UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

JULIO OPOLSKI NETTO

IMPLEMENTAÇÃO DE UMA REDE PARA COMUNICAÇÃO ENTRE ESTAÇÕES BASE PARA GRUPOS DE DRONES

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2019

JULIO OPOLSKI NETTO

IMPLEMENTAÇÃO DE UMA REDE PARA COMUNICAÇÃO ENTRE ESTAÇÕES BASE PARA GRUPOS DE DRONES

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso de Engenharia de Computação - da Universidade Tecnológica Federal do Paraná - UTFPR - Câmpus Pato Branco, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Fábio Favarim

PATO BRANCO
2019

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

TERMO DE APROVAÇÃO

Às 10 horas do dia 06 de dezembro de 2019, na sala V109, da Universidade Tecnológica Federal do Paraná, *Campus* Pato Branco, reuniu-se a banca examinadora composta pelos professores Fábio Favarim (orientador), Adão Robson Elias e Mainara Cristina Lorencena para avaliar o trabalho de conclusão de curso com o título **Implementação de uma Rede para Comunicação entre Estações Base para Grupos de Drones**, do aluno **Julio Opolski Netto**, matrícula 01422588, do curso de Engenharia de Computação. Após apresentação o aluno foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

_____
Prof. Fábio Favarim
Orientador (UTFPR)

_____
Prof. Adão Robson Elias
(UTFPR)

_____
Profa. Mainara Cristina Lorencena
(UTFPR)

_____
Prof. Marco Antonio de Castro Barbosa
Coordenador de TCC

_____
Prof. Pablo Gauterio Cavalcanti
Coordenador do Curso de
Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

Minha família, meus pais Julio *(in memorian)* e Heriberto, mães Silomar e Sirlei por toda a paciência e ajuda. Ao meu irmão Wagner por toda ajuda, vó Ita, vó Nilza *(in memorian)*, vó Elva e vô Julio. À minha noiva Amanda por toda a paciência e força nos momentos mais difíceis, meus pais de coração José e Mara e à espoleta *(in memorian)* pela parceria nos estudos. Todo o meu esforço e dedicação é por vocês. Amo todos vocês.

AGRADECIMENTOS

O meu agradecimento à todos que de alguma forma foram fundamentais na minha formação como estudante, e como pessoa.

À Universidade Tecnológica Federal do Paraná (UTFPR) e Universidade Federal do Rio Grande do Norte (UFRN). Todos os professores que passaram pela minha formação, em especial meu orientador Profº Dr. Fábio Favarim, Professoras Beatriz Borsoi e Mainara Lorencena, Professores Robison Cris Brito e Adão Robson Elias, por toda a ajuda.

À todos os meus amigos e colegas que de alguma forma contribuíram, em especial ao Vinicius Tártari por toda a força, dicas e ajuda.

À minha família, meus pais Julio Opolski Junior *(in memorian)* e Heriberto de Marco, mães Silomar do Carmo Bedenaski e Sirlei A. Bedenaski pelo apoio incondicional, vó Ita, vó Nilza *(in memorian)*, vó Elva e vô Julio. Ao meu irmão Wagner Opolski, por toda a ajuda na monografia, apresentação e testes.

À minha base, minha noiva Amanda Pacce, por todo o apoio, paciência e ajuda. Meus pais do coração José Pacce e Mara Pacce, por todo o apoio e carinho. E minha parceira de estudo espoleta *(in memorian)*, obrigado por tudo.

"Todos neste mundo trabalham com base no seu fuso horário.
As pessoas ao seu redor podem parecer estar à sua frente,
Alguns podem parecer estar atrás de você.
Mas todos estão executando sua própria corrida, em seu próprio tempo, não os inveje e não os zombe.
Eles estão no fuso horário deles e você está no seu.
A vida se resume em esperar o momento certo para agir.
Então, relaxe:
Você não está adiantado.
Você não está atrasado.
Você está no tempo certo.
Você está no seu tempo.".
(Autor desconhecido)

RESUMO

A utilização de Veículos Aéreos Não Tripulados (VANTs) tem se mostrado cada vez mais frequente para diversas aplicações, principalmente na agricultura. O mapeamento de grandes áreas para fins de análise são comuns e é considerado um desafio devido ao curto alcance dos VANTs. A utilização de estações base para recarregamento de drones e obtenção de informações relevantes é uma proposta pertinente.

Este trabalho apresenta um sistema de comunicação de longo alcance e baixo consumo energético entre estações base. Utilizando conceitos de IoT e a possibilidade de utilizar diversos protocolos de comunicação em apenas um único dispositivo, este trabalho mostra a integração entre microcontrolador, servidor e a interface do operador. O sistema desenvolvido é capaz de identificar o drone que pousou na estação base através da tecnologia RFID, e enviar essa e outras informações em tempo real através do *gateway* para o servidor utilizando a tecnologia LoRa e o protocol MQTT.

Palavras-chaves: IoT. Drone. Mapeamento. MQTT. LoRa.

ABSTRACT

The use of Unmanned Aerial Vehicle (UAVs) are widely used for many applications, especially in agriculture. Mapping large areas for analysis is common and is a challenge because the UAVs short range. The use of base stations for recharge UAVs and collect relevant data is a pertinent proposal.

This study presents a long-range and low power consumption communication system between base stations. Using Internet of Things concepts and the possibility to use multiple communication protocols in only one device allows to integrate microcontroller, server and user interface. A system was developed, which was capable to identify a drone (using RFID technology) that landed in base station and sent in real time the informations through the gateway to the server using LoRa technology and MQTT protocol.

Key-words: IoT. Drone. Mapping Area. MQTT. LoRa.

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
AMQP	Advanced Message Queuing Protocol
ANATEL	Agência Nacional de Telecomunicações
AP	Agricultura de Precisão
API	Application Programming Interface
BD	Banco de Dados
BLE	Bluetooth Low Energy
CSS	Chirp Spread Spectrum Modulation
CTI	Ciência, Tecnologia e Inovação
DECOM	Departamento de Computação
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identification
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IoT	Internet of Things
ISM	Industrial Sientific and Medical
JMS	Java Message Service
JSON	JavaScript Object Notation
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LoS	Line of Sight
LPWAN	Low Power Wide Area Network
LR-WPAN	Low Rate - Wireless Personal Area Network
LTE	Long Term Evolution

MAC	Media Access Control
MDE	Modelo Digital de Elevação
MIME	Multipurpose Internet Mail Extensions
MQTT	Message Queue Telemetry Transport
NFC	Near Field Communication
NLoS	Non-Line of Sight
NoSQL	Not Only SQL
PAN	Personal Area Network
REST	Representational State Transfer
RF	Rádiofrequência
RFID	Radio-Frequency Identification
RSSI	Received Signal Strength Indication
SF	Spreading Factor
SPI	Serial Peripheral Interface
SQL	Structured Query Language
UHF	Ultra High Frequency
URL	Uniform Resource Locator
VANT	Veículos Aéreos Não Tripulados
VHF	Very High Frequency
VSAT	Very Small Aperture Terminal
Wi-Fi	Wireless Fidelity
XML	Extensible Markup Language

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

A disponibilidade de recursos naturais, a competência dos agricultores e a organização das cadeias produtivas, juntamente com a Ciência, Tecnologia e Inovação (CT&I), têm contribuído significativamente para o desenvolvimento econômico do Brasil, tornando-o um dos maiores produtores e exportadores de produtos agrícolas nas últimas décadas (ESTRATÉGIA..., 2016).

A utilização de novas tecnologias nos setores primários da economia, principalmente na agricultura, tem auxiliado na coleta de dados das áreas produtivas. Para assegurar o sucesso da produção e evitar diversos problemas enfrentados no plantio como pragas, baixa produção, controle de nutrientes e acidez do solo, a obtenção de informações sobre os fatores que interagem na lavoura e as informações necessárias para maximizar a produção, são de extrema relevância. A união da tecnologia com as práticas agrícolas é denominada Agricultura de Precisão (AP) e ela tem o objetivo de auxiliar na obtenção de maior controle e de melhores resultados da produção.

Tecnologias que utilizem Veículos Aéreos Não Tripulados (VANT), dispositivos de Radiofrequência (RF), aplicações utilizando Internet das Coisas, ou *Internet of Things* (IoT) e máquinas agrícolas podem ser citados como alguns dos principais responsáveis por essa transformação. Segundo (KITE-POWELL, 2018), até 2020 o mercado de AP tem a expectativa de crescer de $730 milhões para $2.4 bilhões de dólares.

Autores como (GEORGE et al., 2013) citam a importância dos veículos aéreos não tripulados na agricultura de precisão. Com rapidez e eficiência, os VANTs são capazes de efetuar as mais variadas tarefas em praticamente qualquer tipo de terreno, como o mapeamento de áreas para obtenção de dados como o Modelo Digital de Elevação (MDE), a análise, o monitoramento automático de solo e de plantios além do controle de pragas.

Uma evolução para a utilização de VANTs no meio agrícola é a prática de voos autônomos. Essa prática abre possibilidade para a inspeção e o monitoramento de grandes áreas de plantio, sem a necessidade de um operador em solo para cada VANT. Porém, o uso de VANTs para grandes áreas ainda apresenta algumas limitações como o consumo de bateria, afetando diretamente o tamanho da área a ser analisada (BRITO, 2018).

Uma possível solução para a abrangência de uma grande área segundo (BRITO, 2018), é a utilização de um grupo de VANTs ou drones, mais especificamente, e estações base sincronizadas, garantindo rapidez e gerando redundância para o caso de algum drone deixe de efetuar a tarefa por algum problema, imediatamente a tarefa é atribuída para outro drone. Essas estações base teriam a capacidade de recarregar os drones, coletar informações do drone e do solo e transmitir novas missões/tarefas aos drones. Assim, a integração das estações base

com os drones, aumenta consideravelmente a quantidade de dados obtidos e de área analisada.

Uma grande quantidade de dispositivos conectados e trocando informações entre si exige a implementação de uma rede de comunicação tolarante a falhas e de alta confiabilidade, para que a troca de dados entre eles ocorra da maneira planejada. Porém, devido à distância, nem sempre uma estação base possui comunicação com a base central (*gateway*). Uma forma de tratar esse problema é por meio do uso de uma rede dinâmica que se adapta automaticamente de modo a manter a conectividade.

1.2 OBJETIVOS

1.2.1 OBJETIVO GERAL

Desenvolver um sistema de comunicação baseado em tecnologias e protocolos de comunicação que permitam a integração entre estações base, gateway e servidor à longas distâncias.

1.2.2 OBJETIVO ESPECÍFICO

- Realizar a comunicação entre a estação base e o gateway utilizando a tecnologia de comunicação LoRa;
- Propor uma arquitetura para comunicação entre os componentes do sistema;
- Implementar a comunicação do *gateway* com um servidor ou *broker* utilizando o protocolo MQTT (Seção 2.5.2);
- Desenvolver um sistema para receber os dados, interpretá-los e apresentá-los por meio de uma interface Web.

1.3 JUSTIFICATIVA

A motivação para desenvolver este trabalho surgiu na qualificação de doutorado de (BRITO, 2018), onde o mesmo propõe a utilização de estações base espalhadas em uma área afim de possibilitar a recarga, a obtenção de informações do local e a transmissão de novas missões à drones que estão fazendo alguma atividade de monitoramento ou mapeamento.

Este trabalho apresenta uma proposta de comunicação que permita a utilização de múltiplas bases, que se comportem de forma dinâmica para a transmissão de dados para um *gateway*. Este *gateway* deve enviar informações para um servidor ou *broker* e através de uma interface web um usuário poder visualizar os dados.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está divido em 5 capítulos. O Capítulo 1 é a Introdução que contém o problema, os objetivos a serem alcançados e a justificativa. O Capítulo 2 apresenta uma breve introdução sobre VANTs, Mapeamento de áreas, IoT, suas tecnologias e protocolos. Na seção

sobre IoT são expostos os conceitos e sua arquitetura juntamente com protocolos e tecnologias de comunicação mais utilizados. No Capítulo 3 são apresentados os materiais e os métodos utilizados no desenvolvimento do trabalho. O Capítulo 4 é referente à todo o desenvolvimento do trabalho, desde o escopo de todos os itens do projeto, testes, modelagem, implementação e apresentação. E finalmente o Capítulo 5 é a conclusão do trabalho, mostrando possíveis trabalhos futuros que poderão ser aproveitados com este projeto.

No final é disponibilizado os códigos que foram utilizados no trabalho.

2 REFERENCIAL TEÓRICO

A fundamentação conceitual deste trabalho é apresentada neste Capítulo e está centrada em Drones (Seção 2.1), Mapeamento ou Monitoramento de áreas (Seção 2.2), IoT (Seção 2.3), Tecnologias (Seção 2.4) e Protocolos de Comunicação (Seção 2.5).

2.1 DRONES

Os primeiros VANTs foram utilizados com propósitos militares tanto para vigilância, como para combate. Os VANTs possuem diversas categorias. A proposta deste trabalho é baseada na utilização de veículos aéreos multirotores, mais conhecidos como drones.

A partir dos anos 90, com o avanço da tecnologia que levou ao desenvolvimento de materiais leves e resistentes e o baixo custo dos drones permitiram a sua disseminação e aplicação em diversas áreas como agricultura, vigilância e monitoramento, mapeamento, reconhecimento, transporte e salvamento, entre outros (CELTEK; DURDU; KURNAZ, 2018).

Geralmente os mais utilizados, os drones com quatro rotores *(Quadcopter)* possuem ótima estabilidade, facilidade de controle e baixo custo. Podendo ser equipados com câmeras de alta qualidade, giroscópio, sensores e processadores para controle de voo e processamento de imagens. Todas essas características aliadas ao baixo custo, permitem a utilização dos drones para diversas tarefas.

A Tabela 1, (BRITO, 2018) apresenta algumas características e cita algumas vantagens da utilização de drones ao invés de VANTs de asa fixa.

- Precisão na leitura dos dados;
- Facilidade de manobra;
- Não necessita da iteração humana, tanto na decolagem como na aterrissagem;
- Custo.

Tabela 1 – Comparação entre as características de um DJI Phantom 4 e AeroVironment Quantix

	DJI Phantom 4	**AeroVironment Quantix**	**Drone x Asa Fixa (%)**
Vel. Horiz.	20 m/s (80 km/h)	15 m/s (57 km/h)	25% mais rápido
Autonomia	25 a 30 min	45 a 55 min	45% menor
Recarga	60 min	75 min	20% mais rápido
Custo	$ 1200	$ 12000	90% mais barato

Fonte: (BRITO, 2018).

2.2 MAPEAMENTO OU ANÁLISE DE ÁREA UTILIZANDO DRONES

O mapeamento de áreas utilizando drones tem sido Utilizado tanto na área militar como por empresas privadas. Na Agricultura de Precisão (AP) eles podem ser utilizados

para diversas finalidades como realização de análise de solo, identificação de áreas férteis e desmatadas e mapeamento geográfico. A expectativa é que o mercado de drones na AP deverá movimentar em 2022, cerca de U$ 3.6 bilhões (MARKETWATCH, 2016)

Para o mapeamento sistemático de áreas, tarefa bastante utilizada em AP, o uso de VANTs de asa fixa era visto como uma vantagem, por possuir maior autonomia pela capacidade de carregar mais baterias. Porém, a utilização de VANTs de asa fixa não dá a possibilidade de recarregar a bateria de forma autônoma.

Conforme (Brito et al., 2019) mesmo sendo eficiente para análise, o mapeamento e o monitoramento de áreas utilizando um único drone não é suficiente para abranger uma grande área em tempo hábil, devido principalmente a autonomia de bateria.

Uma forma de contornar essa limitação é proposta por (BRITO, 2018), com o desenvolvimento de um sistema composto por diversos drones, com cada drone recebendo uma missão específica. E assim que o drone necessitasse recarregar a bateria, ele retornaria para uma estação base de forma autônoma, estando apto a receber novas missões e trocar informações com a estação base.

2.3 IoT

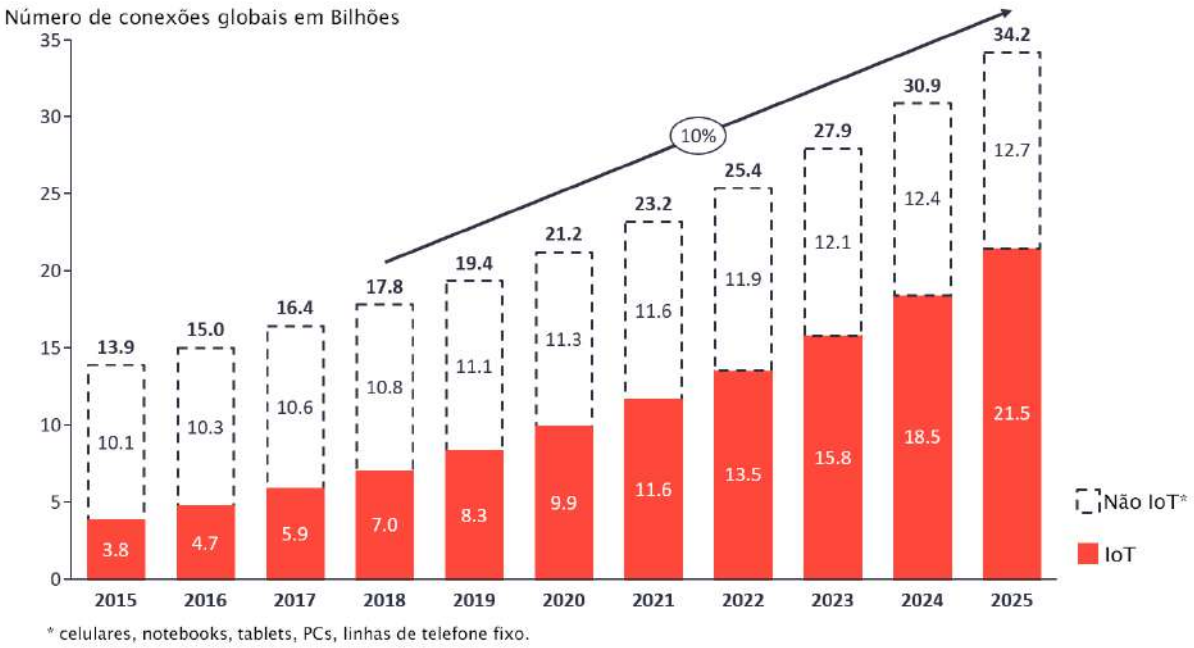
A combinação da Internet com tecnologias emergentes como localização em tempo-real, sensores embarcados e comunicações por campo de proximidade ou *Near Field Communication* (NFC), é capaz de transformar qualquer objeto em um objeto inteligente, com capacidade de obter informações do seu ambiente e reagir de acordo com essas informações (Kortuem et al., 2010).

Internet of Things (IoT), ou Internet das Coisas, é uma expressão utilizada por diversas fontes para descrever a comunicação entre objetos inteligentes (Marginean; Tran; Karzel, 2016), sendo *"coisa"* ou *objeto inteligente* um objeto do dia-a-dia, podendo ser uma geladeira, um micro-ondas, uma tv, um carro, ou ser abstraído como uma casa ou cidade.

Com grande potencial em termos de eficiência energética, sustentabilidade e segurança para a indústria e a sociedade, a IoT é baseada na ideia de que tudo se conecta entre si e pode comunicar-se. Trazendo uma gama de recursos para diversas áreas como automação residencial, permitindo o usuário acionar lâmpadas e eletrodomésticos, para controle de estoque, monitoramento e sensoriamento, oferecendo a capacidade de interagir com objetos que podem ser controlados remotamente de acordo com necessidades ou interesses do usuário.

A crescente demanda por conexão sem fio entre os bilhões de dispositivos existentes tem sido guiada pela IoT. De acordo com a Figura 1, em 2019 o mercado de dispositivos IoT está abastecido com cerca de 8.3 bilhões desses dispositivos e a previsão é de que até 2025, mais de 21.5 bilhões de dispositivos IoT estarão no mercado (LUETH, 2018).

Figura 1 – Número total de dispositivos IoT conectados no mundo



Traduzido de: Lueth (2018).

O crescimento expressivo de dispositivos IoT no mercado vem de diversos setores da indústria. Não somente grandes empresas como Google, Microsoft e Apple, mas empresas do ramo automotivo, eletrodomésticos e produtores de periféricos. Portanto, prover um mecanismo de acesso unificado para IoT é fundamental.

A seguir são apresentadas tecnologias que são utilizadas em sistemas IoT e também neste trabalho, que possibilitam a comunicação, a padronização e a interoperabilidade (capacidade de comunicar-se de forma transparente) entre todos esses dispositivos.

2.4 TECNOLOGIAS DE COMUNICAÇÃO

Levando em consideração que esse trabalho é focado em estações base que serão distribuídas em uma área com distâncias consideráveis, a comunicação por meios cabeados está descartada.

Nos últimos anos, comunicações por meio de redes sem fio estão cada vez mais comuns devido a sua conveniência, mobilidade e facilidade de acesso, principalmente se tratando de IoT. Uma rede sem fio permite a transmissão de dados por radiofrequência, não sendo necessária a utilização de cabeamento.

Redes sem fio são classificadas de acordo com a sua frequência, área de cobertura e taxa de transmissão (IEEE. . . , 2016). A Tabela 2 mostra alguns padrões estabelecidos pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) para redes sem fio.

Tabela 2 – Padrões IEEE para redes sem fio

Padrão IEEE	Frequência	Alcance	Taxa de Transferência
802.11 (Wi-Fi)	2.4 e 5 GHz	até 100m	até 1300 Mbps
802.15.1 (Bluetooth)	2.4 GHz	até 40m	até 50 Mbps
802.15.4 (LR-WPAN)	868 MHz, 915 MHz, 2.4 GHz	até 100m	até 250 Kbps
802.15.4g (LPWAN)	868 MHz, 915 MHz, 2.4 GHz	até 50km	até 1000 Kbps

Adaptado de: IEEE... (2016).

No contexto desse trabalho, considerando o cenário de Agricultura de Precisão (AP) onde as estações base poderão estar em terrenos irregulares, distantes umas das outras e sem a possibilidade de utilizar energia elétrica, as características desejáveis para atender as necessidades da aplicação é de um padrão de comunicação de longo alcance, baixo consumo energético e taxa de transferência razoável.

Nos tópicos a seguir são apresentados alguns dos padrões de comunicação mais utilizados em IoT, destacando suas principais vantagens e desvantagens.

2.4.1 Wi-Fi

Sendo um dos padrões de redes sem fio mais predominante nos dias de hoje, a tecnologia *Wireless Fidelity* (Wi-Fi) utiliza frequências de rádio na faixa de Gigahertz para enviar sinais entre dispositivos, sendo capaz de transmitir dados nas frequências de 2.4 GHz e 5 GHz (ESCOBAR, 2015) a até 100 metros.

Essas frequências são classificadas em licenciadas e não licenciadas e o órgão responsável por essas licenças nos Brasil é a *Agência Nacional de Telecomunicações* (ANATEL). As frequências licenciadas são restritas a determinadas regiões e é necessário o pagamento de taxas para obter o direito de uso. Assim, a ANATEL garante que outros equipamentos não irão operar com a mesma frequência na região. Com as frequências não licenciadas, não é necessário pedir licença e o equipamento deve obedecer os limites impostos com relação à potência e diversos equipamentos podem operar na mesma frequência, aumentando,consequentemente, a chance de interferência.

Utilizando o padrão IEEE 802.11 e com suas variações, o Wi-Fi proporciona um excelente meio pelo qual diversos dispositivos podem se conectar a uma rede, sendo o meio utilizado em aparelhos celulares, notebooks, tablets, smartwatches e todo dispositivo capaz de transmitir informação a altas taxas de transmissão, chegando atualmente a até 1300 Mbps. Porém posui aspectos negativos, entre eles o alto custo energético e o curto alcance, para isso novos padrões como o IEEE 802.15.4 (Seção 2.4.3) foram criados.

2.4.2 Bluetooth

O Bluetooth é considerado um padrão para ser utilizado em redes *Personal Area Network* (PAN). Com frequência de 2.4 GHz, o padrão possui uma variação chamada *Bluetooth Low Energy* (BLE) do padrão Bluetooth 4.0, que tem o objetivo de reduzir o consumo de energia, sendo útil para aplicações que utilizem bateria (GUPTA, 2013). O BLE está presente na maioria dos dispositivos móveis e vem evoluindo, está na versão 5.0 com taxa de transmissão de até 50 Mbps.

Devido ao baixo alcance, de até 40 metros, esse padrão foi descartado para uso neste trabalho.

2.4.3 LR-WPAN

Um ponto importante na concepção de redes IoT é o desafio para implementar dispositivos móveis e sem fio com baixo custo energético, sem necessariamente estarem conectados à energia. Por essa razão, padrões como o IEEE 802.15.4 apresentam uma solução eficiente para a comunicação entre dispositivos autônomos, otimizando a eficiência energética em relação aos outros padrões de comunicação (Fernandez; Jara; Skarmeta, 2013).

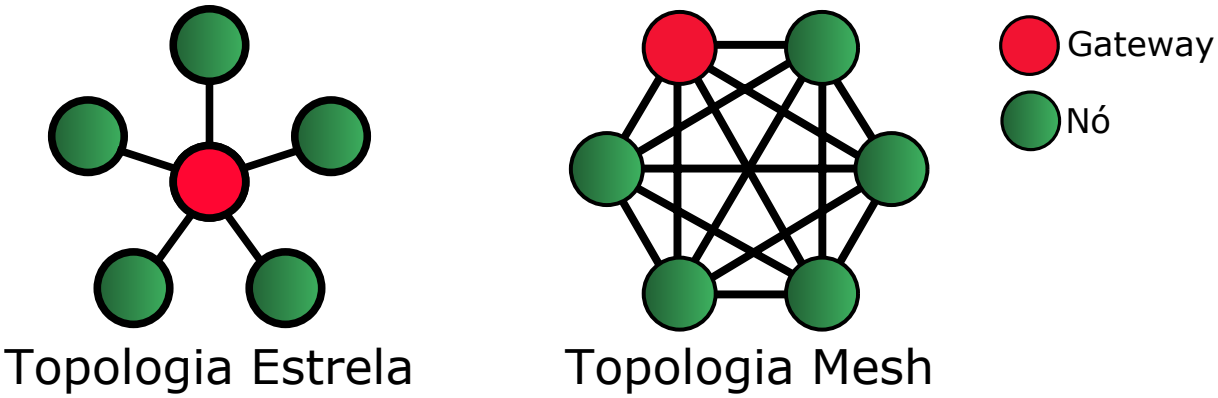
O padrão IEEE 802.15.4 tem sido amplamente utilizado em diversos dispositivos e protocolos de comunicação que tem como principais características: baixo poder computacional, baixa potência de sinal e baixo consumo de energia (IEEE. . . , 2016). Esses dispositivos tem sido utilizados em diversas aplicações, como: automação residencial, sensoriamento industrial, monitoramento ambiental, medicina, aplicações militares e agricultura de precisão.

Por trabalhar com uma taxa de transferência menor e um baixo consumo de energia, redes que utilizam o padrão IEEE 802.15.4 são conhecidas como *Low Rate - Wireless Personal Area Network* (LR-WPAN), trabalhando com taxas de transmissão de até 250 Kb/s.

O protocolo 802.15.4 permite avaliar o nível de interferências em um determinado instante e, assim, ajustar o canal de comunicação. Suportando topologias em estrela e Mesh (Figura 2), a comunicação é estabelecida entre os dispositivos e um controlador, chamado de coordenador PAN ou *gateway* (PINTO, 2017). Essas topologias se diferenciam pelo fato de na topologia estrela os nós serem ligados diretamente ao *gateway* e nas topologias dinâmicas como a topologia Mesh, os nós podem se comunicar entre eles de forma autoconfigurável e auto-organizável criando e mantendo automaticamente a conectividade.

Com a grande demanda com relação à eficiência energética, redução de custos operacionais e alcance, novos padrões de comunicação foram criados. Em Abril de 2012, o padrão IEEE 802.15.4g ou LPWAN (Seção 2.4.4) foi publicado com o objetivo de suprir essa demanda de dispositivos IoT (Kuor-Hsin Chang; Mason, 2012).

Figura 2 – Topologias de redes



Fonte: Autoria Própria

2.4.3.1 ZigBee

Definido por uma aliança de empresas de diferentes segmentos no mercado, denominada "Zigbee Alliance", o padrão ZigBee é baseado no padrão 802.15.4, opera nas frequências de 2,4 GHz (faixa ISM), 868 MHz e 915 MHz, com uma taxa de transferência máxima de 250 Kb/s e alcance de até 100 metros. Por ser um padrão proprietário, o ZigBee possui algumas limitações como a necessidade de licenciamento para utilizar o padrão para uso comercial. Essas limitações, juntamente com o curto alcance, fizeram com que o ZigBee fosse descartado para utilização nesse trabalho.

2.4.4 LPWAN

Com a necessidade de enviar poucos dados à distâncias relativamente grandes e com um baixo custo energético, as redes *Low Power Wide Area Network* (LPWAN) que utilizam o padrão IEEE 802.15.4g são frequentemente utilizadas em IoT. Algumas tecnologias baseadas em LPWAN como o LoRa (Seção 2.4.4.2), Sigfox (Seção 2.4.4.1), e algumas especificações do LTE utilizam a topologia em estrela (demonstrado na Seção 2.4.3), ou seja, os dispositivos integrados à rede são diretamente ligados ao ponto de acesso, eliminando a implementação de protocolos de roteamento de malha sem fio, como a rede Mesh.

Com uma arquitetura de comunicação estruturada em camadas, a camada física da LPWAN trabalha sobre as bandas de frequência *Industrial Sientific and Medical* (ISM), que são frequências isentas de licença de rádio, podendo operar em 3 faixas de frequência (PINTO, 2017):

- 2450 MHz (amplamente utilizada e adotada pelo Brasil) com 16 canais e taxa de 250 Kb/s.
- 915 MHz (apenas nas Américas) com 10 canais e taxa de 40 Kb/s.
- 868 MHz (padrão Europeu) com apenas 1 canal e taxa de 20 Kb/s.

2.4.4.1 SigFox

A tecnologia SigFox é proprietária. A empresa SigFox e seus operadores detêm o controle da tecnologia e dos servidores (NUNES, 2017). Operando em 869 MHz e 915 MHz com um alcance de até 50 quilômetros em área aberta, possui uma taxa de transferência de 100 b/s, com um tamanho máximo de pacote de 12 bytes, sendo que o número máximo de pacotes não pode exceder 140 pacotes/dia.

Todas essas restrições, juntamente com o modelo de negócio proprietário da SigFox, fizeram com que o LoRaWAN (Seção 2.4.4.3) se tornasse mais viável, sendo considerado mais flexível e aberto.

2.4.4.2 LoRa

Long Range (LoRa) é uma técnica de modulação de longo alcance da proprietária LoRa Alliance. Operando em bandas isentas de licença e utilizando uma técnica de modulação de rádio *Chirp Spread Spectrum Modulation* (CSS) na camada física. "LoRa é um esquema proprietário de modulação por espalhamento espectral que é derivado do CSS que negocia uma taxa de dados dentro de uma largura de banda de canal fixo."(SEMTECH CORPORATION, 2015).

O LoRa foi criado pela Cycleo, uma empresa Francesa adquirida pela Semtech em 2012, é baseado em protocolo aberto, mas limitado aos chips da Semtech. A tecnologia LoRa possui baixa taxa de transferência, que varia de acordo com o *Spreading Factor* (SF) e a largura de banda. Podendo ser utilizada em uma faixa de frequência de 137 MHz a 1020 MHz, incluindo as faixas sem necessidade de licença de 169 MHz, 433 MHz, 868 MHz e 915 MHz, sendo um fator fundamental para a utilização em todo o mundo.

Alguns estudos como (Mikhaylov; Juha Petaejaejaervi; Haenninen, 2016) e (Adelantado et al., 2017) analisaram as limitações e o desempenho da modulação LoRa e, assim, alguns pontos podem ser explorados:

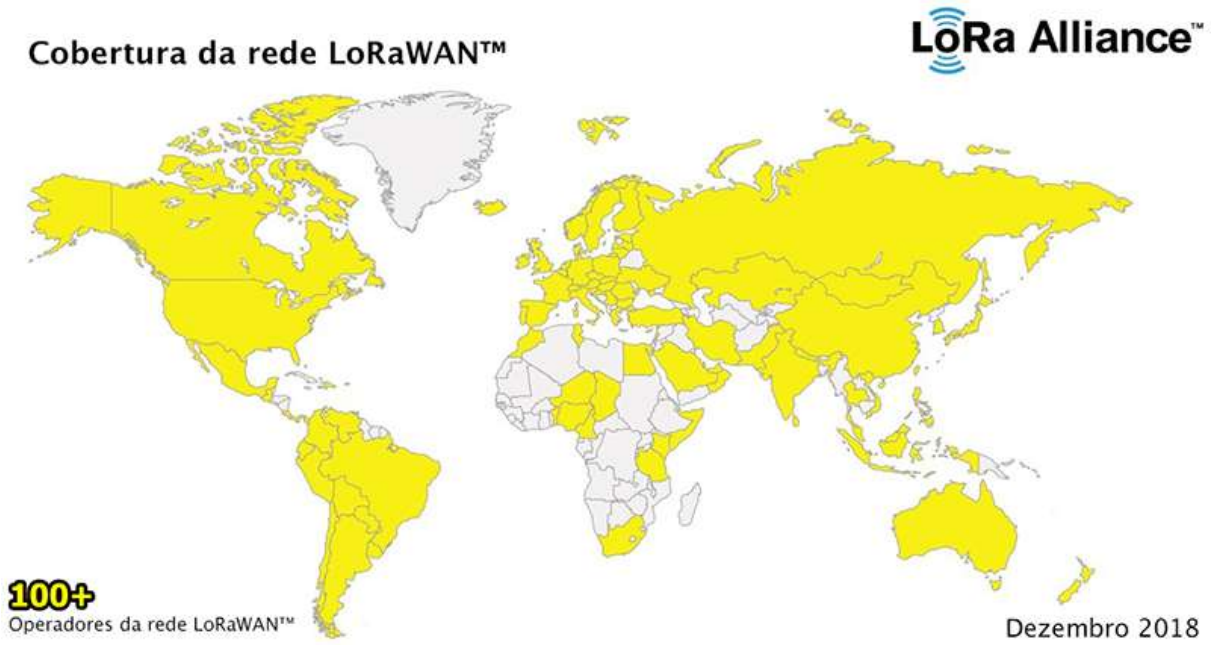
- O baixo consumo de energia, otimizando a vida útil da bateria;
- O alcance de quilômetros em campo aberto *(outdoor)* e de centenas de metros em ambiente fechado *(indoor)*;
- A capacidade da rede receber mensagens de uma variedade de dispositivos;
- Robustez com relação às interferências, graças à modulação CSS.

2.4.4.3 LoRaWAN

Baseado na modulação LoRa (que atua na camada física), o *Long Range Wide Area Network* (LoRaWAN) é um protocolo que atua na camada *Media Access Control* (MAC) e é de código aberto. É uma das tecnologias mais adotadas em IoT, possuindo uma taxa máxima de transferência de 37.5 Kb/s e permitindo que um único gateway possa receber dados de

múltiplos nós a até 20 quilômetros de distância em área aberta. A Figura 3 mostra um mapa da cobertura da rede LoRaWAN no mundo.

Figura 3 – Cobertura da rede LoRaWAN no mundo em dezembro de 2018



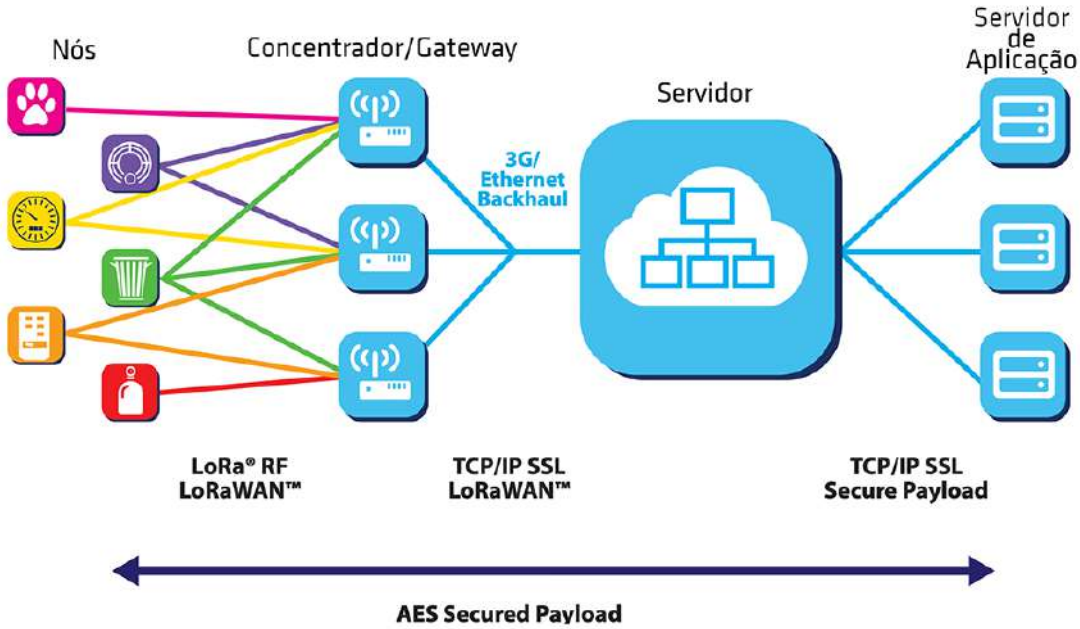
Traduzido de: LoRa Alliance (2019).

O LoRaWAN define o protocolo de comunicação e arquitetura do sistema para a rede garantindo uma comunicação segura e confiável, possibilitando alterações no cabeçalho. Diferentemente das redes de celular, que operam em faixas licenciadas, o LoRaWAN opera em uma faixa de frequência ISM. Esse fato favorece a implantação de redes LoRaWAN privadas, sem a necessidade de utilizar operadoras móveis.

A Figura 4 mostra a arquitetura de uma rede LoRaWAN. Alguns aspectos diferenciam LoRaWAN de outras tecnologias de rádio que afetam a implementação e o desenvolvimento em protocolos de roteamento. LoRaWAN utiliza topologia estrela, com os dispositivos finais conectados a *gateways*, que por sua vez são conectados a um servidor. Embora os dispositivos finais possam receber dados de *gateways* diversos, as transmissões de recebimento dos dados chegarão no dispositivo final por meio de apenas um único *gateway*.

É utilizada a topologia estrela no sentido de manter um consumo energético baixo, pois redes dinâmicas requerem maior complexidade na comunicação, mantendo uma comunicação constante com os dispositivos e consequentemente aumentando o consumo energético. Portanto, os nós da rede LoRaWAN são assíncronos e se comunicam quando eles possuem algum dado pronto para enviar através de eventos ou agendamento. Já em uma rede dinâmica ou Mesh, os nós precisam se comunicar frequentemente para sincronizar a rede e verificar se possuem "mensagens".

Figura 4 – Arquitetura da rede LoRaWAN

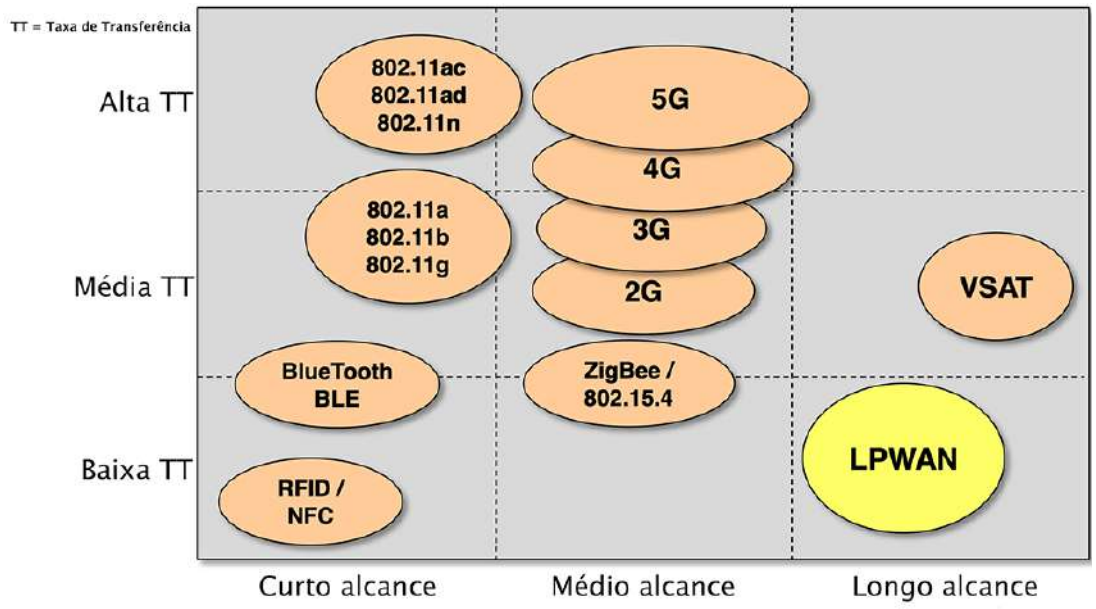


Traduzido de: Workgroup (2015).

2.4.5 Considerações

A Figura 5 ilustra os diferentes padrões de rede utilizados com relação à taxa de transferência e o alcance. Além dos padrões IEEE apresentados na Tabela 2, outros padrões como redes de telefonia (2/3/4/5G) e satélite (VSAT) são apresentados para uma melhor comparação, porém não serão utilizados no trabalho devido a alguns fatores como custo, serem proprietários e as redes de telefonia podem não abranger a área desejada para o mapeamento.

Figura 5 – Taxa de transferência vs. capacidade de alcance das tecnologias de rádio comunicação



Adaptado de: Egli (2015).

A Figura 5 mostra qual tecnologia deverá ser utilizada para atender alguns dos principais requisitos para o trabalho que é o longo alcance. Os dois principais padrões possíveis de serem utilizados neste trabalho são o SigFox e o LoRaWAN que utilizam o padrão 802.15.4g, apresentado na Figura 5 como LPWAN.

Para dispositivos IoT a conexão com a Internet é essencial, permitindo que os dispositivos troquem informações entre si e com serviços de *backend* ou *brokers* que permitam a obtenção e análise dos dados. A Seção 2.5.2 aborda o protocolo *Message Queue Telemetry Transport* (MQTT), especialmente desenvolvido com base na pilha TCP/IP e que se tornou padrão para comunicações de IoT (YUAN, 2017).

2.5 PROTOCOLOS DE COMUNICAÇÃO

Implementados tanto em hardware como em software, os protocolos de comunicação são um conjunto de regras e formatos para mensagens, informando a sintaxe, a semântica e a sincronização que permitem a troca de mensagens por meios analógicos e digitais (ROSS, 2009).

Os protocolos de comunicação abrangem a autenticação, a detecção e a correção de erros, podendo possuir diversas propriedades para uma transmissão, como tamanho do pacote, técnicas de sincronização, mapeamento de endereço, controle de fluxo, rotas, controle de sequência de pacotes, velocidade de transmissão, entre outros.

Diversos protocolos estão disponíveis para serem utilizados em IoT, cada um focado em necessidades que o cliente possa ter. Alguns focados em aplicações que necessitem de transações rápidas e confiáveis, como o *Advanced Message Queuing Protocol* (AMQP) e o *Java Message Service* (JMS). Outros focados na coleta de grande quantidade de dados (como exemplo, sensoriamento) em redes restritas, como o MQTT. E alguns projetados para aplicações que necessitem da comunicação com a Internet, como os protocolos *Representational State Transfer* (REST) e o *Hypertext Transfer Protocol* (HTTP) (Naik, 2017).

A seguir são apresentados os protocolos de comunicação utilizados neste trabalho.

2.5.1 HTTP

O HTTP é um protocolo de comunicação que vem sendo utilizado na Internet desde 1990. As versões iniciais foram desenvolvidas com o objetivo de transferir informações através da Internet, basicamente utilizando páginas em *HyperText Markup Language* (HTML). As versões mais recentes possibilitaram a transferência de informações com cabeçalhos mais elaborados utilizando uma codificação do tipo *Multipurpose Internet Mail Extensions* (MIME), padrão este que possibilita a inserção de documentos (imagens, sons, textos) em uma mensagem (MUXFELDT, 2017).

O HTTP tem sido amplamente utilizado para a transferência de dados. Porém, diferentemente da Web que utiliza um único protocolo de mensagens, para comunicações em IoT com uma grande quantidade de informações que podem ser trocadas, não é recomendado confiar em um único protocolo para todas as necessidades.

O HTTP tem algumas limitações:

- O HTTP é síncrono, ou seja, o cliente espera uma resposta do servidor, o custo disso é a baixa escalabilidade. Em IoT, a comunicação síncrona pode ser um problema, devido à possibilidade de ter um grande número de dispositivos conectados à rede que é muitas vezes não confiável e com alta latência; Um protocolo assíncrono é o mais adequado, permitindo à rede IoT descobrir o caminho e a sincronização ideal para a troca de mensagens.
- HTTP é unidirecional, ou seja, é necessário iniciar a conexão. Como em IoT os

sensores geralmente são clientes, significa que eles não podem receber comandos da rede passivamente;

- O HTTP é um protocolo com muitos cabeçalhos e regras.

A Seção a seguir é sobre o protocolo MQTT, que surgiu com o objetivo de suprir essas limitações do HTTP.

2.5.2 MQTT

O MQTT foi desenvolvido pela IBM nos anos 90 com o objetivo de vincular sensores em oleodutos de petróleo a satélites e tem sido amplamente utilizado como protocolo de comunicação entre dispositivos IoT. O MQTT é um protocolo de mensagens com suporte à comunicação assíncrona entre as partes e apesar do nome, o MQTT não utiliza filas de mensagens, ele utiliza um modelo de publicação e assinaturas (YUAN, 2017). Sendo leve e flexível, o MQTT oferece suporte a diversos cenários de aplicativos para dispositivos e serviços de IoT, permitindo conectar os dispositivos IoT aos serviços da web.

O que faz o MQTT ser um protocolo leve e flexível é a utilização de um cabeçalho simples, especificando o tipo da mensagem, um tópico e uma carga útil binária aleatória e o fato de utilizar um modelo de publicação e assinaturas, desacoplando o publicador e o consumidor de dados. Utilizando dois tipos de entidades na rede, um *message broker* (plataforma intermediária de mensagens) e inúmeros clientes, sendo que o *broker* pode ser um servidor que recebe todas as mensagens enviadas pelos clientes e roteando-as, e os clientes podem ser dispositivos com sensores ou um aplicativo que processa dados de IoT (YUAN, 2017).

O MQTT funciona da seguinte forma:

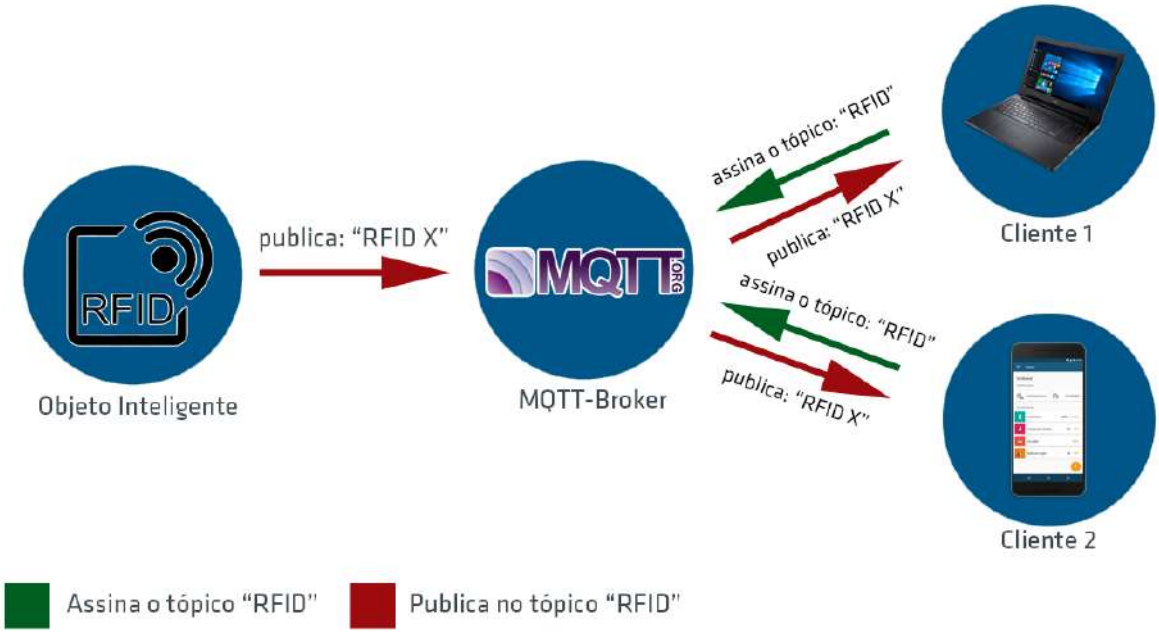
1. O cliente se conecta ao *broker*, assinando qualquer "tópico". Tanto um usuário como o objeto inteligente podem assinar ou publicar nos tópicos, que são organizados em uma linha de texto, sendo cada item separado por uma barra (/);

2. O objeto inteligente envia os dados para o tópico no *broker*, podendo ser uma conexão TLS criptografada para proteger os dados;

3. O *broker* encaminha a mensagem contendo os dados para todos os clientes que assinam o tópico.

A Figura 6 apresenta uma proposta da estrutura do MQTT.

Figura 6 – Estrutura MQTT



Fonte: Autoria própria.

Como o protocolo MQTT não impõe restrições quanto ao formato de dados, o sistema pode utilizar um método de criptografia e um mecanismo de atualização de chave, sendo que todo o conteúdo na carga útil podem ser dados binários criptografados das mensagens JSON ou XML reais (YUAN, 2017).

3 METODOLOGIA

A seguir são apresentados os materiais e os métodos utilizados para o desenvolvimento do trabalho proposto.

3.1 MATERIAIS

3.1.1 Softwares

Nas seções a seguir são apresentados os softwares utilizados para a organização do trabalho em si e para o desenvolvimento e execução dos códigos desenvolvidos.

3.1.1.1 Visual Studio Code e PlatformIO

O Visual Studio Code é um editor de texto gratuito e multiplataforma disponibilizado pela Microsoft para o desenvolvimento de aplicações. Esse editor permite adicionar extensões e *plug-ins*, ampliando a gama de linguagens de programação suportadas (VISUAL..., 2019).

Para o projeto foi utilizado o ambiente PlatformIO (PLATFORMIO, 2019) instalado por meio do Visual Studio Code e utilizado na própria IDE. O PlatformIO é um ambiente voltado para o desenvolvimento IoT, que permite a criação de projetos para diversas placas embarcadas e a instalação e suporte de diversos *frameworks*, contando com um terminal interno juntamente com um monitor de porta serial.

3.1.1.2 Docker

O Docker é uma plataforma *open source* escrita em Go, uma linguagem de programação desenvolvida pelo Google que facilita a criação e a administração de ambientes isolados. Diferentemente de um sistema de virtualização tradicional que virtualiza um Sistema Operacional (SO) completo e isolado, o Docker possui recursos isolados, possibilitando o empacotamento de uma aplicação ou ambiente inteiro dentro de um contêiner, tornando esse ambiente portável para qualquer outro *host* (hospedeiro) que contenha o Docker instalado (DOCKER, 2019).

O Docker foi utilizado neste trabalho para a criação e a manutenção do banco de dados em MongoDB.

3.1.1.3 Studio 3T

O Studio 3T é uma IDE multiplataforma para o MongoDB. Com diversas funcionalidades, entre elas a possibilidade de efetuar consultas e explorar diretamente as informações no banco de dados utilizando consultas básicas em SQL (STUDIO..., 2019). O Studio 3T foi utilizado no projeto para gerenciar o banco de dados que armazenará as informações das estações base.

3.1.1.4 Node-RED

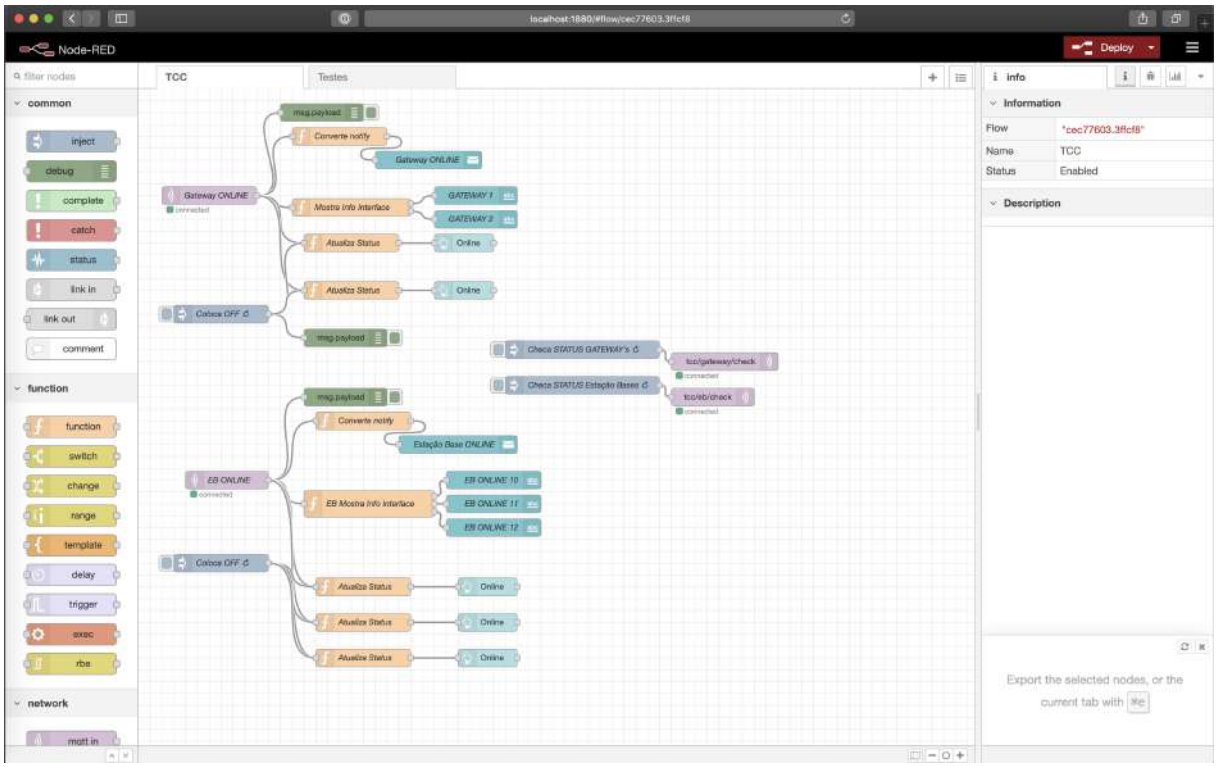
Com o objetivo de conectar dispositivos de hardware, APIs e serviços *online*, o Node-RED é uma ferramenta de programação visual (NODE-RED, 2019). Criado pela IBM Emerging Technology e focado em IoT, o Node-RED permite conectar de forma simplificada dispositivos e possui diversas funcionalidades, como criar rotinas, criar uma interface de usuário, receber eventos nos protocolos HTTP e MQTT e fazer a integração com banco de dados. Permite criar nós e programar de forma nativa em JavaScript ou em diversas outras linguagens (POR..., 2019).

O Node-RED é uma ferramenta baseada em Node.js, que é um interpretador em código aberto que utiliza JavaScript de forma assíncrona e orientado a eventos, sendo capaz de gerenciar milhares de requisições em tempo real. Com o Node.js é possível criar aplicações JavaScript para executar como uma aplicação *standalone* na máquina, ou seja, não depende de um *browser* para a execução. Foi escolhido o Node.js pela sua alta capacidade de escala, sua flexibilidade, baixo custo, arquitetura, sendo útil para a implementação de microsserviços (INTRODUCTION..., 2019).

O Node.js permite a instalação de diversos "pacotes" para a utilização de serviços. Para o *broker* MQTT foi instalado o Mosca (MOSCA..., 2019), e integrado juntamente com o servidor para a utilização do banco de dados MongoDB (MONGODB, 2019).

A Figura 7 ilustra o ambiente de desenvolvimento da ferramenta e as rotinas criadas para o funcionamento do projeto.

Figura 7 – Ambiente de desenvolvimento da ferramenta Node-RED



Fonte: Autoria própria.

3.1.1.5 Mosca MQTT

É um broker MQTT *open source* (código aberto) leve e flexível para ser utilizado tanto em servidores, ou em dispositivos que necessitem de pouco consumo energético como microcontroladores (MOSCA. . . , 2019). Podendo ser executado tanto de forma independente como de forma embarcada em aplicações Node.JS utilizando a linguagem de programação *Javascript*, a instalação e execução do broker é simples.

3.1.1.6 MongoDB

Ao invés de utilizar o conceito de tabelas como a maioria dos bancos de dados, o MongoDB utiliza uma estrutura de dados *NoSQL* utilizando o conceito de coleções. Ambas são parecidas, porém as coleções não são relacionais e não possuem uma estrutura fixa, ou seja, os dados podem ser armazenados como documentos, grafos, chave/valor, ou colunas como no *SQL*. Os dados são armazenados em documentos, utilizando o formato *JSON* (MONGODB, 2019).

3.1.2 Microcontroladores

Para o desenvolvimento das estações base, do *gateway* e para o servidor foram utilizados os seguintes componentes:

3.1.2.1 Heltec ESP32 LoRa

Foi utilizado o módulo da Heltec com o microcontrolador ESP32 para a identificação do drone utilizando uma *tag* RFID acoplada ao drone e um leitor RFID RC522 integrado ao módulo. O módulo, localizado na estação base, é responsável pela comunicação com o *gateway* utilizando a tecnologia LoRaWAN, já integrada ao módulo através do chip LoRa SX1278. Módulo Heltec ESP32 LoRa (Figura 8) possui as seguintes especificações:

- **Processador:** Processador Tensilica LX6 ESP32 240MHz Dual-Core;
- **Comunicação LoRa:** Chip LoRa SX1278;
- **Portas:** 3x UART; 2x SPI; 2x I2C; 1x I2S;
- **Memória:** 4MB(32M-bits) SPI FLASH.

Figura 8 – Módulo Heltec ESP32 LoRa com display integrado

3.1.3 Antena Steelbras DUAL VHF/UHF AP0188

A AP0188 (Figura 9 é uma antena Dual Band que opera nas faixas 1/4 VHF e 5/8 UHF para ser utilizada com transceptores Dual Band.

Figura 9 – Antena AP0188

E possui as seguintes especificações:

- **Frequência VHF:** 144 - 148 MHz;
- **Frequência UHF:** 430 - 440 MHz;
- **Potência Máxima:** 150 Watts;
- **Impedância:** 50 Ohms;
- **Ganho VHF:** 0 dB - 2,15 dBi;
- **Ganho UHF:** 3 dB - 5,15 dBi;
- **Altura:** 440mm;
- **Peso:** 105g.

Para a utilização com a antena foram utilizados dois adaptadores para cada ligação conforme a Figura 10.

Figura 10 – Pigtail e adaptador UHF utilizados com a antena AP0188

3.1.4 Módulo Leitor e Tag RFID

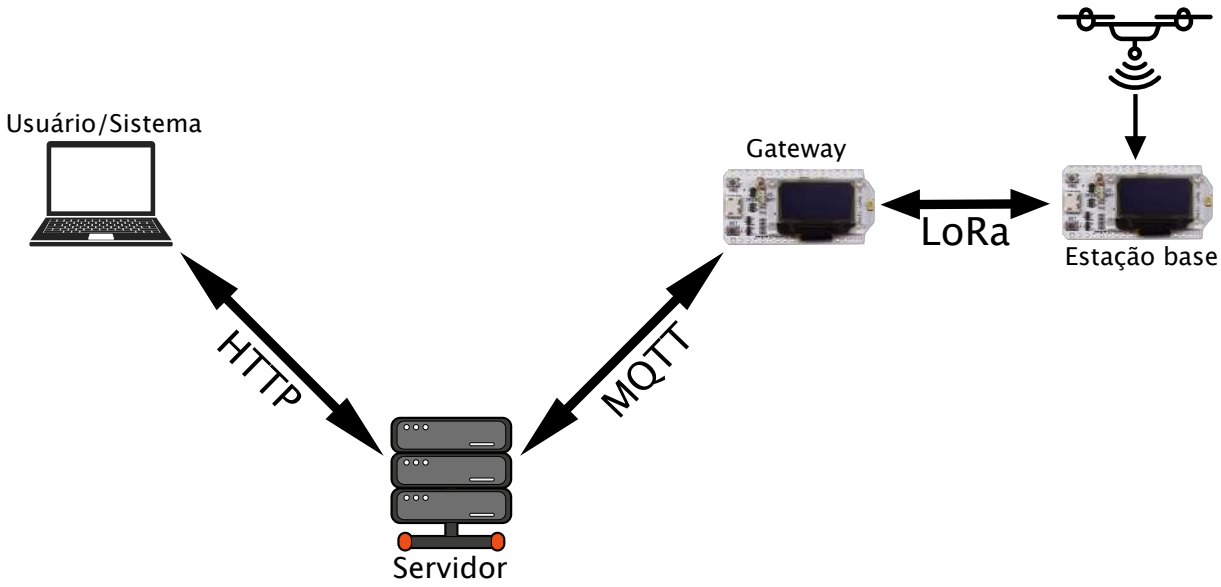
Módulo (Figura 11) capaz de obter uma identificação de uma *tag* ou cartão, possuindo um alcance para leitura de até 5 centímetros de distância. A *tag* estava acoplada ao drone possibilitando identificar o drone através do leitor que ficava na estação base.

Figura 11 – Módulo leitor RFID RC522, tag e cartão

3.2 MÉTODO

O projeto foi construído com base na arquitetura da IoT, possuindo 4 componentes principais, sendo eles: a estação base, o *gateway*, o servidor e o usuário final. Esses componentes são apresentados na Figura 12.

Figura 12 – Arquitetura do projeto

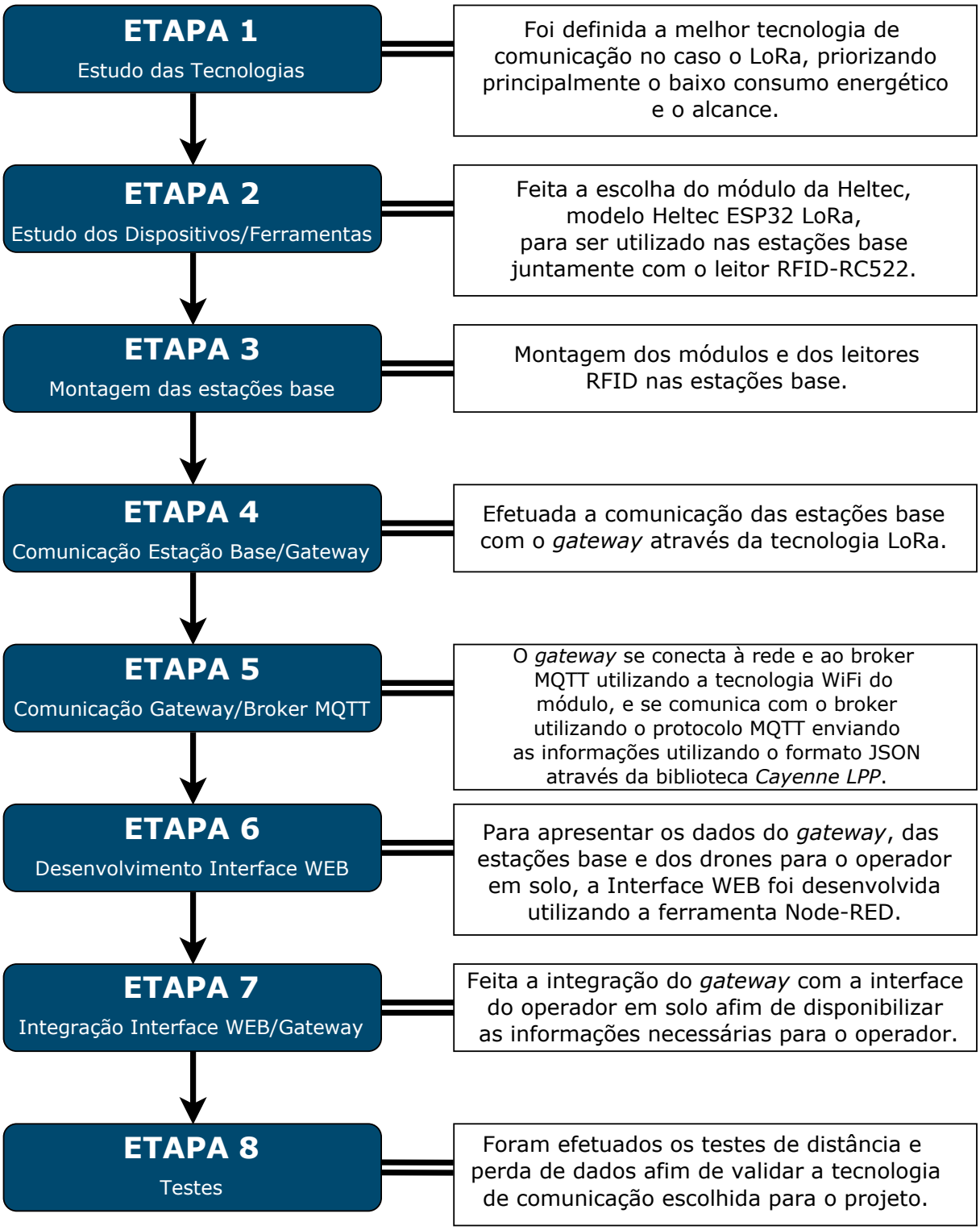


Fonte: Autoria própria.

Assim que o drone pousar na estação base, a estação base contendo os sensores se comunica com o *gateway* utilizando a tecnologia de comunicação LoRaWAN. O *gateway* por sua vez se comunica com o servidor através do protocolo de comunicação MQTT e o servidor recebe as informações do *gateway* permitindo, assim, que o usuário/sistema visualize as informações utilizando o protocolo de comunicação *Hypertext Transfer Protocol* (HTTP), sendo capaz de efetuar a leitura e a interpretação dos dados obtidos.

Como procedimento de trabalho foram definidas as 8 etapas ilustradas na Figura 13 a seguir:

Figura 13 – Fluxograma das Etapas



Fonte: Autoria própria.

4 COMUNICAÇÃO ENTRE ESTAÇÕES BASE PARA UTILIZAÇÃO COM UM GRUPO DE DRONES

Este capítulo apresenta o desenvolvimento realizado e os resultados obtidos.

4.1 Escopo do sistema

O sistema desenvolvido foi divido em partes e processos que são descritos nas seções a seguir.

4.1.1 Servidor

Computador ou microcontrolador capaz de armazenar e gerenciar em um banco de dados todas as informações obtidas pelas estações base e recebidas através do *gateway*, além de manter a interface Web que será utilizada pelo operador e enviar comandos utilizando o *gateway* para as estações base novamente.

Foi utilizado um computador, onde foi instalado o aplicativo Docker (DOCKER, 2019) que é capaz de executar o banco de dados MongoDB (MONGODB, 2019) e a aplicação Node-RED (NODE-RED, 2019). Após a execução do banco de dados e da ferramenta Node-RED, é executado o código desenvolvido em *JavaScript* utilizando a aplicação em NodeJS, onde é feita a conexão com o banco de dados, a inicialização do *broker* MQTT Mosca, a subscrição nos tópicos pré-definidos e todas as rotinas de criação dos *logs* para os eventos recebidos nos tópicos MQTT.

4.1.2 Interface para utilização do operador

Uma interface WEB desenvolvida e mantida pelo servidor com o objetivo de permitir o controle e o monitoramento pelo operador em solo das estações base. Foi desenvolvido uma aplicação utilizando o Node-RED (Seção 3.1.1.4).

A aplicação desenvolvida em Node-RED é capaz de gerenciar as informações que são recebidas pelo *broker* MQTT e permite criar a interface de usuário que foi utilizada. A interface de usuário mostra em tempo real o *status* das estações base e dos *gateways*, se tem algum drone pousado na estação base, e é capaz de enviar requisições para as estações base a fim de obter uma resposta com o *status* das mesmas.

4.1.3 Gateway

O gateway é representado por um módulo da Heltec (Seção 3.1.2.1) capaz de trocar informações com as estações base utilizando protocolo LoRa (Seção 2.4.4.2) e com o servidor utilizando o protocolo MQTT (Seção 2.5.2), o *gateway* é a "ponte" entre as estações base e o servidor.

Para este trabalho, a conexão entre o *gateway* e o servidor se dá por meio de uma rede local, em que o *gateway* se conecta ao *broker* utilizando o endereço IP da rede local do servidor e utiliza a porta 1883.

4.1.4 Estação base

Assim como o *gateway*, a estação base é representada por um módulo da Heltec (Seção 3.1.2.1) utilizando a antena AP0188 (Seção 3.1.3) juntamente com o leitor RFID (Seção 3.1.4) e para este projeto foi alimentado por um carregador externo. Em trabalhos futuros poderá ser implementado na estação base um carregador solar juntamente com uma bateria.

A estação base tem a função de obter a informação do drone por meio da *tag* RFID (Seção 3.1.4) e transmitir para o *gateway* essa informação juntamente com outras informações relevantes. E tem a capacidade de receber solicitações do servidor que são passadas por meio do gateway para a estação base.

4.1.4.1 Testes de distância

Para verificar a distância de comunicação em um ambiente real de uso, foram utilizados diversos cenários para os testes e dois tipos de antena. A antena padrão que acompanha o microcontrolador Heltec ESP32 (Figura 8) e a antena da Steelbras AP0188 (Figura 9). Como o conector do módulo utiliza o padrão IPEX (U.FL), para utilizar a antena AP0188 foi necessário utilizar dois adaptadores, um conector UHF x SMA e um Pigtail SMA x IPEX (U.FL) (Figura 10).

Os códigos utilizados para os teste de distância estão no Apêndice A, tanto para o módulo transmissor (Código A.1) como para o receptor (Código A.2).

Para os testes a seguir foram utilizados os seguintes parâmetros para os módulos LoRa:

- **Frequência de Operação:** 433 MHz;
- **Fator de espalhamento (Spreading Factor):** SF11;
- **Largura de Banda (Bandwidth):** 125 kHz;
- **Taxa de Codificação (Coding Rate):** 4/5.

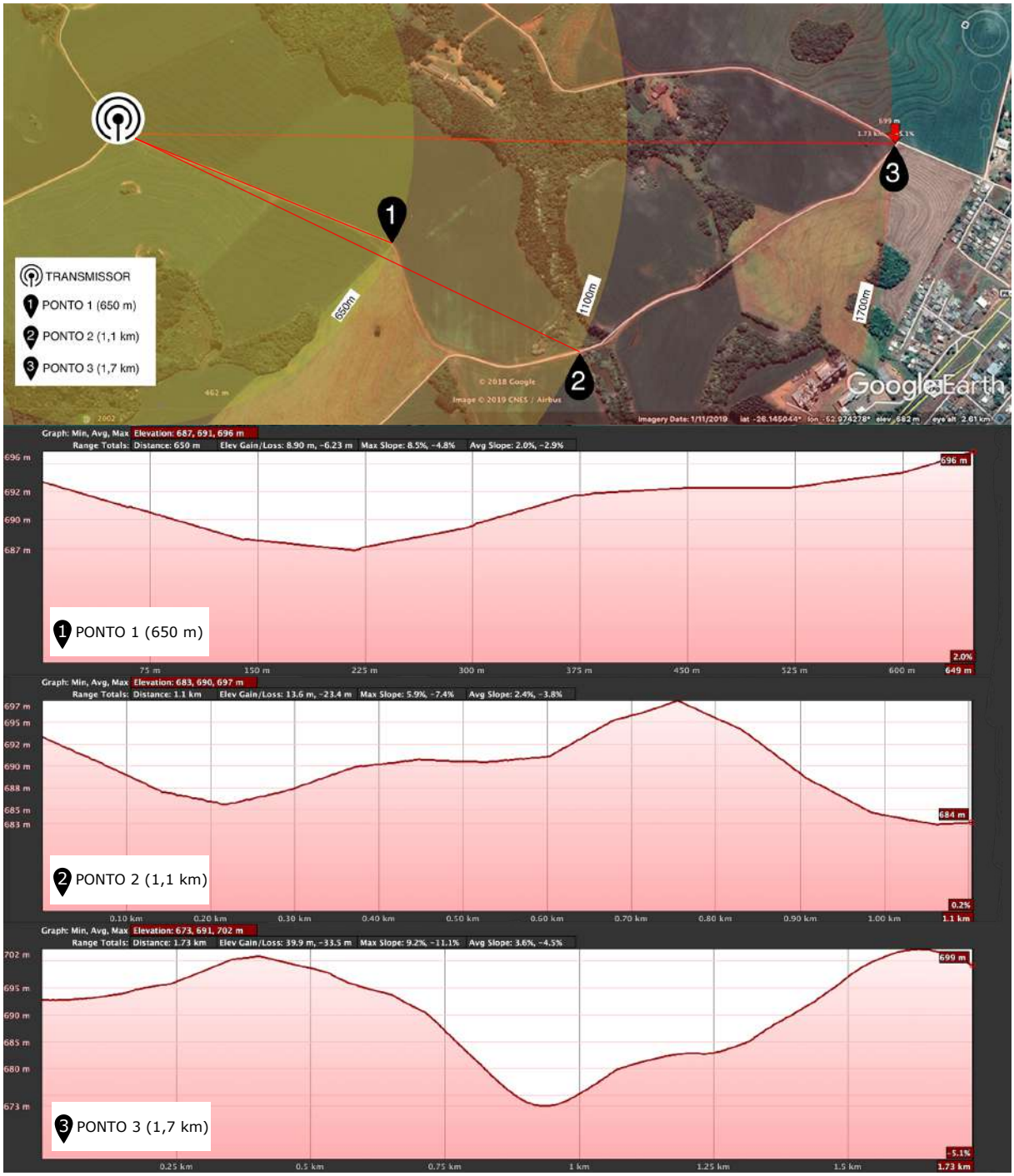
Segundo o Plano de Atribuição, Destinação e Distribuição de Frequência no Brasil disponibilizado pela Anatel (PLANO..., 2019), as radiofrequências disponíveis para o desenvolvimento industrial, científico e médico (Industrial, Scientific, Medical - ISM) estão nas faixas de frequência de 902 MHz-928 MHz, 2.4 GHz-2.5 GHz e 5.725 GHz-5.875 GHz. A tecnologia LoRa opera nas frequências de telefonia móvel (433 MHz, 868 MHz ou 915 MHz).

Como foram efetuados apenas testes, a frequência utilizada neste trabalho foi estabelecida em 433 MHz devido à disponibilidade dos módulos operarem somente nesta frequência. Para uma futura aplicação deste trabalho, será necessário fazer a alteração dos dispositivos para adequar-se às normas da Anatel, ou seja, 915 MHz.

O primeiro teste foi realizado no interior da cidade de Renascença no estado do Paraná, utilizando somente a antena padrão do módulo. A Figura 14 mostra o alcance máximo obtido

considerando o terreno irregular.

Figura 14 – Cenário, pontos e mapa de elevação entre o transmissor e o ponto 3 do primeiro teste de distância

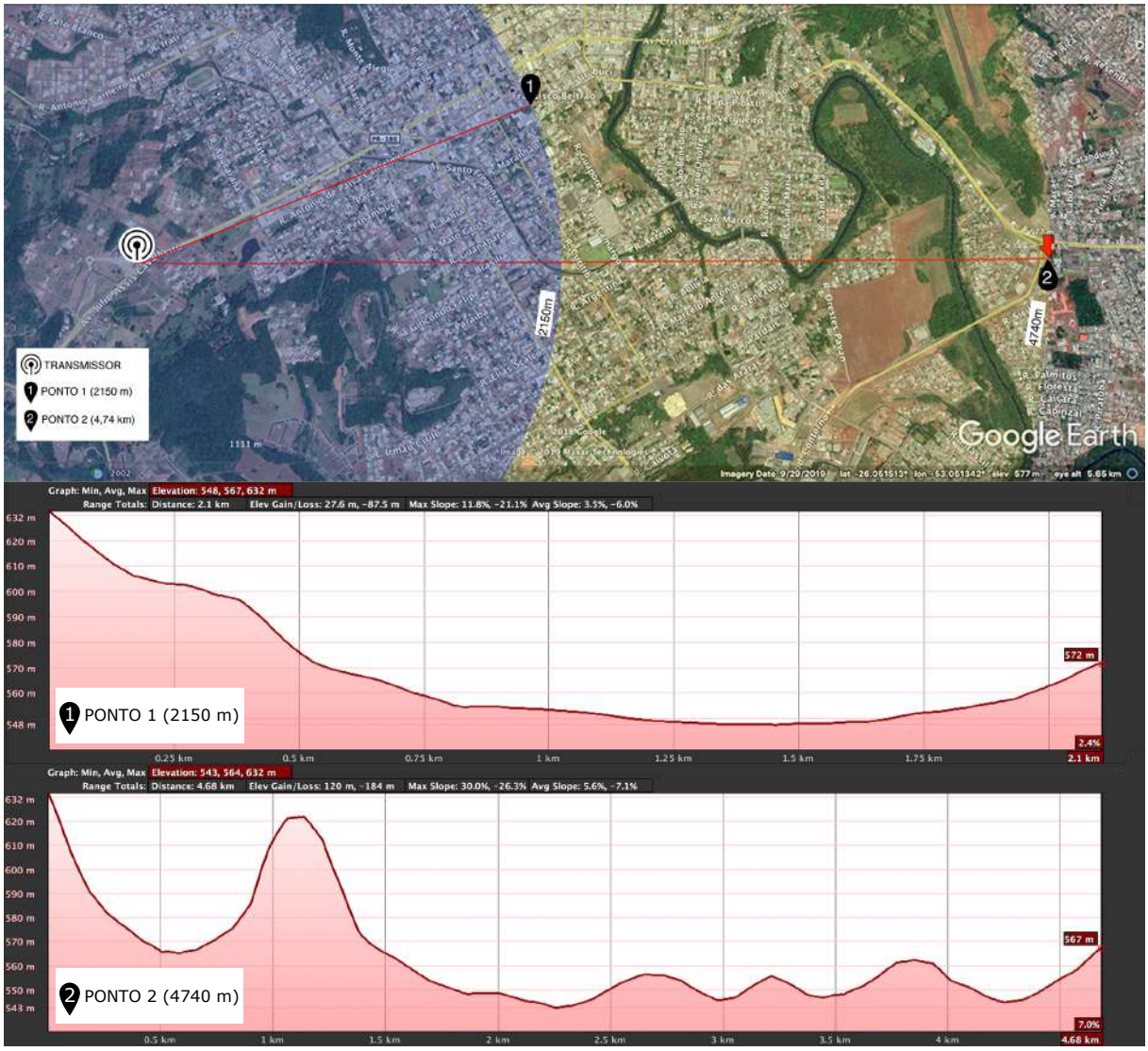


Fonte: Autoria própria.

O teste foi feito com linha de visada ou *Line of Sight* (LoS) até o ponto 2 na imagem, a partir deste ponto e sem linha de visada ou *Non-Line Of Sight* (NLoS) do módulo que estava fixo e transmitindo as informações, a distância máxima obtida foi de 1.7 quilômetros no ponto 3. A partir deste ponto ocorre perda considerável de pacotes.

Devido ao alcance pequeno se considerado o problema proposto neste trabalho, foi adquirida uma antena mais potente para obter um alcance maior. O segundo teste foi feito na cidade de Francisco Beltrão, no Estado do Paraná, utilizando tanto a antena padrão do módulo (Figura 8) como a antena AP0188 (Figura 9), a Figura 15 ilustra a distância máxima obtida utilizando as duas antenas.

Figura 15 – Cenário e pontos de distância máxima do segundo teste



Fonte: Autoria própria.

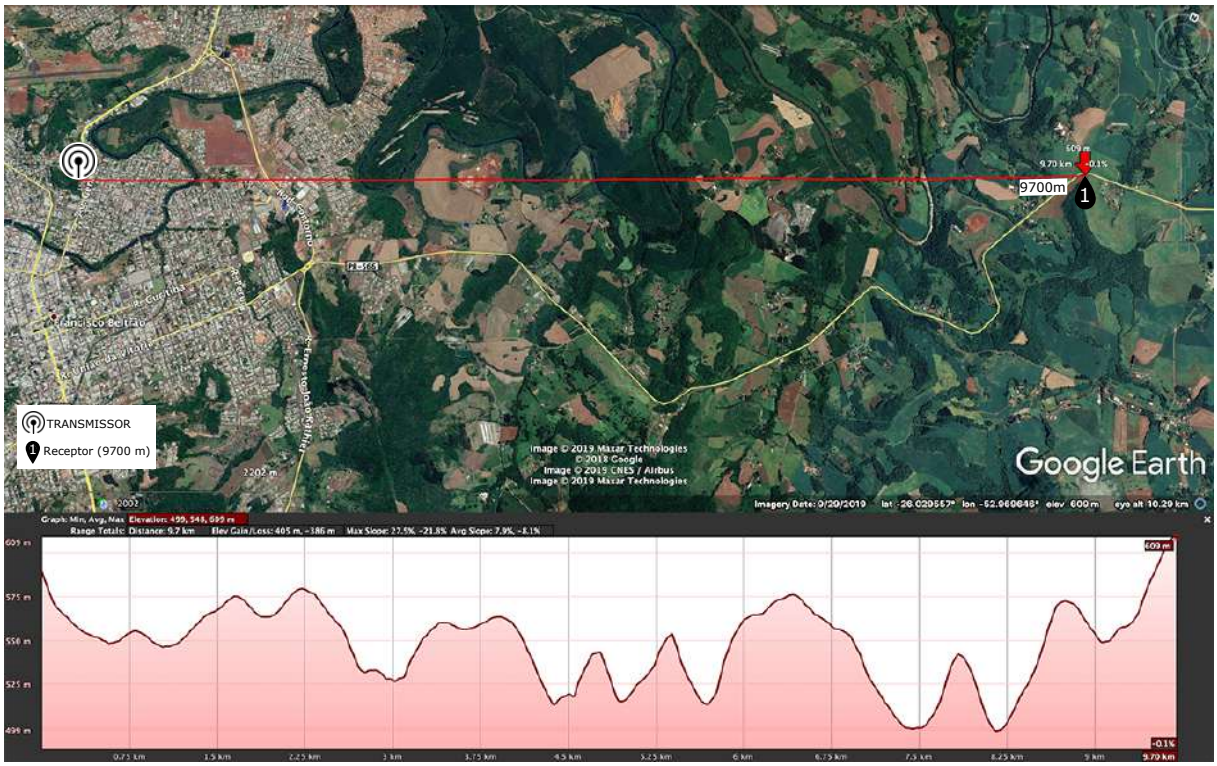
Foram utilizados 3 módulos da Heltec ESP32 LoRa (Seção 3.1.2.1), sendo um transmissor e dois receptores. O transmissor e um dos receptores estavam utilizando a antena AP0188 e o outro receptor a antena padrão.

O transmissor foi colocado em um dos pontos mais altos da cidade e a partir desse ponto foi feito o trajeto de carro em direção ao ponto 2 (Figura 15) a fim de testar a transmissão tanto utilizando a antena padrão quanto a antena AP0188.

A máxima distância obtida com a antena padrão foi de 2.1 quilômetros (Ponto 1), distância essa maior do que a obtida no primeiro teste. Essa distância maior foi devido à linha de visada entre o transmissor e o receptor. Já no módulo receptor utilizando a antena AP0188, a maior distância obtida foi de 4.7 quilômetros (Ponto 2) sem perda de pacotes. A partir desse ponto foi verificado que sem linha de visada ou alguma interferência seja ela de relevo ou edificações, a comunicação é afetada. A figura ilustra o mapa de elevação do transmissor até o ponto 2 comprovando a linha de visada.

Para o terceiro teste foi encontrado um ponto entre as cidades de Francisco Beltrão e Itapejara d'Oeste, no Estado do Paraná, (Figura 16), havendo uma linha de visada entre o transmissor e o receptor de aproximadamente 10 quilômetros. A comunicação entre os dispositivos ocorreu sem perda de dados (na Seção 4.1.4.2 será discutido sobre).

Figura 16 – Mapa de elevação entre o transmissor e o receptor a 9700m



Fonte: Autoria própria.

A seção a seguir apresenta os resultados obtidos dos testes efetuados de perda de pacotes.

4.1.4.2 Testes de perda de pacotes

No teste de perda de pacotes, o código do transmissor (Código A.3) foi desenvolvido de forma a criar um pacote contendo 52 bytes com informações aleatórias e um contador para verificar se houve perda da informação transmitida. O pacote é criado utilizando a biblioteca CayenneLPP (LIBRARY. . . , 2019). Essa biblioteca busca otimizar os dados que necessitam ser enviados por meio de baixas taxas de transferência, organizando as informações por tipo e codificando/decodificando-as para o formato *JavaScript Object Notation* (JSON).

No código foi definido o envio de 200 pacotes por teste em um intervalo de 2 segundos cada, com o receptor recebendo o pacote (Código A.4) e decodificando as informações em JSON para fazer a verificação se o pacote recebido naquele momento é o pacote que realmente deveria ser recebido, ou se houve perda de pacote.

Em tempo real, o *display* do receptor mostra algumas informações (Figura 17) do pacote recebido, como a quantidade de pacotes perdidos, o contador e o tamanho do pacote recebido naquele instante.

Figura 17 – Informações do *display* do receptor no teste de perda de dados

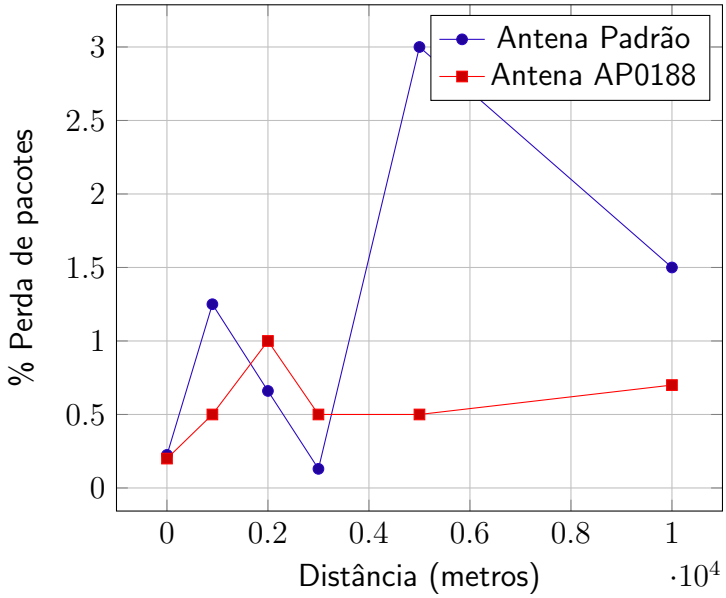
Fonte: Autoria própria.

Os resultados da taxa de perda de pacotes com relação às distâncias de 1m, 900m, 2000m, 3000m, 5000m e 10000m estão apresentados na Tabela 3 e no gráfico da Figura 18.

Tabela 3 – Perda de pacotes Distância vs. Antena (%)

ANTENA	**1m**	**900m**	**2000m**	**3000m**	**5000m**	**10000m**
Padrão	0,225%	1,25%	0,66%	0,13%	3%	2%
AP0188	0,2%	0,5%	1%	0,5%	0,5%	0,7%

Figura 18 – Gráfico da percentagem de perda de pacotes vs. distância



Fonte: Autoria própria.

Analisando os dados obtidos juntamente com a observação nos testes, verificou-se que a comunicação entre os módulos é bastante satisfatória tanto para uma distância pequena, quanto para grandes distâncias desde que haja uma linha de visada. Verifica-se pela linha de erro em distâncias de 900m a 3000m e pela topologia dos pontos onde foram realizados os testes, que os resultados obtidos variaram bastante, significando que alguma fonte de interferência pode estar ocasionando a perda, como o relevo.

De qualquer forma, a perda de aproximadamente 0.7% de pacotes enviados à 10 quilômetros de distância com linha de visada, qualifica, e faz com que o protocolo LoRa cumpra com os requisitos para ser utilizado no trabalho proposto.

4.2 Modelagem e Implementação do Sistema

Esta seção aborda a modelagem de todo o sistema, tanto da interface do operador em solo que é responsável por mostrar as informações das estações base em tempo real, do servidor que é responsável por receber os dados e armazená-los como um *log* para uma futura utilização, como dos microcontroladores que são utilizados nas estações base e nos *gateways*.

4.2.1 Modelagem e Implementação do Servidor

Para a implementação do servidor foi necessária a instalação do Node.js (NODE..., 2019), que é um ambiente de execução JavaScript.

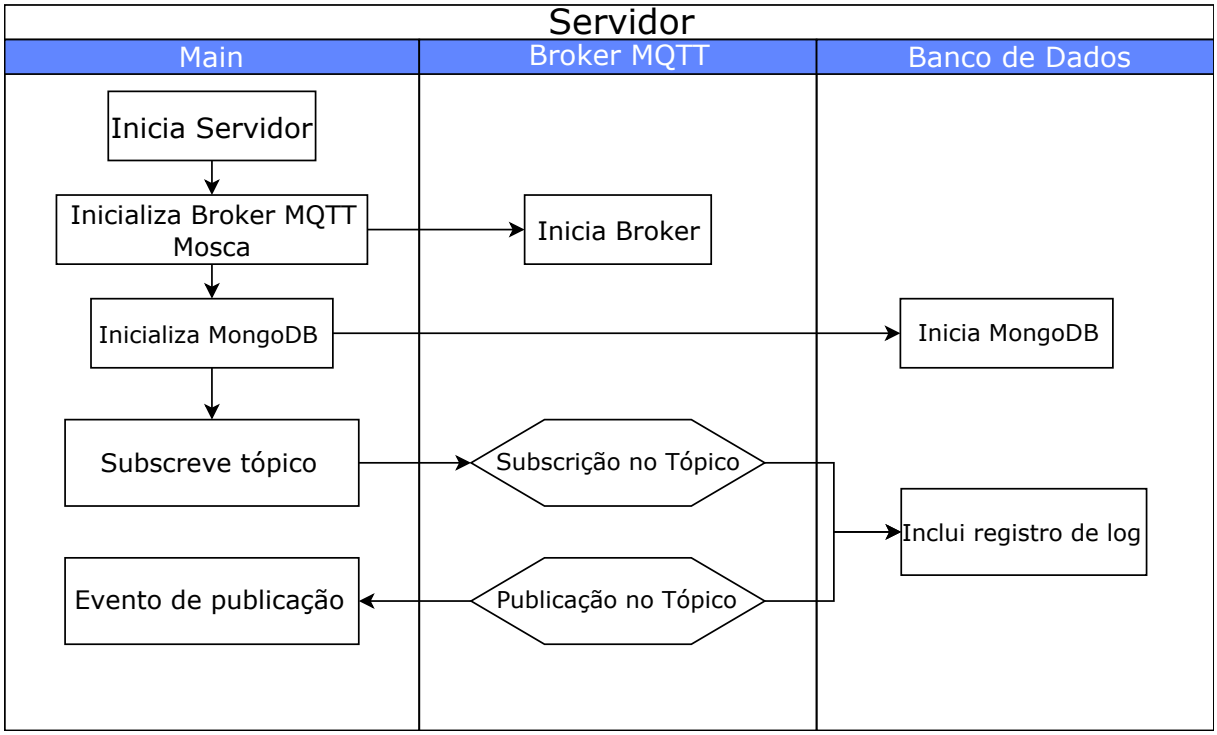
O Código A.5 se refere à implementação para a execução do servidor. A seguir são descritos os pacotes utilizados:

- **Express:** Framework web que fornece mecanismos para gerenciar requisições, rotas, *Uniform Resource Locator* (URLs). Define as configurações da aplicação web como a porta a ser utilizada na aplicação;
- **Express-graphql:** É uma linguagem de consultas de dados para Interface de Programação de Aplicações ou *Application Programming Interface* (APIs);
- **Body-parser:** Permite que clientes externos possam enviar informações para a aplicação em JSON;
- **Mongoose:** Permite modelar os dados da aplicação, fornecendo um mapeamento de objetos do MongoDB e traduzindo os dados para objetos JavaScript para que possam ser utilizados na aplicação;
- **Cors:** Permite a comunicação entre domínios de forma livre;
- **Axios:** Cliente HTTP que permite fazer requisições.

O banco de dados MongoDB foi instalado e executado por meio do Docker (DOCKER, 2019). O MongoDB foi escolhido por ser um banco de dados *open-source* (Código aberto), multiplataforma e que permite a utilização de esquemas em JSON, que é uma forma de tratar os dados.

O diagrama da Figura 19 ilustra a inicialização dos serviços do servidor juntamente com as ações efetuadas pelo *broker* MQTT.

Figura 19 – Diagrama do servidor

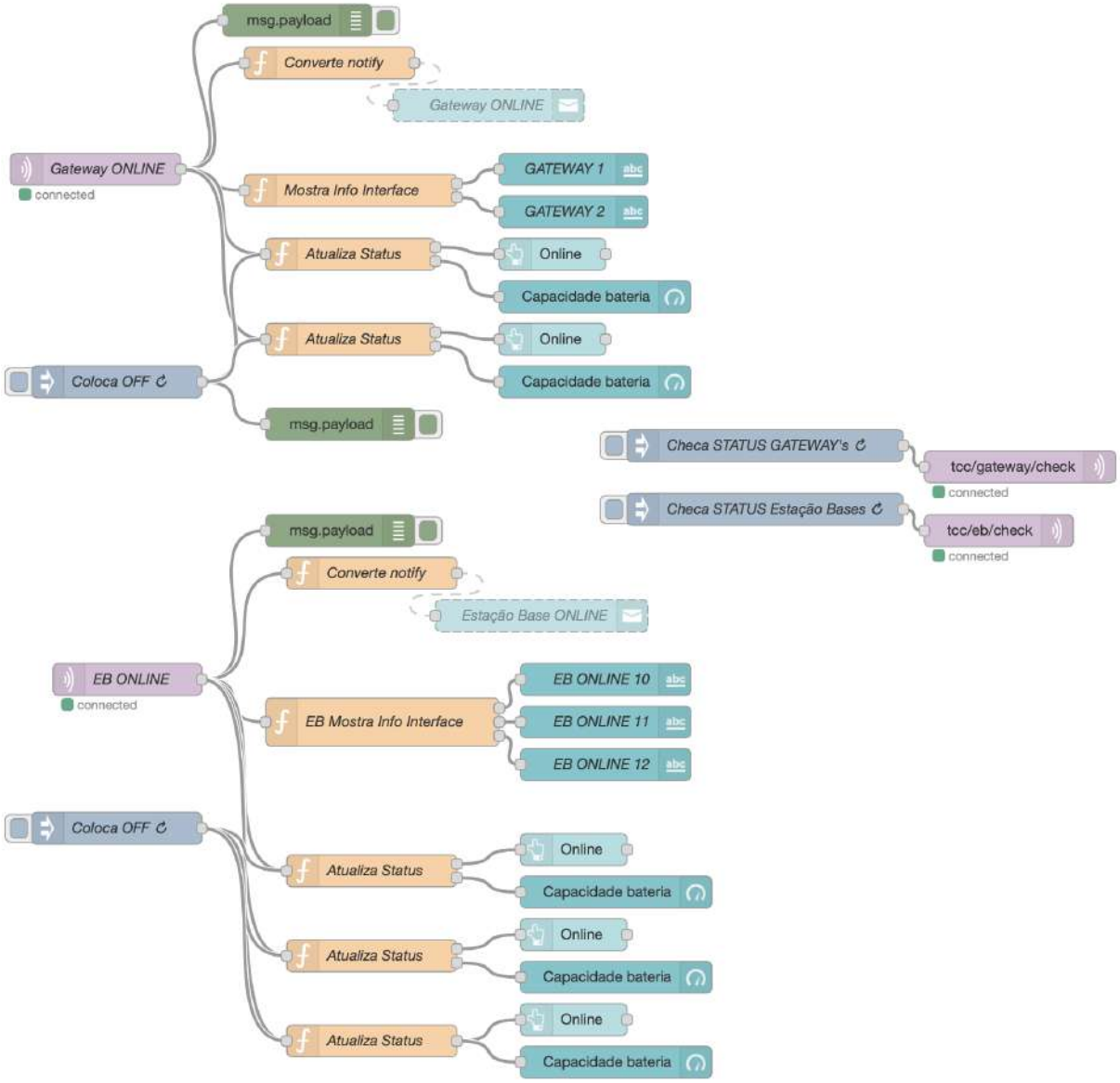


Fonte: Autoria própria.

Para o gerenciamento das informações recebidas pelo *broker* e para prover uma interface para o operador em solo visualizar as informações do *gateway* e das estações base, foi utilizada a ferramente Node-RED (Seção 3.1.1.4). Com ela foi criado um fluxo capaz de gerenciar toda a comunicação do *broker* MQTT, checar o *status* dos *gateways* e estações base e foi disponibilizada uma interface de usuário para o operador ser capaz de verificar as informações.

É necessário inserir no próprio fluxo quantos *gateways* e estações bases forem desejados. Para os testes e para o projeto foram criados no fluxo 2 *gateways* e 3 estações bases. A Figura 20 ilustra o fluxo criado.

Figura 20 – Fluxo criado para o trabalho proposto



Fonte: Autoria própria.

O nó "Gateway ONLINE" se subscreve no tópico *tcc/gateway/status* com o objetivo receber toda a informação dos *gateways* que se conectarem ao *broker*. Ao receber essa informação do *gateway*, a função "Mostra Info Interface" (Código A.6) é utilizada para mostrar o *status* do *gateway* na interface de usuário que é utilizada pelo operador, juntamente com a função "Atualiza Status" (Código A.7).

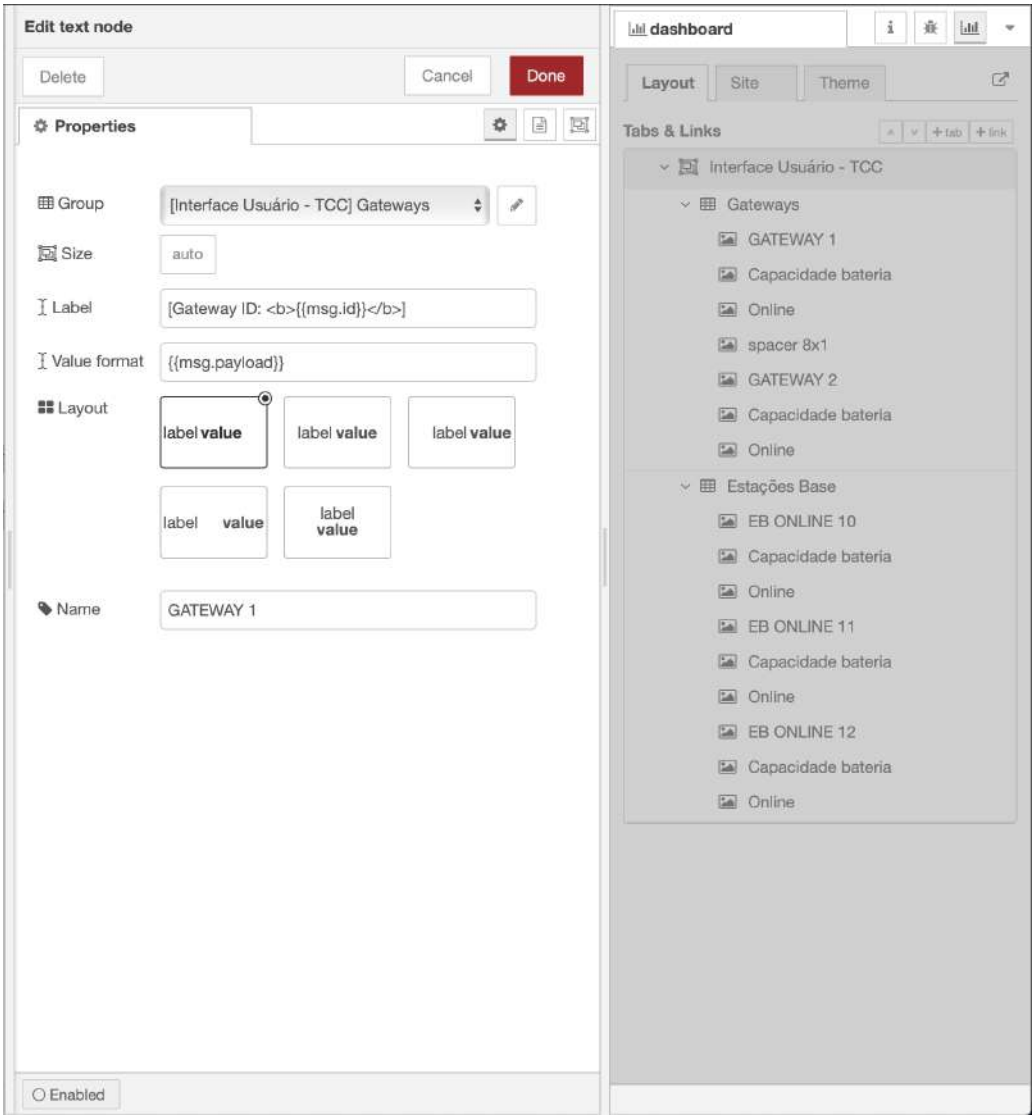
Para a verificação em tempo real do *status* do *gateway* e das estações base, foram criados os nós "Checa STATUS GATEWAY's" e "Checa STATUS Estações Bases" que juntamente com um temporizador envia uma mensagem de checagem nos tópicos *tcc/gateway/check* e *tcc/eb/check*. Os *gateways* estão subscritos nesses tópicos e ao receberem a mensagem enviam um retorno para o *broker* informando o *status* tanto das estações bases como dele próprio, fechando, assim, o "ciclo".

O nó "Coloca OFF" tem o objetivo de limpar o *status* dos dispositivos na interface do usuário, para assim então, receber um novo *status* atual. O mesmo procedimento foi realizado para as estações base (Códigos A.8 e A.9) além de que, assim que um drone pousar na estação, as informações são enviadas para o *broker* e imediatamente recebidas pela aplicação.

4.2.2 Modelagem e Implementação da Interface do Operador

Toda a interface do operador foi modelada na ferramenta do Node-RED (Figura 21).

Figura 21 – Modelagem do *front-end* da interface do usuário utilizando o Node-RED



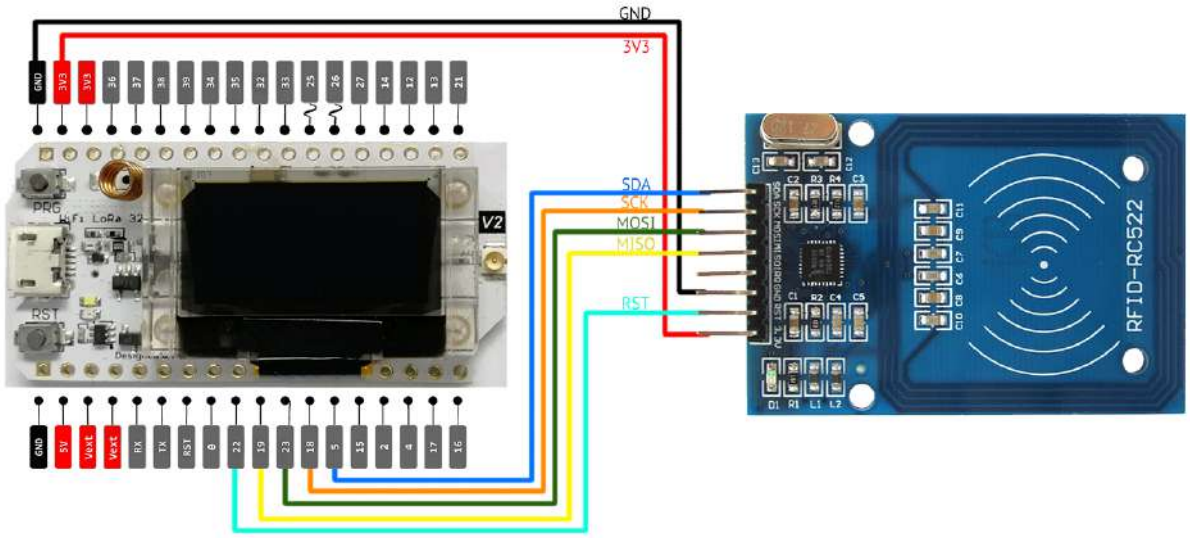
Fonte: Autoria própria.

A ferramenta permite inserir todos os componentes (*labels*, *buttons*, gráficos, notificações) necessários para a visualização do usuário e permite inserir códigos para personalizar a saída. A função "Atualiza Status" (Códigos A.7 e A.9) é acionada assim que uma informação é recebida do *gateway* ou de alguma estação base, atualizando o *status* do botão na interface do operador.

4.2.3 Arquitetura e Modelagem do Microcontrolador

O módulo com o microcontrolador ESP32 foi utilizado tanto nas estações base como no *gateway*. Nas estações base, o módulo (Seção 3.1.2.1) foi ligado diretamente com o leitor RFID RC-522 (Seção 3.1.4) utilizando o protocolo *Serial Peripheral Interface* (SPI), e o próprio módulo conta com um *display* integrado para a visualização das informações recebidas. A Figura 22 ilustra as conexões necessárias entre o módulo e o leitor.

Figura 22 – Ligação do módulo Heltec ESP32 LoRa com o leitor RFID RC-522

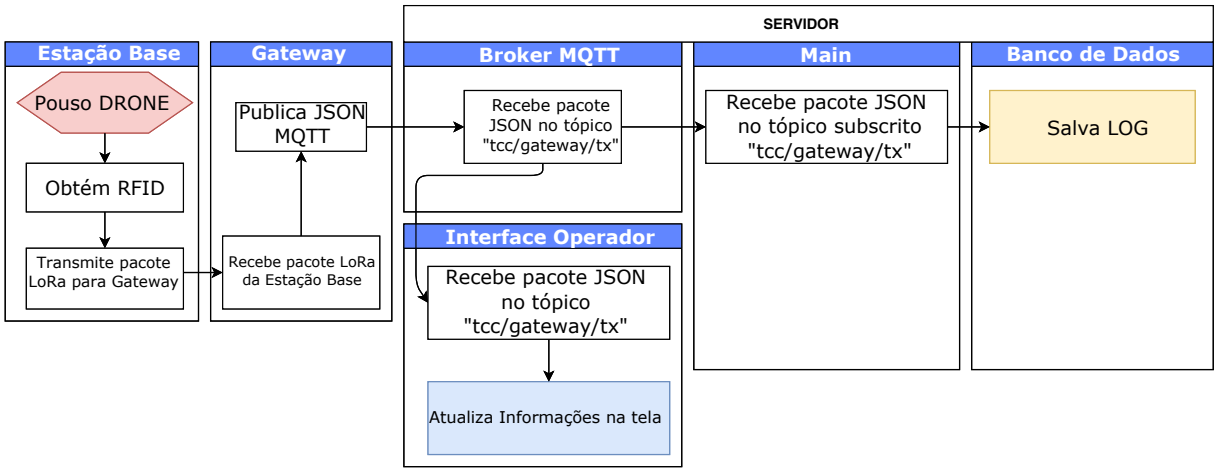


Fonte: Autoria própria.

O microcontrolador deve realizar a leitura da *tag* RFID acoplada ao drone assim que o drone se aproximar da estação. Uma rotina no *firmware* do microcontrolador (Código A.11) deve identificar uma *tag* se aproximando do leitor e efetuar a leitura das informações da *tag*.

Após a leitura da identificação (ID) do drone, essa informação juntamente com o ID da estação base é transmitida para o *gateway* utilizando o protocolo LoRa. O diagrama da Figura 23 ilustra o funcionamento desde o pouso do drone, o processo de salvar um *log* no banco de dados e a atualização da informação na interface do operador.

Figura 23 – Diagrama que ilustra o pouso de um drone na estação base



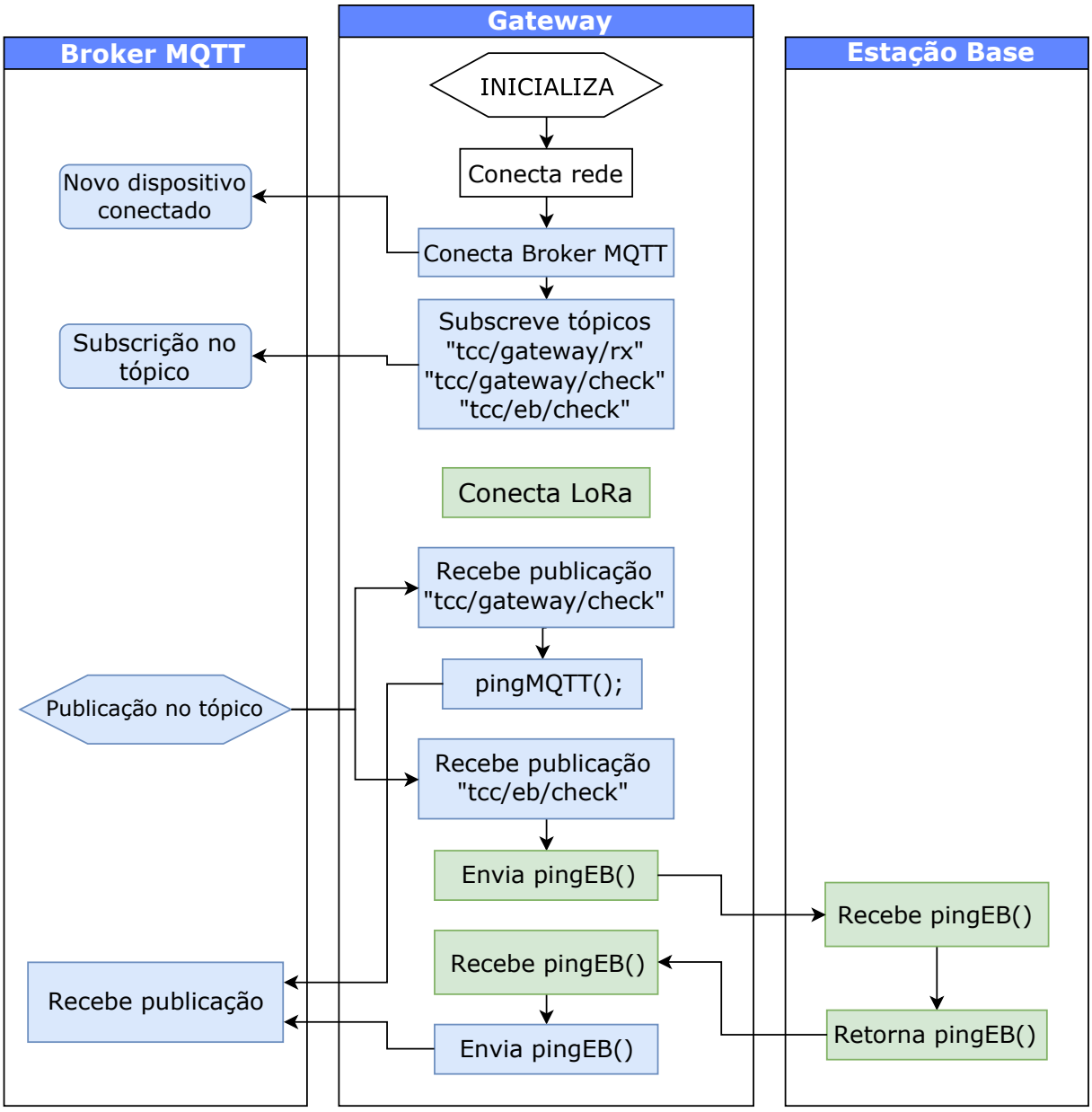
Fonte: Autoria própria.

Assim que o *gateway* receber as informações do ID do drone e da estação base, essas informações são enviadas utilizando o protocolo MQTT por meio de um tópico pré-definido no *firmware* para o *broker* MQTT.

O *gateway* deve se conectar ao *broker* no servidor para permitir a troca de informações utilizando o protocolo MQTT (Seção 2.5.2). A Figura 24 ilusta o funcionamento do *gateway* desde a inicialização, subscrição nos tópicos, recebimento de informações dos tópicos subscritos à checagem do *status* tanto do *gateway* como das estações base. Os nós na cor "azul" se referem à toda a comunicação utilizando o protocolo MQTT, e os nós na cor "verde" à toda a comunicação utilizando o protocolo LoRa.

O *gateway* é o responsável por verificar em um tempo pré-definido o *status* das estações base. A função *pingEB()* envia uma mensagem para as estações base a fim de obter um retorno das mesmas, informando assim que estão ativas e funcionando normalmente. O código no Apêndice A.10 foi utilizado no *gateway* e executa tudo que foi descrito anteriormente.

Figura 24 – Diagrama que ilustra a inicialização e funcionamento do *gateway*



Fonte: Autoria própria.

O *broker* MQTT está subscrito nos tópicos pré-definidos e assim que receber uma informação no tópico, o código implementado no servidor faz a leitura e o tratamento dessas informações recebidas e as armazena em um banco de dados MongoDB.

4.3 Apresentação

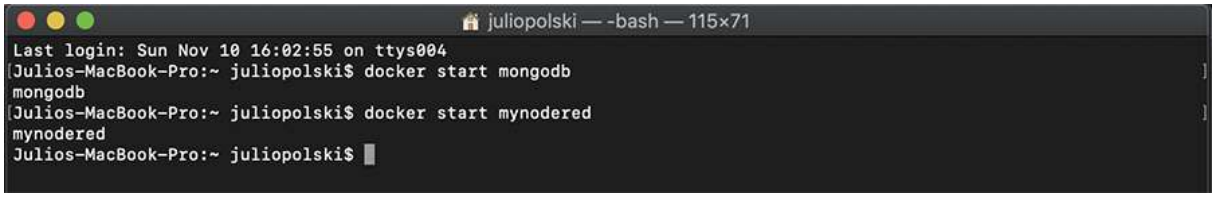
4.3.1 Apresentação do Servidor

Para o servidor que é responsável por executar e manter o *broker* MQTT e a interface do operador em solo, foi utilizado um notebook.

Para inicializar o servidor, primeiramente é necessário inicializar o banco de dados Mon-

goDB e o Node-RED utilizando o Docker (Seção 3.1.1.2). Para isso, é necessário abrir o *prompt* de comando ou o terminal e executar o comando *"docker start <nome_definido_para_a_aplicação>"* (Figura 25).

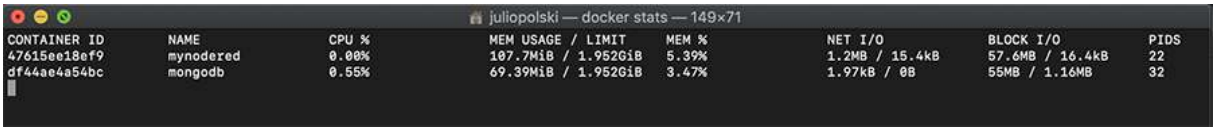
Figura 25 – Comandos para inicializar o banco de dados e o Node-RED



Fonte: Autoria própria.

O comando *"docker stats"* mostra os serviços que estão sendo executados no Docker (Figura 26).

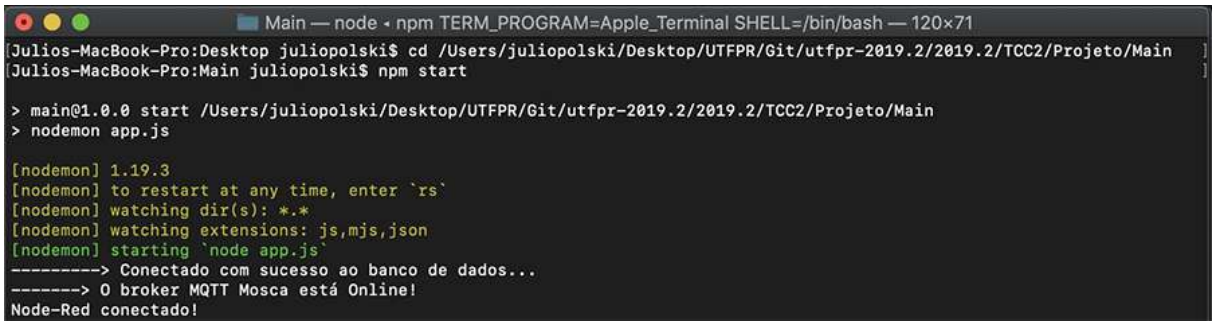
Figura 26 – Serviços sendo executados no Docker



Fonte: Autoria própria.

Depois que o banco de dados e o Node-RED forem executados, é possível inicializar a aplicação desenvolvida para o servidor (código no Apêndice A.5). Utilizando o terminal, é necessário acessar a pasta do projeto em que o código está inserido e executar o comando *"npm start"* (Figura 27).

Figura 27 – Inicialização do servidor



Fonte: Autoria própria.

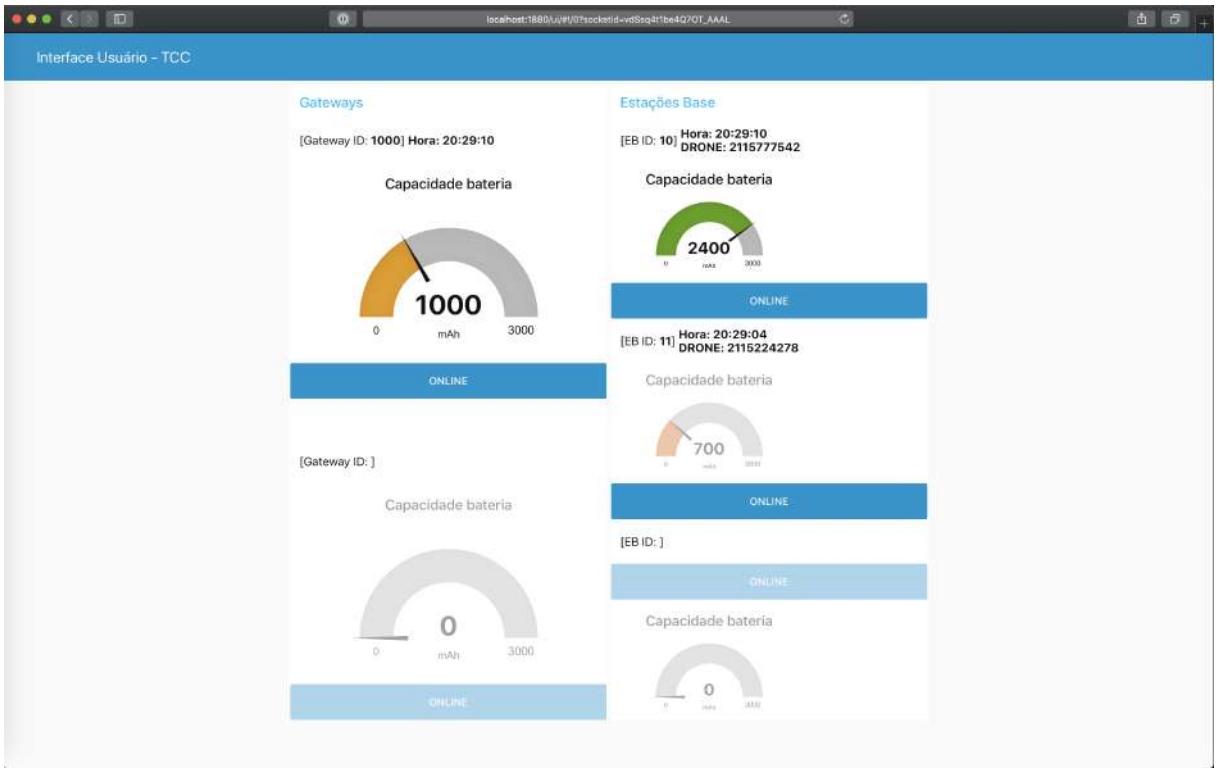
A partir do momento em que o servidor for executado, a aplicação está em pleno funcionamento. A seção a seguir se refere ao funcionamento da interface do operador, em que é possível receber todas as informações necessárias tanto dos *gateways* como das estações base.

4.3.2 Apresentação da Interface do operador

Com a inicialização do servidor, a interface desenvolvida utilizando o Node-RED está disponível para o operador acessar. É possível acessar de qualquer dispositivo que tenha um navegador, podendo ser um computador, *smartphone*, *tablet*, *smart TV*, entre outros.

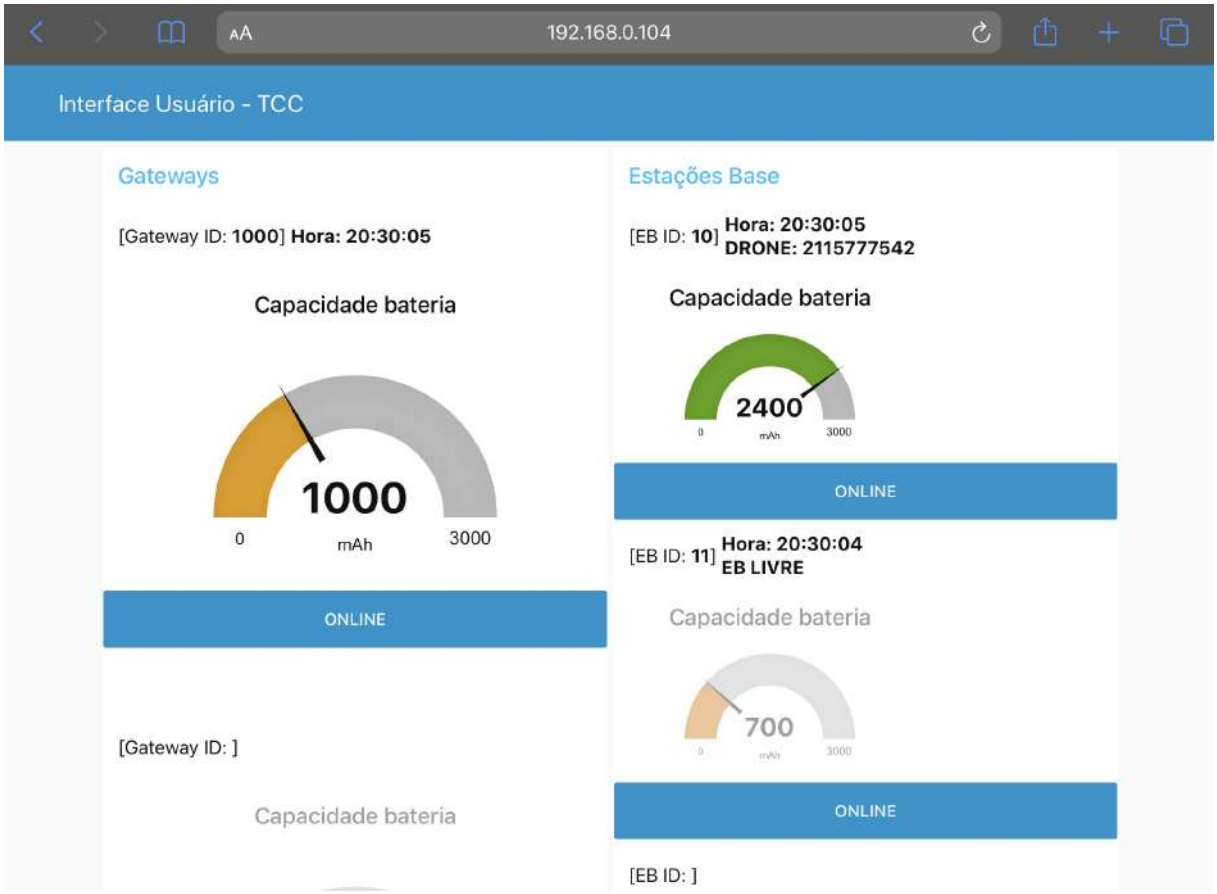
As Figuras 28 e 29 ilustram a interface em um computador e em um *tablet* respectivamente. Em ambas as figuras é possível verificar que um *gateway* está *online* e mostra uma das várias possibilidades de informações que podem ser obtidas das estações como capacidade de bateria.

Figura 28 – Interface do operador sendo executada em um computador



Fonte: Autoria própria.

Figura 29 – Interface do operador sendo executada em um *tablet*



Fonte: Autoria própria.

Como as estações estão sendo alimentadas por um *PowerBank* (carregador externo), o valor do nível da bateria da estação é somente ilustrativo. Os valores reais recebidos e apresentados na interface é o ID do *gateway*, a hora do momento que recebeu a informação e o *status*. Já para as estações base, as mesmas informações são apresentadas juntamente com o *status* do drone. Caso tenha um drone na estação base é mostrado o ID do drone (definido pela tag RFID), e caso não tenha drone, a mensagem "EB LIVRE" é apresentada.

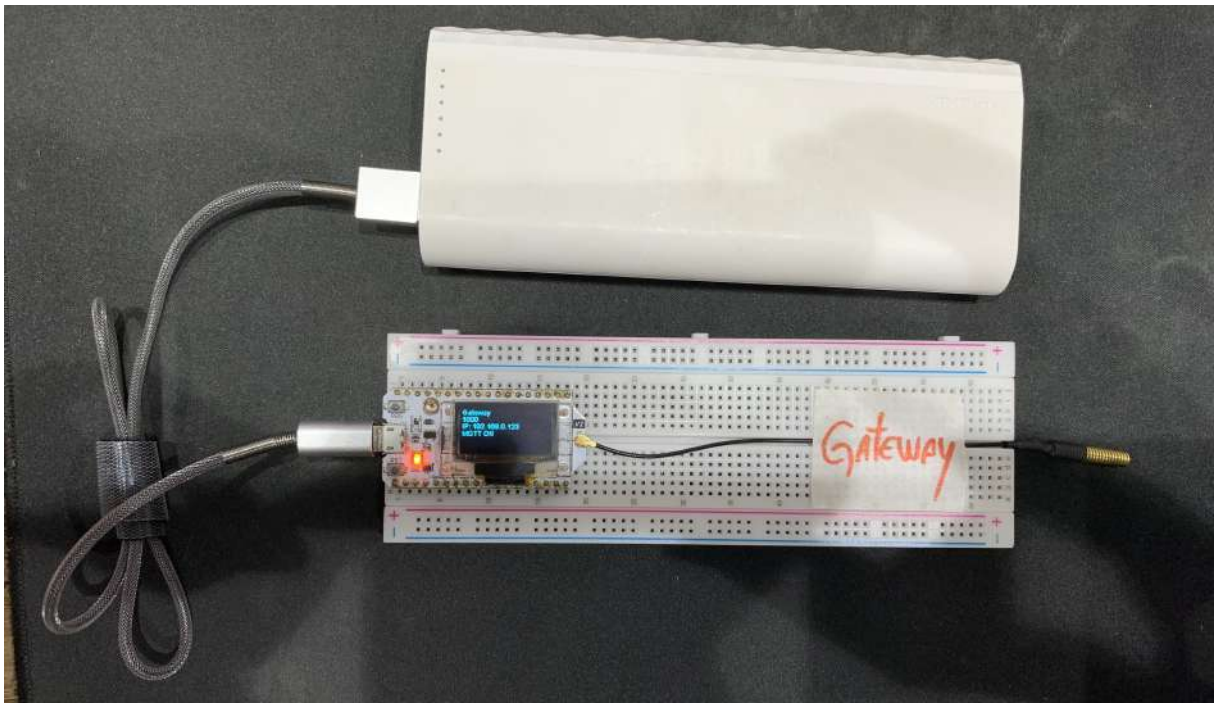
4.3.3 Apresentação do protótipo

O módulo do *gateway* é composto pelos seguintes componentes:

- 1x protoboard;
- 1x módulo Heltec ESP32 LoRa.

A Figura 30 exibe o protótipo utilizado para o *gateway*, que utiliza basicamente tudo que o módulo Heltec pode oferecer, como conexão Wi-Fi, tecnologia LoRa e um *display* integrado.

Figura 30 – Módulo utilizado no *gateway*



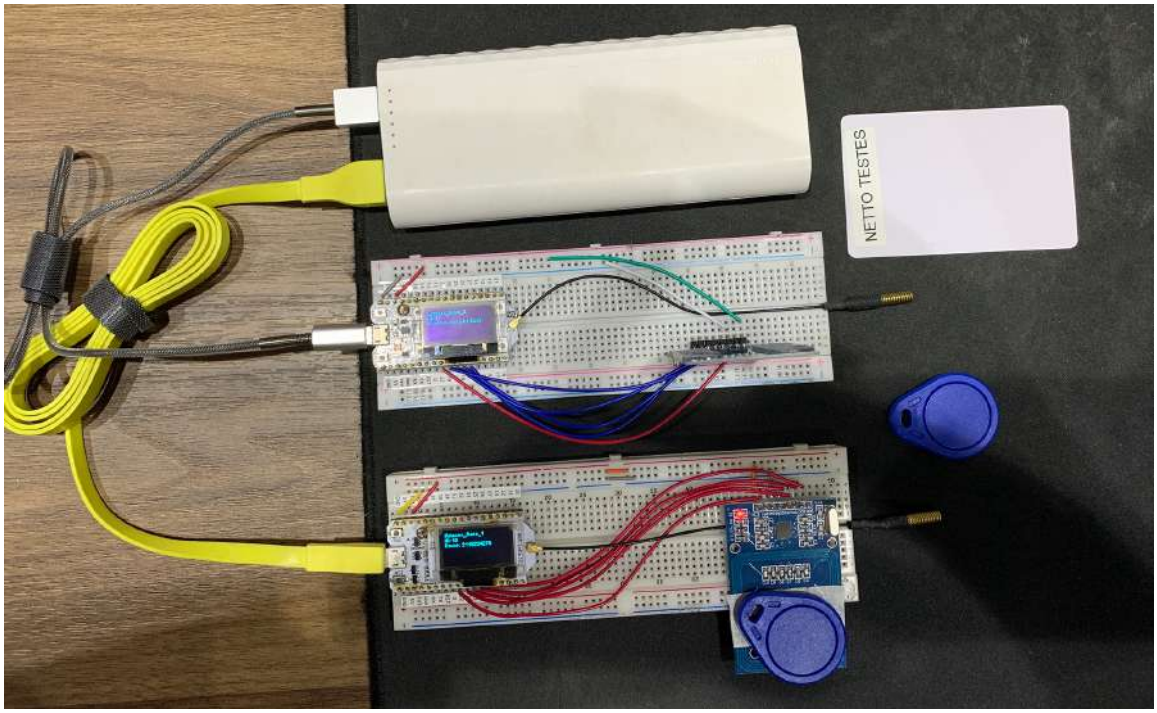
Fonte: Autoria própria.

O módulo da estação base é composto pelos seguintes componentes:

- 1x protoboard;
- 1x módulo Heltec ESP32 LoRa;
- 1x leitor RFID RC522;
- 1x *tag* RFID (cartão ou chaveiro);
- Jumpers para conexão.

A Figura 31 exibe o protótipo utilizado para as estações base.

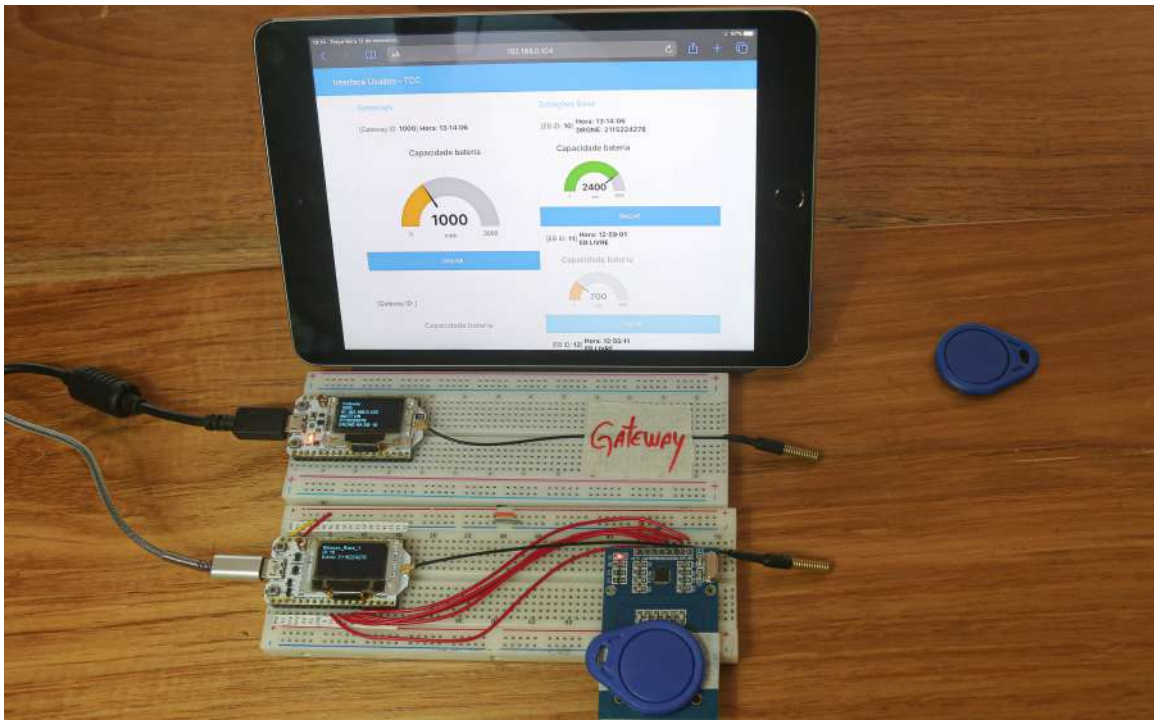
Figura 31 – Módulos e *tags* RFID utilizados nos testes para as estações bases



Fonte: Autoria própria.

A Figura 32 ilustra todo o conjunto de interface do operador, *gateway* e estação base com uma *tag* RFID referenciando um "drone" pousado em funcionamento.

Figura 32 – Conjunto da Interface do operador, *gateway* e estação base em funcionamento.



Fonte: Autoria própria.

Além do display do módulo exibir a ID do *gateway* e das estações, é mostrado o ID do drone pousado na estação base especificada.

5 Conclusão

Este trabalho objetivou desenvolver um sistema de comunicação capaz de atuar, sensoriar, processar e se comunicar de forma eficiente com um longo alcance e baixo consumo energético entre estações base. Por possuírem uma gama de aplicações, principalmente se tratando de Internet das Coisas ou *Internet of Things* (IoT) as redes LPWAN se mostraram ideais e estão se tornando uma opção cada vez mais adotada comercialmente.

Inicialmente o objetivo era obter um alcance de até 5 quilômetros entre a estação base e o *gateway*, distância essa suficiente para atender aos requisitos do trabalho proposto. A tecnologia LoRa mostrou-se capaz de atender esses requisitos, sendo possível a troca de informações a até 10 quilômetros de distância de forma bastante aceitável. Os testes foram realizados no ambiente real de uso, ou seja, na área rural da cidade de Renascença e na área urbana na cidade de Francisco Beltrão.

O módulo da Heltec foi o escolhido para os testes por possuir toda a tecnologia necessária para prover a comunicação tanto utilizando o protocolo LoRa como o MQTT, além de possuir um baixo consumo energético (Mikhaylov; Juha Petaejaejaervi; Haenninen, 2016). O sistema foi desenvolvido para operar com inúmeras estações bases, sendo necessária a configuração de cada estação base e *gateways* individualmente via código, a fim de definir uma ID para cada estação de acordo com a interface de usuário desenvolvida.

A implementação do trabalho foi facilitada devido aos diversos materiais e bibliotecas existentes para as tecnologias utilizadas, como os módulos da Heltec e RFID. A utilização do Node-RED foi fundamental para o sucesso do trabalho, já que a aplicação permite configurar toda a comunicação MQTT entre a interface do operador e os dispositivos, assim como implementar a própria interface diretamente no fluxo.

Da forma que a tecnologia está crescendo, muito rapidamente novas tecnologias, bibliotecas e ferramentas surgirão para facilitar ainda mais a forma de comunicação do protocolo LoRa, possibilitando otimizar de forma crescente a comunicação entre os dispositivos, permitindo alcançar distâncias ainda maiores.

5.1 Trabalhos Futuros

O trabalho desenvolvido se mostrou promissor, porém algumas implementações serão necessárias a fim de obter um produto final, como:

- **Utilização de carregamento solar nas estações base** para atender o consumo energético do microcontrolador e o recarregamento do drone;
- Caso as estações base não possuam Linha de Visada (LoS), será necessário **utilizar uma antena melhor** ou **aumentar a altura da antena**;
- **Utilização de interrupções programadas** no código das estações base visando diminuir

o consumo energético das mesmas;

- Implementação de **criptografia/segurança** adicional em toda a comunicação.

Além da utilização para estações base para grupos de drones, o resultado da realização deste trabalho permite extrapolar o escopo inicialmente definido e ser utilizado para diversas outras implementações, por exemplo, sensoriamento para área de segurança, agrícola, de casas e planta de uma fábrica.

A Códigos utilizados

Código A.1 – Código para o teste de distância do módulo transmissor

```
#include "heltec.h" // Biblioteca do modulo Heltec ESP32 LoRa
#define FREQ    433E6 // Frequencia utilizada para comunicacao LoRa: 433 MHz

unsigned int contador = 0;

void setup()
{
  Heltec.begin(true, true, true, true, FREQ); // Inicializa o modulo (DisplayEnable, LoRaEnable, SerialEnable, PABOOST, Frequencia)
  Heltec.display->init(); // Inicializa o Display
  Heltec.display->flipScreenVertically(); // Coloca a orientacao do Display na vertical
  Heltec.display->setFont(ArialMT_Plain_10); // Define a fonte e o tamanho
  Heltec.display->setTextAlignment(TEXT_ALIGN_LEFT); // Define o alinhamento
  delay(1000); // Aguarda 1 segundo
}

void loop()
{
  Heltec.display->clear(); // Limpa a tela
  Heltec.display->drawString(0, 0, "Enviando Pacote: "); // Define na tela
  Heltec.display->drawString(90, 0, String(contador)); // Define na tela
  Heltec.display->display(); // Mostra na tela

  LoRa.beginPacket(); // Inicializa o envio do pacote
  LoRa.setTxPower(14,RF_PACONFIG_PASELECT_PABOOST); // Define os parametros da comunicacao LoRa
  LoRa.print("Envio Pacote "); // Informacao a ser adicionada ao pacote
  LoRa.print(contador); // Informacao a ser adicionada ao pacote
  LoRa.endPacket(); // "Finaliza" o pacote e envia

  contador++; // Incrementa o contador
  delay(1000); // Aguarda 1 segundo
}
```

Código A.2 – Código para o teste de distância do módulo receptor

```
#include "heltec.h" // Biblioteca do modulo Heltec ESP32 LoRa
#define FREQ    433E6 // Frequencia utilizada para comunicacao LoRa: 433 MHz

String rssi = "RSSI --"; // Variavel para armazenar a potencia do sinal recebido
String tamPacote = "--"; // Variavel para armazenar o tamanho do pacote recebido
String pacote; // Variavel para armazenar o pacote recebido

void LoRaData(){ // Funcao para mostrar as informacoes do pacote recebido
  Heltec.display->clear();
  Heltec.display->setTextAlignment(TEXT_ALIGN_LEFT);
  Heltec.display->setFont(ArialMT_Plain_10);
  Heltec.display->drawString(0 , 15 , "Recebido "+ tamPacote + " bytes");
  Heltec.display->drawStringMaxWidth(0 , 26 , 128, pacote);
  Heltec.display->drawString(0, 0, rssi);
  Heltec.display->display();
}

void trataPacote(int tamanhoPacote) {
  pacote = ""; // Variavel para armazenar as informacoes do pacote
  tamPacote = String(tamanhoPacote,DEC); // Converte o tamanho do pacote para String

  for (int i = 0; i < tamanhoPacote; i++) { // Recebe as informacoes do pacote byte a byte
    pacote += (char) LoRa.read();
  }

  rssi = "RSSI " + String(LoRa.packetRssi(), DEC); // Armazena a potencia do sinal recebido em RSSI
  LoRaData(); // Mostra as informacoes no Display
}

void setup()
{
  Heltec.begin(true, true, true, true, FREQ); // Inicializa o modulo
  Heltec.display->init(); // Inicializa o Display
  Heltec.display->flipScreenVertically(); // Coloca a orientacao do Display na vertical
  Heltec.display->setFont(ArialMT_Plain_10); // Define a fonte e o tamanho
  Heltec.display->setTextAlignment(TEXT_ALIGN_LEFT); // Define o
```

```
    alinhamento
37
38   delay(1000); // Aguarda 1 segundo
39   LoRa.receive(); // Ativa o recebimento LoRa
40 }
41
42 void loop()
43 {
44   int tamanhoPacote = LoRa.parsePacket(); // Variavel para armazenar o
      tamanho do pacote recebido
45   if (tamanhoPacote) { // Se recebeu algum pacote
46     trataPacote(tamanhoPacote);
47   }
48   delay(10);
49 }
```

Código A.3 – Código para o teste de perda de pacotes do módulo transmissor

```
1 #include "heltec.h"
2 #include <CayenneLPPDecode.h>
3
4 #define BAND    433E6           // Frequencia do modulo de 433 MHz
5
6 unsigned int contador = 0;      // Contador utilizado para verificar a
    perda de pacotes
7
8 void setup() {
9   Serial.begin(115200);
10  Heltec.begin(true, true, true, true, BAND);      // Ativa os
     componentes do modulo Heltec
11
12  Heltec.display->init();                          // Inicializa o
     Display
13  Heltec.display->flipScreenVertically();          // Define o Display
     na vertical
14  Heltec.display->setFont(ArialMT_Plain_10);       // Define a fonte
     utilizada
15  Heltec.display->clear();                         // Limpa o Display
16
17  Heltec.display->drawString(0, 0, "Teste de PERDA !");              //
      Imprime no Display
18  Heltec.display->drawString(0, 10, "Envio inicia em 5 segundos!");  //
      Imprime no Display
19  Heltec.display->display();                                         //
      Mostra
20
21  LoRa.receive();       // Inicia o recebimento de pacotes LoRa
```

```
22   delay(5000);           // Aguarda 5 segundos para inicializar a
      transmissao
23 }
24
25 void loop() {
26   if(contador <= 200){
27     Heltec.display->clear();                                    // Limpa o
      Display
28     Heltec.display->drawString(0, 0, "> TESTE DE PERDA <");    // Imprime
      no Display
29     Heltec.display->drawString(0, 15, "Enviando pacote: ");    // Imprime
      no Display
30     Heltec.display->drawString(90, 15, String(contador));      // Imprime
      no Display
31     Heltec.display->display();                                  // Mostra
32
33
34     CayenneLPP lpp(128);  // Iniciliza uma instancia do Cayenne para
     armazenas as informacoes em JSON
35     lpp.addDigitalInput(1, contador);     // Insere o contador na
     primeira posicao do JSON
36     lpp.addDigitalOutput(2, 1);           // Insere informacoes
     aleatorias para preencher o pacote de 52 bytes
37     lpp.addAnalogInput(3, 1.23f);
38     lpp.addAnalogOutput(4, 3.45f);
39     lpp.addLuminosity(5, 20304);
40     lpp.addPresence(6, 1);
41     lpp.addTemperature(7, 26.5f);
42     lpp.addDigitalOutput(8, 1);
43     lpp.addAnalogInput(9, 1.23f);
44     lpp.addAnalogOutput(10, 3.45f);
45     lpp.addAnalogInput(11, 1.23f);
46     lpp.addAnalogOutput(12, 3.45f);
47     lpp.addAnalogOutput(13, 3.45f);
48     lpp.addAnalogOutput(14, 3.45f);
49
50     LoRa.beginPacket();             // Inicializa o envio do pacote
51     LoRa.setTxPower(14,RF_PACONFIG_PASELECT_PABOOST);   // Define o SF =
      14
52     LoRa.write(lpp.getBuffer(), lpp.getSize());         // Envia o
     pacote
53     LoRa.endPacket();               // Finaliza o envio
54
55     contador++;                     // Incrementa o contador
56
57     delay(2000);                    // Aguarda 2 segundos
58   } else {
```

```
59      Heltec.display->clear();                                    // Limpa o
        Display
60      Heltec.display->drawString(0, 0, "> TESTE DE PERDA <");     // Imprime
        no Display
61      Heltec.display->drawString(0, 20, "Envio FINALIZADO!");     // Imprime
        no Display
62      Heltec.display->drawString(0, 35, "Pacotes enviados: ");    // Imprime
        no Display
63      Heltec.display->drawString(0, 50, String(contador - 1));    // Imprime
        no Display
64      Heltec.display->display();      // Mostra
65    }
66
67  }
```

Código A.4 – Código para o teste de perda de pacotes do módulo receptor

```
1  #include "heltec.h"
2  #include <CayenneLPPDecode.h>
3  #include <string>
4
5  #define BAND    433E6
6
7  unsigned int contador = 0;
8  unsigned int pacotesPerdidos = 0;
9  unsigned int pacotesRecebidos = 0;
10 int valor = -1;
11 int valorAnterior = -1;
12 unsigned int mediaRSSI = 0;
13
14 String rssi = "RSSI --";
15 String packSize = "--";
16 String packet;
17
18 void LoRaData(){
19   Heltec.display->clear();
20   Heltec.display->drawString(0, 0, "> TESTE DE PERDA <");
21   Heltec.display->drawString(0, 10, rssi);
22   Heltec.display->drawString(0 , 20 , "Recebido " + packSize + " bytes")
       ;
23   Heltec.display->drawStringMaxWidth(0 , 30, 128, "Pacote: " + packet);
24   Heltec.display->drawString(0 , 40 , "Qtd. Recebida " + String(
       pacotesRecebidos, DEC));
25   Heltec.display->drawString(0 , 50 , "Qtd. Perdidos " + String(
       pacotesPerdidos, DEC));
26
27   Heltec.display->display();
```

```
28 }
29
30 void recebePacote(int tamanhoPacote) {
31   DynamicJsonDocument jsonBuffer(512);
32   CayenneLPPDecode lppd;
33
34   JsonObject root = jsonBuffer.to<JsonObject>();
35
36   packet = "";                                    // Variavel que
      armazena as informacoes do pacote recebido
37   packSize = String(tamanhoPacote,DEC);           // String para
      mostrar o tamanho do pacote recebido
38   rssi = "RSSI " + String(LoRa.packetRssi(), DEC); // String para
      mostrar o valor RSSI do pacote recebido
39
40   mediaRSSI = mediaRSSI + ((-1)*LoRa.packetRssi()); // Incrementa todo
      valor RSSI recebido para no final calcular a media
41
42   while (LoRa.available()) {
43     lppd.write(LoRa.read());
44   }
45
46   Serial.print("Receive: ");
47   Serial.println();
48
49   if (lppd.isValid())
50   {
51     lppd.decode(root);
52
53     serializeJsonPretty(root, Serial);
54     Serial.println();
55   }
56
57   Heltec.display->clear();
58   Heltec.display->drawString(0, 0, "> TESTE DE PERDA <");
59   Heltec.display->drawString(0 , 10 , "Pacote Recebido...");
60   Heltec.display->drawString(0, 20, "Valor do Contador:");
61   Heltec.display->drawString(0 , 30 , root["digital_in_1"]);
62   Heltec.display->drawString(0, 40, "Qtd. Perdidos " + String(
      pacotesPerdidos, DEC));
63   Heltec.display->drawString(0, 50, "Tam.Pacote " + packSize + "bytes");
64   Heltec.display->display();
65
66   valor = root["digital_in_1"];              // Transforma o valor do
      contador para inteiro
67   Serial.println("VALOR: ");
68   Serial.println(valor);
```

```
69    Serial.println("VALOR Anterior: ");
70    Serial.println(valorAnterior);
71
72    if(valor == valorAnterior + 1){           // Se o valor atual for o
       anterior + 1, nao houve perda de pacote
73      valorAnterior++;
74    }else{
75      if(valor < valorAnterior){
76        Heltec.display->clear();
77        Heltec.display->drawString(0, 0, "> TESTE DE PERDA <");
78        Heltec.display->drawString(0 , 10 , "Transmissor Reiniciou...");
79        Heltec.display->display();
80      }else{
81        if((valor - valorAnterior) > 2)
82          pacotesPerdidos += (valor - valorAnterior - 1);
83        else
84          pacotesPerdidos++;
85
86        valorAnterior = valor;
87      }
88    }
89
90 }
91
92 void setup() {
93    Heltec.begin(true, true, true, true, BAND);
94
95    Heltec.display->init();
96    Heltec.display->flipScreenVertically();
97    Heltec.display->setFont(ArialMT_Plain_10);
98    Heltec.display->setTextAlignment(TEXT_ALIGN_LEFT);
99    Heltec.display->clear();
100
101   Heltec.display->drawString(0, 0, "Teste de PERDA !");
102   Heltec.display->display();
103
104   delay(100);
105   LoRa.receive();
106 }
107
108 void loop() {
109   int tamPacote = LoRa.parsePacket();
110
111   if (tamPacote) {
112     recebePacote(tamPacote);
113     pacotesRecebidos++;           // Incrementa toda vez que recebe um
       pacote
```

```
  }

  delay(10);
}
```

Código A.5 – Código de configuração do servidor e execução do broker

```
// Importando os pacotes utilizados
const express = require('express');
const bodyParser = require('body-parser');
const graphqlHttp = require('express-graphql');
const mongoose = require('mongoose');
const cors = require('cors');
const axios = require('axios');

const graphQlSchema = require('./graphql/schema/index');
const graphQlResolvers = require('./graphql/resolvers/index');

const mosca = require('mosca');

//Cria um objeto express para ser utilizado na aplicacao - Iniciando o APP
const app = express();

// Configuracao do MOSCA Broker MQTT
const ascoltatore = {
    type: "mongo",
    url: "mongodb://127.0.0.1/moscaTCC",
    pubsubCollection: "ascoltatori",
    mongo: {}
};

const moscaSettings = {
    port: 1883,
    id: "mosca",
    backend: ascoltatore,
    persistence: {
        factory: mosca.persistence.Mongo,
        url: "mongodb://127.0.0.1/moscaTCC"
    }
};

const moscaServer = new mosca.Server(moscaSettings);
moscaServer.on("ready", setup);

function setup(){
    console.log("O broker MQTT (Mosca) esta funcionando!");
```

```
40 }
41
42
43 app.use(function (req, res, next) {
44     res.setHeader('Access-Control-Allow-Origin', 'http://127.0.0.1:3000'
       );
45     next();
46 });
47
48 //Para utilizar as funcionalidades do bodyParser para separar as
       informacoes em JSON
49 app.use(bodyParser.json());
50
51 app.get('/', (req, res, next) => {
52     res.send('APP Funcionando!');
53 })
54
55 app.use(cors());
56
57 app.use('/graphql', graphqlHttp({
58     schema: graphQlSchema,
59     rootValue: graphQlResolvers,
60     graphiql: true
61 }));
62
63 mongoose
64     .connect('mongodb://127.0.0.1/tcc', { useNewUrlParser: true,
       useUnifiedTopology: true })
65     .then(() => {
66         console.log("---------> Conectado com sucesso ao banco de dados
       ...");
67         /*
68             Porta para a pagina a ser utilizada na aplicacao
69             Se conectar no banco, inicia o server...
70         */
71         app.listen(3000);
72     }).catch(err => {
73         console.log(err);
74     });
75
76 /*
77     ------------------------------------------------------------
78                 Requisicoes do MOSCA (Broker MQTT)
79 */
80 // Dispara quando um cliente se conecta
81 moscaServer.on("clientConnected", function(packet) {
82     const log = {
```

```
        log_type: "connect",
        log_client: packet.id
    };
    // Cadastra o log do cliente no banco de dados
    cadastrarBase(log.log_client);
    console.log(log.log_client + " conectado!");
});

// Dispara quando um cliente se desconecta
moscaServer.on("clientDisconnected", function(packet) {
    const log = {
        log_type: "disconnect",
        log_client: packet.id
    };
    console.log(log.log_client + " desconectado!");
});
/*
            <<END>> Configuracao do MOSCA Broker MQTT
    ------------------------------------------------------------
*/
```

Código A.6 – Código responsável por mostrar na UI o status do gateway

```
function addZero(i) {
  if (i < 10) {
    i = "0" + i;
  }
  return i;
}

var today = new Date();
var time = addZero(today.getHours()) + ":" + addZero(today.getMinutes())
    + ":" + addZero(today.getSeconds());

mensagem = "Hora: " + time;

var msg1 = {payload:null};
var msg2 = {payload:null};

if(msg.payload.ID_GATEWAY == 1000){
    msg1 = {payload:mensagem, id:msg.payload.ID_GATEWAY};
}
else if(msg.payload.ID_GATEWAY == 1001){
    msg2 = {payload:mensagem, id:msg.payload.ID_GATEWAY};
}

return [msg1, msg2];
```

Código A.7 – Código responsável por mostrar na UI o status do gateway no botão STATUS

```
var today = new Date();
var time = today.getHours() + ":" + today.getMinutes() + ":" + today.
    getSeconds();

var msg1 = {enabled:false, payload: "Offline"};

if(msg.payload.ID_GATEWAY == 1000){
    msg1 = {enabled:true};
}else if(msg.payload.limpa){
    msg1 = {enabled:false};
}
else{
    msg1 = null;
}

return msg1;
```

Código A.8 – Código responsável por mostrar na UI o status da estação base

```
function addZero(i) {
  if (i < 10) {
    i = "0" + i;
  }
  return i;
}

var today = new Date();
var time = addZero(today.getHours()) + ":" + addZero(today.getMinutes())
    + ":" + addZero(today.getSeconds());

mensagem = "Hora: " + time;

var msg1 = {payload:null};
var msg2 = {payload:null};
var msg3 = {payload:null};

// node.warn(msg);

if(msg.payload.ID_EB == 10){
    if(msg.payload.DRONE_POUSADO){
        mensagem += "<br /> DRONE: " + msg.payload.ID_DRONE;
    }else{
        mensagem += "<br /> EB LIVRE";
    }
```

```
    msg1 = {payload:mensagem, id:msg.payload.ID_EB};
}
else if(msg.payload.ID_EB == 11){
    if(msg.payload.DRONE_POUSADO){
        mensagem += "<br /> DRONE: " + msg.payload.ID_DRONE;
    }else{
        mensagem += "<br /> EB LIVRE";
    }
    msg2 = {payload:mensagem, id:msg.payload.ID_EB};
}
else if(msg.payload.ID_EB == 12){
    if(msg.payload.DRONE_POUSADO){
        mensagem += "<br /> DRONE: " + msg.payload.ID_DRONE;
    }else{
        mensagem += "<br /> EB LIVRE";
    }
    msg3 = {payload:mensagem, id:msg.payload.ID_EB};
}

return [msg1, msg2, msg3];
```

Código A.9 – Código responsável por mostrar na UI o status da estação base no botão STATUS

```
var today = new Date();
var time = today.getHours() + ":" + today.getMinutes() + ":" + today.
    getSeconds();

var msg1 = {enabled:false, payload: "Offline"};

// node.warn(msg)

if(msg.payload.ID_EB == 10){
    msg1 = {enabled:true};
}else if(msg.payload.limpa){
    msg1 = {enabled:false};
}
else{
    msg1 = null;
}

return msg1;
```

Código A.10 – Código utilizado para o *gateway*

```
#include <heltec.h>
#include <SPI.h>
```

```
3  #include <Wire.h>
4  #include <string.h>
5
6  // Para utilizar o MQTT
7  #include <WiFi.h>
8
9  #define MQTT_SOCKET_TIMEOUT 30000
10 #define MQTT_KEEPALIVE 30000
11 #include<PubSubClient.h>
12
13 // Para a comunicacao Estacoes Base / Gateway
14 #include <LoRaNow.h>
15
16 // Para a manipulacao dos pacotes de dados (JSON)
17 #include <StreamString.h>
18 #include <CayenneLPPDecode.h>
19
20 #define ID_ESTACAO    1000                // Define a IDentificacao do
       Gateway
21 #define NOME_ESTACAO  "Gateway"           // Define o nome do Gateway
22
23 /*
24   pinOut para utilizar o LoRa
25 */
26 #define LORA_SCK     5    // GPIO5 -- SX127x's SCK
27 #define LORA_MISO    19   // GPIO19 -- SX127x's MISO
28 #define LORA_MOSI    27   // GPIO27 -- SX127x's MOSI
29 #define LORA_SS      18   // GPIO18 -- SX127x's CS
30 #define LORA_RST     14   // GPIO14 -- SX127x's RESET
31 #define LORA_DIO     26   // GPIO26 -- SX127x's IRQ (Interrupt Request)
32 #define LORA_BAND 433E6   //Frequencia do radio: 433 MHz
33 #define LORA_PABOOST true
34
35 /*
36   Topico utilizado MQTT
37 */
38 #define GATEWAY_TX      "tcc/gateway/tx"
39 #define GATEWAY_RX      "tcc/gateway/rx"
40 #define GATEWAY_STATUS  "tcc/gateway/status"
41 #define EB_STATUS       "tcc/eb/status"
42 #define GATEWAY_CHECK   "tcc/gateway/check"
43 #define EB_CHECK        "tcc/eb/check"
44
45 String droneId = "";                          // RFID: String que
       armazena o ID (RFID) do Drone
46 int dronePousado = 0;                         // Variavel para verificar
       se o drone continua pousado na base ou decolou
```

```
47
48 const char* ssid = "AP202-Base";                // WiFi: SSID para se
     conectar ao roteador
49 const char* password = "12345";                 // WiFi: Senha para se
     conectar a rede
50 const char* mqtt_server = "192.168.0.104";      // MQTT: IP do broker
51
52 WiFiClient espClient;
53 PubSubClient client(espClient);
54
55 /*
56   ----------------------------------------------------------------------
57                 Funcoes para utilizar o MQTT e WiFi
58 */
59 void pingMQTT(){
60   DynamicJsonDocument doc(128);
61   String input = "{}";
62   deserializeJson(doc, input);
63   JsonObject obj = doc.as<JsonObject>();
64
65   obj["ID_GATEWAY"] = ID_ESTACAO;
66
67   // LoRaNow.showStatus(Serial);
68
69   char JSONbuffer[128];
70   size_t n = serializeJson(obj, JSONbuffer);
71
72   client.loop();
73   client.publish(GATEWAY_STATUS, JSONbuffer, n);
74 }
75
76 void pingEB(){
77   LoRaNow.clear();
78   LoRaNow.print("ping");
79   LoRaNow.send();
80 }
81
82 /* SE RECEBE ALGUM CALLBACK MQTT */
83 void receivedCallback(char* topic, byte* payload, unsigned int length) {
84   // Se receber 'TRUE' no topico de check, ele pinga pro broker
     confirmando que continua online!
85   if((char)payload[0] && strcmp(topic, "tcc/gateway/check") == 0){
86     /* Informa o broker que o GATEWAY esta online */
87     pingMQTT();
88   }
89
90   if((char)payload[0] && strcmp(topic, "tcc/eb/check") == 0){
```

```
91      // /* Informa o broker que a estacao base esta online */
92      // pingEB();
93    }
94
95    Serial.println();
96  }
97
98  void verificaMQTT() {
99    while (!client.connected()) {
100     Serial.print("MQTT conectando ...");
101
102     Heltec.display->clear();
103     Heltec.display->drawString(0, 0, NOME_ESTACAO);
104     Heltec.display->drawString(0, 10, String(ID_ESTACAO));
105     Heltec.display->drawString(0, 20, "IP:");
106     Heltec.display->drawString(15, 20, WiFi.localIP().toString());
107     Heltec.display->drawString(0, 30, "Conectando MQTT...");
108     Heltec.display->display();
109
110     /* ID do cliente MQTT */
111     String clientId = NOME_ESTACAO;
112
113     if (client.connect(clientId.c_str())) {
114       Heltec.display->clear();
115       Heltec.display->drawString(0, 0, NOME_ESTACAO);
116       Heltec.display->drawString(0, 10, String(ID_ESTACAO));
117       Heltec.display->drawString(0, 20, "IP:");
118       Heltec.display->drawString(15, 20, WiFi.localIP().toString());
119       Heltec.display->drawString(0, 30, "MQTT ON");
120       Heltec.display->display();
121
122       Serial.println("Conectado MQTT");
123       /* Informa o broker que o GATEWAY esta online */
124       pingMQTT();
125
126       /* Se inscreve no topico com o QoS DEFAULT = 0 */
127       client.subscribe(GATEWAY_RX, 1);
128       client.subscribe(GATEWAY_CHECK, 1);
129       client.subscribe(EB_CHECK, 1);
130     } else {
131       Serial.print("falha, codigo status =");
132       Serial.print(client.state());
133       Serial.println("tentando reconectar em 5 segundos");
134
135       Heltec.display->clear();
136       Heltec.display->drawString(0, 0, NOME_ESTACAO);
137       Heltec.display->drawString(0, 10, String(ID_ESTACAO));
```

```
138       Heltec.display->drawString(0, 20, "IP:");
139       Heltec.display->drawString(15, 20, WiFi.localIP().toString());
140       Heltec.display->drawString(0, 30, "Conectando MQTT...");
141       Heltec.display->display();
142
143       delay(1000);
144     }
145   }
146 }
147
148 /*
149               <<END>> Funcoes para utilizar o MQTT e WiFi
150   ------------------------------------------------------------------------
151 */
152
153 /* Funcoes LoRaNow: Gateway/No */
154 void enviaRetorno(){
155   LoRaNow.clear();
156   LoRaNow.print("RFID RECEBIDO PELO GTW");
157   LoRaNow.send();
158 }
159
160 /* SE RECEBE ALGUM CALLBACK LoRa */
161 void aoReceber(uint8_t *buffer, size_t size)
162 {
163   DynamicJsonDocument doc(128);
164   deserializeJson(doc, buffer);
165   JsonObject obj = doc.as<JsonObject>();
166
167   obj["ID_GATEWAY"] = ID_ESTACAO;
168
169   Heltec.display->clear();
170   Heltec.display->drawString(0, 0, NOME_ESTACAO);
171   Heltec.display->drawString(0, 10, String(ID_ESTACAO));
172   Heltec.display->drawString(0, 20, "IP:");
173   Heltec.display->drawString(15, 20, WiFi.localIP().toString());
174   Heltec.display->drawString(0, 30, "MQTT ON");
175
176   serializeJsonPretty(obj ,Serial);
177   dronePousado = obj[String("DRONE_POUSADO")];
178
179   if(dronePousado == 1){
180     Heltec.display->drawString(0, 40, obj[String("ID_DRONE")]);
181     Heltec.display->drawString(0, 50, "DRONE NA EB");
182     Heltec.display->drawString(75, 50, obj[String("ID_EB")]);
183   }else{
184     Heltec.display->drawString(0, 40, "-> SEM DRONE NA EB: ");
```

```
185     Heltec.display->drawString(0, 50, obj[String("ID_EB")]);
186   }
187
188   Heltec.display->display();
189
190   char JSONbuffer[128];
191   size_t n = serializeJson(obj, JSONbuffer);
192
193   client.loop();
194
195   size_t tamObj = obj.size();
196   // Se o tamanho e 1, esta recebendo somente a ID da estacao base que
       esta FICANDO ONLINE..
197   // adiciona o campo EB_ONLINE no JSON para informar o node-red
198
199   if(tamObj == 4){
200     obj["EB_ONLINE"] = 1;
201     client.publish(EB_STATUS, JSONbuffer, n);
202     pingMQTT();
203   }else{
204     client.publish(GATEWAY_TX, JSONbuffer, n);
205   }
206
207   delay(500);
208 }
209
210 void printHex(byte *buffer, byte bufferSize) {
211   for (byte i = 0; i < bufferSize; i++) {
212     Serial.print(buffer[i] < 0x10 ? " 0" : " ");
213     Serial.print(buffer[i], HEX);
214   }
215 }
216
217 void printDec(byte *buffer, byte bufferSize) {
218   for (byte i = 0; i < bufferSize; i++) {
219     Serial.print(buffer[i] < 0x10 ? " 0" : " ");
220     Serial.print(buffer[i], DEC);
221   }
222 }
223
224 void setup() {
225   Serial.println("Gateway -> Setando Porta Serial: 115200");
226   Serial.begin(115200);
227
228   Heltec.begin(true, false, true, false);
229
230   Heltec.display->init();
```

```
231   Heltec.display->flipScreenVertically();
232   Heltec.display->setFont(ArialMT_Plain_10);
233
234   WiFi.mode(WIFI_STA);
235   WiFi.begin(ssid, password);
236
237   Heltec.display->clear();
238   Heltec.display->drawString(0, 0, NOME_ESTACAO);
239   Heltec.display->drawString(0, 10, String(ID_ESTACAO));
240   Heltec.display->drawString(0, 20, "WiFi conectando...");
241   Serial.print("Conectando no WiFi");
242   Heltec.display->display();
243
244   while (WiFi.status() != WL_CONNECTED) {
245     delay(500);
246     Serial.print(".");
247   }
248
249   Heltec.display->clear();
250   Heltec.display->drawString(0, 0, NOME_ESTACAO);
251   Heltec.display->drawString(0, 10, String(ID_ESTACAO));
252   Heltec.display->drawString(0, 20, "IP:");
253   Heltec.display->drawString(15, 20, WiFi.localIP().toString());
254   Heltec.display->display();
255
256   Serial.println(WiFi.localIP());
257   Serial.println("\n");
258
259   /* Conecta no broker */
260   client.setServer(mqtt_server, 1883);
261
262   /*
263     A funcao receivedCallback vai ser invocada quando
264     receber alguma publicacao no topico inscrito
265   */
266   client.setCallback(receivedCallback);
267
268   /*
269     Configura os pinos que serao utlizados pela biblioteca (deve ser
      chamado antes do LoRa.begin)
270   */
271   LoRaNow.setPinsSPI(LORA_SCK,LORA_MISO,LORA_MOSI,LORA_SS,LORA_DIO);
272
273   // Inicializa o Lora com a frequencia de 433 MHz.
274   LoRaNow.setFrequency(433E6);
275
276
```

```
277   if (!LoRaNow.begin())
278   {
279     Serial.println("Gateway -> LoRa falhou ao inicializar!");
280     while (1);
281   }
282
283   LoRaNow.setId(ID_ESTACAO);
284   LoRaNow.onMessage(aoReceber);       // Seta a funcao de Callback
285   // LoRaNow.onSleep(onSleep);        // Seta a funcao de Sleep para o
       baixo consumo
286   LoRaNow.gateway();
287
288   delay(1000);
289 }
290
291 void loop() {
292   verificaMQTT();
293   LoRaNow.loop();
294   client.loop();
295 }
```

Código A.11 – Código utilizado na estação base

```
1  #include <heltec.h>
2  #include <SPI.h>
3  #include <Wire.h>
4  #include <time.h>
5  #include <StreamString.h>
6
7  // Para a comunicacao Estacoes Base / Gateway
8  #include <LoRaNow.h>
9
10 // RFID
11 #include <MFRC522.h>
12
13 // Para a manipulacao dos pacotes de dados (JSON)
14 #include <CayenneLPPDecode.h>
15
16 // Biblioteca do Projeto
17 #define ID_ESTACAO    12    // ID que sera definido para a estacao base
18 #define NOME_ESTACAO  "Estacao_Base_3"  // Nome que sera definido para a
       estacao base
19
20 /*
21   pinOut para utilizar o LoRa
22 */
23 #define LORA_SCK     5    // GPIO5 -- SX127x's SCK
```

```
24 #define LORA_MISO     19   // GPIO19 -- SX127x's MISO
25 #define LORA_MOSI     27   // GPIO27 -- SX127x's MOSI
26 #define LORA_SS       18   // GPIO18 -- SX127x's CS
27 #define LORA_RST      14   // GPIO14 -- SX127x's RESET
28 #define LORA_DIO      26   // GPIO26 -- SX127x's IRQ (Interrupt Request)
29 #define LORA_BAND 433E6 //Frequencia do radio: 433 MHz
30 #define LORA_PABOOST true
31
32 /*
33   pinOut para utilizar o RFID
34 */
35 #define RFID_SDA 5
36 #define RFID_SCK 18
37 #define RFID_MOSI 23
38 #define RFID_MISO 19
39 #define RFID_RST 22
40
41 /*
42   Topico utilizado MQTT
43 */
44 // #define SERVER_TOPIC    "tcc/gateway"
45
46 /*
47   Variavel para ficar alternando a comunicacao SPI entre o LoRa e o RFID
     , se:
48   spiAtual = -1 -> Nao iniciado
49   spiAtual = 0  -> RFID
50   spiAtual = 1  -> LoRa
51 */
52 int atualSPI = -1;
53 // Variavel para verificar se o drone continua pousado na base ou
     decolou [-1 -> Decolou; 0 -> Nenhum Drone; 1 -> Pousado]
54 int dronePousado = 0;
55 time_t seconds, tInicio=0;
56
57 MFRC522 mfrc522(RFID_SDA, RFID_RST);    // RFID: Cria uma instancia do
     MFRC522
58 byte nuidPICC[4];   // RFID: Variavel para armazenar o RFID
59 String droneId = "";    // RFID: String que armazena o ID (RFID) do
     Drone
60 int idDrone = 0;    // Se o idDrone = 0 nao tem drone na base
61 bool ebInicializada = false;
62
63 CayenneLPP pacoteJSON(128); // Iniciliza uma instancia do Cayenne para
     armazenas as informacoes em JSON
64
65 /* Funcoes LoRaNow: Gateway/No */
```

```
66 // AO RECEBER MENSAGEM LoRa do No...
67 void onMessage(uint8_t *buffer, size_t size)
68 {
69   enviaInfo();
70 }
71
72 /* Funcoes para utilizar o LoRa e o RFID ao mesmo tempo no mesmo SPI */
73 void selecionaSPI(int qualInterface) {
74   if (qualInterface == atualSPI) return;
75     SPI.end();
76   switch(qualInterface) {
77       case 0:
78         SPI.begin(RFID_SCK, RFID_MISO, RFID_MOSI);
79         mfrc522.PCD_Init();
80         break;
81       case 1:
82         // Configura os pinos que serao utilizados pela biblioteca (deve
     ser chamado antes do LoRa.begin)
83         LoRaNow.setPinsSPI(LORA_SCK,LORA_MISO,LORA_MOSI,LORA_SS,LORA_DIO
    );
84         LoRaNow.setFrequency(433E6);
85         // Inicializa o Lora com a frequencia especifica.
86         if (!LoRaNow.begin())
87         {
88           while (1);
89         }
90         LoRaNow.onMessage(onMessage);       // Seta a funcao de Callback
91         break;
92       default:
93         break;
94   }
95   atualSPI = qualInterface;
96 }
97
98 void enviaInfo(){
99 DynamicJsonDocument doc(256);
100 String input = "{}";
101 deserializeJson(doc, input);
102 JsonObject obj = doc.as<JsonObject>();
103
104 obj["ID_EB"] = ID_ESTACAO;
105 obj["ID_DRONE"] = idDrone;
106 obj["DRONE_POUSADO"] = dronePousado;
107
108 String output;
109 serializeJson(obj, output);
110
```

```
111 serializeJsonPretty(obj ,Serial);  // Mostra o JSON de forma "bonita"
112 // Serial.println();
113
114 selecionaSPI(1);
115
116 // Serial.println("Envia LoRa pacote JSON");
117 LoRaNow.print(output);
118 LoRaNow.send();
119
120 delay(500);
121 }
122
123 void inicializaEB(){
124 enviaInfo();
125 ebInicializada = true;
126 delay(100);
127 }
128
129 int droneIdToInt(byte *buffer){
130 return *(uint32_t *) buffer;
131 }
132
133 int verificaRFID() {
134 selecionaSPI(0);
135 droneId = "";
136
137 // Aguarda 5 segundos para verificar novamente
138 if(seconds - tInicio < 5)
139   return false;
140
141 // Verifica se tem algum contato com o RFID, se tiver, ele le atraves da
      funcao PICC_ReadCardSerial()
142 if (!mfrc522.PICC_IsNewCardPresent() || !mfrc522.PICC_ReadCardSerial())
    {
143   if(dronePousado == 1){     // Se tinha drone, ele decolou
144     dronePousado = -1;      // Marca que decolou [-1 -> Decolou; 0 ->
     Nenhum Drone; 1 -> Pousado
145
146     Serial.println("DRONE DECOLOU");
147     Serial.println("NENHUM DRONE");
148     Heltec.display->clear();
149     Heltec.display->drawString(0, 0, NOME_ESTACAO);
150     Heltec.display->drawString(0, 10, "ID: ");
151     Heltec.display->drawString(15, 10, String(ID_ESTACAO));
152     Heltec.display->drawString(0, 20, "Drone decolou...");
153     Heltec.display->display();
154
```

```
155     idDrone = 0;
156   }else{
157     dronePousado = 0;
158     enviaInfo();
159 
160     Heltec.display->clear();
161     Heltec.display->drawString(0, 0, NOME_ESTACAO);
162     Heltec.display->drawString(0, 10, "ID: ");
163     Heltec.display->drawString(15, 10, String(ID_ESTACAO));
164     Heltec.display->drawString(0, 20, "Nenhum drone na base");
165     Heltec.display->display();
166   }
167 
168   tInicio = seconds;
169   return false;
170 }
171 
172 for (byte i = 0; i < 4; i++) {
173   nuidPICC[i] = mfrc522.uid.uidByte[i];
174 }
175 
176 /*Pega ID do Drone em String*/
177 for (byte i = 0; i < mfrc522.uid.size; i++)
178 {
179   droneId.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? "0" : ""));
180   droneId.concat(String(mfrc522.uid.uidByte[i], HEX));
181 }
182 
183 idDrone = droneIdToInt(mfrc522.uid.uidByte);      // Converte ID do
    drone para Int
184 
185 if(dronePousado == 1){
186   // Serial.println("DRONE CONTINUA NA EB");
187 }else{
188   Serial.println();
189   Serial.println("DRONE POUSOU");
190   dronePousado = 1; // Informa que o drone pousou
191 
192   Heltec.display->clear();
193   Heltec.display->drawString(0, 0, NOME_ESTACAO);
194   Heltec.display->drawString(0, 10, "ID: ");
195   Heltec.display->drawString(15, 10, String(ID_ESTACAO));
196   Heltec.display->drawString(0, 20, "Drone: ");
197   Heltec.display->drawString(35, 20, String(idDrone));
198   Heltec.display->display();
199 }
200 
```

```
201 enviaInfo();
202
203 mfrc522.PICC_HaltA();
204 mfrc522.PCD_StopCrypto1();
205
206 tInicio = seconds;  // Inicia a contagem para em 5 segundos verificar
    novamente se tem algum drone na EB
207 return true;
208 }
209
210 void setup(){
211   Serial.println("Estacao BASE -> Setando Porta Serial: 115200");
212   Serial.begin(115200);
213
214   Heltec.begin(true, false, true , false);
215
216   Heltec.display->init();
217   Heltec.display->flipScreenVertically();
218   Heltec.display->setFont(ArialMT_Plain_10);
219   Heltec.display->clear();
220   Heltec.display->drawString(0, 0, NOME_ESTACAO);
221   Heltec.display->drawString(0, 10, "ID: ");
222   Heltec.display->drawString(15, 10, String(ID_ESTACAO));
223   Heltec.display->display();
224
225   time(&seconds);
226   tInicio = seconds + 5;
227   LoRaNow.setId(ID_ESTACAO);
228   LoRaNow.showStatus(Serial);
229
230   delay(1000);
231 }
232
233 void loop() {
234   selecionaSPI(1);
235   LoRaNow.loop();
236
237   verificaRFID();   // Verifica se algum drone pousou na EB (se alguma
     TAG RFID se aproximou do cartao)
238
239   if(ebInicializada == false){
240     inicializaEB();
241   }
242
243   time(&seconds);   // Contador de segundos para as verificacoes
244   delay(10);
245 }
```

Referências

Adelantado, F. et al. Understanding the limits of lorawan. **IEEE Communications Magazine**, v. 55, n. 9, p. 34–40, Sep. 2017. ISSN 0163-6804.

BRITO, R. C. **Um Modelo de Otimização para Planejamento Dinâmico de Voo para Grupos de Drones utilizando Cooperação com Bases de Recargas Móveis por Meio de Sistema Multiagente e Leilões Recursivos**. 2018. 116 f. Dissertação (Doutorado em Informática - Ciências Exatas) — Universidade Federal do Paraná, Curitiba, 2018.

Brito, R. C. et al. A comparative approach on the use of unmanned aerial vehicles kind of fixed-wing and rotative wing applied to the precision agriculture scenario. In: **2019 IEEE Computer Society Signature Conference on Computers, Software and Applications (COMPSAC)**. [S.l.: s.n.], 2019.

CELTEK, S. A.; DURDU, A.; KURNAZ, E. Design and simulation of the hierarchical tree topology based wireless drone networks. In: **2018 International Conference on Artificial Intelligence and Data Processing (IDAP)**. [S.l.: s.n.], 2018. p. 1–5.

DOCKER. 2019. Disponível em: <https://www.docker.com/why-docker>.

EGLI, P. R. Overview of emerging technologies for low power wide area networks in internet of things and m2m scenarios. p. 11, 2015.

ESCOBAR, E. **How Does Wi-Fi Work?** 2015. Disponível em: <https://www.scientificamerican.com/article/how-does-wi-fi-work/?redirect=1>. Acesso em: 28 de Maio de 2019.

ESTRATÉGIA NACIONAL DE CIÊNCIA, TECNOLOGIA E INOVAÇÃO. 2016. Disponível em: <http://www.finep.gov.br/images/a-finep/Politica/16_03_2018_Estrategia_Nacional_de_Ciencia_Tecnologia_e_Inovacao_2016_2022.pdf>.

Fernandez, P.; Jara, A. J.; Skarmeta, A. F. G. Evaluation framework for ieee 802.15.4 and ieee 802.11 for smart cities. In: **2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing**. [S.l.: s.n.], 2013. p. 421–426.

GEORGE, E. A. et al. Uav systems for parameter identification in agriculture. **2013 IEEE Global Humanitarian Technology Conference: South Asia Satellite (GHTC-SAS)**, p. 270–273, 2013.

GUPTA, N. **Inside Bluetooth Low Energy**. 1. ed. [S.l.: s.n.], 2013.

IEEE Standard for Low-Rate Wireless Networks. **IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)**, p. 1–709, April 2016.

INTRODUCTION to Node.js. 2019. Disponível em: <https://nodejs.dev/>.

KITE-POWELL, J. **Why Precision Agriculture Will Change How Food Is Produced**. 2018. Disponível em: <https://www.forbes.com/sites/jenniferhicks/2018/04/30/why-precision-agriculture-will-change-how-food-is-produced/#744a23e86c65>.

Kortuem, G. et al. Smart objects as building blocks for the internet of things. **IEEE Internet Computing**, v. 14, n. 1, p. 44–51, Jan 2010. ISSN 1089-7801.

Kuor-Hsin Chang; Mason, B. The ieee 802.15.4g standard for smart metering utility networks. In: **2012 IEEE Third International Conference on Smart Grid Communications (SmartGridComm)**. [S.l.: s.n.], 2012. p. 476–480.

LIBRARY to decode CayenneLPP. 2019. Disponível em: <https://github.com/ricaun/CayenneLPPDecode/>.

LORA ALLIANCE. **LoRa Alliance Passes 100 LoRaWAN™ Network Operator Milestone with Coverage in 100 Countries**. [S.l.], 2019. Disponível em: <https://lora-alliance.org/in-the-news/lora-alliance-passes-100-lorawantm-network-operator-milestone-coverage-100-countries>. Acesso em: 28 de Maio de 2019.

LUETH, K. L. **State of the IoT 2018: Number of IoT devices**. 2018. Disponível em: <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>. Acesso em: 17 de Maio de 2019.

Marginean, H.; Tran, T.; Karzel, D. **Uma arquitetura de referência para a Internet das Coisas**. 2016. Disponível em: <https://www.infoq.com/br/articles/internet-of-things-reference-architecture>. Acesso em: 18 de Maio de 2019.

MARKETWATCH. **Agricultural Drones Market Worth $3.69 Billion by 2022**. 2016. Disponível em: <http://www.marketwatch.com/story/agricultural-drones-market-worth-369-billionby-2022-2016-04-06-2203128>. Acesso em: 15 de Maio de 2019.

Mikhaylov, K.; Juha Petaejaejaervi, .; Haenninen, T. Analysis of capacity and scalability of the lora low power wide area network technology. In: **European Wireless 2016; 22th European Wireless Conference**. [S.l.: s.n.], 2016. p. 1–6.

MONGODB. 2019. Disponível em: <https://www.mongodb.com/>.

MOSCA broker MQTT. 2019. Disponível em: <http://www.mosca.io/>.

MUXFELDT, P. **Formatos e extensões de arquivos - Tipo MIME**. 2017. Disponível em: <https://br.ccm.net/contents/649-formatos-e-extensoes-de-arquivos-tipo-mime>. Acesso em: 19 de Maio de 2019.

Naik, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: **2017 IEEE International Systems Engineering Symposium (ISSE)**. [S.l.: s.n.], 2017. p. 1–7.

NODE JS. 2019. Disponível em: <https://www.nodejs.org>.

NODE-RED. 2019. Disponível em: <https://nodered.org/>.

NUNES, B. **Introdução a LoRa®, NB-IoT e Sigfox**. 2017. Disponível em: <https://www.embarcados.com.br/lora-nb-iot-e-sigfox/>.

PINTO, P. **Conheça melhor o protocolo de rede IEEE 802.15.4**. 2017. Disponível em: <https://pplware.sapo.pt/tutoriais/networking/conheca-melhor-o-protocolo-de-rede-ieee-802-15-4/>. Acesso em: 27 de Abril de 2019.

PLANO de Atribuição, Destinação e Distribuição de Frequências no Brasil. 2019. Disponível em: <https://www.anatel.gov.br/institucional/acervo-documental>.

PLATFORMIO. 2019. Disponível em: <https://platformio.org/>.

POR que Node-RED para IOT? 2019. Disponível em: <https://medium.com/@netoolii/por-que-node-red-para-iot-41a4ab170c56>.

ROSS, K. **Redes de Computadores e a Internet**. 5. ed. [S.l.: s.n.], 2009.

SEMTECH CORPORATION. **LoRa™ Modulation Basics**. [S.l.], 2015. 26 p. Disponível em: <https://www.semtech.com/uploads/documents/an1200.22.pdf>. Acesso em: 27 de Abril de 2019.

STUDIO 3T. 2019. Disponível em: <https://studio3t.com/features/>.

VISUAL Studio Code. 2019. Disponível em: <https://code.visualstudio.com/docs/editor/whyvscode>.

WORKGROUP, L. A. T. M. **A technical overview of LoRa® and LoRaWAN™**. 2015. Disponível em: <https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf>. Acesso em: 27 de Junho de 2019.

YUAN, M. **Getting to know MQTT**. 2017. Disponível em: <https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/>. Acesso em: 27 de Abril de 2019.