa.bufferPayload, 4);  // Prepara
pacote e envia via LoRaMesh para o endereço 0
    delay(5000);

    union {
      float f;
      uint8_t bytes[4];
    } lng;

    lng.f = gps.lng();
    LoRa.bufferPayload[0] = lng.bytes[0];
    LoRa.bufferPayload[1] = lng.bytes[1];
    LoRa.bufferPayload[2] = lng.bytes[2];
    LoRa.bufferPayload[3] = lng.bytes[3];

    LoRa.prepareFrameCommand(0, 0x13, LoRa.bufferPayload, 4);
  }
}
```

Gps.h

```
#include <TinyGPSPlus.h>
#include "Stream.h"

class GPS {
  private:
    Stream *SerialGPS;
    TinyGPSPlus gps;
    bool debug = false;

  public:
    void begin(Stream *_SerialGPS, bool _debug) {
      SerialGPS = _SerialGPS;
      //SerialGPS->begin(9600);
      debug = _debug;
    }

    float lat(){
      return gps.location.lat();
    }

    float lng(){
      return gps.location.lng();
    }

    void getLocation() {
```

```
      for (uint16_t i = 0; i < 5000; ++i) {
        while (SerialGPS->available() > 0) {
          byte buff = SerialGPS->read();
          if (debug)
            Serial.write(buff);
          gps.encode(buff);
        }
        delay(1);
      }
    }
};
```