APLICAÇÃO DA TECNOLOGIA LORA EM UM SISTEMA DE GERENCIAMENTO E RASTREAMENTO DE ENTREGAS DE MERCADORIAS

Gabriel Deggau Schmidt, Miguel Alexandre Wisintainer – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

gdschmidt@furb.br, maw@furb.br

Resumo: *Este artigo apresenta o desenvolvimento de um sistema que permite o gerenciamento e rastreamento de entregas de mercadorias. É utilizada a placa de desenvolvimento WiFi LoRa 32 (V2), com um módulo de GPS para se obter a geolocalização do veículo e, enviar através da rede LoRaWAN esses dados até um servidor de rede da empresa KORE, o qual os encaminha para a aplicação web desenvolvida neste trabalho. Dessa maneira, o objetivo deste trabalho é avaliar a disponibilidade e eficiência da rede LoRaWAN em uma região do Vale do Itajaí. Foram realizados testes com o dispositivo em movimento no carro passando por alguns bairros em torno do centro de Gaspar e Blumenau. Foi possível observar que em certas regiões a área de cobertura da rede LoRaWAN não é tão abrangente, fazendo assim com que algumas mensagens não fossem possíveis de serem entregues. Já observando o comportamento em outras regiões, foi constatado uma maior entrega de mensagens com sucesso.*

Palavras-chave*: LoRa. LPWAN. IoT. Logística. Rastrear. GPS.*

1 INTRODUÇÃO

Atualmente com o espaço que a internet vem ganhando e o aumento da globalização, grande parte do comércio está se expandindo a o meio digital para venda de seus produtos. Há lojas que permitem o cliente retirar o seu produto em algum lugar designado, como por exemplo, uma loja física que possui o produto comprado e se situa próximo do cliente. Porém, a maioria das lojas realizam as entregas da mercadoria diretamente ao cliente, utilizando os seus próprios meios de transporte ou contratando um serviço de uma empresa especializada nesta área. Independentemente de quem estiver realizando o transporte e o gerenciamento da mercadoria, um dos maiores desafios para estas empresas no Brasil, é oferecer um serviço de alta qualidade. Isso acontece devido as dificuldades enfrentadas durante o processo de entrega, como por exemplo, falta de segurança nas vias públicas. No ano de 2018, foi registrado um total de 22.183 ocorrências de roubos de cargas pelo país, equivalente ao total de R$ 1,47 bilhão de prejuízo (NTC&LOGÍSTICA, 2018).

Assim, para tentar evitar estes contratempos, reduzir custos e consequentemente ganhar eficiência nas entregas, as empresas de transporte e logística vem investindo em tecnologias que podem auxiliar nestes pontos. Uma delas é o monitoramento de cargas, que oferece diversas vantagens, como a possibilidade de fornecer a localização e situação da entrega tanto ao cliente quanto à própria transportadora em tempo real, além de acompanhar se a carga está percorrendo a rota planejada. Além disso, é possível gerir o desempenho do motorista, do modo que, seja possível averiguar se o mesmo está cumprindo os horários estipulados para as entregas (CONTEFLEX, 2017).

Para se obter esses tipos de informações do veículo, geralmente é utilizado algum dispositivo de Internet of Things (IoT) que seja capaz de coletar os dados de um Global Partition System (GPS), o qual pode conter a localização e a velocidade atual do veículo. Na maioria das aplicações convencionais de monitoramento veicular, é utilizado a rede Global System for Mobile (GSM) para a transmissão dos dados à um servidor de aplicação, este que recebe, classifica os dados, e apresenta ao usuário do sistema. Porém, com o advento da IoT, diversas novas redes surgiram com o intuito de realizar o tráfego de pequenos pacotes de dados e garantir a longevidade da bateria do dispositivo, essas redes são chamadas de Low Power Wide Area Network (LPWAN). Uma dessas redes é a Long Range Wide Area Network (LoRaWAN), sendo capaz de realizar transmissões sem fio em ambientes urbanos a distâncias que variam entre 3 a 4 quilômetros, e em áreas rurais, pode chegar até 15 quilômetros de distância, tudo isso utilizando uma potência baixa, em torno de 20dBm ou 100mW (ALMEIDA, 2019).

A partir das características apresentadas acima, este trabalho propõe apresentar os aspectos da tecnologia LoRa e seu protocolo LoRaWAN, com o objetivo de avaliar a eficiência e disponibilidade da rede LoRaWAN na região de Gaspar e Blumenau. Para isso, também é proposto desenvolver uma aplicação web para realizar o gerenciamento e rastreamento das entregas de mercadorias realizadas por um centro de distribuição, utilizando um dispositivo com GPS dentro do veículo para se obter as geolocalizações e enviar via rede LoRaWAN até a aplicação web.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção tem como objetivo explorar os principais assuntos para realização deste trabalho. A subseção 2.1 aborda a tecnologia LoRa e seus principais fundamentos. A subseção 2.2 apresenta as características do protocolo LoRaWAN. A subseção 2.3 é comentado sobre o conceito de logística. Por fim, na subseção 2.4 são descritos os trabalhos correlatos.

2.1 LORA

Mantida pela a empresa Semtech e desenvolvida para atuar em redes LPWANs, LoRa, abreviação para long-range, é uma tecnologia pertencente a camada física que possui uma modulação *wireless* capaz de realizar transmissão de dados em longo alcance, baixo consumo de energia, segurança na transmissão dos dados e possuir resistência a interferências. Pode operar em diversas bandas ISM, como: 433, 868 e 915 MHz dependendo da região em que está sendo utilizada (BARREIROS, 2019). No Brasil, a banda de frequência de operação é em 915 MHz (AU915 e LA915).

Segundo Augustin *et al.* (2016), LoRa é baseada na técnica de modulação Chirp Spread Spectrum (CSS), a qual realiza uma variação linear da frequência do sinal para codificar a informação. Além disso, a modulação CSS mantém as mesmas características de baixo consumo de energia que a modulação Frequency Shifting Keying (FSK) proporciona, porém, aumentando significativamente a distância do link de comunicação. A modulação CSS foi utilizada durante décadas por militares e comunicações espaciais, devido ao longo alcance que atinge e a robustez a interferências, contudo, LoRa é a primeira implementação de baixo custo para uso comercial (LORA ALLIANCE, 2015).

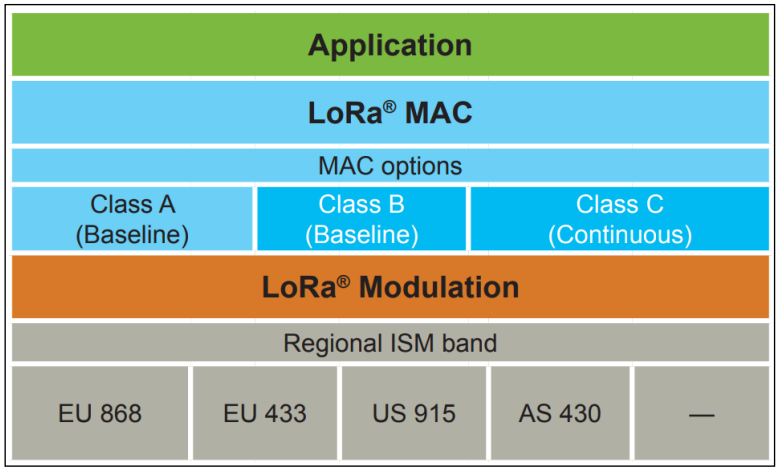
Conforme Ribeiro (2019), existem certos parâmetros para customização da modulação LoRa, os quais podem influenciar na taxa de transmissão da modulação, na resistência a ruídos e na facilidade de decodificação dos dados. O parâmetro Bandwidth (BW), recebe o valor da largura da banda ocupada por um *Chirp*, normalmente é utilizado os valores 125, 250 ou 500 kHz. Outro parâmetro presente é o Spreading Factor (SF), esse é o fator de espalhamento que define a duração do *Chirp* no ar, podendo variar de 6 a 12. Não menos importante, existe o Code Rate (CR), taxa de código responsável por corrigir cada transmissão dos dados. Bor, Vidler e Roeding (2016) descrevem as consequências que cada parâmetro pode trazer ao ser alterado:

a) SF alto: aumenta a relação sinal/ruído (SNR) e assim aumentando o alcance e sensibilidade. Contudo, este também aumenta o tempo do pacote a ser transmitido e o consumo de energia do dispositivo;
b) BW alto: uma maior taxa de dados e um menor tempo para ser transmitido, porém, ocasionando uma baixa sensibilidade, devido a integração de ruído adicional;
c) BW baixo: aumenta a sensibilidade, mas diminui a taxa de dados. Também precisará de mais cristais de precisão de baixo parts per million (ppm);
d) CR alto: oferece uma maior proteção e codificação dos dados, porém aumenta o tempo de transmissão.

2.2 LORAWAN

A rede LoRaWAN é quem define o protocolo de comunicação e toda a arquitetura do sistema da rede, como pode ser visto na Figura 1. Já a camada física LoRa, é responsável por disponibilizar os links de comunicação de longo alcance (LORA ALLIANCE, 2015). As regras desta rede são definidas pela LoRa Alliance, grupo formado por empresas de Tecnologia da Informação (TI) como IBM, Actility, Semtech e Microchip.

Figura 1 – Arquitetura LoRaWAN

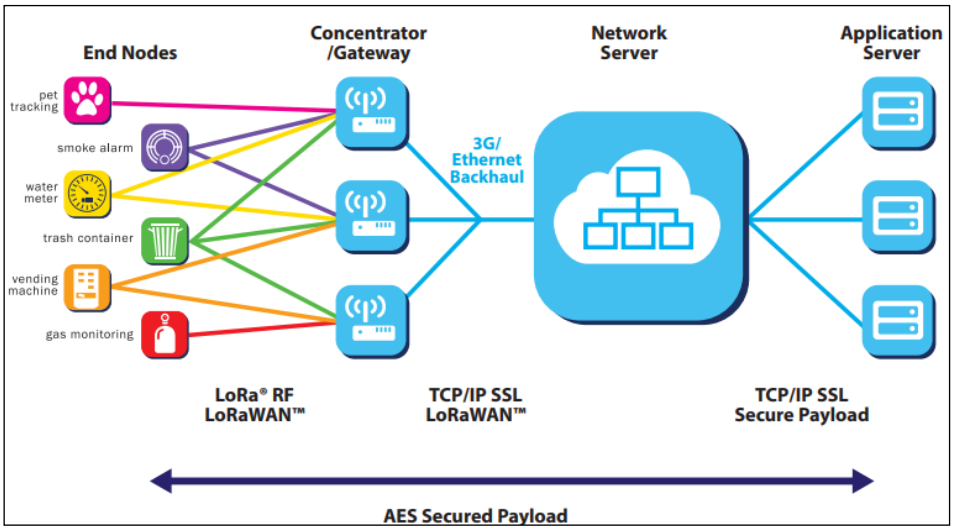


Fonte: LoRa Alliance (2015).

Este protocolo disponibiliza o mecanismo Medium Access Control (MAC), o qual possibilita muitos dispositivos se conectarem a um *gateway*, e um dispositivo não está necessariamente associado a um único *gateway*. Conforme

Augustin *et al.* (2016), normalmente é utilizada a arquitetura de topologia de redes estrela para implementação do protocolo LoRaWAN, a qual inclui três tipos de dispositivos, como mostra a Figura 2.

Figura 2 – Arquitetura da rede LoRaWAN



Fonte: LoRa Alliance (2015).

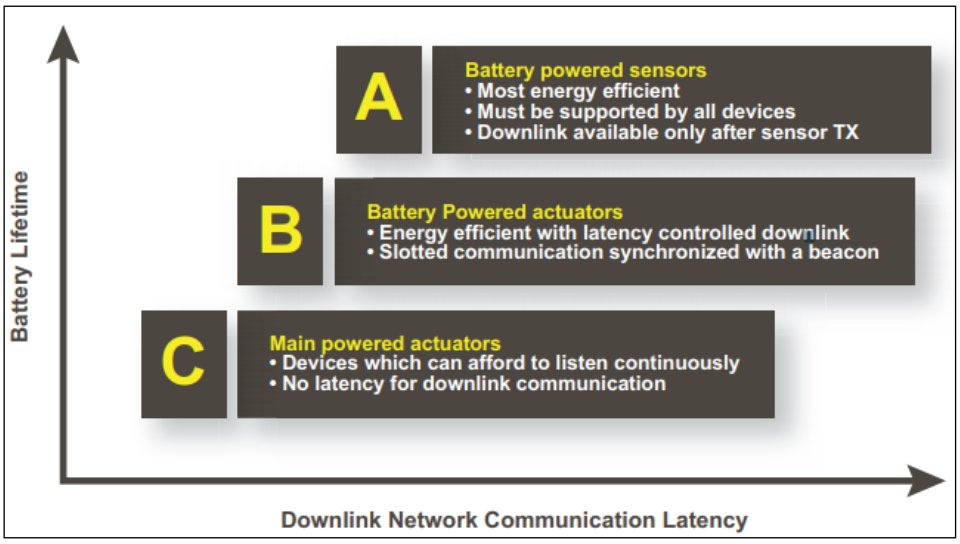
A seguir serão descritas as principais funcionalidades dos equipamentos que constituem uma rede LoRaWAN:

a) *end-device*: também chamados de nós ou simplesmente dispositivos, estes são sensores e atuadores com baixo consumo de energia que enviam os dados aos *gateways* via LoRa;
b) *gateway*: são equipamentos intermediadores que recebem os dados enviados pelos *end-devices* e encaminham a um *network server* via rede IP (os *gateways* são transparentes aos nós). Nesta arquitetura pode existir mais de um *gateway*, assim quando um pacote de dados é enviado por um *end-device*, vários *gateways* podem recebê-los e enviarem o mesmo pacote ao *network server*;
c) *network server*: servidor de rede LoRaWAN responsável por gerenciar os pacotes de dados entregues pelos *gateways*, possui funcionalidades para verificação de pacotes duplicados e decodificação dos mesmos, também possui verificações de segurança, é capaz de escolher o *gateway* mais apropriado para receber um pacote de resposta e até mesmo enviar os dados à servidores de aplicação;
d) *application server*: o servidor de aplicação não está ligado totalmente a arquitetura LoRaWAN, esse é geralmente onde que os dados capturados pelo *end-diveces* serão processados e, toda o conjunto de regras da aplicação é executado, sendo parte responsável do desenvolvedor a sua estruturação e utilização.

Para cada tipo de aplicação que irá utilizar a rede LoRaWAN, é possível configurar os dispositivos de maneira que melhor se enquadram nas especificações das mesmas. Assim, é definido três tipos de classes que os dispositivos podem assumir, essas classes são chamadas de A, B ou C, o que diferencia uma da outra é latência das mensagens de `downlink` (mensagem enviada do servidor de rede LoRaWAN ao dispositivo) e o tempo de duração da bateria dos dispositivos. A LoRa Alliance (2015) define as classes dos nós da seguinte maneira:

a) classe A: todos os dispositivos LoRa implementam esta classe e, podem implementar também as classes B ou C. As características que definem a classe A, são constituídas por permitir a comunicação bi-direcional, onde cada mensagem de `uplink` (mensagem enviada do dispositivo ao servidor de rede LoRaWAN) é seguida por duas mensagens de `downlink` em um curto período de tempo. Para receber outras mensagens de `downlink`, é necessário esperar até a próxima mensagem de `uplink` ser enviada, desta maneira este tipo de classe tem o menor consumo de bateria, porém oferece uma menor flexibilidade para receber os dados do servidor;
b) classe B: permitem agendar novas janelas de `downlink` aos dispositivos, isso é possível devido a uma sincronização com o *gateway* através de pacotes sinalizadores enviados pelos *gateways* periodicamente, desta maneira o servidor de rede sabe quando o dispositivo está “escutando”;
c) classe C: dispositivos que possuem uma janela sempre aberta para receber os dados do servidor a qualquer momento, fechando somente quando estão transmitindo mensagens. A consequência desta classe é possuir um alto consumo de energia do dispositivo. Na Figura 3, é apresentada a comparação entre as classes e as suas características.

Figura 3 – Classes dos end-devices LoRa



Fonte: LoRa Alliance (2015).

Conforme Augustin *et al.* (2016), para que um dispositivo possa fazer parte de uma rede LoRaWAN, o mesmo precisa ser configurado e ativado em um servidor de rede LoRaWAN, desta maneira, existe dois métodos de configuração: Over-The-Air Activation (OTAA) e Activation By Personalization (ABP). Independentemente do método de configuração que o dispositivo irá possuir, as seguintes informações são necessárias:

a) endereço do dispositivo (`DevAddr`): endereço de 32 bits responsável por identificar o dispositivo na rede. O formato deste endereço é da seguinte maneira: os 7 bits mais significantes são usados como um identificador de rede (`NwkID`) e os outros 25 bits restantes são usados como endereço da rede (`NwkAddr`) do dispositivo (AUGUSTIN *et al.*, 2016);
b) identificador da aplicação (`AppEUI`): é um identificador de aplicação único global no espaço de endereço IEEE EUI64, que identifica o responsável pelo dispositivo. Este endereço fica armazenado no dispositivo antes do procedimento de ativação (COMMITTE, 2018);
c) chave de sessão de rede (`NwkSKey`): uma chave utilizada pelo servidor de rede e o dispositivo, a qual serve para calcular e verificar o código de integridade dos dados das mensagens (AUGUSTIN *et al.*, 2016);
d) chave de sessão da aplicação (`AppSKey`): uma chave utilizada pelo servidor de aplicação e dispositivo para criptografar e descriptografar o *payload* das mensagens (COMMITE, 2018).

No método de ativação OTAA, o dispositivo envia uma mensagem de solicitação de conexão à rede e logo em seguida o *gateway* responde com uma mensagem de aceitação, desta forma o dispositivo pode receber as novas chaves de sessão. Este processo é realizado a cada nova sessão que precisa ser criada. Já na ativação ABP, as chaves `NwkSKey` e `AppSKey` ficam armazenadas diretamente nos dispositivos (AUGUSTIN *et al.*, 2018).

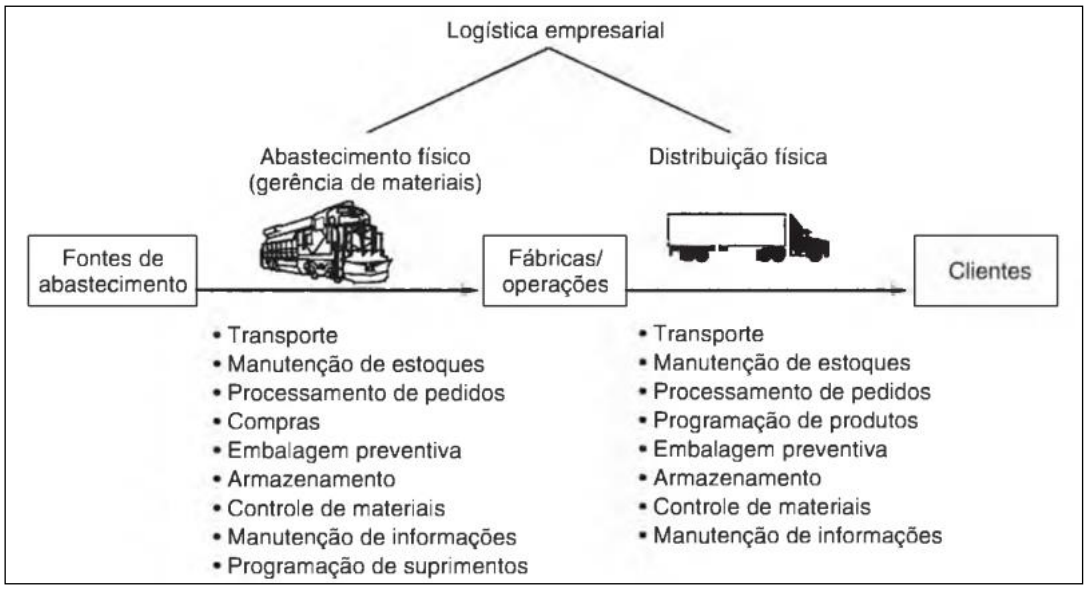
2.3 LOGÍSTICA

Desde as guerras que ocorriam antes de Cristo, a logística já era aplicada em operações militares pelos oficiais da época, onde cada equipe tinha sua organização e momento certo para agir. Ao realizar o avanço de suas tropas, os generais possuíam equipes que disponibilizavam na melhor hora: munição, suplementos, equipamentos, medicamentos e socorro ao campo de batalha.

Hoje, em um contexto mais empresarial, o Council of Logistics Management (CLM) (2001 apud BALLOU, 2004, p. 27), organização formada em 1962 que tem o objetivo de incentivar o ensino do processo de logística, define logística como: “Logística é o processo de planejamento, implantação e controle do fluxo eficiente e eficaz de mercadorias, serviços e das informações relativas desde o ponto de origem até o ponto de consumo com o propósito de atender às exigências dos clientes”.

Ainda segundo o CLM (2001 apud BALLOU, 2004), diversos componentes fazem parte de um sistema típico logístico, como: serviço ao cliente, previsão de demanda, tráfego e transporte, controle de estoque, armazenagem e estocagem, processamento de pedidos e muitos outros como pode der visto na Figura 4.

Figura 4 – Atividades logísticas na cadeia de suprimentos imediata da empresa.



Fonte: Ballou (2004).

De acordo com Ballou (2004), o transporte e a manutenção dos estoques são as atividades logísticas primárias na absorção de custos, a experiência demonstra que cada um deles representará entre metade e dois terços dos custos totais de logística. Ballou (2004) também relata que o transporte é essencial, pelo fato de não haver empresa moderna capaz de operar sem adotar as providências necessárias para a movimentação de seus produtos.

2.4 TRABALHOS CORRELATOS

A seguir serão apresentados três trabalhos correlatos que possuem características semelhantes à proposta deste trabalho. No Quadro 1 detalha o trabalho de Barreiros (2019), que consiste em uma análise e construção de um geolocalizador via Wi-Fi BSSID com transmissão via LoRa. No Quadro 2 será apresentando o trabalho de conclusão de curso de Ferreira e Prazeres (2017), estes que desenvolveram um sistema para acompanhamento de rotas e entregas de uma empresa de transporte. Por fim, o Quadro 3 aborda uma prova de conceito desenvolvida por Salazar-Cabrera, de la Cruz e Molina (2019), um sistema para monitoramento dos ônibus de transporte público utilizando a tecnologia LoRa e arquitetura Intelligent Transportation System (ITS).

Quadro 1 – Análise e Construção de um Geolocalizador via Wi-fi BSSID com transmissão LoRa

Referência	Barreiros (2019)
Objetivos	Desenvolver um protótipo capaz de transmitir dados de BSSIDs e de posicionamento GPS por uma rede LoRa, realizando integrações com APIs de geolocalização para visualização dos dados no Google Earth e Matlab.
Principais funcionalidades	No dispositivo TTGO T-BEAM periodicamente é executada uma rotina responsável por coletar endereços MACs de redes Wi-Fi próximas, juntamente com a posição válida do GPS do dispositivo e enviá-los em um *payload* através da rede LoRaWAN para um servidor de rede da Everynet. O servidor de aplicação estabelece conexão com a Everynet através de *websocket* e filtra as mensagens de `uplinks` recebidas, em seguida realiza requisições Hypertext Transfer Protocol (HTTP) POST para três APIs de geolocalização, informando no corpo da requisição os dados enviados pelo dispositivo. Com os retornos das requisições e a posição do GPS, é gerado arquivos `kml` e `txt`, para que os pontos sejam visualizados no Google Earth e analisados no Matlab respectivamente, dessa forma, podendo comparar os métodos de geolocalização via BSSID e GPS.
Ferramentas de desenvolvimento	TTGO T-BEAM, LoRaWAN, Everynet, Node.js, WebSocket, Google Earth, Visual Studio Code e PlatformIO.
Resultados e conclusões	Nos testes também foi utilizado o GPS GARMIN Oregon 600 para obter a geolocalização, dessa maneira, os testes incluíam dois dispositivos capazes de obterem a geolocalização via GPS e três vias BSSID. Nos resultados obtidos, foi possível identificar que os dados coletados pelos GPS são compatíveis com o esperado, apresentando valores baixos de variação de suas posições. Já na geolocalização via BSSID, a precisão e acurácia é um pouco dispersa em relação ao GPS, porém, houve exceções em que algumas APIs foram melhores ou próximas das obtidas pelo GPS.

Fonte: elaborado pelo autor.

Quadro 2 – Sistema móvel para acompanhamento de rotas e entregas de uma empresa de transporte

Referência	Ferreira e Prazeres (2017)
Objetivos	Desenvolver um sistema para gerenciamento de rotas e acompanhamento das entregas realizadas por pequenas e médias empresas de transporte.
Principais funcionalidades	Composto por uma aplicação web onde a transportadora é capaz de montar as entregas das notas e fazer um gerenciamento diverso. Possui também uma tela onde a central pode acompanhar o status e trajeto de cada veículo em um mapa, sendo possível ter um detalhamento das cargas em entregas. Além disso, o trabalho possui um aplicativo móvel Android onde o motorista é capaz de ter as informações da carga através de um pareamento por QR Code gerado em uma ficha de romaneio da carga. No dispositivo o motorista pode ver o destino de cada nota, informações do cliente, alterar status da nota e marcar a posição atual, utilizando o GPS do dispositivo.
Ferramentas de desenvolvimento	Zxing, OpenStreetMap, Vaadin, Android Studio, Java, Eclipse, PostgreSQL e Enterprise Architect.
Resultados e conclusões	Ao final do desenvolvimento foi aplicado um questionário a uma distribuidora de alimentos em Florianópolis. Houve algumas classificações neutras em relação a redução de custos, pois não houve tempo necessários à ser testado, porém, no geral o sistema obteve 4,60 pontos de 5 no total, não deixando de atender nenhum dos aspectos avaliados pelos usuários do questionário (FERREIRA; PRAZERES, 2017, p. 127).

Fonte: elaborado pelo autor.

Quadro 3 – Proof of Concept of an IoT-Based Public Vehicle Tracking System, Using LoRa (Long Range) and Intelligent Transportation System (ITS) Services

Referência	Salazar-Cabrera, de la Cruz e Molina (2019)
Objetivos	Desenvolver uma Proof of Concept (POC) para rastreamento dos ônibus do transporte público utilizando a tecnologia LoRa.
Principais funcionalidades	Neste projeto a equipe implementou a sua própria infraestrutura de comunicação LoRa, no desenvolvimento foi utilizado dois microcontroladores ESP32 LoRa com cada um conectado com o GPS Ublox Neo 6M. Estes dispositivos seriam responsáveis por coletar a localização do veículo e enviar via LoRa a um *gateway* configurado pela equipe. Foi desenvolvida uma aplicação web, cuja possui um mapa com a última localização do veículo juntamente com o seu histórico de localidade. As informações que a aplicação web utiliza estão armazenadas em um banco de dados, o qual é carregado com dados através de requisições POST realizadas pelo *gateway*.
Ferramentas de desenvolvimento	XAMPP (Apache, PHP, MariaDB e phpMyAdmin), Gateway Dragino LG01, Arduino IDE e ESP32 LoRa.
Resultados e conclusões	A equipe chegou a realizar alguns testes implementando o protocolo LoRaWAN, mas nos testes finais acabaram não o utilizando, ou seja, apenas enviavam os dados via LoRa ao *gateway*. Primeiramente nos testes finais, houve uma perda maior que 60% das mensagens enviadas ao *gateway*, parte disso devido aos obstáculos que tinham entre a comunicação dos dispositivos, principalmente dos edifícios. Descobriram também que aumentando o valor do parâmetro SF para 10, anteriormente em 7, diminuiu o percentual de perda de mensagens para apenas 28%.

Fonte: elaborado pelo autor.

3 DESCRIÇÃO DO SISTEMA

Esta seção descreve as etapas do desenvolvimento da aplicação. Na subseção 3.1, é apresentada a especificação da aplicação, nesta que são descritos os principais requisitos, os casos de uso do sistema e o esquema de montagem do dispositivo. A subseção 3.2 descreve de forma detalha a implementação da aplicação, destacando detalhes do servidor de rede LoRaWAN, dispositivo e a aplicação web.

3.1 ESPECIFICAÇÃO

No Quadro 4 são apresentados os principais Requisitos Funcionais (RF) da aplicação desenvolvida. Em complemento, o Quadro 5 apresenta os principais Requisitos Não Funcionais (RNF).

Quadro 4 – Requisitos Funcionais

RF01: A aplicação web deverá permitir os usuários cadastrarem seus endereços
RF02: A aplicação web deverá obter a latitude e longitude do endereço cadastrado através de uma requisição a uma API de geocodificação
RF03: A aplicação web deverá permitir os usuários realizarem pedidos

RF04: A aplicação web deverá permitir o usuário administrador cadastrar dispositivos, veículos, usuários e produtos
RF05: A aplicação web deverá permitir os usuários administradores a montar cargas a serem entregues
RF06: A aplicação web deverá permitir imprimir ficha de romaneio da respectiva carga
RF07: A aplicação web deverá permitir escanear os códigos de barras contidos na ficha de romaneio ao iniciar a entrega de uma carga
RF08: A aplicação web deverá disponibilizar uma visão de mapa com as localizações da rota percorrida por cada carga, juntamente com as localizações dos pedidos contidos na respectiva carga
RF09: A aplicação web deverá permitir finalizar a entrega de uma carga
RF10: O dispositivo deverá enviar as coordenadas do veículo a cada 10 minutos caso forem válidas

Fonte: elaborado pelo autor.

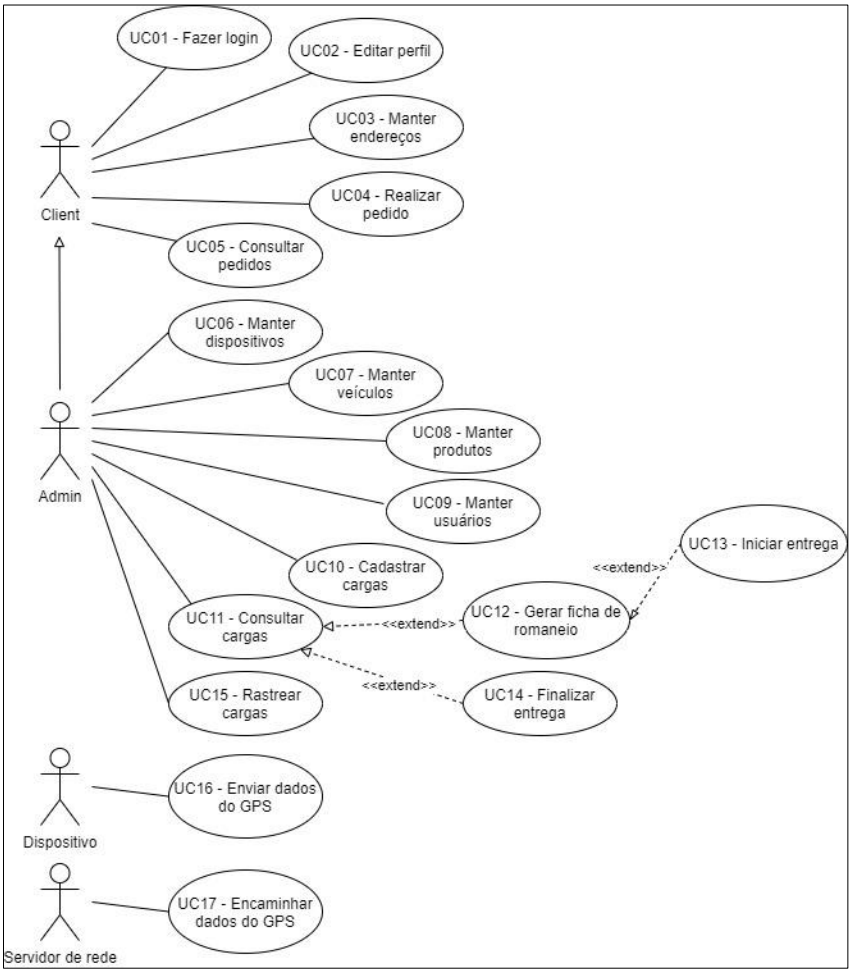
Quadro 5 – Requisitos Não Funcionais

RNF01: O backend da aplicação web deverá ser um Web Service desenvolvido com Node.js
RNF02: O frontend da aplicação web deverá ser desenvolvido com ReactJS
RNF03: O software do dispositivo deverá ser desenvolvido na linguagem C++ (Arduino framework)
RNF04: O dispositivo utilizará o módulo de desenvolvimento WiFi LoRa 32 (V2) e o GPS GY-NEO6MV2
RNF05: O servidor de rede LoRaWAN que deverá ser utilizado é da empresa KORE Wireless
RNF06: O dispositivo deverá utilizar a rede LoRaWAN para envio dos dados a plataforma KORE
RNF07: A comunicação entre o servidor de rede LoRaWAN e aplicação web será através de requisições HTTP
RNF08: A aplicação web utilizará o banco de dados relacional PostgreSQL para armazenamento dos dados de negócio
RNF09: A aplicação web utilizará o banco de dados não relacional MongoDB para armazenamento de todas as requisições enviadas do servidor de rede LoRaWAN à aplicação web

Fonte: elaborado pelo autor.

Na Figura 5, é possível observar o diagrama de caso de uso contendo as principais funcionalidades do sistema.

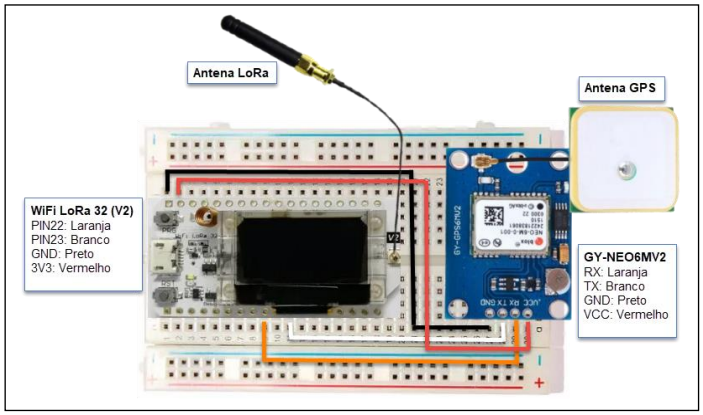
Figura 5 – Diagrama de caso de uso



Fonte: elaborado pelo autor.

Na Figura 6 apresenta-se o esquema de montagem do dispositivo. Para a construção do dispositivo, foi utilizado o módulo de desenvolvimento WiFi LoRa 32 (V2) da Heltec Automation, esse que possui o microcontrolador ESP32, comunicação via WiFi, Bluetooth e LoRa (chip SX1276) operando nas frequências de 868MHz à 915MHz, circuito de carga e descarga de bateria lítio, Display OLED de 0,96 polegadas e uma antena LoRa. Em relação ao GPS, foi escolhido o módulo GY-NEO6MV2 da empresa Ublox, onde por padrão vem com uma antena de cerâmica. A alimentação do dispositivo é feita através de um carregador portátil de 12000mAH, oferecendo carga de 5V DC. O dispositivo foi montado sobre uma protoboard de 400 pinos.

Figura 6 – Esquema de montagem

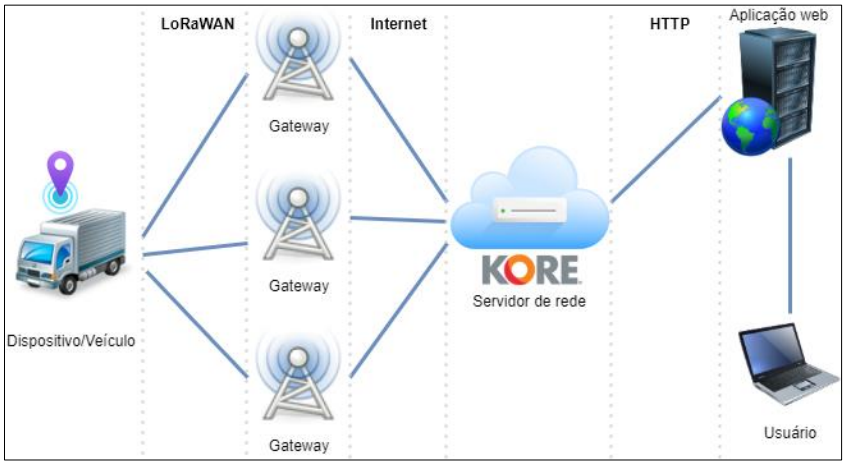


Fonte: elaborado pelo autor.

3.2 IMPLEMENTAÇÃO

O sistema desenvolvido é composto por três itens principais, o servidor de rede LoRaWAN, o dispositivo que captura a geolocalização da carga e envia via LoRaWAN e a aplicação web, onde que toda a regra de negócio é implementada. Na Figura 7, é possível observar a arquitetura do trabalho desenvolvido. Os três itens serão descritos a seguir nas suas subseções correspondentes.

Figura 7 – Arquitetura do sistema



Fonte: elaborado pelo autor.

3.2.1 Servidor de rede LoRaWAN (KORE)

Para a utilização da rede LoRaWAN, não é preciso necessariamente contratar um plano ou serviço de algum fornecedor, é possível comprar um ou mais *gateways* e próprio implementar a rede. Existe a plataforma The Things Network (TTN), a qual é uma infraestrutura de comunicação colaborativa para IoT, mais especificamente a LoRaWAN, onde já possui também um servidor de rede para gerenciar os dispositivos e é gratuita de utilização. Porém, em Gaspar e Blumenau não possui ainda uma infraestrutura nessa plataforma, seria necessário comprar um *gateway* e cadastrá-lo na TTN. Esse novo gateway se tornaria público, ou seja, outros usuários que forem utilizar a plataforma da TTN, se o dispositivo estiver próximo a esse novo *gateway*, o pacote enviado pelo dispositivo será recebido por ele e encaminhado ao servidor de rede da plataforma.

Contudo, neste trabalho, foi escolhido o servidor de rede da empresa KORE Wireless, o qual é um serviço que surgiu da parceria entre a KORE e a American Tower, essa que possui antenas e *gateways* da rede LoRaWAN em diversos países, incluindo o Brasil, onde está presente em mais de 220 cidades. Para se obter acesso, foi necessário entrar em

contato com a KORE e contratar um dos planos oferecidos. A diferença entre os planos é basicamente a quantidade de mensagens de `uplink` e `downlink` que um dispositivo poderá enviar e receber no dia. O plano escolhido para desenvolvimento deste trabalho, foi o plano G, permitindo 160 mensagens de `uplink` e 8 mensagens de `downlink` por dia. Pode ser visto no Quadro 6, os planos de duração anual oferecidos pela KORE Wireless atualmente.

Quadro 6 – Preços LoRaWAN por dispositivo por ano

Plano	Conectividade por dispositivo / Ano	Plataforma de conectividade KORA por dispositivo x Ano (até 5000 dispositivos)	Plataforma de conectividade KORA por dispositivo x Ano (> 5000 dispositivos)	Qnt. mensagens Uplink/dia	Qnt. mensagens Downlink/dia
PP	R$9,00	R$0,72	R$0,48	5	1
P	R$15,00	R$1,08	R$0,72	10	2
M	R$25,00	R$1,92	R$1,32	75	4
G	R$34,00	R$2,04	R$1,56	160	8

Fonte: KORE Wireless (2020).

Com o acesso à plataforma, primeiramente é necessário cadastrar uma `Aplicação`, essa que será responsável por conter os dispositivos associados a ela e também possuir os `encaminhamentos`. Os `encaminhamentos` são responsáveis por encaminhar os dados recebidos no servidor LoRaWAN à um servidor de aplicação, podendo ser através do protocolo HTTP ou Message Queuing Telemetry Transport (MQTT). Na Figura 8, é possível observar os dois `encaminhamentos` que foram cadastrados na plataforma, os quais são descritos na subsecção 3.2.3. O primeiro deles é responsável por enviar todos os tipos de mensagens ao *endpoint* `/lora` e o segundo, somente irá enviar mensagens do tipo `uplink` ao *endpoint* `/cargos/geolocation`, ambos através do protocolo HTTP.

Figura 8 – Encaminhamentos

Fonte: Kore Wireless (2020).

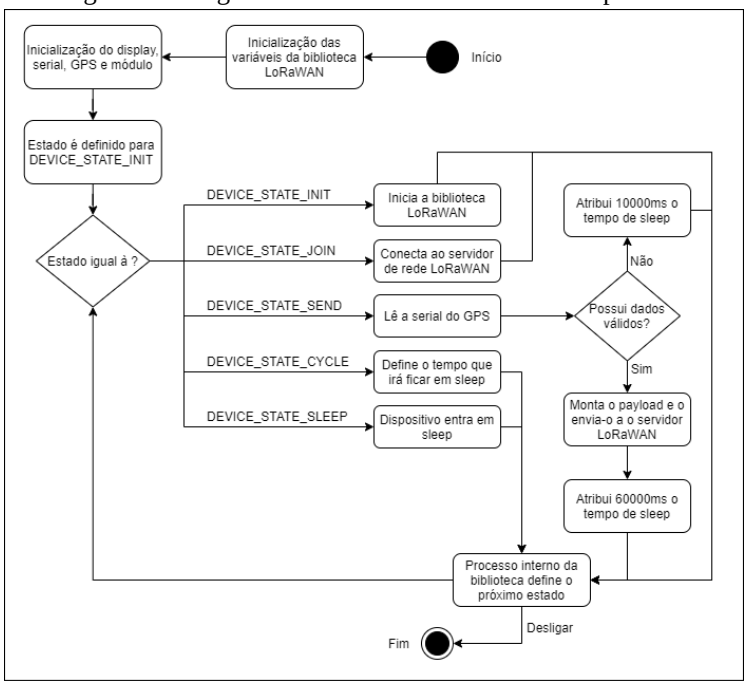
O próximo passo, é necessário cadastrar um dispositivo na plataforma, sendo assim, alguns parâmetros são essenciais para a comunicação LoRa ocorrer, como o Device EUI, Application EUI, classe do dispositivo, o método de conexão segura, região e entre outros que podem ser tornar obrigatórios, dependendo de quais opções forem escolhidas. A configuração do dispositivo utilizada neste trabalho foi ativação pelo método OTAA, criptografia de segurança NS, classe A e banda LA915-928A.

3.2.2 End-device

O dispositivo é a parte responsável na aplicação por obter e enviar através da rede LoRaWAN, a latitude e longitude do veículo que está associado ao mesmo. O código do dispositivo foi desenvolvido na linguagem C++ utilizando o framework Arduino. Primeiramente foi necessário baixar o último pacote do ESP32 no framework Arduino e também, a biblioteca do ESP32 desenvolvida pela Heltec, assim fornecendo todo o suporte necessário para a implementação. Para transmissão dos dados via LoRa, foi utilizada a biblioteca Heltec ESP32 LoRaWAN, essa que foi desenvolvida especialmente para os produtos da Heltec e para que possa ser utilizada, precisa ter primeiramente uma licença única ativada através do site oficial. No seguinte link: http://resource.heltec.cn/search é possível ativar a licença do dispositivo que irá utilizá-la, basta informar o ChipID do ESP32 em questão, dessa maneira será retornado o modelo do dispositivo, o ChipID, a licença e local e data de fabricação. Foi utilizada a biblioteca TinyGPS++ para comunicação com o módulo GPS, facilitando o desenvolvimento e a legibilidade de código, visto que a mesma provê diversas funções que realizam o *parsing* dos dados NMEA enviados pelo GPS.

A biblioteca LoRaWAN desenvolvida pela Heltec, vem com alguns exemplos quando instalada na Arduino IDE deixando a cargo do desenvolvedor apenas carregar as variáveis essenciais para o funcionamento e, implementar a regra de negócio para envio do *payload*. Dessa maneira, grande parte de como é gerenciado a troca das mensagens LoRa, interação com o display OLED, controle do modo *sleep*, etc. é abstraído. Sendo assim, foi utilizado o exemplo `OTAA_OLED.ino` e adaptando-o conforme as necessidades deste trabalho. Basicamente dentro da função `loop` padrão, existe apenas um `switch` que lê a variável `deviceState`, esta que o seu valor é alterado para cada um dos valores a seguir conforme a execução do software: `DEVICE_STATE_INIT`, `DEVICE_STATE_JOIN`, `DEVICE_STATE_SEND`, `DEVICE_STATE_CYCLE` ou `DEVICE_STATE_SLEEP`. Na Figura 9, é possível observar o diagrama de estados do software do dispositivo. Neste trabalho será detalhado apenas o trecho de código quando a variável tiver o valor de `DEVICE_STATE_SEND`. No Apêndice A, é disponibilizado o código completo do software executado pelo dispositivo.

Figura 9 – Diagrama de estados do software do dispositivo



Fonte: elaborado pelo autor.

Quando a variável `deviceState` atender a condição `DEVICE_STATE_SEND`, será executada a função `setLatitudeLongitude()`, que é responsável por ficar lendo durante 5 segundos a serial do GPS e, caso os valores recebidos possuírem uma localização válida, as variáveis `lat` e `lon` são atribuídas com os valores de latitude e longitude obtidos pelo GPS, caso contrário, essas mesmas variáveis recebem o valor 0. No Quadro 7 é possível ver em detalhes a função `setLatitudeLongitude()`.

Quadro 7 – Função setLatitudeLongitude()

```
static void setLatitudeLongitude(){
  for (unsigned long start = millis(); millis() - start < 5000;){
    while (ss.available()){
      if (gps.encode(ss.read())){
        if(gps.location.isValid()){
          lat = gps.location.lat() * 100000;
          lon = gps.location.lng() * 100000;
        }else{
          lat = 0;
          lon = 0;
        }
      }
    }
  }
}
```

Fonte: elaborado pelo autor.

Continuando à condição `DEVICE_STATE_SEND`, é verificado quais valores foram atribuídos as variáveis `lat` e `lon`, caso ambas possuírem o valor 0, então é apenas atribuído o valor 60000 à variável `appTxDutyCycle`, essa que armazena o tempo em milissegundos do próximo envio do *payload*. Se ambas as variáveis possuírem valores válidos, é montado o *payload* de 8 bytes em hexadecimal, sendo os primeiros 4 bytes a latitude e os últimos 4 bytes a longitude. Com o *payload* estruturado, o mesmo é enviado através da rede LoRaWAN sem a confirmação de resposta, em decorrer do limite de mensagens de `downlink` diária. Em seguida, é atribuído o valor de 600000 à variável `appTxDutyCycle`,

ou seja, em 10 minutos o próximo envio de dados será realizado. No Quadro 8 é exibido a condição `DEVICE_STATE_SEND` e a função responsável por montar o *payload*.

Quadro 8 – Estruturação e envio do payload

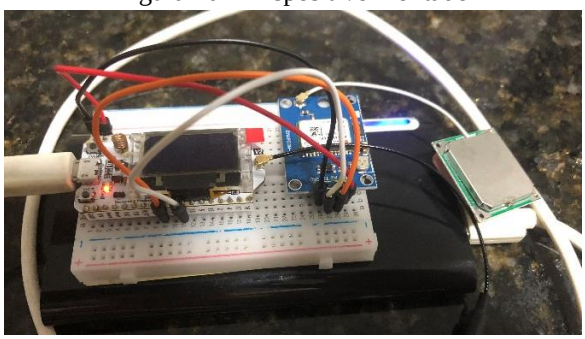
```
static void prepareTxFrame( uint8_t port ){
    appDataSize = 8;//AppDataSize max value is 64
    appData[0] = (lat >> 24) & 0xFF;
    appData[1] = (lat >> 16) & 0xFF;
    appData[2] = (lat >> 8) & 0xFF;
    appData[3] = lat & 0xFF;
    appData[4] = (lon >> 24) & 0xFF;
    appData[5] = (lon >> 16) & 0xFF;
    appData[6] = (lon >> 8) & 0xFF;
    appData[7] = lon & 0xFF;
}

void loop(){
  switch( deviceState ){
    // case DEVICE_STATE_INIT, case DEVICE_STATE_JOIN, default, ...
    case DEVICE_STATE_SEND:{
      setLatitudeLongitude();
      if(lat != 0 && lon != 0) {
        appTxDutyCycle = 300000;
        prepareTxFrame( appPort );
        LoRaWAN.send(loraWanClass);
      }else{
        appTxDutyCycle = 60000;
      }
      deviceState = DEVICE_STATE_CYCLE;
      break;
    }
  }
}
```

Fonte: elaborado pelo autor.

Após o envio dos dados, a condição é `DEVICE_STATE_CYCLE` é atendida e o dispositivo agenda a próxima execução do software, colocando o dispositivo em modo o *sleep* durante o tempo definido na variável `appTxDutyCycle`. A imagem do dispositivo montado é possível ver na Figura 10 e os custos totais no Apêndice C.

Figura 10 – Dispositivo montado



Fonte: fotografado pelo autor.

3.2.3 Aplicação web

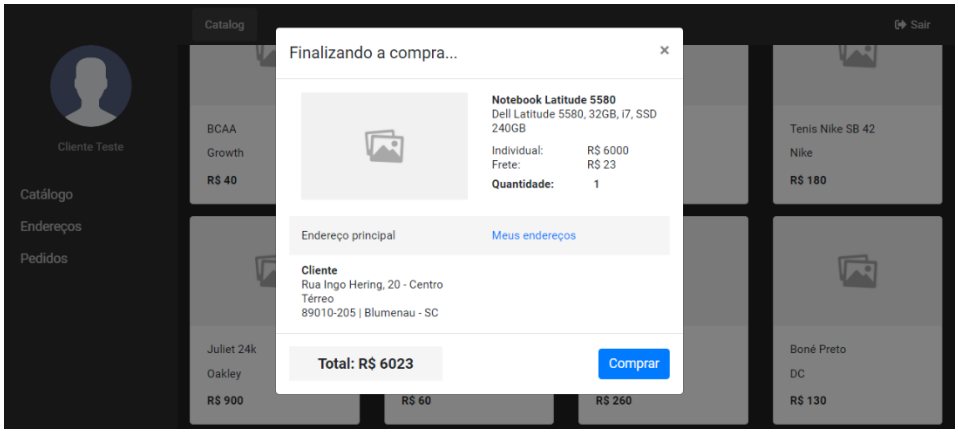
A aplicação web possui a responsabilidade de realizar a interação com o usuário e manipular os dados no banco de dados, dessa maneira, foi divido em duas aplicações, uma responsável por executar o *frontend* e outra por executar o *backend*. O *frontend* foi desenvolvido na linguagem de programação Javascript, com a biblioteca adicional ReactJS. Para aproveitar o conhecimento na mesma linguagem de programação, o *backend* também foi desenvolvido com Javascript, utilizando a plataforma Node.js, assim, foi desenvolvido um *web service* RESTful com diversas rotas HTTP disponíveis para interação com o banco de dados PostgreSQL através do Object Relational Mapper (ORM) Sequelize. Será descrito nesta subseção, as principais funcionalidades disponíveis para os usuários interagir e os principais processamentos realizados no *backend*.

Ao acessar a aplicação web, o usuário precisa realizar o *login* na plataforma ou criar uma nova conta caso não possua. Dependendo do perfil do usuário, sendo eles: `client`, `driver` ou `admin`, certas opções e funcionalidades são disponibilizadas. Os perfis `client` e `driver` possuem as mesmas visões, diferenciando apenas que um usuário do tipo `driver`, pode ser inserido como um motorista do veículo de uma carga. Perfis do tipo `driver` e `admin` somente podem ser criados por usuários `admin`, esses que também tem a possiblidade de criar usuários do tipo `client`. Independente do perfil que realizou o *login*, caso o mesmo não possua um endereço cadastrado, será redirecionado para a tela de cadastro de endereço, pois é necessário possuir ao menos um endereço principal cadastrado para realizar um pedido na aplicação.

Na tela de cadastro de endereço, basta o usuário informar o CEP do seu endereço que as informações de rua, bairro, cidade, estado e estado são preenchidas automaticamente no formulário, isso ocorre devido a uma requisição assíncrona que é realizada pela biblioteca cep-promise. Essa biblioteca busca nos serviços dos Correios, ViaCEP e WideNet as informações sobre o CEP, assim resolvendo a *Promise* com fornecedor que responder mais rápido. O formulário ainda disponibiliza os campos `Número` e `Complemento`, este último que é opcional de preenchimento. Ao submeter o formulário, uma requisição POST é enviada para o *backend*, passando os dados do formulário no corpo da requisição. No *backend*, quando a requisição chega, é concatenado a rua, bairro, cidade, estado e número em uma única *string*, com isso, é realizado uma requisição à API de geocodificação da empresa Mapbox. Para utilização da API, é necessário realizar um cadastro no site da Mapbox e obter um *token* válido, esse *token* é necessário informar em todas as requisições da API. Com o intuito de realizar um filtro mais específico, é passado o parâmetro `country` com o valor BR, filtrando apenas possíveis endereços no Brasil. O retorno da API é um JavaScript Object Notation (JSON) com diversas informações sobre o endereço passado na *string*, incluindo um *array* ordenado decrescentemente dos endereços mais relevantes. Dessa forma, é selecionado o primeiro endereço do *array* e é extraído do campo `coordinates` os valores de latitude e longitude do endereço. Com os valores informados pelo o usuário e os valores obtidos através da API, é criado o endereço no banco de dados com sua respectiva geolocalização.

Já com um endereço principal cadastrado, o usuário pode realizar pedidos na aplicação. Como o intuito deste trabalho não é desenvolver um *e-commerce*, a realização de um pedido foi feita de uma maneira simplificada, onde o usuário pode escolher apenas um produto por pedido e o frete é um valor inteiro aleatório entre 0 e 40. Na Figura 11 pode-se observar o usuário realizando um pedido na aplicação. Após realizar o pedido, o usuário pode consultar todos os seus pedidos realizados e o histórico de movimentação de cada um deles.

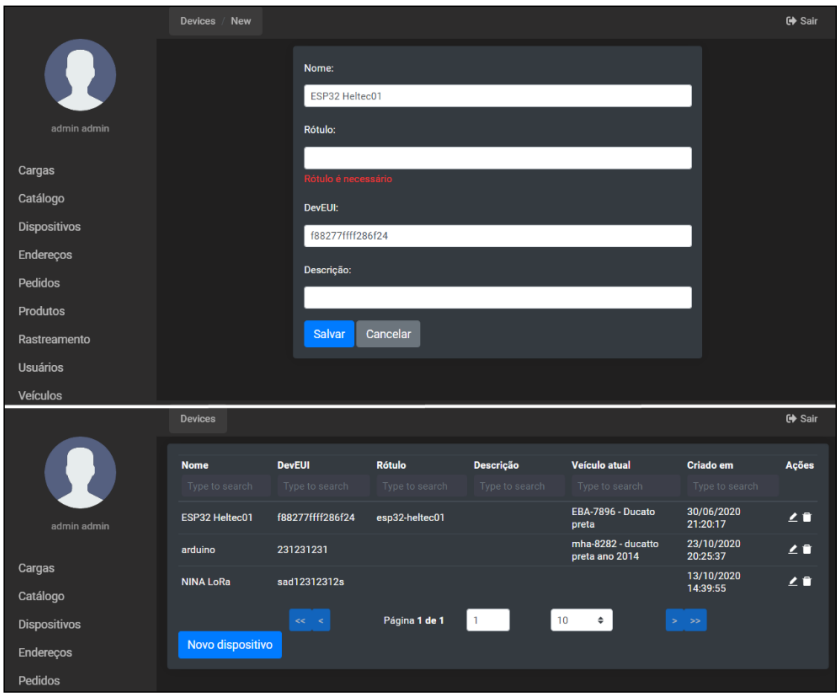
Figura 11 – Tela de realização de um pedido



Fonte: elaborado pelo autor.

As funcionalidades descritas a partir deste ponto, são somente disponíveis para usuários do tipo `admin`. Na aplicação, é possível consultar e cadastrar usuários, produtos, dispositivos e veículos, onde que um veículo precisa ter um dispositivo associado na hora da criação. Para essas funcionalidades descritas, as telas possuem um padrão, onde acessando por exemplo, a sessão `Dispositivos`, é exibido uma tabela paginada com os dispositivos cadastrados, sendo possível filtrar e ordenar os dados contidos na tabela. Na mesma tela, existe um botão `Novo dispositivo` que irá redirecionar para a tela de cadastro de dispositivo, onde que nessa, existe um formulário com seus devidos campos e validações necessárias, um botão de `Salvar` e um botão de `Cancelar`. Na Figura 12 é possível observar as telas de cadastro e listagem de dispositivos respectivamente, as quais são semelhantes as demais descritas acima. Na tabela, é possível observar as ações de editar e excluir, representadas por um ícone de um lápis e uma lixeira, porém, o endereço do usuário é o único item que pode ser editado, nas demais telas não é possível editar nem excluir (incluindo o endereço) um item já cadastrado. No cadastro de um dispositivo, é fundamental que o campo `DevEUI` esteja preenchido igualmente ao dispositivo cadastrado na plataforma KORE, pois é a partir do `DevEUI` enviado nas mensagens do servidor LoRaWAN que, a inserção e atualização dos dados associados a este dispositivo/veículo é realizado. Os campos `Nome` e `Rótulo` também é aconselhado se manter iguais ao cadastro na KORE, pois assim irá existir um padrão e, consequentemente uma maior facilidade de gerenciamento dos dispositivos.

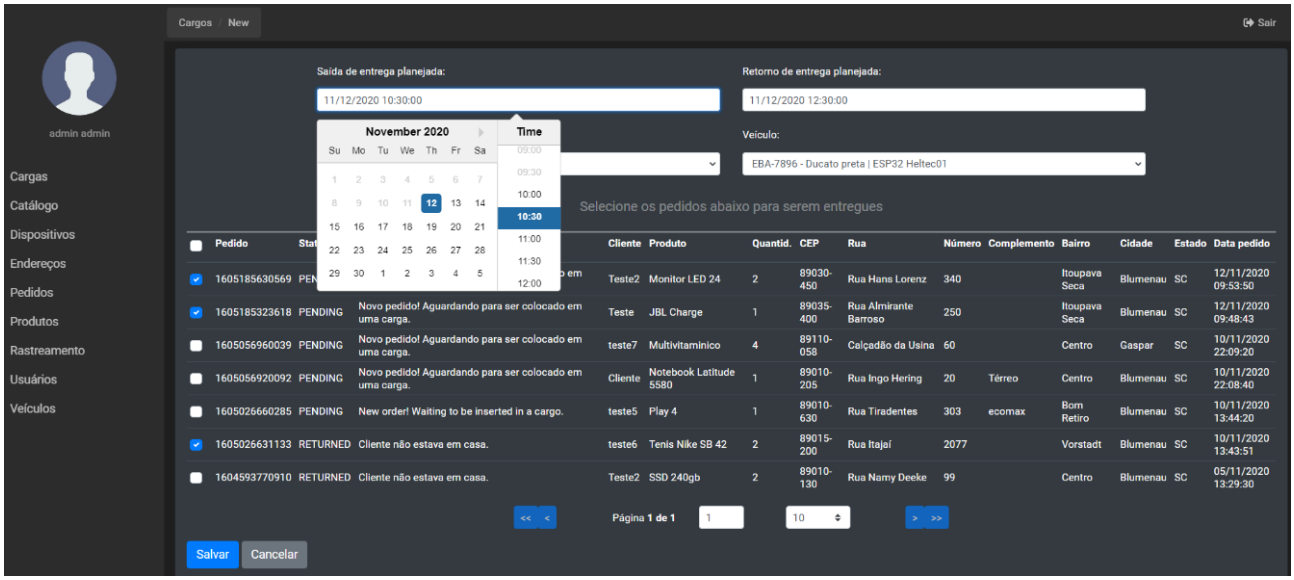
Figura 12 – Telas de cadastro e listagem de dispositivos



Fonte: elaborado pelo autor.

Com ao menos um pedido realizado, o `admin` pode criar uma carga contendo os pedidos que serão entregues nesta carga. Acessando a seção `Cargas`, é disposto de uma tabela contendo as principais informações das cargas cadastradas na aplicação, juntamente com as opções de criar uma nova carga e inicializar a entrega de uma carga, essa última opção, se dá através de um botão com ícone de caminhão em cada uma das linhas da tabela. No cadastro de uma carga, os campos `Saída de entrega planejada`, `Retorno de entrega planejada`, `Motorista` e `Veículo` são essenciais a serem informados, também ao menos um pedido precisa ser selecionado. Nos dois campos de datas, foi utilizado a biblioteca `react-datepicker`, facilitando toda a validação de datas no *frontend*, não permitindo datas e horas passadas, não deixando selecionar uma data e hora passada no campo `Retorno de entrega planejada` em relação ao `Saída de entrega planejada` e, uma série de outras configurações e validações possíveis. É possível observar na Figura 13 a tela de cadastro de uma nova carga na aplicação. Ao submeter o formulário para o *backend*, é realizado diversas outras validações com os dados que chegaram, as principais são: Validações de datas passadas novamente, verificado se as datas de planejamento e retorno para o motorista ou veículo já não coincidem com outras datas de outras carga que os mesmos estão associados e também, outras validações que são feitas caso a requisição seja feita manualmente, sem que seja montada pelo *frontend*. Após passar por todas as validações, a carga é cadastrada no banco de dados e fica com o status `CLOSED`, ou seja, aguardando apenas de ser inicializada a entrega.

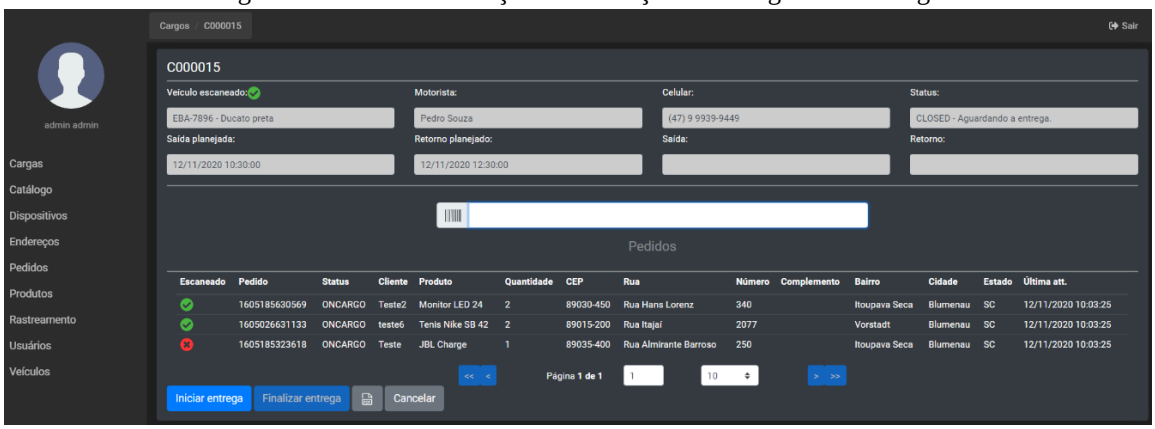
Figura 13 – Tela de cadastro de carga



Fonte: elaborado pelo autor.

Existindo cargas cadastradas na aplicação, o próximo passo é iniciar a entrega das mesmas, sendo assim, é necessário acessar a seção `Cargas` e clicar no botão com ícone de caminhão na carga desejada. Nessa tela é exibido as informações da carga e os pedidos contidos nela. Para poder iniciar a entrega da carga, é necessário escanear o código de barras do veículo designado para realizar a entrega e, todos os pedidos contidos nesta carga. Como neste trabalho não foram feitos os processos de faturamento, geração de nota, impressão da nota, embalagem do pedido, etc. foi disponibilizado um botão com um ícone de Portable Document Format (PDF), ao clicar nesse botão, será gerado a ficha de romaneio da carga. Nessa ficha, contém os dados essenciais da carga, juntamente com o código de barras do veículo e os demais códigos de barra dos pedidos. O operador que estiver inicializando a entrega da carga, imprimi a ficha de romaneio, escanea os códigos de barra, carrega o veículo com os pedidos e entrega a ficha de romaneio ao motorista, o qual irá utilizá-la para checar o endereço de entrega de cada um dos pedidos e coletar as assinaturas dos clientes. Caso não seja possível realizar a entrega do pedido ao cliente, como por exemplo, o mesmo não estar em casa, o motorista informa na ficha que o pedido retornou e escreve uma observação. No Apêndice B é possível ver o conteúdo de uma ficha de romaneio e na Figura 14, a tela de inicialização de entrega de uma carga.

Figura 14 – Tela inicialização e finalização da entrega de uma carga

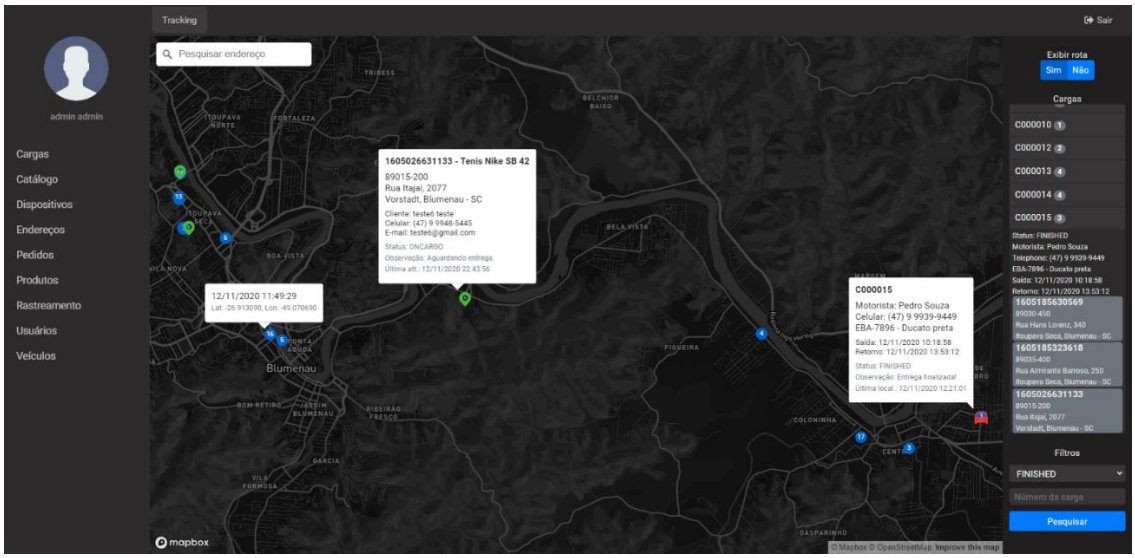


Fonte: elaborado pelo autor.

Já existindo cargas sendo entregues, na seção `Rastreamento` é possível observar em um mapa interativo a localidade das cargas, juntamente com os destinos de entrega de cada um dos pedidos. A biblioteca de mapas utilizada, foi novamente da empresa Mapbox, utilizando o mesmo *token* que o *backend* usa para consumir a API. Para facilitar o trabalho da utilização do Mapbox com ReactJS, foi utilizado a biblioteca `react-map-gl`, a qual é mantida pela Uber. Na tela é disponibilizado filtros para pesquisar pelo número da carga, o status da carga (`ONDELIVERY`, `FINISHED` ou ambos), um endereço no mapa e se deseja exibir a rota tracejada pela carga filtrada. Aplicando o filtro desejado, as cargas serão exibidas no canto direito da tela, clicando em uma delas, será exibido no mapa, a atual localização da mesma através de um ícone vermelho de um veículo e, todos os destinos de entrega em ícones verde. Na Figura 15 é possível observar

uma carga já finalizada, exibindo informações de um pedido, da carga e a rota percorrida. Selecionando os ícones da carga, destino ou numeração sequencial da rota, é possível ver as informações dos mesmos.

Figura 15 – Tela de rastreamento de cargas



Fonte: elaborado pelo autor.

Conforme informando anteriormente, o servidor de rede LoRaWAN encaminha os dados para dois *endpoints*, o *endpoint* `/lora` é responsável por armazenar no banco de dados não relacional MongoDB, todos os tipos de mensagem enviada pelo servidor de rede, isso se dá para manter um histórico das mensagens enviadas caso seja necessário consulta-las, pois na plataforma KORE, as mensagens não são mantidas. Já o *endpoint* `/cargos/geolocation`, é responsável por receber os dados de `uplink`, ou seja, será recebido apenas os dados da geolocalização enviada pelo dispositivo. Quando uma requisição é recebida no *endpoint* `/cargos/geolocation`, primeiramente é validado se a estrutura do corpo da requisição é igualmente a enviada pelo servidor de rede, logo, também é validado se o tipo da mensagem é de `uplink`, por último, é verificado se o `DevEUI` do dispositivo está cadastrado no banco de dados e possui um veículo associado. Caso as validações sejam atendidas, primeiramente é decodificado o *payload* de Base64 para hexadecimal, em seguida é divido a *string* no meio em duas novas e, é convertido ambas as *strings* de hexadecimal para inteiro, em seguida, para número de ponto flutuante. Dessa forma, a primeira *string* irá armazenar a latitude a segunda a longitude, bastando apenas realizar o cadastro de ambas no banco de dados para o veículo/carga associado ao dispositivo. É possível observar no Quadro 9 a decodificação do *payload* recebido.

Quadro 9 – Decodificação do payload

```
// transforma de hexadecimal para inteiro, mantendo o "-" da localização
function hexToSignedInt(hex) {
  if (hex.length % 2 !== 0) {
    hex = `0${hex}`;
  }
  let num = parseInt(hex, 16);
  const maxVal = 2 ** ((hex.length / 2) * 8);
  if (num > maxVal / 2 - 1) {
    num -= maxVal;
  }
  return num;
}
// decodifica o payload de Base64 para hexadecimal
const coordsHex = Buffer.from(req.body.params.payload, 'base64').toString(
  'hex'
);
// divide a string na metade
const hexSplited = splitString(coordsHex);
const latHex = hexSplited[0];
const lonHex = hexSplited[1];
// transforma o retorno da função para float
const latitude = hexToSignedInt(latHex) / 100000;
const longitude = hexToSignedInt(lonHex) / 100000;
```

Fonte: elaborado pelo autor.

Após a carga realizar as entregas dos pedidos e retornar ao ponto de origem, é preciso finalizá-la na aplicação. Acessando novamente a seção `Cargas` e clicando sobre o ícone de caminhão da carga desejada, será redirecionado para a mesma tela de inicialização de entrega, porém, o botão de inicializar estará desativado e apenas o botão de finalizar a entrega estará disponível. Dessa maneira, o operador clica no botão `Finalizar entrega` e preenche o `status` e `observação` de cada um dos pedidos, seguindo o que o motorista preencheu na ficha de romaneio. Após isto, a carga receberá o status `FINISHED` na aplicação e, os pedidos que foram entregues ficam com o status `DELIVERED`, já os pedidos

que retornaram, ficam com o status `RETURNED`, sendo necessário colocá-los em outras cargas para serem entregues posteriormente.

4 RESULTADOS

Os testes foram feitos com o intuito de realizar a avaliação da rede LoRaWAN na região de Gaspar e Blumenau, avaliando a sua disponibilidade e comportamento em um dispositivo que está em movimento. Dessa maneira, foram feitos quatro testes com o dispositivo dentro do carro em movimento, assim simulando entregas dos pedidos, esses testes são descritos pela sigla TM, na Tabela 1. Também foram realizados dois testes deixando o dispositivo estático por um período, os quais são referidos por TE na Tabela 1. Ambos os tipos de testes foram feitos para verificar a porcentagem de mensagens enviadas pelo dispositivo que foram entregues com sucesso no servidor de rede LoRaWAN e posteriormente encaminhados a aplicação web. Nos testes, o dispositivo foi configurado para enviar o *payload* a cada 5 minutos, caso possuísse uma localização válida. Foi escolhido esse valor, pois assim foi possível se obter um maior número de mensagens no período de cada um dos testes. Nos testes com o dispositivo em movimento, foram passados por alguns bairros da cidade de Gaspar e Blumenau, já com o dispositivo estático, foi realizado somente em Gaspar.

Os testes foram feitos seguindo todo o fluxo da aplicação, onde primeiro foram criados diversos usuários do tipo `client`, cadastrado no mínimo um endereço para cada um dos usuários e foram feitos diversos pedidos por usuário. Em seguida, com o usuário `admin`, era montado as cargas com os seus respectivos pedidos, gerava-se a ficha de romaneio, lia todos os códigos de barras através de um *handheld* e iniciava-se a entrega. Após isso, a localização da carga já era possível de se verificar no mapa caso o dispositivo enviasse o *payload*. Na Tabela 1, são dispostos os resultados de cada um dos testes, sendo informado a hora inicial e final, o total de mensagens enviadas pelo dispositivo, a quantidade delas que foram armazenadas no banco de dados da aplicação com sucesso e por fim, a quantidade de mensagens que não foram possíveis de serem entregues.

Tabela 1 – Resultado dos testes realizados

Teste	Início	Fim	Total mensagens enviadas	Entregues com sucesso	Não entregues
TM1	09:09	10:44	21	15 (71%)	6 (29%)
TM2	14:21	22:49	85	70 (82%)	15 (18%)
TM3	10:22	12:21	24	18 (75%)	6 (25%)
TM4	14:32	17:14	29	22 (76%)	7 (24%)
TE1	14:54	16:17	18	13 (72%)	5 (28%)
TE2	11:05	13:12	17	14 (82%)	3 (18%)
Total	**N/A**	**N/A**	**194**	**152 (78%)**	**43 (22%)**

Fonte: elaborado pelo autor.

Como pode ser observado na Tabela 1, a duração dos testes variavam, onde alguns foram com poucas paradas e outros que houve um período que o veículo ficou parado em alguns destinos por um período de tempo, como no TM2. No total, foram enviadas 194 mensagens pelo dispositivo, dessas, 152 foram armazenadas com sucesso no banco de dados da aplicação web (78%). As outras 43 mensagens enviadas pelo dispositivo, não foram possíveis de serem entregues ao servidor de rede LoRaWAN, o qual equivale a 22% das mensagens não entregues. Foi observado que por volta do bairro Bela Vista em Gaspar, até a Ponte do Arcos em Blumenau, na maioria dos testes houve mensagens enviadas, porém, não entregues. Essa é uma via onde os veículos trafegam em uma velocidade considerada relativamente alta, em torno de 60 a 70 kmh, podendo assim causar uma dificuldade para se obter rede LoRaWAN. Também foi observado que na cidade de Gaspar foi onde houve a grande partes dos *gaps* nos intervalos de 5 minutos, diferentemente da cidade de Blumenau, a qual foi possível observar uma quantidade bem inferior de *gaps*, conseguindo entregar o *payload* com sucesso na maioria das mensagens enviadas pelo dispositivo.

Quando uma requisição de `uplink` é realizada para a aplicação web, é possível observar no corpo dela um objeto chamado `gps`, nesse objeto é armazenado a latitude, longitude e altitude do *gateway* que capturou o *payload* enviado pelo dispositivo e encaminhou ao servidor LoRaWAN. Foram analisadas as mensagens e foi possível observar que em Gaspar todos os *payloads* foram recebidos por um único *gateway*, localizado no alto de um morro no bairro Gasparinho, em torno de 1,5 km do centro da cidade. Já em Blumenau, percorrendo com o veículo pelo centro da cidade e por alguns bairros em torno, foi observado que os *payloads* foram obtidos por no mínimo 5 *gateways* diferentes, localizados em diferentes pontos da cidade, como no Centro, Vila Formosa, Garcia, Itoupava Seca e Ponta Aguda. Dessa forma, foi possível observar que a área de abrangência da rede LoRaWAN disponibilizada em Blumenau é de maior abrangência em relação a Gaspar.

Em relação a aplicação web, ela foi testada por um usuário, tanto a parte do `client`, quanto de `admin`. O usuário que testou a aplicação web, realizou todo o fluxo necessário e não teve dificuldades para a utilização dela, descrevendo-a como de fácil entendimento e intuitiva, o qual julgou que atenderia os requisitos básicos de uma empresa de logística, na questão de transporte de cargas. Porém, houve sugestões de melhoria, como no momento de criar uma carga, ao

selecionar as datas de `saída de entrega planejada` e `retorno de entrega planejada`, exibir apenas os motoristas e veículos disponíveis nesse intervalo, pois atualmente a validação somente é realizada quando o usuário clicar no botão `Salvar`. Outra sugestão, seria a implementação de planejamento de rotas inteligente, ou seja, ao criar uma carga, a aplicação sugerir os pedidos que são próximos um dos outros e até mesmo, montar a melhor rota a ser percorrida para entrega desses pedidos. No caso, ao imprimir a ficha de romaneio, os pedidos seriam ordenados conforme a rota planejada, bastando o motorista seguir a ordem de pedidos.

5 CONCLUSÕES

Através do desenvolvimento deste trabalho, permitiu-se verificar a eficiência e comportamento da rede LoRaWAN oferecida pelas empresas KORE e American Tower na região de Gaspar e Blumenau, mais especificamente, em um contexto que o dispositivo está constantemente em movimento. Como informado anteriormente, foi necessário a contratação de um plano que permitia apenas certas quantidades de mensagens de `uplink` no dia (160), limitando assim, realizar um monitoramento em tempo real das geolocalizações das cargas sendo entregues. Porém, caso não haja há necessidade de a localização da carga ser a mais atualizada possível, como foi desenvolvido neste trabalho, a rede LoRaWAN atende os requisitos, principalmente na região de Blumenau, onde foi possível observar uma maior abrangência de disponibilidade. Em Gaspar, é mais interessante de ser utilizada com o dispositivo estático, na região mais do centro da cidade, visto que foi possível obter conexão com apenas um único gateway durante os testes.

Em comparação com o trabalho de Salazar-Cabrera, de la Cruz e Molina (2019), eles obtiveram 28% de perda de mensagens enviadas, enquanto neste trabalho, houve apenas uma perda de 22% das mensagens, porém, no trabalho deles não foi utilizado a rede LoRaWAN, apenas LoRa, e eles próprios implementaram a infraestrutura, com um único *gateway*. Uma maneira que seria interessante de tentar enviar o *payload* novamente com os dados da última geolocalização, caso não fosse entregue no servidor de rede LoRaWAN, seria atribuir `true` para a variável `isTxConfirmed`, essa que é responsável no software do dispositivo, por receber um sinal do servidor de rede informando que o *payload* foi entregue com sucesso. Porém, esse comportamento gasta sempre uma mensagem de `downlink` quando a mensagem é confirmada e, no maior plano atual hoje oferecido pela KORE, o limite do `downlink` é 8 por dia, sendo inviável atualmente receber essa confirmação.

Em relação a aplicação web desenvolvida, a mesma atendeu os requisitos básicos para se realizar o gerenciamento e o rastreamentos dos pedidos. Comparando com o trabalho de Ferreira e Prazeres (2017), um dos diferenciais da aplicação web desenvolvida é no momento que o usuário cadastra os seus endereços, armazenando também a geolocalização do mesmo, facilitando assim, a visualização no mapa o destino do pedido que o usuário fez, utilizando o endereço cadastrado. Diferentemente do trabalho de Ferreira e Prazeres (2017), onde que o motorista do veículo precisa estar no local para marcar a posição atual. Porém, o trabalho deles possui também um aplicativo móvel que o motorista utiliza, que inclusive tem a opção de baixar a ficha de romaneio no aplicativo e abrir os destinos das entregas em aplicativos terceiros, como Waze e Google Maps, fazendo que o motorista sempre percorra a melhor rota possível para chegar ao seu destino.

Além das sugestões do usuário que testou a aplicação web, como de exibir somente os motoristas e veículos disponíveis no período selecionado e permitir a criação de rotas inteligente, uma sugestão para trabalhos futuros seria, quando o dispositivo enviasse um *payload* contendo uma geolocalização que por exemplo, está no raio de 1 km de distância de um dos pedidos, o status desse pedido fosse já atualizado automaticamente no banco de dados para `DELIVERED`. Seguindo essa linha, até mesmo quando o veículo retornasse ao ponto de partida, mudar o status da carga para `WAITING_FINISH`, esse que ficaria com o status em amarelo na listagem de carga, informando que a carga já retornou e falta o `admin` finalizá-la.

Uma outra sugestão para trabalhos futuros, seria enviar no *payload* até 8 geolocalizações coletadas pelo dispositivo. Hoje, o dispositivo envia a geolocalização obtida num *payload* de 8 bytes e fica no modo `sleep` por 10 minutos e quando "acorda" o ciclo se reinicia. Porém, na biblioteca da Heltec, existe a variável `appDataSize`, essa que armazena o tamanho do *payload* em bytes, podendo ser de até 64 bytes. Sendo assim, poderia ser feito da seguinte maneira, ao enviar as 8 geolocalizações, o dispositivo não entra em `sleep` por 10 minutos, mas sim por volta de 80 segundos, coleta a geolocalização, armazena no *payload* e entra em `sleep` novamente por 80 segundos, fazendo isso por 8 vezes. Após isso, o *payload* será montado com 8 geolocalizações, possuindo o tamanho de 64 bytes e assim entregando um histórico maior e mais detalhado da rota percorrida pelo veículo. Continuando no aspecto de implementações futuras, poderia ser testado e utilizado a plataforma TTN, conforme já mencionada neste trabalho, onde poderia ser comprado um ou mais *gateways* e verificar o comportamento dela na região, além de estar contribuindo para a infraestrutura pública da rede LoRaWAN. Também poderia ser aproveitado os outros módulos que o dispositivo oferece, como de Wi-Fi, Bluetooth e até mesmo a rede Global System for Mobile (GSM), caso na localidade que o veículo se encontra não possuir disponibilidade LoRaWAN.

REFERÊNCIAS

ALMEIDA, Orlan. **EasyIoT**: Tecnologia LoRa: O que é, distância e teste prático. [S.I.], 2019. Disponível em: <https://easyiot.com.br/blog/tecnologia-lora/>. Acesso em: 11 abr. 2020.

AUGUSTIN, Aloÿs et al. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things. **Sensors**, Palaiseau, v. 16, n. 9, p.01-18, set. 2016. Disponível em: <https://www.mdpi.com/1424-8220/16/9/1466/htm>. Acesso em: 18 mai. 2020.

BALLOU, Ronald H. **Gerenciamento da cadeia de suprimentos/logística empresarial**. 5. ed. Tradução Raul Rubenich. Revisão Rogério Bañolas. Porto Alegre: Artmed Editora, 2006.

BARREIROS, Nahan. **Análise e Construção de um Geolocalizador via Wi-fi BSSID com transmissão LoRa**. 2019. 101 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Universidade Estadual de Londrina, Londrina. Disponível em: <http://www.uel.br/ctu/deel/TCC/TCC2019_NahanBarreiros.pdf>. Acesso em: 17 mai. 2020.

BOR, Martin; VIDLER, John; ROEDIG, Utz. LoRa for the Internet of Things. In: EWSN '16 PROCEEDINGS OF THE 2016 INTERNATIONAL CONFERENCE ON EMBEDDED WIRELESS SYSTEMS AND NETWORKS, 16., 2016, Graz. **Anais eletrônicos...** Graz: Junction Publishing, 2016. p. 361-366. Disponível em: <https://dl.acm.org/doi/10.5555/2893711.2893802>. Acesso em: 26 mai. 2020.

COMMITTEE, LoRa A. T. **LoRaWAN™ 1.0.3 Specification**. Beaverton, 2018. Disponível em: <https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf>. Acesso em: 23 mai. 2020.

CONTEFLEX. **Monitoramento de cargas: por que é importante e como fazer?**. [Novo Hamburgo], 2017. Disponível em: <http://blog.conteflex.com.br/monitoramento-de-cargas-por-que-e-importante-e-como-fazer/>. Acesso em: 11 abr. 2020.

FERREIRA, Alberto S.; PRAZERES, Pablo. **Sistema móvel para acompanhamento de rotas e entregas de uma empresa de transporte**. 2017. 135 f. Trabalho de Conclusão de Curso (Bacharelado em Sistemas da Informação) – Universidade do Sul de Santa Catarina, Palhoça. Disponível em: <https://riuni.unisul.br/bitstream/handle/12345/2154/TCC%20-%20final.pdf>. Acesso em 17 mar. 2020.

LORA ALLIANCE, **A technical overview of LoRa® and LoRaWAN™.** [S.I.], 2015. Disponível em: <https://lora-alliance.org/sites/default/files/2018-04/what-is-lorawan.pdf>. Acesso em: 23 mai. 2020.

NTC&LOGÍSTICA. **NTC&Logística divulga números da Pesquisa de Roubo de Cargas em 2018**. [São Paulo], 2019. Disponível em: <https://www.portalntc.org.br/publicacoes/blog/noticias/outros/ntc-logistica-divulga-numeros-da-pesquisa-de-roubo-de-cargas-em-2018>. Acesso em: 12 abr. 2020.

RIBEIRO, Jonatas. **Uma aplicação da tecnologia LoRa em um ambiente hospitalar**. 2019. 79 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) – Departamento de Engenharia Elétrica e Eletrônica, Universidade Federal de Santa Catarina, Florianópolis, Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/197683/TCC_final_jonatas_ribeiro.pdf>. Acesso em: 16 mai. 2020.

SALAZAR-CABRERA, Ricardo; DE LA CRUZ, Álvaro P.; MOLINA, Juan M. M. Proof of Concept of an IoT-Based Public Vehicle Tracking System, Using LoRa (Long Range) and Intelligent Transportation System (ITS) Services. 2019. **Journal of Computer Networks and Communications**, v. 2019, p.01-10, dez. 2010. Disponível em: <http://downloads.hindawi.com/journals/jcnc/2019/9198157.pdf>. Acesso em: 19 mar. 2020.

APÊNDICE A – CÓDIGO FONTE DO DISPOSITIVO

Neste apêndice, é apresentado no Quadro 10, o código fonte do software que o dispositivo executa. Foram alterados os valores das variáveis `license` e `AppKey` a fim de integridade.

Quadro 10 – Código fonte do software do dispositivo

```
#include <ESP32_LoRaWAN.h>
#include "Arduino.h"
#include <HardwareSerial.h>
#include <TinyGPS++.h>
/*license for Heltec ESP32 LoRaWan, quary your ChipID relevant license: http://resource.heltec.cn/search */
uint32_t  license[4] = {0x00000000,0x00000000,0x00000000,0x00000000};
/* OTAA para*/
uint8_t DevEui[] = { 0xF8, 0x82, 0x77, 0xFF, 0xFF, 0x28, 0x6F, 0x24 };
uint8_t AppEui[] = { 0x77, 0xBF, 0xE4, 0x90, 0xBC, 0x43, 0x90, 0x45 };
uint8_t AppKey[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00 };
/* ABP para*/
uint8_t NwkSKey[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00 };
uint8_t AppSKey[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00 };
uint32_t DevAddr =  ( uint32_t )0x00000000;
/*LoraWan channelsmask, default channels 0-7*/
uint16_t userChannelsMask[6]={ 0x00FF,0x0000,0x0000,0x0000,0x0000,0x0000 };
/*LoraWan Class, Class A and Class C are supported*/
DeviceClass_t  loraWanClass = CLASS_A;
/*the application data transmission duty cycle.  value in [ms].*/
uint32_t appTxDutyCycle = 600000;
/*OTAA or ABP*/
bool overTheAirActivation = true;
/*ADR enable*/
bool loraWanAdr = true;
/* Indicates if the node is sending confirmed or unconfirmed messages */
bool isTxConfirmed = false;
/* Application port */
uint8_t appPort = 2;
/*Used for confirmed messages*/
uint8_t confirmedNbTrials = 8;
uint8_t debugLevel = LoRaWAN_DEBUG_LEVEL;
/*LoraWan region, select in arduino IDE tools*/
LoRaMacRegion_t loraWanRegion = ACTIVE_REGION;

TinyGPSPlus gps;
HardwareSerial ss(1);

const int RX_PIN = 23;
const int TX_PIN = 22;
const int BAUD_RATE = 9600;
int32_t lat = 0;
int32_t lon = 0;

static void prepareTxFrame( uint8_t port ){
    appDataSize = 8;//AppDataSize max value is 64
    appData[0] = (lat >> 24) & 0xFF;
    appData[1] = (lat >> 16) & 0xFF;
    appData[2] = (lat >> 8) & 0xFF;
    appData[3] = lat & 0xFF;
    appData[4] = (lon >> 24) & 0xFF;
    appData[5] = (lon >> 16) & 0xFF;
    appData[6] = (lon >> 8) & 0xFF;
    appData[7] = lon & 0xFF;
}

static void setLatitudeLongitude(){
  Display.clear();
  for (unsigned long start = millis(); millis() - start < 5000;) {
    while (ss.available()) {
      if (gps.encode(ss.read())) {
        Display.clear();
        if(gps.location.isValid()){
          lat = gps.location.lat() * 100000;
          lon = gps.location.lng() * 100000;
          Serial.print(gps.location.lat(), 6);
          Serial.print(F(","));
          Serial.println(gps.location.lng(), 6);
          Display.drawString(0, 30, "Lat:");
          Display.drawString(30, 30, String(gps.location.lat(),6));
          Display.drawString(0, 40, "Lon:");
          Display.drawString(30, 40, String(gps.location.lng(),6));
        }else{
          lat = 0;
          lon = 0;
          Display.drawString(0, 15, "-------- ATENTION --------");
          Display.drawString(0, 35, "Can't get lat/lon, please");
          Display.drawString(0, 45, "expose GPS antenna in air!");
        }
```

```
      }
    }
    Display.display();
  }
}

void setup(){
  Display.init();
  Display.clear();
  Display.setFont(ArialMT_Plain_10);
  if(mcuStarted==0){
    Display.drawString(0, 30, "STARTING");
    Display.display();
    delay(1000);
  }
  Serial.begin(115200);
  ss.begin(BAUD_RATE, SERIAL_8N1, RX_PIN, TX_PIN);
  while (!Serial);
  SPI.begin(SCK,MISO,MOSI,SS);
  Mcu.init(SS,RST_LoRa,DIO0,DIO1,license);
  deviceState = DEVICE_STATE_INIT;
}

void loop(){
  switch( deviceState ){
    case DEVICE_STATE_INIT:{
      Display.clear();
      LoRaWAN.init(loraWanClass,loraWanRegion);
      break;
    }
    case DEVICE_STATE_JOIN:{
      Display.clear();
      Display.drawString(0, 30, "JOINING...");
      Display.display();
      LoRaWAN.join();
      break;
    }
    case DEVICE_STATE_SEND:{
      setLatitudeLongitude();
      Display.clear();
      if(lat != 0 && lon != 0) {
        Display.drawString(0, 30, "SENDING...");
        prepareTxFrame( appPort );
        LoRaWAN.send(loraWanClass);
        appTxDutyCycle = 600000;
      }else{
        Display.drawString(0, 30, "TRYING AGAIN IN 60S");
        appTxDutyCycle = 60000;
      }
      Display.display();
      deviceState = DEVICE_STATE_CYCLE;
      break;
    }
    case DEVICE_STATE_CYCLE:{
      txDutyCycleTime = appTxDutyCycle; //+ randr( -APP_TX_DUTYCYCLE_RND, APP_TX_DUTYCYCLE_RND );
      LoRaWAN.cycle(txDutyCycleTime);
      deviceState = DEVICE_STATE_SLEEP;
      break;
    }
    case DEVICE_STATE_SLEEP:{
      Display.clear();
      Display.drawString(0, 30, "SLEEPING...");
      Display.display();
      LoRaWAN.sleep(loraWanClass,debugLevel);
      break;
    }
    default:{
      deviceState = DEVICE_STATE_INIT;
      break;
    }
  }
}
```

Fonte: adaptado de Heltec (2020).

APÊNDICE B – FICHA DE ROMANEIO

Neste apêndice é apresentado, na Figura 16, a ficha de romaneio que o motorista leva durante a entrega de uma carga. A ficha contém os dados necessários da carga e da entrega de cada um dos pedidos. Além disso, possui os códigos de barras que precisam ser escaneados para iniciar a entrega da carga. É disposto também, espaço para coletar as assinaturas dos clientes e do motorista que realizou a entrega.

Figura 16 – Ficha de romaneio

FICHA DE ROMANEIO: C000015

Motorista: Pedro Souza Celular: (47) 9 9939-9449 Status: CLOSED Observação: Aguardando a entrega.

Saída planejada: 12/11/2020 10:30:00

Retorno planejado: 12/11/2020 12:30:00

Barcode	Veículo
VVD6FB10D6	Marca: FIAT Modelo: Ducato Placa: EBA-7896 Dispositivo: ESP32 Heltec01 Referência: Ducato preta

Barcode	Endereço	Cliente	Status
RREFD5329C	Rua Hans Lorenz, 340 - Itoupava Seca Complement: 89030-450 | Blumenau - SC	Pedido: 1605185630569 Nome: Teste2 Teste2 Celular: null Assinatura: _____	DELIVERED:____ RETURNED:____ Observação:
RRB5D0E920	Rua Itajaí, 2077 - Vorstadt Complement: 89015-200 | Blumenau - SC	Pedido: 1605026631133 Nome: teste6 teste Celular: (47) 9 9948-5445 Assinatura: _____	DELIVERED:____ RETURNED:____ Observação:
RR5270263B	Rua Almirante Barroso, 250 - Itoupava Seca Complement: 89035-400 | Blumenau - SC	Pedido: 1605185323618 Nome: Teste Testeeee Celular: (47) 9 8895-5665 Assinatura: _____	DELIVERED:____ RETURNED:____ Observação:

Data: 12/11/2020 22:44:29 Assinatura do motorista:_____

Fonte: elaborado pelo autor.

APÊNDICE C – COMPONENTES UTILZADOS E SEUS PREÇOS

A Tabela 2 contém todos os componentes utilizados para o desenvolvimento do dispositivo e seus preços aproximados.

Tabela 2 – Custos de implementação

Item	Preço (R$)
WiFi LoRa 32 (V2)	135,90
GY-NEO6MV2	15,56
Protoboard 400 pinos	5,40
4 jumpers macho-macho	0,40
Carregador portátil	- (já possuía)
Cabo USB	- (já possuía)
KORE - Plano G	36,04
Total	**R$ 193,30**

Fonte: elaborado pelo autor.