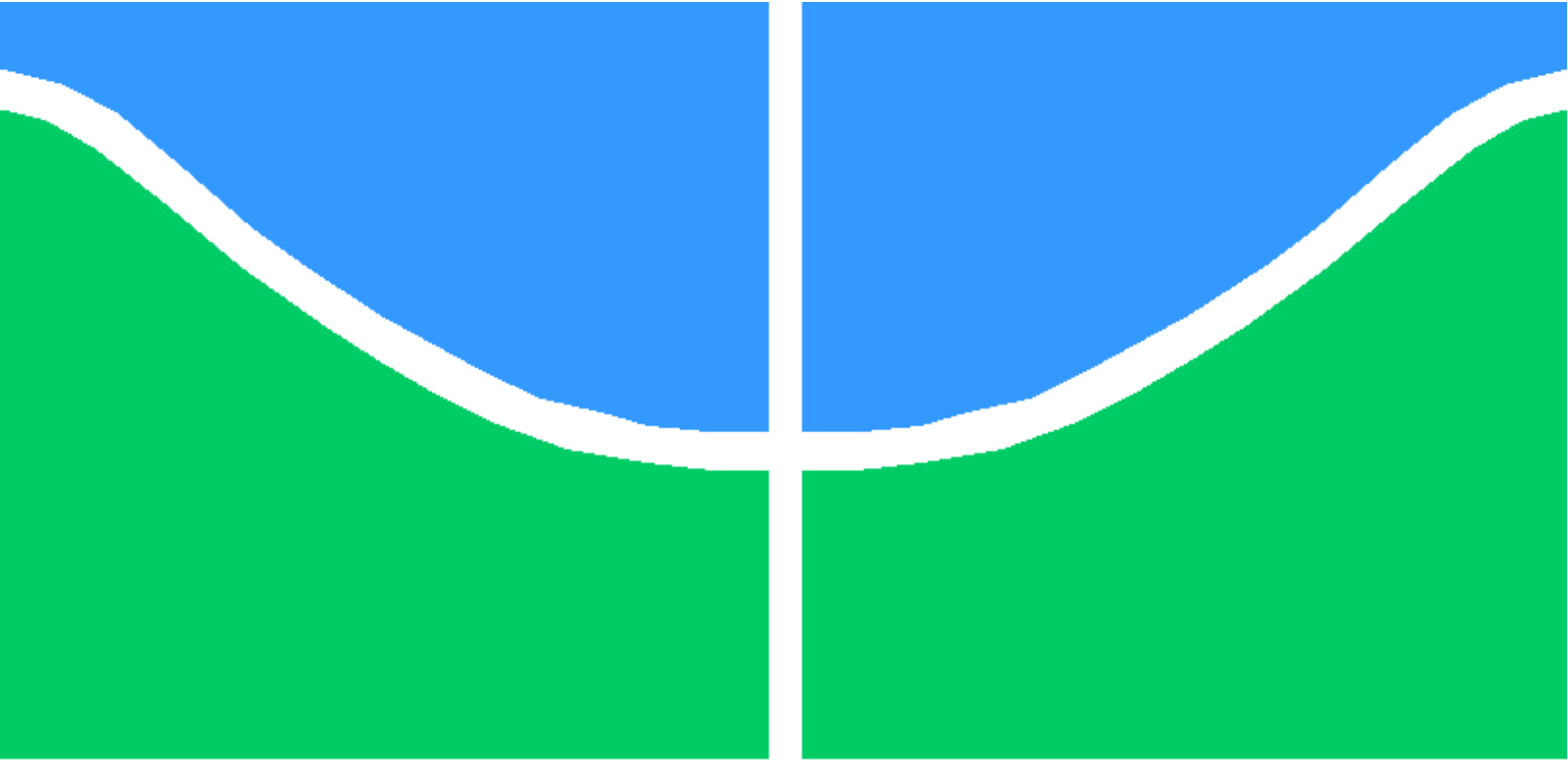

TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE UM PROTOCOLO EFICIENTE IOT COM LORA PARA AUTOMAÇÃO PREDIAL

Leonardo Nunes Cornelio Rêgo

Brasília, dezembro de 2020

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

DESENVOLVIMENTO DE UM PROTOCOLO EFICIENTE IOT COM LORA PARA AUTOMAÇÃO PREDIAL

Leonardo Nunes Cornelio Rêgo

Relatório submetido ao Departamento de Engenharia
Elétrica como requisito parcial para obtenção
do grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Adolfo Bauchspiess, ENE/UnB _____
Orientador

Prof. Marcelo Menezes de Carvalho, ENE/UnB _____
Examinador

Dr. Vinícius Galvão Guimarães, ENE/UnB _____
Examinador

FICHA CATALOGRÁFICA

REGO, LEONARDO N. C.
Desenvolvimento de um protocolo eficiente IoT com LoRa para automação predial,

[Distrito Federal] 2020.

x, 128p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2020). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Automação predial 2.LoRa
3. IoT 4.Protocolo de Comunicação

I. Mecatrônica/FT/UnB II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

REGO, LEONARDO N. C, (2020). Desenvolvimento de um protocolo eficiente IoT com LoRa para automação predial. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$022, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 128p.

CESSÃO DE DIREITOS

AUTOR: Leonardo Nunes Cornelio Rêgo

TÍTULO DO TRABALHO DE GRADUAÇÃO: Desenvolvimento de um protocolo eficiente IoT com LoRa para automação predial.

GRAU: Engenheiro ANO: 2020

Leonardo Nunes Cornelio Rêgo

Condomínio Bela Vista Módulo D Casa 41.

73105-909 Brasília – DF – Brasil.

Dedicatória

Dedico este trabalho à minha família e à minha namorada, que sempre me apoiaram incondicionalmente, nos momentos de dificuldade e estudos. Sem eles, a minha trajetória até esse momento não seria possível.

Leonardo Nunes Cornelio Rêgo

Agradecimentos

Agradeço primeiramente aos meus familiares pelo apoio prestado na trajetória da graduação. Estiveram comigo nos momentos alegres e difíceis, me dando forças, conselhos e incentivos.
À minha namorada, por ser uma verdadeira companheira, que sempre me apoia nas decisões tomadas, me dá forças para lutar e se mostra um pilar em minha vida.
Por fim, gostaria de agradecer ao professor Adolfo, pelas aulas ministradas e posteriormente pelo apoio e suporte prestados durante o trabalho de graduação.

Leonardo Nunes Cornelio Rêgo

RESUMO

A proposta deste trabalho é desenvolver um protocolo *Internet of Things* (IoT) eficiente que seja robusto a interferências, energeticamente eficaz, que possa realizar comunicações com bons alcances, para que não seja necessária a utilização de protocolos de roteamento, de fácil utilização e confiável no contexto de automação predial. O ponto de partida deste trabalho é a definição do protocolo de comunicação dos dispositivos. Na segunda parte, é realizado o desenvolvimento do protocolo e a implementação de uma aplicação que utilize o protocolo, no contexto de automação predial. As informações foram dispostas em um sistema supervisório que atua como a central do sistema. Finalmente, a última etapa consiste da validação do sistema: teste da aplicação implementada de modo a se obter o desempenho do protocolo de comunicação em relação à confiabilidade da comunicação, eficiência energética do sistema, distâncias de comunicação, outros sistemas.

A tecnologia utilizada para a comunicação do sistema é LoRa, que permite comunicações em longas distâncias com uma boa eficiência energética. Por ser baseado na técnica de modulação por espalhamento espectral, derivada da técnica de espalhamento espectral *Chirp Spread Spectrum* (CSS), a energia do sinal transmitido passa a ocupar uma banda muito maior do que a da informação, o que torna as comunicações resistentes a interferências.

A arquitetura geral proposta consiste da utilização de microcontroladores alimentados por bateria para atuação e sensoriamento do sistema. Os mesmos, se comunicaram com os *gateways*, regidos pela estrutura do protocolo. Os *gateways* são responsáveis por receber as mensagens e encaminhá-las para o sistema supervisório, por meio do protocolo *Message Queuing Telemetry Transport* (MQTT). O sistema supervisório é executado em um *Raspberry Pi*.

Testes foram realizados para se averiguar a eficiência da rede. Por meio de testes de 15 dias e de 7 dias, verificou-se o sistema em relação à consumo energético, tempo de latência, alcance e confiabilidade das comunicações. Comunicações de aproximadamente 450 metros de distância foram possíveis utilizando o protocolo, e obteve-se uma confiabilidade das comunicações de 97,6% em um cenário de teste. Além disso, constatou-se o baixo consumo energético dos dispositivos, na ordem de 40,2 μA, ao se utilizar o modo de baixo consumo energético, um consumo até 22 vezes menor que outras tecnologias similares.

Palavras Chave: Automação Predial, Protocolo de Comunicação, LoRa, Microcontroladores.

ABSTRACT

This project proposes the development of an Internet of Things (IoT) power eficient protocol that may be interference robust and can perform long-range communication avoiding unnecessary routing protocols. It may also be easy to get to and reliable to building automation applications. The first step of this project is defining de communication protocol of the devices. Subsequently, the development of the protocol and the implementation of an application that uses the protocol in building automation. The information obtained from the application will be concentrated in a supervisory system that is the midpoint of the structure. Finally, the last step consists of the validation of the system. This validation will be made through some set of tests: performance test, that will validate the eficiency of the protocol in communicating; power eficiency, that will validate the power consumption of the devices, and long-range communication distances.

The technology used to communicate the devices is LoRa, a technological innovation that allows long range communications with a good power eficiency. It is based on the spreading factor modulation technique, a derivation of Chirp Spread Spectrum, and for this reason the energy of the transmitted signal occupies a large bandwidth relative to the data which makes the communication resistant to interference.

The proposed architecture is based in the use of battery-powered microcontrollers to actuate and sense. These microcontrollers will communicate ruled by the developed protocol. Gateways will be responsible to receive the messages from microcontrollers and send them to the supervisory system using MQTT protocol. The supervisory system will work in a raspberry Pi.

Tests were performed to measure the eficiency of the proposed system. Through 15 days and 7 days tests, it was possible to stress the system in relation to power eficiency, latency time, communication distance, and reliability in the communication. It was able to achieve 450 meters communication and a communication reliability of 97,6%. Besides that, it was measured a power consumption of approximately $40.2\mu A$ with the use of the low power mode that is up to 22 times more efficient.

Keywords: Building Automation, Communication Protocol, LoRa, Microcontrollers.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

IoT	Internet of Things
NIC	National Intelligence Council
RFID	Radio-Frequency Identification
LARA	Laboratório de Automação e Robótica
LPWAN	Low-Power Wide-Area Network
TCP/IP	Transmission Control Protocol/Internet Protocol
MQTT	Message Queuing Telemetry Transport
CSS	Chirp Spread Spectrum
AM	Amplitude Modulada
FM	Frequência Modulada
SF	Spreading Factor
FEC	Forward Error Correction
CR	Coding Rate
EEPROM	Electrically-Erasable Programmable Read-Only Memory
SRAM	Static Random Access Memory
RTC	Real Time Clock
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
AES	Advanced Encryption Standard
openHAB	Open Home Automation Bus
SQL	Structured Query Language
DEV_ID	Device Identification
APP_ID	Application Identification
MAC	Message Authentication Code
SEQ_ID	Sequencial Identification
RTS	Request to Send
CTS	Clear to Send
ACK	Acknowledge
DOS	Denial of Service
RSSI	Received Signal Strength Indication
SNR	Signal-to-Noise Ratio

CRC	Cyclic Redundancy Check
IDE	Integrated Development Environment
SD Card	Secure Digital Card
SSH	Secure Shell
AP	Access Point
A/D	Conversor Analógico-Digital
ANATEL	Agência Nacional de Telecomunicações

Capítulo 1

Introdução

1.1 Motivação

Sistemas de automação predial buscam, de modo geral, prover condições de conforto térmico ao ambiente predial, por meio de controle de condicionadores de ar, ventilação e iluminação. Bem como, eles podem prover aspectos de segurança patrimonial, controle de acesso, iluminação, energia, prevenção de incêndios, elevadores, entre outros. Utilizando sensores para monitorar o ambiente e atuadores para modificar o mesmo, o processo de automação tem a principal função de tornar os processos mais eficientes, atribuindo funções que antes eram desempenhadas por seres humanos para as máquinas, que as realizam com mais precisão, de forma mais rápida e a um custo reduzido. O foco no conforto térmico visa produzir um ambiente ergonômico, mais favorável ao trabalho, além de racionalizar o consumo de energia.

De acordo com International Energy Agency (IEA), os edifícios são responsáveis por, aproximadamente, 36% do consumo energético global, além de 40% de toda emissão de $C0_2$. Essa demanda energética continua a crescer a uma taxa de, aproximadamente, 3% ao ano. No contexto brasileiro, estima-se que as edificações representam 46,7% do consumo total de energia elétrica no Brasil [1, 2]. Visto isso, é notório o impacto que pode ser causado por sistemas de automação predial sobre o consumo de energia.

Outros estudos [3], também mostram que os prédios, comerciais e residenciais, representam um dos maiores consumos de energia elétrica no mundo, principalmente em países desenvolvidos, onde as taxas de consumo podem se aproximar dos 40% do total de energia consumida pelo país. Nesse contexto, uma redução de 23% da energia consumida pelos prédios utilizando automação [4] pode gerar uma redução de até 9,2% na energia total do país. De acordo com uma pesquisa da On World [5], 59% de um grupo de usuários de 600 pessoas que estão abertos à experimentação de novidades (*early adopters*) estão interessados em tecnologias para melhorar o monitoramento de consumo de energia e 81% estão dispostos a pagar por um sistema de automação que gere economias de pelo menos 30% nas contas de energia.

Segundo Kastner [6], sistemas de automação predial têm como meta, além da redução de custos e do uso energético, como exposto acima, a utilização das informações de diversos subsistemas

de automação com o objetivo de criar prédios inteligentes. A definição de prédios inteligentes não é um consenso entre os autores, porém no contexto de cidades inteligentes (*smart cities*), a principal ideia está na utilização de tecnologias de modo a tornar os ambientes o mais eficientes possíveis, utilizando e gerenciando recursos de forma integrada, minimizando os custos de operação, melhorando a qualidade dos serviços oferecidos e tornando o ambiente um local confortável para seus ocupantes [7]. A figura 1.1 ilustra um sistema de automação predial típico.

Por ser um assunto complexo, que pode gerar contribuições relevantes, há uma diversidade enorme de soluções propostas, com os mais diversos tipos de tecnologia empregados. Isso se deve ao fato de existirem muitos problemas a serem resolvidos, e as soluções existentes não contemplarem todas as variáveis, ou possuírem limitações que impactam diretamente na eficiência do sistema. Alguns dos problemas encontrados nesses sistemas são a dificuldade de integração dos mesmos, limitações energéticas para sistemas sem fio, problemas com cobertura, segurança dos dados, robustez, custos elevados com infraestrutura, entre outros [8, 9, 10].

Figura 1.1: Subsistemas típicos no contexto de automação predial [11].

Nesse contexto, um conceito não muito recente que vem ganhando uma expressiva importância, é o conceito de IoT. O termo IoT, foi apresentado primeiramente por Kevin Ashton do MIT Auto Centre, em uma apresentação sobre *Radio-Frequency Identification* (RFID) e a cadeia de suprimentos, em 1999 [12]. Segundo Kevin Ashton, "Se tivéssemos computadores que soubessem tudo que há pra saber sobre as coisas - utilizando os dados que eles adquirem, sem a intervenção

humana - seriamos capazes de rastrear e contar tudo, reduzindo os desperdícios, perdas e custos. A Internet of Things (IoT) tem o potencial de mudar o mundo, assim como a Internet mudou."

Embora o termo IoT tenha surgido há muito tempo, não há um consenso em relação a sua definição. Segundo Whitmore [13], *Internet of Things* é um paradigma onde, os objetos do nosso dia a dia se tornam capazes de identificar, sensoriar, trabalhar em rede e processar informações que irão permitir que os mesmos se comuniquem uns com os outros e com serviços e dispositivos por meio da Internet, com o intuito de realizar uma tarefa em conjunto. Os dispositivos IoT serão capazes de tornar os ambientes inteligentes.

Ainda, de acordo com o *National Intelligence Council* (NIC) dos Estados Unidos [14], a Internet of Things é uma das seis tecnologias civis disruptivas. Assim, uma possível abordagem para a automação predial e a criação de prédios inteligentes é a utilização do paradigma IoT.

1.2 Objetivos

Os objetivos deste trabalho foram divididos em duas categorias: objetivos gerais e objetivos específicos. Ambos estão descritos a seguir.

1.2.1 Objetivos Gerais

A proposta central do projeto é a implementação de um sistema de Internet of Things para utilização em um contexto de automação predial. Para isso, é necessário o desenvolvimento de um protocolo de comunicação entre os dispositivos e a Internet, bem como a criação de uma solução para automação predial, que envolva sensores e atuadores conectados pela rede para validação do sistema.

Uma vez que os sistemas de automação predial existentes não são facilmente integráveis [8], é de interesse um sistema de fácil integração entre os dispositivos, sejam eles sensores ou atuadores de diversas aplicações distintas, com alta cobertura, de fácil expansão, além de uma robustez no envio das informações de forma a garantir a integridade e segurança dos dados trafegados.

Além disso, é de interesse do projeto ainda, um sistema em que haja um baixo consumo energético, pois em um contexto de automação predial, são esperados grandes sistemas, com muitos dispositivos conectados, e que muitas vezes necessitarão estar conectados a baterias.

1.2.2 Objetivos Específicos

Os objetivos específicos do projeto são:

- Criação de um protocolo de rede para comunicação dos dispositivos com a Internet;
- Análises de desempenho do sistema, como eficiência energética, robustez etc;
- Comparação dos resultados obtidos com resultados de outros trabalhos da área;

- Validação do protocolo para o contexto de automação predial;
- Criação de um cenário simplificado com sensores e atuadores.

O cenário de testes escolhido para o projeto foi inicialmente a sala de reuniões do Laboratório de Automação e Robótica (LARA). Devido às restrições externas e excepcionais geradas pela pandemia ocorrida durante a realização deste trabalho, o cenário de testes foi modificado para a casa do autor, que não acarreta em prejuízos para a validação do protocolo. Logo, o cenário simplificado para realização de testes do sistema de automação predial serão realizados em um ambiente residencial ao invés de um ambiente que melhor retrata um ambiente predial.

1.3 Apresentação do Manuscrito

No capítulo 2 é feita uma revisão bibliográfica sobre o tema de estudo e no capítulo 3 uma descrição dos materiais e ferramentas utilizadas. Em seguida, o capítulo 4 descreve o protocolo de rede proposto, seguido da metodologia empregada no desenvolvimento do projeto e cenários de teste no capítulo 5 . Por fim, as conclusões do trabalho são discutidas no capítulo 6 . Os anexos contém material complementar.

Capítulo 2

Fundamentação Teórica

2.1 Automação Predial

A automação, genericamente falando, tem o objetivo principal de tornar os processos mais eficientes, de modo a não ser necessária a intervenção do ser humano em tarefas que podem ser resolvidas pelas máquinas de forma mais rápida, com maior precisão e custos reduzidos. Esse conceito é aplicável em diversos cenários de nossas vidas, desde os mais simples, como máquinas para lavar roupa ou louça de forma autônoma, até sistemas complexos como a automação de prédios ou processos de manufatura [15].

De acordo com Domingues [16], apesar da popularidade do tema, um aspecto surpreendente de automação predial é a escassez de referências literárias a respeito do tema. Essa escassez literária gera problemas de comunicação entre os desenvolvedores e contribui para um problema comum em sistemas de automação, a dificuldade de integração de soluções de diferentes fabricantes.

Um sistema de automação predial consiste de um sistema instalado em um prédio que controle e monitore, sem a necessidade da intervenção humana, serviços responsáveis por aquecimento, ventilação, condicionador de ar, iluminação, serviços para garantir a segurança das pessoas, sistemas de alarme, entre outros, de modo a tornar os processos mais eficientes. Para isso, as informações do sistema, como por exemplo a potência do ar condicionado, a temperatura das salas e diversas outras variáveis, precisam ser trocadas entre os dispositivos que compõem o sistema. Essas informações podem ser trocadas por meio de fios, o que torna o sistema menos flexível a mudanças, ou utilizando tecnologias sem fio de curto alcance, como Zigbee [17, 18, 19] e Bluetooth [20], ou longo alcance, como LoRa e SigFox [21], que tornam os sistemas mais versáteis.

Com a diversidade de tecnologias possíveis de se utilizar, uma infinidade de soluções podem ser propostas, cada uma com suas características particulares. Ao analisar as diversas soluções propostas na literatura, notam-se alguns problemas que estão presentes na grande maioria das propostas. Dentre esses problemas, podemos destacar a dificuldade de integração de sistemas, por exemplo, um sistema de condicionamento de ar não trocar informações com um sistema de cortinas, aonde o fechamento de cortinas pode impactar diretamente na eficácia do sistema de condicionamento de ar, pois a radiação solar gera um aumento na temperatura do ambiente

[6]. Destaca-se ainda problemas com a segurança, privacidade e confiabilidade dos dados, onde são impostos desafios para garantir a corretude dos dados e garante que a lei de proteção de dados pessoais não seja quebrada [10, 22]. Além de todos os desafios expostos, as dificuldades de adoção de sistemas de automação estão relacionadas também com os altos custos e dificuldades de instalação [23].

Automação predial vem ganhando muito espaço devido ao seu potencial de reduzir custos energéticos [15, 16, 24], além de facilitar a operação, monitoramento e manutenção do prédio enquanto melhora o conforto dos ocupantes. Isso só é possível devido a uma grande quantidade de sensores (temperatura, concentração de $C0_2$, níveis de iluminação, de presença, etc.) que geram informações que possibilitam decisões de como os equipamentos do prédio serão controlados. Além de sensores, é necessário um sistema que interprete os dados dos sensores da melhor maneira possível. Ainda em [15], um fator muito importante para sistemas de automação são as informações, e mais do que isso, o processamento dessas informações é uma tarefa essencial. Essa importância é reconhecida em [25] e ganhou maior relevância com o crescimento de complexidade e tamanho dos sistemas de automação.

Assim como retratada acima, há uma dificuldade de integração de sistemas de automação predial, que muitas vezes decorre pelos sistemas se comunicarem com tecnologias diferentes. Mais do que simplesmente dificuldade de comunicação entre tecnologias, a incompatibilidade dos sistemas existentes advém também das mais diversas aplicações, e suas particularidades, que acabam por moldar a arquitetura dos sistemas, e que gera as incompatibilidades [26]. Um bom exemplo que retrate o exposto, seria a utilização da tecnologia Bluetooth, aonde o sistema funcionaria bem se os sensores se encontrassem próximos um dos outros, mas ao utilizar em um cenário em que estes se encontrem distantes, problemas surgiriam advindos do pequeno alcance da tecnologia. Existem diversas soluções no mercado, porém as mais consolidadas são BACnet[27], KNX[28], LonWorks[29], Modbus[30], assim como tecnologias de comunicações sem fio como Zigbee[31] e EnOcean. Recentemente, as tecnologias *Low-Power-Wide-Area-Networks* (LPWANs) vem ganhando crescente interesse, justificado pelos longos alcances e baixo consumo energético.

Além dos problemas citados, que estão mais relacionados com a parte de transporte das informações, grandes desafios no contexto de automação predial estão relacionados com o tratamento das informações, e a disponibilização dos dados para os usuários. Em outras palavras, gerar informações a partir dos dados obtidos pelos sensores, processá-los e tomar a melhor decisão para o sistema, levando em conta as variáveis observadas. Com a crescente disponibilidade de informações que podem ser obtidas, processamento inteligente de dados se torna necessário, como abordagens utilizando técnicas de aprendizado de máquina, que gerem valor ou melhorias em conforto para os usuários. Abordagens recentes buscam resolver o problema de processamento de dados de forma inteligente, se baseando em modelos biológicos [32], adicionando os seres humanos no *loop* de controle [33] ou até mesmo na utilização de ferramentas analíticas para processamento de dados para gerar comportamento autônomo e inteligente [34]. Na Figura 2.1, pode-se observar uma grande quantidade de sensores e atuadores presentes em uma sala e em um contexto predial, porém essa quantidade pode aumentar muito.

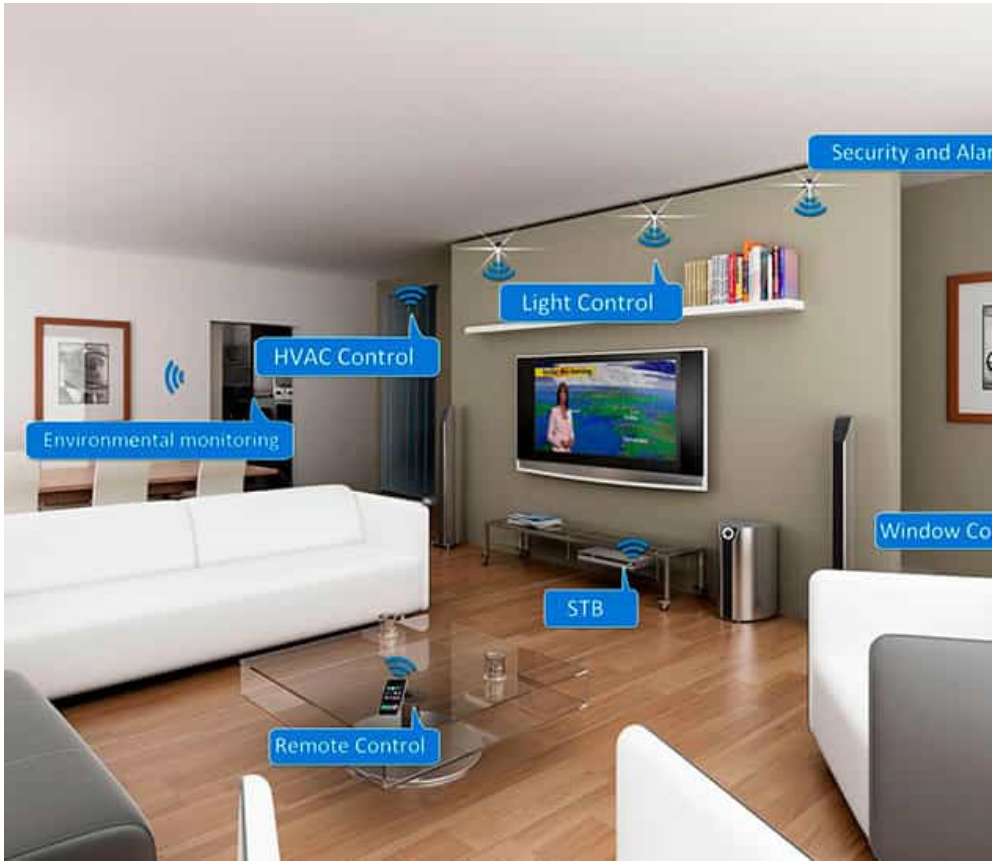

Figura 2.1: Variedade de sensores em uma sala de estar moderna [35].

2.2 Internet das Coisas

A internet das coisas é um recente paradigma de comunicação no qual os objetos do nosso cotidiano são equipados com microcontroladores, transceptores para comunicação digital e protocolos de comunicação que tornarão possível a comunicação entre esses objetos e dos objetos com as pessoas, tudo isso utilizando a Internet [36]. A partir do momento em que haja interação e fácil comunicação entre uma grande quantidade de dispositivos, como eletrodomésticos, sistemas de segurança, sensores, atuadores, veículos, entre outros, será possível desenvolver uma grande quantidade de serviços a partir dos dados gerados e da interação entre os dispositivos. Esse paradigma pode encontrar aplicações nas mais diversas áreas, pois praticamente todos os serviços se beneficiam dessa interação entre os dispositivos, como automação residencial [18, 20], predial [17], industrial [19], indústria automotiva, gerenciamento de tráfego, assistência médica, dentre outros diversos.

Apesar de não haver consenso em sua definição, o propósito da IoT consiste em facilitar a troca de dados entre os dispositivos de modo seguro e confiável, servindo como infraestrutura que possa permitir a criação de ambientes inteligentes, automação de processos ou até a computação ubíqua [37]. Segundo *International Telecommunication Union* (ITU) [38], a Internet das Coisas é uma revolução tecnológica que representa o futuro da computação e da telecomunicação, porém seu desenvolvimento depende da inovação tecnológica em áreas como as comunicações sem fio e

nanotecnologia. Na Figura 2.2, podemos observar aplicações em redes IoT no contexto de cidades inteligentes.

Figura 2.2: Abrangência de aplicações em Smart Cities via IoT [39].

Um sistema de Internet das Coisas normalmente é dividido em camadas, para facilitar o entendimento e a dinâmica do sistema. A divisão mais comum encontrada na literatura é a divisão em 3 camadas: A camada de percepção, camada de rede e a camada de aplicações. A camada de percepção é composta pelos dispositivos, que são responsáveis pela obtenção de dados. Já a camada de rede é responsável por transmitir e processar os dados obtidos pelos dispositivos da camada de percepção. Por fim, a camada de aplicações recebe as informações da camada de rede e é responsável por tratar os dados, criando interfaces para dar clareza nos dados, se assemelhando a um sistema supervisório, monitorando e supervisionando as variáveis e os dispositivos do sistema. Um exemplo de sistema supervisório pode ser visualizado na Figura 2.3.

No entanto, existem propostas na literatura que dividem a arquitetura de um sistema de Internet das Coisas em 4 camadas [41]: Camada Sensorial, Camada de Rede, Camada de Serviços e a Camada de Interface. Na camada sensorial se encontram os hardwares para sensoriar e controlar o mundo físico, além da aquisição de dados. A camada de Rede provê a infraestrutura necessária para as comunicações entre os dispositivos e suporte de rede, sendo ela cabeada ou sem fio. Já a Camada de Serviços é responsável por criar e gerir serviços para o usuário. Por fim, a Camada de Interface provê métodos de interação com o usuário e outras aplicações.

Outras propostas [42] acreditam ainda que, uma divisão em 5 camadas torna a arquitetura de mais fácil entendimento e funcionalidade, dividindo o sistema em: camada de percepção, camada de transporte, camada de processamento, camada de aplicações e camada de negócios. Ao analisar essas propostas, no entanto, percebe-se que os nomes, e as funcionalidades de cada camada se assemelham muito. Os elementos comuns que compõem um sistema de IoT [43] são: dispositivos para aquisição de dados do meio e atuação, um protocolo de comunicação entre os dispositivos e

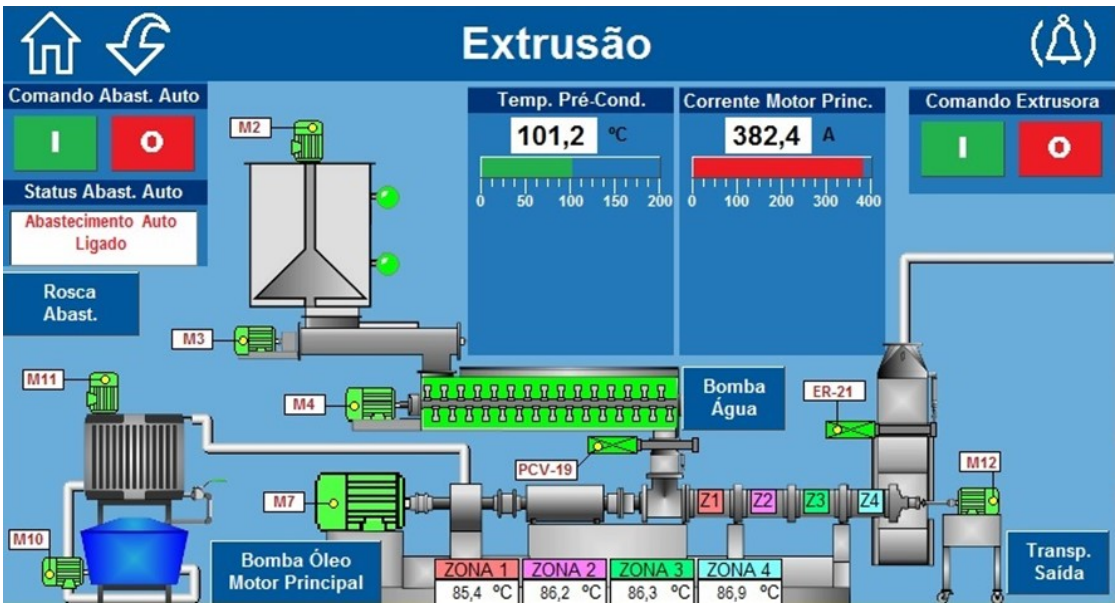

Figura 2.3: Ilustração: tela típica de um sistema supervisório [40].

a Internet, um sistema central para processamento de dados e tomada de decisões e uma interface para disponibilização das informações.

Por ser um paradigma recente, há diversos desafios que precisam ser vencidos, sendo eles tanto sociais quanto técnicos [13]. Em relação aos aspectos técnicos existem os seguintes desafios: limitação energética (quanto tempo um dispositivo pode operar sendo alimentado por baterias), latência (quanto tempo é necessário para as mensagens serem enviadas e processadas), vazão (quantidade máxima de dados/s que podem ser enviados pela rede), escalabilidade (quantos dispositivos são suportados pela rede), topologia (quais são as regras de comunicação, e quem pode se comunicar com quem) e segurança (quão seguro e confiável a aplicação é). Em relação aos desafios sociais temos: com a maior quantidade de dispositivos estando conectados em rede e trocando informações, a privacidade pessoal pode ser ameaçada? É necessária a criação de soluções que assegurem que os dados pessoais não serão utilizados sem o consentimento das pessoas.

Diversos fatores contribuem para a relevância que vem ganhando a Internet das Coisas, sejam eles fatores relacionados com a alta capacidade de processamento em nuvem, a alta em processamento de dados utilizando paradigmas como redes neurais, Big Data, além da mudança de hábitos das pessoas, em que se deseja automatizar a maior quantidade de processos possíveis, com o intuito de gerar comodidade e conforto para as pessoas. Outro fator muito importante são os recentes avanços tecnológicos em circuitos integrados e comunicações sem fio de baixo consumo energético, com baixo custo e que se tornam cada vez menores. A combinação desses fatores aumentou a viabilidade de utilizar uma larga quantidade de dispositivos da Camada de Sensoriamento para coletar, processar, analisar e transmitir informações nos mais diversos cenários.

2.3 LoRa

LoRa (**Lo**ng **Ra**nge) é uma tecnologia relativamente nova de transceptores que permitem comunicações em longas distâncias com boa eficiência energética (Low-Power Wide-Area Networks ou LPWAN's). É baseado na técnica de modulação por espalhamento espectral, que é derivada da técnica de espalhamento espectral, Chirp Spread Spectrum (CSS). Para entender o funcionamento da tecnologia, primeiro é necessário entender a modulação do sinal, e os parâmetros que podem ser configurados na comunicação. A modulação é a alteração sistemática de uma onda portadora a partir de uma mensagem (sinal modulante). Em outras palavras, como uma informação, seja ela analógica ou digital, é codificada para ser transmitida por meio de ondas eletromagnéticas. Para a informação ser recuperada, é necessário fazer o processo reverso, chamado demodulação. São exemplos de modulação, a modulação em amplitude (AM) e modulação em frequência (FM), que podem ser visualizadas na Figura 2.4.

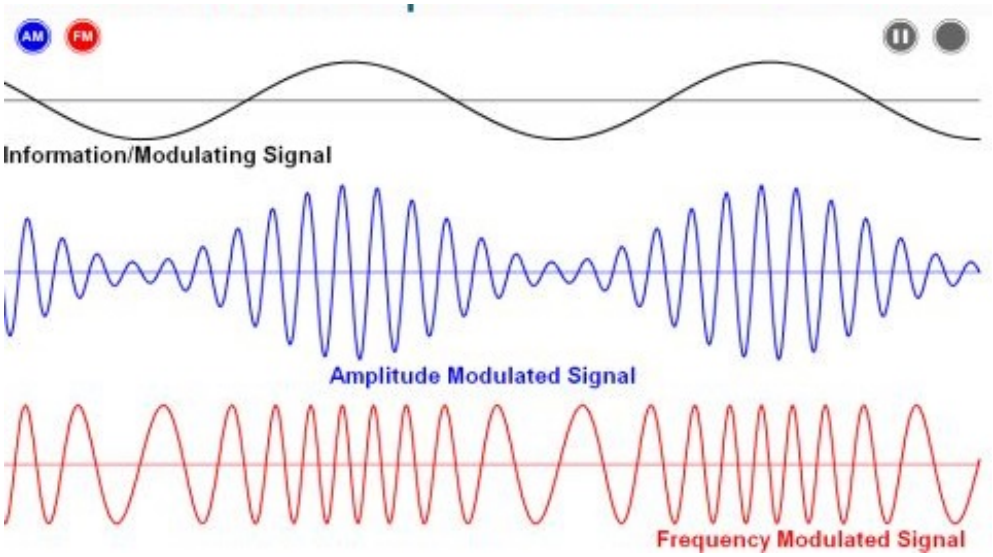

Figura 2.4: Modulação AM e FM [44].

Além das modulações AM e FM citadas acima, existem diversas outras técnicas para modulação de sinais. Espalhamento espectral é uma técnica de modulação em que a largura de banda usada para transmissão é muito maior que a banda mínima necessária para transmitir a informação. Dessa forma, a energia do sinal transmitido passa a ocupar uma banda muito maior do que a da informação. Como exposto, LoRa é uma técnica de modulação por espalhamento espectral, que é baseado em Chirp Spread Spectrum (CSS), que utiliza uma larga banda linear de frequência para modular a informação em chirps. Um chirp, nada mais é do que um sinal no qual ocorre aumento na frequência com o tempo (up-chirp), ou um decréscimo na frequência (down-chirp) com o tempo. Um exemplo de up-chirp e down-chirp podem ser visualizados na Figura 2.5.

Tendo em vista a modulação utilizada, a tecnologia LoRa possui um padrão para transmissão, aonde o início das transmissões deve ser inicializado com uma sequência de up-chirps, seguido de 2 down-chirps. Esse conjunto de up-chirps é denominado preamble ou preâmbulo e os dois down-chirps sync, responsável pela sincronização. Após isso, as informações úteis são enviadas. As

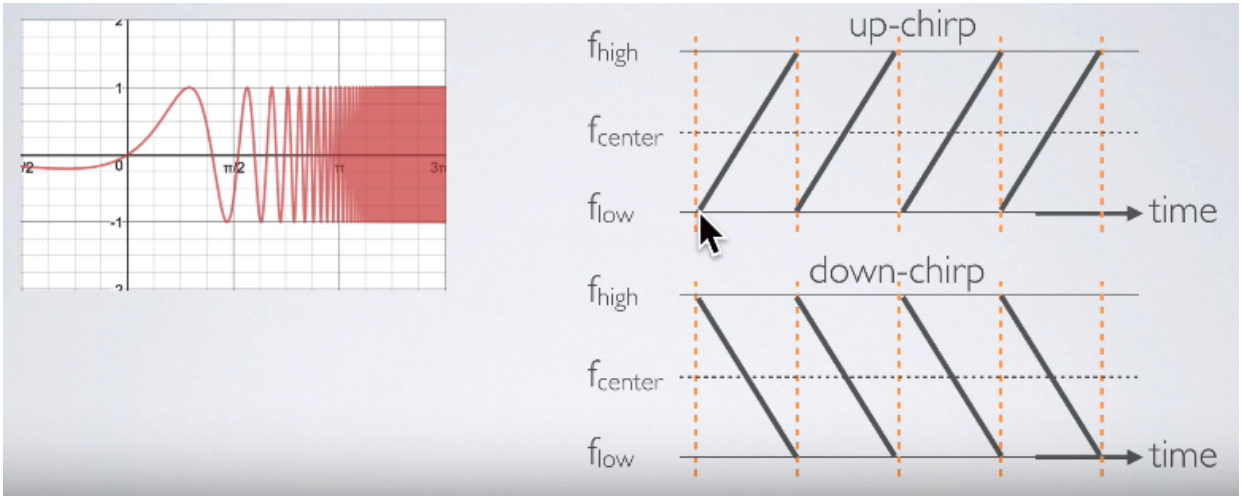

Figura 2.5: Exemplo de um up-chirp e um down-chirp. No gráfico à esquerda, podemos visualizar um up-chirp, em um gráfico amplitude vs tempo e os gráficos à direita são gráficos frequência vs tempo [45].

transições de frequência utilizando chirps representam os símbolos. A codificação dos símbolos é dada pela seguinte forma: A depender do fator de espalhamento, as informações serão codificadas em x bits, onde x é o fator de espalhamento. Em x bits, são possíveis de se representar 2^x símbolos. O chirp completo será dividido em 2^x passos e a frequência inicial do chirp representa o símbolo. Essa codificação pode ser melhor visualizada na figura 2.7. Nesta técnica, cada símbolo é enviado em um sinal de banda estreita propagado sobre uma banda de frequência mais ampla, com a mesma densidade de potência [21, 46]. Logo, o sinal resultante se assemelha a um ruído, o tornando resistente à interferências [47]. Uma desvantagem da técnica é a subutilização da banda larga. Um exemplo de transmissão de mensagem utilizando LoRa pode ser visualizado na Figura 2.6.

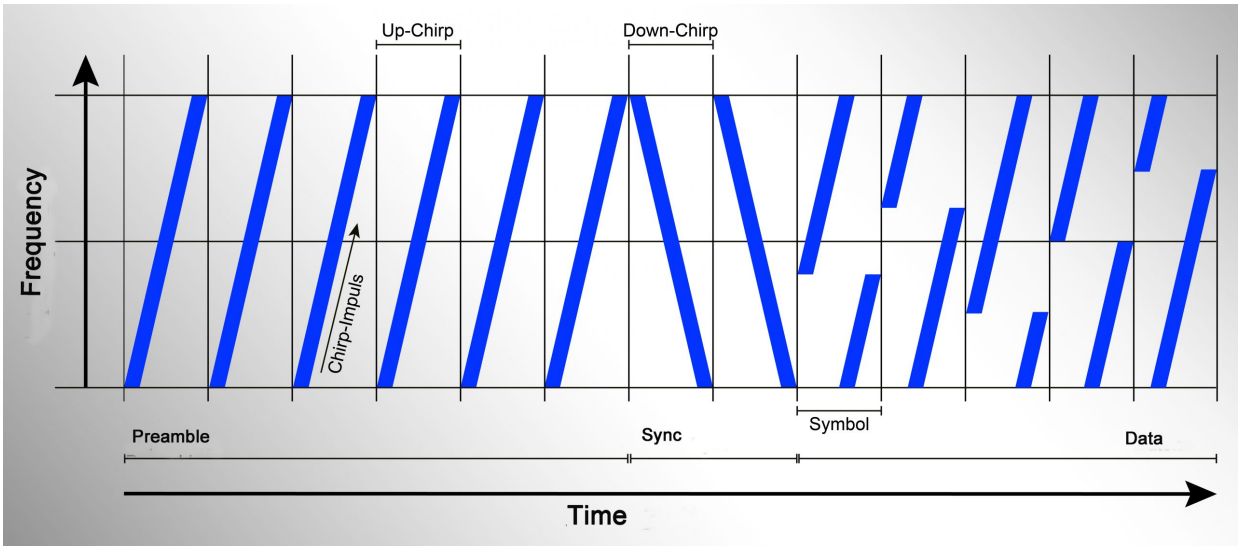

Figura 2.6: Exemplo de transmissão LoRa [48].

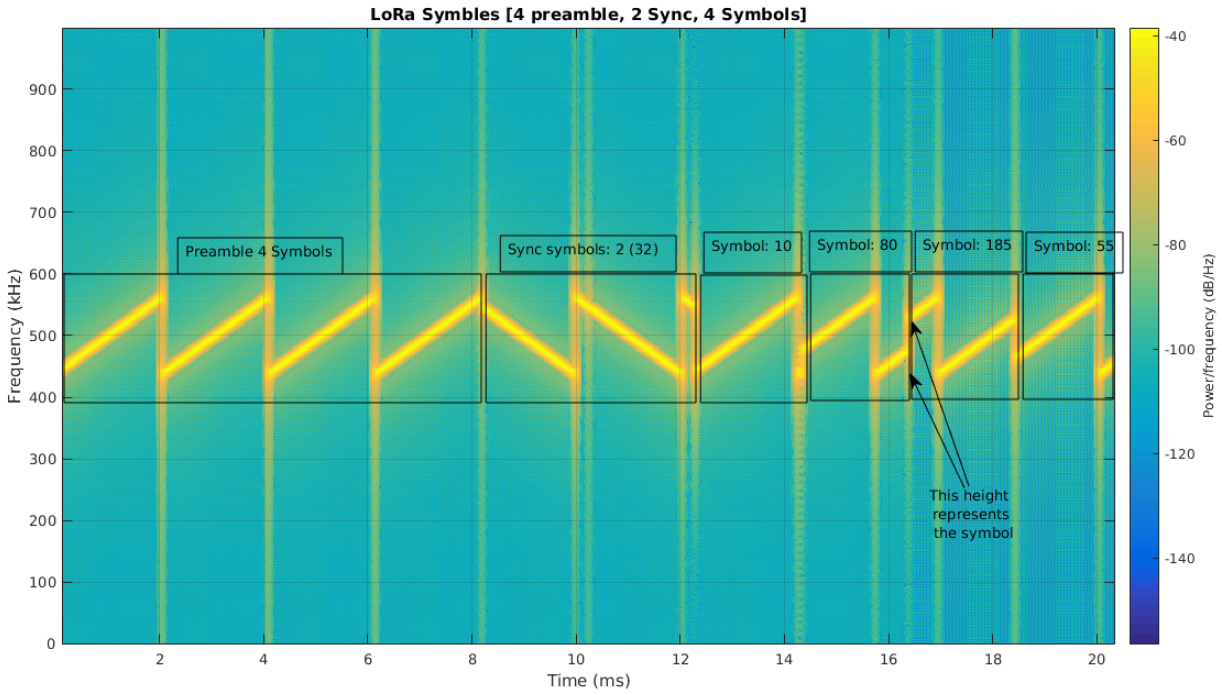

Figura 2.7: Exemplo de transmissão LoRa e a codificação dos símbolos [49].

Alguns parâmetros importantes que afetam a qualidade, velocidade e o alcance da transmissão serão apresentados. São eles: *spreading factor* (fator de espalhamento), *bandwidth* (largura de banda), frequência central e *coding rate* (taxa de código).

2.3.1 Fator de Espalhamento

A tecnologia LoRa conta com 6 spreading factors (SF), indo de SF7 a SF12. O *spreading factor* nada mais é do que a quantidade de bits codificados por símbolo. Quanto maior o valor dele, maior se torna a sensibilidade de recepção e o alcance, porém há um aumento no tempo de envio das informações. Outro ponto importante a ser comentado está no fato de cada *spreading factor* ser ortogonal em relação aos demais, o que permite duas ou mais transmissões e as respectivas recepções ao mesmo tempo, contanto que cada par de comunicações esteja utilizando fatores de espalhamento diferentes. O comportamento da mudança de *spreading factor* pode ser visualizado na Figura 2.8.

2.3.2 Frequência Central e Bandwidth

A frequência central do sinal e a largura de banda estão diretamente relacionadas, pois são elas que definem as frequências que o sinal pode oscilar. Essa relação pode ser visualizada na Figura 2.9. A largura de banda define a largura de espectro que um chirp pode ocupar e influencia diretamente na sensibilidade e na taxa de dados. Ao se aumentar a largura de banda, há um aumento na taxa de dados, ou seja, menor o tempo de transmissão, porém uma menor sensibilidade de recepção. A largura de banda de um chirp LoRa pode ser visualizado na Figura 2.10.

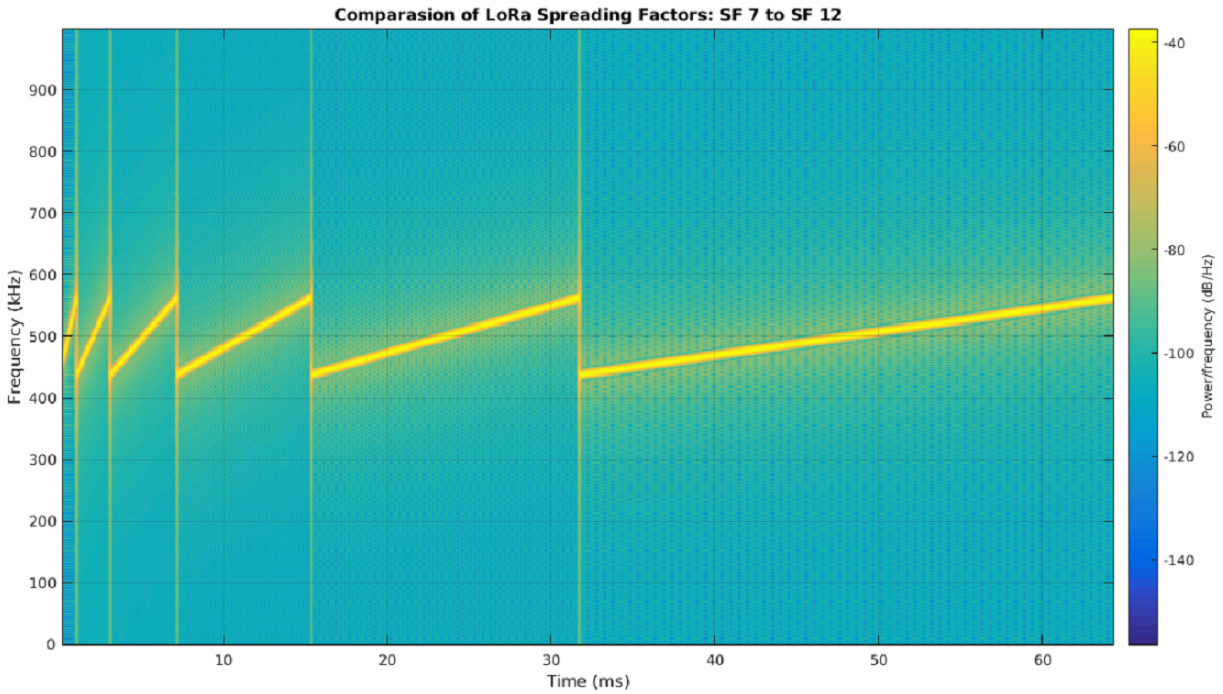

Figura 2.8: Modulação LoRa com diferentes valores de Spreading Factor. Note que com o aumento do Spreading Factor, há um aumento no tempo de transmissão. A escala de azul para amarelo representa a potência do sinal dividido pela frequência. [50]

2.3.3 Coding Rate

LoRa utiliza de uma técnica conhecida como *Forward Error Correction* (FEC), a qual adiciona bits de redundância à mensagem, de modo a ser possível detectar erros de codificação/decodificação, tornando ainda possível a correção da informação para alguns bits incorretos sem a necessidade de retransmissões. A taxa de códigos (*coding rate*) define quantos bits serão utilizados como bits redundantes. Um *coding rate* maior oferece maior proteção, porém, incrementa o tempo de transmissão.

2.3.4 Considerações Finais

As transmissões LoRa utilizam uma banda larga para reduzir a interferência e lidar com descompensações na frequência gerados por cristais de baixo custo. Um receptor LoRa pode decodificar transmissões com 19,5 dB abaixo do piso de ruído. As principais propriedades do LoRa são: longo alcance, alta robustez, resistência a multicaminhos, resistência ao efeito Doppler e baixo consumo energético [53].

Mesmo sendo uma tecnologia recente, nota-se a importância que vem ganhando, pelas excelentes características que possui e pela grande quantidade de interesse nas pesquisas. Para sistemas de IoT, estudos [54, 55] avaliam a viabilidade da utilização de LoRa para o desenvolvimento de sistemas para *smart cities*, motivados pelo baixo consumo enegético dos rádios, grande alcance

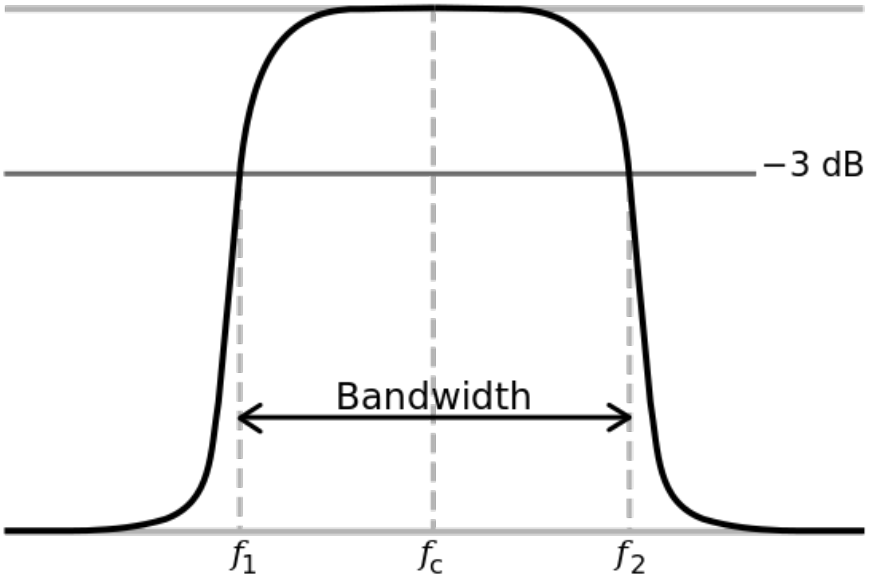

Figura 2.9: Largura de banda e frequência central de um sinal [51].

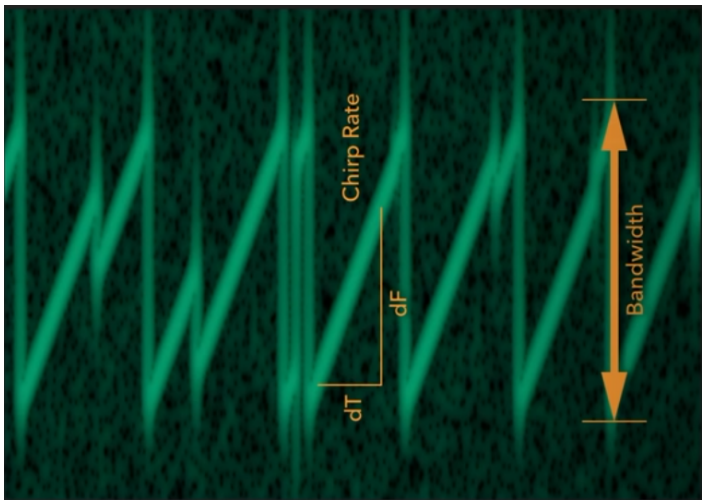

Figura 2.10: Largura de banda de um chirp LoRa em um gráfico frequência [kHz] x tempo[ms] [52].

de comunicação e a resistência a interferências. Outros trabalhos [56], avaliam a influência dos parâmetros de configuração do rádio na comunicação. Por fim, existem estudos [57] para avaliar a escalabilidade de sistemas utilizando rádios LoRa, em outras palavras, com o aumento da quantidade de comunicações verifica-se alterações em sua eficiência. Um resultado interessante dessa pesquisa, é que sistemas LoRa não apresentam perdas em eficiência significativas, contanto que haja uma dinâmica nos parâmetros de transmissão. Isso significa que para sistemas LoRa terem escalabilidade, é necessário um protocolo de comunicação para tornar dinâmicos os parâmetros de transmissão, como frequência, *spreading factor* e largura de banda.

Capítulo 3

Materiais e Ferramentas

Serão apresentados neste capítulo os equipamentos utilizados para o desenvolvimento do projeto, além de algumas ferramentas utilizadas. Vale ressaltar que os códigos serão desenvolvidos para o *hardware* da empresa Nestin.

3.1 EasyDevice

O EasyDevice é um microcontrolador desenvolvido pela empresa Nestin com o objetivo de atuar como um dispositivo em um sistema de Internet das Coisas para obter dados de sensores, controlar atuadores, além de enviar e receber as informações para *gateways* utilizando radiofrequência. Assemelha-se a um *Arduino Pro Mini*, levando em conta a capacidade de processamento, que é a mesma, por utilizarem o mesmo processador, porém o *hardware* do EasyDevice é muito mais enxuto e robusto.

O dispositivo pode ser programado através de um ambiente de desenvolvimento integrado, do inglês Integrated Development Environment (IDE) próprio do *Arduino*, uma linguagem que se assemelha muito ao C/C++. A placa conta com um *bootloader*, que permite a gravação do programa. As especificações gerais do microcontrolador são as seguintes:

- Processador: ATmega328PB 8MHz 8-bit
- Alimentação: Conector Barra de Pinos 5V ou 3 pilhas 1,5V.
- 1 kByte EEPROM
- 2 kBytes SRAM
- Real-time Clock DS1337 (Relógio de tempo real)
- 6 portas Digitais
- 1 porta analógica de entrada
- 1 porta analógica de saída (saída amplificada)

- Radiofrequência: Semtech SX1276
- Antena: RF ANT 915MHZ

Como pode ser observado na Figura 3.1, o dispositivo utiliza o mesmo processador que um *Arduino Pro Mini*. Visto isso, os códigos implementados serão compatíveis para os microcontroladores que possuírem o mesmo processador, contanto que possua também alguns *hardwares* necessários, como os transceptores LoRa, necessários para comunicação entre os dispositivos e o *gateway*.

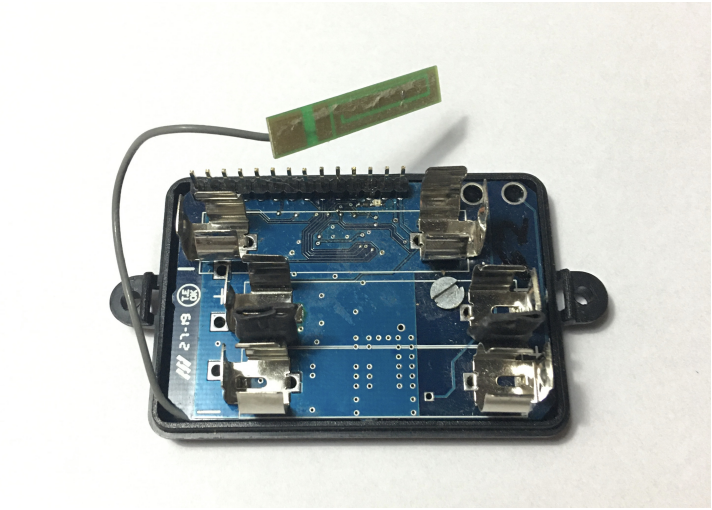

Figura 3.1: EasyDevice.

3.2 Semtech SX1276

A Semtech, que possui a patente da técnica de modulação LoRa, é a fabricante de transceptores LoRa. Esses transceptores possuem a característica de obterem longo alcance nas comunicações, sendo altamente robustos contra interferências, além de possuírem um baixo consumo energético. O microcontrolador se comunica com o modulador LoRa por meio de comunicação *Serial Peripheral Interface* (SPI), e já está embutido no circuito impresso do EasyDevice. Para utilizar outros hardwares, é necessário, por exemplo, conectar ao microcontrolador ao rádio LoRa. As especificações técnicas do transceptor são visualizadas abaixo:

- Alimentação: 1.8 – 3.7V
- Range de frequência: 137 – 1020 MHz
- Fator de espalhamento: 7 – 12
- Largura de Banda: 7.8 – 500kHz
- Bitrate: 0.018 – 37.5 kbps

- Consumo em Sleep mode: 0.2 µA
- Sensibilidade de recepção mínima: -148 dBm
- Consumo em modo de transmissão: 20 – 120mA (+7dBm a +20dBm)

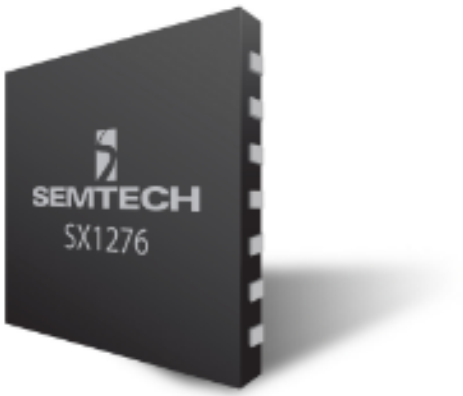

Figura 3.2: Transceptor SX1276 Semtech [58].

3.3 Gateway

Assim como o EasyDevice, o gateway é um microcontrolador desenvolvido pela empresa Nestin com o objetivo de atuar como um concentrador de rede em um sistema de Internet das Coisas, atuando como uma ponte entre os dispositivos do sistema e o servidor de rede. Pode realizar comunicação com os dispositivos via modulação LoRa e com o servidor de rede por *WiFi*. As especificações gerais do microcontrolador são obtidas a seguir:

- Processador: Tensilica Xtensa 32-bit LX6 dual core
- Alimentação: 5V conectado a uma fonte ou USB
- WiFi embutido: padrão 802.11 B/G/N, operando na faixa de 2.4 a 2.5GHz
- Clock ajustável de 80MHz até 240MHz
- 512 kBytes SRAM
- 448 kBytes ROM
- Radiofrequência: Semtech SX1276
- Antena: RF ANT 915MHZ

Seguindo a mesma linha de raciocínio de compatibilidade entre o EasyDevice e o *Arduino Pro Mini*, o *gateway* utiliza o mesmo processador que o microcontrolador *ESP32*. Visto isso, os códigos implementados serão compatíveis para os microcontroladores que possuírem o mesmo processador, contanto que possuam também alguns *hardwares* necessários, como os transceptores LoRa, necessários para comunicação entre os dispositivos e o *gateway*. Existem módulos *ESP32* com rádio LoRa Semtech SX1276 do fabricante *HELTEC*, que poderão ser utilizados como *gateway*, pela similaridade de *hardware*.

Figura 3.3: Gateway.

3.4 Sensor de Temperatura DS18B20

O sensor DS18B20 é um termômetro digital com resolução programável de 9 a 12 bits para medição de temperaturas. Comunica-se com os microcontroladores por meio do protocolo 1-Wire, um sistema de barramento projetado pela Dallas Semiconductor Corp. Cada sensor possui um código serial único de 64 bits, que permite que múltiplos sensores DS18B20 funcionem em um mesmo barramento. As especificações gerais do sensor podem ser visualizadas na tabela 3.1.

Tabela 3.1: Especificações Gerais do Sensor DS18B20.

Alimentação	Consumo do Sensor	Faixa de Temperatura	Precisão	Tempo de Conversão
3V a 5.5V	1mA	$-55°C$ a $+125°C$	$\pm 0.5°C$	$<750ms$

É um sensor a prova d'água do tipo sonda, que é muito utilizado em projetos que envolvam medição de temperatura em ambientes úmidos ou em recipientes com liquido. O sensor é revestido por um material à prova d'água e sua ponta é encapsulada em aço inoxidável. O sensor pode ser

visualizado na Figura 3.4.

Figura 3.4: Sensor de Temperatura DS18B20 [59].

3.5 Relé

O relé é uma chave eletromecânica, projetado para ligar e desligar dispositivos. O chaveamento ocorre quando a corrente elétrica percorre as espiras da bobina do relé, criando assim um campo magnético que atrai a alavanca responsável pela mudança do estado dos contatos. Um das vantagens da utilização de relés é a possibilidade de controle de sistemas utilizando baixas tensões e correntes no chaveamento. Um exemplo seria chavear um relé utilizando microcontroladores, para o acionamento de uma lâmpada. A tensão de funcionamento da lâmpada está na ordem de 220V, enquanto o controle utilizando microcontroladores está na ordem de 5V.

Para validação do sistema proposto, busca-se criar uma aplicação que se assemelhe a um problema real de automação predial. Os sistemas de condicionamento do ar são os mais comuns no contexto de automação predial. O relé será utilizado de forma a simular a atuação em um sistema. O chaveamento do relé pode se associar com o acionamento de um ar condicionado em um sistema de condicionamento do ar. Na Figura 3.5, visualiza-se o relé utilizado. Já o sensor de temperatura tem a função de medir a temperatura da sala, para que a decisão de acionar ou não o relé seja tomada.

3.6 Bibliotecas

Alguns códigos para determinados sensores e/ou atuadores, rádios de comunicação estão disponíveis para determinados microprocessadores. O EasyDevice e o *Gateway* possuem, respecti-

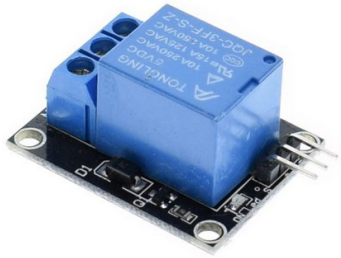

Figura 3.5: Relé JQC-3FF-S-Z [60].

vamente, os mesmos processadores do *Arduino Pro Mini* e *ESP32*, que são muito comuns. Visto isso, os códigos desenvolvidos para estes controladores, também funcionam para o EasyDevice e o *Gateway*. Foram então utilizadas bibliotecas para a utilização dos rádios LoRa e *WiFi*, além de bibliotecas para utilização dos sensores de temperatura, realização de criptografia, entre outras funções. As bibliotecas utilizadas, que estão disponíveis nos links do *Github*, são:

- LoRa: `https://github.com/sandeepmistry/arduino-LoRa`

  Biblioteca responsável por controlar o rádio Semtech SX1276 utilizando os processadores ATmega328PB e Tensilica Xtensa 32-bit;

- WiFi: `https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi`

  Biblioteca com a função de conectar o dispositivo a Internet;

- MQTT: `https://github.com/knolleary/pubsubclient`

  Biblioteca que realiza a comunicação do Gateway com a central do sistema de automação. Utilizando a tecnologia WiFi, e o protocolo MQTT, os dados são enviados e consultados por meio de um broker;

- AES-128 bits: `https://github.com/spaniakos/AES`

  Biblioteca responsável pela realização da criptografia dos dados;

- EEPROM: `https://github.com/arduino/ArduinoCore-avr/tree/master/libraries/EEPROM`

  Biblioteca responsável por controlar a EEPROM (Electrically-Erasable Programmable Read-Only Memory), um tipo de memória não-volátil utilizada em microcontroladores para armazenamento de dados, onde os dados armazenados são mantidos após desligarmos os microcontroladores;

- Dallas Temperature: `https://github.com/collin80/dallas_temperature_control`

  Biblioteca responsável pela obtenção da temperatura do sensor DS18B20;

- One Wire: `https://github.com/PaulStoffregen/OneWire/tree/master/examples`

Biblioteca responsável por realizar a comunicação do sensor DS18B20 com o microcontrolador;

- Time: `https://github.com/PaulStoffregen/Time`

  Biblioteca responsável pela criação de estruturas temporais para Real Time Clocks (RTC), um relógio de tempo real;

- DS1337RTC: `https://github.com/etrombly/DS1337RTC`

  Biblioteca responsável pelo controle do Real Time Clock DS1337, para criação de alarmes que serão responsáveis por retirar os hardwares do modo de baixo consumo;

- SPI: `https://github.com/arduino/ArduinoCore-avr/tree/master/libraries/SPI`

  Biblioteca responsável pela comunicação SPI, realizada entre os microcontroladores e os rádios LoRa;

- AVR Watchdog: `https://www.nongnu.org/avr-libc/`

  Biblioteca nativa da interface de programação do arduino, responsável pela criação de watchdogs, que são temporizadores que disparam um reset automático ao sistema se o programa principal travar;

3.7 OpenHAB

Open Home Automation Bus (openHAB) é um software open source voltado para automação residencial desenvolvido na linguagem Java. Tem como funcionalidade principal atuar como uma central de um sistema de automação, recebendo os dados, seja por soquetes TCP/IP ou *brokers* MQTT, processando as informações, gerando alertas, se comunicando com um banco de dados externo, possibilitando a criação de interfaces, e atuando como um sistema supervisório. Esse software pode ser instalado em um *Raspberry Pi* e utilizado em uma rede local que não depende da conectividade com a Internet, ou pode ser implementado em um servidor na nuvem, se tornando dependente da conectividade com a Internet. Nesse trabalho, optou-se pela instalação do sistema em um *Raspberry Pi*.

Figura 3.6: Sistema para automação residencial [61].

3.8 Raspberry Pi

O *Raspberry Pi* é uma série de embarcados desenvolvidos pela Fundação Raspberry Pi e baseados no conceito de *System on a Chip* (SoC). Seu principal objetivo é difundir e promover o estudo

de Ciências da Computação e Informática em escolas, a um preço acessível. Sua versatilidade e baixo custo lhe permite ser utilizado para projetos simples para o aprendizado, mas também para pesquisa e desenvolvimento de sistemas mais complexos em áreas como automação residencial e predial. Neste trabalho, foi utilizado o *Raspberry Pi* 3 Model B mostrado na Figura 3.7. Tem como especificações gerais:

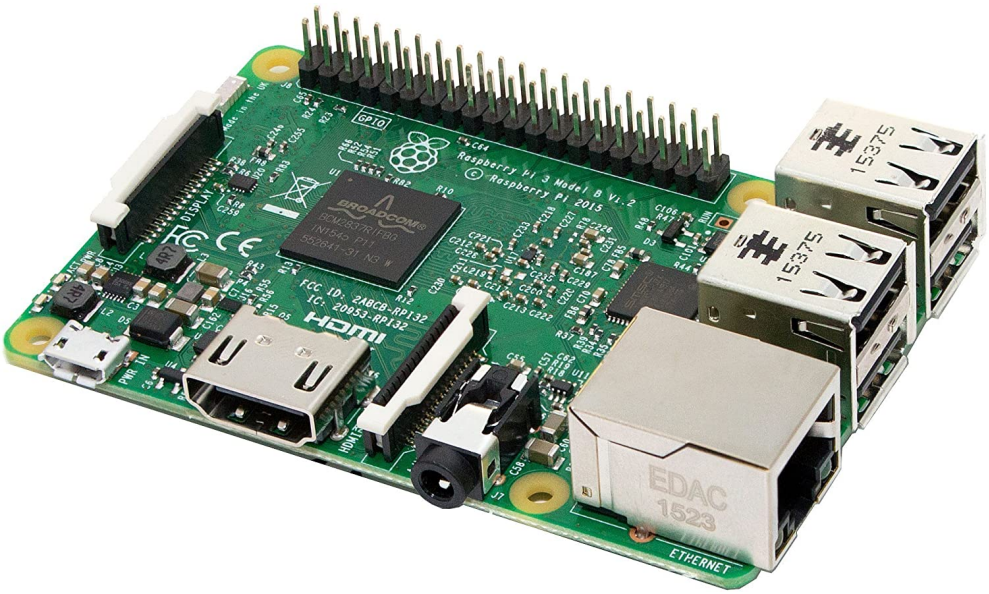

Figura 3.7: Raspberry Pi 3 Model B [62].

- Processador: Quad Core 1.2 GHz Broadcom BCM2837 64 bit CPU;
- Alimentação: Socket Micro USB 5V 2,5A;
- 1 GB de memória RAM;
- 4 portas USB 2.0;
- 15 pinos MIPI interface serial da câmera (CSI 2);
- HDMI para saída de vídeo;
- Porta micro SD para carregar o Sistema Operacional (SO) e salvar os dados;
- Dimensões: 85 x 56 x 17 mm.

O sistema operacional utilizado no *Raspberry Pi* é o *Linux*, configurado a partir do *OpenHABian*, onde é instalado uma imagem no cartão SD pré-configurada com o Linux e todos os softwares para a instalação do *openHAB*. No projeto, o *Raspberry Pi* servirá como central do sistema de automação e onde serão instalados o sistema supervisório *openHAB*, o banco de dados *InfluxDB* e um software analítico de dados, o *Grafana*. O procedimento de instalação do sistema é descrito na seção 5.3.

3.9 InfluxDB

InfluxDB é um banco de dados de código aberto designado para lidar com um alto volume de consultas e escritas por segundo sem causar muito impacto no sistema operacional. O InfluxDB provê uma sintaxe para a realização da consulta muito parecida com a do SQL. O principal uso do InfluxDB é para aplicações de monitoramento em tempo real, já que estas demandam uma quantidade enorme de leitura e escrita no banco de dados. Dessa forma, o uso de uma base de dados que comporte esse volume é essencial para não travar todo os sistema.

3.10 Grafana

O Grafana é uma plataforma para visualizar e analisar métricas por meio de gráficos. Para facilitar a visualização dos gráficos, é possível criar *dashboards* dinâmicos que podem ser compartilhados. Além disso, a ferramenta permite configurar alertas com base nas métricas, que são analisadas de forma contínua para notificar o usuário sempre que preciso, de acordo com as regras definidas por ele. É bastante utilizado por sistemas de monitoramento para gerar gráficos em tempo real. Um exemplo de aplicação pode ser visualizado na Figura 3.8.

Figura 3.8: Exemplo de aplicação de temperatura utilizando Grafana.

3.11 LoRa Modem Calculator Tool

LoRa Modem Calculator Tool é um software desenvolvido pela Semtech para que seja possível simular consumo energético, tempo de transmissão, taxa de transferência de um rádio LoRa. É utilizada nesse projeto para auxiliar o desenvolvimento. Na Figura 3.9, é possível ver o funcionamento do software.

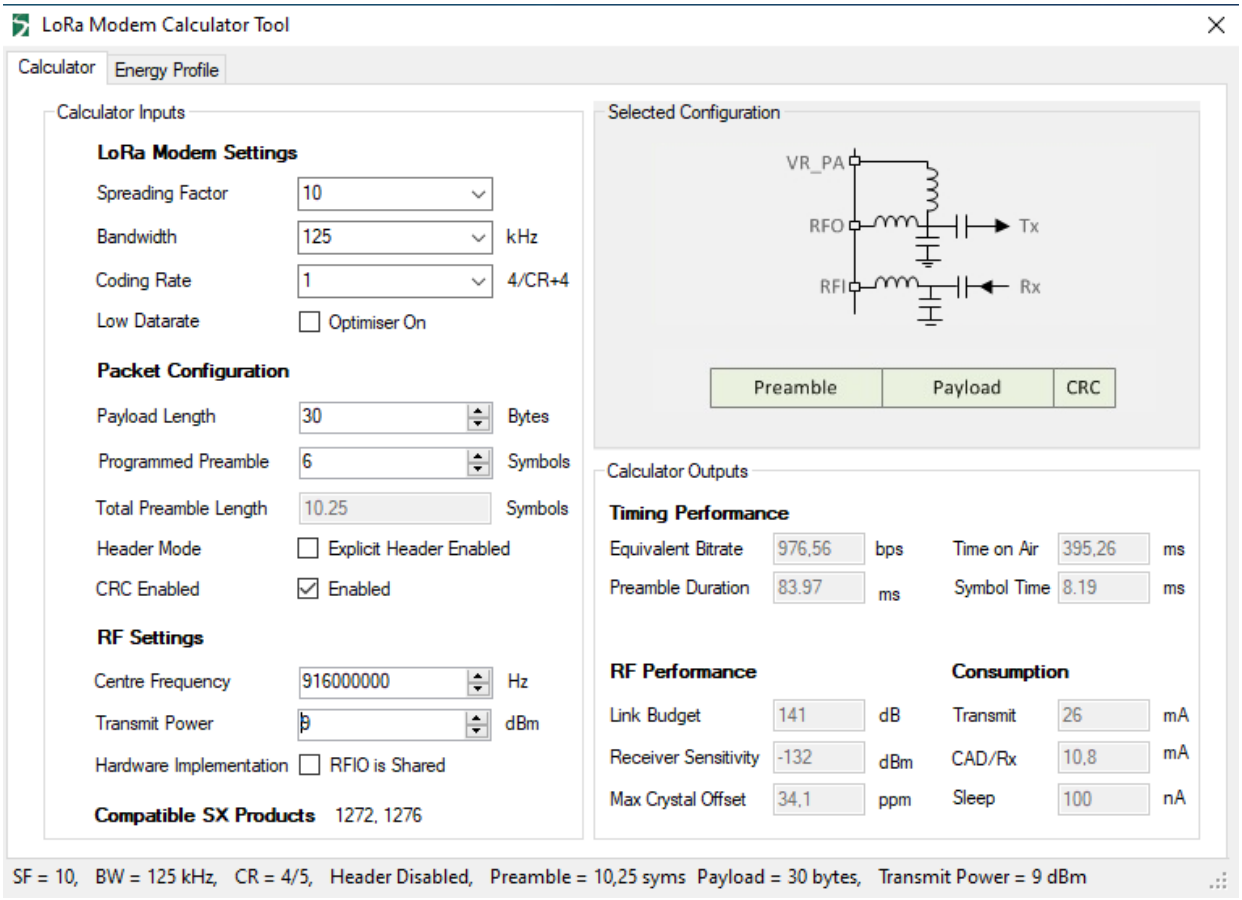

Figura 3.9: Simulador de parâmetros LoRa.

3.12 Arquitetura Geral

O trabalho proposto tem o objetivo da criação de um sistema de automação predial. Para isso, é necessário definir a arquitetura do sistema e um protocolo de comunicação entre os dispositivos. O protocolo de comunicação será definido no Capítulo 4 e um esboço da arquitetura será retratado na figura 3.10 para mostrar a conexão dos hardwares e ferramentas utilizadas.

O microcontrolador EasyDevice tem a função de atuar na parte de sensoriamento e atuação do sistema. Por ser alimentado por baterias, necessita se comunicar com os dispositivos utilizando seu rádio LoRa. É desejável que a bateria dure o máximo possível, para evitar manutenções excessivas nos microcontroladores, pois a quantidade de dispositivos presentes em um sistema de automação predial pode ser na ordem de milhares. Ele troca informações com o Gateway, comunicação esta, que será descrita no protocolo no Capítulo 4.

O *Gateway* tem a função de atuar como um concentrador de rede, recebendo informações dos microcontroladores por radiofrequência, realizando alguns processamentos necessários para assim encaminhar as informações para a central do sistema, o *Raspberry Pi*, por meio do protocolo de rede MQTT.

Já o *Raspberry Pi*, tem a função de receber as informações do *Gateway*, criar interfaces gráficas

para visualização, gerar informações a partir dos dados, criação de alertas, armazenamento de dados a partir do banco de dados *InfluxDB*, geração de análises com o software Grafana, além de alterar as referências para que os microcontroladores atuem no sistema, a depender da entrada desejada pelo usuário.

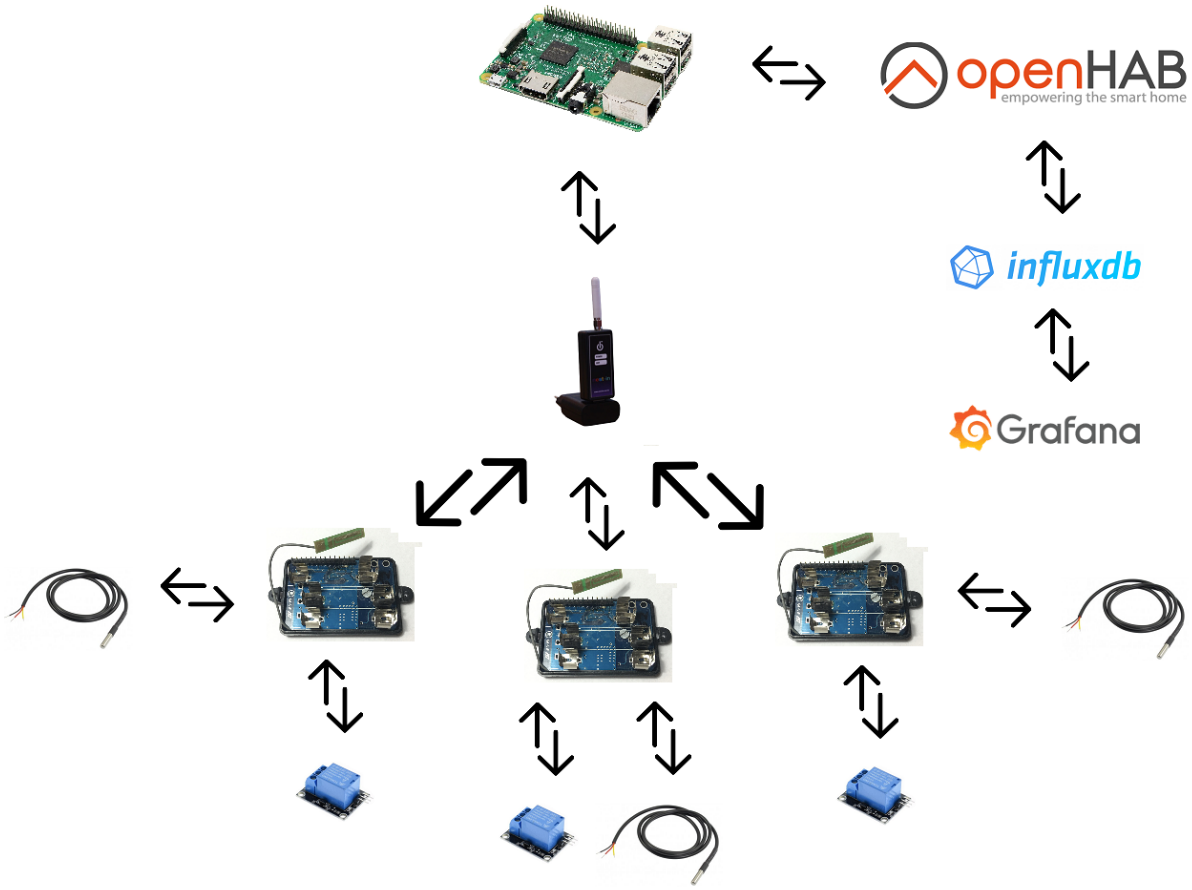

Figura 3.10: Arquitetura geral do sistema de automação proposto.

Capítulo 4

O Protocolo Aéreo

4.1 Introdução

O Protocolo Aéreo é o protocolo de rede proposto nesse trabalho para resolver a comunicação entre os dispositivos, responsáveis pelo sensoriamento e atuação, e os *Gateways*, responsáveis pela comunicação com o sistema central de automação. São diversas as tecnologias possíveis de se utilizar para a comunicação entre os dispositivos e *Gateways* e entre esses e o sistema central de automação, porém a escolha das tecnologias influencia diretamente na arquitetura proposta para o sistema. Por exemplo, ao se utilizar uma tecnologia sem fio de baixo alcance, os dispositivos não podem estar muito distantes uns dos outros, e para cobrir maiores áreas, faz-se necessária a utilização de técnicas para contornar esse problema.

Em [43] há uma comparação entre as diversas tecnologias sem fio, realizadas pela Tabela 4.1, comparando alguns fatores importantes de cada tecnologia. A partir da tabela e do problema de automação predial a ser resolvido, nota-se um destaque positivo na tecnologia LoRa, justificado pelo consumo energético muito baixo, que é importante em dispositivos que serão alimentados por baterias, e alcances enormes, que torna mais flexível as possibilidades de comunicação. A pequena taxa de dados de LoRa, quando comparadas com as outras tecnologias, não se torna grande problema, justificado pela pequena quantidade da dados transitados.

Em relação à comunicação dos *Gateways* com o sistema central de automação, a utilização de WiFi (IEEE 802.11) é justificada pela dispensabilidade de baixos consumos energéticos, visto que o Gateway não é alimentado por baterias, além de uma necessidade de uma maior taxa de dados, para acelerar o processo de comunicação com a central do sistema de automação. Logo, as tecnologias escolhidas para as comunicações sem fio foram LoRa e IEEE 802.11.

A topologia de rede escolhida é a topologia estrela, que consiste de um elemento central, denominado *gateway*, que receberá as requisições dos nós. Os nós não podem se comunicar entre si diretamente, pois toda informação deve primeiramente trafegar pelo *gateway*. Uma imagem que ilustra um exemplo de rede em estrela pode ser visualizado na Figura 4.1.

Devido ao limitado raio de transmissão das redes sem fio, múltiplos saltos (*hops*) podem ser

Tabela 4.1: Comparação entre algumas tecnologias de comunicação.

Tecnologia	WiFi	WiMAX	LR-WPAN	Comunicação Mobile	Bluetooth	LoRa
Norma	IEEE 802.11 a/c/b/ d/g/n	IEEE 802.16	IEEE 802.15.4 (ZigBee)	2G-GSM, CDMA 3G-UMTS, CDMA2000 4G-LTE	IEEE 802.15.1	LoRaWAN R1.0
Banda de frequência	5–60 GHz	2–66 GHz	868/915 MHz, 2.4 GHz	865 MHz, 2.4 GHz	2.4 GHz	868/900 MHz
Taxa de dados	1 Mb/s–6.75 Gb/s	1 Mb/s–1 Gb/s (Fixed) 50–100 Mb/s (mobile)	40–250 Kb/s	2G: 50–100 kb/s 3G: 200 kb/s 4G: 0.1–1 Gb/s	1–24 Mb/s	0.3–50 Kb/s
Alcance	20–100 m	<50Km	10–20 m	Zona de telefonia móvel	8–10 m	<30 Km
Consumo Energético	Alto	Médio	Baixo	Médio	Bluetooth: Médio BLE: Muito baixo	Muito baixo
Custo	Alto	Alto	Baixo	Médio	Baixo	Alto

necessários para efetuar a troca de dados entre os nós da rede. Algumas tecnologias, como *Zigbee*, que possuem um raio de transmissão pequeno, justificam a utilização de topologias de rede que realizam *multi-hop*. Contudo, o uso da topologia estrela nesse trabalho é justificado pelo grande alcance que a modulação LoRa nos provê, além de uma maior simplicidade de implementação, quando comparadas, por exemplo, com redes *mesh* (que podem ser *multi-hop*). Um exemplo de redes *mesh* podem ser visualizadas na Figura 4.2. Note que não há um nó central, diferentemente da topologia em estrela, e cada um dos nós pode se comunicar com os outros, contanto que não haja limitações provenientes do raio de transmissão.

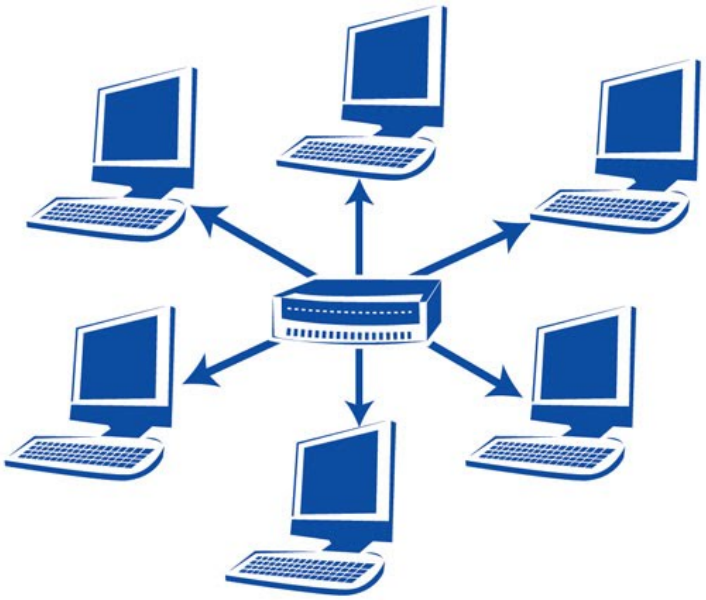

Figura 4.1: Topologia de rede em estrela. Nesse exemplo, o roteador atua como *gateway* e os computadores como nós da rede [63].

Visto isso, o fluxo de dados idealizado quando um dispositivo deseja enviar informações é: quando um nó deseja realizar o envio das informações para a Internet, o mesmo, por meio da modulação LoRa, se comunica com o *gateway*, regido por um conjunto de regras do protocolo. Ao receber as informações, o *gateway* realiza o envio destas para um servidor de rede, utilizando

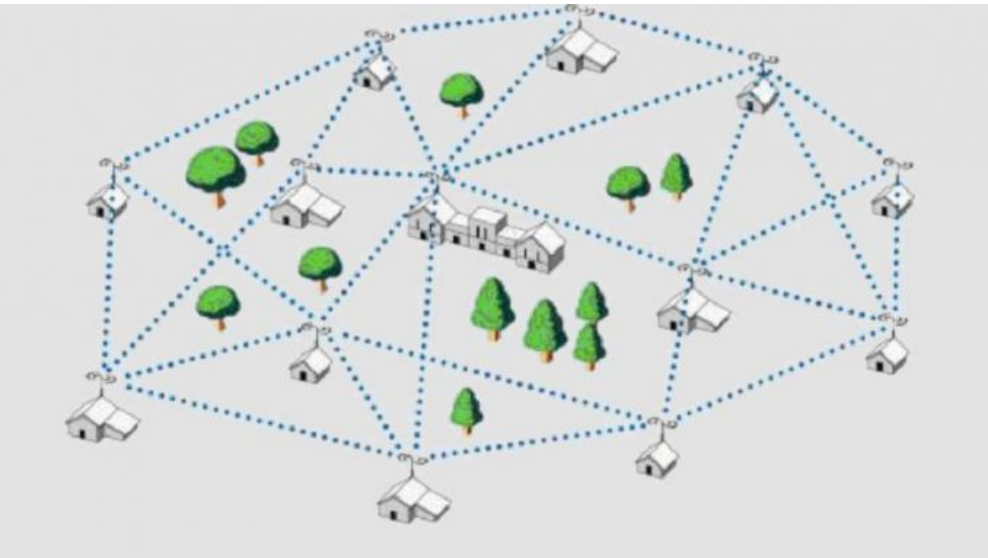

Figura 4.2: Exemplificação de uma Rede mesh. Note que mesmo que não haja comunicação direta entre todos os nós, é possível realizar a comunicação entre quaisquer nós, utilizando-se de múltiplos saltos [64].

IEEE 802.11 e o protocolo MQTT. Ao receber a confirmação de envio pelo servidor de rede, além de uma resposta de retorno para que o servidor possa se comunicar com o dispositivo, o *gateway* realiza o envio do retorno de rede para o dispositivo, encerrando a comunicação. O fluxo dos dados pode ser exemplificado pela Figura 4.3.

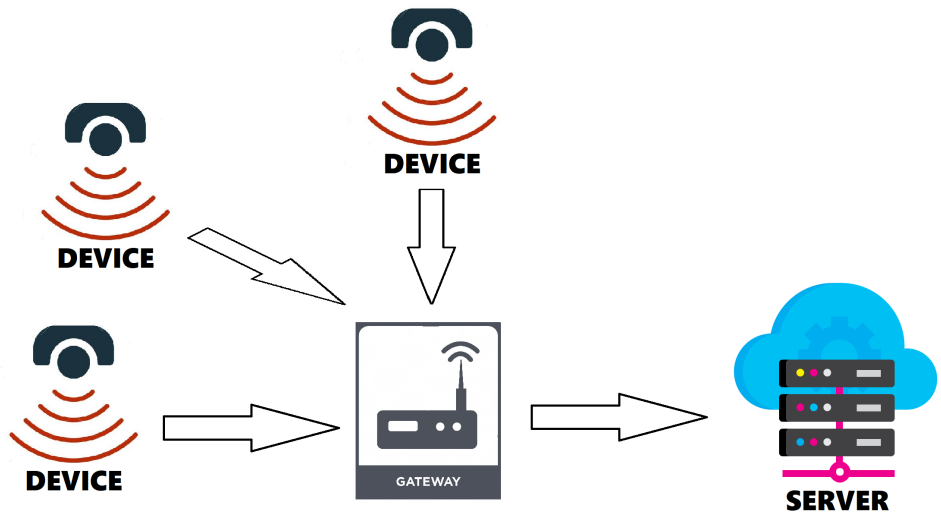

Figura 4.3: Exemplificação do fluxo de dados simplificado. Não é representado o retorno de informações do servidor de rede para os dispositivos.

Levando em conta a arquitetura geral do sistema proposto na Figura 3.10, o Protocolo Aéreo é responsável por uma fração do sistema total. Essa parcela pode ser visualizada na Figura 4.4.

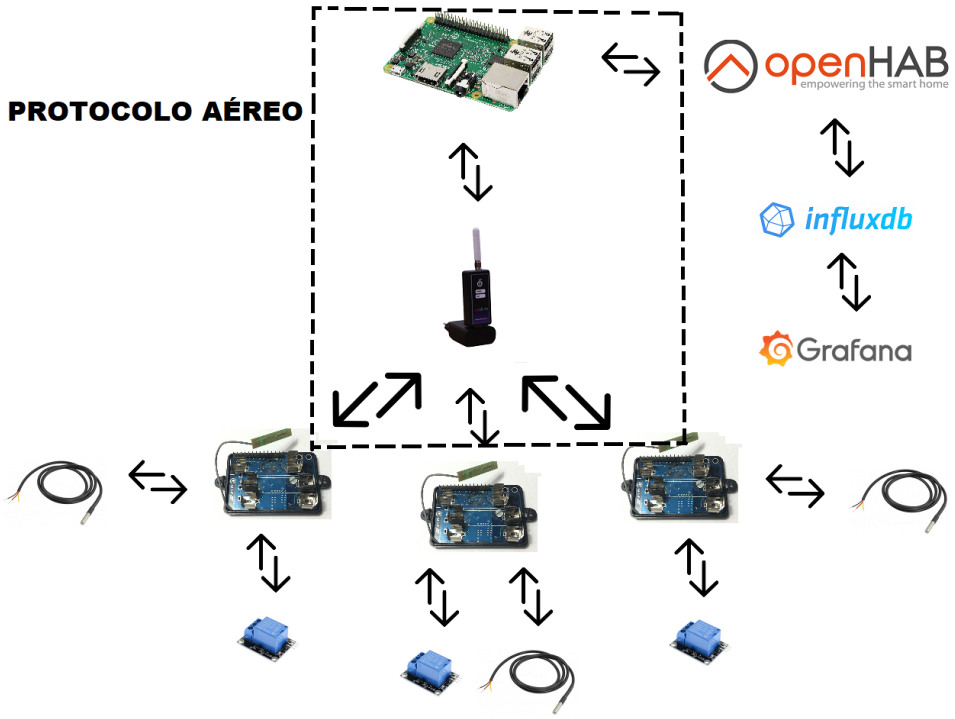

Figura 4.4: Fração do sistema de automação proposto regido pelo protocolo aéreo.

4.2 Arquitetura do Protocolo

O protocolo tem como objetivo realizar o envio de informações dos dispositivos (*end-devices*) até o sistema central de automação. Para isso, criou-se um conjunto de regras e formatos de mensagens. O protocolo possui uma máquina de estados para o dispositivo e uma para o *gateway*, que possuem três grandes funcionalidades:

- Teste de rede: tem como objetivo verificar se há cobertura na região, em outras palavras, se há um *gateway* próximo em que seja possível realizar o envio de informações;
- Envio de dados: tem como objetivo realizar os envios das informações para a central;
- Envio de dados prioritário: é uma variação da funcionalidade de envio, onde as regras de comunicação são diferentes, visando a agilidade no envio das informações. Geralmente, constitui-se por dados mais sensíveis ao tempo de resposta, como por exemplo um botão de pânico.

Cada uma das funcionalidades serão explicadas detalhadamente a seguir nas seções 4.2.2, 4.2.3, 4.2.4, e o comportamento tanto de um dispositivo quanto do *gateway*, em relação às máquinas de estados, podem ser visualizadas na Figuras 4.5 e 4.6.

Por fim, será necessário entender a estrutura e a formatação das mensagens antes de serem explicadas as três funcionalidades. Essa estrutura será explicada na Seção 4.2.1.

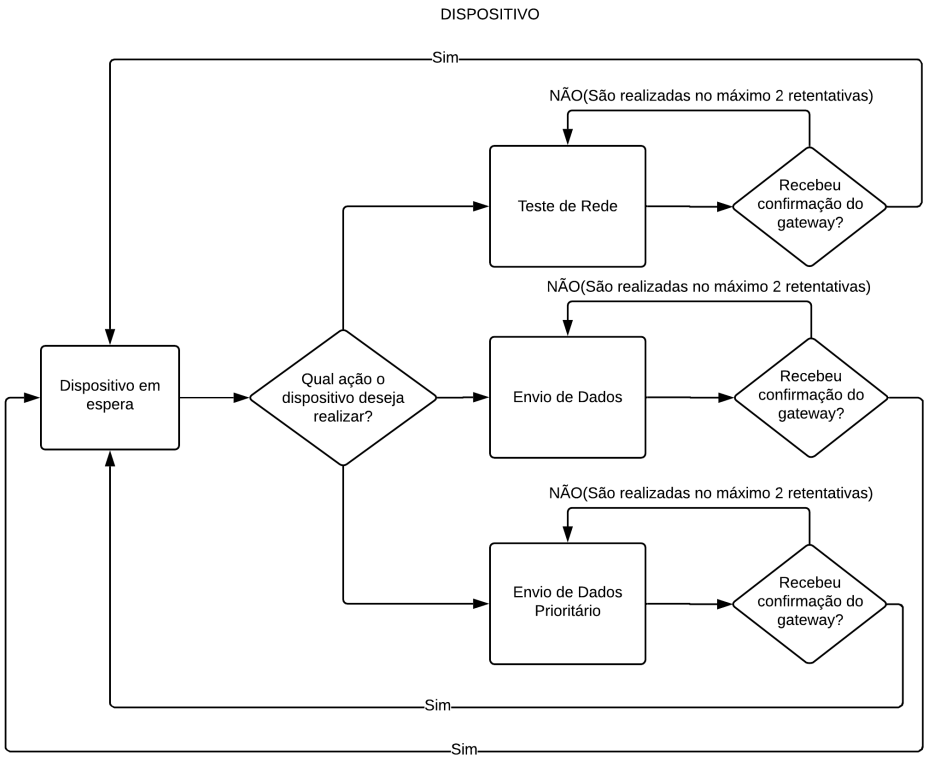

Figura 4.5: Máquina de Estados dos dispositivos simplificada com as 3 funcionalidades.

4.2.1 Formatação das mensagens

Toda as mensagens que são trocadas entre os dispositivos e o *gateway*, por meio da modulação LoRa, seguem um formato específico de modo a padronizar o envio das informações. As mensagens enviadas são dividas em dois blocos: *Header* e *Payload*. O *Header* tem como objetivo conter as informações necessárias para validar e garantir que a comunicação seja realizada com sucesso enquanto o *Payload* carrega a informação que se deseja transmitir. O *Header* é um campo obrigatório para toda comunicação e o *Payload* não, como será visto mais a diante.

É feita uma divisão em mensagens de *Uplink* e *Downlink*, onde *Uplink* representam as mensagens que são transmitidas do dispositivo para o *gateway* e *Downlink* as mensagen que são transmitidas do *gateway* para o dispositivo. A disposição dos elementos constituintes do *Header* pode ser visualizada na Figura 4.7. Uma descrição mais detalhada de cada um dos elementos será realizada a seguir:

- APP_ID: Application Identification é um número de 5 bytes que tem como função identificar para qual aplicação deve ser encaminhada a mensagem no servidor. Como meio de exemplificação, todos os dispositivos responsáveis pela medição de temperatura de um prédio terão o mesmo APP_ID, pois serão regidos pelos mesmos conjuntos de regras no servidor.
- DEV_ID: Device Identification é um número de 5 bytes, único para cada dispositivo, que

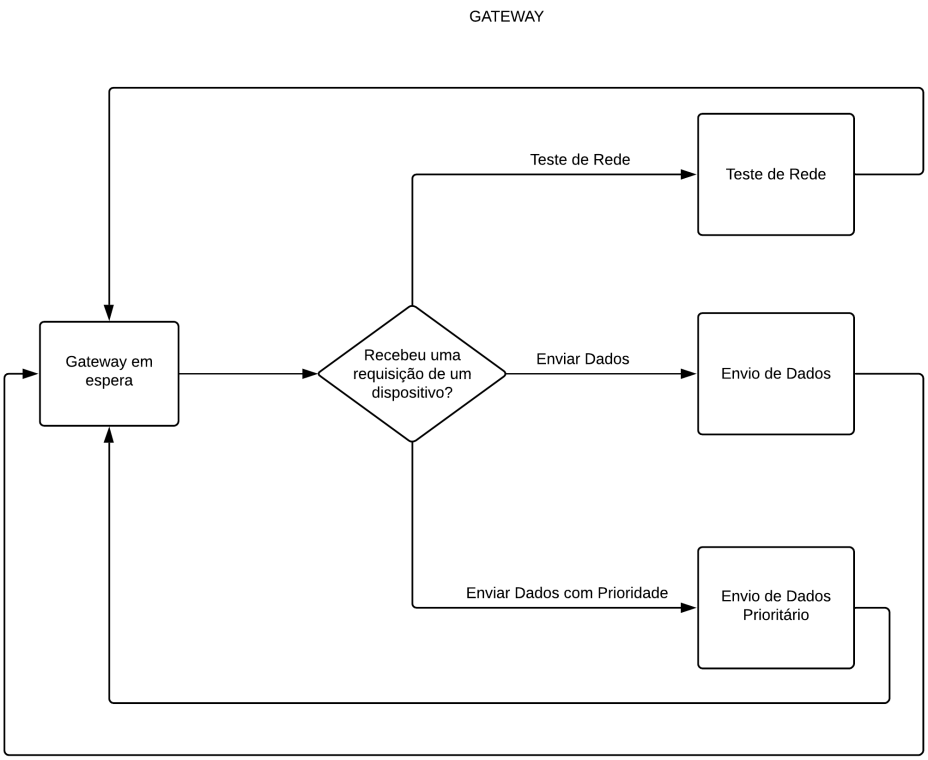

Figura 4.6: Máquina de Estados do gateway simplificada com as 3 funcionalidades.

tem como função identificar o dispositivo.

- MAC: *Message Authentication Code* é um número de 8 bytes responsável por garantir a integridade dos dados trafegados, além de segurança para o sistema. O MAC é gerado por meio do algoritmo de criptografia AES-128, utilizando-se dos dados contidos no *Header*. Ao receber uma mensagem, tanto de *Uplink* como de *Downlink*, é verificado se o MAC é válido, e caso não seja, o fluxo dos dados é interrompido, pois a mensagem não é válida.

- SEQ_ID: *Sequence Identification* é um número de 2 bytes que tem como objetivo garantir o fluxo sequencial dos dados. Por exemplo, ao receber um dado, o *sequence identification* garante que os dados enviados possuam uma cronologia correta.

- TYPE: *Type* é um número de 4 bits (meio byte), que se encontra nos bits mais significativos do byte em que está contido. Tem como função identificar o tipo da mensagem enviada, pois cada tipo possui algumas particularidades em relação ao *Header* e a presença, ou não de *Payload*. Os types existentes são: Request to Send(RTS), Clear to Send(CTS), Data, Acknowledge(ACK), Priority, Denial of Service(DOS) e Test. Na Tabela 4.2, se encontram os tipos existentes, além de suas funcionalidades.

- CHANNEL: Channel é um número enviado que varia de 1 a 8, que identifica em qual canal o dispositivo (Se for um *Uplink*) ou o *gateway* (Se for um Downlink) está esperando a próxima mensagem da comunicação. Os canais estão relacionados com as frequências da modulação

Tabela 4.2: Identificadores do tipo de mensagem.

Tipos	**Código (Hexa)**	**Funcionalidade**
Request to Send (RTS)	0x01	Quando um dispositivo deseja comunicar na rede, é feita a requisição para comunicar.
Clear to Send (CTS)	0x02	Quando é realizada a requisição e a rede se encontra disponível para comunicação.
Acknowledge (ACK)	0x03	Confirmação de entrega do pacote no servidor.
Data	0x04	Envio do dado
Priority	0x07	Pacote prioritário na rede sendo encaminhado.
Denial of Service (DOS)	0x0E	Quando é realizada a requisição e a rede se encontra indisponível para comunicação.
Test	0x0F	Realização de um teste na cobertura da rede para verificar a existência de conectividade.

LoRa em que se deseja comunicar, sendo necessário o dispositivo e o gateway estarem no mesmo canal para que haja a troca de informações. No Brasil, a frequência em que LoRa pode ser utilizada, definida pela Agência Nacional de Telecomunicações (ANATEL), é na faixa de 915 MHz a 928 MHz. Os canais e as frequências utilizadas são definidas nas Tabelas 4.3 e 4.4. As frequências foram definidas de modo a evitar sobreposição dos espectros dos sinais.

Tabela 4.3: Canais de uplink disponíveis.

Canais Uplink	1	2	3	4	5	6	7	8
Frequência (MHz)	916,8	917,0	917,2	917,4	917,6	917,8	918,0	918,2

Tabela 4.4: Canais de downlink disponíveis.

Canais Downlink	1	2	3	4	5	6	7	8
Frequência (MHz)	923,3	923,9	924,5	925,1	925,7	926,3	926,9	927,5

- *Received signal strength indication* (RSSI): RSSI é um número que mede a intensidade do sinal recebido na comunicação, sendo uma medida da potência presente em um sinal de rádio recebido. Mede o quão bom um determinado rádio pode ouvir as informações de

outros rádios. RSSI é medido em decibéis miliwatt e quanto mais próximo de zero for, mais forte o sinal.

- *Cyclic Redundancy Check* (CRC): CRC é um código verificador que tem como função garantir a integridade do *Payload* de *Downlink*
- *Payload*: *Payload* é a informação útil que se deseja enviar para o servidor ou para o dispositivo. Possui o tamanho de 0 a 16 bytes se for um *payload* de *Uplink*, ou 0 a 8 bytes se for um *payload* de *Downlink*. Só está contida em mensagens do tipo Data, Priority e Acknowledge.

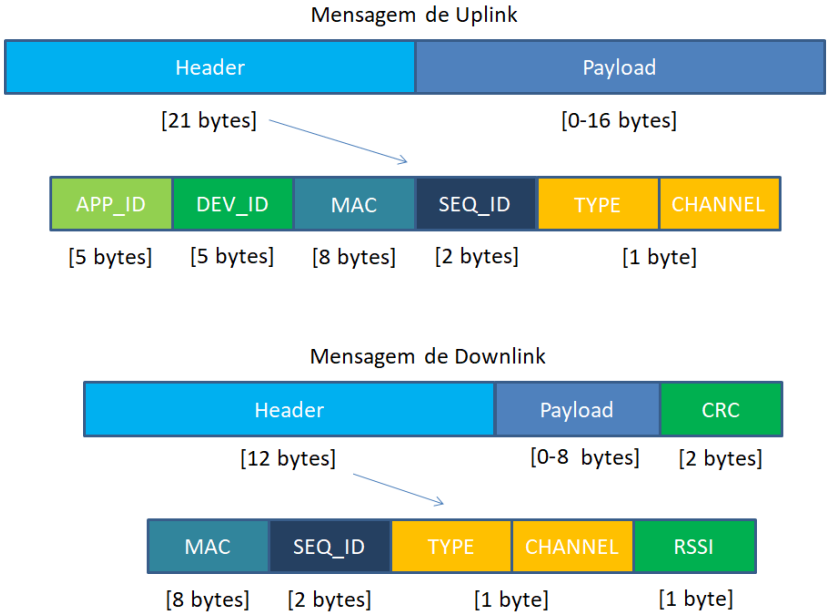

Figura 4.7: Formatação das mensagens de *Uplink* e de *Downlink*.

4.2.2 Teste de rede

O teste de rede tem a simples funcionalidade de verificar se há cobertura no local onde o dispostivo se encontra, ou seja, se há um *gateway* próximo e se o servidor de rede se encontra disponível. Para isso, o dispositivo envia um pacote, via modulação LoRa para o *gateway*. Como o dispositivo está enviando um pacote para o *gateway*, é uma mensagem de *Uplink*. O campo Type será preenchido com o código 0x0F(Test), que é o código que identifica que o pacote enviado para o *gateway* é um teste de rede e o campo Channel será preenchido com um número de 1 a 8, aleatoriamente definido, que contem a informação de em qual canal o dispositivo está esperando a resposta do *gateway*. Os campos APP_ID e DEV_ID são preenchidos, de acordo com o dispositivo, o código sequencial SEQ_ID é preenchido de acordo com o último SEQ_ID computado adicionado de um e o MAC é gerado utilizando o algortimo AES-128 com a entrada sendo os campos do Header.

Esse pacote, ao ser recebido por um *gateway*, é verificado que o Type é um Test(código 0x0F), então deve executar a rotinas que irão testar o servidor de rede. Antes de fazer essas rotinas,

verifica a validade e integridade dos dados por meio do algoritmo AES-128 com os dados do *Header*, e compara com o campo MAC. Se o resultado do algoritmo AES-128 em cima dos dados do *Header* for igual ao campo MAC, isso garante que os dados estão corretos, e se não estiverem iguais, os dados recebidos não são válidos. Após essa validação, é realizado um *ping* no servidor, para verificar seu status. Se o servidor estiver online, será enviado um Acknowledge(código 0x03), caso não esteja online, será enviado um Denial of Service(código 0x0E). O *gateway* envia a resposta, via modulação LoRa para o dispositivo, na frequência do canal sorteado pelo dispositivo, que está no campo Channel da mensagem de *Uplink*. Como o *gateway* está enviando um pacote para o dispositivo, é uma mensagem de *Downlink*. O campo Type será preenchido com o código 0x03(ACK) ou 0x0E(DOS), a depender se o servidor está online ou não. O campo Channel será preenchido com um número de 1 a 8, randomicamente definido, que contêm a informação de em qual canal o *gateway* está esperando a resposta do dispositivo. Como após o Acknowledge ou Denial of Service, o dispositivo não enviará mais resposta para o *gateway*, essa informação não será utilizada, mas deve ser preenchida pelas regras do protocolo. O código sequencial SEQ_ID é preenchido de acordo com o último SEQ_ID computado adicionado de 1 e o MAC é gerado utilizando o algortimo AES-128 com a entrada sendo os campos do *Header*. O fluxo da funcionalidade de teste de rede pode ser visualizado na Figura 4.8, e a máquina de estados que representa as ações do dispositivo na Figura 4.9.

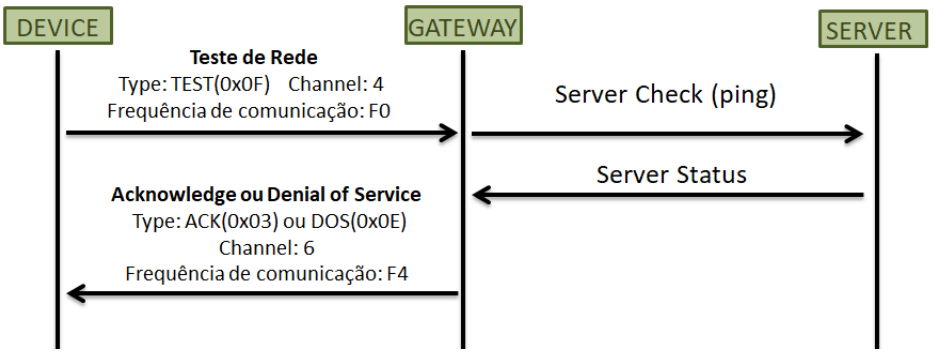

Figura 4.8: Modo de funcionamento do teste de rede.

4.2.3 Envio de Dados

Essa funcionalidade é a mais importante do protocolo. Com ela, é possível enviar os dados dos dispositivos para o servidor de rede. O dispositivo, para realizar o envio dos dados, deve seguir o seguinte passo a passo: Primeiramente, deve enviar um pacote para o gateway, denominado Request to Send. Para isso, deve criar um pacote com um Header como em qualquer envio de pacote, preenchendo seu APP_ID, DEV_ID, código sequencial SEQ_ID, sortear um canal em que a resposta do gateway será escutada. O campo Type será preenchido com o código 0x01(RTS), que é o código que identifica que o pacote enviado para o gateway é uma requisição para enviar dados para o servidor. Além disso, como em toda comunicação, é gerado um MAC com a entrada sendo os dados do Header para garantir a integridade dos dados.

Esse pacote, ao ser recebido pelo gateway é verificado que o Type é um Request to Send(código

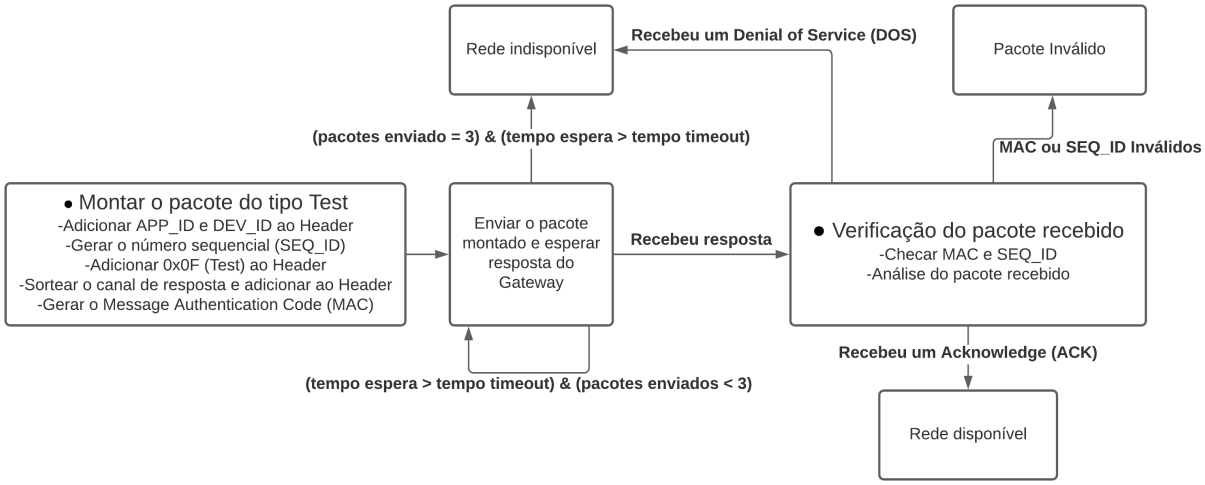

Figura 4.9: Máquina de Estados que representa o envio de um teste de rede, pela perspectiva do dispositivo.

0x01), então deve fazer as rotinas para verificar se os dados podem ser enviados para o servidor. Muito similar a quando se recebe um Test, quando se recebe um RTS, deve-se verificar se o servidor se encontra disponível. Além de todas as verificações, que são comuns a todos os pacotes enviados, após realizar as mesmas e verificar o status do servidor, monta um pacote de Clear to Send(código 0x02 no campo Type), caso o servidor esteja online ou um Denial of Service(DOS), caso o servidor esteja offline. O pacote de resposta será enviado na mesma frequência do canal sorteado pelo dispositivo, que foi enviado no pacote de RTS no campo Channel. A partir desse momento, todas as mensagens, tanto de Uplink quanto de Downlink serão transmitidas nesse canal sorteado.

Quando o dispositivo recebe o pacote de Clear to Send, realiza as verificações de integridade da mensagem, e posteriormente monta um pacote com Type DATA(código 0x04) e com todos os outros campos de Header devidamente preenchidos. Diferentemente dos outros pacotes, esse possui além de um Header, um Payload de até 16 bytes, contendo as informações úteis que se deseja enviar para o servidor de rede. Esse pacote é enviado para o gateway no mesmo canal em que ocorreu a última transmissão.

Ao receber o pacote DATA(0x04), o gateway realiza as verificações padrões e se prepara para enviar os dados para o servidor de rede. Esse envio para o servidor de rede será feito por meio do protocolo de rede MQTT, sendo necessário o endereço de Internet Protocol (IP), a porta do broker MQTT e as credenciais para autenticação. Os dados enviados para o servidor são: App_ip, Dev_id, o RSSI do último pacote recebido, a Relação sinal-ruído, do inglês Signal-to-Noise Ratio(SNR) do último pacote recebido e o *payload*. Todas essas informações são escritas em tópicos MQTT e podem ser acessadas por qualquer dispositivo conectado no *broker*.

Os dados serão processados no servidor, e será enviado uma resposta do servidor de rede para o gateway, que deverá reencaminhar essa resposta para o dispositivo. Essa resposta do servidor de rede, será denominada payload de Downlink, possuindo tamanho variável de 0 a 8 bytes. Além disso, para garantir a corretude do payload de Downlink, são gerados 2 bytes de

CRC, que são enviados para o gateway, para que o mesmo possa verificar a integridade das informações transitadas por MQTT. Por fim, é enviado um pacote de Acknowledge(código 0x03) para o dispositivo, que terá as informações de Header devidamente preenchidos, como nos outros pacotes, junto com o payload de Downlink e o CRC.

Com isso, todo o fluxo de envio de dados é finalizado, e para facilitar a compreensão é possível visualizar um esquemático na Figura 4.10 que exemplifica toda a comunicação. A máquina de estados que representa as ações do dispositivo no envio de dados pode ser visualizado na Figura 4.11.

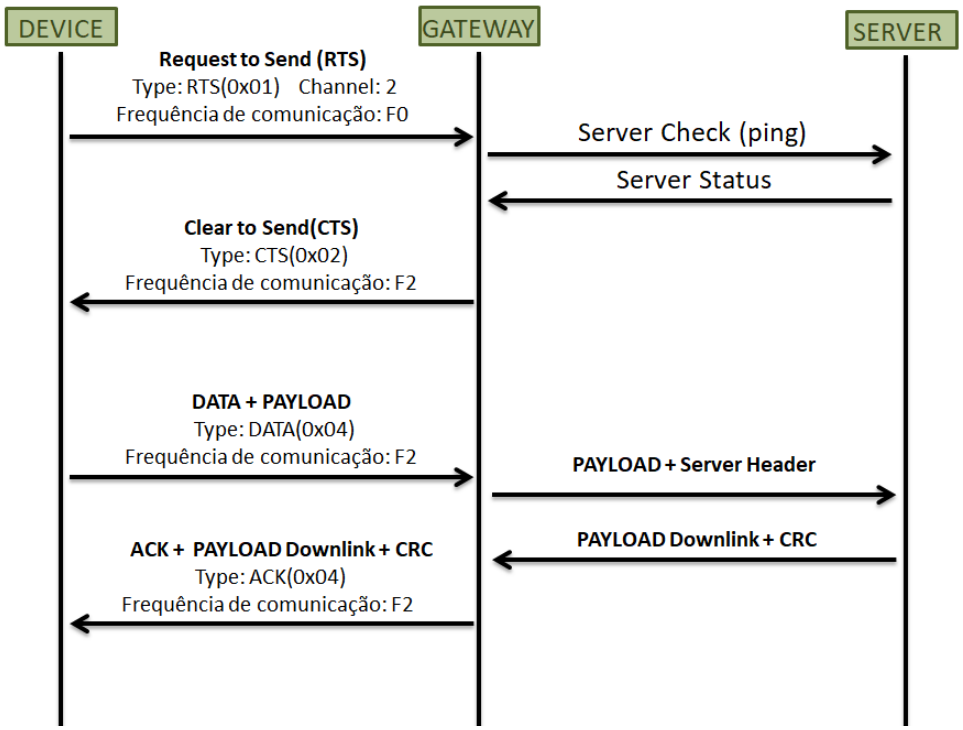

Figura 4.10: Modo de funcionamento do envio de dados.

4.2.4 Envio de Dados Prioritário

Por fim, essa funcionalidade do protocolo tem a função de enviar os dados, assim como a funcionalidade de Envio de Dados, com algumas pequenas diferenças para agilizar o processo de envio das informações. Essa funcionalidade é útil quando se possuem dispositivos com menor capacidade de bateria ou quando a agilidade do envio das informações é crítico. A principal diferença em relação ao Envio de Dados comum, é que não é enviado um pacote requisitando a comunicação e após a confirmação, enviado os dados. O envio prioritário envia um pacote com Type Priority(código 0x07) e envia juntamente o payload desejado. Quando o gateway recebe o pacote prioritário, ele envia o payload para o servidor de rede. Ele se assemelha ao envio de dados, sendo que as etapas de Request to Send e Clear to Send são descartadas. O esquemático que representa o envio de dados prioritário pode ser visualizado na Figura 4.12. A máquina de estados que representa as ações do dispositivo no envio prioritário pode ser visualizado na Figura

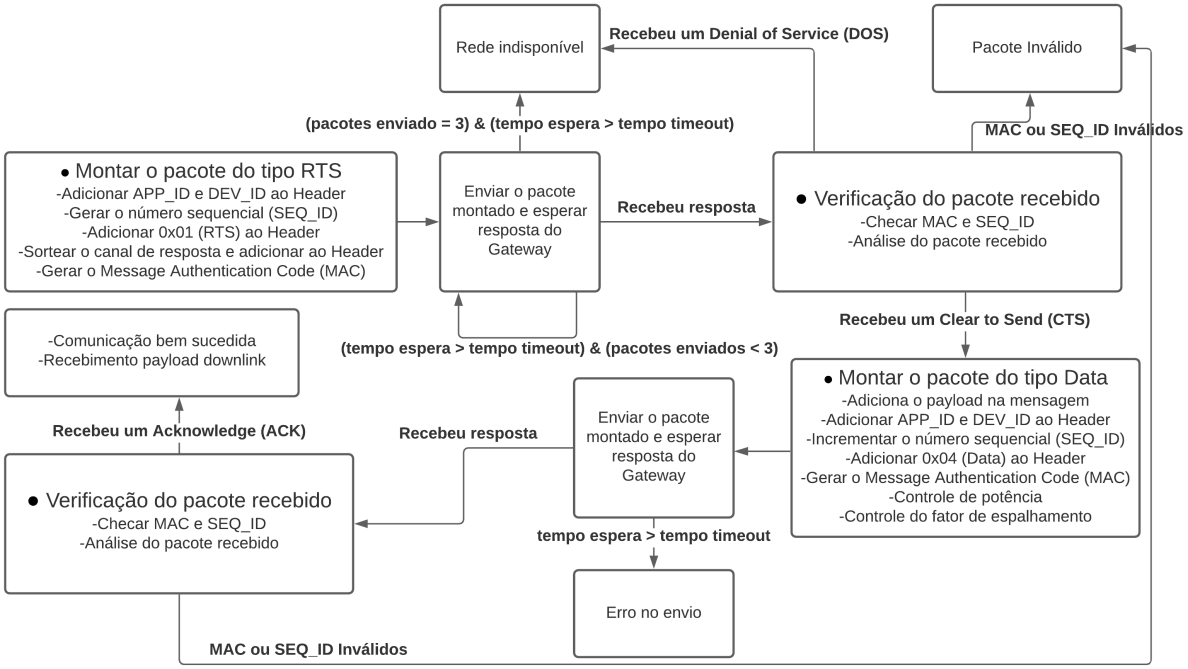

Figura 4.11: Máquina de Estados que representa o envio de um envio de dados, pela perspectiva do dispositivo.

4.13.

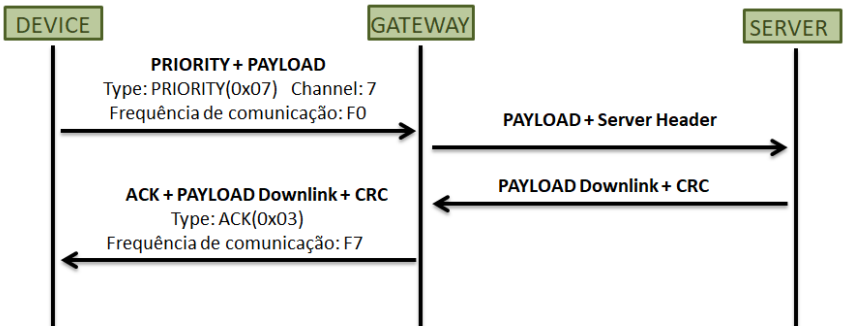

Figura 4.12: Modo de funcionamento do envio de dados prioritário.

4.3 Segurança e integridade do Dados

Em um sistema de internet das coisas, dois dos pontos principais a serem levados em consideração são a segurança da rede e a confiabilidade nas informações trafegadas. Visto isso, uma estratégia idealizada para garantir a segurança da rede, além da confiabilidade dos dados enviados foi a de utilizar o algoritmo de criptografia AES-128. AES é um algoritmo de criptografia de chave simétrica, ou seja, a mesma chave é usada para cifrar e decifrar os dados. O algoritmo utiliza uma chave de 128 bits para criptografar/descriptografar os dados e 128 bits de entrada.

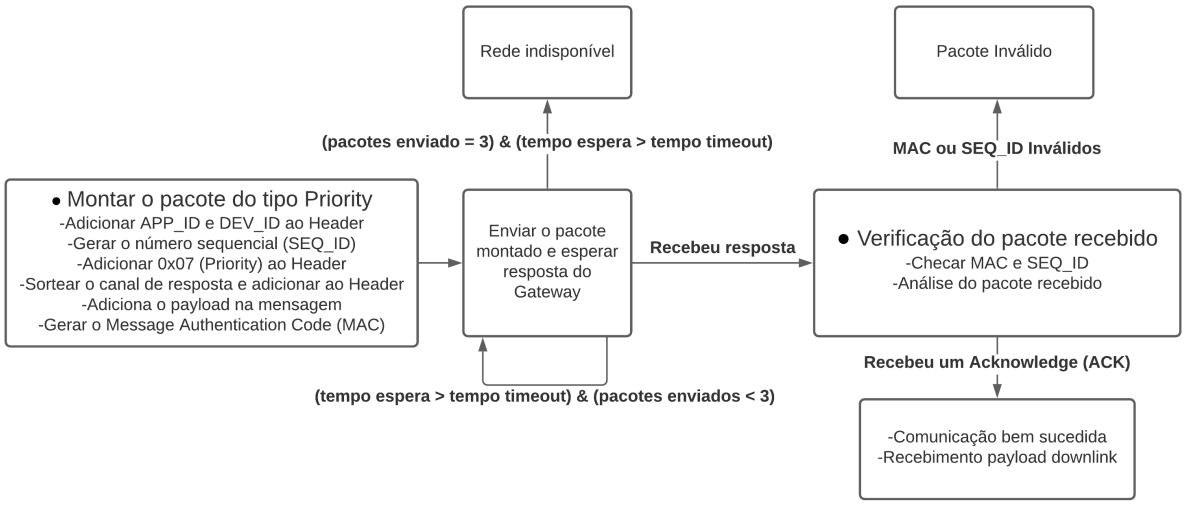

Figura 4.13: Máquina de Estados que representa o envio de um envio de dados prioritário, pela perspectiva do dispositivo.

A estratégia é a utilização dos dados do Header como entrada para o algoritmo. No caso de Uplink, o Header é constituído pelos campos App_id, Dev_id, Seq_id, type e channel, totalizando 13 bytes de informação. Como o algoritmo espera 16 bytes de entrada, os 3 bytes finais são completados com números 0x00. Como saída, o algoritmo AES-128 irá gerar uma palavra de 16 bytes. Desses 16 bytes, os 8 primeiros bytes serão utilizados para compor o campo MAC(Message Authentication Code) de qualquer pacote criado pelo protocolo. Com isso, ao receber um pacote, o dispositivo ou gateway realizam a mesma operação, utilizando como entrada para o algoritmo o Header recebido. Os 8 primeiros bytes da saída do algoritmo serão comparados com o campo MAC e se forem iguais, a mensagem é válida. No caso de Downlink, o Header possui somente 4 bytes, então os 12 bytes finais são completados com números 0x00. Essa validação só é possível, pois o dispositivo e o gateway possuem a mesma chave de criptografia.

Como estratégia para aumentar a segurança da rede, foi criado ainda um mecanismo em que são conhecidas 10 chaves de criptografia no gateway e no dispositivo, e que a chave para realizar a criptografia é sorteada aleatoriamente cada vez em que é realizado a geração do campo MAC. Com isso, na hora de validar as informações, deve-se testar para cada uma das chaves conhecidas, até que o resultado seja válido. Se nenhuma das 10 chaves gerar um resultado válido, a informação transitada é inválida.

Visto isso, além de garantir a integridade dos dados transitados, pois podem ocorrer erros na hora em que são transmitidos, é criada uma proteção para que somente dispositivos que conheçam a regra de criação do campo MAC, além das senhas utilizadas na criptografia, consigam utilizar a infraestrutura da rede. Além disso, impede que mensagens que tenha tido o seu conteúdo corrompido, sejam transitadas pela rede de forma errônea.

4.4 Controle de Potência

Outro fator importante em um sistema de Internet das Coisas é o consumo energético. Isso é verdade, pois diversos dispositivos que constituem o sistema, serão alimentados por bateria. Como se deseja minimizar a manutenção, como a troca de baterias, faz-se necessário uma eficiência energética maior. Para isso, idealizaram-se algumas estratégias para reduzir o consumo energético dos dispositivos. Para calcular o consumo energético do rádio que realiza a modulação LoRa, utilizou-se uma calculadora disponibilizada pela Semtech, fabricante de rádios LoRa. O nome da calculadora é *LoRa Modem Calculator Tool*, e pode ser obtida no site da Semtech.

A partir da configuração dos parâmetros, percebeu-se que o parâmetro que mais influencia o consumo energético é a potência de transmissão, que pode ser variada entre +5dBm e +17dBm. O consumo energético para uma transmissão com potência +5dBm é de 25mA, enquanto o consumo para uma transmissão de +17dBm é de 90mA. Outro ponto importante, é o consumo do rádio quando em modo Sleep, que representa apenas 100nA de consumo. Logo, uma boa estratégia é colocar o rádio em modo Sleep após finalizar as transmissões.

Além disso, será controlada a potência de transmissão do dispositivo a depender do RSSI recebido pelo gateway. A informação de RSSI do gateway é enviada nos pacotes de Downlink. Assim, o controle de potência só será realizado após receber um pacote do gateway e o dispositivo ainda for se comunicar com o gateway. Nesse caso, somente no Envio de dados, mais especificamente no momento de enviar o pacote Data, que será realizado o controle de potência a partir do RSSI. Convencionou-se sempre as primeiras transmissões ocorrem com máxima potência (+17dBm) e a partir do RSSI, realizar o controle da potência. As regras utilizadas para o controle podem ser visualizadas abaixo.

Criaram-se três valores de potência desejados e intervalos em que se deve utilizar cada uma das potências. De acordo com a calculadora, transmitir a +5dBm e a +8dBm gera o mesmo consumo energético, assim, escolheu-se +8dBm para ser um valor de potência desejado. Outro valor de potência escolhido é +17dBm. Por fim, escolheu-se o valor de +14dBm para ser o último valor de potência. A potência de +8dBm foi escolhida para ser a menor potência, devido ao seu pequeno consumo energético em transmissão, por volta de 25mA, que será utilizada quando o sinal estiver muito forte. A potência de +17dBm é a máxima potência possível, e com o maior consumo energético (90mA), por isso, só é utilizada nos casos em que o sinal está muito fraco. Por fim, a potência de +14dBm foi escolhida para ser uma potência intermediária entre as potências, pois o rádio, segundo a fabricante Semtech, possui a melhor eficiência energética vs alcance na potência de +14dBm.

Os intervalos para cada valor de potência são: Se RSSI > -96, a potência utilizada será +8dBm. Se RSSI está entre -95 e -104, a potência utilizada será +14dBm. Por fim, se RSSI < -103, a potência utilizada será +17dBm.

4.5 Controle do fator de espalhamento

O fator de espalhamento é uma variável importante quando se pensa em comunicação LoRa. Essa importância se deve, principalmente à relação com o fator de espalhamento utilizado, o alcance obtido e a taxa de transferência dos dados. De modo geral, o aumento do fator de espalhamento, que varia de 7 a 12, gera um aumento no alcance das transmissões, porém diminui as taxas de transmissão, aumentando o tempo necessário para se enviar os dados. Outro ponto importante, se encontra na propriedade da tecnologia LoRa, em que sinais com diferentes fatores de espalhamento são ortogonais entre si, diminuindo as colisões entre os sinais.

O controle do fator de espalhamento é uma ferramenta muito importante, pois como os sinais em fatores de espalhamento diferentes são ortogonais entre si, ao variar o parâmetro, diminuem as chances de colisões entre dispositivos que se encontram na mesma rede. Além disso, os tempos de transmissão são diretamente impactados com a alteração desse fator. Essa alteração pode ser visualizada na Tabela 4.5. Os tempos de envio de cada um dos tipos de pacote do protocolo, serão obtidos a partir da calculadora *LoRa Modem Calculator Tool*. Os seguintes parâmetros foram utilizados: a largura de banda em 125kHz e Coding Rate em 4. O tamanho do pacote e o fator de espalhamento foram alterados.

Tabela 4.5: Influência do fator de espalhamento e tamanho dos pacotes no tempo de transmissão.

Tipo do pacote	Tamanho do pacote	Fator de espalhamento	Tempo de transmissão(ms)	Bitrate (bps)
Uplink sem payload	21	7	67,84	3417,97
Uplink sem payload	21	10	411,65	610,35
Uplink com payload	37	7	108,8	3417,97
Uplink com payload	37	10	608,26	610,35
Downlink sem payload	12	7	51,46	3417,97
Downlink sem payload	12	10	280,58	610,35
Downlink com payload	22	7	76,03	3417,97
Downlink com payload	22	10	411,65	610,35

Visto isso, utilizando a mesma estratégica utilizada no controle de potência, será variado o fator de espalhamento com base no RSSI do gateway. Arbitrou-se que se o RSSI for menor que -110, ou seja, o sinal recebido é muito ruim, o fator de espalhamento utilizado será o fator de espalhamento 10. Caso seja maior que -110, o fator de espalhamento utilizado será 7. A escolha do fator de espalhamento em 10, ao invés de utilizar 11 ou 12, está relacionado com os altos tempos de transmissão ao utilizar o fator de espalhamento 11 ou 12. Para fins de comparação, ao se modular 21 bytes, com bandwidth de 125kHz, Coding Rate 4 e fator de espalhamento 10, o tempo de transmissão teórico é de 411,65ms. Ao se alterar para o fator de espalhamento 11, sem alterar os outros fatores, o tempo de transmissão teórico se torna de 823,3 ms, o que tornaria a comunicação muito lenta.

4.6 Mudança de frequências

A frequência em que as informações são moduladas é um parâmetro importante em telecomunicações. Se existirem 2 ou mais dispositivos em uma mesma localidade, tentando se comunicar com o gateway, uma possível interferência que impeça a comunicação é um problema a ser contornado. Também pensando em expansão de rede, se existirem 2 gateways próximos, um não pode interferir nas funcionalidades do outro. Para resolver esses problemas, concebeu-se possíveis soluções, que envolvem modificar as frequências de comunicação. Foram idealizados 3 modos de mudança da frequência e que serão testadas para se verificar a eficiência.

- Frequência fixa: consiste em se fixar um canal de comunicação no início da comunicação, e ser mantido esse canal durante todos os subsequentes pacotes enviados. Aparentemente é uma estratégia em que se é suscetível a interferências de outros dispositivos, pois não há mudança na frequência.

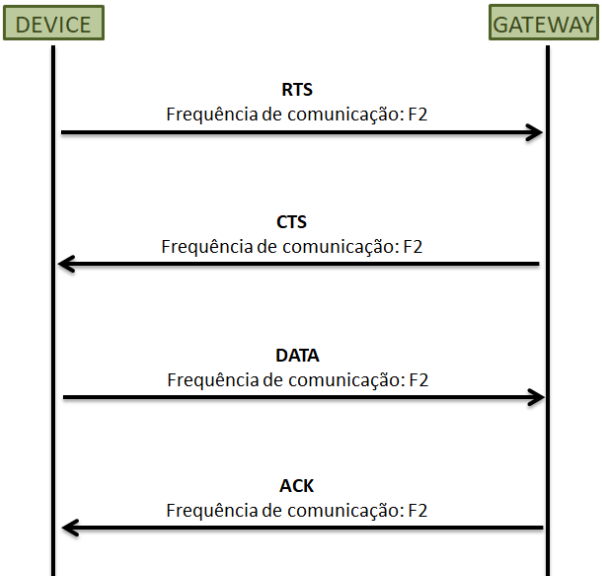

Figura 4.14: Comunicação com frequência fixa.

- Frequência randômica: consiste em se variar a frequência em todas os pacotes enviados. O canal inicial deve ser comum tanto ao gateway quanto ao dispositivo, como em qualquer um dos três tipos de mudança de frequência, porém o canal da próxima comunicação sempre é sorteado e enviado no pacote. Por exemplo, o dispositivo envia um pacote RTS para o gateway e dentro desse pacote, envia o canal do próximo pacote. Quando o gateway for responder no canal de resposta recebido, ele sorteia o canal da próxima comunicação, e assim por diante.
- Frequência híbrida: consiste em se fixar o canal de comunicação como o primeiro canal sorteado pelo dispositivo.

Cada um dos modos idealizados possuem pontos positivos e negativos, a depender do cenário

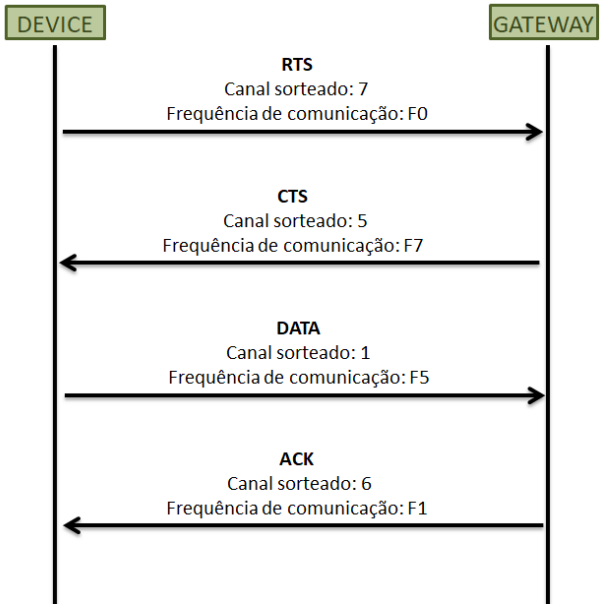

Figura 4.15: Comunicação com frequência randômica.

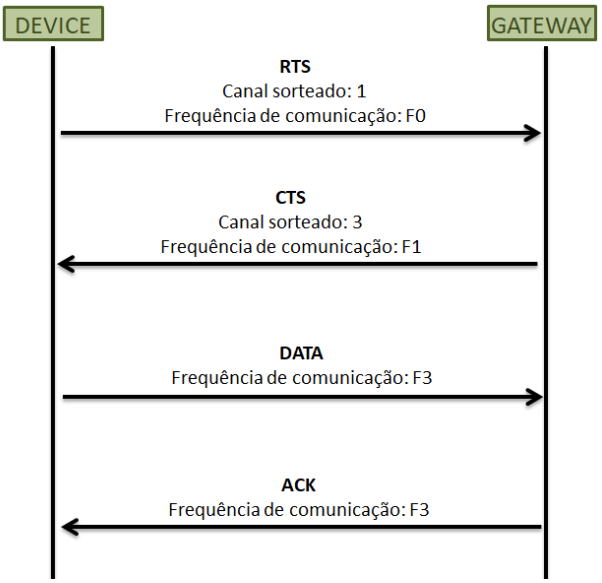

Figura 4.16: Comunicação com frequência híbrida.

que se encontre. A frequência fixa, por exemplo, pode ser uma boa opção para sistemas não muito grandes que possuam vários gateways, cada um fixado em uma frequência diferente. O grande ponto negativo, é a necessidade de se balancear as frequências dos dispositivos de forma manual, diminuindo a flexibilidade da rede. Já na frequência randômica, todos os pacotes recebem uma frequência aleatório para se comunicar, que é definida pelo último pacote recebido. Nesse caso, se dois dispositivos se comunicarem com o gateway em tempos muito próximos um do outro, os dois dispositivos saberão em qual frequência enviar a próxima mensagem, o que pode gerar colisões. Por fim, a frequência híbrida aparenta ser a melhor opção. Se dois dispositivos se comunicarem com o gateway em tempos muito próximos um do outro, somente um dos dispositivos saberá em

qual frequência a comunicação será fixada, diminuindo as chances de colisão.

O modo utilizado nesse projeto será a frequência híbrida, escolhida empiricamente, pois para realizar os testes e avaliar qual a melhor estratégia, há uma dependência de uma quantidade muito grande de dispositivos se comunicando, que não é possível de se realizar nesse projeto.

Capítulo 5

Desenvolvimento

5.1 Introdução

Esse capítulo será dedicado à apresentação da metodologia utilizada no trabalho para a construção do protocolo de IoT para automação predial. Com a definição do protocolo no Capítulo 4, em que se encontram as regras e a estrutura do protocolo, é necessária a construção efetiva do mesmo, que será desenvolvida na linguagem C/C++ para o ambiente de desenvolvimento integrado do Arduino. Além da construção do protocolo, será abordado a metodologia de preparação e instalação do sistema central de automação no Raspberry Pi.

Um passo-a-passo de como utilizar o protocolo será também apresentado. Para a validação do projeto, será construída uma aplicação de automação predial simplificada, de modo a verificar o comportamento da rede IoT criada. A partir dessa aplicação, serão criados alguns cenários de teste com o intuito de avaliar o desempenho e ser possível comparar o resultado da eficiência do protocolo com outros trabalhos.

5.2 Protocolo Aéreo

O desenvolvimento do código do Protocolo Aéreo consiste basicamente na criação de uma biblioteca com funções na linguagem C/C++ para o ambiente de desenvolvimento integrado do Arduino, que realizem as funcionalidades descritas no Capítulo 4. Para isso, utilizaram-se algumas bibliotecas descritas na Seção 3.6 para o controle dos *hardwares* utilizados e para o uso de algoritmos necessários para manipulação e tratamento dos dados.

Primeiramente, foram construídas funções para realizar a comunicação por meio da tecnologia LoRa entre dois dispositivos. Utilizando-se de códigos exemplos disponibilizados pelo desenvolvedor da biblioteca que controla o rádio LoRa, foi possível entender como se realiza a codificação e decodificação de transmissões LoRa, além de como modificar os parâmetros de comunicação, como fator de espalhamento, frequência central, *bandwidth* e potência.

Após essa etapa concluída, foram construídas as funções para comunicação do Gateway com

o sistema central de automação. Tendo o *mosquitto broker* MQTT instalado e funcionando no Raspberry Pi, se torna possível enviar e receber mensagens dos tópicos MQTT. A partir disso, e utilizando a biblioteca para conexão, envio e recebimento de dados com o *broker*, criaram-se funções de conexão, envio e recebimento de dados.

Com a parte das comunicações entre os dispositivos resolvida, o próximo passo tomado foi da implementação das funcionalidades do protocolo. A partir das definições e das máquinas de estados criadas, desenvolveu-se o código central, responsável por utilizar as funções de comunicação. Fez-se necessária a criação de funções para a manipulação dos dados, montagem das mensagens a serem moduladas, funções para criação e validação dos campos de verificação da integridade das mensagens, por meio de protocolos como AES-128 e CRC-16, além do "esqueleto do código", que assegura a corretude do fluxo de dados a partir dos possíveis caminhos a serem tomados, descritos na máquina de estados.

Por fim, foi criado um código que disponibiliza as informações na interface serial do microcontrolador para facilitar a visualização do fluxo de dados, se necessário, utilizado para encontrar erros no código. Um exemplo de uma comunicação realizada, visualizada através da interface serial pode ser vista nas Figuras 5.1 e 5.2.

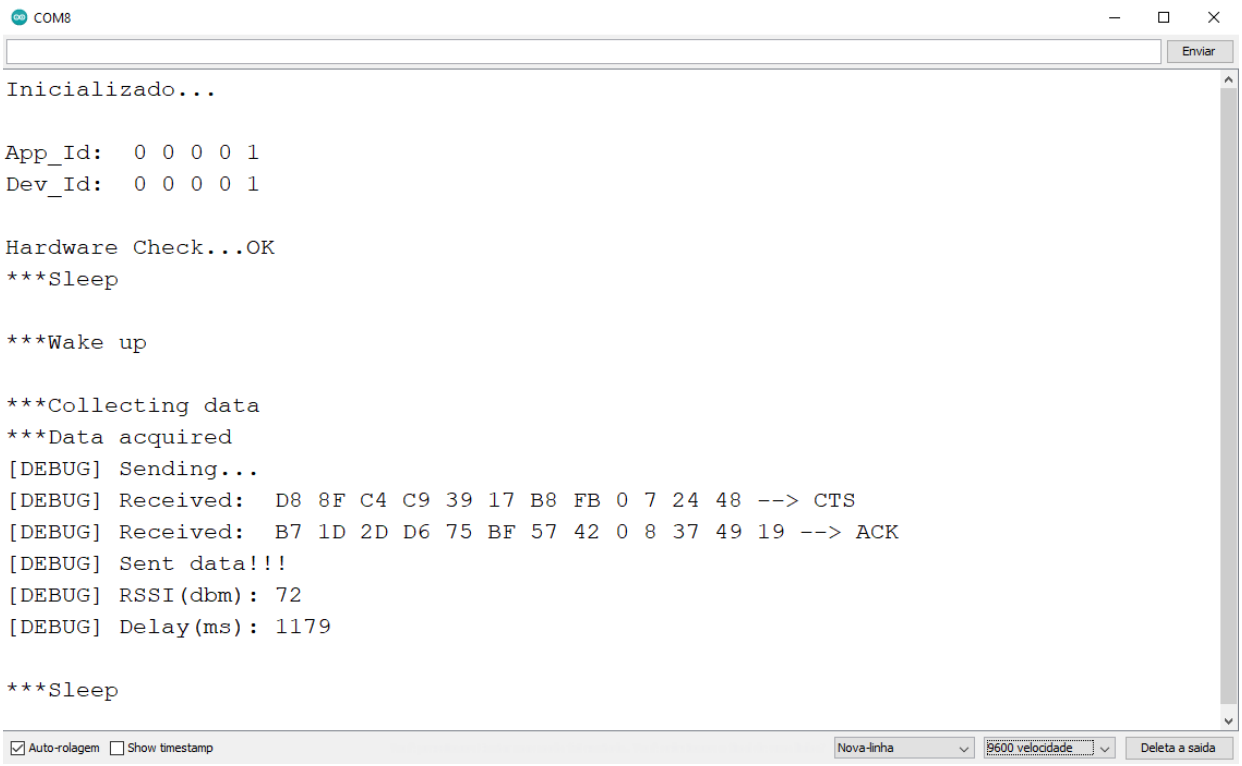

Figura 5.1: Interface serial para debug do dispositivo.

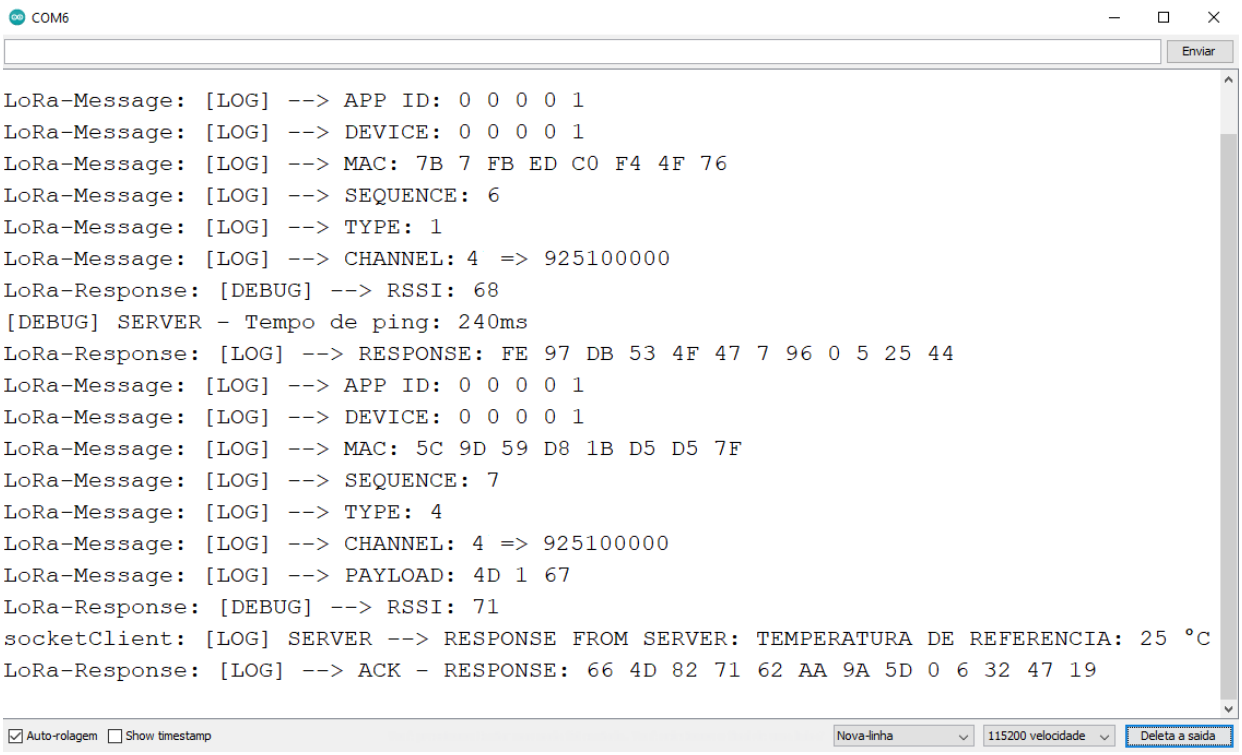

Figura 5.2: Interface serial para debug do gateway.

5.3 Sistema Central de Automação

O sistema central de automação consiste do sistema *openHABian* instalado no Raspberry Pi. Para isso, há diversas opções de instalação e preparação do software. O método utilizado consiste da instalação de uma imagem de cartão SD, que já é previamente configurada com os pacotes essenciais do software openHAB. As etapas de instalação são:

- Download da última versão do arquivo da imagem de cartão SD openHABian, disponível em `https://github.com/openhab/openhabian/releases`;
- Gravar a imagem baixada no cartão SD do Raspberry Pi, utilizando software Win32DiskImager, balenaEtcher ou similar;
- Inserir o cartão SD no Raspberry Pi, conectar o cabo Ethernet (WiFi também é possível) e ligá-lo;
- Esperar aproximadamente 15-45 minutos para que o openHABian seja atualizado e esteja funcional;

Após essas etapas, o software OpenHAB já está configurado, porém ainda é necessário instalar o banco de dados InfluxDB, o software de analise gráfica Grafana e o MQTT Mosquitto broker. Para isso, é necessário utilizar o software PuTTY para conectar remotamente, via Secure Shell (SSH) ao Raspberry Pi. Ao se conectar via SSH com o Raspberry Pi, utilizando username:

openhabian e password: openhabian, utilizaremos alguns comandos para realizar a instalação dos softwares.

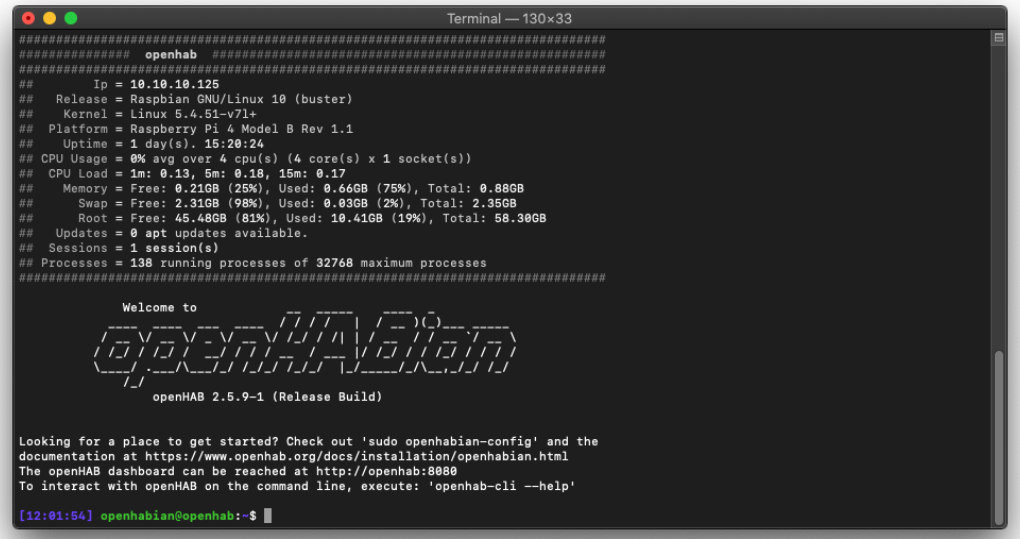

Figura 5.3: Conexão via SSH ao Raspberry Pi.

Uma vez conectado ao console do sistema openHABian, executaremos a ferramenta de configuração do openHABian a partir do seguinte comando:

```
1 $ sudo openhabian−config
```

Ao iniciar a ferramenta de configuração, visualizada na Figura 5.4, basta selecionar a opção *Optional Components* e instalar o banco de dados InfluxDB juntamente com a ferramenta gráfica Grafana e o MQTT *Mosquitto broker*.

Com isso, a instalação e preparação do software openHAB no Raspberry Pi está concluída. A partir desse ponto, utilizou-se a documentação do openHAB, disponível em `https://www.openhab.org/docs/` para a criação das interfaces gráficas e o encaminhamento das informações disponíveis no MQTT *Mosquitto broker* para as interfaces gráficas. O *sitemap*, local onde são disponibilizadas as interfaces gráficas no *openHAB*, criado para a aplicação construída na Seção 5.5 pode ser visualizado na Figura 5.5.

5.4 Instruções de uso do Protocolo Aéreo

A partir do código disponibilizado nos Apêndices do Protocolo Aéreo (biblioteca OnAirProtocol) e tendo a IDE do Arduino instalado, basta adicionar os arquivos da biblioteca OnAirProtocol, que é uma biblioteca criada a partir das funções desenvolvidas nesse projeto. Esse processo é exemplificado na Figura 5.6.

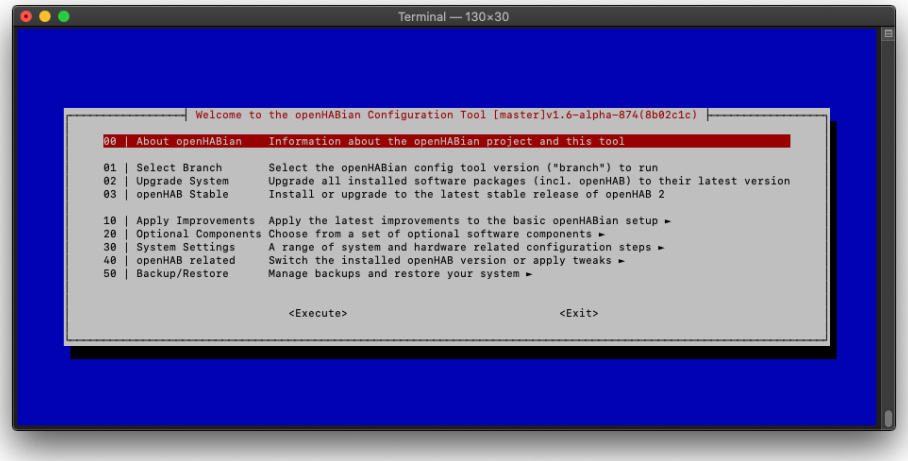

Figura 5.4: *OpenHABian Configuration Tool*, utilizado para atualização de softwares, configurações do sistema, entre outros.

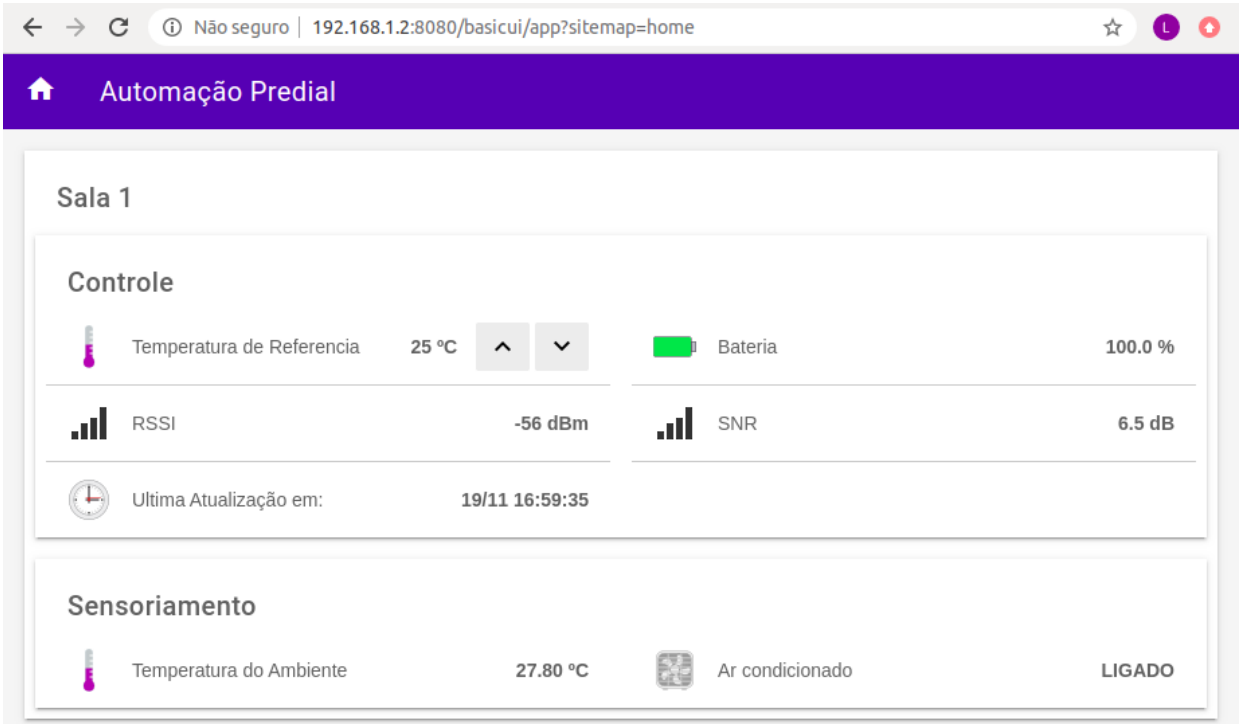

Figura 5.5: Interface gráfica para monitoramento dos dados.

O intuito da criação da biblioteca visa facilitar o uso do Protocolo Aéreo, sendo somente necessária a utilização de uma função para que seja possível enviar os dados. Exemplos de códigos para o Gateway e para os dispositivos são disponibilizados nos Apêndices.

O primeiro passo para o uso, tendo a biblioteca instalada, é gravar os códigos no Gateway e no dispositivo. Ao gravar o código no Gateway, será inicializada uma rotina para configuração das credenciais da rede em que se encontra conectado seu Raspberry Pi. Eles precisam estar

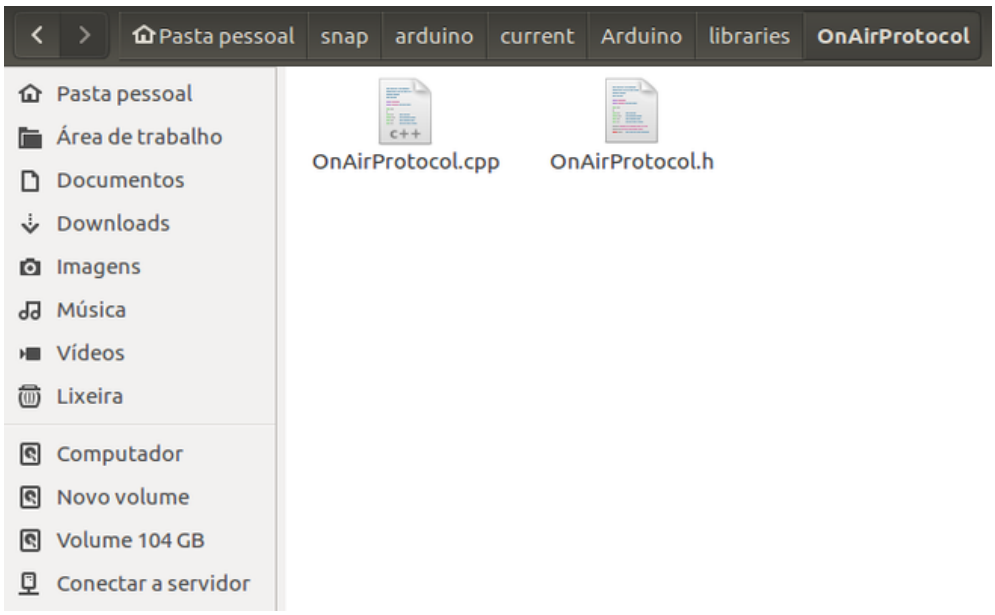

Figura 5.6: Adicionando a biblioteca do protocolo na pasta de bibliotecas da IDE do Arduino.

conectados na mesma rede local, para que seja possível o envio das informações.

Para a configuração das credenciais, o Gateway se torna um Acess Point(AP), e a partir de um celular, deve-se conectar na rede criada por ele, utilizando ssid: gateway_config e senha: 123456789. Após a conexão, deve-se entrar em um browser e se conectar ao *IP* 192.168.4.1. Um código *html* será executado para a obtenção das credenciais. O processo de configuração de credenciais pode ser visualizado nas Figuras 5.7 e 5.8.

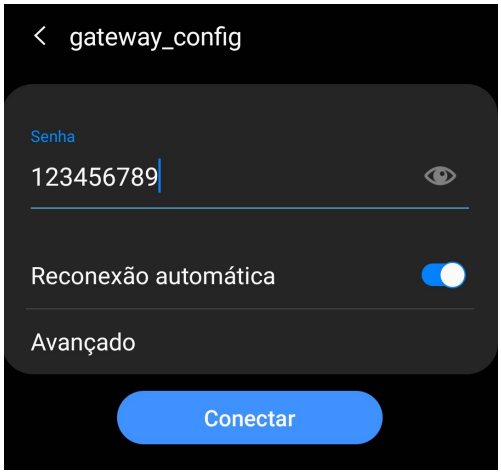

Figura 5.7: Conectando no Acess Point criado pelo Gateway para envio das credenciais.

As credenciais são salvas na memória não-volátil do microcontrolador e serão utilizadas para se conectar à rede WiFi sempre que o Gateway for ligado. Se a rede não estiver disponível, a rotina de configuração é iniciada novamente. Esse é o processo inicial para configuração do gateway, que está apto a receber pacotes dos dispositivos e realizar o encaminhamento das mensagens para o sistema central. O próximo passo é a configuração dos dispositivos.

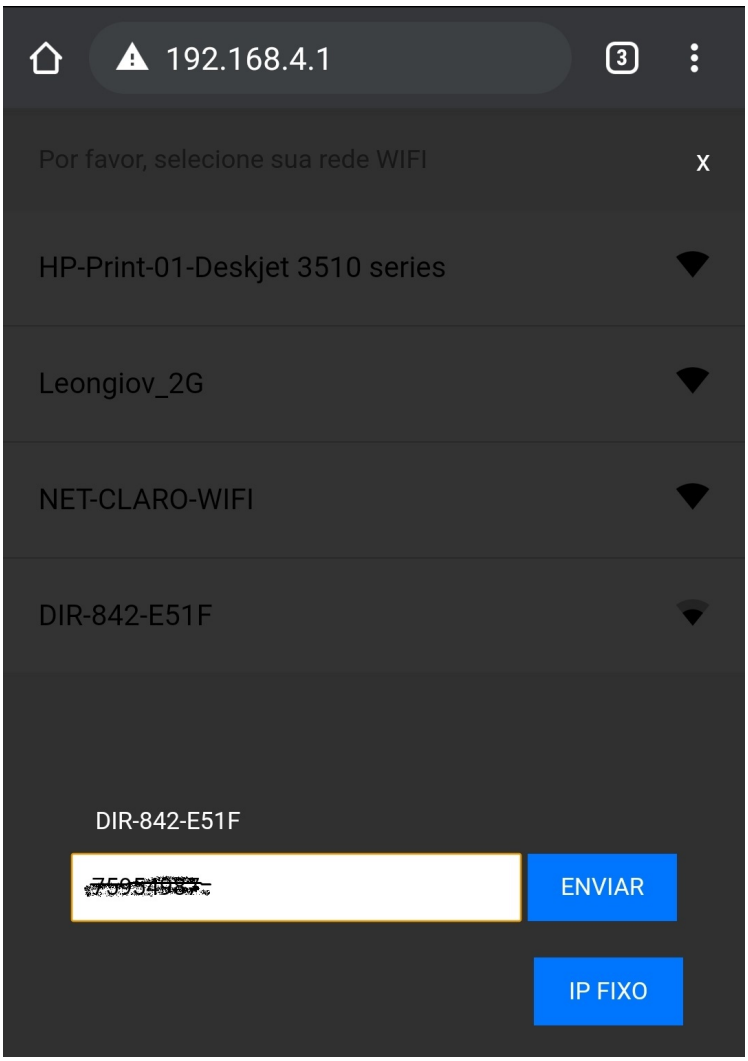

Figura 5.8: Interface html para detecção das redes WiFi disponíveis e configuração da senha.

De modo a simplificar o envio, o código base dos dispositivos para envio de informações necessita de poucos parâmetros a serem configurados em código antes de ser possível se comunicar com o gateway. Deve-se incluir a biblioteca do protocolo, assim como feito no código do gateway, definir duas variáveis, APP_ID e DEV_ID, que estão relacionadas com para qual aplicação a informação será enviada e o número identificador do sensor (código único). A definição desses parâmetros, de modo a evitar conflitos, é de responsabilidade do usuário, mas rotinas de cadastro poderiam ter sido implementadas. Uma função da biblioteca, *start_LoRa()* , deve ser inicializada no setup, para configuração inicial do rádio LoRa. Por fim, basta preencher na variável *payload*, uma informação de até 16 bytes, no formato hexadecimal e chamar a função *sendMessage(SEND_DATA)*, que irá enviar as informações contidas em *payload* para o gateway, de acordo com o Protocolo Aéreo. O código exemplo, que foi explicado acima, pode ser visualizado na Figura 5.9. Para utilizar as outras duas funcionalidades do protocolo, envio de dados prioritário e teste de rede, basta mudar o parâmetro na função *sendMessage()*, para, respectivamente, SEND_PRIORITY e SEND_TEST.

Figura 5.9: Exemplo de código para envio básico, utilizando o protocolo aéreo.

5.5 Aplicação de Automação Predial

A aplicação voltada para automação predial que se deseja construir tem papel fundamental nesse trabalho. O desenvolvimento do protocolo eficiente IoT proposto para automação predial, para ser validado, necessita de testes que confirmem a eficiência do protocolo no contexto de automação predial. O protocolo tem o objetivo de resolver pontos críticos identificados na literatura, como, por exemplo, eficiência energética da rede. Visto isso, e dado que uma das aplicações mais comuns em automação predial é o controle de climatização, optou-se por criar uma solução simples, que simule um controle de climatização.

A partir da biblioteca desenvolvida, torna-se necessário somente se preocupar com o desenvolvimento de um código que obtenha as informações dos sensores, controle de atuadores, entre outros funcionalidades, porém a conectividade é resolvida utilizando uma função. Para o controle de climatização, utilizaremos o sensor DS18B20, que é um sensor de temperatura, e um relé, que irá simular o acionamento de um ar condicionado. Como o intuito é simular uma solução de climatização, e o foco não é maximizar o consumo energético, e sim testar o protocolo, o controle proposto é o mais simples possível, um controle liga-desliga.

A solução proposta tem como funcionalidade desejada realizar a medição da temperatura a partir do sensor DS18B20. A partir de uma temperatura de referência escolhida pelo usuário a partir da interface gráfica, como pode ser visualizado na Figura 5.5, o dispositivo decidirá, a

partir de uma controle liga-desliga, se o relé deve estar energizado ou não. Se a temperatura medida pelo sensor for maior que a temperatura referência, o relé deve ser energizado, de modo a simular o acionamento de um ar condicionado. Quando a temperatura medida for menor que a temperatura referência, o relé é desenergizado. A temperatura de referência é atualizada em toda comunicação do dispositivo com a rede, informação esta que será enviada pelo payload de downlink. As informações do dispositivo que são enviadas para o sistema central, contidas no payload de uplink, são a temperatura medida, o nível da bateria e o estado do relé. As rotinas no código seguem a seguinte ordem:

1. As bibliotecas utilizadas são incluídas no código e as funções de inicialização de sensores e do protocolo são executadas;

2. No *loop* principal, o primeiro parâmetro configurado é o *watchdog*. O *watchdog* tem a função de detectar falhas no hardware, e se detectado, força um *reset* no microcontrolador. O *watchdog* é reativado várias vezes durante o código e se não for reativado em um período de 8 segundos, é realizado o *reset*. O único momento em que o *watchdog* é desabilitado durante o código, é logo antes de entrar em modo de baixo consumo, e logo após o fim do modo de baixo consumo, é reabilitado;

3. Após a configuração do *watchdog*, o microcontrolador será colocado em modo de baixo consumo. Antes de entrar em modo de baixo consumo, é configurado o *Real Time Clock* (RTC) para gerar um alarme para 20 segundos depois, que será responsável por tirar o microcontrolador do modo de baixo consumo, através de uma interrupção de hardware. O tempo de 20 segundos foi escolhido, pois deseja-se enviar os dados para a central do sistema a cada 20 segundos, aproximadamente;

4. Após o fim do modo de baixo consumo, são adquiridos os dados de temperatura e bateria. Os terminais da bateria são conectados no conversor A/D do microcontrolador, que possui 1024 passos (10 bits), para que seja possível obter a voltagem das baterias e se verificar a carga disponível. Tendo a temperatura medida e utilizando a temperatura referência da última comunicação salva, é energizado ou não o relé;

5. Com essas três informações obtidas anteriormente (temperatura do ambiente, bateria e estado do relé), é montado o payload com as informações para enviar para o servidor e executada a função *sendMessage(SEND_DATA)*. Ao finalizar a comunicação, é atualizado o valor da temperatura de referência, recebida pelo payload de downlink;

6. Com isso, um ciclo de execução é finalizado, e será executado enquanto o dispositivo estiver ligado.

A montagem final do dispositivo pode ser visualizado na figura 5.10. O relé simulando o acionamento do ar condicionado está utilizando as ligações de modo ao seu acionamento ser normalmente aberto, ou seja, o estado natural sem o relé estar energizado é tal, de modo ao ar condicionado estar desligado. Isso ocorre, pois se acabar a energia no microcontrolador, o ar condicionado é desligado.

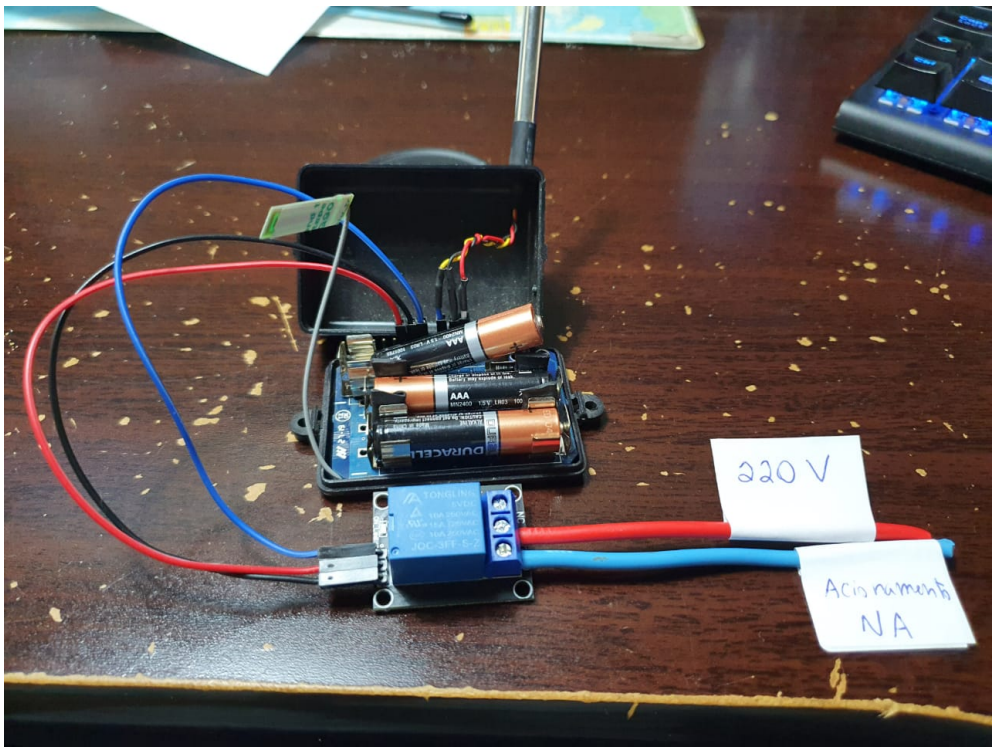

Figura 5.10: Montagem final do dispositivo responsável por adquirir a temperatura e simular o acionamento de um ar condicionado.

5.6 Cenários de teste

A partir da aplicação desenvolvida na seção 5.5, deseja-se avaliar o comportamento do sistema em cenários de teste. É desejável que esses cenários sejam o mais próximos o possível de um ambiente predial, o ambiente de enfoque desse trabalho. O intuito dos testes é de estressar a rede, analisando e comparando o desempenho com outros trabalhos e tecnologias, analisar se o desempenho da rede é suficiente para resolução do problema proposto e que sejam identificados pontos que precisam de melhorias.

Serão analisadas quatro métricas a partir dos cenários: consumo energético, latência de comunicação, confiabilidade da rede e alcances de transmissão. Para os testes de consumo energético e latência de comunicação, os sensores foram dispostos no ambiente de testes, de acordo com a Figura 5.11. As unidades do desenho estão na escala de centímetros. Já para o teste de confiabilidade, a disposição dos três sensores utilizados estão de acordo com a Figura 5.12. Por fim, o teste de alcance de transmissão está de acordo com a Figura 5.18. O tempo de intervalo entre transmissões é de aproximadamente 20 segundos.

5.6.1 Teste de consumo

Uma das principais vantagens que justificaram o uso da tecnologia LoRa nesse trabalho advém do baixo consumo energético necessário para a comunicação entre os dispositivos, em relação as tecnologias mais utilizadas. Isso se torna muito importante, pois em grandes sistemas de

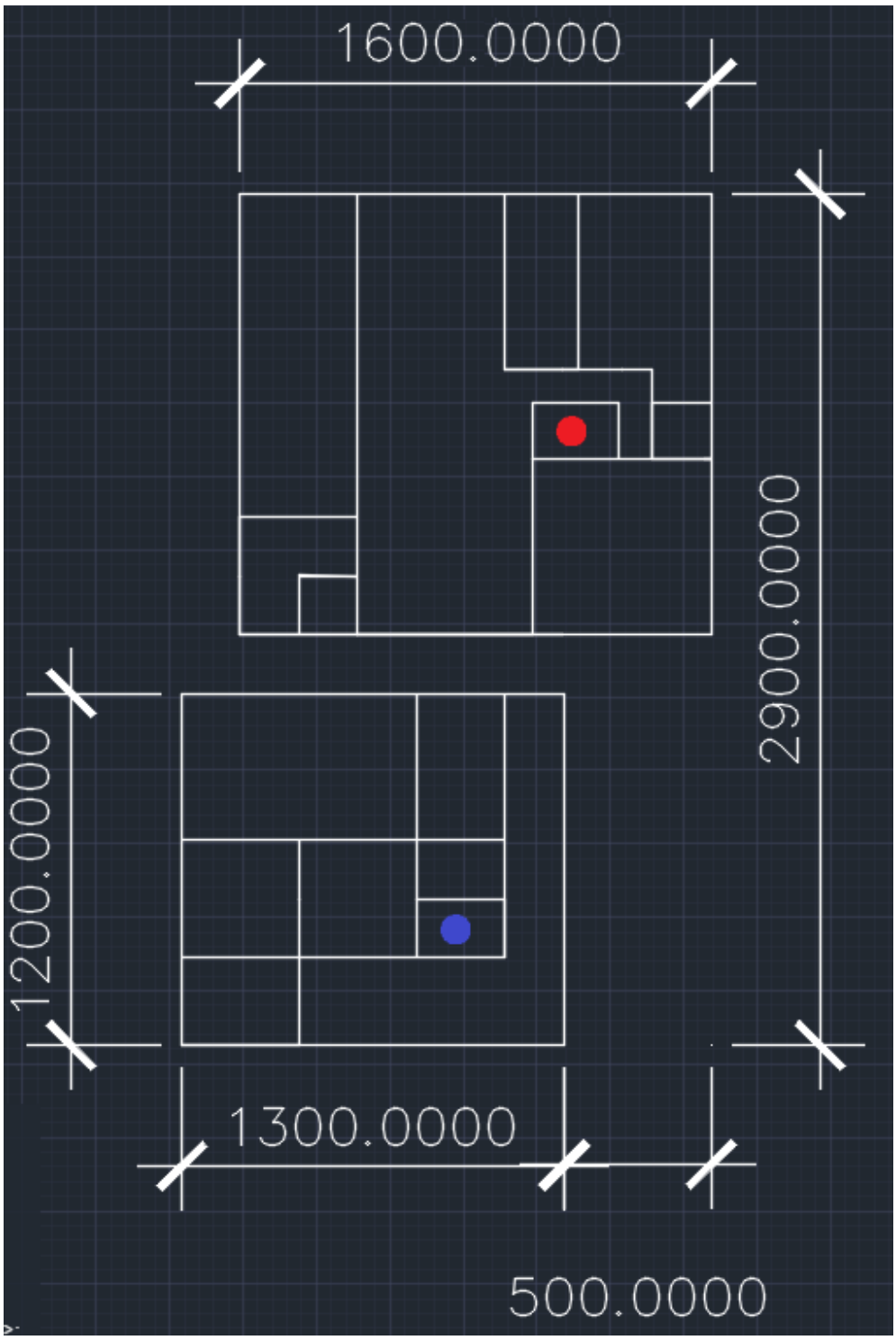

Figura 5.11: Disposição dos sensores no ambiente de testes. O círculo azul representa o *gateway* e o círculo vermelho, o dispositivo. Unidades em centímetros.

automação, onde os dispositivos são alimentados por baterias, é desejável que as baterias durem longos períodos de tempo.

A estratégia utilizada na aplicação construída para minimizar o consumo energético é de, após as medições e a comunicação com o servidor, ativar o modo de baixo consumo do microproces-

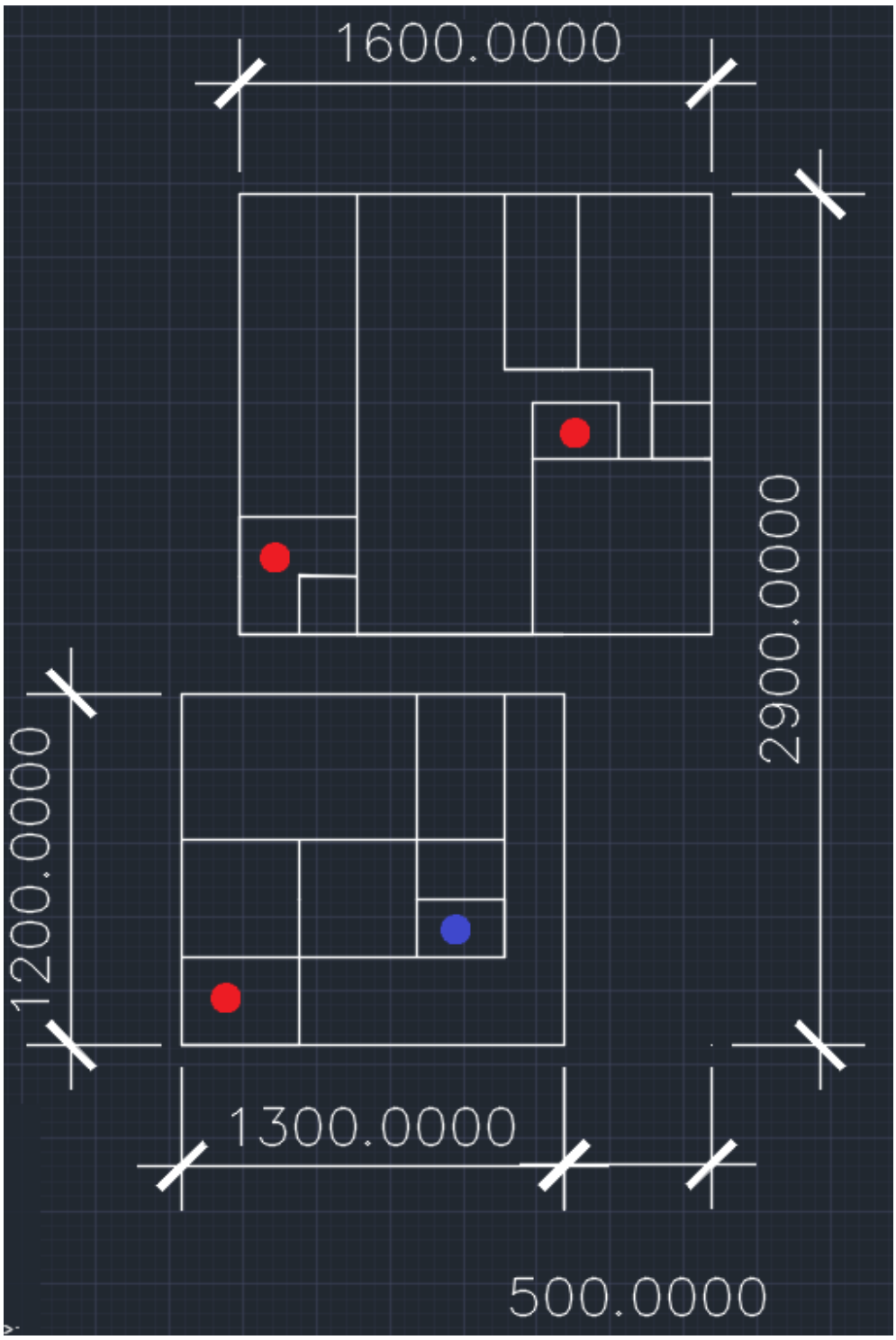

Figura 5.12: Disposição dos sensores no ambiente de testes. O círculo azul representa o *gateway* e os círculos vermelhos, os dispositivos. Unidades em centímetros.

sador. O modo de baixo consumo desabilita os principais componentes do dispositivo, como os conversores A/D e as estruturas de comunicação SPI e I2C, além de reduzir a velocidade do clock do processador.

Para realizar a medição do consumo de corrente do dispositivo, será utilizado um multímetro

no modo amperímetro. O amperímetro mede a corrente que está passando pelo circuito, e deve ser ligado em série com o dispositivo. A medição utilizando esse equipamento não é a ideal, pois as medições do visor mostram a corrente instantânea, não sendo possível visualizar a quantidade de carga utilizada nos transitórios que ocorrem, por exemplo, quando o rádio está modulando informações. Quando o microprocessador estiver em modo de baixo consumo, as medições serão mais fáceis de analisar, pois o sistema estará utilizando uma quantidade de corrente próxima de um valor fixo.

Serão analisadas duas situações para comparação da eficiência das estratégias utilizadas para minimização do consumo energético. A primeira situação consiste em realizar a medição da corrente, por meio do amperímetro, utilizando o modo de baixo consumo. A segunda situação consiste em realizar a medição da corrente, por meio do amperímetro, sem utilização do modo de baixo consumo do microprocessador.

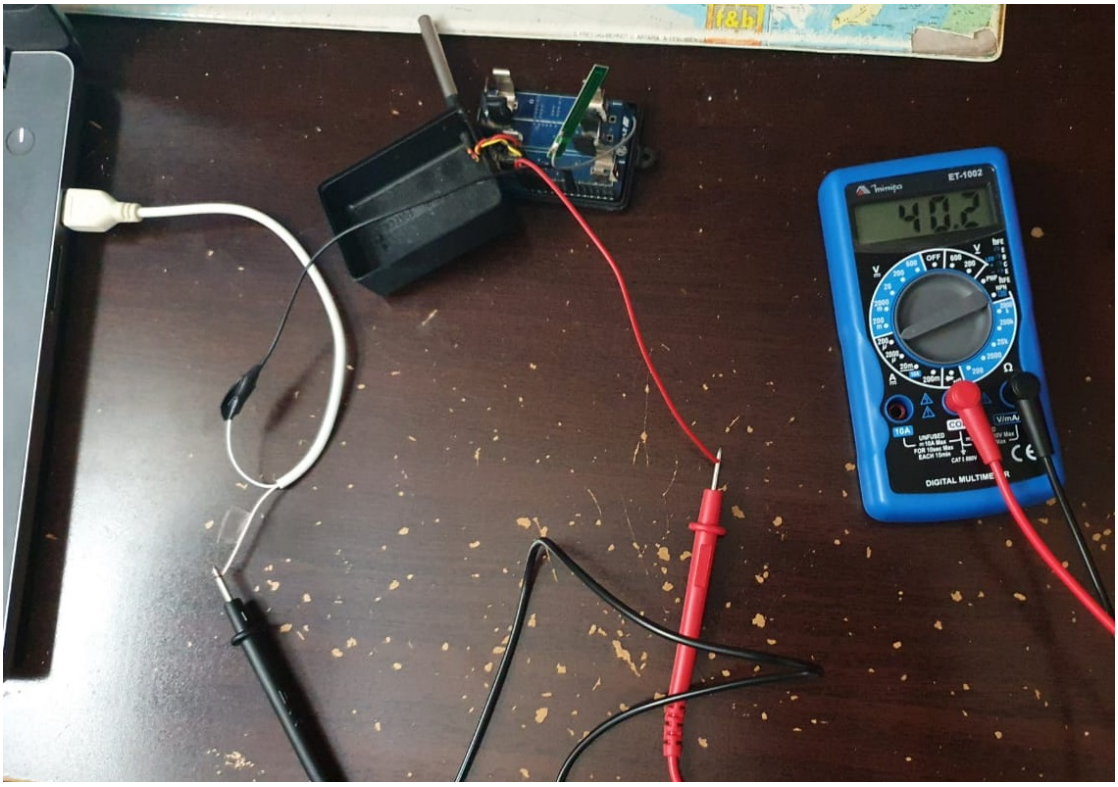

Figura 5.13: Medição de corrente do dispositivo em modo de baixo consumo. A escala do amperímetro se encontra em $200\mu A$.

A primeira medição foi realizada e pode ser visualizada na Figura 5.13. Nela foi possível se obter o consumo de 40,2μA quando o dispositivo se encontra em modo de baixo consumo. A segunda medição possui o objetivo de se obter o consumo do dispositivo sem a utilização do modo de baixo consumo. Essa medição pode ser visualizada na Figura 5.14. Nela foi possível se obter o consumo de 11,2mA quando o dispositivo se encontra sem a utilização do modo de baixo consumo.

Por fim, realizou-se a medição do pico de corrente que ocorre quando o dispositivo não se encontra em modo de baixo consumo e está modulando a informação a ser enviada. Essa medição

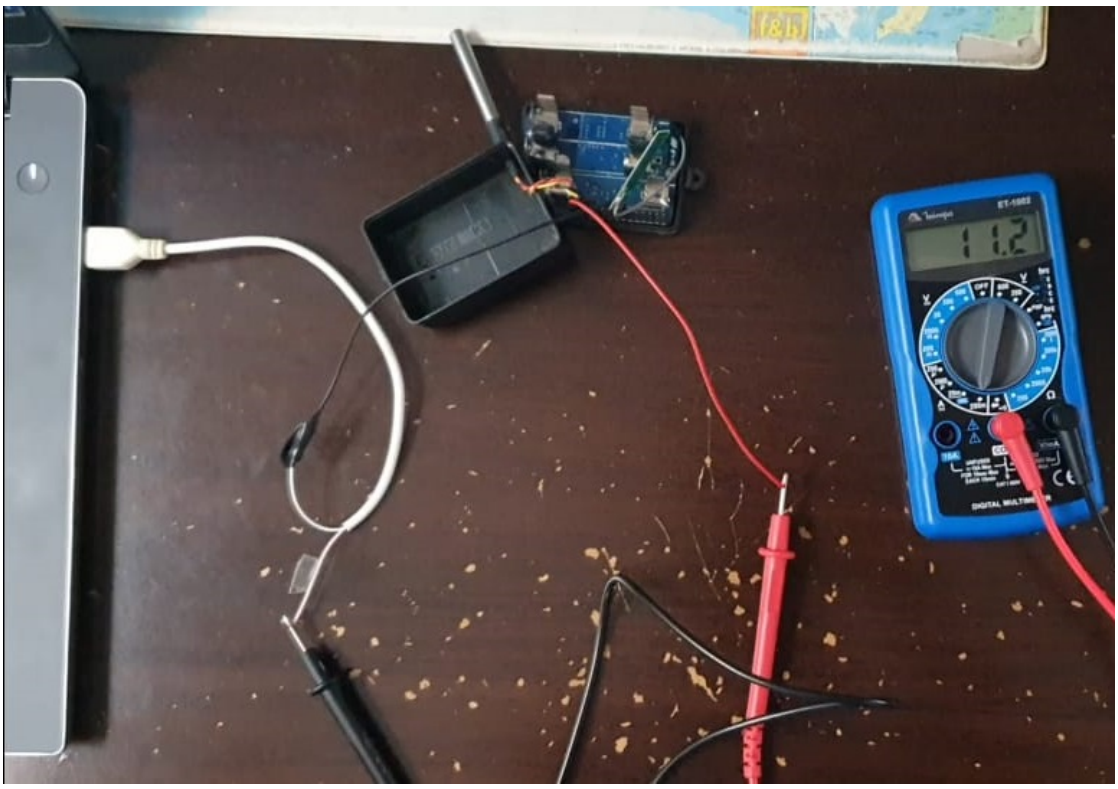

Figura 5.14: Medição de corrente do dispositivo sem a utilização do modo de baixo consumo. A escala do amperímetro se encontra em 200mA.

pode ser visualizada na Figura 5.15. Nela foi possível se obter o consumo de 101,9mA de pico quando o dispositivo está modulando a informação. Para se realizar uma medição mais precisa, se torna necessário a utilização de um osciloscópio, sendo possível estimar melhor os consumos energéticos quando o dispositivo modula as informações.

A partir dos dados de consumo obtidos, é possível se realizar uma estimativa muito simplificada da duração das baterias no hardware. Essa estimativa se torna muito simplificada, pois serão desconsiderados os consumos referentes as modulações das informações, que mesmo que por intervalos pequenos, são os momentos em que se há os maiores picos de corrente. Considerando que o hardware é alimentado por 3 pilhas alcalinas AAA, com capacidade de 1200mAh cada, temos uma capacidade de 3,6Ah. Para se obter o tempo de duração da pilha, basta se dividir a capacidade das pilhas, pelo consumo de corrente médio. Desconsiderando os envios de informação, o hardware possui um consumo médio de corrente de 40,2μA. Ao se realizar a divisão de 3,6Ah pelo consumo de corrente de 40,2μA, obtemos um tempo de duração de 89.552 h, aproximadamente, o que nos confere uma autonomia de aproximadamente 10 anos.

Essas estimativas estão muito simplificadas, pois diversos fatores estão sendo desconsiderados, como a descarga natural das pilhas, o consumo ao enviar as informações, além do ciclo de trabalho. O intuito da estimativa é somente mostrar que o modo de baixo consumo é muito eficaz ao reduzir drasticamente o consumo de energia pelo microcontrolador. Ao se comparar as medições utilizando o modo de baixo consumo e não utilizando o modo de baixo consumo, Figuras 5.13 e

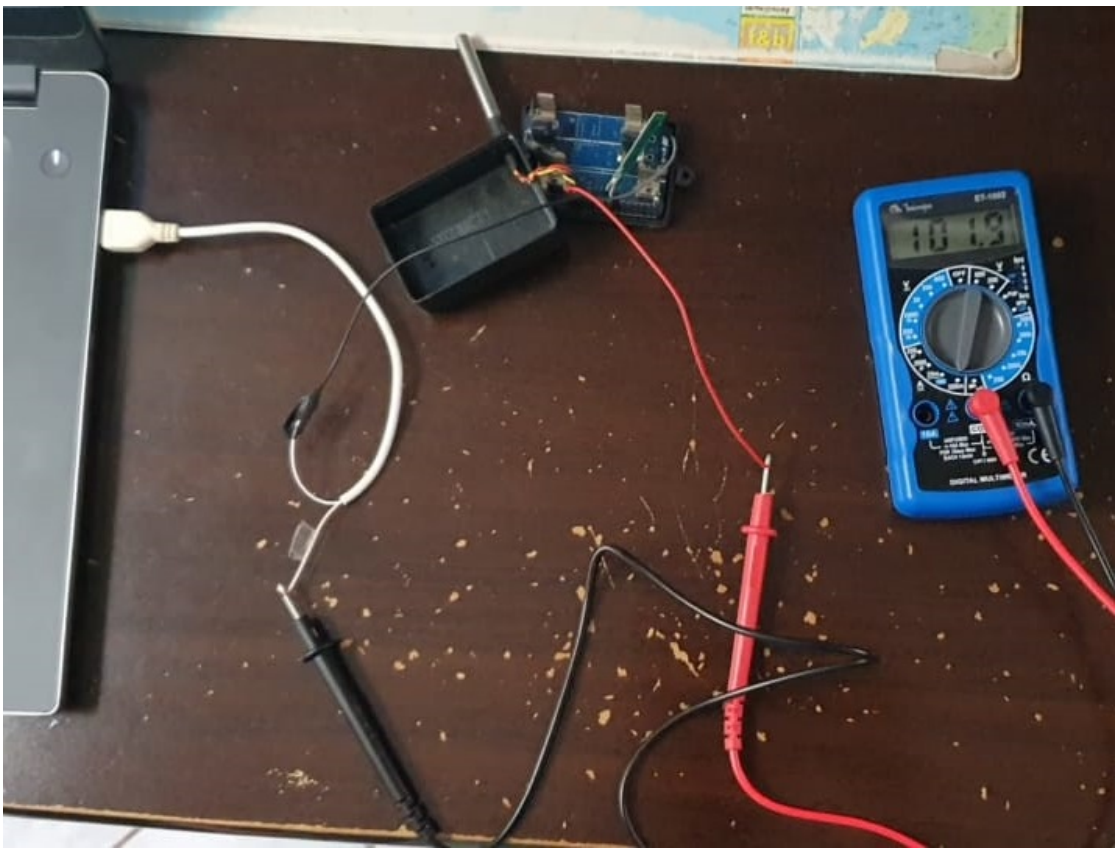

Figura 5.15: Medição do pico de corrente do dispositivo quando modula as informações. A escala do amperímetro se encontra em 200mA.

5.14, respectivamente, há uma redução de consumo de 278 vezes.

Ao se comparar esse consumo com o consumo de outras tecnologias, como Bluetooth, podemos perceber a eficiência da tecnologia LoRa. Utilizando os dados de um módulo Bluetooth HC-05, obtidos pelo seu datasheet, observamos um consumo de corrente de 100mA em modo de transmissão e o modo de baixo consumo consome entre 360μA e 900μA. Esse consumo é de 9 a 22 vezes maior que o consumo energético do dispositivo utilizado nesse projeto. Visto isso, percebe-se a eficiência energética da tecnologia LoRa.

5.6.2 Teste de latência

Em um sistema de automação, um fator essencial é o tempo de resposta. A depender das variáveis a serem controladas, o sistema pode tolerar tempos de resposta altos ou não. Por exemplo, em um sistema de controle de temperatura de uma usina nuclear, o sistema não pode tolerar altos tempos de resposta, pois as vidas das pessoas dependem da estabilidade do sistema nuclear. Visto isso, será testado o tempo necessário para que o dispositivo se comunique com o sistema central e receba a resposta.

O teste consiste em somar os tempos de latência de cada comunicação com o servidor durante um período de tempo estipulado. A divisão do somatório dos tempos de latência, pela quantidade

de comunicações realizadas nos fornece o tempo médio de latência. Esse teste será realizado por um período de 15 dias.

Após o período de testes realizado, e utilizando o software Grafana, pode se obter gráficos da temperatura medida pelo sensor. Os dados podem ser visualizados na Figura 5.16.

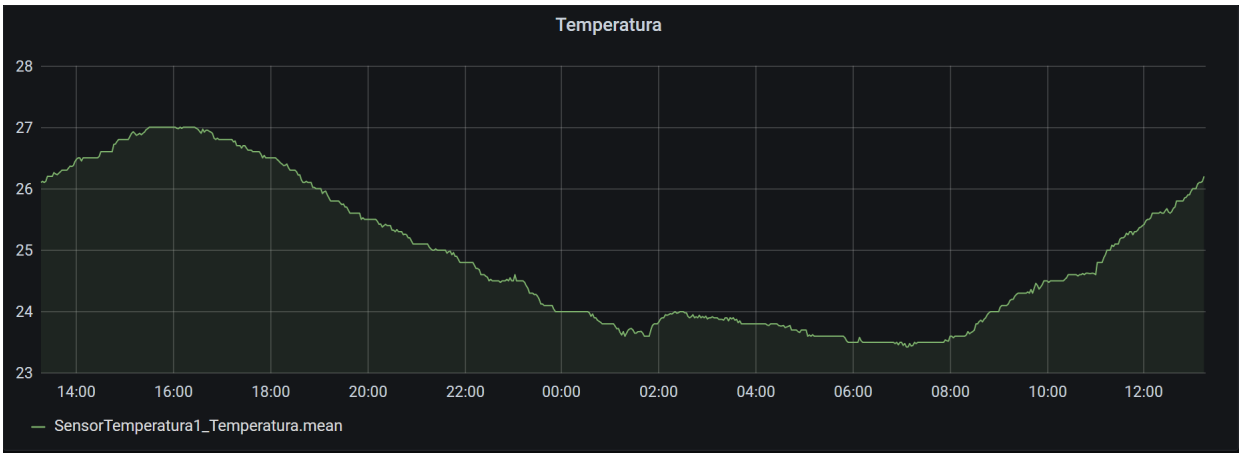

Figura 5.16: Medições de temperatura durante o período de um dia.

Durante o período estipulado, foram registradas 45.630 tentativas de comunicações. A média de latência obtida nesses testes foi de 1249 milissegundos para que o dispositivo envie as informações para o servidor e receba a resposta. A partir desse resultado e utilizando a Tabela 4.5, que nos mostra os tempos teóricos das comunicações LoRa, podemos chegar a algumas conclusões.

A primeira conclusão é que, pelo fato do dispositivo e o gateway estarem próximos, o sinal se encontra forte e o fator de espalhamento utilizado nas transmissões é o valor 7. Teoricamente, utilizando a Tabela 4.5, o somatório de tempo necessário para envio da informação, somente das transmissões LoRa, utilizando o fator de espalhamento 7 é de 877,06 milissegundos. O somatório de tempo necessário para o envio da informação, somente das transmissões LoRa, utilizando o fator de espalhamento 10 é de 1.712,14 milissegundos. Para confirmar essa conclusão, utilizamos o gráfico do Indicador de Potência do Sinal Recebido(RSSI), que nos mostrou que o sinal estava muito forte como pode ser visualizado na Figura 5.17.

Logo, a diferença entre o tempo de 877,06 milissegundos, que são teoricamente esperados e a média de latência de 1249 milissegundos se encontra no tempo de comunicação que o gateway precisa para se comunicar com a central de automação, além do tempo de processamento dos microcontroladores e possíveis variações que se encontram entre os valores teóricos e os valores obtidos em experimentos práticos.

A segunda conclusão está relacionada com a dinâmica de um sistema de climatização. Ao se analisar a variação térmica de um dia, como observado na Figura 5.16, conclui-se que a constante de tempo do sistema é muito alta. Esse fator valida o tempo de resposta do sistema, pois para que haja a variação de 1° C são necessárias várias horas e o tempo de resposta do sistema se encontra na ordem de segundos.

Ao se analisar outras soluções de automação predial, como por exemplo, controle de iluminação,

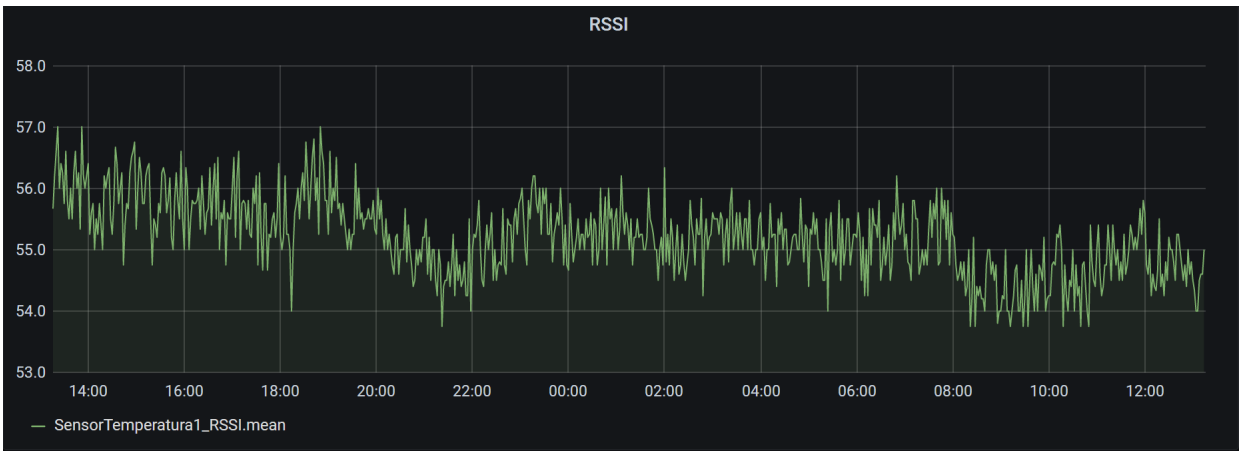

Figura 5.17: Indicador de Potência do Sinal Recebido durante o período de um dia.

controle de acesso, monitoramento hídrico, dentre outros, tempos de resposta do sistema na ordem de segundos não acarretam em prejuízos ao sistema. Por outro lado, em sistemas de automação industrial, onde as vidas das pessoas dependem de tempos de respostas baixos, respostas do sistema na ordem de segundos inviabilizariam a utilização do protocolo.

5.6.3 Teste de confiabilidade

Por fim, será realizado um teste que visa avaliar a quantidade de comunicações realizadas com sucesso em relação a quantidade de tentativas. Além disso, tem a função de analisar o impacto da abordagem de retransmissões utilizada no protocolo. Para isso, serão utilizados três dispositivos. Esse teste será realizado por um período de 15 dias, e será realizado ao mesmo tempo que o teste de latência da Seção 5.6.2.

Durante o período estipulado, foram registradas 45.630 tentativas de comunicações. Dessas tentativas, 5.190 não necessitaram de retransmissões, 38.430 necessitaram de uma retransmissão, 915 necessitaram de duas retransmissões e 1.095 comunicações não foram bem sucedidas. Isso confere uma confiabilidade de 97,6% para a rede. Além disso, se não houvessem retransmissões, a rede teria uma confiabilidade de apenas 11,37%.

Ao se analisar as comunicações bem sucedidas sem a necessidade de retransmissão, percebe-se uma alta taxa de erros nas primeiras comunicações. Após esse resultado, analisou-se o código do protocolo novamente, em busca de erros de desenvolvimento que justifiquem as altas taxas de erros obtidas, porém as buscas não foram bem sucedidas. Não se descarta a possibilidade de erro na implementação que justifique essa alta taxa de erros. De qualquer modo, percebe-se a importância das retransmissões, haja visto que conferem maior confiabilidade para a rede. Um bom exemplo disso, são as terceiras transmissões, que conseguiram impedir comunicações mal sucedidas e aumentar a taxa de confiabilidade da rede em aproximadamente 2%.

Para fins de comparação, a Agência Nacional de Telecomunicações (ANATEL), aprovou a Resolução nº 574 em 28 de outubro de 2011, que aprovou o Regulamento de Gestão da Qualidade do Serviço de Comunicação Multimídia (RGQ-SCM). Essa resolução estabelece uma meta de

perdas de pacotes pelas operadoras de banda larga de até 2%. Comparado com esse valor, o desempenho de 97,6% de confiabilidade está muito próximo. Por fim, é bom salientar que esse desempenho provavelmente será reduzido com uma alta quantidade de sensores comunicando, onde colisões tem a maior probabilidade de ocorrerem.

5.6.4 Teste de distância

Outro fator muito importante de se avaliar no sistema, está relacionado com o alcance de comunicação. Uma das grandes vantagens da tecnologia LoRa em relação às outras tecnologias, está nos grandes alcances obtidos. Esse teste será realizado por um período de 7 dias a uma distância de aproximadamente 450 metros. O dispostivo e o gateway foram dispostos no locais marcados no Google Maps, que pode ser visualizado na Figura 5.18.

Figura 5.18: Teste de distância, aonde o dispostivo e o gateway se encontram nos pontos marcados no mapa a uma distância de aproximadamente 450m.

Após os 7 dias de período de teste, foram registradas 20.440 tentativas de comunicações. Dessas tentativas, 2.358 não necessitaram de retransmissões, 16.346 necessitaram de uma retransmissão, 387 necessitaram de duas retransmissões e 1.353 comunicações não foram bem sucedidas. Isso confere uma confiabilidade de 93,4% para a rede nesse teste.

O desempenho apresentado teve uma redução significativa nas comunicações bem sucedidas. Apesar disso, é um bom resultado, devido aos testes estarem sendo feitos em uma área residencial, aonde o sinal necessita percorrer diversas paredes, que reduzem a intensidade do sinal recebido.

Outro fator a ser levado em consideração é o Indicador de Potência do Sinal Recebido (RSSI), que oscilou entre -110dBm e -106dBm, como visualizado na Figura 5.19. No datasheet do rádio LoRa, semtech SX1276, as especificações nos dizem que o rádio tem altas sensibilidades, podendo receber sinais de até -148dBm. Visto isso, é possível que transmissões mais longas sejam possíveis, mas devem reduzir significativamente as taxas de comunicações bem sucedidas.

Para o contexto de automação predial, não há transmissões de 450 metros, então o teste

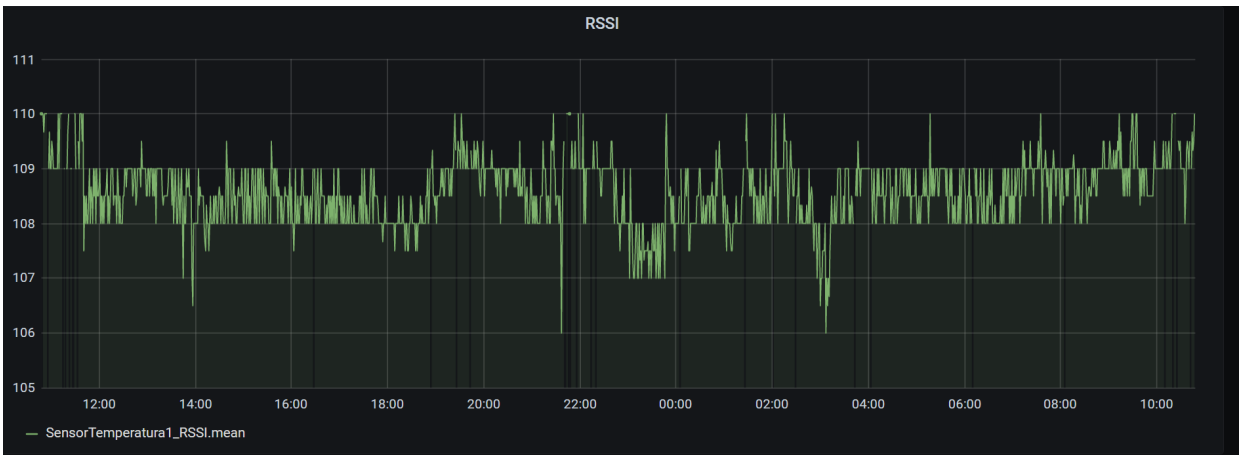

Figura 5.19: Indicador de Potência do Sinal Recebido durante o período de um dia.

realizado mostra que comunicações de longo alcance podem ser alcançadas, mas não irão ocorrer em prédios, pois não existem prédios com extensão próxima de 450m. Um ponto positivo desse teste, diz respeito à quantidade de paredes percorridas, pois em um prédio, normalmente diversos obstáculos existem, e o protocolo se mostrou eficiente a percorrer os mesmos.

Esse teste que confirma os longos alcances obtidos com o protocolo, reforça a hipótese de utilização da rede em arquitetura estrela, não sendo necessária a utilização de protocolos mais complexos para o envio dos dados, como a criação de redes *mesh*. Ao se comparar o alcance obtido com o alcance de tecnologias como Bluetooth, em torno de 10m, e Zigbee, em torno de 20m, percebemos uma grande diferença de alcances. Esse maior alcance possui um custo, que se encontra na baixa taxa de dados, ou seja, consegue enviar poucos dados por segundo em relação as outras tecnologias.

Ao se comparar LoRa com Bluetooth e Zigbee, a taxa de dados é menor em, aproximadamente, 1000 vezes e 10 vezes, respectivamente. No entanto, para o nosso problema, isso não se torna grande problema, pois não se deseja enviar dados extensos pela rede de comunicação. O protocolo tem a capacidade de enviar 16 bytes de *payload*, e para a aplicação construída, utilizou-se somente 6 bytes.

Capítulo 6

Conclusões

Sistemas de automação predial buscam automatizar o controle das condições do ambiente predial com a principal função de tornar os processos mais eficientes sem a intervenção humana. Esses sistemas têm a capacidade de melhorar o conforto dos ocupantes do prédio, ao mesmo tempo que geram economias.

Para alcançar esse objetivo, os sistemas possuem alguns obstáculos a serem vencidos, como a dificuldade de integração de sistemas, limitações energéticas e problemas de segurança e robustez nos dados.

Nesses sistemas, a conectividade dos dispositivos é um fator crucial, pois é a partir do conhecimento do ambiente, advindo dos dados, que o sistema pode tomar as melhores decisões para resolução dos problemas, de forma a maximizar sua eficiência.

Para isso, é necessário um sistema que seja regido por regras, de modo a garantir a segurança, robustez e eficiência. Para que seja possível gerar valor aos ocupantes, as mais distintas aplicações necessitam se integrar e comunicar.

A proposta principal deste trabalho foi desenvolver um protocolo IoT eficiente de fácil utilização e integração, que seja confiável em relação ao envio de mensagens, robusto, com bons alcances e energeticamente eficiente.

A primeira contribuição do projeto foi o desenvolvimento do protocolo de fácil utilização e de simples integração entre os diversos sistemas que podem ser criados utilizando o mesmo. É um protocolo para utilização em um contexto de automação predial.

A segunda contribuição do projeto se encontra na construção de um sistema de baixo consumo energético, ao se comparar com outros sistemas existentes. Além disso, um sistema que seja robusto a interferências, que consiga realizar comunicações a grandes distâncias com uma confiabilidade muito boa.

Ao se comparar com outros sistemas existentes, o protocolo desenvolvido possui muitas vantagens e seus pontos negativos não têm grande impacto no sistema final, pois podem ser contornados. Se mostrou ser uma boa solução para sistemas de automação predial.

6.1 Trabalhos Futuros

Um dos pontos a serem melhorados no trabalho, estão relacionados com a maximização da eficiência do protocolo a partir das mudanças dos parâmetros da tecnologia LoRa. Trabalhos que avaliem os parâmetros LoRa e consigam selecionar as melhores combinações de parâmetros a depender do cenário, podem aumentar a eficiência do protocolo.

A partir do protocolo desenvolvido, é possível também a criação de sistemas de automação predial, com a utilização de diversos dispositivos, que resolvam algum problema complexo, sem a necessidade de se preocupar com a conectividade do sistema.

Outra possível abordagem está relacionada com a melhoria do sistema supervisor, ao se adicionar sistemas para processamento de dados inteligente, como a utilização de redes neurais para processamento dos dados, que possa tomar as melhores decisões de controle, a partir das entradas do sistema.

Pensando em redes que necessitam de uma maior taxa de transmissão de dados, utilizar uma outra arquitetura, onde os dispositivos se encontrem mais próximos, de modo a ser possível diminuir a sensibilidade em troca de um aumento nas taxas de transmissão.

Por fim, é possível a criação de novas funcionalidades ao protocolo, de modo que o servidor se comunique com o dispositivo, por iniciativa do gateway. Para isso, os dispositivos precisam estar sincronizados com a rede, e por um período de tempo em seu ciclo, estar disponível para receber mensagens do gateway.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] COELHO, D. F. B.; CRUZ, V. H. do N. Eficiência energética nas edificações. In: BLUCHER, E. (Ed.). *Edifícios Inteligentes: uma visão das tecnologias aplicadas*. São Paulo: Blur, 2017. p. 61–88.

[2] FERRER, J. A.; GARRIDO, A. Eficiência Energética em Edifícios. In: *ONUDI Observatório e Energias Renováveis para a América Latina e Caribe*. [S.l.: s.n.], 2013.

[3] PETERSEN, D.; STEELE, J.; WILKERSON, J. WattBot: a residential electricity monitoring and feedback system. In: . [S.l.: s.n.], 2009. p. 2847–2852.

[4] CANO, M. V. et al. How can We Tackle Energy Efficiency in IoT BasedSmart Buildings? *Sensors (Basel, Switzerland)*, v. 14, p. 9582–9614, 06 2014.

[5] HATLER, D. G. M.; CHI, C. *Smart Energy Homes: A Market Dynamics Report*. San Diego, CA, 2012. Disponível em: <http://onworld.com/smartenergyhomes/>.

[6] KASTNER, W. et al. Communication Systems for Building Automation and Control. *Proceedings of the IEEE*, v. 93, p. 1178 – 1203, 07 2005.

[7] WANG, S. Intelligent buildings and building automation. *Intelligent Buildings and Building Automation*, p. 1–248, 01 2009.

[8] Park, P. et al. Wireless Network Design for Control Systems: A Survey. *IEEE Communications Surveys Tutorials*, v. 20, n. 2, p. 978–1013, 2018.

[9] FERRANDEZ, J. et al. Deployment of IoT Edge and Fog Computing Technologies to Develop Smart Building Services. *Sustainability*, v. 10, p. 3832:1–23, 10 2018.

[10] Suo, H. et al. Security in the Internet of Things: A Review. In: *2012 International Conference on Computer Science and Electronics Engineering*. [S.l.: s.n.], 2012. v. 3, p. 648–651.

[11] SOLUTIONS, S. *Sistema de Supervisão Automação Predial*. [S.l.], (Acessado em Novembro 2020). Disponível em: <https://www.shapesolutions.com.br/product-page/sistema-de-supervisão-automação-predial>.

[12] ASHTON, K. et al. That 'internet of things' Thing. *RFID journal*, v. 22, n. 7, p. 97–114, 2009.

[13] WHITMORE, A.; AGARWAL, A.; XU, L. D. The Internet of Things—A survey of topics and trends. *Information Systems Frontiers*, Springer, v. 17, n. 2, p. 261–274, 2015.

[14] NIC, N. Disruptive civil technologies: Six technologies with potential impacts on us interests out to 2025. *Tech. Rep.*, 2008.

[15] SAUTER, T. et al. The Evolution of Factory and Building Automation. *IEEE Industrial Electronics Magazine*, IEEE, v. 5, n. 3, p. 35–48, 2011.

[16] DOMINGUES, P. et al. Building automation systems: Concepts and technology review. *Computer Standards & Interfaces*, Elsevier, v. 45, p. 1–12, 2016.

[17] EGAN, D. The emergence of ZigBee in building automation and industrial controls. *Computing and Control Engineering*, IET, v. 16, n. 2, p. 14–19, 2005.

[18] GILL, K. et al. A zigbee-based home automation system. *IEEE Transactions on consumer Electronics*, IEEE, v. 55, n. 2, p. 422–430, 2009.

[19] CASTRO, P.; AFONSO, J. L.; AFONSO, J. A. A low-cost ZigBee-based wireless industrial automation system. In: *CONTROLO 2016*. [S.l.]: Springer, 2017. p. 739–749.

[20] SRISKANTHAN, N.; TAN, F.; KARANDE, A. Bluetooth based home automation system. *Microprocessors and microsystems*, Elsevier, v. 26, n. 6, p. 281–289, 2002.

[21] Raza, U.; Kulkarni, P.; Sooriyabandara, M. Low Power Wide Area Networks: An Overview. *IEEE Communications Surveys Tutorials*, v. 19, n. 2, p. 855–873, 2017.

[22] Badii, C. et al. Smart City IoT Platform Respecting GDPR Privacy and Security Aspects. *IEEE Access*, v. 8, p. 23601–23623, 2020.

[23] HUI, T.; SHERRATT, R.; DíAZ-SáNCHEZ, D. Major requirements for building Smart Homes in Smart Cities based on Internet of Things technologies. *Future Generation Computer Systems*, v. 76, 11 2016.

[24] AGARWAL, Y. et al. Occupancy-Driven Energy Management for Smart Building Automation. In: ACM. *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*. [S.l.], 2010. p. 1–6.

[25] TRAVIS, I. Management views automation. *IRE Transactions on Production Techniques*, v. 6, n. 1, p. 35–38, 1959.

[26] ZANELLA, A. et al. Internet of Things for Smart Cities. *Internet of Things Journal, IEEE*, v. 1, 01 2012.

[27] ISO 16484-5, Building automation and control systems (BACS) — Part 5: Data communication protocol. [S.l.], maio 2017. v. 6.

[28] ISO 14543-3-1, Home electronic systems (HES) architecture — Part 3-1: Communication layers — Application layer for network based control of HES Class 1. [S.l.], 2006. v. 1.

[29] ISO/IEC 14908-1, Information technology — Control network protocol — Part 1: Protocol stack. [S.l.], nov. 2012. v. 1.

[30] MODBUS, I. Modbus application protocol specification v1. 1a. *North Grafton, Massachusetts (www. modbus. org/specs. php)*, 2004.

[31] ALLIANCE, Z. *Zigbee specification version 1.0*. [S.l.]: April, 2005.

[32] VELIK, R.; ZUCKER, G. Autonomous perception and decision making in building automation. *IEEE Transactions on Industrial Electronics*, IEEE, v. 57, n. 11, p. 3645–3652, 2010.

[33] CAMBEIRO, J. et al. A building automation case study setup and challenges. In: . [S.l.: s.n.], 2018. p. 41–44.

[34] GUBBI, J. et al. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*, v. 29, 07 2012.

[35] AUTOMAçãO, G. *Automação residencial casa inteligente*. [S.l.], (Acessado em Novembro 2020). Disponível em: <https://www.guiaautomacao.com.br/automacao-residencial-casa-inteligente>.

[36] ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

[37] WEBER, R. Internet of things – Need for a new legal environment? *Computer Law Security Review*, v. 25, p. 522–527, 11 2009.

[38] ITU. Internet Reports 2005: The Internet of Things. In: *ITU Internet Report*. [S.l.: s.n.], 2005.

[39] SOLUTIONS, R. *Smart City, IoT and AI*. [S.l.], (Acessado em Novembro 2020). Disponível em: <https://riberasolutions.com/smart-city-iot-and-ai/>.

[40] MáQUINAS, F. *Telas e relatórios de IHM e Supervisório*. [S.l.], (Acessado em Dezembro 2020). Disponível em: <http://www.ferrazmaquinas.com.br/conteudo/telas-e-relatorios-de-ihm-e-supervisorio-ferraz-maquinas.html>.

[41] XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, IEEE, v. 10, n. 4, p. 2233–2243, 2014.

[42] Miao Wu et al. Research on the architecture of Internet of Things. In: *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*. [S.l.: s.n.], 2010. v. 5, p. V5–484–V5–487.

[43] RAY, P. A Survey on Internet of Things Architectures. *EAI Endorsed Transactions on Internet of Things*, v. 2, p. 151714, 12 2016.

[44] CRUZ, P. J. *AM VS FM*. [S.l.], 2013 (Acessado em Novembro 2020). Disponível em: <https://radioetecno.wordpress.com/2013/06/16/am-vs-fm/>.

[45] GISERMAN, B. L. G. e. A. C. J. L. F. *LoRa - Long Range*. [S.l.], (Acessado em Novembro 2020). Disponível em: <https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/lora/modulacao.html>.

[46] REYNDERS, B.; MEERT, W.; POLLIN, S. Range and coexistence analysis of long range unlicensed communication. In: . [S.l.: s.n.], 2016. p. 1–6.

[47] CENTENARO, M. et al. *Long-Range Communications in Unlicensed Bands: the Rising Stars in the IoT and Smart City Scenarios*. 2016.

[48] MOKOSMART. *lora Frequency*. [S.l.], (Acessado em Dezembro 2020). Disponível em: <https://www.mokosmart.com/lora-frequency/lora-frequency/>.

[49] GHOSLYA, S. *LoRa Decoding*. [S.l.], (Acessado em Novembro 2020). Disponível em: <https://www.sghoslya.com/p/lora$_9$.*html*>.

[50] GHOSLYA, S. *LoRa: Symbol Generation*. [S.l.], (Acessado em Novembro 2020). Disponível em: <https://www.sghoslya.com/p/lora-is-chirp-spread-spectrum.html>.

[51] WIKIPEDIA. *Largura de banda*. [S.l.], 2019 (Acessado em Dezembro 2020). Disponível em: <https://pt.wikipedia.org/wiki/Largura$_d$e$_b$*anda*>.

[52] KNIGHT, M. *Decoding LoRa*. [S.l.], (Acessado em Dezembro 2020). Disponível em: <https://lab.dsst.io/slides/33c3/slides/7945/73.jpg>.

[53] BOR, M. C.; VIDLER, J.; ROEDIG, U. LoRa for the Internet of Things. In: *EWSN*. [S.l.: s.n.], 2016. v. 16, p. 361–366.

[54] Mikhaylov, K. et al. Multi-RAT LPWAN in Smart Cities: Trial of LoRaWAN and NB-IoT Integration. In: *2018 IEEE International Conference on Communications (ICC)*. [S.l.: s.n.], 2018. p. 1–6.

[55] MAGRIN, D.; CENTENARO, M.; VANGELISTA, L. Performance evaluation of LoRa networks in a smart city scenario. In: IEEE. *2017 IEEE International Conference on Communications (ICC)*. [S.l.], 2017. p. 1–7.

[56] ORTIZ, F. M. Análise de desempenho de uma rede sem-fio de baixa potência e longo alcance para a internet das coisas. Universidade Federal do Rio de Janeiro, 2018.

[57] BOR, M. et al. Do LoRa Low-Power Wide-Area Networks Scale? In: . [S.l.: s.n.], 2016.

[58] SEMTECH. *Semtech SX1276*. [S.l.], (Acessado em Setembro 2020). Disponível em: <https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276>.

[59] SUL, T. *Sensor de Temperatura – DS18B20*. [S.l.], (Acessado em Setembro 2020). Disponível em: <https://techsuleletronicos.com.br/product/sensor-de-temperatura-ds18b20-prova-dagua/>.

[60] ONLINE, E. *Módulo Relé 3V a 5V 1 Canal*. [S.l.], (Acessado em Setembro 2020). Disponível em: <https://www.eletronlineshop.com.br/modulo-rele-5v-1-canal->.

[61] OPENHAB. *Openhab*. [S.l.], (Acessado em Setembro 2020). Disponível em: <https://www.openhab.org/>.

[62] AMAZON. *Placa Raspberry PI 3 Model B Quadcore 1.2ghz 1Gb Wifi Bluetooth*. [S.l.], (Acessado em Setembro 2020). Disponível em: <https://images-na.ssl-images-amazon.com/images/I/91zSu44%2B34L.$_{A}C_{S}L1500_jpg$>.

[63] SANTOS, R. *Topologia Estrela*. [S.l.], (Acessado em Setembro 2020). Disponível em: <https://sites.google.com/site/topologiasderedexd/topologia-estrela>.

[64] VMIA. *O que é redes Mesh*. [S.l.], (Acessado em Setembro 2020). Disponível em: <https://www.vmia.com.br/blog/index.php/2019/04/27/o-que-e-redes-mesh/>.

ANEXOS

I. PROGRAMAS UTILIZADOS

Neste anexo se encontram os programas desenvolvidos para este trabalho. Os códigos referentes a biblioteca do Protocolo Aéreo, os códigos do Gateway e da aplicação construída. Os seguintes programas foram realizados utilizando a linguagem C++:

- OnAirProtocol.h
- OnAirProtocol.cpp
- Gateway.ino
- SensorTemperatura.ino

Nas páginas seguintes estarão os códigos referentes aos programas feitos para a realização deste trabalho.

Abaixo se encontra o código das definições das funções e variáveis globais do Protocolo Aéreo.

OnAirProtocol.h

```
#include <LoRa.h>
#include <AES.h>


#define ESP32
//#define CANAL_FIXO
#define CANAL_SEMI_RAND

#ifdef ESP32
  #include <WiFi.h>
  #include <WiFiUdp.h>
  #include <NTPClient.h>
  #include <EEPROM.h>
  #include <DNSServer.h>
  #include <PubSubClient.h>
#else
  #include <Time.h>
  #include <DS1337RTC.h>
  #include <avr/wdt.h>
  #include <SPI.h>
  #include <Arduino.h>
#endif

#define SCK 5
#define MISO 19
#define MOSI 18
#define SS 21
#define RST 17
#define DI0 0
#define BAND 916.8E6
#define PABOOST true
#define CRC16 0x8005
#define EEPROM_SIZE 1024
#define ENC_TYPE_WEP 5
#define ENC_TYPE_TKIP 2
#define ENC_TYPE_CCMP 4
#define ENC_TYPE_NONE 7
#define ENC_TYPE_AUTO 8
#define RESET_DELAY 30000
#ifndef CONFIGSERVER_H
  #define CONFIGSERVER_H
#endif


/*
  DEFINICAO DOS CANAIS LORA -> CANAIS DE 0 A 7 CANAIS DE 8 A 15
                               UPLINK DOWNLINK
*/

#define CHANNEL_0 916.8E6
#define CHANNEL_1 917.0E6
#define CHANNEL_2 917.2E6
#define CHANNEL_3 917.4E6
#define CHANNEL_4 917.6E6
#define CHANNEL_5 917.8E6
#define CHANNEL_6 918.0E6
#define CHANNEL_7 918.2E6
#define CHANNEL_8 923.3E6
#define CHANNEL_9 923.9E6
#define CHANNEL_10 924.5E6
#define CHANNEL_11 925.1E6
#define CHANNEL_12 925.7E6
#define CHANNEL_13 926.3E6
#define CHANNEL_14 926.9E6
#define CHANNEL_15 927.5E6


#define RTS 1
#define CTS 2
#define ACK 3
#define DATA 4
#define PRIORITY 7
#define DOS 14
#define TEST 15
#define MAC_ID_INVALIDO -1
#define SEQ_ID_INVALIDO -1
#define UP 1
```

```
#define DOWN 2
#define DEBUG 1
#define SEND_DATA 1
#define SEND_PRIORITY 2
#define SEND_TEST 3


#ifdef ESP32
  void _start_LoRa();
  void start_WiFi();
  void checar_wifi();
  int ping_server();
  void callback(char* topic, byte* payload, unsigned int length);
  //int Keepalive();
  void listNetworks();
  bool check_ssid(String ssid);
  void print_status();
  uint16_t gen_crc16(uint8_t *data, uint16_t size);
  int check_crc(unsigned char *bufwifi);
  void waitCTS();
  void addTime(unsigned char *response);
  void montartoSend(unsigned char *buf, char *toSend, String &payload, uint8_t rssi);
  int send_data(char * data, int len, unsigned char * buf);
  void _receive();
  void start_AP(const char * ssid, const char * password);
  void start_DNS(String);
  void Handle(String Json);
  String respond(String);
  String Error400();
  String Error404();
  int parse(String Json, String * values);
  String read_flash();
  void save_config(const char * str);
  void save_ipfixo(String str1, String str2, String str3);
  void parsing(char * str, char * arg1, char * arg2, char * arg3, char * arg4);
  void delete_config();
  String get_path(String request);
  String get_method(String request);
  void get_reset(WiFiClient client);
  void get_root(WiFiClient client, String);
  void get_list(WiFiClient client);
  void get_test(WiFiClient client, String (*test)());
  void post_test(String currentLine, void (*test)(String), WiFiClient client);
  void post_setup(String currentLine, WiFiClient client);
  void post_reset(WiFiClient client);
  void resolve_get(String response, WiFiClient client, String);
  void resolve_post(String currentLine, WiFiClient client);
  void resolve_delete(String currentLine, WiFiClient client);
  void start_config_server(const char * ssid, const char * password, String, String);
  void use_MDNS(String);
  String read_request(WiFiClient client);
  String get_protocol(String request);
  String encryption(byte encryption);
  void IRAM_ATTR resetModule();
  void configureWatchdog();
#else
  void start_LoRa();
  int initLoRa();
    void configuraRTC (int tempo);
  void sendMessage(int FSM_SEND);
  void waitSEND();
#endif


int rcvLora(unsigned char * buf, long frequency_2);
void gen_type(unsigned char * msg, int type, int UpouDown);
void gen_SEQ_ID(unsigned char * msg, int UpouDown, int type);
void gen_rssi(unsigned char *msg, int *rssi);
void gen_APP_ID(unsigned char * msg);
void gen_DEV_ID(unsigned char * msg);
void gen_MAC_ID(unsigned char * msg, int UpouDown);
void gen_PAYLOAD(unsigned char *msg, byte *PAYLOAD, int UpouDown);
void SEND_LoRa(const unsigned char * msg, int len, char *CHANNEL, long frequency_1, long *frequency_2, int UpouDown);
void get_CHANNEL(char *CHANNEL, int frequency, int UpouDown);
long set_CHANNEL(char *CHANNEL, int UpouDown);
char convertCharToHex(char ch, int opcao);
char convertDecToHex(int dec, int opcao);
char convertDecToChar(int dec);
void gen_CHANNEL(unsigned char * msg, int UpouDown);
int parseLoRa_Msg(int Msg_size, unsigned char * Msg, char * APP_ID, char * DEV_ID, char * CH, String &PAYLOAD, int *Rssi, unsigned
    ↪ long seq_id, int UpouDown);
```

```
158 unsigned long hexToDec(String hexString);
159 int getRSSI(const unsigned char * Msg_copy);
160 void DecToHex(unsigned long Dec, char *cvt);
161 void getMsg_AppID(const unsigned char * Msg_copy, char * AppID);
162 void getMsg_SeqID(const unsigned char * Msg_copy, unsigned long *seq_ID, int UpouDown);
163 void getMsg_DevID(const unsigned char * Msg_copy, char * DevID);
164 int check_SeqID(const unsigned char *Msg_copy, unsigned long seq_id, int UpouDown);
165 int cripto(int key, const unsigned char *Msg_copy, byte *inp);
166 bool check_MacID(const unsigned char * Msg_copy, int UpouDown);
167 void getMsg_Type(unsigned char * Msg_copy, int UpouDown);
168 void getMsg_CHANNEL(const unsigned char * Msg_copy, char *CH, int UpouDown);
169 void montar_resposta(unsigned char *buf, int *rssi, int type, int UpouDown);
170 String ajustConversion(String ch);
171 int ajustchar(char* seqid, int opcao);
172
173
174 extern String payload;
175 extern char _devId[5];
176 extern char _appId[5];
177 extern bool criptografia;
178 #ifdef ESP32
179   extern const int pinoLED;
180   extern int toConfig;
181   //extern unsigned long comeco;
182   //extern unsigned long fim;
183   //extern unsigned long _comeco;
184   //extern unsigned long _fim;
185   extern char _HomeBaseId[5];
186   extern hw_timer_t *timer;
187 #else
188   extern bool at, alarm;
189   extern int rch;
190   extern int interval;
191   extern unsigned char received[20];
192   extern int tam_recv;
193   extern bool test;
194   extern int sinal;
195   extern byte AppKey[16];
196 #endif
```

Abaixo se encontra o código do Protocolo Aéreo.

OnAirProtocol.cpp

```
#include "OnAirProtocol.h"

String tmp="";
int interval = 1;
unsigned char received[20]="";
int tam_recv=0;
bool test = false;
int rch=0;
bool at = false;
int sinal = -1;
bool priority = false;



byte key1[16]= {15,31,26,23,45,59,76,27,84,91,110,3,2,1,2,51}, key2[16]= {34,65,27,39,43,55,6,27,86,90,40,71,18,63,24,1};
byte key3[16]= {102,31,52,83,47,50,23,63,6,96,12,80,27,92,11,15}, key4[16]= {60,17,29,3,54,25,56,77,88,99,140,21,142,113,114,175};
byte key5[16]= {52,66,249,255,97,84,75,137,84,56,181,178,62,156,144,155}, key6[16]= {50,61,82,4,87,89,91,67,34,171,128,103,164,42,56,15};
byte key7[16]= {156,71,112,83,48,53,67,76,87,93,0,21,28,143,61,51}, key8[16]= {50,61,85,33,44,59,62,76,87,89,180,165,112,143,65,74};
byte key9[16]= {56,11,21,37,43,58,68,26,138,9,13,162,73,184,185}, key10[16]= {56,12,24,234,17,54,66,77,88,53,122,111,182,103,4,35};

//App Key para criptografar payload
byte AppKey[16] = {102,31,52,83,47,50,23,63,6,156,144,155,27,92,11,15};

uint8_t FSM = 0; //Controla a máquina de estados

#ifdef ESP32
  // Constantes
  WiFiServer server(80);
  const byte DNS_PORT = 53;
  IPAddress APIP(192, 168, 4, 1);
  DNSServer dnsserver;

  // Globals
  String AP_name = "";
  String AP_pass = "";
  bool ip_fixo = false;
  int _ip1, _ip2, _ip3, _ip4, _gate1, _gate2, _gate3, _gate4, _sub1, _sub2, _sub3, _sub4;
  String (*test_get)() = NULL;
  void (*test_post)(String) = NULL;
  bool servidor = false;
  bool ping = false;
  bool mqtt = false;
  char retorno[20];
  int leng = 0;
  int ping_ = 0;

  //faz o controle do temporizador (interrupção por tempo)
  hw_timer_t *timer = NULL;

#else
  bool alarm = false;
#endif



#ifdef ESP32

  void _start_LoRa() {
    Serial.println("LoRa Receiver");
    SPI.begin(SCK,MISO,MOSI,SS);
    LoRa.setPins(SS,RST,DI0);
    if (!LoRa.begin(BAND,PABOOST )) {
        Serial.println("Starting LoRa failed!");
        while (1);
    }
    LoRa.setPreambleLength(6);
    LoRa.setTxPower(17, PA_OUTPUT_PA_BOOST_PIN);
    LoRa.setSpreadingFactor(10);
    LoRa.setSignalBandwidth(125E3);
    LoRa.setCodingRate4(4);
    LoRa.setSyncWord(0xAA);
    LoRa.beginPacket(true); //implicitHeader
    LoRa.crc();
  }

  void start_WiFi(){
    int cont=0;

```

```
78
79      listNetworks();
80      Serial.println();
81      Serial.print("Configurated SSID: ");
82      Serial.println(AP_name.substring(AP_name.indexOf(":") + 1).c_str());
83
84      Serial.print("Configurated password: ");
85      Serial.println(AP_pass.substring(AP_pass.indexOf(":") + 1).c_str());
86
87      if(check_ssid(AP_name.substring(AP_name.indexOf(":") + 1))){
88        Serial.println("Network found");
89        if(ip_fixo == true) {
90          // Static IP addr
91          IPAddress local_IP(_ip1, _ip2, _ip3, _ip4);
92          //Gateway IP addr
93          IPAddress gateway(_gate1, _gate2, _gate3, _gate4);
94          //Subnet IP addr
95          IPAddress subnet(_sub1, _sub2, _sub3, _sub4);
96
97          // Configures static IP address
98          if (!WiFi.config(local_IP, gateway, subnet)) {
99              Serial.println(F("Failed to configure Static IP"));
100           }
101           else{
102             Serial.println(F("Configured Static IP"));
103           }
104       }
105       WiFi.begin(AP_name.substring(AP_name.indexOf(":") + 1).c_str(), AP_pass.substring(AP_pass.indexOf(":") + 1).c_str());
106       delay(10000);
107     }
108     else {
109         Serial.println("Network not found");
110         cont = 5;
111     }
112
113     while((WiFi.status() != WL_CONNECTED) && (cont!=10)) {
114       digitalWrite(pinoLED, HIGH);
115       delay(500);
116       digitalWrite(pinoLED, LOW);
117       delay(500);
118       cont++;
119     }
120     if(WiFi.status() == WL_CONNECTED) {
121         digitalWrite(pinoLED, HIGH);
122         toConfig = 0;
123         Serial.println("Conectado a rede " + AP_name.substring(AP_name.indexOf(":") + 1));
124         Serial.println("");
125         Serial.println("Endereço IP: ");
126         Serial.println(WiFi.localIP());
127         delay(1500);
128     }
129     else{
130       WiFi.disconnect();
131     }
132   }
133
134   void checar_wifi() {
135     unsigned long inicio = millis();
136
137     while(WiFi.status() != WL_CONNECTED) {
138       timerWrite(timer, 0); //reseta o temporizador do Watchdog
139         digitalWrite(pinoLED, HIGH);
140         delay(500);
141         digitalWrite(pinoLED, LOW);
142       delay(500);
143       if(millis() - inicio > 300000 ){
144         ESP.restart();
145             while(true) yield();
146       }
147     }
148
149     digitalWrite(pinoLED, HIGH);
150   }
151
152   int ping_server() {
153     //MQTT
154     WiFiClient espClient;
155     PubSubClient client(espClient);
156
157     const char* mqtt_server = "192.168.15.6";
158     const char* mqttDeviceID = "Homebase";
```

```
159
160     int i = 0;
161     timerWrite(timer, 0);
162     client.setServer(mqtt_server, 1883);
163     client.setCallback(callback);
164
165     unsigned long comeco = 0, fim = 0;
166     comeco = millis();
167     client.connect(mqttDeviceID, "openhabian", "liga159753");
168     if ( (millis() - comeco) < 1000 ) {
169       client.subscribe("automation/ping");
170       while ((ping == false) && ((millis() - comeco) < 1000)){
171       client.loop();
172       }
173       if (ping == true) {
174       i = 1;
175       }
176       client.disconnect();
177       fim = millis();
178       ping_ = fim - comeco;
179       if (i == 0) {
180       Serial.println(F("[DEBUG] FAIL TO PING SERVER"));
181       i = -1;
182       }
183       else {
184       Serial.print(F("[DEBUG] SERVER - Tempo de ping: "));
185       Serial.print(fim - comeco);
186       Serial.println(F("ms"));
187       }
188     }
189     else {
190       Serial.println(F("[DEBUG] FAIL TO PING SERVER"));
191       i = -1;
192     }
193
194     ping = false;
195
196       return i;
197 }
198
199 void callback(char* topic, byte* payload, unsigned int length) {
200   char topico1[40] = "automation/SensorTemperatura1/aux2";
201   char topico2[40] = "automation/ping";
202
203
204   if(strcmp(topic, topico2) == 0){ // Ping
205     ping = true;
206   }
207   else {
208     if(strcmp(topic, topico1) == 0) { //Temperatura de referência
209         for (int i = 0; i < length; i++) {
210           retorno[i] = (char)payload[i];
211         }
212         leng = length;
213         mqtt = true;
214     }
215     else {
216       Serial.println("Unknown topic");
217     }
218   }
219
220 }
221
222
223   void listNetworks() {
224     // scan for nearby networks:
225       Serial.println("** Scan Networks **");
226       int numSsid = WiFi.scanNetworks();
227       if (numSsid == -1) {
228         Serial.println("Couldn't get a wifi connection");
229         while (true);
230       }
231
232       // print the list of networks seen:
233       Serial.print("number of available networks:");
234       Serial.println(numSsid);
235
236       // print the network number and name for each network found:
237       for (int thisNet = 0; thisNet < numSsid; thisNet++) {
238         Serial.print(thisNet);
239         Serial.print(") ");
```

```
        Serial.print(WiFi.SSID(thisNet));
        Serial.print("\tSignal: ");
        Serial.print(WiFi.RSSI(thisNet));
        Serial.print(" dBm");
        Serial.println();
      }
    }

bool check_ssid(String ssid){
    int num = WiFi.scanNetworks();
    if (num == -1){
      Serial.println("Failed to find any network");
    }
    else {
      for (int i = 0; i < num; i++){
          if(String(WiFi.SSID(i)) == ssid){
            return true;
          }
        }
    }
    return false;
  }

void print_status(){
    Serial.print("WiFi.status() == ");
    switch(WiFi.status()){
        case WL_IDLE_STATUS:
          Serial.println("WL_IDLE_STATUS");
          break;
        case WL_NO_SSID_AVAIL:
          Serial.println("WL_NO_SSID_AVAIL");
          break;
        case WL_SCAN_COMPLETED:
          Serial.println("WL_SCAN_COMPLETED");
          break;
        case WL_CONNECTED:
          Serial.println("WL_CONNECTED");
          break;
        case WL_CONNECT_FAILED:
          Serial.println("WL_CONNECT_FAILED");
          break;
        case WL_CONNECTION_LOST:
          Serial.println("WL_CONNECTION_LOST");
          break;
        case WL_DISCONNECTED:
          Serial.println("WL_DISCONNECTED");
    }
  }

uint16_t gen_crc16(uint8_t *data, uint16_t size){
    uint16_t out = 0;
    int bits_read = 0, bit_flag;

  /* Sanity check: */
    if(data == NULL)
        return 0;

    while(size > 0){

        bit_flag = out >> 15;

      /* Get next bit: */
        out <<= 1;
        out |= (*data >> bits_read) & 1; // item a) work from the least significant bits

        /* Increment bit counter: */
        bits_read++;
        if(bits_read > 7){
            bits_read = 0;
            data++;
            size--;
        }

        /* Cycle check: */
        if(bit_flag){
            out ^= CRC16;
        }

    }

    // item b) "push out" the last 16 bits
```

```
    int i;
    for (i = 0; i < 16; ++i) {
        bit_flag = out >> 15;
        out <<= 1;
        if(bit_flag)
            out ^= CRC16;
    }

    // item c) reverse the bits
    uint16_t crc = 0;
    i = 0x8000;
    int j = 0x0001;
    for (; i != 0; i >>=1, j <<= 1) {
        if (i & out) crc |= j;
    }

    return crc;
  }

int check_crc(unsigned char *bufwifi){
    uint8_t data[12];
    uint16_t _crc, crc;
    uint16_t aux;

  //if(servidor == false) {
    for(int j=0; j<12; j++) {
        data[j] = bufwifi[j];
      }
      crc = gen_crc16(data, 12);

      _crc = bufwifi[12];
      _crc = _crc << 8;
      _crc = _crc | (bufwifi[13]);
  //}
    /*else{
      for(int j=0; j<8; j++) {
        data[j] = bufwifi[j];
      }
      crc = gen_crc16(data, 8);

      _crc = bufwifi[8];
      _crc = _crc << 8;
      _crc = _crc | (bufwifi[9]);
    }*/


    if(_crc == crc) {
      return 1;
    }
    else {
      return 0;
    }
  }

void waitCTS() { //Para evitar colisão entre duas HomeBases, gera um delay aleatório de 0-100ms

    long randomico;

    //Serial.println(esp_random());

    randomico = random(10) + 1;
    //Serial.print(F("Numero sorteado: ")); Serial.println(randomico);
    delay(30 * randomico);
  }

void addTime(unsigned char *response) {
    unsigned long tempo;
    char _tempo[4]="";

    // Configurações do Servidor NTP
    const char* servidorNTP = "a.st1.ntp.br"; // Servidor NTP para pesquisar a hora

    const int fusoHorario = -10800; // Fuso horário em segundos (-03h = -10800 seg)
    const int taxaDeAtualizacao = 1800000; // Taxa de atualização do servidor NTP em milisegundos

    WiFiUDP ntpUDP; // Declaração do Protocolo UDP
    NTPClient timeClient(ntpUDP, servidorNTP, fusoHorario, 60000);

    // Iniciar cliente de aquisição do tempo
    timeClient.begin();
    timeClient.update();
```

```
402         tempo = timeClient.getEpochTime();
403         timeClient.end();
404 
405         DecToHex(tempo, _tempo); //Pega o tempo em hexa em uma string de tamanho 4
406 
407         response[12] = _tempo[0]; //Preenche o buffer a ser enviado com o payload(tempo atual)
408         response[13] = _tempo[1];
409         response[14] = _tempo[2];
410         response[15] = _tempo[3];
411       }
412 
413       void montartoSend(unsigned char *buf, char *toSend, String &payload, uint8_t rssi) {
414         char BaseID[5];
415         BaseID[0] = _HomeBaseId[0];
416         BaseID[1] = _HomeBaseId[1];
417         BaseID[2] = _HomeBaseId[2];
418         BaseID[3] = _HomeBaseId[3];
419         BaseID[4] = _HomeBaseId[4];
420         float snr = abs(LoRa.packetSnr());
421         //uint8_t rssi = abs(LoRa.packetRssi());
422 
423         for(int i=0; i<10; i++) {
424             toSend[i] = buf[i];
425         }
426         toSend[10] = (uint8_t)((ping_ & 0xFF00) >> 8);
427         toSend[11] = (uint8_t)(ping_ & 0x00FF);
428         toSend[12] = buf[18];
429         toSend[13] = buf[19];
430         toSend[14] = buf[20];
431         for(int i=15; i<20; i++) {
432             toSend[i] = BaseID[i-15];
433         }
434         toSend[20] = (uint8_t)((rssi & 0xFF00) >> 8);
435         toSend[21] = (uint8_t)(rssi & 0x00FF);
436 
437         toSend[22] = (uint8_t)((int)snr & 0xFF);
438           snr = (snr - (int)snr) * 10;
439           toSend[23] = (uint8_t)((int)snr & 0xFF);
440 
441         int tam = payload.length();
442         for(int i=24; i<24+tam; i++) {
443             toSend[i] = payload.charAt(i-24);
444         }
445       }
446 
447       int send_data(char * data, int len, unsigned char * buf){
448         //MQTT
449         WiFiClient espClient;
450         PubSubClient client(espClient);
451 
452         const char* mqtt_server = "192.168.15.6";
453         const char* mqttDeviceID = "";
454         int i = 0;
455         timerWrite(timer, 0);
456         client.setServer(mqtt_server, 1883);
457         client.setCallback(callback);
458 
459         if((data[0] == 0x00) && (data[1] == 0x00) && (data[2] == 0x00) && (data[3] == 0x00) && (data[4] == 0x01) ) {
460           if((data[5] == 0x00) && (data[6] == 0x00) && (data[7] == 0x00) && (data[8] == 0x00) && (data[9] == 0x01) ) {
461             mqttDeviceID = "SensorTemperatura1";
462           }
463           else{
464             if((data[5] == 0x00) && (data[6] == 0x00) && (data[7] == 0x00) && (data[8] == 0x00) && (data[9] == 0x02) ) {
465               mqttDeviceID = "SensorTemperatura2";
466             }
467           }
468 
469           unsigned long comeco = 0;
470           comeco = millis();
471           client.connect(mqttDeviceID, "openhabian", "liga159753");
472           if ( (millis() - comeco) < 4000 ) {
473 
474             char temp[6] = {0,0,0,0,0};
475             char hvac[10] = {0,0,0,0,0,0,0,0,0,0};
476             char rssi[4] = {0,0,0};
477             char snr[6] = {0,0,0,0,0};
478             char bateria[6] = {0,0,0,0,0};
479 
480             int dec = (unsigned char) ((data[24] & 0xF0) >> 4) * 16 + (data[24] & 0x0F);
481             dec = dec - 55;
482             if(dec > 99) {
```

```
483             int j, k, l;
484             j = dec / 100;
485             k = ((dec / 10) % 10);
486             l = dec % 10;
487             temp[0] = convertDecToChar(j);
488             temp[1] = convertDecToChar(k);
489             temp[2] = convertDecToChar(l);
490             temp[3] = '.';
491             temp[4] = (((data[25] & 0xF0) >> 4) + 0x30);
492           }
493           else{
494             if(dec > 9) {
495               int k, l;
496               k = dec / 10;
497               l = dec % 10;
498               temp[0] = convertDecToChar(k);
499               temp[1] = convertDecToChar(l);
500               temp[2] = '.';
501               temp[3] = (((data[25] & 0xF0) >> 4) + 0x30);
502             }
503             else { // Menor ou igual a 9
504               temp[0] = convertDecToChar(dec);
505               temp[1] = '.';
506               temp[2] = (((data[25] & 0xF0) >> 4) + 0x30);
507             }
508           }
509
510           if( (data[25] & 0x0F) == 0x00 ) {
511             hvac[0] = 'D';
512             hvac[1] = 'E';
513             hvac[2] = 'S';
514             hvac[3] = 'L';
515             hvac[4] = 'I';
516             hvac[5] = 'G';
517             hvac[6] = 'A';
518             hvac[7] = 'D';
519             hvac[8] = 'O';
520           }
521           else {
522             if((data[25] & 0x0F) == 0x01) {
523               hvac[0] = 'L';
524               hvac[1] = 'I';
525               hvac[2] = 'G';
526               hvac[3] = 'A';
527               hvac[4] = 'D';
528               hvac[5] = 'O';
529             }
530             else {
531               hvac[0] = 'E';
532               hvac[1] = 'R';
533               hvac[2] = 'R';
534               hvac[3] = '0';
535             }
536           }
537
538           dec = (unsigned char) ((data[21] & 0xF0) >> 4) * 16 + (data[21] & 0x0F);
539           if(dec > 99) {
540             int j, k, l;
541             j = dec / 100;
542             k = ((dec / 10) % 10);
543             l = dec % 10;
544             rssi[0] = convertDecToChar(j);
545             rssi[1] = convertDecToChar(k);
546             rssi[2] = convertDecToChar(l);
547           }
548           else{
549             if(dec > 9) {
550               int k, l;
551               k = dec / 10;
552               l = dec % 10;
553               rssi[0] = convertDecToChar(k);
554               rssi[1] = convertDecToChar(l);
555             }
556             else { // Menor ou igual a 9
557               rssi[0] = convertDecToChar(dec);
558             }
559           }
560
561           dec = (unsigned char) ((data[22] & 0xF0) >> 4) * 16 + (data[22] & 0x0F);
562           if(dec > 99) {
563             int j, k, l;
```

```
564             j = dec / 100;
565             k = ((dec / 10) % 10);
566             l = dec % 10;
567             snr[0] = convertDecToChar(j);
568             snr[1] = convertDecToChar(k);
569             snr[2] = convertDecToChar(l);
570             snr[3] = '.';
571             snr[4] = (data[23] + 0x30);
572           }
573           else{
574             if(dec > 9) {
575               int k, l;
576               k = dec / 10;
577               l = dec % 10;
578               snr[0] = convertDecToChar(k);
579               snr[1] = convertDecToChar(l);
580               snr[2] = '.';
581               snr[3] = (data[23] + 0x30);
582             }
583             else { // Menor ou igual a 9
584               snr[0] = convertDecToChar(dec);
585               snr[1] = '.';
586               snr[2] = (data[23] + 0x30);
587             }
588           }
589
590
591           dec = (unsigned char) ((data[26] & 0xF0) >> 4) * 16 + (data[26] & 0x0F);
592           float bat = dec * 0.393;
593           if((int)bat > 99) {
594             int j, k, l;
595             j = 1;
596             k = 0;
597             l = 0;
598             bateria[0] = convertDecToChar(j);
599             bateria[1] = convertDecToChar(k);
600             bateria[2] = convertDecToChar(l);
601             bateria[3] = '.';
602             bateria[4] = '0';
603           }
604           else{
605             if((int)bat > 9) {
606               int k, l;
607               float m;
608               k = (int)bat / 10;
609               l = (int)bat % 10;
610               m = (bat - (int) bat) * 10;
611               bateria[0] = convertDecToChar(k);
612               bateria[1] = convertDecToChar(l);
613               bateria[2] = '.';
614               bateria[3] = ((int)m + 0x30);
615             }
616             else { // Menor ou igual a 9
617               float m;
618               m = (bat - (int) bat) * 10;
619               bateria[0] = convertDecToChar(dec);
620               bateria[1] = '.';
621               bateria[2] = ((int)m + 0x30);
622             }
623           }
624
625           client.publish("automation/SensorTemperatura1/temperatura", temp, true);
626           client.publish("automation/SensorTemperatura1/hvac", hvac, true);
627           client.publish("automation/SensorTemperatura1/rssi", rssi, true);
628           client.publish("automation/SensorTemperatura1/snr", snr, true);
629           client.publish("automation/SensorTemperatura1/bateria", bateria, true);
630           client.subscribe("automation/SensorTemperatura1/aux2");
631           while ((mqtt == false) && ((millis() - comeco) < 4000)){
632             client.loop();
633           }
634           if (mqtt == true) {
635             i = 1;
636           }
637           client.disconnect();
638           if (i == 0) {
639             Serial.println(F("[DEBUG] CONNECTION BROKEN SERVER"));
640             i = -1;
641           }
642           else {
643             Serial.print(F("socketClient: [LOG] SERVER -—> RESPONSE FROM SERVER: "));
644             Serial.print("TEMPERATURA DE REFERENCIA: ");
```

```
          for (int c = 0; c < leng; c++) {
            Serial.print(retorno[c]);
          }
          Serial.println(" C");

          String conversao = "00";
          conversao.setCharAt(0, retorno[0]);
          conversao.setCharAt(1, retorno[1]);
          Serial.println(conversao);
          dec = conversao.toInt();

          int aux[10] = {0,0,0,0,0,0,0,0,0,0};
          char tmp[5];
          int k=0, j=0, tam=0;

          while((dec/16) != 0) { //Gera vetor invertido com cada um dos hexas
            aux[k] = dec % 16;
            k++;
            dec = (dec / 16);
          }
          aux[k] = dec % 16;
          for(j=0; j<k+1; j++) { //Inverte o vetor e converte para hexa
            int a = aux[k-j];
            tmp[j] = convertDecToChar(a);
          }
          tam = k + 1;
          byte extract;
          char a;
          char b;
          switch(tam){
            case 1:

              a = '0';
              b = tmp[0];
              extract = ((convertCharToHex(a,1)) + (convertCharToHex(b,2)));
              buf[0] = extract;
              Serial.println(buf[0], HEX);

            break;

            case 2:

              a = tmp[0];
              b = tmp[1];
              extract = ((convertCharToHex(a,1)) + (convertCharToHex(b,2)));
              buf[0] = extract;
              Serial.println(buf[0], HEX);

            break;
          }


        }
      }
      else {
        Serial.println("[DEBUG] Timeout MQTT SERVER");
        i = -1;
      }

    }


    mqtt = false;
    for (int b = 0; b < leng; b++) {
        retorno[b] = 0;
    }
    leng = 0;


    return i;
}



void _receive() {
  unsigned char buf[37] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  unsigned char response[25] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  unsigned char bufwifi[64] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  int i=0, tam, rssi=0, _rssi=0;
  char APP_ID[5]="", DEV_ID[5]="", CH[2]="";
  long frequency_1=0, frequency_2=0;
```

```
726     unsigned long seq_id=0;
727     char toSend[40] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
728     #ifdef CANAL_SEMI_RAND
729       char CH_SEMI_RAND[2]="";
730     #endif
731     #ifdef CANAL_FIXO
732       char CH_FIXO[2]="";
733     #endif
734
735     //servidor = false;
736     frequency_2 = CHANNEL_5; //CANAL INICIAL DEFINIDO AQUI
737
738     #ifdef CANAL_FIXO
739       get_CHANNEL(CH_FIXO, (int)frequency_2, UP);
740     #endif
741
742     LoRa.setSpreadingFactor(10);
743     LoRa.setTxPower(17, PA_OUTPUT_PA_BOOST_PIN);
744     i = rcvLora(buf, frequency_2);
745     if(i > 0) {
746       //_comeco = millis();
747       getMsg_SeqID(buf, &seq_id, UP);
748       seq_id--;
749       Serial.println();
750       Serial.println();
751       int n = parseLoRa_Msg(i,buf,APP_ID, DEV_ID,CH,payload, &rssi, seq_id, UP);
752       if(n == -1) { //Se for -1, ocorreu erro de MAC ou SEQID
753         LoRa.idle();
754         //LoRa.sleep();
755         delay(10);
756         return;
757       }
758       getMsg_SeqID(buf, &seq_id, UP);
759       if(FSM == 1){ //Recebeu um RTS, deve enviar um CTS se todas as verificações estiverem corretas
760         #ifdef CANAL_FIXO
761             frequency_1 = set_CHANNEL(CH_FIXO, DOWN);
762         #else
763             frequency_1 = set_CHANNEL(CH, DOWN);
764         #endif
765
766         montar_resposta(response, &rssi, CTS, DOWN);
767         getMsg_CHANNEL(response, CH, DOWN);
768
769         #ifdef CANAL_SEMI_RAND
770           for(int j=0; j<2; j++){
771             CH_SEMI_RAND[j] = CH[j];
772             //Serial.println(CH_FIXO[0], HEX);
773           }
774         #endif
775         gen_MAC_ID(response, DOWN);
776
777         Serial.print("LoRa-Message: [LOG] --> APP ID: ");
778         for(int j=0; j<5; j++) {
779           Serial.print(APP_ID[j], HEX);
780           Serial.print(" ");
781         }
782         Serial.println();
783         Serial.print("LoRa-Message: [LOG] --> DEVICE: ");
784         for(int j=0; j<5; j++) {
785           Serial.print(DEV_ID[j], HEX);
786           Serial.print(" ");
787         }
788         Serial.println();
789         Serial.print("LoRa-Message: [LOG] --> MAC: ");
790         for(int j=10; j<18; j++) {
791           Serial.print(buf[j], HEX);
792           Serial.print(" ");
793         }
794         Serial.println();
795         Serial.print("LoRa-Message: [LOG] --> SEQUENCE: ");
796         Serial.println(seq_id, HEX);
797
798         Serial.println("LoRa-Message: [LOG] --> TYPE: 1");
799         Serial.print("LoRa-Message: [LOG] --> CHANNEL: ");
800         #ifdef CANAL_SEMI_RAND
801           Serial.print(CH_SEMI_RAND[0], HEX);
802         #elif defined(CANAL_FIXO)
803           Serial.print(CH_FIXO[0], HEX);
804         #else
805           Serial.print(CH[0], HEX);
806         #endif
```

```
807         Serial.print(" => ");
808         Serial.println(frequency_1);
809
810         Serial.print("LoRa-Response: [DEBUG] --> RSSI: ");
811         Serial.println(rssi);
812
813         _rssi = rssi;
814
815         int l;
816         l = ping_server();
817         if(l == -1) {
818           //Erro de ping -> Enviar DOS e sair da funcao
819           montar_resposta(response, &rssi, DOS, DOWN);
820           gen_MAC_ID(response, DOWN);
821           getMsg_CHANNEL(response, CH, DOWN);
822           Serial.print("LoRa-Response: [LOG] --> RESPONSE: ");
823           for(int j=0; j<12; j++) {
824             Serial.print(response[j], HEX);
825             Serial.print(" ");
826           }
827           Serial.println();
828           #ifdef CANAL_SEMI_RAND
829             waitCTS();
830             SEND_LoRa(response, 12, CH_SEMI_RAND, frequency_1, &frequency_2, DOWN);
831           #elif defined(CANAL_FIXO)
832             waitCTS();
833             SEND_LoRa(response, 12, CH_FIXO, frequency_1, &frequency_2, DOWN);
834           #else
835             waitCTS();
836             SEND_LoRa(response, 12, CH, frequency_1, &frequency_2, DOWN);
837           #endif
838
839           checar_wifi();
840
841           LoRa.idle();
842           //LoRa.sleep();
843           delay(10);
844           return;
845         }
846           getMsg_CHANNEL(response, CH, DOWN);
847           Serial.print("LoRa-Response: [LOG] --> RESPONSE: ");
848           for(int j=0; j<12; j++) {
849             Serial.print(response[j], HEX);
850             Serial.print(" ");
851           }
852           Serial.println();
853           #ifdef CANAL_SEMI_RAND
854             waitCTS();
855             SEND_LoRa(response, 12, CH_SEMI_RAND, frequency_1, &frequency_2, DOWN);
856           #elif defined(CANAL_FIXO)
857             waitCTS();
858             SEND_LoRa(response, 12, CH_FIXO, frequency_1, &frequency_2, DOWN);
859           #else
860             waitCTS();
861             SEND_LoRa(response, 12, CH, frequency_1, &frequency_2, DOWN);
862           #endif
863
864
865           if(rssi < 110) { //O valor de rssi éabsoluto, então <110 -> rapido
866             //Serial.println("Velocidade setada para rápido");
867             LoRa.setSpreadingFactor(7);
868           }
869           i = rcvLora(buf, frequency_2);
870           if(i > 0) {
871             int n = parseLoRa_Msg(i,buf,APP_ID, DEV_ID,CH,payload, &rssi,seq_id, UP);
872             if(n == -1) { //Se for -1, ocorreu erro de MAC ou SEQID
873                 LoRa.idle();
874                 //LoRa.sleep();
875                 delay(10);
876             return;
877           }
878             getMsg_SeqID(buf, &seq_id, UP);
879             if(FSM == 4) { //Recebeu DATA
880               tam = 24 + payload.length();
881
882               #ifdef CANAL_SEMI_RAND
883                 frequency_1 = set_CHANNEL(CH_SEMI_RAND, DOWN);
884               #elif defined(CANAL_FIXO)
885                 frequency_1 = set_CHANNEL(CH_FIXO, DOWN);
886               #else
887                 frequency_1 = set_CHANNEL(CH, DOWN);
```

```
            #endif

            Serial.print("LoRa-Message: [LOG] --> APP ID: ");
            for(int j=0; j<5; j++) {
              Serial.print(APP_ID[j], HEX);
              Serial.print(" ");
            }
            Serial.println();
            Serial.print("LoRa-Message: [LOG] --> DEVICE: ");
            for(int j=0; j<5; j++) {
              Serial.print(DEV_ID[j], HEX);
              Serial.print(" ");
            }
            Serial.println();
            Serial.print("LoRa-Message: [LOG] --> MAC: ");
            for(int j=10; j<18; j++) {
              Serial.print(buf[j], HEX);
              Serial.print(" ");
            }
            Serial.println();
            Serial.print("LoRa-Message: [LOG] --> SEQUENCE: ");
            Serial.println(seq_id , HEX);

            Serial.println("LoRa-Message: [LOG] --> TYPE: 4");
            Serial.print("LoRa-Message: [LOG] --> CHANNEL: ");
            #ifdef CANAL_SEMI_RAND
              Serial.print(CH_SEMI_RAND[0], HEX);
            #elif defined(CANAL_FIXO)
              Serial.print(CH_FIXO[0], HEX);
            #else
              Serial.print(CH[0], HEX);
            #endif
            Serial.print(" => ");
            Serial.println(frequency_1);

            Serial.print("LoRa-Message: [LOG] --> PAYLOAD: ");
            for(int j=0; j<tam-24; j++) {
              Serial.print(payload.charAt(j), HEX);
              Serial.print(" ");
            }
            Serial.println();

            montar_resposta(response, &rssi, ACK, DOWN);

            Serial.print("LoRa-Response: [DEBUG] --> RSSI: ");
            Serial.println(rssi);

            getMsg_CHANNEL(response, CH, DOWN);

            montartoSend(buf, toSend, payload, (uint8_t)_rssi);
            int m = send_data(toSend, tam, bufwifi);
            //int n = check_crc(bufwifi);
            //if(n == 0) {
            // Serial.println("[DEBUG] Erro CRC16");
            // LoRa.idle();
              //LoRa.sleep();
            // delay(10);
            // return;
            //}

            //if(servidor == false) {
              //for(int j=12; j < 26; j++) {
              // response[j] = bufwifi[j-12];
              //}
            //}
            /*else {
              addTime(response);
              for(int j=16; j < 26; j++) {
                response[j] = bufwifi[j-16];
              }
            }*/

        response[12] = bufwifi[0];
        tam = 13;
            //tam = 26;
            gen_MAC_ID(response, DOWN);
            Serial.print("LoRa-Response: [LOG] --> ACK - RESPONSE: ");
            for(int j=0; j<tam; j++) {
              Serial.print(response[j], HEX);
              Serial.print(" ");
            }
```

```
969                 Serial.println();
970
971                 #ifdef CANAL_SEMI_RAND
972                     SEND_LoRa(response, tam, CH_SEMI_RAND, frequency_1, &frequency_2, DOWN);
973                 #elif defined(CANAL_FIXO)
974                     SEND_LoRa(response, tam, CH_FIXO, frequency_1, &frequency_2, DOWN);
975                 #else
976                     SEND_LoRa(response, tam, CH, frequency_1, &frequency_2, DOWN);
977                 #endif
978
979               }
980           }
981           else {
982             Serial.println("[DEBUG] Timeout DATA");
983             Serial.println();
984           }
985         }
986         if(FSM == 7) { //Recebeu um Priority
987           tam = 24 + payload.length();
988
989             /*#ifdef CANAL_SEMI_RAND
990                 frequency_1 = set_CHANNEL(CH_FIXO, DOWN);
991             #elif defined(CANAL_FIXO)
992                 frequency_1 = set_CHANNEL(CH_INICIAL, DOWN);
993             #else
994                 frequency_1 = set_CHANNEL(CH, DOWN);
995             #endif*/
996             #ifdef CANAL_FIXO
997                 frequency_1 = set_CHANNEL(CH_FIXO, DOWN);
998             #else
999                 frequency_1 = set_CHANNEL(CH, DOWN);
1000            #endif
1001
1002            Serial.print("LoRa-Message: [LOG] --> APP ID: ");
1003            for(int j=0; j<5; j++) {
1004                Serial.print(APP_ID[j], HEX);
1005                Serial.print(" ");
1006            }
1007            Serial.println();
1008            Serial.print("LoRa-Message: [LOG] --> DEVICE: ");
1009            for(int j=0; j<5; j++) {
1010                Serial.print(DEV_ID[j], HEX);
1011                Serial.print(" ");
1012            }
1013            Serial.println();
1014            Serial.print("LoRa-Message: [LOG] --> MAC: ");
1015            for(int j=10; j<18; j++) {
1016                Serial.print(buf[j], HEX);
1017                Serial.print(" ");
1018            }
1019            Serial.println();
1020            Serial.print("LoRa-Message: [LOG] --> SEQUENCE: ");
1021            Serial.println(seq_id , HEX);
1022
1023            Serial.println("LoRa-Message: [LOG] --> TYPE: 7");
1024            Serial.print("LoRa-Message: [LOG] --> CHANNEL: ");
1025
1026            #ifdef CANAL_FIXO
1027                Serial.print(CH_FIXO[0], HEX);
1028            #else
1029                Serial.print(CH[0], HEX);
1030            #endif
1031            Serial.print(" => ");
1032            Serial.println(frequency_1);
1033
1034            Serial.print("LoRa-Message: [LOG] --> PAYLOAD: ");
1035            for(int j=0; j<tam-24; j++) {
1036                Serial.print(payload.charAt(j), HEX);
1037                Serial.print(" ");
1038            }
1039            Serial.println();
1040
1041            montar_resposta(response, &rssi, ACK, DOWN);
1042
1043            Serial.print("LoRa-Response: [DEBUG] --> RSSI: ");
1044            Serial.println(rssi);
1045
1046            _rssi = rssi;
1047
1048
1049            int l;
```

```
1050          l = ping_server();
1051          if(l == -1) {
1052            //Erro de ping -> Enviar DOS e sair da funcao
1053            montar_resposta(response, &rssi, DOS, DOWN);
1054            gen_MAC_ID(response, DOWN);
1055            getMsg_CHANNEL(response, CH, DOWN);
1056            Serial.print("LoRa-Response: [LOG] --> RESPONSE: ");
1057            for(int j=0; j<12; j++) {
1058              Serial.print(response[j], HEX);
1059              Serial.print(" ");
1060            }
1061            Serial.println();
1062            #ifdef CANAL_FIXO
1063              waitCTS();
1064              SEND_LoRa(response, 12, CH_FIXO, frequency_1, &frequency_2, DOWN);
1065            #else
1066              waitCTS();
1067              SEND_LoRa(response, 12, CH, frequency_1, &frequency_2, DOWN);
1068            #endif
1069
1070            checar_wifi();
1071
1072            LoRa.idle();
1073            //LoRa.sleep();
1074            delay(10);
1075            return;
1076          }
1077
1078            getMsg_CHANNEL(response, CH, DOWN);
1079
1080            montartoSend(buf, toSend, payload, (uint8_t)_rssi);
1081            int m = send_data(toSend, tam, bufwifi);
1082            int n = check_crc(bufwifi);
1083            if(n == 0) {
1084                Serial.println("[DEBUG] Erro CRC16");
1085                LoRa.idle();
1086                //LoRa.sleep();
1087                delay(10);
1088                return;
1089            }
1090
1091            //if(servidor == false) {
1092                for(int j=12; j < 26; j++) {
1093                  response[j] = bufwifi[j-12];
1094                }
1095            //}
1096            /*else {
1097                addTime(response);
1098              for(int j=16; j < 26; j++) {
1099                  response[j] = bufwifi[j-16];
1100              }
1101            }*/
1102
1103            tam = 26;
1104            gen_MAC_ID(response, DOWN);
1105            Serial.print("LoRa-Response: [LOG] --> ACK - RESPONSE: ");
1106            for(int j=0; j<tam; j++) {
1107                Serial.print(response[j], HEX);
1108                Serial.print(" ");
1109            }
1110            Serial.println();
1111
1112            #ifdef CANAL_FIXO
1113                SEND_LoRa(response, tam, CH_FIXO, frequency_1, &frequency_2, DOWN);
1114            #else
1115                SEND_LoRa(response, tam, CH, frequency_1, &frequency_2, DOWN);
1116            #endif
1117
1118        }
1119        if(FSM == 15) { //Recebeu um Test
1120          #ifdef CANAL_FIXO
1121              frequency_1 = set_CHANNEL(CH_FIXO, DOWN);
1122          #else
1123              frequency_1 = set_CHANNEL(CH, DOWN);
1124          #endif
1125
1126          montar_resposta(response, &rssi, ACK, DOWN);
1127          gen_MAC_ID(response, DOWN);
1128
1129          Serial.print("LoRa-Message: [LOG] --> APP ID: ");
1130          for(int j=0; j<5; j++) {
```

```
1131            Serial.print(APP_ID[j], HEX);
1132            Serial.print(" ");
1133          }
1134          Serial.println();
1135          Serial.print("LoRa-Message: [LOG] --> DEVICE: ");
1136          for(int j=0; j<5; j++) {
1137            Serial.print(DEV_ID[j], HEX);
1138            Serial.print(" ");
1139          }
1140          Serial.println();
1141          Serial.print("LoRa-Message: [LOG] --> MAC: ");
1142          for(int j=10; j<18; j++) {
1143            Serial.print(buf[j], HEX);
1144            Serial.print(" ");
1145          }
1146          Serial.println();
1147          Serial.print("LoRa-Message: [LOG] --> SEQUENCE: ");
1148          Serial.println(seq_id, HEX);
1149
1150          Serial.println("LoRa-Message: [LOG] --> TYPE: 0F");
1151          Serial.print("LoRa-Message: [LOG] --> CHANNEL: ");
1152          #ifdef CANAL_FIXO
1153            Serial.print(CH_FIXO[0], HEX);
1154          #else
1155            Serial.print(CH[0], HEX);
1156          #endif
1157          Serial.print(" => ");
1158          Serial.println(frequency_1);
1159
1160          Serial.print("LoRa-Response: [DEBUG] --> RSSI: ");
1161          Serial.println(rssi);
1162
1163          _rssi = rssi;
1164
1165          int l;
1166          l = ping_server();
1167          if(l == -1) {
1168            //Erro de ping -> Enviar DOS e sair da funcao
1169            montar_resposta(response, &rssi, DOS, DOWN);
1170            gen_MAC_ID(response, DOWN);
1171            getMsg_CHANNEL(response, CH, DOWN);
1172            Serial.print("LoRa-Response: [LOG] --> RESPONSE: ");
1173            for(int j=0; j<12; j++) {
1174              Serial.print(response[j], HEX);
1175              Serial.print(" ");
1176            }
1177            Serial.println();
1178            #ifdef CANAL_FIXO
1179              waitCTS();
1180              SEND_LoRa(response, 12, CH_FIXO, frequency_1, &frequency_2, DOWN);
1181            #else
1182              waitCTS();
1183              SEND_LoRa(response, 12, CH, frequency_1, &frequency_2, DOWN);
1184            #endif
1185
1186            checar_wifi();
1187
1188            LoRa.idle();
1189            //LoRa.sleep();
1190            delay(10);
1191            return;
1192          }
1193
1194            getMsg_CHANNEL(response, CH, DOWN);
1195            Serial.print("LoRa-Response: [LOG] --> RESPONSE: ");
1196            for(int j=0; j<12; j++) {
1197              Serial.print(response[j], HEX);
1198              Serial.print(" ");
1199            }
1200            Serial.println();
1201            //addTime(response);
1202            response[12] = 0x00;
1203            response[13] = 0x00;
1204            response[14] = 0x00;
1205            response[15] = 0x00;
1206            #ifdef CANAL_FIXO
1207              waitCTS();
1208              SEND_LoRa(response, 16, CH_FIXO, frequency_1, &frequency_2, DOWN);
1209            #else
1210              waitCTS();
1211              SEND_LoRa(response, 16, CH, frequency_1, &frequency_2, DOWN);
```

```
            #endif
        }

    }
    LoRa.idle();
    //LoRa.sleep();
    delay(10);

}


void start_config_server(const char * ssid, const char * password, String configPage, String Domain){
    start_AP(ssid, password);
    unsigned int i = 0;
// start_DNS(Domain);

    server.begin();

    while(true){
        //dnsserver.processNextRequest();
        WiFiClient client = server.available(); // listen for incoming clients
        while (client.connected()) { // if you get a client,
            i = 0;
            while (client.available() < 1);
            while (client.available()) {
                String currentLine = read_request(client);
                String protocol = get_protocol(currentLine);
                String request_method = get_method(currentLine);
                if (protocol != "HTTP/1.1") {
                    request_method = "";
                }
                if (request_method == "GET") {
                    resolve_get(currentLine, client, configPage);
                } else if (request_method == "POST") {
                    resolve_post(currentLine, client);
                } else if (request_method == "DELETE") {
                    resolve_delete(currentLine, client);
                } else if (request_method == "") {
                    Serial.println("Received some invalid request"); // DO NOTHING :)
                    Serial.println("THIS IS NOT AN ERROR");
                    Serial.println();
                    Serial.println();
                    Serial.println();
                    client.println(Error400());
                } else {
                    Serial.print("Method : "); // TODO MAKE 404 ERROR RESPONSE
                    Serial.println(request_method);
                    Serial.println("Not implemented");
                    Serial.println("THIS IS NOT AN ERROR");
                    Serial.println();
                    Serial.println();
                    Serial.println();
                    client.print(Error400());
                }
                currentLine = "";
                client.stop();
            }
        }
        delay(10);
        if (i > RESET_DELAY){
            ESP.restart();
            while(true) yield();
        }
        i++;
    }
}

String read_request(WiFiClient client){
    String currentLine = "";
    while (client.available() != 0) {
        char c = client.read();
        currentLine += c;
        delay(1);
    }
    Serial.println(currentLine);
    return currentLine;
}

void resolve_post(String currentLine, WiFiClient client){
    String path = get_path(currentLine);
    Serial.println(path);
```

```
    if (path == "/setup"){
      post_setup(currentLine, client);
    } else if (path == "/reset") {
      post_reset(client);
    } else if (path == "/test") {
      post_test(currentLine, test_post, client);
    } else {
      Serial.print("Path : "); // TODO MAKE 404 ERROR RESPONSE
      Serial.println(path);
      Serial.println("Not implemented for POST method");
      Serial.println("THIS IS NOT AN ERROR");
      client.print(Error404());
    }
  }

  void resolve_get(String currentLine, WiFiClient client, String configPage){
    String path = get_path(currentLine);
    if (path == "/") {
      get_root(client, configPage);
      Serial.println("Returned configuration page.");
    } else if (path == "/hotspot-detect.html") {
      get_root(client, configPage);
    } else if (path == "/gen_204") {
      get_root(client, configPage);
    } else if (path == "/reset") {
      get_reset(client);
    } else if (path == "/test") {
      get_test(client, test_get);
    } else if (path == "/list") {
      get_list(client);
    } else {
      Serial.print("Path : "); // TODO MAKE 404 ERROR RESPONSE
      Serial.println(path);
      Serial.println("Not implemented for GET method");
      Serial.println("THIS IS NOT AN ERROR");
      client.print(Error404());
    }
  }

  void resolve_delete(String currentLine, WiFiClient client){
    String path = get_path(currentLine);
    if (path == "/setup"){
      delete_config();
    } else {
      Serial.print("Path : "); // TODO MAKE 404 ERROR RESPONSE
      Serial.println(path);
      Serial.println("Not implemented for DELETE method");
      Serial.println("THIS IS NOT AN ERROR");
      client.print(Error404());
    }
  }

  void post_setup(String currentLine, WiFiClient client){
    Handle(currentLine.substring(currentLine.lastIndexOf('{')));
    client.print(respond("Configurated"));
  }

  void post_reset(WiFiClient client){
    get_reset(client);
  }

  void post_test(String currentLine, void (*test)(String), WiFiClient client){
    if (test != NULL){
      client.print(respond("Testing Action"));
      test(currentLine);
    } else {
      client.print(Error404());
    }
  }

  void get_root(WiFiClient client, String configPage){
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println();
    client.print(configPage);
    client.println();
  }

  void get_reset(WiFiClient client){
    client.println(respond("Reseting System"));
    delay(100);
```

```
    client.stop();
    delay(30000);
    ESP.restart();
    while(true) yield(); // this should force a HW reset.
  }

  void get_list(WiFiClient client){
    String response = "[";
    int number = WiFi.scanNetworks(); // number is for number of networks found
    for (int i = 0; i < number; i++) {
      response += "{\n\"SSID\": \"";
      response += WiFi.SSID(i);
      response += "\",\n\"Encryption\": \"";
      response += encryption(WiFi.encryptionType(i));
      response += "\",\n\"RSSI\": ";
      response += String(WiFi.RSSI(i));
      response += "\n}";
      if (i != (number - 1)){
        response += ",";
      }
    }
    response += "\n]";
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:application/json");
    client.println();
    client.print(response);
    client.println();
  }

  void get_test(WiFiClient client, String (*test)()){
    if (test != NULL)
    {
      client.println("HTTP/1.1 200 OK");
      client.println("Content-type:application/json");
      client.println();
      client.print(test());
      client.println();
    } else {
      client.print(Error404());
    }
  }

  String encryption(byte encryption){
    switch(encryption){
      case ENC_TYPE_WEP:
        return "WEP";
      case ENC_TYPE_TKIP:
        return "WPA";
      case ENC_TYPE_CCMP:
        return "WPA2";
      case ENC_TYPE_NONE:
        return "OPEN";
      case ENC_TYPE_AUTO:
        return "WPA/WPA2";
      default:
        return "?";
    }
  }

  String get_method(String request){
    String method = request.substring(0, request.indexOf(" "));
    method.trim();
    return method;
  }

  String get_path(String request){
    String path = request.substring(request.indexOf(" ") + 1, request.indexOf(" ", request.indexOf(" ") + 1));
    path.trim();
    return path;
  }

  String get_protocol(String request){
    request.trim();
    String protocol = request.substring(request.indexOf(" ", request.indexOf(" ") + 1), request.indexOf('\r'));
    protocol.trim();
    return protocol;
  }

  void start_AP(const char * ssid, const char * password){
    Serial.println();
    Serial.println("Configurando ponto de acesso...");
```

```
    Serial.print("SSID: ");
    Serial.println(ssid);
    delay(50);
    Serial.print("PASS: ");
    Serial.println(password);
    WiFi.softAP(ssid, password);
    IPAddress myIP = WiFi.softAPIP();
    Serial.print("Endereço IP: ");
    Serial.println(myIP);
  }

  void Handle(String Json){
    String values[10];
    int size = parse(Json, values);
    save_config((values[0] + "\n" + values[1]).c_str());
    if( (values[2].length() > 7) && (values[3].length() > 12) && (values[4].length() > 11) ) { // IP FIXO
    save_ipfixo((values[2]), (values[3]), (values[4]));
    }
  }


  String respond(String message){
    String str;
    str += "HTTP/1.1 200 OK\r\n";
    str += "Content-type:application/json\r\n";
    str += "\r\n";
    str += "{\n\"code\": 200,\n\"message\": \"" + message + "\"\n}\r\n\r\n";
    return str;
  }

  String Error400(){
    String Bad_Request = "";
    Bad_Request += "HTTP/1.1 400 Bad Request\r\n";
    Bad_Request += "Content-type:application/json\r\n";
    Bad_Request += "\r\n";
    Bad_Request += "{\n\t\"code\": 400,\n\t\"message\": \"Your Request is Invalid\"\n}\n";
  }

  String Error404(){
    String Not_Found = "";
    Not_Found += "HTTP/1.1 404 Not Found\r\n";
    Not_Found += "Content-type:application/json";
    Not_Found += "\r\n";
    Not_Found += "{\n\t\"code\": 404,\n\t\"message\": \"Path Not Found\"\n}\n";
    return Not_Found;
  }

  int parse(String Json, String * values){
    int start = 0;
    int i = 0;
    Serial.println();
    Serial.print("Json Received: ");
    Serial.println(Json);
    Json.replace("\r","");
    Json.replace("\n","");
    Json.replace("\t","");
    Json.replace("{","");
    Json.replace("}","");
    Json.replace("]","");
    Json.trim();
    while(Json.indexOf(",") != -1){
      values[i] = Json.substring(0, Json.indexOf(","));
      Json.remove(0, Json.indexOf(",") + 1);
      i++;
    }
    values[i] = Json.substring(0, Json.indexOf(","));
    i++;
    return i;
  }

  String read_flash(){
    for(int i = 0; EEPROM.read(i) != 0 && i < 64; i++){
      AP_name += char(byte(EEPROM.read(i)));
    }
    AP_name.replace("\"","");
    AP_pass = AP_name.substring(AP_name.indexOf("\n") + 1);
    AP_name.remove(AP_name.indexOf("\n"));

    if(EEPROM.read(1012) == 1) {
      ip_fixo = true;
      _ip1 = EEPROM.read(1000);
```

```
    _ip2 = EEPROM.read(1001);
    _ip3 = EEPROM.read(1002);
    _ip4 = EEPROM.read(1003);
    _gate1 = EEPROM.read(1004);
    _gate2 = EEPROM.read(1005);
    _gate3 = EEPROM.read(1006);
    _gate4 = EEPROM.read(1007);
    _sub1 = EEPROM.read(1008);
    _sub2 = EEPROM.read(1009);
    _sub3 = EEPROM.read(1010);
    _sub4 = EEPROM.read(1011);
    }


    return (AP_name + "\n" + AP_pass);
  }

  void save_config(const char * str){
    Serial.println();
    Serial.print("Saving");
    Serial.print(str);
    Serial.println(" as configuration");
    for(int addr = 0; addr <= EEPROM_SIZE; addr++){
      int val = byte(str[addr]);

      EEPROM.write(addr, val);

      if (addr == EEPROM_SIZE){
       EEPROM.write(1012, 0);
       EEPROM.commit();
      }
    }
  }

  void save_ipfixo(String str1, String str2, String str3) {
    char ip1, ip2, ip3, ip4, gate1, gate2, gate3, gate4, sub1, sub2, sub3, sub4, aux[20];

    Serial.println();
      Serial.print(F("Salvando "));
      str1.replace('"',' ');
      str1.trim();
      Serial.println(str1.substring(str1.indexOf(":") + 1));
      str1.substring(str1.indexOf(":") + 1).toCharArray(aux, 20);
      parsing(aux, &ip1, &ip2, &ip3, &ip4);
      Serial.println(ip1, DEC);
      Serial.println(ip2, DEC);
      Serial.println(ip3, DEC);
      Serial.println(ip4, DEC);
      Serial.println(F(" para configuracao de ip fixo"));

      Serial.println();
      Serial.print(F("Salvando "));
      str2.replace('"',' ');
      str2.trim();
      Serial.println(str2.substring(str2.indexOf(":") + 1));
      str2.substring(str2.indexOf(":") + 1).toCharArray(aux, 20);
      parsing(aux, &gate1, &gate2, &gate3, &gate4);
      Serial.println(gate1, DEC);
      Serial.println(gate2, DEC);
      Serial.println(gate3, DEC);
      Serial.println(gate4, DEC);
      Serial.println(F(" para configuracao de ip fixo"));

      Serial.println();
      Serial.print(F("Salvando "));
      str3.replace('"',' ');
      str3.trim();
      Serial.println(str3.substring(str3.indexOf(":") + 1));
      str3.substring(str3.indexOf(":") + 1).toCharArray(aux, 20);
      parsing(aux, &sub1, &sub2, &sub3, &sub4);
      Serial.println(sub1, DEC);
      Serial.println(sub2, DEC);
      Serial.println(sub3, DEC);
      Serial.println(sub4, DEC);
      Serial.println(F(" para configuracao de ip fixo"));

      EEPROM.write(1000, ip1);
      EEPROM.write(1001, ip2);
      EEPROM.write(1002, ip3);
      EEPROM.write(1003, ip4);
      EEPROM.write(1004, gate1);
```

```
1617        EEPROM.write(1005, gate2);
1618        EEPROM.write(1006, gate3);
1619        EEPROM.write(1007, gate4);
1620        EEPROM.write(1008, sub1);
1621        EEPROM.write(1009, sub2);
1622        EEPROM.write(1010, sub3);
1623        EEPROM.write(1011, sub4);
1624        EEPROM.write(1012, 1);
1625
1626        EEPROM.commit();
1627
1628    }
1629
1630    void parsing(char * str, char * arg1, char * arg2, char * arg3, char * arg4){
1631      int j=1,k=0, l;
1632      char aux[3];
1633
1634      for(int i=0; i < strlen(str); i++){
1635        if(str[i] == 46){
1636          if(k < 3){
1637            aux[k] = '\0';
1638          }
1639          if(j == 1) {
1640            sscanf(aux, "%d", &l);
1641            *arg1 = l;
1642          }
1643          else {
1644            if(j == 2){
1645              sscanf(aux, "%d", &l);
1646              *arg2 = l;
1647            }
1648            else{
1649              if(j == 3) {
1650                sscanf(aux, "%d", &l);
1651                *arg3 = l;
1652              }
1653            }
1654          }
1655          j++;
1656          k=0;
1657        }
1658        else{
1659          aux[k] = str[i];
1660          k++;
1661        }
1662      }
1663
1664      if(k < 3){
1665        aux[k] = '\0';
1666      }
1667      sscanf(aux, "%d", &l);
1668      *arg4 = l;
1669
1670    }
1671
1672    void delete_config(){
1673      Serial.println("Deleting config");
1674      for (int i = 0; i < EEPROM_SIZE; i++) {
1675        EEPROM.write(i, 0);
1676      }
1677      EEPROM.commit();
1678    }
1679
1680    // DNS
1681    void start_DNS(String Domain) {
1682      dnsserver.start(DNS_PORT, Domain.c_str(), APIP);
1683    }
1684
1685
1686    //função que o temporizador irá chamar, para reiniciar o ESP32
1687    void IRAM_ATTR resetModule(){
1688        ets_printf("[Watchdog] Reiniciar\n"); //imprime no log
1689        ESP.restart();
1690        while(true) yield(); //reinicia o chip
1691    }
1692
1693    void configureWatchdog() {
1694
1695        timer = timerBegin(0, 80, true); //timerID 0, div 80
1696        //timer, callback, interrupção de borda
1697        timerAttachInterrupt(timer, &resetModule, true);
```

```
1698        //timer, tempo (us), repetição
1699        timerAlarmWrite(timer, 15000000, true);
1700        timerAlarmEnable(timer); //habilita a interrupção //enable interrupt
1701    }
1702
1703    #else
1704
1705      void start_LoRa(){
1706          Serial.print(F("Hardware Check..."));
1707          if (!initLoRa()){
1708              Serial.println(F("Failed"));
1709              while (1);
1710          }
1711          else {
1712              Serial.println(F("OK"));
1713          }
1714
1715      }
1716
1717      int initLoRa(){
1718
1719          SPI.begin();
1720          if (!LoRa.begin(916.8E6)) {
1721            return -1;
1722          }
1723          LoRa.setSpreadingFactor(10);
1724          LoRa.setTxPower(17, PA_OUTPUT_PA_BOOST_PIN); //Max Power 17; Min Power 5
1725          LoRa.setPreambleLength(6);
1726          LoRa.setSignalBandwidth(125E3);
1727          LoRa.setCodingRate4(4);
1728          LoRa.setSyncWord(0xAA);
1729          LoRa.beginPacket(true);
1730          LoRa.enableCrc();
1731
1732          return 1;
1733      }
1734
1735
1736        void configuraRTC (int tempo) {
1737
1738          tmElements_t tmSet;
1739
1740          if (tmp.length() > 0) {
1741            int _year = year(hexToDec(tmp));
1742            int _month = month(hexToDec(tmp));
1743            int _day = day(hexToDec(tmp));
1744            int _hour = hour(hexToDec(tmp));
1745            if (_hour < 0) _hour = (24 + _hour); //Ajuste para o caso de (zero horas);
1746            int _minute = minute(hexToDec(tmp));
1747            int _second = second(hexToDec(tmp));
1748
1749            tmSet.Year = _hour;
1750            tmSet.Month = _month;
1751            tmSet.Day = _day;
1752            tmSet.Hour = _hour;
1753            tmSet.Minute = _minute;
1754            tmSet.Second = _second;
1755
1756            RTC.set(makeTime(tmSet), CLOCK_ADDRESS); // set the clock
1757            tmp = "";
1758
1759          if(at == false) {
1760                // Print do tempo do alarme
1761                Serial.print(F("[DEBUG] Network clock: "));
1762                Serial.print( tmSet.Hour );
1763                Serial.print (F(":"));
1764                Serial.print( tmSet.Minute );
1765                Serial.print (F(":"));
1766                Serial.println ( tmSet.Second );
1767                Serial.println (F(""));
1768            }
1769          }
1770
1771          //delay(2000);
1772
1773          time_t clock = RTC.get(CLOCK_ADDRESS);
1774          breakTime(clock, tmSet);
1775
1776          if(at == false) {
1777             // Print do tempo do alarme
1778             //Serial.println("***Hora atual");
```

```
1779            Serial.print(F("[DEBUG] Clock: "));
1780            Serial.print( tmSet.Hour );
1781            Serial.print (F(":"));
1782            Serial.print( tmSet.Minute );
1783            Serial.print (F(":"));
1784            Serial.println ( tmSet.Second );
1785          }
1786          if(alarm == true) {
1787            //Setup do alarme
1788            if ( tempo < 2 ) tmSet.Second += 5;
1789            else (tmSet.Minute += tempo);
1790
1791            RTC.set(makeTime(tmSet), ALARM1_ADDRESS); // set the alarm for 4 seconds later
1792            RTC.enableAlarm(ALARM1_ADDRESS);
1793            RTC.freqSelect(1); // set the squarewave freq on alarm pin b to 4.096kHz
1794
1795            time_t alarm = RTC.get(ALARM1_ADDRESS); // get the time the alarm is set for
1796            breakTime(alarm, tmSet);
1797
1798            if(at == false) {
1799               // Print do tempo do alarme
1800               Serial.print(F("[DEBUG] Alarm: "));
1801               Serial.print( tmSet.Hour );
1802               Serial.print (F(":"));
1803               Serial.print( tmSet.Minute );
1804               Serial.print (F(":"));
1805               Serial.println ( tmSet.Second );
1806               Serial.println();
1807               delay(50);
1808            }
1809          }
1810        }
1811
1812
1813        void sendMessage(int FSM_SEND) {
1814        unsigned char buf[25] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
1815        unsigned char response[37] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
1816        int i=0, len, rssi=0, _rssi=0, envios=0;
1817        char APP_ID[5]="", DEV_ID[5]="", CH[2]="";
1818        String PAYLOAD="";
1819        long frequency_1=0, frequency_2=0;
1820        unsigned long seq_id=0, comeco=0, fim=0;
1821        bool envio_dados = false, envio_test = false, envio_priority = false;
1822        static int x = 1;
1823        #ifdef CANAL_SEMI_RAND
1824          char CH_SEMI_RAND[2]="";
1825        #endif
1826        char CH_FIXO[2]="";
1827
1828          while(((envios < 3) && (envio_dados == false) && (FSM_SEND == SEND_DATA)) || ((envio_test == false) && (envios < 3) && (
                ↪ FSM_SEND == SEND_TEST)) || ((envio_priority == false) && (envios < 3) && (FSM_SEND == SEND_PRIORITY))){
1829            wdt_reset();
1830            envios++;
1831          if(FSM_SEND == SEND_DATA) {
1832            tam_recv = 0;
1833            montar_resposta(response, &rssi, RTS, UP);
1834            getMsg_CHANNEL(response, CH, UP);
1835
1836            if(at == false) {
1837              Serial.println(F("[DEBUG] Sending..."));
1838            }
1839            if(rch == 0) {
1840              frequency_1 = CHANNEL_5; //CANAL INICIAL DEFINIDO AQUI
1841            }
1842            else {
1843              CH_FIXO[0] = random(8) + 1;
1844              frequency_1 = set_CHANNEL(CH_FIXO, UP);
1845            }
1846
1847            #ifdef CANAL_FIXO
1848              get_CHANNEL(CH_FIXO, (int)frequency_1, UP);
1849            #endif
1850            LoRa.setSpreadingFactor(10);
1851            LoRa.setTxPower(17, PA_OUTPUT_PA_BOOST_PIN);
1852            //LoRa.set20dBm_sx127x(true); //MODIFICADO
1853            comeco = millis();
1854            #ifdef CANAL_FIXO
1855              SEND_LoRa(response, 21, CH_FIXO, frequency_1, &frequency_2, UP);
1856            #else
1857              SEND_LoRa(response, 21, CH, frequency_1, &frequency_2, UP);
1858            #endif
```

```
1859
1860
1861        i = rcvLora(buf, frequency_2);
1862        if(i > 0) {
1863
1864          getMsg_SeqID(buf, &seq_id, DOWN);
1865          seq_id--;
1866          int n = parseLoRa_Msg(i,buf,APP_ID, DEV_ID,CH, PAYLOAD,&rssi, seq_id, DOWN);
1867          if(n == -1) { //Se for -1, ocorreu erro de MAC ou SEQID
1868            return;
1869          }
1870          _rssi = rssi;
1871          sinal = _rssi;
1872          getMsg_SeqID(buf, &seq_id, DOWN);
1873          if(FSM == 2){ //Recebeu um CTS, deve enviar DATA se todas as verificações estiverem corretas
1874            if(at == false){
1875              Serial.print(F("[DEBUG] Received: "));
1876              for(int j=0; j<i; j++) {
1877                Serial.print(buf[j], HEX);
1878                Serial.print(F(" "));
1879              }
1880              Serial.println(F("--> CTS"));
1881            }
1882            ////////////////////////////////////
1883            #ifdef CANAL_SEMI_RAND
1884            for(int j=0; j<2; j++){
1885                CH_SEMI_RAND[j] = CH[j];
1886                //Serial.println(CH_FIXO[0], HEX);
1887              }
1888            #endif
1889            ////////////////////////////////////
1890            #ifdef CANAL_SEMI_RAND
1891              frequency_1 = set_CHANNEL(CH_SEMI_RAND, UP);
1892            #elif defined(CANAL_FIXO)
1893              frequency_1 = set_CHANNEL(CH_FIXO, UP);
1894            #else
1895              frequency_1 = set_CHANNEL(CH, UP);
1896            #endif
1897
1898            montar_resposta(response, &rssi, DATA, UP);
1899            getMsg_CHANNEL(response, CH, UP);
1900            if(payload.length() % 2 == 0) {
1901              len = payload.length()/2;
1902            }
1903            else {
1904              len = payload.length()/2 + 1;
1905            }
1906            if((payload.length() > 32) || criptografia == true) {
1907              len = 16;
1908            }
1909            len += 21;
1910
1911            //Serial.print(F("RSSI: ")); Serial.println(rssi);
1912            if(rssi < 96) {
1913                //Serial.println(F("[DEBUG] Potencia setada para 8."));
1914                LoRa.setTxPower(8, PA_OUTPUT_PA_BOOST_PIN);
1915            }
1916            else {
1917              if(rssi < 104) {
1918                //Serial.println(F("[DEBUG] Potencia setada para 14."));
1919                LoRa.setTxPower(14, PA_OUTPUT_PA_BOOST_PIN);
1920              }
1921              else {
1922                if(rssi > 103) {
1923                  //Serial.println(F("[DEBUG] Potencia setada para 17."));
1924                  LoRa.setTxPower(17, PA_OUTPUT_PA_BOOST_PIN);
1925                }
1926              }
1927            }
1928
1929            if(rssi < 110) { //O valor de rssi éabsoluto, então <110 -> rapido
1930              LoRa.setSpreadingFactor(7);
1931            }
1932            #ifdef CANAL_SEMI_RAND
1933              SEND_LoRa(response, len, CH_SEMI_RAND, frequency_1, &frequency_2, UP);
1934            #elif defined(CANAL_FIXO)
1935              SEND_LoRa(response, len, CH_FIXO, frequency_1, &frequency_2, UP);
1936            #else
1937              SEND_LoRa(response, len, CH, frequency_1, &frequency_2, UP);
1938            #endif
1939
```

```
1940            i = rcvLora(buf, frequency_2);
1941            if(i > 0) {
1942                fim = millis();
1943                int n = parseLoRa_Msg(i,buf,APP_ID, DEV_ID,CH, PAYLOAD,&rssi, seq_id, DOWN);
1944                if(n == -1) { //Se for -1, ocorreu erro de MAC ou SEQID
1945              return;
1946            }
1947                getMsg_SeqID(buf, &seq_id, DOWN);
1948                if(FSM == 3) { //Recebeu ACK
1949                  envio_dados = true;
1950                  //tam_recv = 0;
1951                  for(int j=16; j < 26; j++) {
1952                    received[(j-16)*2] = (buf[j] & 0xF0) >> 4;
1953                    received[(j-16)*2 + 1] = buf[j] & 0x0F;
1954                    tam_recv = tam_recv + 2;
1955                  }
1956                  if(at == false) {
1957                    Serial.print(F("[DEBUG] Received: "));
1958                    for(int j=0; j<i; j++) {
1959                      Serial.print(buf[j], HEX);
1960                      Serial.print(F(" "));
1961                    }
1962
1963                    Serial.println(F("--> ACK"));
1964                    Serial.println(F("[DEBUG] Sent data!!!"));
1965                    Serial.print(F("[DEBUG] RSSI(dbm): "));
1966                    Serial.println(_rssi);
1967                    Serial.print(F("[DEBUG] Delay(ms): "));
1968                    Serial.println(fim-comeco);
1969                    Serial.println();
1970                }
1971
1972                  for (int i = 12; i < 16; i++) {
1973                    if ( String((unsigned char)buf[i], HEX).length() >= 2 ) {
1974                      tmp += String((unsigned char)buf[i], HEX);
1975                    }
1976                    else {
1977                      tmp += ajustConversion(String((unsigned char)buf[i], HEX));
1978                    }
1979                  }
1980
1981
1982
1983                  String _interval = "";
1984              _interval = String((unsigned char)buf[23], HEX);
1985              interval = hexToDec(_interval);
1986              if(interval > 238) {
1987                interval = 5;
1988              }
1989                  configuraRTC(interval);
1990
1991
1992
1993                }
1994            }
1995            else {
1996              if(at == false) {
1997                Serial.print(F("[DEBUG] Timeout ACK"));
1998                Serial.println();
1999                Serial.println();
2000            }
2001            }
2002          }
2003          else {
2004            if(FSM == 14) { // Recebeu um DOS
2005              if(at == false){
2006                Serial.print(F("[DEBUG] Received: "));
2007                for(int j=0; j<i; j++) {
2008                  Serial.print(buf[j], HEX);
2009                  Serial.print(F(" "));
2010                }
2011                Serial.println(F("--> DOS"));
2012            }
2013              //return;
2014            }
2015            else {
2016              if(at == false) {
2017                Serial.print(F("[DEBUG] Não recebeu CTS"));
2018                  Serial.println();
2019                  Serial.println();
2020                }
```

```
2021                }
2022
2023              }
2024            }
2025            else {
2026              if(at == false) {
2027                Serial.println(F("[DEBUG] Timeout CTS"));
2028                Serial.println();
2029                Serial.println();
2030              }
2031            }
2032            }
2033            if(FSM_SEND == SEND_PRIORITY) {
2034              priority = true;
2035              tam_recv = 0;
2036              montar_resposta(response, &rssi, PRIORITY, UP);
2037            getMsg_CHANNEL(response, CH, UP);
2038            /*#ifdef CANAL_SEMI_RAND
2039              for(int j=0; j<2; j++){
2040                CH_FIXO[j] = CH[j];
2041              }
2042            #endif*/
2043
2044            Serial.println(F("[DEBUG] Sending..."));
2045
2046            if(rch == 0) {
2047              frequency_1 = CHANNEL_5; //CANAL INICIAL DEFINIDO AQUI
2048            }
2049            else {
2050              CH_FIXO[0] = random(8) + 1;
2051              frequency_1 = set_CHANNEL(CH_FIXO, UP);
2052            }
2053
2054            #ifdef CANAL_FIXO
2055              get_CHANNEL(CH_FIXO, (int)frequency_1, UP);
2056            #endif
2057            LoRa.setSpreadingFactor(10);
2058            LoRa.setTxPower(17, PA_OUTPUT_PA_BOOST_PIN);
2059            comeco = millis();
2060            if(payload.length() % 2 == 0) {
2061              len = payload.length()/2;
2062            }
2063            else {
2064              len = payload.length()/2 + 1;
2065            }
2066            if(payload.length() > 32) {
2067              len = 16;
2068            }
2069            len += 21;
2070            #ifdef CANAL_FIXO
2071              SEND_LoRa(response, len, CH_FIXO, frequency_1, &frequency_2, UP);
2072            #else
2073              SEND_LoRa(response, len, CH, frequency_1, &frequency_2, UP);
2074            #endif
2075
2076
2077            i = rcvLora(buf, frequency_2);
2078            if(i > 0) {
2079              fim = millis();
2080              getMsg_SeqID(buf, &seq_id, DOWN);
2081              seq_id--;
2082              int n = parseLoRa_Msg(i,buf,APP_ID, DEV_ID,CH, PAYLOAD,&rssi, seq_id, DOWN);
2083              if(n == -1) { //Se for -1, ocorreu erro de MAC ou SEQID
2084                return;
2085              }
2086              getMsg_SeqID(buf, &seq_id, DOWN);
2087              if(FSM == 3) { //Recebeu ACK
2088                for(int j=16; j < 26; j++) {
2089                        received[(j-16)*2] = (buf[j] & 0xF0) >> 4;
2090                        received[(j-16)*2 + 1] = buf[j] & 0x0F;
2091                        tam_recv = tam_recv + 2;
2092                  }
2093                envio_priority = true;
2094                  Serial.print(F("[DEBUG] Received: "));
2095                  for(int j=0; j<i; j++) {
2096                       Serial.print(buf[j], HEX);
2097                       Serial.print(F(" "));
2098                  }
2099                  Serial.println(F("--> ACK"));
2100                  Serial.println(F("[DEBUG] Sent data!!!"));
2101                  Serial.print(F("[DEBUG] RSSI(dbm): "));
```

```
2102                Serial.println(rssi);
2103                Serial.print(F("[DEBUG] Delay(ms): "));
2104                Serial.println(fim-comeco);
2105                Serial.println();
2106
2107                for (int i = 12; i < 16; i++) {
2108                    if ( String((unsigned char)buf[i], HEX).length() >= 2 ) {
2109                      tmp += String((unsigned char)buf[i], HEX);
2110                    } else {
2111                      tmp += ajustConversion(String((unsigned char)buf[i], HEX));
2112                      }
2113                }
2114
2115                /*
2116                String _interval = "";
2117              _interval = String((unsigned char)buf[23], HEX);
2118              interval = hexToDec(_interval);
2119                configuraRTC(interval);
2120                */
2121
2122
2123
2124              }
2125              else {
2126                Serial.println(F("[DEBUG] Timeout ACK"));
2127                Serial.println();
2128              }
2129            }
2130            else {
2131              Serial.println(F("[DEBUG] Timeout PRIORITY"));
2132              Serial.println();
2133            }
2134            }
2135            if(FSM_SEND == SEND_TEST) {
2136              test = false;
2137              montar_resposta(response, &rssi, TEST, UP);
2138            getMsg_CHANNEL(response, CH, UP);
2139            /*#ifdef CANAL_SEMI_RAND
2140              for(int j=0; j<2; j++){
2141                CH_FIXO[j] = CH[j];
2142              }
2143            #endif*/
2144
2145            if(at == false) {
2146              Serial.println(F("[DEBUG] Sending..."));
2147            }
2148            if(rch == 0) {
2149              frequency_1 = CHANNEL_5; //CANAL INICIAL DEFINIDO AQUI
2150            }
2151            else {
2152              CH_FIXO[0] = random(8) + 1;
2153              frequency_1 = set_CHANNEL(CH_FIXO, UP);
2154            }
2155            #ifdef CANAL_FIXO
2156              get_CHANNEL(CH_FIXO, (int)frequency_1, UP);
2157            #endif
2158            LoRa.setSpreadingFactor(10);
2159            LoRa.setTxPower(17, PA_OUTPUT_PA_BOOST_PIN);
2160            //LoRa.set20dBm_sx127x(true); //MODIFICADO
2161            comeco = millis();
2162            #ifdef CANAL_FIXO
2163              SEND_LoRa(response, 21, CH_FIXO, frequency_1, &frequency_2, UP);
2164            #else
2165              SEND_LoRa(response, 21, CH, frequency_1, &frequency_2, UP);
2166            #endif
2167
2168
2169            i = rcvLora(buf, frequency_2);
2170            if(i > 0) {
2171              fim = millis();
2172              getMsg_SeqID(buf, &seq_id, DOWN);
2173              seq_id--;
2174              int n = parseLoRa_Msg(i,buf,APP_ID, DEV_ID,CH, PAYLOAD,&rssi, seq_id, DOWN);
2175              if(n == -1) { //Se for -1, ocorreu erro de MAC ou SEQID
2176                return;
2177              }
2178              _rssi = rssi;
2179              sinal = _rssi;
2180              getMsg_SeqID(buf, &seq_id, DOWN);
2181              if(FSM == 3) { //Recebeu ACK
2182                envio_test = true;
```

```
            if(at == false){
                Serial.print(F("[DEBUG] Received: "));
                for(int j=0; j<i; j++) {
                    Serial.print(buf[j], HEX);
                    Serial.print(F(" "));
                }
                Serial.println(F("--> ACK"));
                Serial.println(F("[DEBUG] Sent data!!!"));
                Serial.print(F("[DEBUG] RSSI(dbm): "));
                Serial.println(_rssi);
                Serial.print(F("[DEBUG] Delay(ms): "));
                Serial.println(fim-comeco);
                Serial.println();
            }

              /*for (int i = 12; i < 16; i++) {
                  if ( String((unsigned char)buf[i], HEX).length() >= 2 ) {
                    tmp += String((unsigned char)buf[i], HEX);
                  } else {
                     tmp += ajustConversion(String((unsigned char)buf[i], HEX));
                  }
              }

              //String _interval = "";
          //_interval = String((unsigned char)buf[23], HEX);
          //interval = hexToDec(_interval);
              //configuraRTC(1);*/
              test = true;


          }
          if(FSM == 14) { // Recebeu um DOS
            if(at == false){
              Serial.print(F("[DEBUG] Received: "));
            for(int j=0; j<i; j++) {
                Serial.print(buf[j], HEX);
                Serial.print(F(" "));
            }
            Serial.println(F("--> DOS"));
          }
          test = false;

            //return;
          }
        }
        else {
          if(at == false){
            Serial.println(F("[DEBUG] Timeout TEST"));
          }
        }

        }
        if((envio_dados == false) && (envio_priority == false) && (envio_test == false)){
          waitSEND();
        }

    }
  }

void waitSEND(){
  long randomico;

  randomico = random(4000);
  randomico = randomico + 1000;
  delay(randomico);
}

#endif


int rcvLora(unsigned char * buf, long frequency_2){

    LoRa.setFrequency(frequency_2);
    int i = 0;
    bool flag = false;
    int timeout = 3000;
    if(priority == true) {
      timeout = 5000;
    }
    unsigned long inicio = millis();

```

```
    while(!(flag) && ((millis()−inicio) < timeout)){
        int packetSize = LoRa.parsePacket();
        if (packetSize) {
            while (LoRa.available()) {
                buf[i] = ((char)LoRa.read());
                i++;
            }
        flag = true;
      }
    }
    return i;
}

void gen_type(unsigned char * msg, int type, int UpouDown){

    switch(UpouDown) {
        case 1: //Uplink
            switch(type) {

            case 1: //Type 1 −> 0x01 − RTS
                msg[20] = (0x01 << 4);
                break;
            case 2: //Type 2 −> 0x02 − CTS
                msg[20] = (0x02 << 4);
                break;
            case 3: //Type 3 −> 0x03 − ACK
                msg[20] = (0x03 << 4);
                break;
            case 4: //Type 4 −> 0x04 − DATA
                msg[20] = (0x04 << 4);
                break;
            case 7: //Type 7 −> 0x07 − PRIORITY
              msg[20] = (0X07 << 4);
              break;
            case 14: //Type 5 −> 0x0E − DOS
                msg[20] = (0x0E << 4);
                break;
            case 15: //Type 15 −> 0x0F − TEST
              msg[20] = (0x0F << 4);
              break;
            }
        break;

        case 2: //Downlink
            switch(type) {

            case 1: //Type 1 −> 0x01 − RTS
                msg[10] = (0x01 << 4);
                break;
            case 2: //Type 2 −> 0x02 − CTS
                msg[10] = (0x02 << 4);
                break;
            case 3: //Type 3 −> 0x03 − ACK
                msg[10] = (0x03 << 4);
                break;
            case 4: //Type 4 −> 0x04 − DATA
                msg[10] = (0x04 << 4);
                break;
            case 7: //Type 7 −> 0x07 − PRIORITY
              msg[10] = (0X07 << 4);
              break;
            case 14: //Type 5 −> 0x0E − DOS
                msg[10] = (0x0E << 4);
                break;
            case 15: //Type 15 −> 0x0F − TEST
              msg[10] = (0x0F << 4);
              break;
            }
        break;
    }
}

void gen_SEQ_ID(unsigned char * msg, int UpouDown, int type){
    static unsigned short i;

    i++;
    if((type == CTS) && (i >= 65533)) {
        i=0;
    }
    if((type == RTS) && (i >= 65534)) {
        i=0;
```

```
    }
    if (i >= 65535){ //maximo do unsigned short
        i = 0;
    }
    switch(UpouDown) {
        case 1: //Uplink
            msg[18] = (i >> 8) & 0xFF;
            msg[19] = i & 0xFF;
            break;

        case 2: //Downlink
            msg[8] = (i >> 8) & 0xFF;
            msg[9] = i & 0xFF;
            break;
    }

}

void gen_rssi(unsigned char *msg, int *rssi) {

  *rssi = LoRa.packetRssi();
  *rssi = abs(*rssi);
  msg[11] = (*rssi) & 0xFF;

}

void gen_APP_ID(unsigned char * msg){

    msg[0] = _appId[0];       //0x30;//0x30; //0x30 Aplicacao de temperatura
    msg[1] = _appId[1];       //0x30;//0x61; //0x61
    msg[2] = _appId[2];       //0x30;//0x66; //0x66
    msg[3] = _appId[3];       //0x30;//0x31; //0x31
    msg[4] = _appId[4]; //0x35;//0x30; //0x30

}

void gen_DEV_ID(unsigned char * msg){

    msg[5] = _devId[0];       //0x59;//0x00; //0x00 Aplicacao de temperatura
    msg[6] = _devId[1];       //0xF4;//0x00; //0x00
    msg[7] = _devId[2];       //0x32;//0x00; //0x00
    msg[8] = _devId[3];       //0xDA;//0x00; //0x00
    msg[9] = _devId[4];       //0x17;//0x02; //0x02

}

void gen_MAC_ID(unsigned char * msg, int UpouDown){

  int tam;
  AES aes; //Cria a classe aes.
  int randomico = random(10)+1;

  byte out[16], inp[16] ="";//Cria arrays (vetores) para a chave, input e output de dados.

  switch(UpouDown) {
    case 1: { //Uplink

      tam = 13;

      for(int i=0; i<10; i++) {
        inp[i] = msg[i];
      }
      for(int i=10; i<tam; i++) {
        inp[i] = msg[i+8];
      }
      switch(randomico) {
        case 1:
          aes.set_key(key1, 16);
          aes.encrypt(inp, out);
        break;
        case 2:
          aes.set_key(key2, 16);
          aes.encrypt(inp, out);
        break;
        case 3:
          aes.set_key(key3, 16);
          aes.encrypt(inp, out);
        break;
        case 4:
          aes.set_key(key4, 16);
          aes.encrypt(inp, out);
```

```
2426            break;
2427            case 5:
2428              aes.set_key(key5, 16);
2429              aes.encrypt(inp, out);
2430            break;
2431            case 6:
2432              aes.set_key(key6, 16);
2433              aes.encrypt(inp, out);
2434            break;
2435            case 7:
2436              aes.set_key(key7, 16);
2437              aes.encrypt(inp, out);
2438            break;
2439            case 8:
2440              aes.set_key(key8, 16);
2441              aes.encrypt(inp, out);
2442            break;
2443            case 9:
2444              aes.set_key(key9, 16);
2445              aes.encrypt(inp, out);
2446            break;
2447            case 10:
2448              aes.set_key(key10, 16);
2449              aes.encrypt(inp, out);
2450            break;
2451          }
2452
2453          for(int i=10; i<18; i++) { //Preencher o uplink com o MAC a partir do resultado do AES128
2454            msg[i] = out[i-10];
2455          }
2456          aes.clean();//Limpa a chave e residuos sensiveis da encriptação.
2457
2458        }
2459          break;
2460
2461        case 2:{ //Downlink
2462
2463          tam = 4;
2464
2465          for(int i=0; i<tam; i++) {
2466            inp[i] = msg[i+8];
2467          }
2468          switch(randomico) {
2469            case 1:
2470              aes.set_key(key1, 16);
2471              aes.encrypt(inp, out);
2472            break;
2473           case 2:
2474              aes.set_key(key2, 16);
2475              aes.encrypt(inp, out);
2476            break;
2477            case 3:
2478              aes.set_key(key3, 16);
2479              aes.encrypt(inp, out);
2480            break;
2481            case 4:
2482              aes.set_key(key4, 16);
2483              aes.encrypt(inp, out);
2484            break;
2485            case 5:
2486              aes.set_key(key5, 16);
2487              aes.encrypt(inp, out);
2488            break;
2489            case 6:
2490              aes.set_key(key6, 16);
2491              aes.encrypt(inp, out);
2492            break;
2493            case 7:
2494              aes.set_key(key7, 16);
2495              aes.encrypt(inp, out);
2496            break;
2497            case 8:
2498              aes.set_key(key8, 16);
2499              aes.encrypt(inp, out);
2500            break;
2501            case 9:
2502              aes.set_key(key9, 16);
2503              aes.encrypt(inp, out);
2504            break;
2505            case 10:
2506              aes.set_key(key10, 16);
```

```
            aes.encrypt(inp, out);
          break;
        }

        for(int i=0; i<8; i++) { //Preencher o downlink com o MAC a partir do resultado do AES128
          msg[i] = out[i];
        }
        aes.clean();//Limpa a chave e residuos sensiveis da encriptação.
      }
        break;
    }
}

void gen_PAYLOAD(unsigned char *msg, byte *PAYLOAD, int UpouDown) {
        int tam, index=15, aux;
        AES aes;//Cria a classe aes.
        byte out[16], inp[16] ="";//Cria arrays (vetores) para a chave, input e output de dados.
        byte test[16]="";

        if(payload.length() % 2 == 0) {
            tam = payload.length()/2;
        }
        else {
            tam = payload.length()/2 + 1;
        }
        if(payload.length() > 32) {
            tam = 16;
        }
        if(criptografia == true) {
            if(tam < 16){
              aux = tam;
              while(aux > 0){ /////
                PAYLOAD[index] = PAYLOAD[aux-1]; /////
                index--; /////
                aux--; /////
              } /////
          for(int i=0; i<(16-tam); i++) {
            PAYLOAD[i] = 0x00;
          }
            }
            for(int i=0; i < 16; i++) {
              //Serial.print(PAYLOAD[i]);
              //Serial.print(F(" "));
            inp[i] = PAYLOAD[i];
          }
          Serial.println();
          aes.set_key(AppKey, 16);
          aes.encrypt(inp, out);
          aes.clean();
        }


        switch(UpouDown) {
          case 1: //Uplink
            if(criptografia == true) {
              for (int i = 0; i < 16; i++) {
                  msg[i + 21] = out[i];
              }
              }
            else {
              for (int i = 0; i < tam; i++) {
                  msg[i + 21] = PAYLOAD[i];
              }
            }

          break;

          case 2: //Downlink
              for (int i = 0; i < tam; i++) {
                  msg[i + 12] = PAYLOAD[i];
              }
          break;
        }

    }

void SEND_LoRa(const unsigned char * msg, int len, char *CHANNEL, long frequency_1, long *frequency_2, int UpouDown){

  if(UpouDown == UP) {
    *frequency_2 = set_CHANNEL(CHANNEL, DOWN);
  }
```

```
    else {
      *frequency_2 = set_CHANNEL(CHANNEL, UP);
    }
    LoRa.setFrequency(frequency_1);
    LoRa.beginPacket();
    LoRa.write((uint8_t *) msg, len);
    LoRa.endPacket();
    LoRa.setFrequency(*frequency_2);

}

void get_CHANNEL(char *CHANNEL, int frequency, int UpouDown) {

  switch(UpouDown){
    case 1: //Uplink
      switch(frequency) {
        case 916800000:
          CHANNEL[0] = 0x01;
        break;

        case 917000000:
          CHANNEL[0] = 0x02;
        break;

        case 917200000:
          CHANNEL[0] = 0x03;
        break;

        case 917400000:
          CHANNEL[0] = 0x04;
        break;

        case 917600000:
          CHANNEL[0] = 0x05;
        break;

        case 917800000:
          CHANNEL[0] = 0x06;
        break;

        case 918000000:
          CHANNEL[0] = 0x07;
        break;

        case 918200000:
          CHANNEL[0] = 0x08;
        break;
      }
    break;

    case 2: //Downlink
      switch(frequency) {
        case 923300000:
          CHANNEL[0] = 0x01;
        break;

        case 923900000:
          CHANNEL[0] = 0x02;
        break;

        case 924500000:
          CHANNEL[0] = 0x03;
        break;

        case 925100000:
          CHANNEL[0] = 0x04;
        break;

        case 925700000:
          CHANNEL[0] = 0x05;
        break;

        case 926300000:
          CHANNEL[0] = 0x06;
        break;

        case 926900000:
          CHANNEL[0] = 0x07;
        break;

        case 927500000:
```

```
            CHANNEL[0] = 0x08;
          break;
        }
      break;
    }

}

long set_CHANNEL(char *CHANNEL, int UpouDown) {
  long frequency=0;

  switch(UpouDown){
    case 1: //Uplink
      switch(CHANNEL[0]) {
        case 0x01:
          frequency = 916.8E6;
        break;

        case 0x02:
          frequency = 917.0E6;
        break;

        case 0x03:
          frequency = 917.2E6;
        break;

        case 0x04:
          frequency = 917.4E6;
        break;

        case 0x05:
          frequency = 917.6E6;
        break;

        case 0x06:
          frequency = 917.8E6;
        break;

        case 0x07:
          frequency = 918.0E6;
        break;

        case 0x08:
          frequency = 918.2E6;
        break;
      }
    break;

    case 2: //Downlink
      switch(CHANNEL[0]) {
        case 0x01:
          frequency = 923.3E6;
        break;

        case 0x02:
          frequency = 923.9E6;
        break;

        case 0x03:
          frequency = 924.5E6;
        break;

        case 0x04:
          frequency = 925.1E6;
        break;

        case 0x05:
          frequency = 925.7E6;
        break;

        case 0x06:
          frequency = 926.3E6;
        break;

        case 0x07:
          frequency = 926.9E6;
        break;

        case 0x08:
          frequency = 927.5E6;
        break;
```

```
        }
      break;
    }

    return frequency;
}

char convertCharToHex(char ch, int opcao) {

    char returnType;

    switch(opcao){

      case 1:
        switch (ch) {
        case '0':
          returnType = 0x00;
          break;
        case '1' :
          returnType = 0x10;
          break;
        case '2':
          returnType = 0x20;
          break;
        case '3':
          returnType = 0x30;
          break;
        case '4' :
          returnType = 0x40;
          break;
        case '5':
          returnType = 0x50;
          break;
        case '6':
          returnType = 0x60;
          break;
        case '7':
          returnType = 0x70;
          break;
        case '8':
          returnType = 0x80;
          break;
        case '9':
          returnType = 0x90;
          break;
        case 'A':
          returnType = 0xA0;
          break;
        case 'B':
          returnType = 0xB0;
          break;
        case 'C':
          returnType = 0xC0;
          break;
        case 'D':
          returnType = 0xD0;
          break;
        case 'E':
          returnType = 0xE0;
          break;
        case 'F' :
          returnType = 0xF0;
          break;
        default:
          returnType = 0x00;
          break;
      }
      return returnType;
      break;

      case 2:
        switch (ch) {
        case '0':
          returnType = 0x00;
          break;
        case '1' :
          returnType = 0x01;
          break;
        case '2':
          returnType = 0x02;
          break;
```

```
      case '3':
        returnType = 0x03;
        break;
      case '4' :
        returnType = 0x04;
        break;
      case '5':
        returnType = 0x05;
        break;
      case '6':
        returnType = 0x06;
        break;
      case '7':
        returnType = 0x07;
        break;
      case '8':
        returnType = 0x08;
        break;
      case '9':
        returnType = 0x09;
        break;
      case 'A':
        returnType = 0x0A;
        break;
      case 'B':
        returnType = 0x0B;
        break;
      case 'C':
        returnType = 0x0C;
        break;
      case 'D':
        returnType = 0x0D;
        break;
      case 'E':
        returnType = 0x0E;
        break;
      case 'F' :
        returnType = 0x0F;
        break;
      default:
        returnType = 0x00;
        break;
    }
    return returnType;
    break;

  }

}

char convertDecToHex(int dec, int opcao) {

  char returnType;

  switch(opcao){

    case 1:
      switch (dec) {
      case 0:
        returnType = 0x00;
        break;
      case 1 :
        returnType = 0x10;
        break;
      case 2:
        returnType = 0x20;
        break;
      case 3:
        returnType = 0x30;
        break;
      case 4 :
        returnType = 0x40;
        break;
      case 5:
        returnType = 0x50;
        break;
      case 6:
        returnType = 0x60;
        break;
      case 7:
        returnType = 0x70;
```

```
        break;
      case 8:
        returnType = 0x80;
        break;
      case 9:
        returnType = 0x90;
        break;
      case 10:
        returnType = 0xA0;
        break;
      case 11:
        returnType = 0xB0;
        break;
      case 12:
        returnType = 0xC0;
        break;
      case 13:
        returnType = 0xD0;
        break;
      case 14:
        returnType = 0xE0;
        break;
      case 15 :
        returnType = 0xF0;
        break;
      default:
        returnType = 0x00;
        break;
    }
    return returnType;
    break;

    case 2:
      switch (dec) {
        case 0:
          returnType = 0x00;
          break;
        case 1 :
          returnType = 0x01;
          break;
        case 2:
          returnType = 0x02;
          break;
        case 3:
          returnType = 0x03;
          break;
        case 4 :
          returnType = 0x04;
          break;
        case 5:
          returnType = 0x05;
          break;
        case 6:
          returnType = 0x06;
          break;
        case 7:
          returnType = 0x07;
          break;
        case 8:
          returnType = 0x08;
          break;
        case 9:
          returnType = 0x09;
          break;
        case 10:
          returnType = 0x0A;
          break;
        case 11:
          returnType = 0x0B;
          break;
        case 12:
          returnType = 0x0C;
          break;
        case 13:
          returnType = 0x0D;
          break;
        case 14:
          returnType = 0x0E;
          break;
        case 15:
          returnType = 0x0F;
```

```
            break;
          default:
            returnType = 0x00;
            break;
      }
      return returnType;
      break;

    }

}

char convertDecToChar(int dec) {

  char returnType;

      switch (dec) {
      case 0:
        returnType = '0';
        break;
      case 1 :
        returnType = '1';
        break;
      case 2:
        returnType = '2';
        break;
      case 3:
        returnType = '3';
        break;
      case 4 :
        returnType = '4';
        break;
      case 5:
        returnType = '5';
        break;
      case 6:
        returnType = '6';
        break;
      case 7:
        returnType = '7';
        break;
      case 8:
        returnType = '8';
        break;
      case 9:
        returnType = '9';
        break;
      case 10:
        returnType = 'A';
        break;
      case 11:
        returnType = 'B';
        break;
      case 12:
        returnType = 'C';
        break;
      case 13:
        returnType = 'D';
        break;
      case 14:
        returnType = 'E';
        break;
      case 15 :
        returnType = 'F';
        break;
      default:
        returnType = '0';
        break;
    }
    return returnType;


}



void gen_CHANNEL(unsigned char * msg, int UpouDown){

  switch(UpouDown) {
    case 1: //Uplink
      msg[20] |= (random(8) + 1);
```

```
      break;
    case 2: //Downlink
      msg[10] |= (random(8) + 1);
      break;
  }

}

int parseLoRa_Msg(int Msg_size, unsigned char * Msg, char * APP_ID, char * DEV_ID, char * CH, String &PAYLOAD, int *Rssi, unsigned
    ↪ long seq_id, int UpouDown){

    int i = 0, len;

    switch(UpouDown){
      case 1: //Uplink

        getMsg_AppID(Msg, APP_ID);
        getMsg_DevID(Msg, DEV_ID);

        if(!check_MacID(Msg, UP)){
          if(at == false){
            Serial.println(F("[DEBUG] MAC ID Invalido."));
          }
          return MAC_ID_INVALIDO;
        }


       if(!check_SeqID(Msg, seq_id, UP)){
          if(at== false){
            Serial.println(F("[DEBUG] SeqID invalido"));
          }
          return SEQ_ID_INVALIDO;
       }

        getMsg_Type(Msg, UP);
        getMsg_CHANNEL(Msg, CH, UP);

        if(Msg_size > 21) { //Significa que há payload
          len = Msg_size - 21;
          PAYLOAD.reserve(len);
          PAYLOAD = "1";
          for(int i=0; i<len-1; i++) { //E necessario inicializar a String, senao setCharAt nao funciona
            PAYLOAD += "1";
          }
          for(int i = 0; i < len ; i++) {
            PAYLOAD.setCharAt(i, Msg[21 + i]);
          }
        }

        return 0;

      break;

      case 2: //Downlink

        if(!check_MacID(Msg, DOWN)){
          if(at == false){
            Serial.println(F("[DEBUG] MAC ID Invalido."));
          }
          return MAC_ID_INVALIDO;
        }

        if(!check_SeqID(Msg, seq_id, DOWN)){
          if(at== false){
            Serial.println(F("[DEBUG] SeqID invalido"));
          }
          return SEQ_ID_INVALIDO;
        }

        getMsg_Type(Msg, DOWN);
        getMsg_CHANNEL(Msg, CH, DOWN);
        *Rssi = getRSSI(Msg);

        if(Msg_size > 12) { //Significa que há payload
          len = Msg_size - 12;
          PAYLOAD.reserve(len);
          PAYLOAD = "1";
          for(int i=0; i<len-1; i++) { //E necessario inicializar a String, senao setCharAt nao funciona
            PAYLOAD += "1";
          }
          for(int i = 0; i < len ; i++) {
```

```
            PAYLOAD.setCharAt(i, Msg[12 + i]);
          }
        }

        return 0;

      break;
    }

}

unsigned long hexToDec(String hexString) {

  unsigned long decValue = 0;
  int nextInt;

  for (int i = 0; i < hexString.length(); i++) {

    nextInt = int(hexString.charAt(i));
    if (nextInt >= 48 && nextInt <= 57) nextInt = map(nextInt, 48, 57, 0, 9);
    if (nextInt >= 65 && nextInt <= 70) nextInt = map(nextInt, 65, 70, 10, 15);
    if (nextInt >= 97 && nextInt <= 102) nextInt = map(nextInt, 97, 102, 10, 15);
    nextInt = constrain(nextInt, 0, 15);

    decValue = (decValue * 16) + nextInt;
  }

  return decValue;
}

int getRSSI(const unsigned char * Msg_copy) {
  int rssi;

  String _rssi = String((unsigned char)Msg_copy[11], HEX);
  rssi = hexToDec(_rssi);

  return rssi;

}

void DecToHex(unsigned long Dec, char *cvt) { //So funciona se o numero de hexas for par
  int num, i=0;
  int vetor[10] = {0,0,0,0,0,0,0,0,0,0};
  num = Dec;

  while((num/16) != 0) {
    vetor[i] = num % 16;
    i++;
    num = (num / 16);
  }
  vetor[i] = num % 16;
  int j=0;
  for(i; i>=0; i--) {
    int a = vetor[i];
    int b = vetor[i-1];
    cvt[j] = convertDecToHex(a,1) + convertDecToHex(b,2);
    //cvt[j] = (convertDecToHex(a,2) << 4) | convertDecToHex(b,2);
    j++;
    i--;
  }

}

void getMsg_AppID(const unsigned char * Msg_copy, char * AppID){

    int i;
    for(i = 0; i < 5; i++){
      AppID[i] = Msg_copy[i];
    }
}

int ajustchar(char* seqid, int opcao) {

  switch(seqid[opcao]){

    case 0x00:
    return 1;
    break;

    case 0x01:
    return 1;
```

```
      break;

      case 0x02:
      return 1;
      break;

      case 0x03:
      return 1;
      break;

      case 0x04:
      return 1;
      break;

      case 0x05:
      return 1;
      break;

      case 0x06:
      return 1;
      break;

      case 0x07:
      return 1;
      break;

      case 0x08:
      return 1;
      break;

      case 0x09:
      return 1;
      break;

      case 0x0A:
      return 1;
      break;

      case 0x0B:
      return 1;
      break;

      case 0x0C:
      return 1;
      break;

      case 0x0D:
      return 1;
      break;

      case 0x0E:
      return 1;
      break;

      case 0x0F:
      return 1;
      break;

      default:
      return 0;
      break;
    }


}


void getMsg_SeqID(const unsigned char * Msg_copy, unsigned long *seq_ID, int UpouDown) {
  char SeqID[2];
  String _seqid="", _seqid2="";
  int n;

  switch(UpouDown) {
    case 1: { //Uplink
      SeqID[0] = Msg_copy[18];
      SeqID[1] = Msg_copy[19];
      //_seqid = String((unsigned char)Msg_copy[18], HEX);
      //_seqid2 = String((unsigned char)Msg_copy[19], HEX);
      n = ajustchar(SeqID, 0);
      if(n == 1) {
        _seqid = (String(0, HEX) + String((unsigned char)Msg_copy[18], HEX));
```

```
3316        }
3317        else{
3318          _seqid = String((unsigned char)Msg_copy[18], HEX);
3319        }
3320        n = ajustchar(SeqID, 1);
3321        if(n == 1) {
3322          _seqid2 = (String(0, HEX) + String((unsigned char)Msg_copy[19], HEX));
3323        }
3324        else{
3325          _seqid2 = String((unsigned char)Msg_copy[19], HEX);
3326        }
3327      }
3328      break;
3329
3330      case 2: { //Downlink
3331        SeqID[0] = Msg_copy[8];
3332        SeqID[1] = Msg_copy[9];
3333        //_seqid = String((unsigned char)Msg_copy[8], HEX);
3334        //_seqid2 = String((unsigned char)Msg_copy[9], HEX);
3335        n = ajustchar(SeqID, 0);
3336        if(n == 1) {
3337          _seqid = (String(0, HEX) + String((unsigned char)Msg_copy[8], HEX));
3338        }
3339        else{
3340          _seqid = String((unsigned char)Msg_copy[8], HEX);
3341        }
3342        n = ajustchar(SeqID, 1);
3343        if(n == 1) {
3344          _seqid2 = (String(0, HEX) + String((unsigned char)Msg_copy[9], HEX));
3345        }
3346        else{
3347          _seqid2 = String((unsigned char)Msg_copy[9], HEX);
3348        }
3349      }
3350      break;
3351    }
3352
3353    String _seqid3 = _seqid + _seqid2;
3354    *seq_ID = hexToDec(_seqid3);
3355
3356  }
3357
3358  void getMsg_DevID(const unsigned char * Msg_copy, char * DevID){
3359
3360      int i;
3361      for(i = 5; i < 10; i++){
3362      DevID[i-5] = Msg_copy[i];
3363    }
3364  }
3365
3366  int check_SeqID(const unsigned char *Msg_copy, unsigned long seq_id, int UpouDown) {
3367    unsigned long _seq_id;
3368
3369    switch(UpouDown) {
3370      case 1: //Uplink
3371        getMsg_SeqID(Msg_copy, &_seq_id, UP);
3372      break;
3373
3374      case 2: //Downlink
3375        getMsg_SeqID(Msg_copy, &_seq_id, DOWN);
3376      break;
3377    }
3378
3379    //if(_seq_id <= seq_id) {
3380    if((_seq_id != seq_id+1) && ((_seq_id != seq_id+2)) && ((_seq_id != seq_id+3))) {
3381      //Serial.println("Seq_ID invalido");
3382      return 0;
3383    }
3384    else {
3385      return 1;
3386    }
3387
3388  }
3389
3390  int cripto(int key, const unsigned char *Msg_copy, byte *inp, int UpouDown) {
3391
3392    AES aes;//Cria a classe aes.
3393    byte out[16];//Cria arrays (vetores) para a chave, input e output de dados.
3394    int flag=0;
3395
3396    switch(key) {
```

```
3397        case 1:{
3398          aes.set__key(key1, 16);
3399          aes.encrypt(inp, out);
3400        }
3401        break;
3402        case 2:{
3403          aes.set__key(key2, 16);
3404          aes.encrypt(inp, out);
3405        }
3406        break;
3407        case 3:{
3408          aes.set__key(key3, 16);
3409          aes.encrypt(inp, out);
3410        }
3411        break;
3412        case 4:{
3413          aes.set__key(key4, 16);
3414          aes.encrypt(inp, out);
3415        }
3416        break;
3417        case 5:{
3418          aes.set__key(key5, 16);
3419          aes.encrypt(inp, out);
3420        }
3421        break;
3422        case 6:{
3423          aes.set__key(key6, 16);
3424          aes.encrypt(inp, out);
3425        }
3426        break;
3427        case 7:{
3428          aes.set__key(key7, 16);
3429          aes.encrypt(inp, out);
3430        }
3431        break;
3432        case 8:{
3433          aes.set__key(key8, 16);
3434          aes.encrypt(inp, out);
3435        }
3436        break;
3437        case 9:{
3438          aes.set__key(key9, 16);
3439          aes.encrypt(inp, out);
3440        }
3441        break;
3442        case 10:{
3443          aes.set__key(key10, 16);
3444          aes.encrypt(inp, out);
3445        }
3446        break;
3447
3448      }
3449      if(UpouDown == UP){
3450        for(int i=0; i<8; i++) {
3451          if(out[i] != Msg_copy[i+10]) {
3452              aes.clean();//Limpa a chave e residuos sensiveis da criptografia.
3453              return 1;
3454          }
3455      }
3456      }
3457      else {
3458        for(int i=0; i<8; i++) {
3459          if(out[i] != Msg_copy[i]) {
3460              aes.clean();//Limpa a chave e residuos sensiveis da criptografia.
3461              return 1;
3462          }
3463      }
3464      }
3465
3466
3467      if(flag == 0){
3468        aes.clean();//Limpa a chave e residuos sensiveis da criptografia.
3469        return 0;
3470      }
3471
3472  }
3473
3474  bool check_MacID(const unsigned char * Msg_copy, int UpouDown){
3475
3476      int i, flag=0;
3477      byte inp[16] ="";
```

```
3478
3479
3480  switch(UpouDown) {
3481    case 1:{ //Uplink
3482      for(i=0; i<10; i++) {
3483        inp[i] = Msg_copy[i];
3484      }
3485      for(i=18; i<21; i++){
3486        inp[i-8] = Msg_copy[i];
3487      }
3488
3489      i=1;
3490      while(i != 0) {
3491        i=0;
3492        flag = cripto(1, Msg_copy, inp, UP);
3493        if(flag == 0) {
3494          return true;
3495        }
3496        else {
3497          flag =0;
3498        }
3499        flag = cripto(2, Msg_copy, inp, UP);
3500        if(flag == 0) {
3501          return true;
3502        }
3503        else {
3504          flag =0;
3505        }
3506        flag = cripto(3, Msg_copy, inp, UP);
3507        if(flag == 0) {
3508          return true;
3509        }
3510        else {
3511          flag =0;
3512        }
3513        flag = cripto(4, Msg_copy, inp, UP);
3514        if(flag == 0) {
3515          return true;
3516        }
3517        else {
3518          flag =0;
3519        }
3520        flag = cripto(5, Msg_copy, inp, UP);
3521        if(flag == 0) {
3522          return true;
3523        }
3524        else {
3525          flag =0;
3526        }
3527        flag = cripto(6, Msg_copy, inp, UP);
3528        if(flag == 0) {
3529          return true;
3530        }
3531        else {
3532          flag =0;
3533        }
3534        flag = cripto(7, Msg_copy, inp, UP);
3535        if(flag == 0) {
3536          return true;
3537        }
3538        else {
3539          flag =0;
3540        }
3541        flag = cripto(8, Msg_copy, inp, UP);
3542        if(flag == 0) {
3543          return true;
3544        }
3545        else {
3546          flag =0;
3547        }
3548        flag = cripto(9, Msg_copy, inp, UP);
3549        if(flag == 0) {
3550          return true;
3551        }
3552        else {
3553          flag =0;
3554        }
3555        flag = cripto(10, Msg_copy, inp, UP);
3556        if(flag == 0) {
3557          return true;
3558        }
```

```
3559        }
3560
3561        if(flag == 1) {
3562          //Serial.println("[DEBUG] Mac invalido.");
3563          return false;
3564        }
3565      }
3566      break;
3567
3568      case 2:{ //Downlink
3569
3570        for(i=8; i<12; i++) {
3571          inp[i-8] = Msg_copy[i];
3572        }
3573
3574        i=1;
3575        while(i != 0) {
3576          i=0;
3577          flag = cripto(1, Msg_copy, inp, DOWN);
3578          if(flag == 0) {
3579            return true;
3580          }
3581          else {
3582            flag =0;
3583          }
3584          flag = cripto(2, Msg_copy, inp, DOWN);
3585          if(flag == 0) {
3586            return true;
3587          }
3588          else {
3589            flag =0;
3590          }
3591          flag = cripto(3, Msg_copy, inp, DOWN);
3592          if(flag == 0) {
3593            return true;
3594          }
3595          else {
3596            flag =0;
3597          }
3598          flag = cripto(4, Msg_copy, inp, DOWN);
3599          if(flag == 0) {
3600            return true;
3601          }
3602          else {
3603            flag =0;
3604          }
3605          flag = cripto(5, Msg_copy, inp, DOWN);
3606          if(flag == 0) {
3607            return true;
3608          }
3609          else {
3610            flag =0;
3611          }
3612          flag = cripto(6, Msg_copy, inp, DOWN);
3613          if(flag == 0) {
3614            return true;
3615          }
3616          else {
3617            flag =0;
3618          }
3619          flag = cripto(7, Msg_copy, inp, DOWN);
3620          if(flag == 0) {
3621            return true;
3622          }
3623          else {
3624            flag =0;
3625          }
3626          flag = cripto(8, Msg_copy, inp, DOWN);
3627          if(flag == 0) {
3628            return true;
3629          }
3630          else {
3631            flag =0;
3632          }
3633          flag = cripto(9, Msg_copy, inp, DOWN);
3634          if(flag == 0) {
3635            return true;
3636          }
3637          else {
3638            flag =0;
3639          }
```

```
3640          flag = cripto(10, Msg_copy, inp, DOWN);
3641          if(flag == 0) {
3642            return true;
3643          }
3644        }
3645
3646        if(flag == 1) {
3647          //Serial.println("[DEBUG] Mac invalido.");
3648          return false;
3649        }
3650      }
3651
3652      break;
3653    }
3654
3655    return true;
3656  }
3657
3658  void getMsg_Type(unsigned char * Msg_copy, int UpouDown){
3659    char type;
3660
3661    switch(UpouDown) {
3662
3663      case 1: //Uplink
3664
3665        type = Msg_copy[20] >> 4;
3666        type = type & 0x0F;
3667
3668        switch (type){
3669
3670          case 0x01:
3671            FSM = 1; // FSM = 1 indica RTS
3672            break;
3673
3674          case 0x02: //FSM = 2 indica CTS
3675            FSM = 2;
3676            break;
3677
3678          case 0x03: //FSM = 3 indica ACK
3679            FSM = 3;
3680            break;
3681
3682          case 0x04: //FSM = 4 indica DATA
3683            FSM = 4;
3684            break;
3685
3686          case 0x07: //FSM = 7 indica PRIORITY
3687            FSM = 7;
3688            break;
3689
3690          case 0x0E: //FSM = 14 indica DOS
3691            FSM = 14;
3692            break;
3693
3694          case 0x0F:          //FSM = 15 indica TEST
3695            FSM = 15;
3696            break;
3697
3698          default:
3699            FSM = 0;
3700            break;
3701          }
3702          return;
3703
3704      break;
3705
3706      case 2: //Downlink
3707
3708        type = Msg_copy[10] >> 4;
3709        type = type & 0x0F;
3710
3711        switch (type){
3712
3713          case 0x01:
3714            FSM = 1; // FSM = 1 indica RTS
3715            break;
3716
3717          case 0x02: //FSM = 2 indica CTS
3718            FSM = 2;
3719            break;
3720
```

```
3721          case 0x03: //FSM = 3 indica ACK
3722            FSM = 3;
3723            break;
3724
3725          case 0x04: //FSM = 4 indica DATA
3726            FSM = 4;
3727            break;
3728
3729          case 0x07: //FSM = 7 indica PRIORITY
3730            FSM = 7;
3731            break;
3732
3733          case 0x0E: //FSM = 14 indica DOS
3734            FSM = 14;
3735            break;
3736
3737          case 0x0F:          //FSM = 15 indica TEST
3738            FSM = 15;
3739            break;
3740
3741          default:
3742            FSM = 0;
3743            break;
3744          }
3745          return;
3746
3747      break;
3748    }
3749
3750  }
3751
3752  void getMsg_CHANNEL(const unsigned char * Msg_copy, char *CH, int UpouDown){
3753    char CHANNEL;
3754
3755    switch(UpouDown) {
3756
3757      case 1: //Uplink
3758        CHANNEL = (Msg_copy[20] & 0x0F);
3759        CH[0] = CHANNEL;
3760      break;
3761
3762      case 2: //Downlink
3763        CHANNEL = (Msg_copy[10] & 0x0F);
3764        CH[0] = CHANNEL;
3765      break;
3766    }
3767
3768
3769  }
3770
3771  void montar_resposta(unsigned char *buf, int *rssi, int type, int UpouDown) {
3772    switch(UpouDown){
3773
3774      case 1:
3775
3776        char i;
3777          gen_APP_ID(buf);
3778          gen_DEV_ID(buf);
3779          gen_SEQ_ID(buf, UP, type);
3780          gen_type(buf, type, UP);
3781          gen_CHANNEL(buf, UP);
3782          if((type == DATA) || (type == PRIORITY)) {
3783              byte tmp[16];
3784              if(payload.length() % 2 == 0) { // Se a quantidade de hexas for par
3785                  for (i = 0; i < (payload.length()/2); i++) {
3786                    byte extract;
3787                    char a = payload[2 * i];
3788                    char b = payload[2 * i + 1];
3789                    extract = ((convertCharToHex(a,1)) + (convertCharToHex(b,2)));
3790                    tmp[i] = extract;
3791                  }
3792              }
3793              else { // Se a quantidade de hexas for impar, deve-se pegar 1 hexa na ultima posição
3794                i=0;
3795                char a = payload[0];
3796                  char b = 0;
3797                  byte extract = ((convertCharToHex(b,1)) + (convertCharToHex(a,2)));
3798                  tmp[i] = extract;
3799                  for (i = 1; i < (payload.length()/2 + 1); i++) {
3800                      byte extract;
3801                      char a = payload[2 * i - 1];
```

```
                    char b = payload[2 * i];
                    extract = ((convertCharToHex(a,1)) + (convertCharToHex(b,2)));
                    tmp[i] = extract;
                }
            }
            gen_PAYLOAD(buf, tmp, UP);
        }
        gen_MAC_ID(buf, UP);

    break;

    case 2:

      gen_SEQ_ID(buf, DOWN, type);
      gen_type(buf, type, DOWN);
      gen_CHANNEL(buf, DOWN);
      gen_rssi(buf, rssi);

    break;
  }
}

String ajustConversion(String ch) {
    String tmp = "";
    if (ch == "0")
      tmp = "00";

    else if (ch == "1")
      tmp = "01";

    else if (ch == "2")
      tmp = "02";

    else if (ch == "3")
      tmp = "03";

    else if (ch == "4")
      tmp = "04";

    else if (ch == "5")
      tmp = "05";

    else if (ch == "6")
      tmp = "06";

    else if (ch == "7")
      tmp = "07";

    else if (ch == "8")
      tmp = "08";

    else if (ch == "9")
      tmp = "09";

    else if (ch == "a")
      tmp = "0a";

    else if (ch == "b")
      tmp = "0b";

    else if (ch == "c")
      tmp = "0c";

    else if (ch == "d")
      tmp = "0d";

    else if (ch == "e")
      tmp = "0e";

    else if (ch == "f")
      tmp = "0f";

    return tmp;

}
```

Abaixo se encontra o código do Gateway.

Gateway.ino

```
#include <OnAirProtocol.h>

String payload="";
bool criptografia = false; //Homebase nao possui criptografia de payload, mas deve existir essa variavel
char _devId[5] = { 0x30, 0x31, 0x32, 0x33, 0x34 };
char _appId[5] = { 0x30, 0x31, 0x32, 0x33, 0x34 };
char _HomeBaseId[5] = { 0xF0, 0x0D, 0xF4, 0xD7, 0x48 };
const int pinoLED = 16;
//unsigned long comeco, fim, _comeco, _fim;
int toConfig = 1;

void setup() {
  Serial.begin(115200);

  //Inicializar LoRa
  _start_LoRa();

  pinMode(pinoLED, OUTPUT);

  Serial.print(F("HomeBase ID: "));
  for (int i = 0; i < 5; i++) {
    Serial.print(F(" "));
    Serial.print((unsigned char)_HomeBaseId[i], HEX);
  }
  Serial.println(F(" "));

  // Start da EEPROM
  EEPROM.begin(EEPROM_SIZE);
  const char* ssid = "gateway_config";
  const char* pass = "123456789";
  String Domain = "gateway.com";
  String configPage = "<!DOCTYPE html>\n<html>\n<head>\n<meta charset='utf-8'>\n<meta name='viewport' content='width=
      ↪ device-width, initial-scale=1, maximum-scale=1, minimum-scale=1, user-scalable=no, minimal-ui, viewport-fit=cover'>\n<
      ↪ meta name='apple-mobile-web-app-capable' content='yes'>\n<title>Home base</title>\n</head>\n<style>\nbody\n{\n
      ↪ background-color: #efefef;\n margin: 0;\n padding: 0;\n font-family:'helvetica';\n}\n.listNetworks\n{\n background-color: #fff
      ↪ ;\n box-sizing: border-box;\n}\n#lista\n{\n width: 100%;\n list-style:none;\n margin: 0;\n padding: 0;\n}\n#lista li\n{\n
      ↪ padding:20px;\n border-bottom: solid 1px #cdcdcd;\n}\n#lista li:last-child\n{\n border-bottom: none;\n}\n#lista div\n{\n
      ↪ display:inline-block;\n vertical-align: middle;\n}\n#lista div:last-child\n{\n text-align:right;\n}\nsvg\n{\n zoom:80%;\n}\n.
      ↪ config\n{\n display:none;\n position:fixed;\n top:0px;\n left:0px;\n width:100%;\n height:100%;\n background-color:rgba
      ↪ (0,0,0,0.8);\n text-align:center;\n}\n#SSID\n{\n background-color:transparent;\n border:0px;\n padding:10px;\n color:#fff;\n
      ↪ text-align:left;\n width:75%;\n}\n#password\n{\n padding:10px;\n width:55%;\n border:1px solid #fff;\n}\nbutton\n{\n
      ↪ background-color:#0076FF;\n text-transform: uppercase;\n color: #fff;\n border: none;\n padding:10px;\n border:1px solid
      ↪ #0076FF;\n width:20%;\n}\n.box\n{\n margin-top:35vh;\n}\n.active\n{\n display:block !important;\n}\n.close\n{\n float:
      ↪ right;\n color:#fff;\n margin-top:20px;\n margin-right:20px;\n}\n.success\n{\n display:none;\n position:fixed;\n top:0px;\n left
      ↪ :0px;\n width:100%;\n height:100%;\n background-color:#fff;\n text-align:center;\n}\n\n.success div\n{\n margin-top:35vh;\n
      ↪ }\n\n.ip-fixo\n{\n visibility:hidden;\n}\n\n.input-data\n{\n padding:10px;\n text-align:left;\n width:55%;\n border:1px solid
      ↪ #fff;\n float:left;\n margin-left:9%;\n}\n\n</style>\n\n<body>\n\n<div class='success' id='success'>\n <div>
      ↪ CONFIGURADO COM SUCESSO<br><br><span style='font-size:0.8em'>Por favor verifique se a luz da WIFI<br>da home
      ↪ base está acesa.</span></div>\n\n <svg version='1.0' id='Layer_1' xmlns='http://www.w3.org/2000/svg' xmlns:xlink='http
      ↪ ://www.w3.org/1999/xlink' x='0px' y='0px'\n width='160px' height='100px' viewBox='0 0 40.354 13.188' enable-background='
      ↪ new 0 0 40.354 13.188' xml:space='preserve'>\n <path fill='#A7A6A6' d='M39.354,9.389c0,1.951-1.96,3.535-4.377,3.535H5.795c
      ↪ -2.417,0-4.377-1.584-4.377-3.535V3.716\n c0-1.953,1.96-3.536,4.377-3.536h29.182c2.417,0,4.377,1.583,4.377,3.536V9.389z'/>\
      ↪ n <g>\n <path fill='#FFFFFF' d='M5.443,4.926h0.906l0.27,1.83C6.647,6.96,6.674,7.165,6.7,7.371s0.053,0.413,0.081,0.621h0.024\n
      ↪  c0.036-0.208,0.073-0.415,0.111-0.621S6.993,6.96,7.033,6.756l0.42-1.83h0.75l0.426,1.83c0.04,0.2,0.079,0.403,0.117,0.609\n C8
      ↪ .784,7.57,8.823,7.779,8.863,7.992h0.024c0.024-0.213,0.049-0.42,0.075-0.625C8.988,7.164,9.015,6.96,9.043,6.756l0.27-1.83\n h0.846
      ↪ L9.463,8.838h-1.11L7.969,7.062C7.937,6.91,7.908,6.757,7.882,6.603C7.856,6.449,7.833,6.3,7.813,6.156H7.789\n C7
      ↪ .765,6.3,7.741,6.449,7.717,6.603C7.693,6.757,7.665,6.91,7.633,7.062L7.261,8.838H6.169L5.443,4.926z'/>\n <path fill='#FFFFFF' d
      ↪ ='M10.693,4.926h0.888v3.912h-0.888V4.926z'/>\n <path fill='#FFFFFF' d='M12.493,4.926h2.472V5.67h-1.584v0.906h1.35V7
      ↪ .32h-1.35v1.518h-0.888V4.926z'/>\n <path fill='#FFFFFF' d='M15.631,4.926h0.888v3.912h-0.888V4.926z'/>\n </g>\n <
      ↪ ellipse fill='#47C6E4' cx='32.444' cy='6.355' rx='4.656' ry='5.166'/>\n </svg>\n</div>\n\n<div class='config' id='config'>\
      ↪ n\n <div class='close' onclick='closeConfig()'>x</div>\n\n <div class='box'>\n <input class='input-data ip-fixo' type='text'
      ↪  id='IP' name='ip' placeholder='IP' value=''><br>\n <input class='input-data ip-fixo' type='text' id='GATEWAY' name='
      ↪ gateway' placeholder='GATEWAY' value=''><br>\n <input class='input-data ip-fixo' type='text' id='SUBNET' name='
      ↪ subnet' placeholder='SUBNET' value=''><br>\n <input type='text' id='SSID' name='' placeholder='' value=''><br>\n <
      ↪ input type='text' id='password' name='' placeholder='Senha' value=''>\n <button type='button' onclick='loadDoc()'>Enviar</
      ↪ button>\n </div>\n <button style=' float: right; margin: 20px 8.55% 0 0;' type='button' onclick='ipFixo()'>IP Fixo</button
      ↪ >\n\n</div>\n\n<div style='color:#777;font-size:0.9em;padding:20px;'>Por favor, selecione sua rede WIFI</div>\n\n<div
      ↪ class='listNetworks active'>\n\n<ul id='lista'></ul>\n\n</div>\n\n<iframe name='reset' style='width:0px;height:0px;border
      ↪ :0px;'></iframe>\n\n<script type='text/javascript'>\n let visible = 0;\nfunction ipFixo()\n{\n let ipElement = document.
      ↪ querySelector('#IP');\n let gatewayElement = document.querySelector('#GATEWAY');\n let subnetElement = document.
      ↪ querySelector('#SUBNET');\n if (visible === 0) {\n ipElement.style.visibility = 'visible';\n gatewayElement.style.visibility = '
      ↪ visible';\n subnetElement.style.visibility = 'visible';\n visible = 1;\n }else {\n ipElement.style.visibility = 'hidden';\n
      ↪ gatewayElement.style.visibility = 'hidden';\n subnetElement.style.visibility = 'hidden';\n visible = 0;\n }\n }\n \nfunction
      ↪ loadDoc()\n{\n let jsonData =[{\n 'SSID': document.getElementById('SSID').value,\n 'password': document.getElementById('
      ↪ password').value,\n 'ip': document.getElementById('IP').value, 'gateway': document.getElementById('GATEWAY').value, subnet:
      ↪ document.getElementById('SUBNET').value }\n ];\n\n let jsonInfo = JSON.stringify(jsonData)\n\n var xhttp = new
      ↪ XMLHttpRequest();\n\n xhttp.onreadystatechange = function() {\n if (this.readyState == 4 && this.status == 200) {\n
```

```
        ↪ closeConfig();\n document.getElementById('success').classList.add('active');\n window.open('/reset', 'reset');\n }\n };\n\n xhttp.
        ↪ open('POST', '/setup', true);\n xhttp.setRequestHeader('Content-type', 'application/json');\n xhttp.send(jsonInfo);\n}\n\
        ↪ nfunction goToForm(index)\n{\n document.getElementById('config').classList.add('active');\n let SSID = index.children[0].
        ↪ innerHTML;\n document.getElementById('SSID').value = SSID\n}\n\nfunction appendReceivedData(gotData)\n{\n let data =
        ↪ JSON.parse(gotData)\n\n for (let i = 0; i < data.length; i++ ) {\n var item = document.createElement('li');\n\n var icon1o = '<
        ↪ svg xmlns=\"http://www.w3.org/2000/svg\" width=\"24\" height=\"24\" viewBox=\"0 0 24 24\"><path fill-opacity=\".3\" d
        ↪ =\"M12.01 21.49L23.64 7c-.45-.34-4.93-4-11.64-4C5.28 3 .81 6.66.36 7l11.63 14.49.01.01.01-.01z\"/><path d=\"M0 0
        ↪ h24v24H0z\" fill=\"none\"/><path d=\"M6.67 14.86L12 21.49v.01l.01-.01 5.33-6.63C17.06 14.65 15.03 13 12 13s-5.06 1.65-5.33
        ↪  1.86z\"/></svg>';\n var icon1c = '<svg xmlns=\"http://www.w3.org/2000/svg\" width=\"24\" height=\"24\" viewBox=\"0 0
        ↪ 24 24\"><path fill=\"none\" d=\"M0 0h24v24H0V0z\"/><path d=\"M23 16v-1.5c0-1.4-1.1-2.5-2.5-2.5S18 13.1 18 14.5V16c
        ↪ -.5 0-1 .5-1 1v4c0 .5.5 1 1 1h5c.5 0 1-.5 1-1v-4c0-.5-.5-1-1-1zm-1 0h-3v-1.5c0-.8.7-1.5 1.5-1.5s1.5.7 1.5 1.5V16z
        ↪ \"/><path d=\"M15.5 14.5c0-2.8 2.2-5 5-5 .4 0 .7 0 1 .1L23.6 7c-.4-.3-4.9-4-11.6-4C5.3 3 .8 6.7.4 7L12 21.5l3.5-4.3v-2.7z
        ↪ \" opacity=\".3\"/><path d=\"M6.7 14.9l5.3 6.6 3.5-4.3v-2.6c0-.2 0-.5.1-.7-.9-.5-2.2-.9-3.6-.9-3 0-5.1 1.7-5.3 1.9z
        ↪ \"/></svg>';\n\n var icon2o = '<svg xmlns=\"http://www.w3.org/2000/svg\" width=\"24\" height=\"24\" viewBox=\"0 0 24
        ↪ 24\"><path fill-opacity=\".3\" d=\"M12.01 21.49L23.64 7c-.45-.34-4.93-4-11.64-4C5.28 3 .81 6.66.36 7l11.63
        ↪ 14.49.01.01.01-.01z\"/><path d=\"M0 0h24v24H0z\" fill=\"none\"/><path d=\"M4.79 12.52l7.2 8.98H12l.01-.01 7.2-8.98C18
        ↪ .85 12.24 16.1 10 12 10s-6.85 2.24-7.21 2.52z\"/></svg>';\n var icon2c = '<svg xmlns=\"http://www.w3.org/2000/svg\" width
        ↪ =\"24\" height=\"24\" viewBox=\"0 0 24 24\"><path fill=\"none\" d=\"M0 0h24v24H0V0z\"/><path d=\"M23 16v-1.5c0
        ↪ -1.4-1.1-2.5-2.5-2.5S18 13.1 18 14.5V16c-.5 0-1 .5-1 1v4c0 .5.5 1 1 1h5c.5 0 1-.5 1-1v-4c0-.5-.5-1-1-1zm-1 0h-3v-1.5
        ↪ c0-.8.7-1.5 1.5-1.5s1.5.7 1.5 1.5V16z\"/><path d=\"M15.5 14.5c0-2.8 2.2-5 5-5 .4 0 .7 0 1 .1L23.6 7c-.4-.3-4.9-4-11.6-4
        ↪ C5.3 3 .8 6.7.4 7L12 21.5l3.5-4.3v-2.7z\" opacity=\".3\"/><path d=\"M4.8 12.5l7.2 9 3.5-4.4v-2.6c0-1.3.5-2.5 1.4-3.4C15.6
        ↪ 10.5 14 10 12 10c-4.1 0-6.8 2.2-7.2 2.5z\"/></svg>';\n \n var icon3o = '<svg xmlns=\"http://www.w3.org/2000/svg\" width
        ↪ =\"24\" height=\"24\" viewBox=\"0 0 24 24\"><path fill-opacity=\".3\" d=\"M12.01 21.49L23.64 7c-.45-.34-4.93-4-11.64-4
        ↪ C5.28 3 .81 6.66.36 7l11.63 14.49.01.01.01-.01z\"/><path d=\"M0 0h24v24H0z\" fill=\"none\"/><path d=\"M3.53 10.95l8.46
        ↪ 10.54.01.01.01-.01 8.46-10.54C20.04 10.62 16.81 8 12 8c-4.81 0-8.04 2.62-8.47 2.95z\"/></svg>';\n var icon3c = '<svg xmlns
        ↪ =\"http://www.w3.org/2000/svg\" width=\"24\" height=\"24\" viewBox=\"0 0 24 24\"><path opacity=\".3\" d=\"M12 3C5.3 3
        ↪  .8 6.7.4 7l3.2 3.9L12 21.5l3.5-4.3v-2.6c0-2.2 1.4-4 3.3-4.7.3-.1.5-.2.8-.2.3-.1.6-.1.9-.1.4 0 .7 0 1 .1L23.6 7c
        ↪ -.4-.3-4.9-4-11.6-4z\"/><path fill=\"none\" d=\"M0 0h24v24H0V0z\"/><path d=\"M23 16v-1.5c0-1.4-1.1-2.5-2.5-2.5S18
        ↪ S18 13.1 18 14.5V16c-.5 0-1 .5-1 1v4c0 .5.5 1 1 1h5c.5 0 1-.5 1-1v-4c0-.5-.5-1-1-1zm-1 0h-3v-1.5c0-.8.7-1.5 1.5-1.5s1
        ↪ .5.7 1.5 1.5V16zm-10 5.5l3.5-4.3v-2.6c0-2.2 1.4-4 3.3-4.7C17.3 9 14.9 8 12 8c-4.8 0-8 2.6-8.5 2.9\"/></svg>';\n \n var
        ↪ icon4o = '<svg xmlns=\"http://www.w3.org/2000/svg\" width=\"24\" height=\"24\" viewBox=\"0 0 24 24\"><path d=\"M12
        ↪ .01 21.49L23.64 7c-.45-.34-4.93-4-11.64-4C5.28 3 .81 6.66.36 7l11.63 14.49.01.01.01-.01z\"/><path d=\"M0 0h24v24H0z\" fill
        ↪ =\"none\"/></svg>';\n var icon4c = '<svg xmlns=\"http://www.w3.org/2000/svg\" width=\"24\" height=\"24\" viewBox=\"0
        ↪ 0 24 24\"><path fill=\"none\" d=\"M0 0h24v24H0V0z\"/><path d=\"M23 16v-1.5c0-1.4-1.1-2.5-2.5-2.5S18 13.1 18 14.5
        ↪ V16c-.5 0-1 .5-1 1v4c0 .5.5 1 1 1h5c.5 0 1-.5 1-1v-4c0-.5-.5-1-1-1zm-1 0h-3v-1.5c0-.8.7-1.5 1.5-1.5s1.5.7 1.5 1.5
        ↪ V16zm-6.5-1.5c0-2.8 2.2-5 5-5 .4 0 .7 0 1 .1L23.6 7c-.4-.3-4.9-4-11.6-4C5.3 3 .8 6.7.4 7L12 21.5l3.5-4.3v-2.6z\"/></svg
        ↪ >';\n\n var sinal = data[i].RSSI;\n\n if (sinal > -80) {\n // Level 4\n if (data[i].Encryption == '?') {\n icon = icon4o;\n } else
        ↪ {\n icon = icon4c;\n }\n } else if (sinal > -85) {\n // Level 3\n if (data[i].Encryption == '?') {\n icon = icon3o;\n } else {\n
        ↪ icon = icon3c;\n }\n } else if (sinal > -90) {\n // Level 2\n if (data[i].Encryption == '?') {\n icon = icon2o;\n } else {\n icon
        ↪ = icon2c;\n }\n } else {\n // Level 1\n if (data[i].Encryption == '?') {\n icon = icon1o;\n } else {\n icon = icon1c;\n }\n }\n\n
        ↪  item.innerHTML = '<div style=\"width:70%\">' + data[i].SSID + '</div><div style=\"width:30%\">' + icon + '</div>';\n
        ↪ item.setAttribute('onclick', 'goToForm(this)');\n document.getElementById('lista').appendChild(item);\n }\n}\n\nfunction
        ↪ httpGetAsync(theUrl, callback)\n{\n var xmlHttp = new XMLHttpRequest();\n\n xmlHttp.onreadystatechange = function() {\n
        ↪ if (xmlHttp.readyState == 4 && xmlHttp.status == 200)\n callback(xmlHttp.responseText);\n }\n\n xmlHttp.open('GET',
        ↪ theUrl, true);\n xmlHttp.send(null);\n}\n\nfunction closeConfig()\n{\n document.getElementById('config').classList.remove('
        ↪ active');\n}\n\nhttpGetAsync('list', appendReceivedData);\n\ndocument.onkeydown = function (e) {\n e = e || window.event;\n
        ↪ if (e.keyCode == 27) {\n closeConfig();\n }\n};\n\n</script>\n</body>\n</html>\n\n";

  delay(10000);

  // Leitura da configuração
  read_flash();

  // Processos do WiFi
  start_WiFi();

  randomSeed(esp_random());

  //Se nao houver Wifi cadastrado disponivel, configurar
  if (toConfig) {
    Serial.println("ERRO ao conectar ao WiFi...\nIniciando servidor para configuração");
    start_config_server(ssid, pass, configPage, Domain);
  }

  //Configura Watchdog
  configureWatchdog();


}


void loop() {

    //reseta o temporizador (alimenta o watchdog)
    timerWrite(timer, 0);
    checar_wifi();
    _receive();


}
```

Abaixo se encontra o código de automação desenvolvido para controle de temperatura.

SensorTemperatura.ino

```
#include <OnAirProtocol.h>
#include <LowPower.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <EEPROM.h>


#define alarmPin 3 // RTC INT
#define ledR A2 // LED
#define ONE_WIRE_BUS 4 // SENSOR
#define batteryPin A7 //DIVISOR TENSÃO DA BATERIA
#define SEND_DATA 1
#define SEND_PRIORITY 2
#define SEND_TEST 3

String payload = "";
char lowpower = 1;
bool criptografia = false;
int tempo = 20, tipo;

//Globais para LORA
char _appId[5] = { 0x00, 0x00, 0x00, 0x00, 0x01 };
char _devId[5] = { 0x00, 0x00, 0x00, 0x00, 0x01 };

// Setup a oneWire instance to communicate with any OneWire devices (not just Maxim/Dallas temperature ICs)
OneWire oneWire(ONE_WIRE_BUS);

// Pass our oneWire reference to Dallas Temperature.
DallasTemperature sensors(&oneWire);

void setup() {
  // ************CONFIGURAÇÃO GPIO************//

  pinMode(alarmPin, INPUT_PULLUP);
  pinMode(ledR, OUTPUT);
  pinMode(ONE_WIRE_BUS, INPUT);
  pinMode(batteryPin, INPUT);
  pinMode(5, OUTPUT);

  digitalWrite(ledR, LOW);
  digitalWrite(5, LOW);

  // ****************************************//

  Serial.begin(9600);

  sensors.begin();

  delay(1000);


  // ************SINALIZAÇÃO INICIAL************//

  Serial.println (F("Inicializado..."));
  Serial.println(F(" "));

  Serial.print(F("App_Id: "));
  for (int i = 0; i < 5; i++) {
    Serial.print(F(" "));
    Serial.print((unsigned char)_appId[i], HEX);
  }
  Serial.println(F(" "));

  Serial.print(F("Dev_Id: "));
  for (int i = 0; i < 5; i++) {
    Serial.print(F(" "));
    Serial.print((unsigned char)_devId[i], HEX);
  }
  Serial.println(F(" "));
  Serial.println(F(" "));

  digitalWrite(ledR, HIGH);
  delay(1000);
  digitalWrite (ledR, LOW);
  delay(1000);

  alarm = true;
```

```
78
79    //***********INIT LORA*************************************************************************************
80
81    start_LoRa();
82
83    //*****************************//
84    randomSeed(analogRead(6));
85
86  }
87
88  //*********************************************************************************************************
89
90  int medeBateria (void) {
91    int passos;
92    float tensao;
93    int nivelbateria = 0;
94
95    // * Nova funcao de bateria
96    tensao = passos * (3.3 / 1024);
97
98    nivelbateria = (2 * tensao - 2.8) / 0.00784;
99
100   if (nivelbateria < 0) nivelbateria = 0;
101
102   if (nivelbateria > 255) nivelbateria = 255;
103
104
105   return nivelbateria;
106
107 }
108
109 //*********************************************************************************************************
110
111
112 void interrupcao() {
113
114   delay(1);
115
116 }
117
118
119 //*********************************************************************************************************
120
121 void loop() {
122
123   wdt_enable(WDTO_8S);
124
125   // Declarações temperatura
126   unsigned char inteiroHEX[2];
127   unsigned char decimalHEX[1];
128   unsigned char temperaturaHEX[3];
129   unsigned char releHEX[2];
130   float temperatura;
131   float temperatura_armazenar = -56;
132
133   // Declarações bateria
134   int bateria_armazenar = 0;
135   unsigned char bateriaHEX[2];
136
137   // Declaração arrays
138   unsigned char enviar_HEX[6];
139
140   // Limpa Payload
141   payload = "";
142
143   // ******************ATIVANDO LOW POWER*****************************************************
144   tmElements_t tmSet;
145
146   tmSet.Year = 2020;
147   tmSet.Month = 9;
148   tmSet.Day = 2;
149   tmSet.Hour = 10;
150   tmSet.Minute = 0;
151   tmSet.Second = 0;
152
153   RTC.set(makeTime(tmSet), CLOCK_ADDRESS); // set the clock
154
155   tmSet.Second = tempo;
156
157   RTC.set(makeTime(tmSet), ALARM1_ADDRESS); // set the alarm for 4 seconds later
158   RTC.enableAlarm(ALARM1_ADDRESS);
```

```
159   RTC.freqSelect(1); // set the squarewave freq on alarm pin b to 4.096kHz
160
161   Serial.println (F("***Sleep"));
162   digitalWrite(5, HIGH);
163   delay(50);
164
165   if (lowpower != 0) {
166     LoRa.sleep();
167     attachInterrupt(1, interrupcao, LOW);
168     wdt_disable();
169     LowPower.powerDown(SLEEP_FOREVER, ADC_OFF, BOD_OFF);
170     detachInterrupt(1);
171   }
172
173   wdt_enable(WDTO_8S);
174
175   // ******************WAKE UP**************************************
176
177   if (digitalRead(alarmPin) == HIGH) { //pin is set low when alarm is triggered
178     Serial.println(F("ALARM off"));
179   } else {
180     Serial.println(F(""));
181     Serial.println(F("***Wake up"));
182
183     RTC.resetAlarms();
184
185   }
186   digitalWrite(5, LOW);
187   Serial.println (F(""));
188   Serial.println (F("***Collecting data"));
189
190   digitalWrite(ledR, HIGH);
191   delay(10);
192   digitalWrite (ledR, LOW);
193
194   wdt_reset();
195
196   // ******************RTC**************************************
197
198   time_t clock = RTC.get(CLOCK_ADDRESS);
199   time_t alarm = RTC.get(ALARM1_ADDRESS); // get the time the alarm is set for
200   tmElements_t clockSet;
201   tmElements_t alarmSet;
202   breakTime(clock, clockSet);
203   breakTime(alarm, alarmSet);
204
205   wdt_reset();
206
207   // ******************TEMPERATURA SENSOR SONDA**************************************
208
209   // Comando para pegar as temperaturas
210   sensors.requestTemperatures();
211
212   // Armazena a temperatura em uma variável
213   temperatura = sensors.getTempCByIndex(0);
214
215   if (temperatura > temperatura_armazenar) {
216     temperatura_armazenar = temperatura; //Armazena a maior temperatura
217   }
218
219   // Faz a separação de inteiro e decimal
220   int temperatura_inteiro = temperatura_armazenar;
221   int temperatura_decimal = abs((temperatura_armazenar - temperatura_inteiro) * 10);
222
223   // Deixa os dados da temperatura inteira de acordo com a tabela de conversão
224   temperatura_inteiro = map(temperatura_inteiro, -55, 125, 0, 180);
225
226   // Transforma os dados de temperatura em hex
227   tipo = 2;
228
229   converter_int_hex(temperatura_inteiro, inteiroHEX, tipo);
230
231   tipo = 3;
232   converter_int_hex(temperatura_decimal, decimalHEX, tipo);
233
234   //concatenando duas strings
235   temperaturaHEX[0] = inteiroHEX[0];
236   temperaturaHEX[1] = inteiroHEX[1];
237   temperaturaHEX[2] = decimalHEX[0];
238
239
```

```
240   wdt_reset();
241
242
243   // ******************BATERIA*****************************************************************
244
245   bateria_armazenar = medeBateria();
246
247   tipo = 2;
248   converter_int_hex(bateria_armazenar, bateriaHEX, tipo);
249
250   wdt_reset();
251
252   // ******************RELE*******************************************************************
253
254   for(int i = 0; i < tam_recv; i++) {
255     Serial.print(received[i]);
256   }
257   Serial.println();
258   if(1 == 1) {
259     releHEX[0] = '1';
260   }
261   else {
262     releHEX[0] = '0';
263   }
264
265
266   //*****************************************************************************************
267
268   // ******************MONTANDO STRING PARA ENVIO*****************************************
269
270   // Armazena no array as medidas
271
272   enviar_HEX[0] = temperaturaHEX[0];
273   enviar_HEX[1] = temperaturaHEX[1];
274   enviar_HEX[2] = temperaturaHEX[2];
275   enviar_HEX[3] = releHEX[0];
276   enviar_HEX[4] = bateriaHEX[0];
277   enviar_HEX[5] = bateriaHEX[1];
278
279
280   Serial.println ("***Data acquired");
281
282   payload.reserve(6);
283   payload = "1";
284   for (int i = 0; i < 5; i++) { //E necessario inicializar a String, senao setCharAt nao funciona
285     payload += "1";
286   }
287   for (int j = 0; j < 6; j++) {
288     payload.setCharAt(j, enviar_HEX[j]);
289   }
290   payload.toUpperCase();
291   //Serial.println(payload);
292
293   wdt_reset();
294   sendMessage(SEND_DATA);
295   wdt_reset();
296   payload = "";
297
298 }
```