Camila Lobianco, Enzo Neves e Paulo Sestini

Sensoriamento WiFi e rede LPWAN: uma arquitetura fim-a-fim

São Paulo, SP

2023

Camila Lobianco, Enzo Neves e Paulo Sestini

Sensoriamento WiFi e rede LPWAN: uma arquitetura fim-a-fim

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Profa. Drª Cíntia Borges Margi

São Paulo, SP

2023

Catalogação-na-publicação

CEP, CEP
Sensoriamento WiFi e rede LPWAN - Uma arquitetura fim-a-fim / C. CEP, C. Lobianco, E. Neves, P. Sestini -- São Paulo, 2023.
86 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.WiFi Sensing 2.LoRaWAN 3.IoT 4.Machine Learning I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Lobianco, Camila IV.Neves, Enzo V.Sestini, Paulo

Resumo

O monitoramento de estradas permite detecção de acidentes, melhoria da segurança no trânsito e melhor planejamento urbano. Entretanto, nem sempre é possível sua implementação, em vias remotas ou em locais com poucos recursos financeiros. Isto porque soluções como o monitoramento por vídeo requerem uma infraestrutura que não é viável no cenário previamente apresentado. Neste trabalho, é proposta uma aplicação de monitoramento de baixo custo e baixo consumo energético, baseada em Internet das Coisas (IoT), a partir da distribuição de nós sensores integrados por meio de uma rede de comunicação de longo alcance para que seja possível o recolhimento e análise dos dados do monitoramento de vias remotas, propondo-se assim uma solução completa. Como forma de sensoriamento, utilizou-se a medição das perturbações da Informação de Estado de Canal (CSI) do rádio WiFi IEEE 802.11 a partir de placas ESP32. Já para a comunicação, dado os requisitos de baixo consumo energético e longas distâncias das transmissões, utilizou-se a *Long Range Wide Area Network* (LoRaWAN). No que diz respeito ao processamento dos dados, avaliou-se os *trade-offs* entre computação de borda e processamento via *Cloud Computing*, concluindo-se que para o contexto deste projeto, foi mais interessante a utilização da última. Sendo assim, foi implementada uma aplicação completa, que envolveu a coleta dos dados de monitoramento de vias remotas por meio de nós sensores, construção da infraestrutura de rede de comunicação, centralização, análise e processamento de dados. Como resultados, demonstrou-se a viabilidade da solução, com um sistema de monitoramento capaz de discriminar entre automóveis e pessoas, além de alertar sobre potenciais perigos na via. Obteve-se métricas de acurácia, perda de pacotes e tempo de resposta satisfatórias, que serão profundamente discutidas ao final deste trabalho. É possível obter acurácias de classificação de até 98%, a depender do cenário de implantação do sistema e sua configuração.

Palavras-chave: LoRaWAN, LoRA, IoT, WiFi, Sensing, CSI, Machine Learning.

Abstract

Road monitoring allows for the detection of accidents, improvement of traffic safety, and better urban planning. However, its implementation is not always possible, especially in remote roads or areas with limited financial resources. This is because solutions like video monitoring require an infrastructure that is not feasible in the previously described scenario. In this work, we propose a low-cost and low-energy consumption monitoring application, based on the Internet of Things (IoT), through the distribution of integrated sensor nodes via a long-range communication network to enable the collection and analysis of data from remote road monitoring, thus proposing an end-to-end solution. For sensing, the measurement of the disturbances in the Channel State Information (CSI) of the IEEE 802.11 WiFi radio using ESP32 boards was used. For communication, given the requirements of low energy consumption and long transmission distances, the Long Range Wide Area Network (LoRaWAN) was used. Regarding data processing, the trade-offs between edge computing and processing via Cloud Computing were evaluated, concluding that for the context of this project, the latter was more suitable. Therefore, a complete application was implemented, involving the collection of remote road monitoring data through sensor nodes, construction of the communication network infrastructure, data centralization, data analysis, and data processing. As results, the feasibility of the solution was demonstrated, with a monitoring system capable of discriminating between automobiles and people, as well as alerting to potential dangers on the road. Satisfactory metrics of accuracy, packet loss, and response time were obtained, which will be discussed in depth at the end of this work. It is possible to achieve classification accuracies of up to 98%, depending on the system deployment scenario and its configuration.

Keywords: LoRaWAN, LoRA, IoT, WiFi, Sensing, CSI. Machine Learning.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ADR	Adaptive Data Rate
AES	Advanced Encryption Standard
CASAGRAS	Coordination and Support Action for Global RFID-related Activities and Standardisation
CMAC	Cipher-based Message Authentication Code
CSI	Channel State Information
DSSS	Direct Sequence Spread Spectrum
EAMMH-RP	Energy Aware Multiuser & Multihop Hierachial-based Routing Protocol
EECRP	Energy-Efficient Centroid-based Routing Protocol
IERC	IoT European Research Cluster
IoT	Internet of Things
IP	Internet Protocol
kbps	kilobits por segundo
LoRa™	Long Range
LoRaWAN™	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network
ms	milissegundos
WiFi	Wireless Fidelity
WSN	Wireless Sensor Network

Sumário

1 Introdução

Neste capítulo é elucidado o contexto do projeto, as motivações, justificativas para realizá-lo e os objetivos que pretende-se atingir.

1.1 Contexto

O trânsito é uma parte integral da dinâmica de uma cidade e, por extensão, do país. Tanto nas zonas urbanas quanto nas rodovias interurbanas, o fluxo constante de veículos e pessoas demanda uma atenção especial. O monitoramento dessas vias não é apenas uma questão de manutenção, mas também um meio estratégico para garantir a segurança, eficiência e a qualidade de vida daqueles que as utilizam diariamente (BLISS; RAFFO, 2013). Contudo, essa não é uma tarefa facilmente generalizável, tendo em vista que cada via, seja em uma metrópole agitada ou em uma área rural tranquila, tem suas peculiaridades e necessidades.

Emergências de trânsito frequentemente têm consequências imediatas e graves, com o tempo de resposta ao incidente sendo um fator crucial entre salvar vidas ou enfrentar uma tragédia. No Brasil, essa preocupação assume uma dimensão ainda mais crítica, uma vez que os acidentes de trânsito são um dos principais desafios de saúde pública (LIMA et al., 2019), impactando milhares de vidas e familiares anualmente. A complexidade e extensão das vias brasileiras, aliadas a variedade de veículos e condições de tráfego, intensificam a necessidade de um monitoramento eficaz. Nesse cenário, não se trata apenas de observar o fluxo, mas sim de possuir sistemas capazes de detectar emergências e acionar os socorros rapidamente. A agilidade e precisão nesse processo podem ser determinantes no desfecho de tais situações, sendo relevante o desenvolvimento e implementação de uma solução eficiente e adaptável à realidade nacional.

Vias remotas apresentam desafios únicos no cenário de monitoramento, em comparação com os centros urbanos. Enquanto as cidades geralmente possuem infraestrutura robusta e acesso confiável a internet, as regiões mais isoladas não contam com as mesmas facilidades. Nelas, a conectividade com a internet pode ser extremamente limitada ou até mesmo inexistente, o que dificulta o uso de métodos convencionais de monitoramento (infraestrutura para câmeras e radares, por exemplo). Além disso, a instalação e manutenção de dispositivos como câmeras, podem representar um grande desafio em termos de custo, por falta infraestrutura de rede elétrica e comunicação. Dada esta realidade, ressalta-se a importância de uma solução de monitoramento eficiente, de baixo custo e que opere independente de uma infraestrutura que normalmente os outros métodos de monitoramento dependem.

1.2 Motivação

A obtenção de um método de monitoramento de tráfego de baixo custo e baixo consumo energético (de modo a se ter autonomia utilizando-se baterias e painéis solares), levaria à ampliação da cobertura deste serviço, por meio de sua implantação em vias antes não atendidas pelos métodos convencionais, como radares e câmeras de monitoramento. O emprego de uma abordagem IoT (*Internet of Things*) para a construção da solução pode ser adequado, propiciando baixo custo e fácil implantação. Não há um consenso sobre a definição de IoT, portanto para o contexto deste trabalho é utilizado a definição exposta na seção 2.1.

Com o desenvolvimento das tecnologias de IoT, surgiram inúmeras possibilidades de inovação e otimização, especialmente no que se refere ao monitoramento de vias. Essas tecnologias, quando combinadas com dispositivos de baixo poder de processamento, têm o potencial de transformar o modo de como as vias são monitoradas. No entanto, apesar dos avanços tecnológicos, um desafio persiste: a integração *end-to-end* que abranja desde a coleta de dados até o processamento deles.

A literatura apresenta diversas soluções fragmentadas. Algumas focam na utilização de rádio WiFi para monitoramento (HERNANDEZ; BULUT, 2023) (MA; ZHOU; WANG, 2019) (JIANG et al., 2020) (ZHOU et al., 2018), enquanto outras concentram-se na implementação de redes IoT, como as *Low Power Wide Area Networks*, para a transmissão de dados medidos por sensores (agricultura de precisão, meteorologia, *smart cities* etc) (ADELANTADO et al., 2017) (QUETÉ et al., 2020) (VARSIER; SCHWOERER, 2017) (RADCLIFFE et al., 2017). Porém, uma solução que combine e integre essas abordagens ainda é uma lacuna no campo de estudo. Este hiato representa não apenas uma oportunidade de pesquisa, mas também uma demanda crescente na prática, onde a integração de tecnologias pode potencializar o monitoramento e gestão de vias remotas.

A vantagem da abordagem IoT é a disponibilidade de dispositivos computacionais que apresentam de forma integrada os módulos necessários para a implementação da solução, ao mesmo tempo que possuem baixo custo, baixo consumo energético e capacidade de comunicação de longa distância. Tais pontos são importantes para garantir a atratividade do sistema monitor para implantação em áreas remotas, onde existe o requisito de baixo orçamento e baixa dependência de infraestrutura local.

1.3 Objetivos

O objetivo deste trabalho é pesquisar, desenvolver e implantar uma solução completa para o monitoramento de tráfego em vias remotas. Esta solução busca combinar aspectos de infraestrutura física e digital de forma escalável. Sendo assim, adota-se o uso de

tecnologias IoT, de modo a satisfazer requisitos de baixo custo, baixo consumo de energia e comunicação a longa distância. Tal proposta surge como uma alternativa econômica às opções já existentes no mercado, cujos altos custos muitas vezes inviabilizam sua aplicação em regiões mais remotas.

Dentro do escopo técnico, este trabalho propõe o uso de análise das perturbações no CSI (*Channel State Information*) do rádio WiFi como meio de detecção de elementos em transito na via. Este método de detecção é implementado de modo a identificar e distinguir os diferentes tipos de elementos que possam trafegar por vias remotas, tais como carros, pessoas e animais. Para atingir tal capacidade discriminatória, é empregado um modelo de *Machine Learning*.

Com o intuito de ampliar a funcionalidade e aplicabilidade da solução, este trabalho também visa implementar uma *dashboard* interativa. Através dela, é possível visualizar não somente as detecções de objetos nas vias mas também receber notificações imediatas de situações que indiquem um potencial acidente na via remota, permitindo, assim, uma respostas rápida a possíveis emergências.

Na figura 1, observa-se uma visão de alto nível da solução proposta. Sensores dispostos nas estradas detectam a passagem de carros, animais e pessoas. Os dados coletados pelos sensores são transmitidos para um servidor central, que os encaminham para o processamento de classificação dos objetos detectados a partir do modelo de *Machine Learning*. Esses dados são então salvos em um banco de dados, que alimenta a *dashboard*, permitindo a análise das informações de monitoramento, incluindo os alertas de possíveis acidentes.

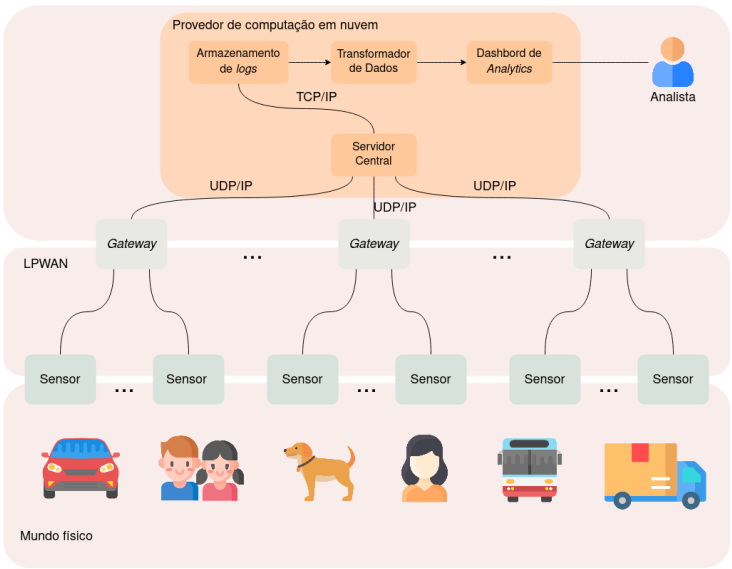

Figura 1 – Visão de alto nível da arquitetura da solução proposta. Ícones obtidos em Flaticon.

Há alguns aspectos importantes a serem considerados em relação aos sensores,

comunicação e apresentação das análises. Os sensores devem ser de baixo custo e baixo consumo de energia, de forma a haver incentivo para serem instalados em vias remotas. Da mesma forma, a tecnologia de comunicação deve ser de longo alcance e baixo consumo de energia, e portanto é buscado um protocolo pertencente à classe das LPWANs (*Low-Power Wide Area Networks*), que satisfaça estes requisitos. A apresentação das análises deve ser feita apenas para usuários autorizados e deve levar em conta os dados de todos os sensores, e portanto emprega-se o uso de computação em nuvem para centralizar os dados e realizar controle de acesso.

1.4 Arquitetura da solução

De maneira a elaborar a solução proposta, discutir e justificar as alternativas de tecnologias a serem empregadas, adota-se a divisão do projeto em três grandes blocos, sendo eles: sensores, comunicação e, processamento e visualização. Os três blocos, como apresentado na figura 2, constituem a aplicação ponto-a-ponta de monitoramento de tráfego, e serão abordados de maneira recorrente ao longo deste trabalho.

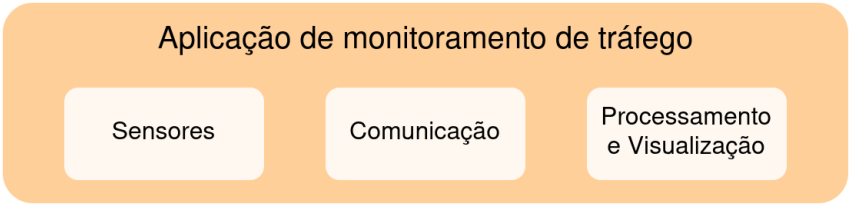

Figura 2 – Blocos que constituem a aplicação de monitoramento de tráfego.

Na figura 3 apresenta-se a arquitetura da solução IoT proposta. Sensores, baseados em detecção WiFi IEEE 802.11 (HERNANDEZ; BULUT, 2023), são dispostos ao longo das estradas, responsáveis por detectar a passagem de veículos, pessoas e animais. Os sensores enviam os dados via rede LoRaWAN para *gateways*, que encaminha via UDP/IP para o *Network Server*, responsável por centralizar os dados e gerenciar a rede LoRaWAN. Os dados centralizados são transmitidos (via TCP/IP) e processados na nuvem de forma a gerar uma *dashboard* de visualização de análise de dados.

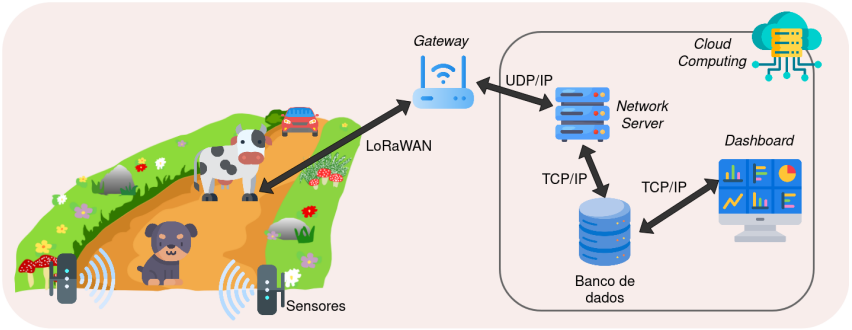

Figura 3 – Arquitetura da solução. Ícones obtidos em Flaticon.

2 Fundamentos

Neste capítulo aborda-se os conceitos chave para o trabalho, além de uma revisão dos principais trabalhos relacionados disponíveis na literatura. Na seção 2.1 discorre-se sobre diferentes visões do conceito de Internet das Coisas. Em 2.2 apresenta-se o monitoramento baseada em WiFi padrão IEEE 802.11, e métodos de inferência. Na seção 2.3 explica-se *LoRa* e *LoRaWAN* e a apresentação do *framework* ChirpStack.

2.1 IoT

Definição

Há uma diversidade de definições sobre o que é Internet das Coisas (IoT) que depende do contexto ao qual está sendo utilizada (alianças comerciais, pesquisadores, organizações de padronização etc), e as vezes em um mesmo contexto não há um consenso (LI; XU; ZHAO, 2015). A Comissão Europeia oferece uma definição semântica para "Internet das Coisas" sendo "uma rede global de objetos interconectados exclusivamente endereçáveis, baseado em um protocolo padrão de comunicação" (INFSO, 2008). Levando funcionalidade e identidade como aspectos principais, eles oferecem uma outra definição sendo "Coisas tendo identidades e personalidades virtuais operando espaços inteligentes utilizando interfaces inteligentes para se conectar e comunicar dentro de contextos sociais, ambientais e de usuário". Já a CASAGRAS propõe que uma rede IoT seja uma infraestrutura global conectando objetos virtuais e físicos, também incluindo os dispositivos previamente existentes já conectados à Internet (ATZORI; IERA; MORABITO, 2010) (VASSEUR; DUNKELS, 2008). Por último, a IERC define como "Uma infraestrutura de rede global dinâmica com recursos de autoconfiguração baseados em protocolos de comunicação padrão e interoperáveis, onde "coisas" físicas e virtuais têm identidades, atributos físicos e personalidades virtuais, usam interfaces inteligentes e são perfeitamente integradas à rede de informações" (VERMESAN; FRIESS, 2014).

Fica claro que, mesmo com os dispositivos IoT cada dia mais pervasivos em contextos pessoais, comerciais e industriais, não há uma definição que se encaixe perfeitamente em todos eles. Sendo assim, para o contexto deste trabalho é interessante uma definição própria que melhor se adéque ao que é proposto como solução. Portanto, IoT é uma infraestrutura de rede sem fio que conecta dispositivos de baixo consumo energético e com identificadores únicos, a um servidor centralizador que está conectado à *Internet*. A comunicação destes dispositivos é feita por interface e protocolos comuns à todas as "coisas" da rede.

Revisão da Literatura

O desenvolvimento de padrões técnicos têm se mostrado de extrema importância para o avanço na adoção de tecnologias IoT (LI; XU; ZHAO, 2015). Eficiência e requisitos especificados são considerados ao elaborá-los. Sendo assim, é necessário uma colaboração entre os órgãos globais e regionais de padronização para garantir a consistência entre eles. Especialmente, no que diz respeito a interfaces e *middleware*, há um esforço de pesquisa para que se desenvolva políticas e arquiteturas distribuídas e, garantia de privacidade e proteção dos usuários.

Uma infraestrutura robusta é indispensável para a adoção generalizada de aplicações de IoT. Para aplicações no contexto de vigilância e monitoramento, é importante que cada elemento seja devidamente identificado. Mesmo assim, a grande interconexão entre dispositivos é uma potencial ameaça de segurança (ROMAN; LOPEZ, 2009). Vale ressaltar que as técnicas de segurança já existentes para redes convencionais não se aplicam inteiramente para IoT. Sendo assim a "informação deve ser protegida desde o início de sua existência até o final do seu ciclo de vida" (LI; XU; ZHAO, 2015).

Uma rede de sensor sem fio (WSN) é uma rede na qual diversos dispositivos de sensoriamento espalhados por uma área, coletam dados, monitoram e se comunicam com uma unidade central conectada a internet. No contexto de IoT, uma das principais desvantagens deste tipo de rede é o suprimento energético limitado que frequentemente os sensores têm (SHEN et al., 2017). Dispositivos localizados em locais remotos e que utilizam baterias devem ter um gerenciamento energético excepcional para que seja possível manter a integridade da rede pelo maior tempo possível (PRIYANGA; VIMALRAJ; LYDIA, 2018).

2.2 Sensoriamento

O padrão WiFi IEEE 802.11 permite realizar detecção de objetos (HERNANDEZ; BULUT, 2023), utilizando como base informações de monitoramento de envio e recepção de dados entre dispositivos que se comunicam utilizando o protocolo. Objetos que transitem entre os dispositivos causam perturbações eletromagnéticas no sinal, que são detectáveis pelo dispositivo receptor.

A técnica de modulação OFDM (*Orthogonal Frequency Division Multiplexing*) é utilizada pelo padrão IEEE 802.11, na qual múltiplas ondas subportadoras ortogonais são utilizadas para transmissão de dados. Para avaliar perturbações de amplitude e fase em cada frequência de subportadora, o padrão utiliza o CSI (*Channel State Information*), que é uma matriz complexa representando a amplitude e fase do sinal de cada subportadora.

Para estimar-se o CSI em um dispositivo receptor, um outro dispositivo transmissor

envia um conjunto de símbolos conhecidos por ambos, denotado pelo vetor x, de modo que o receptor receba o vetor de símbolos y. Assim, a matriz CSI, denotada por $\mathbb{H}$, é estimada como:

$$y = \mathbb{H}x + \eta \tag{2.1}$$

Onde η é o ruído incorporado durante a transmissão, normalmente modelado como um ruído gaussiano.

A matriz $\mathbb{H}$ é constituída pelos valores de CFR (*Channel Frequency Response*) de cada subportadora, que representa sua amplitude e fase. Os valores de CFR possuem a forma $h_i = A_i e^{j\phi_i}$, sendo i o índice da subportadora, A_i sua amplitude, e ϕ_i sua fase. Assim, para a quantidade total de subportadoras S, tem-se:

$$\mathbb{H} = \begin{bmatrix} h_1 & h_2 & \dots & h_S \end{bmatrix} \tag{2.2}$$

Caso múltiplas estimativas de CSI sejam coletadas ao longo de uma janela de tempo w anterior a um instante t, pode-se construir a matriz $\mathcal{H}[t]$:

$$\mathcal{H}[t] = \begin{bmatrix} h_1[t-w+1] & h_1[t-w+2] & ... & h_1[t] \\ h_2[t-w+1] & h_2[t-w+2] & ... & h_2[t] \\ \vdots & \vdots & \ddots & \vdots \\ h_S[t-w+1] & h_S[t-w+2] & ... & h_S[t] \end{bmatrix} \tag{2.3}$$

A matriz $\mathcal{H}[t]$ permite visualizar perturbações ao longo do tempo devido à passagem de objetos. Na figura 4 apresenta-se uma representação gráfica da amplitude dos valores da matriz $\mathcal{H}$ em um certo intervalo de tempo. Nota-se que a passagem de um carro e de um cachorro entre o transmissor e o receptor provoca a atenuação das amplitudes das subportadoras. Objetos diferentes causam perturbações de intensidade e durações distintas, o que possibilita a inferência de qual tipo de objeto transitou entre os dispositivos envolvidos na comunicação.

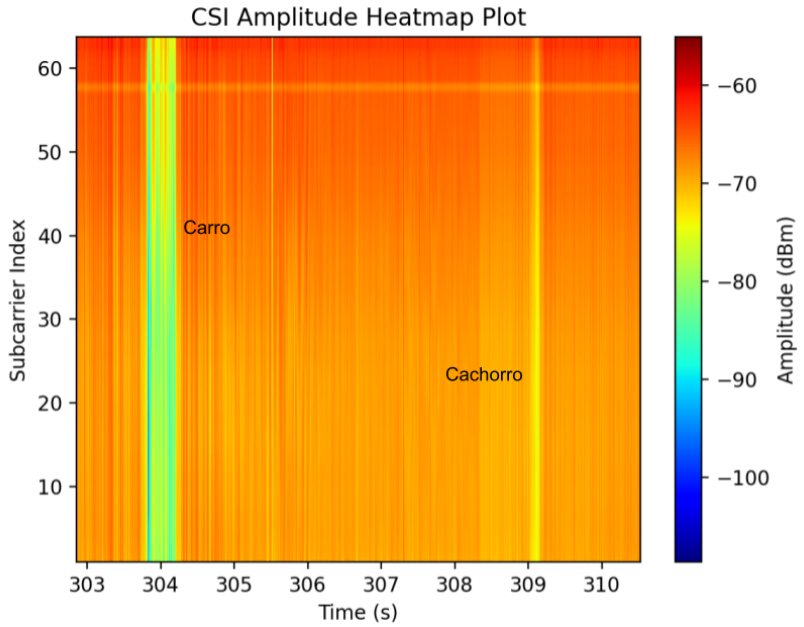

Figura 4 – Gráfico *heatmap* da amplitude de cada elemento da matriz $\mathcal{H}$ para uma janela de tempo. Nota-se perturbações causadas por um carro e por um cachorro transitando entre o transmissor e o receptor. Cortesia de Samuel Vieira Ducca.

Inteligência Artificial

Para a realização da inferência das entidades detectadas pelos sensores, é possível a utilização de inteligência artificial. Dentro deste tópico, existem alguns conceitos relevantes para este trabalho que serão abordados a seguir.

Coleta de dados

A coleta de dados é uma etapa importante já que ela é capaz de determinar o resultado positivo ou negativo do desempenho do modelo de aprendizado de máquina. Isso se dá porque dados de alta qualidade são capazes de levar à construção de um modelo mais confiável e preciso, além de determinar se o mesmo será capaz de reproduzir um *outcome* que esteja alinhado com a realidade do problema. Ressalta-se também que os dados devem ser fiéis ao contexto do problema, cobrindo todos os cenários atuais possíveis. Não obstante, dados não são a prova de falha, sendo necessário fazer um processo de limpeza e pré-processamento, de maneira a eliminar valores inconsistentes antes que eles sejam inseridos no modelo.

Pré-Processamento de dados

Para que os dados coletados sejam capazes de se adaptar ao contexto do modelo, é necessário adapta-los de maneira a simplificá-los ao máximo, para que o modelo seja capaz de abstrair informações relevantes de forma mais correta (SCIKIT-LEARN, 2023b).

Para isso, existem duas técnicas principais que são a normalização e a padronização. A normalização é o processo de reescalar os dados para um intervalo de [0, 1] ou de [-1,1] e tem como objetivo alterar os valores de características numéricas sem distorcer as diferenças nos intervalos de valores. Ela melhora a estabilidade numérica e a convergência do algoritmo, fazendo com que os dados fiquem mais consistentes. Já a padronização é o processo de fazer com que todos os dados tenham uma média de 0 e um desvio padrão de 1 e é útil quando eles têm diferentes variações ou unidades diferentes. Isso faz com que o modelo fique menos sensível a *outliers*, uma vez que não é afetada pelos valores de mínimos e máximos e também mantém as relações dos dados, o que pode ser útil para algoritmos que dependam disso.

Outro pré-processamento relevante é a seleção de características, mais conhecida por *feature selection* (SCIKIT-LEARN, 2023a). Ela identifica as variáveis que mais contribuem para a variável de saída e as escolhe. Ou seja, o objetivo é escolher apenas as características que melhorem o desempenho do modelo e reduzam dua complexidade e custo computacional. Esse processo é importante pois ele reduz a complexidade, uma vez que remove *features* redundantes ou irrelevantes, tornando o modelo mais simples e rápido de treinar. Além disso, ela melhora o desempenho, já que reduz o *overfitting* (SCIKIT-LEARN, 2023c) (condição em que um modelo de *machine learning* se ajusta de maneira excessiva aos dados de treino, capturando peculiaridades e ruídos ao invés de padrões generalizáveis, resultando em performance enganosamente alta nos dados de treino, mas pobre em dados novos ou não vistos, demandando estratégias cuidadosas de validação e regularização para assegurar sua aplicabilidade e precisão em cenários reais), pois menos características são menos possibilidades para o modelo aprender com os ruídos dos dados, e também em função da redução de custo computacional de ter que processar um número maior de *features*.

Divisão do conjunto de dados

Outra etapa relevante para o contexto deste trabalho é o procedimento de coleta de dados e a sua divisão em dois grupos distintos: o conjunto de treinamento e o conjunto de teste. O conjunto de treinamento tem a função crucial de educar o modelo, permitindo que ele adquira a capacidade de realizar predições ou classificações de maneira eficaz. Por outro lado, o conjunto de teste é utilizado para avaliar o desempenho do modelo, sendo essencial que este seja segregado para assegurar uma avaliação precisa e imparcial do modelo.

Posteriormente a esta divisão, realiza-se a identificação dos dados, categorizando-os em rótulos (ou *labels*) no caso de aprendizado supervisionado, que servirão como referenciais para o aprendizado do modelo. Os rótulos fornecem os resultados desejados durante a fase de treinamento, orientando o modelo na direção correta. Desta forma, o processo

executado pelo algoritmo de aprendizado de máquina pode ser concebido como um esforço para minimizar a discrepância entre suas previsões e os rótulos fornecidos, o que em última análise, espelha o processo de aprendizagem.

Extração de *features*

Durante a etapa de extração de *features* (HERNANDEZ; BULUT, 2023), as características mais significativas do problema são identificadas e selecionadas a partir de um conjunto de dados brutos. Esta seleção é necessária para diminuir a complexidade computacional e amplificar a eficácia do modelo, culminando em uma melhora acentuada no seu desempenho. Assim, transforma-se os dados brutos em um formato que se alinhe com as características e restrições do modelo empregado. Isso pode incluir tanto a geração de novas características a partir dos dados originais quanto a seleção cuidadosa de características preexistentes. O resultado final deste processo é um modelo que não só é mais preciso, mas também requer menos tempo para treinamento e é mais intuitivo para interpretação. Após a implementação dessas transformações, torna-se essencial conduzir uma análise detalhada para compreender os efeitos e as implicações dessas mudanças no modelo desenvolvido, garantindo assim que as alterações contribuam positivamente para os objetivos do projeto.

Construção do modelo

Inicialmente, é necessário se ter uma compreensão aprofundada do problema em questão para realizar uma escolha informada sobre qual modelo de *machine learning* melhor se adequá ao cenário em questão (HERNANDEZ; BULUT, 2023). Além disso, uma análise detalhada das características dos dados é fundamental. Esta análise deve levar em consideração diversos fatores, tais como a distribuição dos dados, bem como a presença ou ausência de variáveis categóricas e valores faltantes.

Neste estágio, o modelo é "alimentado" com o conjunto de treinamento, permitindo que ele ajuste seus hiperparâmetros com base nos padrões observados nos dados. Em seguida, na fase de validação, utiliza-se métricas para avaliar sua performance final. A conclusão se dá com o teste do modelo utilizando-se o conjunto de teste, que não foi previamente utilizado no processo de treinamento. Este passo é necessário para assegurar que o modelo não só captou os padrões nos dados de treinamento, mas também é capaz de generalizá-los eficientemente para novos conjuntos de dados, garantindo assim sua aplicabilidade e precisão em cenários do mundo real.

Avaliação de Modelo

A fase de avaliação em projetos de *machine learning* é necessária para assegurar a confiabilidade e eficácia do modelo desenvolvido. Utilizando-se um conjunto de dados

de teste, com dados que não foram utilizados no treinamento do modelo, avalia-se a sua capacidade de generalização e desempenho em cenários novos.

A comparação com modelos de referência e a avaliação de *overfitting* e *underfitting* são etapas necessárias para assegurar que o modelo está performando de maneira o mais próximo da ótima possível. A interpretabilidade e explicabilidade do modelo também são consideradas, especialmente em domínios que exigem transparência nas decisões do modelo. Por fim, a viabilidade para o ambiente produção é avaliada, assegurando que o modelo está apto a performar adequadamente em condições reais e estabelecendo planos para monitoramento contínuo e manutenção após a implantação.

Inferência

As perturbações causadas no CSI devem ser analisadas de forma a identificar qual tipo de entidade causou a interferência, como carros, pessoas e animais.

Para realizar a inferência (HERNANDEZ; BULUT, 2023), é necessário primeiramente extrair *features* do CSI, que são valores que caracterizam a perturbação, por meio de técnicas de processamento de sinais. Estas *features* podem ser simples, como amplitude (figura 4) e fase de cada subportadora, ou mais complexas como resultados de transformadas e aplicação de filtros.

Além da extração das *features*, outras etapas também são importantes, como remoção de ruído e redução de dimensionalidade. Naturalmente, ruídos no ambiente podem distorcer os sinais, e afetar o processo de inferência, e portanto técnicas de redução de ruído aumentam a acurácia das predições. O CSI é composto de 64 subportadoras, e algumas delas podem conter informação redundante, que podem ser removidas e facilitar o processamento dos sinais.

Após a extração das *features* e etapas adicionais, aplica-se um algoritmo de inferência. Uma abordagem possível é a aplicação de modelos de *Machine Learning*, como KNN (*K-Nearest Neighbors*), SVM (*Support Vector Machine*) e redes neurais artificiais. Dado um conjunto de dados anotados, que associe perturbações de CSI com a classe que causou a perturbação, é possível treinar um modelo capaz de associar informações de CSI captadas à uma classe correspondente. Dessa forma, dada uma nova detecção, é possível inferir qual entidade foi detectada.

Revisão da Literatura

A partir do trabalho de Hernandez e Bulut (2023) é possível constatar os diferentes contexto para os quais a utilização da Informação de CSI juntamente com técnicas de aprendizado de maquina podem ser utilizados para a detecção de objetos (pessoas, automóveis, nível de umidade do solo etc). A partir dos dados de CSI, é possível identificar

quais são os efeitos de perturbações do ambiente na amplitude e fase do sinal de rádio-frequência WiFi IEEE 802.11. Sendo assim, esta técnica tem se mostrado mais acurada do que técnicas anteriores como a que utiliza a métrica de Indicador de Potência de Sinal Recebido (RSSI).

Mesmo com uma literatura extensa sobre o uso de CSI neste contexto, há uma gama limitada de alternativas de ferramentas para a captura de dados da Informação de Estado do Canal. Opções comerciais como a placa de rede Intel 5300 com a ferramenta *Linux 802.11 CSI Tool* ou uma placa de rede *Atheros* com a ferramenta *Atheros CSI Tool* necessitam estarem conectadas a um computador *Desktop* ou *notebook*. Sendo assim, além dessas soluções ocuparem muito espaço e não serem de fácil transporte, também são de um alto custo monetário, o que inviabiliza sua implementação em um sistema de larga escala. Uma alternativa que remedia esses pontos é a placa embarcada ESP32, que permite o acesso aos dados de CSI diretamente a partir do módulo de WiFi integrado.

O ESP32 é uma solução completa na qual é possível realizar a coleta e processamento dos dados de CSI. Segundo os autores, mesmo que as técnicas de processamento tenham sido desenvolvidas para plataformas mais potentes, é possível utilizá-las no hardware em questão com uma performance adequada. Além disso, o ESP32 se mostrou capaz de de executar modelos de *machine learning*, o que possibilita o total processamento no próprio dispositivo (*edge-computing*).

Algumas das principais implementações de sensoriamento WiFi por meio de CSI são localização, reconhecimento de atividade humana, reconhecimento de gestos, contagem de pessoas em uma multidão, detecção de ocupação e monitoramento de saúde. Dentre essas, a que está mais alinhada com o contexto deste trabalho é a de localização. A maneira mais usual que esta implementação é realizada é, além dos dispositivos de rádio WiFi serem dispostos de maneira estática pelo ambiente, também é necessário um hardware de transmissão ou recepção no objeto a ser localizado. Contudo, trabalhos mais recentes (JIANG et al., 2020) (ZHOU et al., 2018) demonstram ser possível a localização apenas com os dispositivos rádio WiFi espalhados pelo ambiente. Isso é feito a partir do uso de dados CSI, que é usado para gerar uma impressão digital do local em questão e, sempre que algum objeto causa uma perturbação no CSI, esta impressão muda.

2.3 Comunicação

LoRa

Long Range (LoRa) é uma modulação de longo alcance por espalhamento de espectro implementada na camada física do modelo OSI, derivada da modulação *Chirp Spread Spectrum* (SEMTECH, 2015). Os dados são transmitidos em uma frequência

variável, com fatores de espalhamento ortogonais, o que garante uma não interferência entre as transmissões. Além disso, o fator de espalhamento (SF) controla a taxa de dados da transmissão, permitindo que se tenha um *trade-off* entre ela, o tempo de transmissão, o gasto energético e a distância máxima da transmissão. Essa relação será melhor explicada a seguir.

Para melhor entender comunicações por espalhamento de espectro, é necessário compreender o teorema de Shannon-Hartley, que define a máxima taxa na qual a informação pode ser transmitida por meio de um canal, em frequência específica e na presença de ruido. Sendo assim, temos:

$$C = B \times \log_2\left(1 + \frac{S}{N}\right) \tag{2.4}$$

Onde:

$$\begin{aligned} C &= \text{capacidade de canal (bit/s)} \\ B &= \text{largura de banda do canal (Hz)} \\ S &= \text{potência média do sinal recebido (Watts)} \\ N &= \text{potência média do ruido (Watts)} \\ \frac{S}{N} &= \text{razão sinal ruido (SNR)} \end{aligned}$$

Neste contexto, temos que S << N, então:

$$\frac{C}{B} \approx \frac{S}{N} \tag{2.5}$$

De 2.5, nota-se que em um canal com SNR fixo, com o aumento da largura de banda do sinal transmitido, é possível mitigar a degradação do sinal causada por ruido.

O método *Direct Sequence Spread Spectrum* (DSSS), aumenta a largura de banda de um sinal para compensar a degradação causada por ruído. Para isso, é utilizado uma sequência de espalhamento para alterar a fase da portadora do transmissor. Isso é feito a partir da multiplicação do sinal a ser transmitido pela sequência de espalhamento (sequência "chip"). A frequência desta sequência é muito maior que a do sinal a ser transmitido, garantindo assim um aumento na relação 2.5 e **espalhando** a largura de banda além da do sinal original.

Na Figura 5 apresenta-se um sinal de dados com período T_{Bit} e largura de banda R_b, a sequência de espalhamento com período T_{Chip} e largura de banda R_c, e o sinal de banda base, resultado da multiplicação dos dois anteriores, com período T_{Chip} e largura de banda R_c.

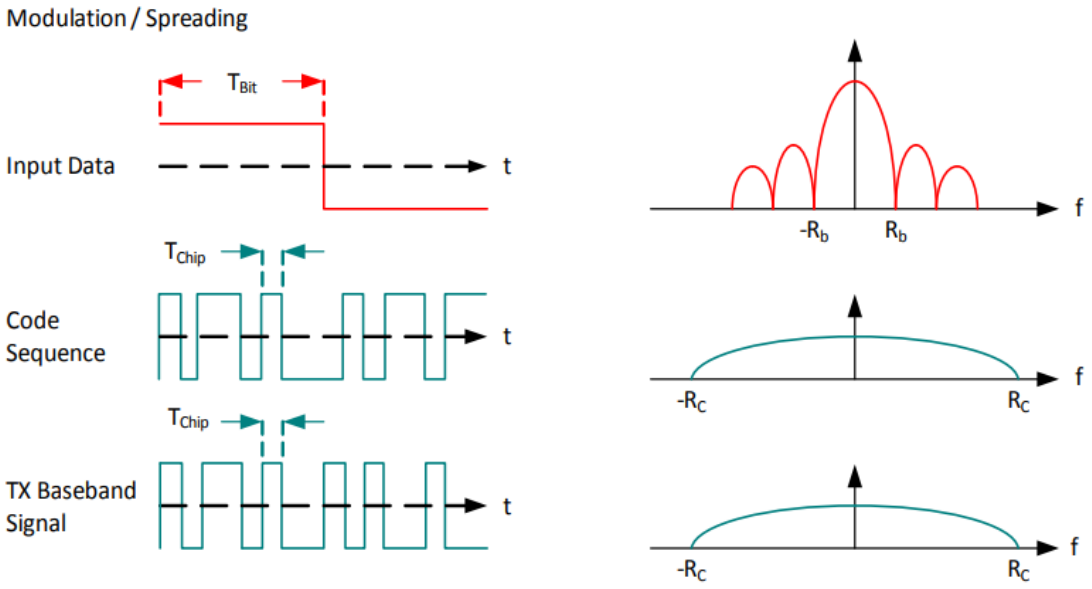

Figura 5 – Modulação DSSS. Obtido em LoRaWAN (SEMTECH, 2015)

No receptor, deve existir uma cópia localmente gerada da sequência de espalhamento. A demodulação do sinal é feita a partir da multiplicação do sinal recebido pela sequência de espalhamento local. Esse processo é demonstrado na Figura 6.

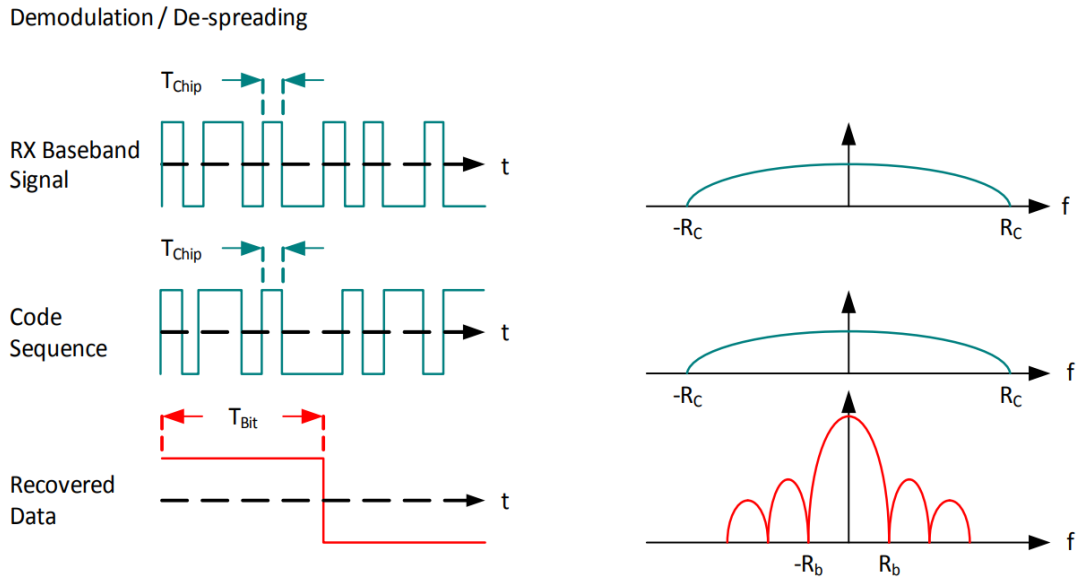

Figura 6 – Demodulação DSSS. Obtido em LoRaWAN (SEMTECH, 2015)

É possível definir o ganho de processamento(G_p) como:

$$G_p = 10 \times \log_{10}\left(\frac{R_c}{R_b}\right)(dB) \tag{2.6}$$

Onde:

$$R_c = \text{frequência de chips (chips/s)}$$
$$R_b = \text{frequência de bits (bits/s)}$$

Apesar de ser bastante utilizado, DSSS não é ideal para o contexto de aplicações *IoT*, tendo em vista que estas necessitam ter baixo consumo energético, como foi definido na seção 2.1. Isso porque a implementação do DSSS requer fontes de *clock* de referência altamente precisas. Além disso, quanto maior a sequência de espalhamento, maior o tempo de decodificação.

Como alternativa que remedia essas questões, tem-se a modulação por espalhamento de espectro LoRa baseada em *Chirp Spread Spectrum*(CSS). Para isso, ao invés de uma sequência de espalhamento, é utilizado um sinal "chirp" que varia continuamente e linearmente em frequência. Existem *up-chirp*, como apresentado na Figura 7, que a frequência aumenta com o tempo, e *down-chirp*, que a frequência cai com o tempo. Sendo assim, o deslocamento de tempo e frequência entre o transmissor e o receptor é facilmente compensado, já que o receptor pode ajustar o seu *chirp* local de demodulação, de modo a ser compatível com o do sinal recebido (CHEN; VENOSA; HARRIS, 2021). Isso reduz significativamente a complexidade do receptor, além de tornar o consumo energético do processo de modulação/demodulação inferior ao do DSSS.

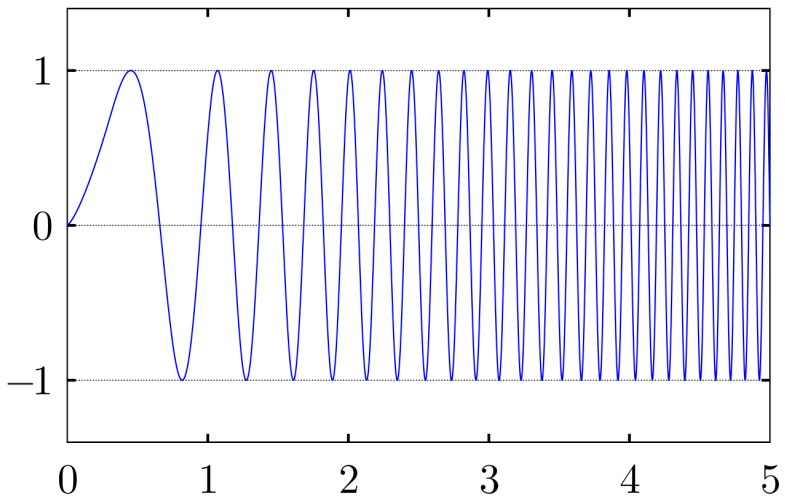

Figura 7 – Uma forma de onda linear *Chirp*. Uma onda senoidal que aumenta de frequência com o tempo. Obtido em Wikipedia.

É definido o fator de espalhamento (SF), um valor inteiro, geralmente entre 7 e 12, sendo quantos bits de informação são enviados em um símbolo CSS (CHEN; VENOSA;

HARRIS, 2021). Então a frequência de símbolos é:

$$R_s = \frac{1}{T_s} = \frac{BW}{2^{SF}} \tag{2.7}$$

E a frequência de chips, R_c, é:

$$R_c = R_s \times 2^{SF} \tag{2.8}$$

Onde:

$$\begin{aligned} R_s &= \text{Frequência de símbolos (símbolos/s)} \\ R_c &= \text{Frequência de chips (chips/s)} \\ T_s &= \text{Período de um símbolo (s)} \\ BW &= \text{largura de banda da modulação (Hz)} \\ SF &= \text{fator de espalhamento (7 a 12)} \end{aligned}$$

Sendo assim, quanto menor o fator de espalhamento, maior a frequência de símbolos, menor o tempo de transmissão e consequentemente e menor o gasto energético para a transmissão. Por outro lado, de 2.6 e 2.8, nota-se que quanto maior o SF, maior o ganho e consequentemente, maior a distância que a transmissão poderá ser recebida, porém também é maior o tempo necessário para a transmissão e maior o gasto energético.

LoRaWAN

Long Range Wide Area Network (*LoRaWAN*) é um protocolo de camada de enlace do modelo OSI, especificado pela *LoRa Alliance*, otimizado para dispositivos de baixo consumo energético. Uma rede *LoRaWAN* possui uma topologia de estrela de estrelas(*star-of-stars*). A segurança das transmissões na rede é feita a partir de criptografia simétrica, na qual se utiliza chaves de sessão para realizá-la.

Como apresentado na figura 8, a topologia é composta por 5 elementos distintos, possuindo pelo menos 1 de cada um deles: *end-device*, *gateway*, *Network Server*, *Application Server* e *Join Server*. A especificação trata o *Network Server*, *Application Server* e *Join Server* como um único servidor que desempenha a função dos três (ALLIANCE, 2017), portanto, no decorrer do presente texto este conjunto será referenciado como *LoRaWAN Network Server*.

Na figura 9 apresenta-se a topologia de uma rede LoRaWAN, estruturada em uma estrela de estrelas. Os *gateways* são estações que retransmitem mensagens provenientes dos *end-devices* para o *LoRaWAN Network Server*. A comunicação entre o *LoRaWAN Network Server* e o *gateway* é feita a partir de uma conexão IP segura.

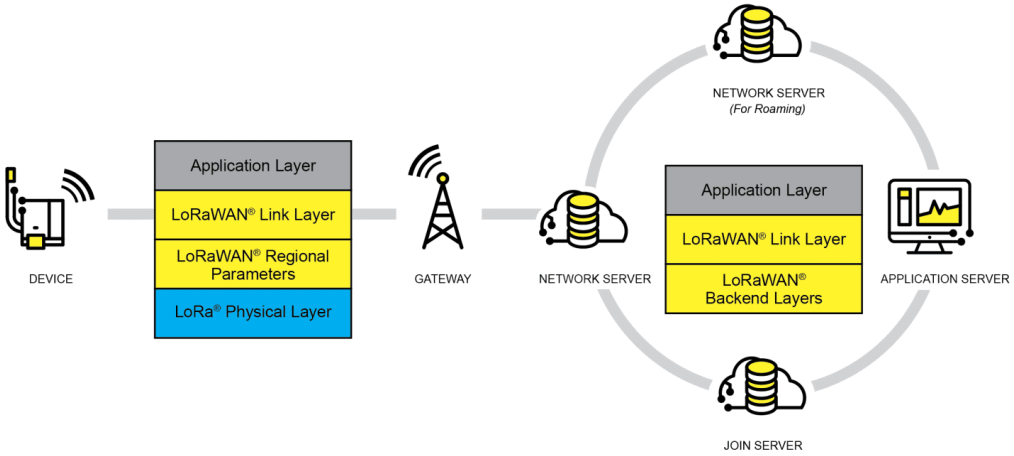

Figura 8 – Arquitetura LoRaWAN. Obtido em LoRaWAN Alliance (ALLIANCE, 2022)

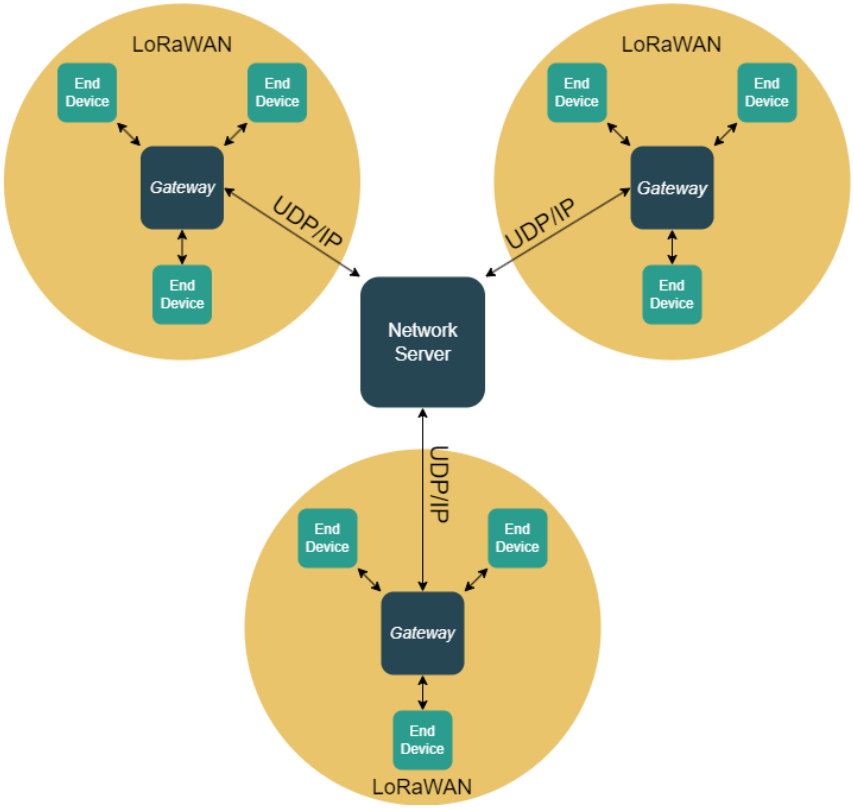

Figura 9 – Topologia estrela de estrelas na LoRaWAN.

Os *end-devices* são o ponto de entrada da rede. São dispositivos que recebem os dados e os transmitem para os *gateways*. É utilizado LoRa (2.3) para isso. Toda comunicação entre eles é bidirecional, porém, a maior parte do tráfego é *uplink* do *end-device* para o *gateway*. Um mesmo *end-device* pode se comunicar com vários gateways.

O *Network Server* faz o encaminhamento de mensagens para cada *Application Server* da aplicação destinatária. O *Application Server* gerencia as aplicações e encaminha

os dados para fora da rede LoRAWAN, podendo ser para um servidor local ou para a *Internet*. O *Join Server* realiza a autenticação dos *end-devices*. Para isso ele armazena as chaves raízes dos *end-devices* e as chaves derivadas utilizadas para a criptografia da comunicação. Após a autenticação, as comunicações entre os dispositivos são bidirecionais. Os *end-devices* e os *gateways* se comunicam utilizando vários canais de frequência e várias frequências de transmissão de dados. Desta maneira, é possível mitigar a interferência das diversas transmissões na rede. As taxas de transmissão ficam entre 0,3 kbps e 50 kbps. Existe um *trade-off* entre a taxa de transmissão de dados, a máxima distância de transmissão e o tempo de transmissão. Quanto maior a taxa, menor a distância máxima e menor o tempo de transmissão. Para transmitir em um dado canal de frequência, o *end-device* segue as seguintes regras:

- Para cada transmissão, o *end-device* seleciona um canal de maneira pseudo-aleatória.
- Respeita-se as limitações de *duty cycle* máximo de acordo com as legislações locais(melhor explicado a seguir).
- Respeita o tempo máximo de transmissão de acordo com as legislações locais.

A tecnologia LoRa realiza transmissão de dados por meio da técnica *Chirp Spread Spectrum* (CSS), e tem sua taxa de transferência determinada pelo *Spreading Factor*(SF), que geralmente varia de 7 a 12. Valores mais baixos de SF propiciam maior velocidade de transmissão, porém menor alcance. Da mesma forma, valores mais altos permitem maior alcance da rede, porém baixa velocidade. Para estender o alcance da rede, também é possível utilizar técnicas *multihop*(COTRIM; KLEINSCHMIDT, 2020), onde há mais de um *gateway* na conexão entre um *device* e o *LoRaWAN Network Server*. Entretanto, o uso de *multihop* pode introduzir atrasos na rede devido ao aumento de nós e distância.

Existem 3 classes de *end-devices*: bidirecional (classe A, deve sempre existir na rede), bidirecional com *slot* de recepção esquedulado (classe B, opcional) e bidirecional com slot de recebimento máximo (classe C, opcional) (ALLIANCE, 2017). Para o escopo deste trabalho, serão implementados *end-devices* classe A apenas. Eles permitem comunicação bidirecional, sendo que cada transmissão *uplink* do *end-device* é seguida por duas janelas curtas para o recebimento *downlink*. É usado um protocolo baseado em ALOHA para isso. Além disso, dispositivos classe A são os de menor consumo energético, já que as janelas para comunicação *downlink* são curtas.

Na Figura 10 é apresentada a pilha de tecnologia LoRaWAN. Seguindo o modelo OSI, na camada física tem-se exclusivamente o protocolo LoRa, e na camada de enlace é implementado o protocolo LoRaWAN. Destaca-se que na camada física, é definido a frequência de operação da rede em MHz.

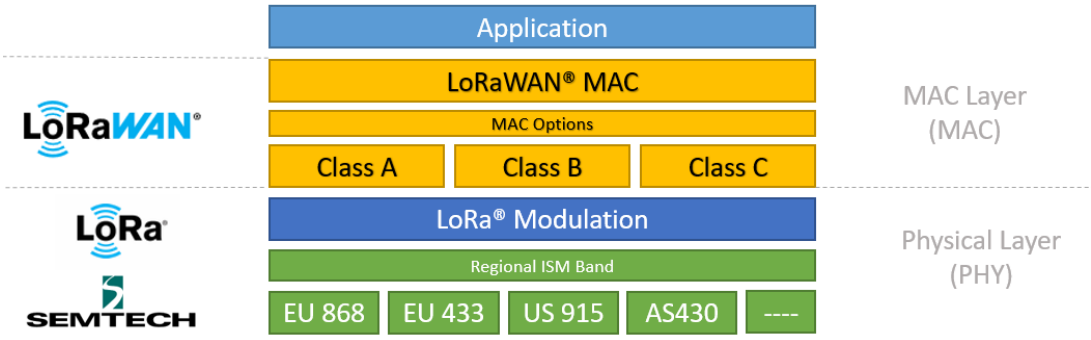

Figura 10 – Pilha de tecnologia LoRaWAN. Obtido em LoRa Allience.

Para se integrar em uma rede LoRaWAN, o *end-device* deve ser ativado na rede. Isso pode ser feito de duas maneiras diferentes: *Over-The-Air Activation*(OTAA) ou *Activation By Personalization*(ABP). Para OTAA, é necessário que o *end-device* tenha as seguintes informações definidas, para que seja possível realizar o processo de *join* na rede: DevEUI, JoinEUI, NwkKey e AppKey. Essas informações são utilizadas para gerar o DevAddr e 4 chaves de sessão (FNwkSIntKey, SNwkSIntKey, NwkSEncKey e AppSKey). Por meio deste procedimento, a conexão entre o *end-device* e o *LoRaWAN Network Server* possui contexto e, sempre que ele é perdido, é necessário que seja realizado o procedimento de *join* novamente. Já a ABP é muito mais simples, já que o DevAddr e as chaves de sessão são armazenados diretamente no *end-device*, não sendo necessário o procedimento de *join*. Para o contexto do projeto, este último é suficiente, tendo em vista que não será realizado nenhum tipo de federação ou conexão com a The Things Network(o que requereria o uso de OTAA).

A especificação LoRaWAN propõem a utilização de *duty-cycle*, em conformidade com a legislação local de cada região (Na UE é 1%, por exemplo)(SEMTECH, 2015). O *duty-cycle* é a porcentagem máxima de tempo no qual o *end-device* pode utilizar o canal. Então, por exemplo, caso uma região tenha *duty-cycle* de 20% e ocorra uma transmissão que tenha duração de 2 segundos, o *end-device* deve ficar 8 segundos sem transmitir. Porém, é possível implementar uma mudança de canal pseudo-aleatória de maneira a mitigar os impactos da limitação de *duty-cycle*. Para o contexto deste trabalho, não será considerado o nenhum *duty-cycle* e a transmissão poderá ser realizada a qualquer momento. Isso porque, além de o Brasil não ter um *duty-cycle* para redes LoRaWAN definido (Agência Nacional de Telecomunicações (ANATEL), 2017), e dado que o fluxo de objetos (carros, pessoas ou animais) que causam um envio de mensagem é baixo, além de ser necessário o envio imediato dos dados no caso de um potencial acidente, a frequência média de envio é extremamente baixa.

ChirpStack

O ChirpStack é uma implementação de *LoRaWAN Network Server* escrito na linguagem de programação *Rust*. Na figura 11 apresenta-se a arquitetura do ChirpStack. O bloco maior central inclui o ChirpStack, Chirpstack *Gateway Bridge* e MQTT *Broker*, e reside no *LoRaWAN Network Server*. Este bloco é responsável pelo gerenciamento da rede e centralização dos dados. Ao redor do bloco central, contam as conexões com *gateways* e integrações com sistemas externos.

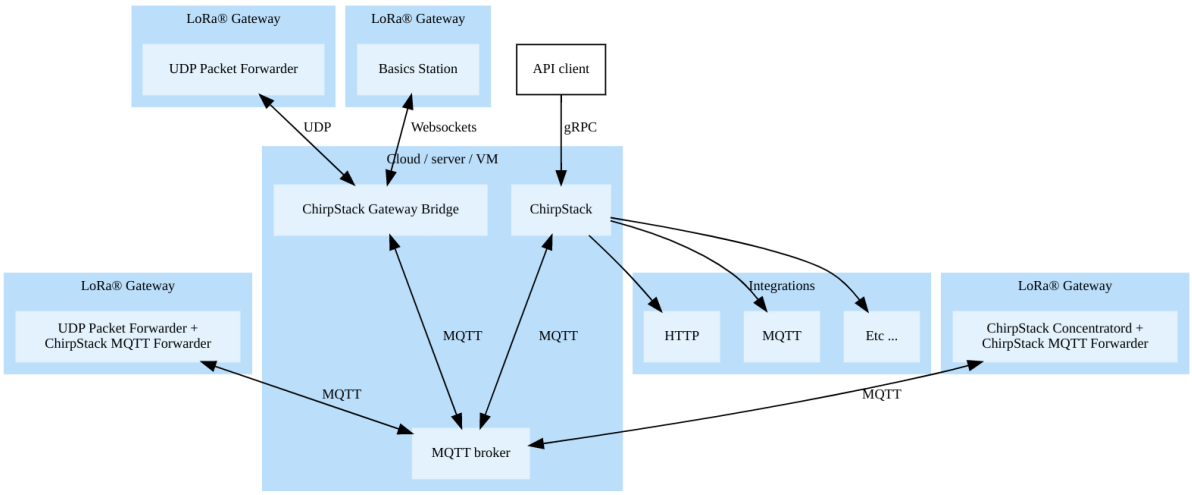

Figura 11 – Arquitetura do ChirpStack (CHIRPSTACK, 2023).

O *LoRaWAN Network Server* ChirpStack pode receber mensagens por meio de conexão UDP/IP, *broker* MQTT (publisher-subscriber) ou via *websockets*.

A forma de comunicação depende principalmente do *software* instalado nos *gateways*. Os *gateways* necessitam de um *packet forwarder* para encaminhamento de pacotes. Caso apenas um packet forwarder UDP esteja instalado no *gateway*, a comunicação UDP é utilizada. Caso um MQTT *forwarder* seja instalado junto com o Packet Forwarder UDP, é possível a comunicação MQTT. Um *packet forwarder* do tipo *Basics Station* permite conexão via *websockets*.

Revisão da Literatura

As aplicações possíveis para uma rede LoRaWAN foram profundamente analisadas por Haxhibeqiri et al. (2018). No contexto de IoT, são evidenciados os seguintes requisitos de projeto: baixo consumo energético, baixo custo e baixa complexidade na comunicação sem fio entre os dispositivos. Para cumprir tais requisitos é necessário a implementação de uma *Low Power Wide Area Network*(LPWAN).

Atualmente existem várias soluções de mercado de LPWANs como: SigFox, NB-IoT e LoRaWAN. Dentre essas, a que tem se mostrado mais difundida no meio acadêmico

é a LoRaWAN. Isso porque, além da facilidade de se integrar com plataformas de redes globais como a *The Things Network* e ter grande parte da sua implementação *Open-Source* (PAULTRE, 2015), é possível criar redes privadas diferentemente das outras alternativas.

Para dispositivos que estão a uma distância curta entre si (de alguns metros à centenas de metros) é possível a utilização de rede cabeada, *Bluetooth*, WiFi 802.11, *Zigbee* ou outras tecnologias de curto alcance. Já para dispositivos a uma distância da ordem de centenas de quilômetros e que o cabeamento não é possível, se utiliza rede de celular ou satélite. Finalmente, para o caso de redes que esteja entre essas duas classificações, o custo monetário e energético não justifica o uso de rede de celular, mas as tecnologias de menor alcance anteriormente citadas também não são suficientes. Sendo assim, avaliando o desempenho de uma rede *LoRaWAN* neste cenário intermediário, Wixted et al. (2016) demonstra que a tecnologia pode ser de fato implementada para um contexto de baixo consumo energético para aplicações de sensoriamento. Apesar de colinas e prédios terem um impacto considerável na perda de mensagens, com *gateways* suficientes, a cobertura de rede se mostrou satisfatória para locais remotos.

Adelantado et al. (2017) discute quais cenários são ideais para a implementação de uma rede *LoRaWAN* e quais não são. A tecnologia foi desenvolvida para lidar com aplicações nas quais se têm um número reduzido de mensagens em um dado período e que o atraso não é um parâmetro crítico. Nesse caso, tendo *gateways* suficientes para atender todos os *end-devices*, a rede irá operar da maneira esperada. Já para aplicações nas quais baixa latência e baixo *jitter* são requisitos, uma rede *LoRaWAN* não é a alternativa mais adequada. Por exemplo, para aplicações como automação industrial com necessidade de tempo de resposta de 1 ms à 100 ms, mesmo para pacotes pequenos de 10 Bytes, o tempo de transmissão para o SF7 é de 40ms. No contexto de aplicações de *smart cities* (estacionamento, iluminação pública, coleta de lixo etc) se tem mensagens periódicas e pouco sensíveis a atraso. Uma aplicação que se aproxima do que será proposto nesta monografia é controle de vaga de estacionamento inteligente. Neste caso se tem eventos apenas na detecção de mudança (alguém estacionou ou saiu da vaga), o que torna o fluxo de mensagem baixo. Sendo assim, esta é mais uma aplicação na qual uma rede *LoRaWAN* se mostra viável.

3 Método do trabalho

Neste capítulo, apresenta-se as metodologias e etapas de desenvolvimento do trabalho, sendo elas: apresentação e estudo do problema, desenvolvimento e implementação. Por fim, apresenta-se um cronograma das atividades a serem realizadas no decorrer do projeto.

3.1 Apresentação e estudo do problema

Em conjunto com a professora orientadora e pesquisadores do LARC-USP (Laboratório de Arquitetura e Redes de Computadores da USP), identificou-se o problema de monitoramento tráfego em vias remotas, onde falta incentivo e não há infraestrutura adequada para sua realização. Isto ocorre principalmente devido ao alto custo dos sistemas existentes, alto consumo energético e falta de cobertura de rede. Dessa forma, após definido o sistema, buscou-se uma arquitetura viável. De forma a prover baixo custo e baixo consumo de energia pelos sensores, optou-se pelo uso de nós sensores que monitoram o ambiente ao utilizar rádio WiFi IEEE 802.11 para sensoriamento. Devido à necessidade de instalação em vias remotas e os dados dos sensores precisarem ser coletados para análise, motivou-se a busca por uma rede capaz de comunicação a longas distâncias e com baixo consumo energético, o que culminou na definição da LoRaWAN como tecnologia de rede para o projeto.

Com o escopo bem definido, foram realizadas as definições de requisitos de sistema e de projeto. A partir de testes realizados com equipamentos de propriedade do LARC-USP, nos quais se buscou avaliar na prática alguns dos requisitos propostos. Finalmente, a partir disto, fechou-se a definição dos requisitos e a especificação dos dispositivos.

3.2 Atividades e cronograma

Uma vez que o projeto foi especificado e seus componentes definidos, lista-se as atividades necessárias para seu desenvolvimento, junto às suas respectivas etapas.

- **Criação de nó sensor**
  - Código para coleta de dados CSI, em C++
  - Código para tratamento de dados CSI e detecção de objetos, em C++
  - Código para transmissão de eventos via LoRaWAN, em C++
  - Integração com *gateway*

- **Criação de *gateway***
  - Instalação do sistema operacional
  - Instalação do software de rede LoRaWAN
  - Configuração de faixas de frequência
  - Integração com *LoRaWAN Network Server*
- **Criação de *LoRaWAN Network Server***
  - Instalação de software do *LoRaWAN Network Server* em instância virtual em um serviço de *cloud computing*, ou em computador *on premise*.
  - Configuração da aplicação no *LoRaWAN Network Server*, onde define-se os *gateways* e *end-devices* presentes na rede.
  - Integração do *LoRaWAN Network Server* com o banco de dados, para armazenamento de eventos.
- **Criação de modelo de identificação de eventos**
  - Coleta de dados para treino do modelo
  - Extração de *features* dos dados coletados
  - Treinamento e avaliação de modelos de *machine learning*, utilizando Python
  - Implantação do modelo na aplicação
- **Criação de banco de dados**
  - Criação do banco de dados em serviço de *cloud computing*
  - Criação das tabelas para armazenamento dos dados
  - Integração do banco de dados com o *LoRaWAN Network Server* e *dashboard*
- **Criação de *dashboard***
  - Desenvolvimento do código da *dashboard*, com o uso de *frameworks* Python
  - Construção de gráficos e métricas dos eventos, com opções de filtros e agregações
  - Integração da *dashboard* com o banco de dados
  - Implantação da *dashboard* em máquina virtual, em serviço de *cloud computing*.

Na tabela 1 é apresentado o cronograma do trabalho.

Fase	**Período**	**Etapa**
I	janeiro à abril	Estudo do problema e definição das especificações
II	maio à agosto	Desenvolvimento, implementação e testes
III	setembro à dezembro	Finalizando do desenvolvimento e testes, atualização e reporte dos resultados na monografia

Tabela 1 – Cronograma das etapas do trabalho

3.3 Plano de testes

Por se tratar de um projeto com vários componentes, é necessário testar as partes individualmente e depois sua integração. Lista-se as etapas do plano de testes.

- **Coleta de dados CSI via ESP32:** Coletar, via saída serial, medições CSI para avaliar o funcionamento das medições.
- **Transmissão de pacote LoRa:** Realizar envio e recebimento de pacotes LoRa via ESP32, para avaliar que a antena LoRa está funcionando. Para isto, serão utilizadas duas ESP32, uma atuando como transmissor e outra como receptor.
- **Coleta de dados CSI e transmissão LoRa:** Coletar os dados CSI, via WiFi, e realizar transmissão LoRa na mesma ESP32, para garantir que as antenas de ambas tecnologias funcionem corretamente quando há um processo utilizando a antena WiFi e outro a antena LoRa.
- **Coleta de dados CSI via ESP32 ao vivo:** Coletar e visualizar, ao vivo e em notebook externo, as medições CSI, para avaliar a influência de objetos na amplitude do sinal.
- **Conexão entre *gateway* e *LoRaWAN Network Server*:** Conectar *gateway* e *LoRaWAN Network Server* e atestar que a conexão de fato ocorre, por meio da transmissão de pacotes de *status*.
- **Fluxo completo do *end-device* até o *LoRaWAN Network Server*:** Realizar envios de pacote via LoRaWAN por meio da ESP32, e atestar que o *gateway* os recebe, repassa para o *LoRaWAN Network Server*, e que neste é possível visualizar os pacotes.

- **Capacidade de detecção em ambiente fechado:** Avaliar que o sensor é capaz de detectar objetos em ambiente fechado.
- **Capacidade de detecção na rua:** Avaliar que o sensor é capaz de detectar objetos na rua.
- **Conexão entre *LoRaWAN Network Server* e *dashboard*:** Avaliar que os eventos que chegam ao *LoRaWAN Network Server* são repassados para a dashboard
- **Fluxo completo de uma detecção na rua até a dashboard:** Avaliar que um evento de detecção, ocorrido em uma via, percorre a rede até a *dashboard*.

3.4 Experimentos

Além dos testes para avaliar o correto funcionamento e integração de cada bloco do projeto, experimentos serão realizados para atestar performance e alcance da aplicação.

- **Alcance entre *end-device* e *gateway*:** Aumentar a distância entre um *end-device* e o *gateway* continuamente, para avaliar perda de pacotes e alcance máximo.
- **Tempo de resposta total:** Avaliar quanto tempo demora entre a passagem de um objeto no sensor, e a visualização do evento na *dashboard*.
- **Acurácia da detecção:** Estimar a porcentagem de eventos que o sensor é capaz de detectar.
- **Acurácia de identificação:** Estimar a porcentagem de eventos que a aplicação é capaz de identificar corretamente, avaliando falso positivos e falso negativos para cada classe de evento.
- **Detecção de perigos na via:** Atestar a capacidade do sensor em detectar objetos parados na via, que indicam possíveis acidentes ou perigos, e fazer o envio à rede.

4 Especificação

Neste capítulo aborda-se os requisitos e o planejmaneto do projeto. Para os requisitos, levou-se em consideração as métricas necessárias para atingir as funcionalidades propostas pelo trabalho. Por fim, para o planejamento do projeto, foram discutidas as tecnologias que irão compor a solução final.

4.1 Requisitos

Para prover monitoramento de tráfego a baixo custo, baixo consumo energético e sem a dependência de infraestrutura complexa (como as de rede móvel ou de internet cabeada), é necessário detalhar os requisitos da aplicação e de seus 3 grandes blocos(figura 2).

A aplicação deve realizar medições de tráfego em tempo real, detectando veículos, pessoas e animais nas vias. A detecção deve ser feita de maneira a registrar dados de fluxo e identificar obstruções que possam indicar possíveis acidentes ou perigo. Os dados coletados devem ser tratados, por meio de técnicas de análise de dados, e os resultados apresentados em formato de *dashboard*.

Sensores

O nó sensor é composto por 2 placas ESP 32. Cada placa deve possuir um consumo energético que garanta autonomia para funcionar por meio de baterias comerciais e de pequeno porte. Além disso, a bateria deve ser recarregada de forma autônoma e sem necessidade de acesso a rede elétrica externa (e.g. por meio de painéis solares instalados junto aos sensores). Desta forma, os nós podem ser instalados em vias remotas de municipalidades com baixo orçamento para infraestrutura. Os nós também devem apresentar processamento suficiente para realizar o tratamento de dados CSI, realizar transmissão LoRa e a comunicação WiFi, possuindo um elemento WiFi *Access Point* e outro elemento WiFi *Receiver*.

O dispositivo de borda deve detectar eventos de passagem de carros, pessoas e animais, e encaminhar os dados do evento pela rede, para o devido tratamento. A resolução do sensor deve ser suficiente para detectar um evento a cada 30 segundos.

Comunicação

O *LoRaWAN Network Server* (figura 1) deve suportar conexão UDP para comunicação com os *gateways* e possuir capacidade de processamento suficiente para recepção dos dados, gerenciamento da rede e encaminhado dos dados para tratamento via integrações externas. Além disso, deve suportar também conexão TCP para comunicação com a rede externa.

Como requisito não funcional, inclui-se a qualidade de serviço da comunicação. Aqui, considera-se a vazão de dados, atraso e perda de pacotes. A seguir, analisa-se o requisito mínimo para que o sistema de monitoramento de tráfego possa funcionar de maneira adequada:

- **Vazão de dados**: a vazão na rede é uma função da quantidade de dispositivos de borda e a quantidade de eventos gerados em cada localização onde estão presentes. Dessa forma, a vazão entre um *gateway* e o *LoRaWAN Network Server* varia com a escala de implantação do sistema e o local onde é implantado.

  Para avaliar a vazão necessária, pode-se considerar um cenário fictício plausível. Caso cada gateway esteja conectado a 50 dispositivos de borda, onde cada dispositivo de borda transmita 100 eventos por hora, enviando pacotes com o tamanho de 242 bytes (limite máximo LoRaWAN), um único *gateway* necessitaria de uma vazão média de

  $$50 * 100 * 242 = 1210 \text{ kilobytes/hora} \approx 336,11 \text{ bytes/s}$$

  Caso existam 10 gateways no sistema, o servidor central precisaria de uma vazão total média de

  $$336,11 * 10 \approx 3,36 \text{ kilobytes/s}$$

  Sendo assim, para cenários similares ao considerado, uma vazão na ordem de kilobytes/s é considerada suficiente.

- **Tempo de resposta**: Há dois pontos importantes a se considerar em relação à latência, que é a chegada dos dados para análise, e a resposta a possíveis perigos na via. Uma latência da ordem de segundos garante precisão na análise dos dados e possibilidade de atender a emergências na via. Dessa forma, considera-se como meta um tempo de resposta de até 30 segundos.

- **Perda de pacotes**: Para os pacotes que trafegam na rede, é importante garantir no máximo uma perda de 1%, para que não haja deturpação das análises. Entretanto, a eventual perda de pacotes comuns não é custosa. Sendo assim, para estes, é transmitido apenas um pacote por evento. Porém, para potenciais perigos, a solução deve ser implementada de modo que seja enviado um pacote a cada segundo enquanto

houver um objeto entre os sensores. Desta maneira, é garantido que o evento seja enviado ao *dashboard*.

Outro requisito não funcional abordado é a segurança. Em relação à comunicação entre os dispositivos, o acesso à rede deve ser protegido de forma que apenas nós sensores e *gateways* autorizados tenham acesso, e somente estes sejam capazes de enviar dados de trânsito ao servidor central, de modo a evitar que agentes externos distorçam as análises de dados e influenciem na construção de políticas públicas e planejamento urbano decorrentes. Os dados também devem ser confidenciais e apresentados somente a profissionais autorizados. Por fim, deve-se garantir que os dados estejam disponível para serem processados e acessados via *dashboard*. Dessa forma, são necessários os serviços de disponibilidade, confidencialidade, integridade e autenticidade.

Processamento e visualização

Os dados medidos e derivados devem ser armazenados para posterior análise e consulta. Como forma de acesso, deve ser disponibilizado uma *dashboard* aos usuários da aplicação, como gestores públicos, planejadores urbanos, serviços de policiamento e bombeiros. A *dashboard* apresenta as informações em forma de gráficos e tabelas, além de permitir agregação de dados por vias e regiões. As análises apresentadas devem conter:

- Quantidade de carros, pessoas e animais que atravessaram cada via individual, em intervalo de tempo especificado pelo usuário.
- Notificações de obstrução e possíveis perigos nas vias.
- Possibilidade de agregação e agrupamento de dados por região e data.

Como requisito não funcional, inclui-se a usabilidade e responsividade da interface. Os gráficos e tabelas apresentados devem utilizar esquema de cores propícios para melhor visualização, e a interface deve ser de fácil uso pelo usuário e permitir sua interação com os dados e análises.

Deve ser possível a integração das notificações de acidentes a serviços externos, como sistemas públicos do governo, para possibilitar notificação direta a bombeiros e policiais.

4.2 Projeto

A seguir apresenta-se o planejamento do projeto para cada um dos 3 grandes blocos (figura 2) do sistema final.

Comunicação

Como forma de comunicação, adota-se uma arquitetura de rede baseada em topologia estrela de estrelas, que utiliza o protocolo LoRaWAN. A tecnologia LoRaWAN pertence à classe das LPWANs, e portanto é de baixo consumo energético e de longa distância(da ordem de dezenas de quilômetros em zonas rurais), o que permite a implantação em vias remotas e sem infraestrutura elétrica adequada para os sistemas convencionalmente utilizados para o monitoramento de tráfego.

A topologia estrela de estrelas centraliza as mensagens dos dispositivos IoT em um servidor central. Dessa forma, dados poderão ser reservados para análise.

Como exibido na figura 3, estão presentes como componentes da comunicação *gateways* e *LoRaWAN Network Server*. A função de cada um está melhor elucidado na subseção 2.3. Dessa forma, define-se os dispositivos correspondentes:

- ***Gateways***: Os *gateways* devem possuir suporte a módulos *LoRa* de alta capacidade e vários canais para suportar comunicação com vários *end-devices*. Também deve suportar comunicação UDP/TCP para conexão à internet e envio dos dados ao *LoRaWAN Network Server*. Para este fim, será utilizado o computador de placa única Raspberry Pi acoplado a um módulo *LoRa*.
- ***LoRaWAN Network Server***: O servidor central deve se conectar aos *gateways* via internet para centralização e análise dos dados. Poderá optar-se por um computador *on premise* ou máquina virtual em *Cloud Computing*.

Sensoriamento

Como sensor de objetos, propõe-se o monitoramento baseado em WiFi padrão IEEE 802.11 (HERNANDEZ; BULUT, 2023). Objetos que se deslocam próximos a um receptor WiFi perturbam o sinal eletromagnético no canal de comunicação, o que faz ser possível inferir o tipo de objeto, como carros, pessoas e animais. Dispositivos IoT que possuam módulos WiFi podem ser utilizado sem a necessidade de aquisição de módulos extras específicos para a detecção de objetos. A falta de necessidade de adquirir equipamentos mais caros, como câmeras, aliada ao preço reduzido dos dispositivos IoT, faz com que a abordagem de detecção via WiFi apresente a redução de custo necessária.

Dessa forma, os dispositivos de borda utilizados para o sensoriamento necessitam de WiFi IEEE 802.11, baixo consumo de energia, suporte a módulo de comunicação *LoRa* e capacidade de processamento suficiente para execução do algoritmo de *Machine Learning* para inferência. O microcontrolador ESP32 atende a estes requisitos e será o utilizado.

Processamento e Visualização

Os dados centralizados no *LoRaWAN Network Server* serão enviados para um provedor de computação em nuvem, como GCP (*Google Cloud Plataform*) ou AWS (*Amazon Web Services*), onde técnicas de análise e ciência de dados serão empregadas, de modo a gerar métricas que auxiliem profissionais de trânsito a guiar tomadas de decisões em relação ao tráfego nas estradas. A centralização dos dados também permitirá informar acidentes que ocorram nas vias. Como forma de apresentação aos profissionais, será desenvolvida uma interface em formato de *dashboard* para disponibilização das estatísticas e gráficos.

A *dashboard* será desenvolvida em linguagem Python, devido ao seu amplo ecossistema de bibliotecas *open source* de análise de dados, construção de *dashboards* e autenticação de usuários.

5 Desenvolvimento do Trabalho

5.1 Projeto e Implementação

5.1.1 Fluxo de dados completo da aplicação

Na figura 12, apresenta-se todos os blocos componentes da aplicação, o fluxo de dados entre eles, e o protocolo de comunicação utilizado.

Primeiramente, os dados de amplitude do CSI de um evento detectado são coletados pelo dispositivo de borda. Estes dados são transmitidos via *LoRaWAN* para o *Gateway*, responsável por encaminhar via UDP ao LoRaWAN Network Server. A partir do *LoRaWAN Network Server*, toda comunicação é feita via HTTP. Com o evento disponível, o modelo de identificação pode consumi-lo, associar uma classe à ele, e encaminhá-lo para armazenamento em banco de dados. Finalmente, a *dashboard* recupera e agrega os dados, exibindo gráficos e métricas ao usuário final da aplicação.

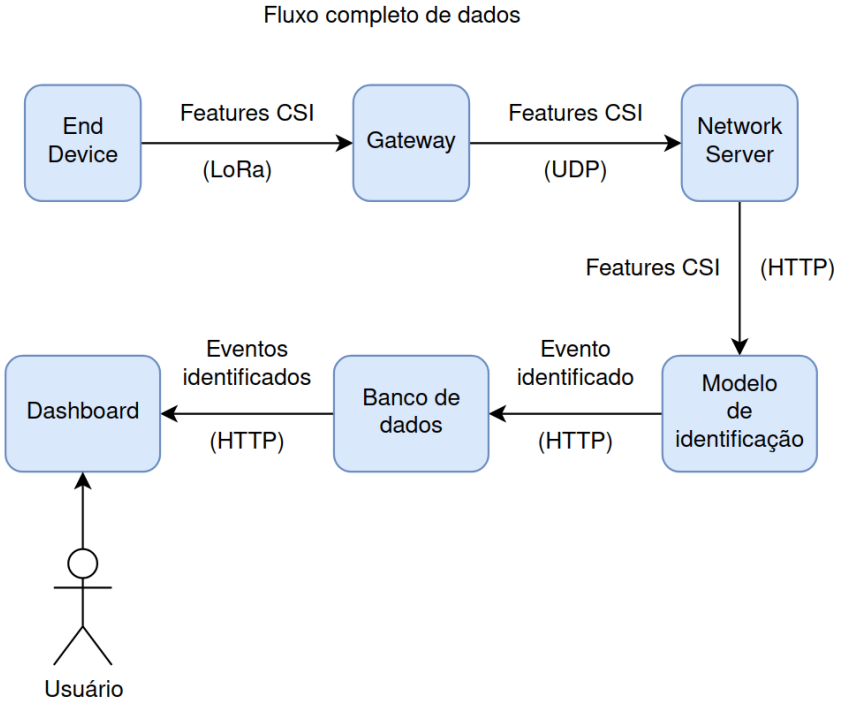

Figura 12 – Fluxo completo dos dados na aplicação, desde a coleta do CSI até a exibição do evento identificado na *dashboard*.

Na figura 13 apresenta-se a implementação concreta do fluxo descrito na figura 12. Como dispositivo de borda, utiliza-se um ESP32, e, como *Gateway*, um Raspberry Pi 3b juntamente com um HAT LoRa RHF0M301 . Para o *LoRaWAN Network Server*, utiliza-se o *ChirpStack* em ambiente Linux (Ubuntu) em uma instância EC2 (máquina virtual na

AWS). A *dashboard* é executada em ambiente similar, também em uma EC2. O modelo de identificação é implementado de maneira serverless a partir de uma AWS lambda, uma máquina virtual de duração efêmera, que é provisionada somente durante uso. Já o banco de dados utilizado é o MySQL, relacional, executando no serviço AWS RDS.

A decisão de hospedar ambos *LoRaWAN Network Server* e *dashboard* em máquinas virtuais se dá devido à necessidade de execução contínua. Ambos necessitam estar sempre disponíveis, para o gerenciamento da rede e acesso de usuários legítimos. Já a identificação de eventos só deve ser executada quando novos eventos forem identificados. Dessa forma, a partir do uso do serviço AWS lambda, é possível economizar poder computacional e memória, o que leva a uma maior eficiência e corte de custos.

A escolha do MySQL se dá devido à capacidade de realizar operações de agregação nos dados, o que facilita a implementação de visualizações na *dashboard*. O serviço AWS RDS foi escolhido pois já é gerenciado pela própria AWS, sem necessidade de configuração de máquinas virtuais, e possui fácil integração com demais serviços utilizados.

Um ponto importante é a integração entre o *LoRaWAN Network Server* e o modelo de identificação. O *ChirpStack*, dentre seus meios de integração externa, possui a capacidade de realizar uma operação de *POST* HTTP para uma API para todo novo evento recebido. Dessa forma a AWS lambda recebe as requisições HTTP de novos eventos, e providencia os recursos computacionais somente quando isto ocorre. Sendo assim, esta API para o recebimento dos eventos foi implementada a partir de um *AWS API Gateway*, responsável pelo gerenciamento da API e repasse dos dados para a Lambda de fato.

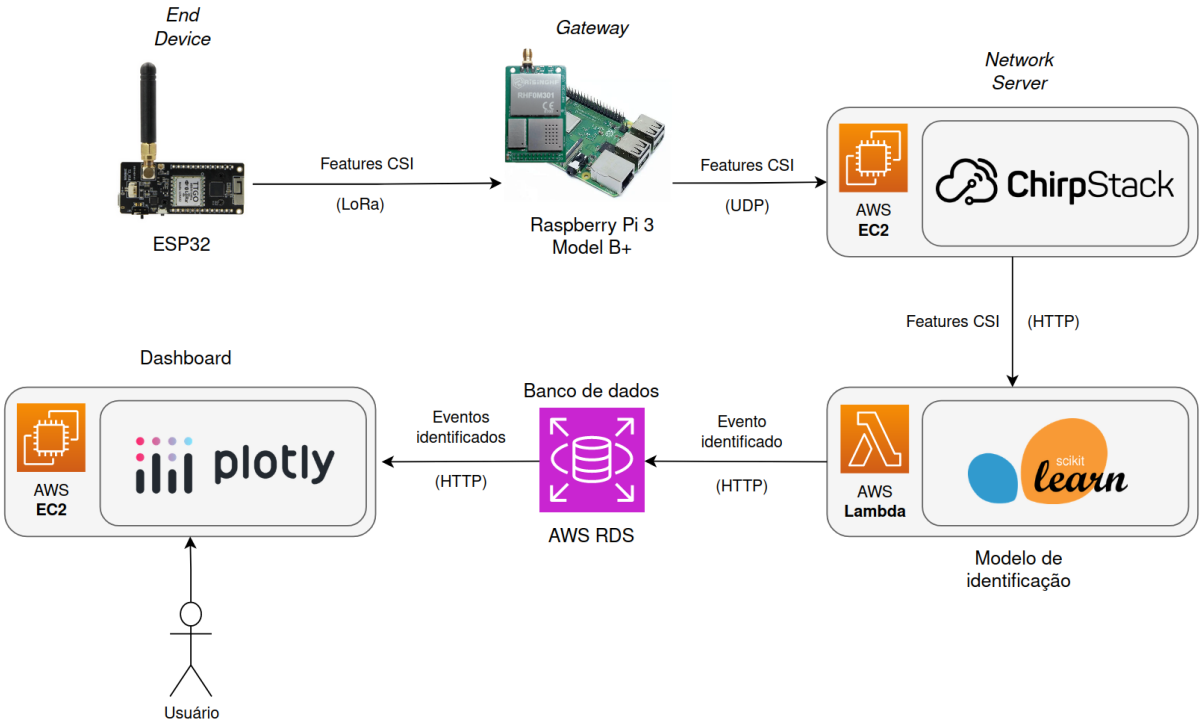

Figura 13 – Implementação concreta do fluxo da figura 12

5.1.2 Sensores

O sensoriamento foi implementado na ESP32, modelo LILYGO V2.1_1.6 (figura 14), dotada de módulo WiFi IEEE 802.11 e transmissor LoRa. Para o desenvolvimento do programa a ser executado na placa, foi utilizada a *toolchain* ESP-IDF, da Espressif, com código em C++. A ESP-IDF foi escolhida devido à disponibilidade das bibliotecas necessárias para o desenvolvimento do projeto, como a LMIC para comunicação LoRaWAN, Eigen para operações vetoriais e matriciais, e o projeto ESP32 CSI Toolkit, que proporciona a coleta e exportação de dados CSI.

Figura 14 – ESP32 utilizada, modelo Lilygo V2.1_1.6.

Como o sensoriamento é realizado a partir da análise das perturbações no CSI de um rádio WiFi, para cada nó sensor, necessita-se de 2 dispositivos, um para ser o *Access Point* WiFi e o outro o Receiver WiFi. No *Receiver* também será processado os dados CSI e o envio de pacotes LoRaWAN. Dado as tarefas: medição do CSI do WiFi, tratamento dos dados e envio destes via rede LoRaWAN; para cada um dos dois dispositivos que compõem o nó sensor, pode-se utilizar uma placa com a microcontroladora ESP32, que suporta funcionar como *Access Point* WiFi e possui módulo LoRaWAN SX1276. A RAM integrada de pelo menos 520 KB SRAM já é suficiente para a realização das tarefas anteriormente descritas. Um armazenamento interno de 4 MB, encontrado em placas comerciais, atende as necessidades do sistema, tendo em vista que a maior parte dos dados do fluxo são temporários.

No que diz respeito ao consumo energético por hora e usando como referência SEMTECH (2016) e Espressif Systems (2017), pode-se estimá-lo da seguinte maneira:

- módulo WiFi:

$$80mA * 1h = 80mAh$$

- módulo LoRaWAN:

$$(120mA * 10s)/3600 \approx 0,33mAh$$

Sendo assim, o consumo total por hora é de 80,33 mAh. Com uma bateria de 12.000 mAh para cada dispositivo do nó, o sistema tem uma autonomia de 6 dias, o que

está dentro das especificações(4). Para uma maior autonomia, é possível a utilização de painéis solares que recarreguem as baterias no período do dia e forneçam energização para o sistema. Desta maneira, diminui-se a frequência de manutenção referente a alimentação, tendo em vista que as baterias serão utilizadas apenas a noite e quando o tempo estiver nublado. Para o contexto deste projeto, os painéis solares não foram utilizados, já que isto foge do escopo proposto.

Escalonamento de tarefas

Todo o sensoriamento depende da execução de 3 tarefas distintas: aquisição de dados CSI do WiFi, tratamento desses dados e posterior envio via rede LoRaWAN. Como a placa ESP32 possui apenas 2 núcleos, foi necessário uma implementação para fazer o gerenciamento de recursos e garantir o devido funcionamento do sistema. Dado um carro de 4,5 m de comprimento a uma velocidade de 20 km/h, o automóvel consegue passar entre os sensores em menos de 1 segundo. Sendo assim, para garantir que nenhuma detecção será perdida por conta de chaveamento de processos, é interessante alocar um núcleo dedicado exclusivamente para o processo de aquisição de dados CSI do WiFi. Os outros dois processos, tratamento e envio dos dados via rede LoRaWAN, podem ser chaveados entre si, dado que suas execuções não dependem de nenhum evento instantâneo. Sendo assim, a aquisição dos dados ficou em um núcleo próprio e os demais processos compartilham o outro núcleo, chaveando entre si, como demonstrado na figura 15.

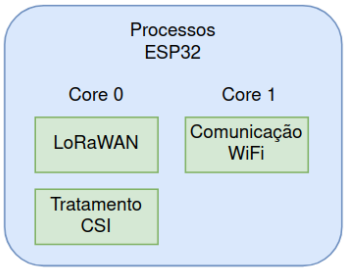

Figura 15 – Processos que executam em cada núcleo da ESP32.

Tratamento dos dados CSI

Assim que o sistema é iniciado, é necessário medir a amplitude CSI média do ambiente para se definir um valor de referência. Isso é feito colhendo os primeiros 500 vetores amplitude CSI (cada um contendo 52 valores representados em float32) medidos pelo sistema a uma frequência de 100 Hz. Sem que haja nenhum objeto entre as placas, e tomando a média dos vetores, obtém-se um único vetor de 52 elementos. Em seguida é tomada a média desses elementos, definindo-se assim o valor de referência, como demonstrado na figura 16.

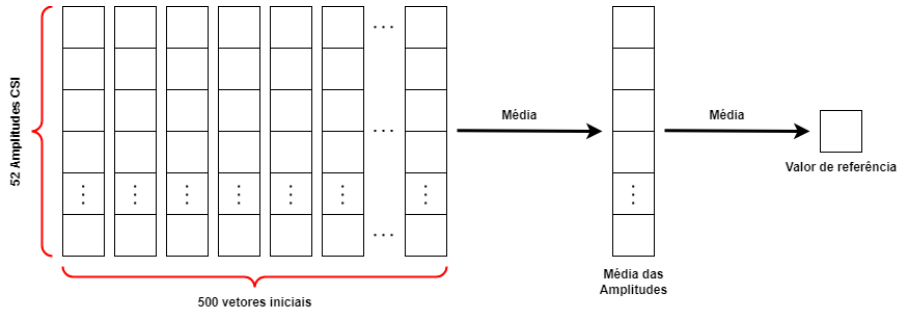

Figura 16 – Cálculo do valor de referência a partir dos 500 vetores de subportadoras inicialmente medidos pelo sistema.

Com um valor de referência estabelecido, o sistema segue funcionando com uma janela deslizante dos últimos 500 vetores amplitude CSI medidos, como demonstrado na figura 17. A cada nova medição, adiciona-se o vetor medido e remove-se o mais antigo da janela. Em todo momento tem-se uma matriz 500x52 de amplitudes CSI. Caso a média dos 150 vetores centrais da janela esteja abaixo do valor de referência, os dados da janela são processados como a detecção de um objeto. Após a detecção, calcula-se a média e o desvio padrão em linha dos 500 vetores da janela, obtendo-se 2 vetores com 52 elementos como resultado. Na figura 18 tem-se a representação deste fluxo completo.

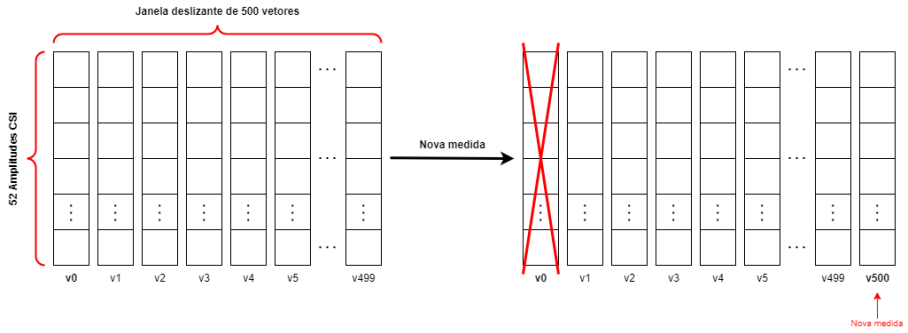

Figura 17 – Janela deslizante contendo sempre as últimas 500 medições das subportadoras do CSI.

Sendo assim, dada a frequência de medição de 100 Hz, um objeto é detectado após 2,5 segundos depois de ter passado entre os sensores. Portanto, levando em consideração o tempo necessário para que os dados sejam enviados e processados na nuvem, tem-se mais 1 segundo de *overhead*. Ao todo, então

$$2,5 + 1 = 3,5$$

Concluí-se que é seguro assumir um tempo de resposta inferior à 5 segundos, estando dentro das especificações do projeto.

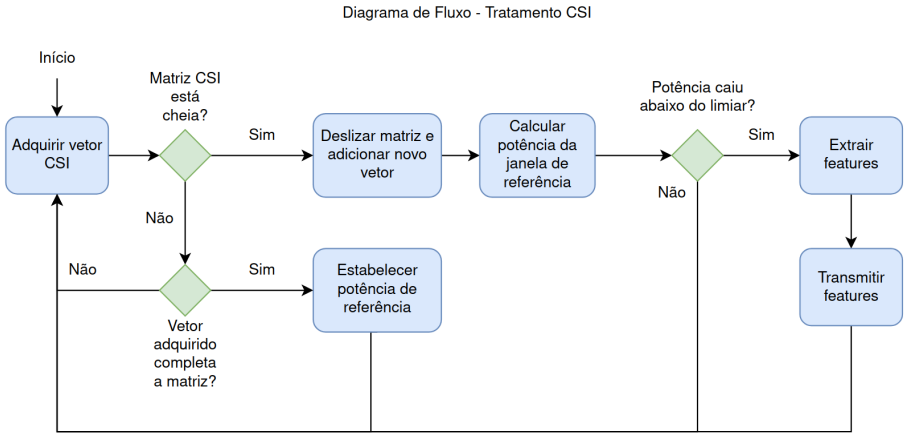

Figura 18 – Fluxo de tratamento dos dados CSI na ESP32.

Detecção de potencial perigo

Após ser estabelecido um valor de referência, inicia-se um *timer* com um limite de tempo configurável. É feita a medição do vetor amplitude CSI a uma frequência de 100hz. Sempre que uma medição é realizada e está acima do valor de referência, o *timer* é reiniciado. Sendo assim, se medições subsequentes estiverem abaixo do valor de referência, o *timer* seguirá a contagem normalmente, até finalmente atingir o tempo limite. Neste momento, ao invés de se enviar as *features* para o LoRaWAN Network Server, é enviado uma mensagem alertando sobre o potencial perigo. Este fluxo está melhor elucidado na figura 19.

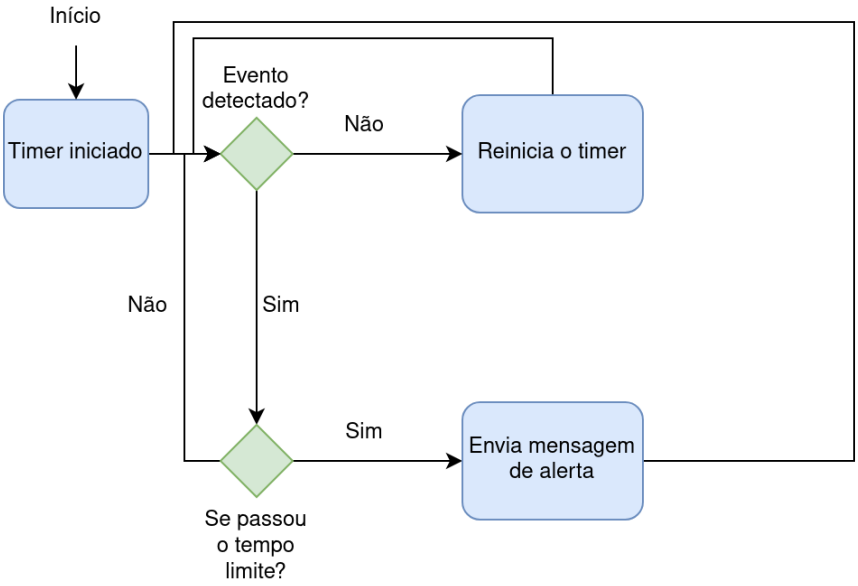

Figura 19 – Fluxo de dados do timer de detecção de potencial perigo.

Transmissão dos dados do end-device para o LoRaWAN Network Server

Como discutido em 2.3, a taxa de transmissão de dados via LoRa é limitada pelo SF. Um SF maior garante que o envio seja menos suscetível a interferências, porém o tempo necessário para transmissão e o consumo energético são maiores também. Dado os requisitos definidos em 4, para o contexto deste projeto é interessante que a transmissão, o processamento e a geração de resposta na *dashboard* seja menor que 5 segundos, além de sempre se priorizar pela longevidade dos sensores em operação que funcionam a bateria.

Utilizando-se o canal de 915 MHz, padrão US, com largura de banda 125 kHz e SF7(menor possível), tem-se um tamanho máximo de pacote de 242 bytes(The Things Network, 2023). A alteração do SF7 para SF10, mantendo-se as demais configurações, faz com que o payload máximo da mensagem LoRaWAN caia para 11 Bytes. Portanto, para o contexto do projeto, é mais interessante o uso do SF7, de forma a poder enviar mais dados. Eventualmente, caso seja necessário maior alcance da rede, pode-se realizar *tradeoffs* de performance para aumentar o SF.

Como será melhor elucidado em 5.1.4, para cada detecção de um objeto, tem-se 104 *features*, sendo a combinação dos vetores de desvio padrão e média apresentados em 5.1.2, cada uma de 32 bits. Sendo assim, ao todo seria necessário pelo menos 416 bytes por transmissão, sendo superior ao tamanho máximo de pacote que o protocolo LoRaWAN suporta. Porém, é possível reduzir o tamanho do pacote pela metade ao codificar-se os dados em *float16*. Além disso, é possível também selecionar apenas as *features* mais relevantes, o que diminui o tamanho do pacote. Ambas as técnicas para redução do pacote serão apresentadas na seção de construção do modelo de *Machine Learning*. Essa diminuição na quantidade de dados a serem enviados, é interessante pois a tarefa de envio de dados via LoRaWAN compartilha um núcleo do processador com a tarefa de tratamento dos dados de CSI (figura 15). Sendo assim, quanto menos dados forem enviados, menos a tarefa de envio bloqueará o núcleo compartilhado. Além disso, um pacote de menor tamanho também possibilita uso de valores maiores de SF, que aumentam o alcance da rede.

Como a linguagem C++ não realiza nativamente a conversão de float32 para float16, foi necessário uma implementação própria, baseando-se no trabalho de (LEHMANN et al., 2022). Para tanto, seguiu-se o padrão IEEE-754 para representação de float16. Além disso, como foi utilizada a biblioteca LMIC(KOOIJMAN, 2023) para realizar o envio dos dados via rede LoRaWAN, o payload a ser enviado é limitado ao formato string. Uma string é um vetor de chars. Cada char possui 8 bits, portanto não é possível enviar diretamente um vetor de float16 em que cada elemento possui 16 bits. Por conta disso, foi necessário a a separação de cada uma das N *features* em bits mais significativos e menos significativos, obtendo um vetor resultante com 2N elementos, cada um com 8 bits. A reconstrução dos dados é realizada no fluxo de processamento.

O processo de envio dos dados via rede LoRaWAN que executa na placa ESP32 fica em um estado de *idle* sempre que não está enviando nada. Como as detecções são usualmente muito espaçadas no tempo umas das outras e esse processo divide um núcleo de CPU com outro(5.1.2), o *Task Watchdog Timer*, nativo da *toolchain* ESP-IDF, interrompe o processo que bloqueia a CPU por mais de 3 segundos. Sendo assim, foi necessária a implementação de um semáforo binário para controlar o acesso à CPU compartilhada. Quando o semáforo é instanciado a partir da função *xSemaphoreCreateBinary(void)*, nativa do FreeRTOS que roda na ESP32, ele é criado em um estado "vazio" e para que algum processo tome o semáforo, este precisa ser liberado antes a partir da função *xSemaphoreGive(SemaphoreHandle_t xSemaphore)* (FreeRTOS, 2023). Sempre que o processo de identificação de evento e tratamento do CSI termina, ele libera o semáforo que é subsequentemente tomado pelo processo de envio via rede LoRaWAN. Esse fluxo é melhor demonstrado na figura 20.

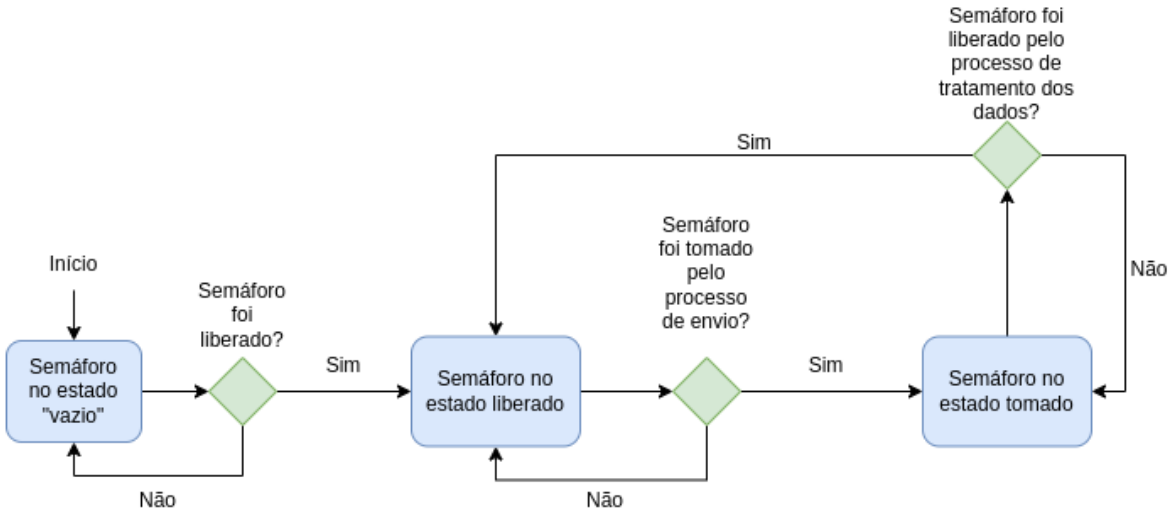

Figura 20 – Diagrama de fluxo do semáforo que gerencia o acesso a CPU compartilhada entre os 2 processos: tratamento e envio dos dados CSI.

Definição de mensagem LoRaWAN

Para o *payload* da mensagem LoRaWAN, tem-se duas possibilidades: envio das *features* extraídas dos vetores de amplitude CSI ou o envio de uma mensagem de alerta, referente a um potencial perigo na via. Como foi explicado em 5.1.2, foi necessária a conversão do vetor de *features* com N elementos, e 16 bits cada, para um vetor com 2N elementos, e 8 bits (2N bytes) cada para que pudesse ser enviado como payload da mensagem LoRaWAN. Na figura 21 está demonstrado melhor como os dados são estruturados. Já para o caso do potencial perigo, é enviado a string "POTENTIAL DANGER" como payload da mensagem, totalizando 16 bytes.

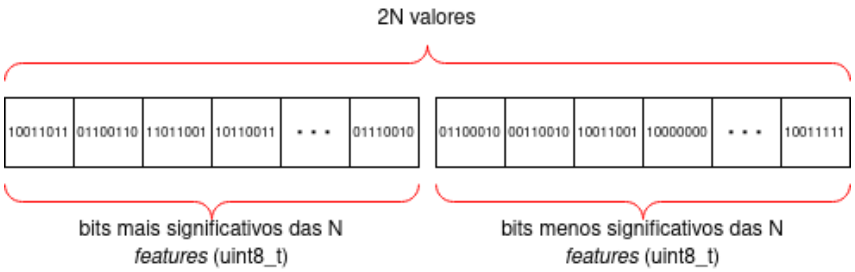

Figura 21 – N *features* que foram selecionadas sendo enviadas de maneira a separar os bits mais significativos e menos significativos de cada uma.

Como LoRaWAN é um protocolo de camada 2 (Enlace) no modelo OSI, as mensagens que transitam na rede são definidas como *frames* MAC. Na figura 22 é apresentado como esse frame é estruturado. O FRMPayload é o *payload* gerado pelo sensor.

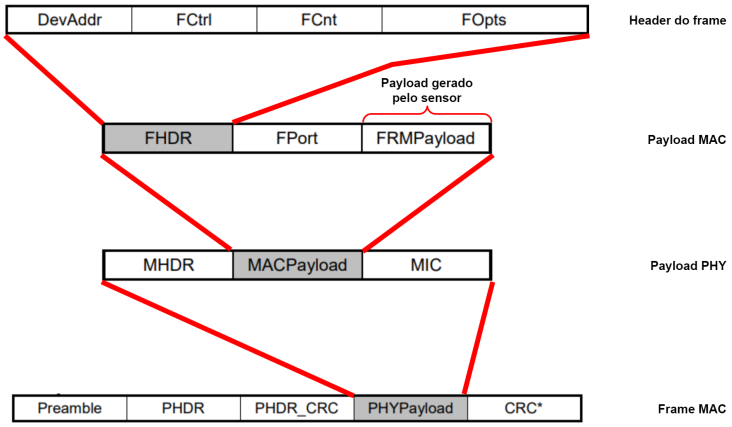

Figura 22 – Estrutura do frame MAC que é transmitido via protocolo LoRaWAN. Estruturas extraidas de Alliance (2017).

5.1.3 Comunicação

Os elementos que compõem a rede deste projeto são: *end-device*, que é o nó sensor, *Gateway* e *LoRaWAN Network Server*. O *end-device* já foi profundamente explicado em 5.1.2 e a seguir será explicado o funcionamento dos demais componentes da rede de comunicação.

Gateway

O *Gateway* é o componente da rede responsável pela interface entre os *end-devices* e o *LoRaWAN Network Server*, e faz o encapsulamento de mensagens LoRaWAN em pacotes IP.

Na figura 23 tem-se o *Gateway* utilizado. Trata-se de um computador de placa única de propósito geral, Raspberry Pi 3 Model B+, em conjunto com um *HAT* LoRa

Figura 23 – *Gateway* configurado, constituído de um Raspberry Pi 3 Model B+ e HAT LoRa RHF0M301.

modelo RisingHF - RHF0M301. Seu sistema operacional é o Raspberry OS Lite 32 bits, *open source* baseado em Linux.

Há dois importantes componentes de *software* para a interface LoRaWAN/IP. O primeiro é o componente que controlará o concentrador, peça de *hardware* existente no *hat* acoplado, responsável por receber e enviar pacotes LoRaWAN. O segundo componente realiza a interface entre LoRaWAN e IP, e realiza o reencaminhamento de pacotes de um tipo de rede para a outra. Os dois componentes concretos utilizados para cada função são o SEMTECH LoRa *Gateway* e SEMTECH LoRa *Packet Forwarder*, respectivamente. Ambos foram desenvolvidos pela SEMTECH e disponibilizados de forma *open source*.

Quando o *end-device* envia uma mensagem, ele o faz usando a técnica de modulação LoRa. Sendo assim, a transmissão não é feita para nenhum *Gateway* específico; a modulação LoRa se dá de maneira omnidirecional, portanto qualquer *Gateway* que estiver no alcance do sensor e estiver configurado para receber transmissões em uma dada frequência e com um dado fator de espalhamento, poderá receber as mensagens de um *end-device* com as mesmas configurações.

A frequência utilizada para a comunicação foi configurada para 915 MHz, permitida no Brasil. Dessa forma, toda mensagem LoRaWAN que seja transmitida nas bandas ao redor da frequência 915 MHz será captada pelo *gateway*. Porém, apenas os dispositivos cadastrados e autenticados na aplicação de monitoramento terão suas mensagens devidamente decriptadas no LoRaWAN Network Server.

Gateway - Semtech LoRa *Gateway*

O Semtech LoRa *Gateway* é o componente de software responsável por controlar a recepção e transmissão das comunicações LoRaWAN no *gateway*. Por meio da *Serial Peripheral Interface* do Raspberry Pi, o LoRa *Gateway* se comunica com o concentrador SX1276 presente no *HAT*. Dessa forma, o software atua como uma *Hardware Abstraction Layer* (HAL), e faz ser possível receber e transmitir mensagens LoRaWAN no dispositivo.

Gateway - Semtech LoRa Packet Forwarder

O Semtech LoRa Packet Forwarder recebe as mensagens LoRaWAN do concentrador do *gateway*, por meio da HAL estabelecida pelo Semtech LoRa *Gateway*, e os encaminha via UDP/IP para o *LoRaWAN Network Server*. Da mesma forma, pacotes recebidos do *LoRaWAN Network Server* são transmitidos pelo caminho inverso, chegando via UDP/IP e sendo enviados para os *end-devices* via rede LoRaWAN. Isto torna possível encaminhar os dados enviados pelos *end-devices* para o LoRaWAN Network Server, e também gerenciar os *end-devices* .

É neste componente que se configura as faixas de frequência utilizadas pelo *gateway* e também o endereço para comunicação com o *LoRaWAN Network Server*, por meio de arquivos de configuração.

LoRaWAN Network Server

Para o *LoRaWAN Network Server*, tem-se uma ampla variedade de soluções candidatas. A AWS oferece a solução comercial *IoT Core for LoRaWAN* de fácil configuração e gerenciamento, porém pouco customizável. A *The Things Network* tem uma das *stacks* mais populares para a implementação de uma rede LoRaWAN, prezando pela interconexão de diversas redes, tendo-se uma infraestrutura compartilhada pelos membros aderentes. A *The Things Industries* oferece uma solução robusta que suporta todas as versões de especificação e planos de operação do protocolo LoRaWAN, sendo ideal para implementações de grande escala e uso comercial. Por último, e a solução que foi escolhida para este projeto, tem-se o *Chirpstack*, uma solução gratuita e *OpenSource* de configuração complexa, porém oferece total controle do *LoRaWAN Network Server*, graças a grande gama de customização oferecida pela solução.

Além de ser gratuita, sendo solução mais adequada para um cenário com limitações orçamentárias como deste projeto, com um *LoRaWAN Network Server Chirpstack* é possível realizar a conexão com uma grande gama de *end-devices* sensores, até mesmo de construção própria, como é o caso deste projeto, e também a implementação de rotinas específicas para o tratamento dos dados, como por exemplo o envio dos dados para um serviço de processamento. Além disso, sendo uma solução *OpenSource*, tem-se uma comunidade ativa para auxiliar na depuração de problemas que possam surgir na

configuração e gerenciamento do servidor. Por fim, é uma solução que funciona bem tanto em escala menor, como uma prova de conceito, quanto para escala maior, o que facilita uma posterior conversão deste trabalho em uma solução comercial.

Para conectar o *Gateway* ao *LoRaWAN Network Server* basta fornecer seu *Gateway ID*, como exposto na figura 24. Como foi melhor explicado em 2.3, é possível ativar um *end-device* no *LoRaWAN Network Server* de duas maneiras diferentes: via OTAA ou via ABP. Para o contexto do projeto, este último é suficiente, tendo em vista que não será realizado nenhum tipo de federação ou conexão com a *The Things Network*(o que requereria o uso de OTAA). Neste caso, basta replicar o DevAddr e as chaves de sessão que foram definidas no *end-device*, no *LoRaWAN Network Server*. Na figura 25 essa configuração é demonstrada.

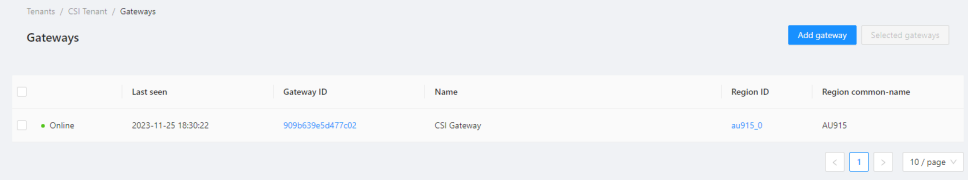

Figura 24 – *Gateway* conectado ao *LoRaWAN Network Server*.

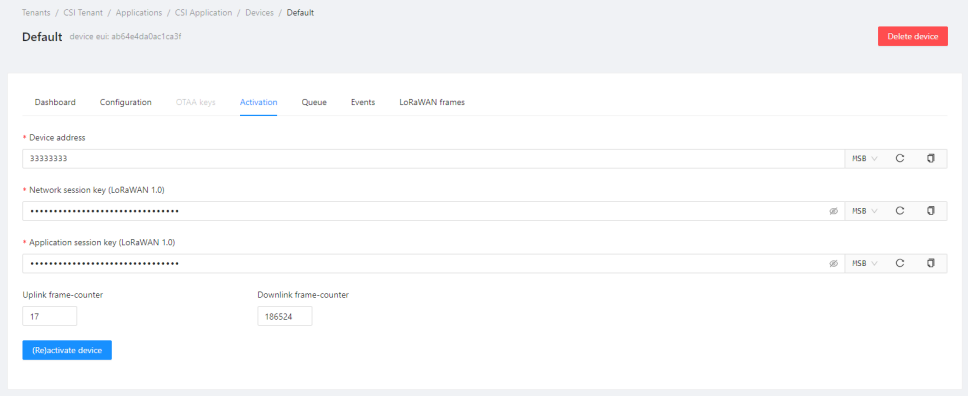

Figura 25 – *end-device* conectado ao *LoRaWAN Network Server*.

É possível monitorar os *frames* LoRaWAN que chegam nos *gateways* que foram adicionados ao *Chirpstack*. Todo *gateway* que estiver no alcance de um *end-device*, já que o envio dos dados via modulação LoRa se dá de forma omnidirecional, receberá seus *frames* LoRaWAN mesmo que o dispositivo não tenha sido ativado no *LoRaWAN Network Server*. Porém, para acessar as informações do *payload*, é necessário a ativação via OTAA ou ABP. Na figura 26 apresenta-se os *frames* de *end-devices* não ativados, já na figura 27 são os de um ativado. Nesta última é possível visualizar a informação do *payload* em hexadecimal dos caracteres ASCII correspondentes.

Para processar as informações enviadas pelo *end-device*, recebidas pelo *Gateway* e encaminhadas para o *LoRaWAN Network Server*, realizou-se uma integração HTTP.

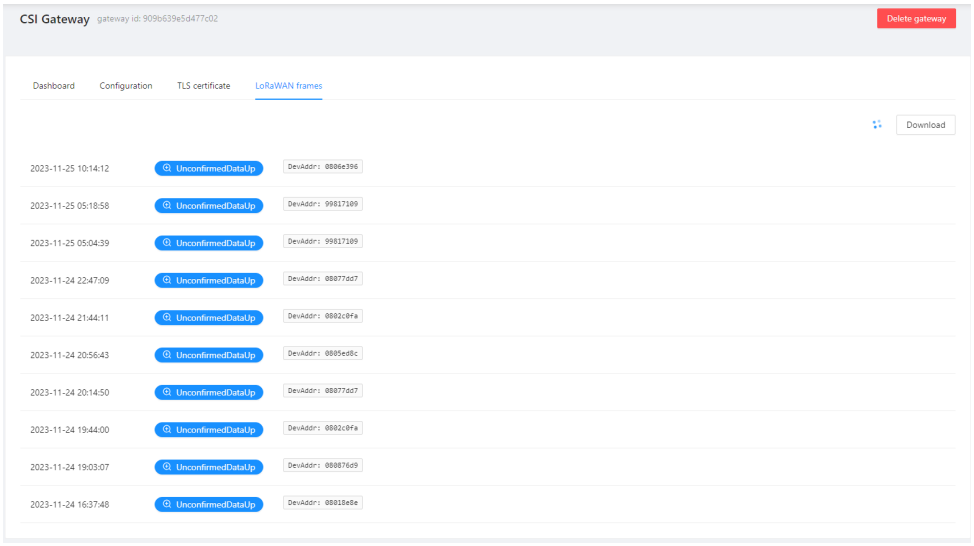

Figura 26 – LoRaWAN *frames* de um *end-devices* que não foram ativados no *LoRaWAN Network Server*.

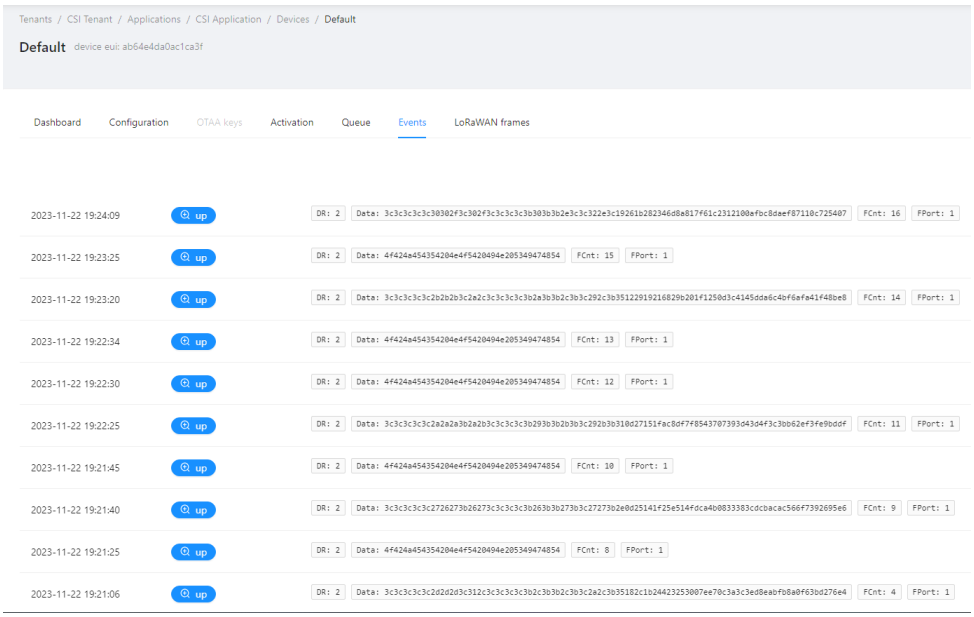

Figura 27 – LoRaWAN *frames* de um *end-device* que foi ativado no *LoRaWAN Network Server*.

A partir da URL do API Gateway configurado na AWS (melhor explicado em 5.1.4) e de uma chave de autorização "x-api-key", foi realizada a integração que encaminha todo *payload* recebido de um *end-device* ativado para o fluxo de processamento via requisição HTTP POST, com o *payload* em formato JSON. Na figura 28 é demonstrado como essa configuração é feita via web UI do Chirpstack.

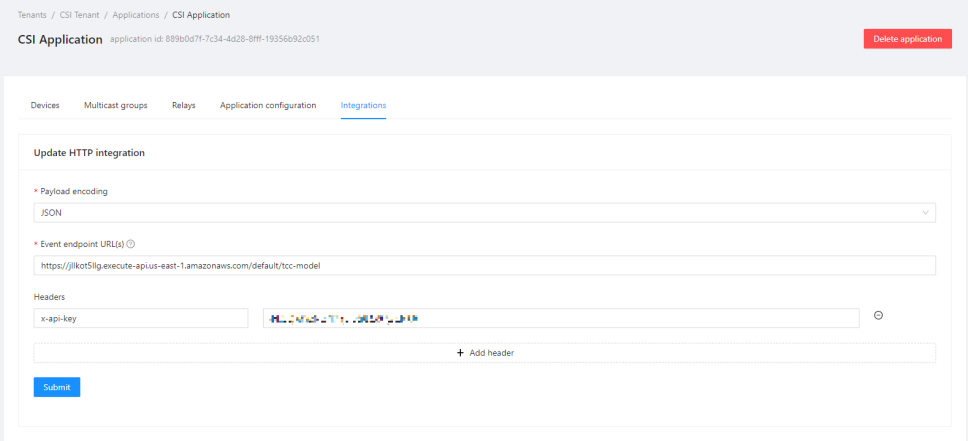

Figura 28 – Configuração de integração HTTP da aplicação a qual o *end-device* ativado faz parte.

5.1.4 Processamento e visualização

Identificação de eventos

Além da detecção de eventos, a aplicação também deve identificá-los. É necessário definir em qual etapa da aplicação a identificação será realizada. A identificação pode ser feita na borda, ou em nuvem. Cada uma das alternativas implica em vantagens e desvantagens, que serão discutidas nesta seção.

Optar por realizar na borda causa um maior uso de recursos computacionais no *end-device*. Essa maior demanda de processamento acarreta em maior gasto de energia, e pode comprometer outros processos executando em conjunto. Sendo assim, há uma maior concorrência pelo uso de um dado núcleo do processador, podendo causar uma perda de resolução das medições do CSI. Por exemplo, em um sensor baseado em WiFi, é necessário um processo responsável por coletar as medições de CSI. Caso o processo tenha seu tempo de processamento reduzido, a taxa de amostragem de CSI pode reduzir. Do ponto de vista prático, isto implica em objetos que transitam de maneira muito rápida não sejam detectados. Além disso, dependendo do método de identificação, uma maior quantidade de RAM precisará ser utilizada, o que pode diminuir a quantidade de amostras que o sensor consegue armazenar de uma vez, e portanto diminuir sua percepção temporal do ambiente.

Outro ponto importante a se avaliar é a qualidade da identificação realizada. Na borda, algoritmos de *Machine Learniung* que demandam menos memória precisam ser utilizados. Por exemplo, caso uma abordagem baseada em redes neurais artificiais esteja em uso, o tamanho da rede deverá ser menor, devido à limitação de RAM. A capacidade de aprendizado de uma rede neural é proporcional ao seu tamanho. Quanto mais complexo o problema a ser resolvido, mais parâmetros a rede precisará ter para aprender uma solução. Dessa forma, a acurácia da identificação pode ser menor ao se utilizar uma rede pequena. Outros modelos, como o KNN (*K-Nearest Neighboors*), são modelos não-paramétricos e

que dependem de dados existentes em tempo de execução para realização da inferência. Com a quantidade limitada de RAM, menos dados estarão disponíveis no momento da identificação de um evento, e portanto a acurácia também poderá ser reduzida.

Já quando se trata de execução da identificação em nuvem, é necessário o envio dos dados do evento para um servidor ou serviço de processamento, o que pode gera dois problemas. O primeiro deles está relacionado a transmissão dos dados pela infraestrutura de rede. Quando a inferência é executada em borda, o dispositivo IoT precisa apenas enviar o tipo de evento classificado, com eventuais informações adicionais. Já para a execução em nuvem, será necessário enviar todos os dados requeridos pelo modelo de inferência, que podem ser vetores, matrizes, imagens, áudio, entre outros. Esse envio aumenta o tempo de transmissão necessário e a vazão requerida da rede. O segundo problema é a latência. Caso o dispositivo IoT precise do resultado da identificação para realizar uma ação por meio de atuadores, será necessário esperar pela resposta da nuvem. Neste caso, os requisitos de latência baixa da rede serão mais restritivos, e pode não ser viável realizar a inferência em nuvem.

Em compensação, a inferência em nuvem permite que modelos mais complexos sejam utilizados, o que permite acurácias maiores. Entretanto, o aumento de acurácia nem sempre será realidade, visto que a necessidade de transmitir os dados pela rede pode fazer com que dados simplificados ou reduzidos sejam enviados, por questões de limitação de vazão, o que também impacta na performance dos modelos.

Identificação de eventos na aplicação de monitoramento de tráfego

No contexto deste projeto, a discussão entre executar a inferência em borda ou em nuvem deve levar em conta os dispositivos e tecnologias sendo utilizados.

O par de ESP32, utilizados como dispositivo de borda, cada um possui dois núcleos de processamento e 520 kilobytes de memória RAM. Para que a coleta de dados CSI não seja interrompida, um dos núcleos deve ficar responsável somente pela comunicação WiFi. Caso a inferência seja realizada em borda, ela deverá dividir tempo de um núcleo com o tratamento dos dados CSI, detecção de eventos, e transmissão LoRa. Essa divisão de tempo de processamento afeta os outros processos. O tratamento dos dados CSI, com tempo em CPU reduzido, pode se tornar um gargalo, diminuindo a resolução efetiva do sensor. Além disso, os algoritmos de inferência precisam ser mais leves para não tomar muito tempo de CPU, o que impacta na acurácia das predições.

A baixa quantidade de memória RAM também limita os algoritmos de inferência, necessitando que contenham menos parâmetros, no caso de modelos paramétricos, ou utilizem menos dados em tempo de inferência, no caso de modelos não paramétricos. Para ambos os tipos de modelo, poderia-se tentar contornar parcialmente a limitação de memória RAM utilizando a memória flash de 4 MB para armazenar parâmetros ou

dados. Por exemplo, redes neurais poderiam ter uma camada carregada por vez, e a inferência realizada em etapas, enquanto um KNN poderia calcular as distâncias em partições dos dados, em vez de utilizar o conjunto todo de uma vez. Entretanto, isto introduz maior complexidade na criação dos algoritmos, além de introduzir potenciais atrasos de processamento, devido à leitura dos dados na flash ser mais lenta. Os atrasos poderiam ser parcialmente resolvidos com um chaveamento bem planejado de processos, entretanto o *overhead* de troca de contexto ainda existiria.

A implementação de um modelo em borda também é mais complexa, e, caso uma modelagem nova seja desenvolvida, é mais difícil reimplementar o código embarcado e fazer a substituição em dispositivos existentes.

Por outro lado, o código executando em nuvem não possui as restrições de processamento e tamanho comentadas anteriormente. Neste caso, o gargalo passa a ser a rede utilizada para transmissão de dados. Ao se tratar de LoRaWAN, o tamanho máximo de pacote é 242 bytes, o que limita o tamanho das *features* extraídas do CSI. Para enviar mais dados em um único pacote, é necessário diminuir o parâmetro SF da transmissão, o que limita o alcance desta, e aumenta o tempo necessário para envio. Para remediar o problema de alcance, é possível enviar os dados de forma fragmentada em vários pacotes, o que possibilita o uso de um SF menor. De ambas as formas, o tempo total de transmissão é maior do que quando a inferência é realizada em borda, pois neste caso o pacote de dados seria constituído apenas da classificação do evento. Esse maior tempo de transmissão limita a quantidade de eventos que podem ser reportados em um certo intervalo de tempo.

A inferência em nuvem também permite a fácil substituição de um modelo por outro, pois não necessita substituição do código do modelo no embarcado. Também se torna possivel o uso de bibliotecas não embarcadas de linguagens de alto nível, como Python, para criação facilitada dos modelos.

A aplicação de monitoramento de tráfego visa ser posta em prática em vias remotas, onde há menor quantidade de eventos ocorrendo em relação a vias urbanas. Dessa forma, o impacto causado pelo maior tempo de transmissão pode ser tolerado. A linha de visada entre receptores e transmissores LoRa possui menos obstáculos, devido à ausência de grandes edifícios e construções, e portanto também é possível diminuir o SF para envio de mais dados.

Para este projeto, optou-se por realizar o processamento em nuvem, e portanto será necessário projetar a rede e o modelo de *Machine Learning* de forma que seja possível enviar os dados necessários para a identificação dos eventos. Dessa forma, os eventos serão detectados em borda, porém identificados na nuvem.

Modelo de Machine Learning para identificação de eventos

Nesta aplicação, foi utilizado *Machine Learning* para criação do software capaz de identificar os eventos. Para esta abordagem, foram necessários dados de treino, utilizados para ajustar o modelo, e dados de teste, para medir sua performance.

Para ambos, dados treino e teste, dispõe-se de *datasets* de matrizes CSI coletadas para diferentes objetos passando em frente aos sensores. Cada *dataset* possui exemplos de carros, pessoas, cachorros, e vacas. Os dados foram coletados com os sensores a uma distância de 12 metros entre si, em estradas de terra, cascalho e em pasto. A coleta e disponibilização de dados para o projeto foram cortesia de Samuel Vieira Ducca.

Os *datasets* passaram por um processo de *data augmentation*, no qual novos exemplos foram criados de forma sintética, para aumentar a quantidade de dados disponíveis para o aprendizado do modelo sem que seja necessário coletar mais dados. Mais especificamente, foram utilizadas *sliding windows*. Isto é, cada exemplo gerou novos por meio do deslocamento temporal da matriz, representando objetos sendo visualizados em instantes diferentes. No total, há 1106 exemplos no conjunto de treino, e 301 exemplos no de teste. Na figura 29 apreseta-se um exemplo de dado coletado para cada classe.

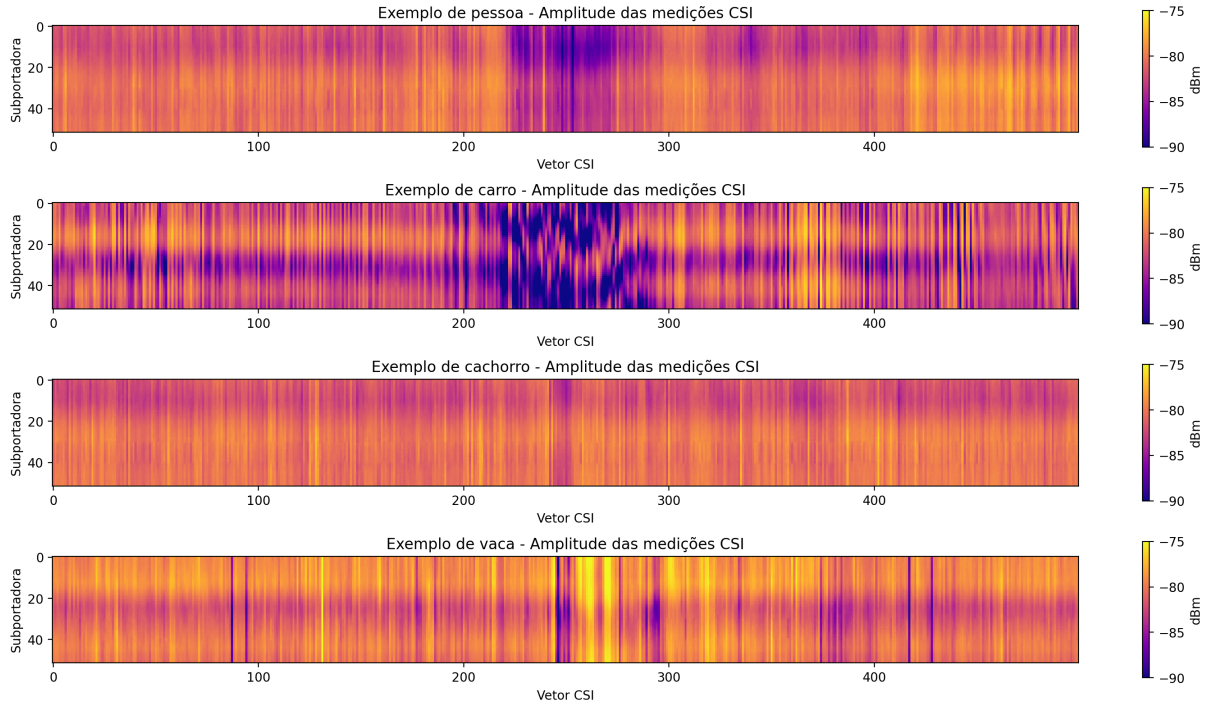

Figura 29 – Exemplos de dados CSI coletados, para cada classe.

Ajuste do modelo de identificação

Para o treino de modelos, primeiramente é necessário extrair *features* dos dados. As *features* de um modelo de Machine Learning são os dados referentes a um exemplo

a ser classificado, e não necessariamente o dado bruto. Por exemplo, para modelos que atuam sobre áudios, a *feature* de entrada pode ser a própria onda sonora, entretanto, isto é um dado muito complexo e grande. Portanto, pode ser mais vantajoso extrair *features* derivadas, como frequências componentes da onda, que serão mais simples, menores e fáceis de um modelo lidar.

Para os dados CSI, as *features* utilizadas serão estatísticas. As matrizes consistem de 500 vetores de 52 medidas de amplitude, uma para cada subportadora não nula. Dessa forma, são extraídas 52 médias e 52 desvios padrões, ao longo das 500 medições, totalizando um vetor de 104 *features* para um dado exemplo. Antes de calcular as médias e desvios, a matriz CSI é inteira normalizada pela média de seus elementos.

Na figura 30 apresenta-se as médias e desvios padrão para cada classe. É possível notar que cada classe possui padrões diferentes de média e desvio padrão, o que pode ser utilizado por modelos de *Machine Learning* para aprender a diferença entre elas.

Além disso, também é importante avaliar a codificação das *features* e quais realmente são relevantes. Por padrão, os dados estão codificados em *floating point* de 32 bits. O uso de 32 bits para cada *float* faz com que as features contidas em cada evento de detecção tenham $104 * (4 \text{ bytes}) = 416$ bytes. Como o tamanho máximo de mensagem LoRaWAN é de 242 bytes no menor SF possível (SF7), isto acarreta na necessidade de quebra de pacotes mesmo no SF de maior capacidade de transmissão. Portanto, é necessário reduzir o tamanho dos dados enviados. O processo de tratamento dos dados para que seja possível enviá-los em uma única mensagem foi explicado em 5.1.2.

Com as *features* extraídas, pode-se prosseguir para o ajuste de um modelo. A abordagem utilizada foi o KNN (*K-Nearest Neighbors*) (CUNNINGHAM; DELANY, 2020). Esta é uma abordagem não paramétrica, onde o próprio conjunto de dados de treino é utilizado em tempo de inferência. O algoritmo recebe vetores como entrada, e associa uma nova observação à uma classe existente nos dados de treino, com base nos vetores mais próximos levando-se em conta a distância euclidiana entre eles. Para isso, é necessário definir o hiperparâmetro K. Por exemplo, se o K escolhido for 5, e, dada uma nova observação, as 5 mais próximas corresponderem a 3 carros e 2 cachorros, a classificação da observação será 'carro'. Hiperparâmetros de modelos de *Machine Learning* geralmente são escolhidos de forma empírica, por meio de buscas e métodos heurísticos.

Usualmente, valores de K ímpares são escolhidos, para evitar empate de classes durante a classificação. Na figura 31 apresenta-se a performance do KNN na classificação das *features* extraídas do CSI. A performance é avaliada para diferentes valores de K ímpares, e para dados codificados em *floating point* de 16 e 32 bits. Nota-se que, em geral, há perda de performance ao diminuir a precisão dos *floats*, em até 1%. Entretanto, isso reduz o pacote de dados de 416 bytes para 208 bytes, valor que já cabe em uma única mensagem LoRaWAN, e portanto esse *trade off* é necessário.

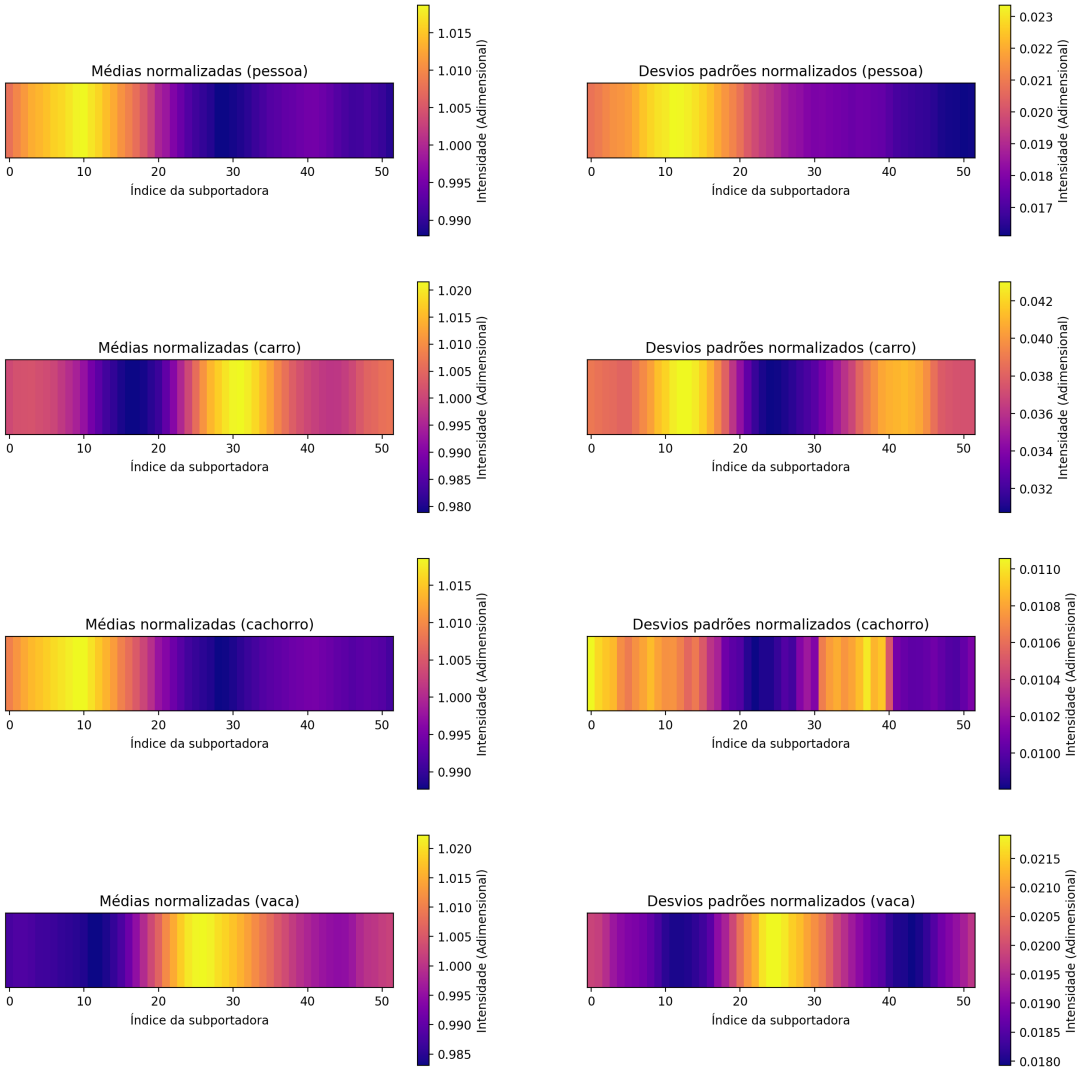

Figura 30 – Médias e desvios padrões das subportadores. Cada linha representa um exemplo de cada uma das classes. A primeira coluna apresenta as médias, e a segunda coluna os desvios padrões.

Valores baixos de K fazem o KNN ser suscetível a ruído nos dados (MELEK; MELEK; KAYIKCIOGLU, 2017). Por exemplo, para K = 1, a remoção ou adição de um exemplo do conjunto de treino é facilmente capaz de alterar a classificação de diversos novos dados, já que a classificação depende somente do único vizinho mais próximo. Portanto, isto pode fazer com que o modelo não generalize bem para novos dados, deixando características muito específicas dos exemplos coletados influenciar as predições.

Por outro lado, valores de K muito altos diminuem a capacidade do modelo de diferenciar as classes. Imagine o caso extremo em que K seja igual ao número de exemplos no conjunto de treino. Neste caso, o KNN sempre associaria uma nova observação à classe mais frequente. Em outra palavras, a predição seria sempre a moda das classes do *dataset* de treino, o que é uma predição que poderia ser feita ignorando-se completamente as *features* calculadas, utilizando apenas o cálculo da moda.

Portanto, é necessário escolher um valor intermediário de K adequado, com base na performance observada. Uma abordagem comum é assumir $k = \sqrt{n}$, onde n é a quantidade de observações no conjunto de dados de treino. Neste caso, seguindo esta abordagem, o valor de k deve ser $k = \sqrt{1106} \approx 33$. A partir da figura 31, nota-se que as maiores performances ocorrem para valores entre 21 e 41, o que está de acordo com o valor calculado anteriormente.

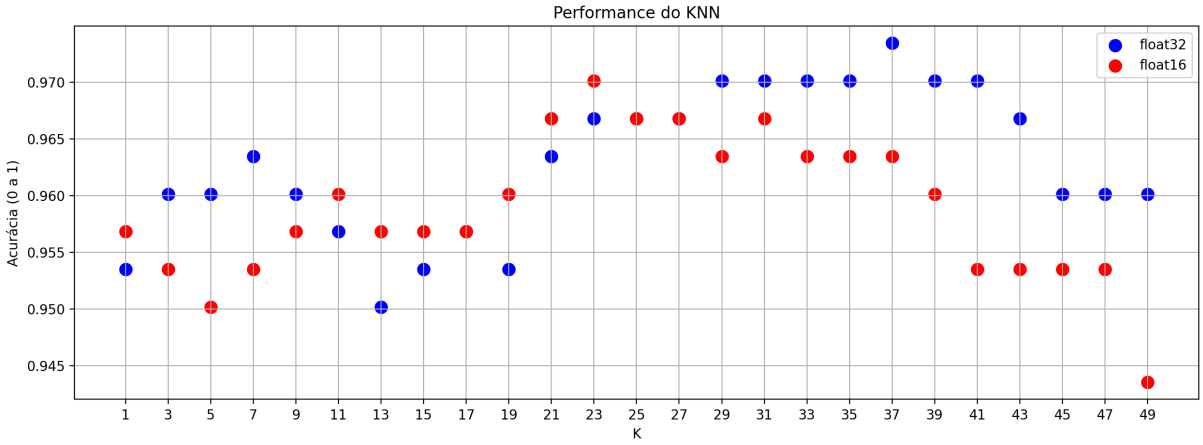

Figura 31 – Acurácia do KNN no problema de classificação de carros, pessoas, cachorros e vacas, utilizando as *features* extraídas da matriz CSI.

Além da acurácia, é interessante visualizar quais dados foram classificados incorretamente, qual seria a classe real e qual foi a predita. Na figura 32 apresenta-se a matriz de confusão do modelo ajustado com K=31 e dados em *float16*. O eixo vertical representa a classe real, e a horizontal o que o modelo identificou. Vê-se que houve confusão do modelo com dados de pessoas, cachorros e vacas. Entretanto, todos os dados de carro foram classificados corretamente.

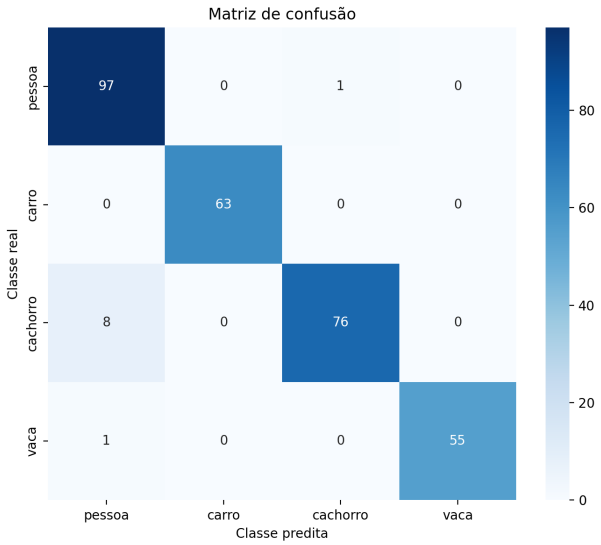

Figura 32 – Matriz de confusão do modelo treinado com *float16* e K=31.

Seleção de *features*

Outro ponto importante a se considerar é quais *features* são as mais importantes para discernir as classes. Eliminar *features* pouco relevantes é o processo conhecido como *feature selection*. A seleção de *features* é necessária pois pode aumentar a acurácia das classificações, e reduz o tamanho do vetor de entrada para o algoritmo de identificação. No caso do projeto, a redução do vetor é de extrema importância para a diminuição das mensagens transmitidas via LoRaWAN.

Há diversas maneiras de se realizar o *feature selection*. A que será utilizada neste projeto é a técnica de *permutation importance* (SCIKIT-LEARN, 2023a). Nesta abordagem, é necessário um modelo já ajustado com todas as *features* originais, e um *dataset* de exemplos para identificação. Para cada feature, permuta-se aleatoriamente seu valor entre os exemplos do *dataset*. Isto é, cada vetor de entrada irá receber a *feature* de um outro vetor. A ideia é quebrar a relação entre a distribuição da variável preditora e variável predita. Após a permutação, o conjunto é inserido no modelo, e a performance da classificação é medida. O processo é repetido uma certa quantidade de vezes para cada variável preditora. Ao final, avalia-se quais features tiveram o maior impacto na queda de performance quando foram embaralhadas. Estas serão as mais importantes.

Para este processo, será utilizado o modelo KNN ajustado com $k = 31$ e dados codificados em *float16*. Na figura 33, é possível avaliar o decréscimo médio na acurácia ao se permutar cada *feature*, junto ao desvio padrão da medição.

Como este método é estatístico, há sensibilidade à quantidade de permutações feitas e condições iniciais. Para sua execução, foi utilizada a função *permutation_importance* da biblioteca *sklearn*, com *random state* (semente aleatória) de valor 1, e 50 permutações de cada variável preditora.

Com as importâncias calculadas, é necessário estabelecer um critério para seleção. Neste caso, estabelecem-se limites de corte para a importância de cada *feature*. Chamando-se de μ_i e σ_i a média e o desvio da importância da feature i, respectivamente, calcula-se o limite de corte $f(\mu_i, \sigma_i)$, e seleciona-se a variável preditora caso $f(\mu_i, \sigma_i) > 0$, sendo

$$f(\mu_i, \sigma_i) = \begin{cases} \mu_i - N\sigma_i, se\ \mu_i >= 0 \\ \mu_i + M\sigma_i, se\ \mu_i < 0 \end{cases}$$
$$\text{onde } N,\ M \in \mathbb{R}^+$$

Isto é, o critério de seleção equivale a estabelecer um intervalo de confiança para μ_i. Os valores de N e M controlam o nível de confiança. Quando a importância medida é positiva, considera-se o corte N desvios padrões abaixo, e quando é negativo, considera-se M desvios padrões acima.

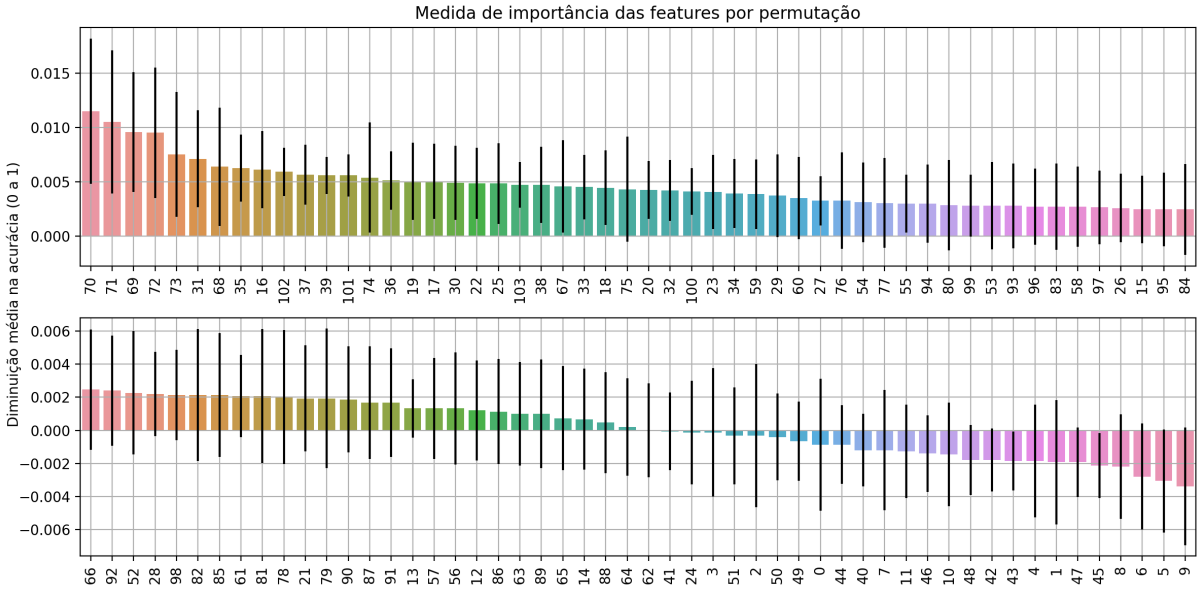

Figura 33 – Medição da importância de cada *feature*. A altura das barras representa a diminuição média na acurácia e as barras pretas o desvio padrão.

Dessa forma, quanto menor o valor de N, e maior o valor de M, mais variáveis preditoras serão inclusas na seleção final. Na figura 34, o critério foi aplicado para valores de $0 <= N, M <= 3$, em incrementos de 0.5. Nota-se que é possível reduzir a quantidade de features de 104 para 33, com uma acurácia de 98%. Isto reduz o tamanho do *payload* das mensagens na rede de 208 bytes para 66 bytes, ao mesmo tempo que mantém a performance do modelo.

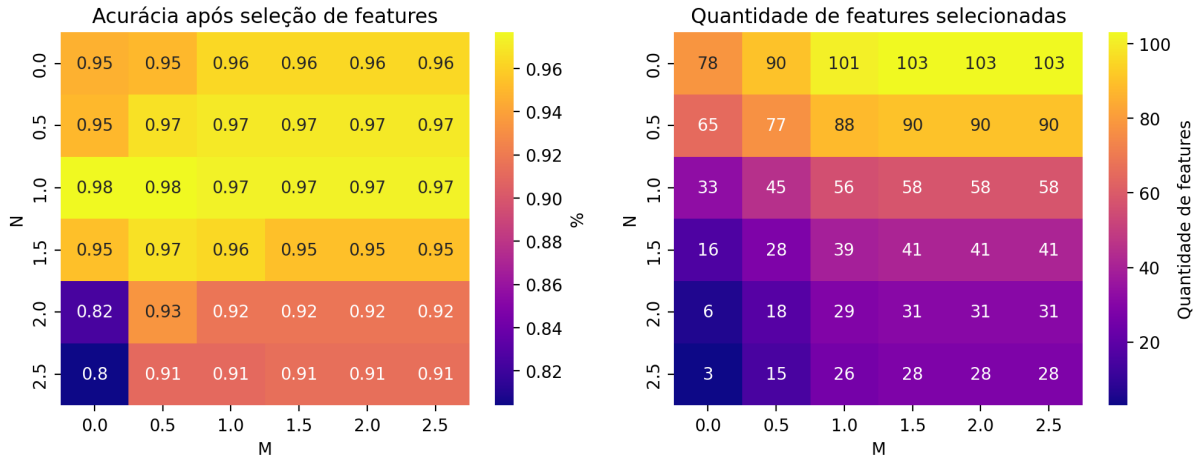

Figura 34 – Acurácia e quantidade de *features* para cada critério de seleção.

Comparação do KNN com outros modelos de *Machine Learning*

Além do ajuste do KNN, é interessante avaliar alguns outros modelos no mesmo conjunto de dados, para avaliar o quão bem performam comparados ao KNN. Os algoritmos

de classificação avaliados serão *Random Forest*, Regressão Logística e *Support Vector Machine*.

O modelo *Random Forest* (HO, 1995) faz a criação de uma quantidade pré determinada de árvores de decisão, em seguida, cada uma das árvores realiza uma classificação a partir das *features* e a classificação final é dada por uma votação entre elas. A regressão logística (JAMES et al., 2021) visa fazer a separação dos dados por meio da combinação linear das *features*, seguido da aplicação da função logística, de forma a obter uma medida de probabilidade de o dado observado pertencer a uma certa classe. O algoritmo *Support Vector Machine* (JAMES et al., 2021) realiza a separação das classes por meio de uma fronteira de decisão no espaço das *features*. Essa fronteira pode ser tanto linear quanto não linear, por meio da seleção de kernels utilizados no ajuste do modelo.

Cada um dos modelos possui um conjunto de hiperparâmetros que devem ser ajustado, pois controlam o funcionamento do modelo, e um subconjunto de hiperparâmetros pode apresentar performance melhor do que outros. Esse ajuste é análogo ao ajuste do valor de K, feito para o algoritmo KNN, sendo naquele caso o único hiperparâmetro considerado.

O ajuste de hiperparâmetros é feito por meio de uma busca heurística. Aqui, será utilizada a classe *RandomizedSearchCV* da biblioteca *scikit-learn*, que faz uma busca aleatória em um espaço pré definido de valores. Os três modelos também serão obtidas por meio de classes disponíveis na mesma biblioteca.

Na tabela 2, apresenta-se a performance de cada modelo no conjunto de teste, após seu ajuste no conjunto de treino. Também são apresentados os hiperparâmetros considerados na busca. Os nomes dos hiperparâmetros são os mesmos aceitos pelas classes na biblioteca *scikit-learn*.

	Acurácia	Hiperparâmetros ajustados
Random Forest	91,03%	n_estimators, max_depth, max_features, min_samples_split, min_samples_leaf, criterion
Regressão Logística	85,38%	C, penalty, solver
Support Vector Machine	86,05%	C, gamma, kernel

Tabela 2 – Acurácia de cada modelo. Junto à acurácia, são apresentados os hiperparâmetros considerados na busca de hiperparâmetros.

Nota-se que os três modelos foram capazes de aprender a discernir as classes, com acurácias que vão de 85% a 91%. É possível obter acurácias maiores por meio de mais ajustes de hiperparâmetros, visto que a busca é heurística e não necessariamente encontrará os melhores valores possíveis. Além disso, a aplicação de um processo de *feature selection* a cada modelo pode auxiliar no aumento de performance.

Embora os três modelos sejam alternativas viáveis para a aplicação de monitoramento, suas performances não ultrapassam à do KNN, mesmo em sua versão antes da seleção de *features*. Dessa forma, o modelo escolhido foi o KNN.

Coleta de dados em ambiente urbano

Anteriormente, demonstrou-se possível realizar o treinamento de um modelo de *machine learning* para identificação de eventos com base nos dados coletados em um ambiente rural. Entretanto, o ambiente de testes do projeto não foi o mesmo, pois este foi conduzido em local urbano, e não houve acesso por parte dos membros deste trabalho a uma localidade rural. Por conta disso, dada as entidades que usualmente se encontram no ambiente de teste, o modelo foi treinado para classificar apenas duas entidades: pessoas e carros.

A distribuição dos dados que os sensores coletaram em ambiente urbano não foi a mesma do ambiente rural, havendo fatores que podem ter influenciado os valores medidos das matrizes CSI. Entre eles, o fato de modelos diferentes da ESP32 terem sido utilizados para a coleta de dados, haver objetos diferentes ao redor do sensor, como edifícios e carros estacionados, e o chão ser constituído de outro material (terra no ambiente rural e asfalto no ambiente urbano). Dessa forma, foi necessário coletar dados novos no ambiente em que o projeto foi testado. Na figura 35, apresenta-se o *setup* montado para coleta de dados. Na figura 36 apresenta-se uma das matrizes coletadas.

Foram coletadas 32 matrizes, sendo 18 de carros e 14 de pessoas. Após extração das *features*, foi ajustado um novo modelo KNN com $k = \lfloor\sqrt{32}\rfloor = 5$. Além disso, foram utilizadas as 16 *features* correspondentes a 95% de acurácia na figura 34. A codificação usada foi a de *float16*.

Os dados foram aleatoriamente separados em 50% para treino e 50% para teste, e o modelo foi ajustado. No conjunto de testes, constituído de 10 carros e 6 pessoas, o modelo foi capaz de identificar todos corretamente, com uma acurácia de 100%. Para teste em campo, o modelo foi ajustado com os 32 dados, e sua performance verificada na prática. Neste caso, onde o modelo é treinado com todos os dados, ele obtém 96,87% de acurácia no próprio conjunto de treinos, falhando apenas em identificar um dado de pessoa, que é classificado como carro.

Assim, o *payload* de dados foi de 32 bytes. Considerando um *overhead* de 13 bytes nos pacotes, isso permitirá que a rede funcione com valores de SF7 a SF9, possibilitando maior alcance quando necessário.

Figura 35 – *Setup* para coleta de dados em ambiente urbano, em frente ao prédio onde está localizado o Departamento de Engenharia de Computação e Sistemas Digitais (PCS) da Poli-USP. À esquerda, vê-se o receptor CSI, responsável pela coleta das medições CSI, e o notebook para onde os dados são exportados via saída serial. À direita, vê-se o *access point* WiFi, ao qual o receptor se conecta.

Dashboard

A *dashboard* foi construída na linguagem Python, a partir do *framework* Dash/Plotly. Na figura 37 apresenta-se sua interface, na qual é possível escolher filtros e agregações para os dados, de forma a apresentar os eventos em um gráfico. Pode-se escolher a rua, intervalo de tempo, tipo de evento (pessoa, carro, vaca, cachorro e perigo) e qual escala de tempo (minuto, hora, dia, semana, mês) deseja-se analisar os dados.

Os dados são obtidos por meio de uma *query* SQL feita ao banco de dados, e manejados por meio da biblioteca Pandas. Para que seja possível saber a qual via cada evento pertence, há uma estrutura de dados que mapeia o ID de cada *end-device* para o nome da localização onde ele está.

Os componentes da interface, como os seletores de opções e datas, são disponibilizados pela biblioteca Dash, e a organização da tela é feita utilizando uma estrutura análoga a HTML. O gráfico é construído utilizando a biblioteca Plotly.

A *dashboard* também disponibiliza uma tabela de métricas relevantes a respeito dos dados. Na figura 38 é apresentado um exemplo desta tabela. Calcula-se o número total de eventos, quantidade média, máxima e mínima de eventos, e o desvio padrão. Os valores dizem respeito ao intervalo de tempo, unidade de tempo, localidade e tipo de evento selecionados na interface.

Para que seja possível visualizar novos dados que chegam dinamicamente, sem que seja necessário recarregar a página, a *query* SQL é executada a cada 5 segundos, tempo de

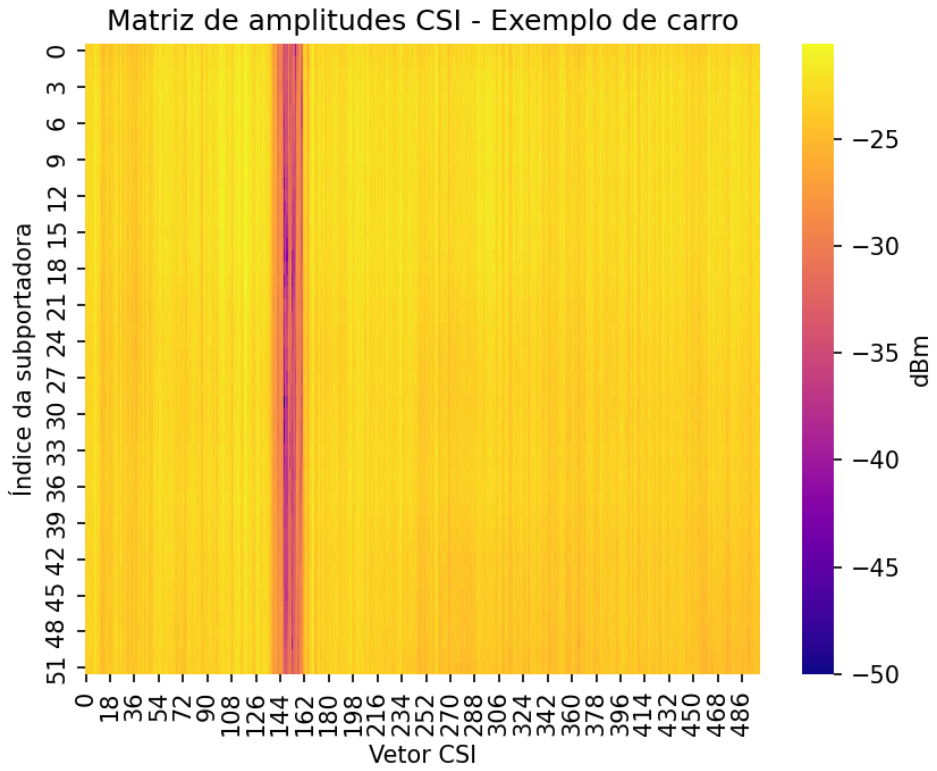

Figura 36 – Exemplo de matriz CSI para um dos eventos coletados. É possível enxergar uma faixa mais escura, momento em que um carro passou entre os sensores, resultando na queda das amplitudes das subportadoras.

atualização que é configurável.

5.2 Testes e Avaliação

5.2.1 Testes em laboratório

Ao longo do desenvolvimento das partes do sistema, realizou-se testes em ambiente interno, dos componentes individuais e suas integrações, conforme descrito no capítulo 3.

Coleta de dados CSI

Os sensores foram utilizados para coletar dados CSI, enviá-los via saída serial para um computador, e exibidos ao vivo. Dessa forma, foi possível analisar a influência que objetos passando entre os sensores exercem no sinal, e também o efeito da orientação relativa das antenas WiFi. Constatou-se que os objetos afetavam de maneira perceptível as amplitudes da matriz CSI, e portanto seria possível detectar eventos por meio da queda das amplitudes. Além disso, a orientação relativa das antenas, assim como o ambiente ao redor, afetam o CSI. Caso o alinhamento e o ângulo entre as antenas seja alterado durante a coleta, há perturbação no sinal, embora as antenas sejam omnidirecionais. Além

Figura 37 – *Dashboard* construída em Plotly. É possível filtrar os dados por tipo de evento, localidade e intervalo de tempo.

Metric	Value
Total Number of Events	2179
Average Events per Week	41.90
Maximum Events in a Week	55.00
Minimum Events in a Week	29.00
Standard Deviation	5.55

Figura 38 – Tabela de métricas da *Dashboard*.

disso, objetos passantes atrás ou ao redor dos sensores também afetam as medições, devido a reflexões de ondas. Dessa forma, é necessário que os sensores estejam alinhados e se mantenham em posição e orientação fixa durante a operação. Também deve-se evitar a passagem de objetos próximo aos sensores fora da linha de detecção.

Detecção de eventos

Para a detecção de eventos, é necessário que o sensor seja calibrado, por meio da definição de uma amplitude média de referência das subportadoras. Em seguida, o sensor monitora a queda das amplitudes, e considera um evento detectado caso as amplitudes caiam abaixo de um percentual em uma certa janela de tempo. Dessa forma, há dois parâmetros importantes a serem avaliados para a detecção, que são o percentual de queda da amplitude média do CSI, abaixo do qual um evento é detectado, e o tamanho do intervalo de tempo.

Em relação a calibragem, constatou-se que os sensores não devem ser movidos após esta ser feita, pois a referência de amplitude se altera. Já em relação aos parâmetros, notou-se que os valores ideais dependem da configuração espacial dos sensores e do ambiente. Caso o intervalo de tempo e o percentual de queda de amplitude sejam baixos, ruídos no ambiente podem levar a falsos positivos. Já caso os parâmetros sejam elevados, objetos que se locomovam rapidamente, ou que atenuem menos as amplitudes CSI, podem não ser detectados. Dessa forma, é necessário encontrar a configuração ideal dos parâmetros por meio de testes no ambiente de implantação dos sensores.

Envio de pacotes LoRaWAN

Primeiramente, utilizou-se 2 dispositivos ESP32 para transmissão de pacotes LoRaWAN, onde um atuava como transmissor WiFi e outro como receptor WiFi, e verificou-se o funcionamento da transmissão. Além disso, os dispositivos foram configurados para se comunicar via LoRaWAN e WiFi ao mesmo tempo, de forma a garantir que ambas as antenas fossem capazes de funcionar concomitantemente, o que foi constatado.

Em seguida, os dispositivos ESP32 foram configurados para enviar dados CSI via LoRaWAN para o *gateway*, com este já configurado. Constatando-se o recebimento dos dados.

Comunicação entre *gateway* e *LoRaWAN Network Server*

Foi configurada uma aplicação no *LoRaWAN Network Server* ChirpStack, de modo a receber eventos do *gateway*. Em seguida, dados CSI foram enviados pelos sensores ESP32, e foi possível recebê-los corretamente no servidor.

Fluxo do sensor até o banco de dados

Foi configurada uma integração do ChirpStack com o serviço AWS Lambda na qual o modelo de *Machine Learning* é executado, e a lambda foi configurada para envio dos dados ao banco de dados. Dessa forma, todo evento recebido pelo *LoRaWAN Network Server* seria encaminhado para a lambda, e salvo no banco de dados após ser processado

pelo modelo. Em seguida, os dispositivos ESP32 foram utilizados para transmitir eventos, e constatou-se que chegavam corretamente ao banco de dados.

Implantação do modelo

Com o fluxo completo desde o nó sensor até o banco de dados testado, implantou-se o modelo de *Machine Learning* no serviço AWS Lambda. Em seguida, eventos foram transmitidos pelo sensores, e contatou-se que o modelo os identificava e os repassa ao banco de dados.

Integração do banco de dados e *dashboard*

A *dashboard* inicialmente foi desenvolvida com dados fictícios gerados aleatoriamente. Com o desenvolvimento finalizado, foi integrada ao banco de dados real, e verificou-se a correta aquisição e exibição dos eventos.

5.2.2 Testes em ambiente externo

Foram realizados testes externos em duas datas diferentes, para se avaliar principalmente a diferença nas condições da disposição física de objetos (especialmente automóveis estacionados), condições climáticas e até mesmo interferência de outros rádios nas imediações de onde se realizou o teste. Na figura 39 é apresentado o arranjo do *end-device* e *gateway* durante os testes do fluxo completo. As datas foram 29/11/2023 e 06/12/2023. Na primeira, o tempo estava levemente nublado, com poucos carros estacionados próximo aos sensores e, de acordo com os testes, com pouca interferência de outros sinais de rádios. Já na ultima, o tempo estava limpo, com bastante automóveis estacionados próximo ao local de teste e possivelmente com intensa interferência de outros rádios na mesma frequência que os sensores estavam operando para realizar a transmissão LoRaWAN.

Testes em 29/11/2023

Os dispositivos ESP32 foram colocados o mais próximo possível da disposição na qual se realizou a coleta dos dados CSI em via urbana (figura 35), para assegurar que o ambiente de teste seja o mesmo utilizado no treino do modelo de *Machine Learning*.

Primeiramente, constatou-se que os eventos enviados pelos sensores estavam sendo recebidos pelo banco de dados. Em seguida, os membros do projeto monitoraram a passagem de carros e pessoas, de forma a medir a acurácia da aplicação, tempo de resposta e perda de pacotes.

Após a ocorrência de eventos na via, verificou-se que a aplicação discerniu corretamente carros e pessoas na maior parte dos testes. Na figura 40 apresenta-se a matriz de

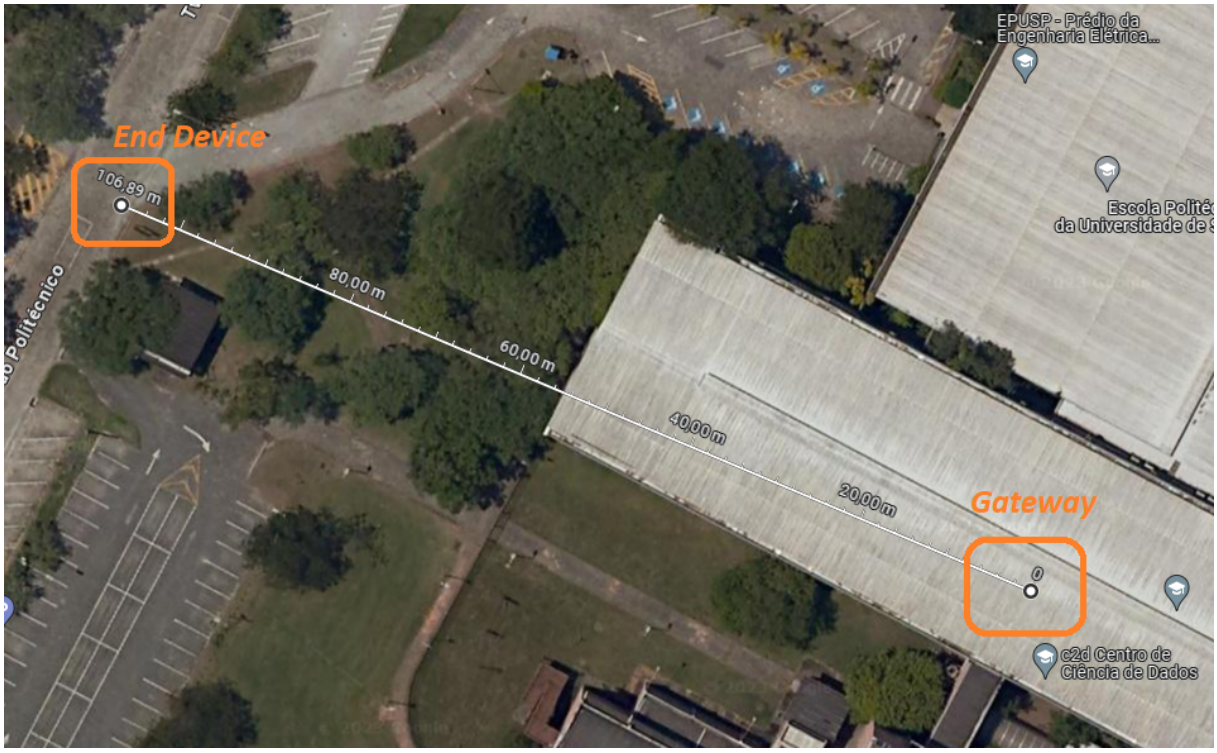

Figura 39 – Vista aérea da posição do *gateway* e *end-device* no cenário de teste do fluxo completo. A distância é de aproximadamente 100 metros.

confusão obtida. Todos os carros foram identificados corretamente, enquanto 1 pessoa foi classificada incorretamente, levando a uma acurácia de 95,5% na classificação.

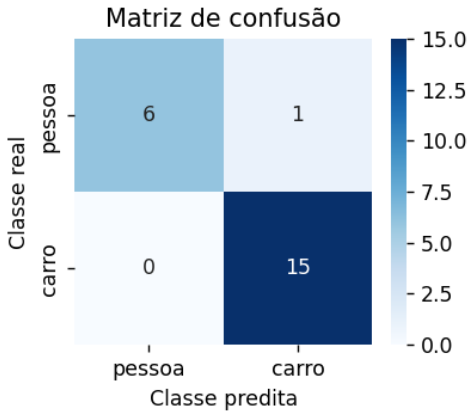

Figura 40 – Matriz de confusão obtida após teste prático da aplicação completa, em ambiente externo.

Para considerar um evento como detectado, foi utilizado uma janela de referência de 150 medições CSI e um limiar de queda de amplitude de 4,15%. Estes valores foram encontrados experimentalmente. Devido a influências do ambiente, como ruídos e carros passando atrás dos sensores, ocorreram detecções falsas de eventos, que foram classificadas como passagens de pessoa. Essa classificação, do ponto de vista do modelo de *Machine*

Learning, ocorre devido a estes eventos falsos não conterem objetos passando entre os sensores, o que leva a baixas perturbações na matriz CSI quando comparadas com passagem de pessoas e carros. Dessa forma, uma matriz CSI com poucas perturbações estará mais próxima de uma pessoa do que de um carro, já que o carro causa maior queda nas amplitudes.

Outro ponto analisado foi a perda de pacotes. Além dos 22 eventos de classificação recebidos corretamente no banco de dados, outros 3 eventos foram perdidos durante a etapa de transmissão LoRaWAN. Dessa forma, 12% dos eventos não foram recebidos pelo *gateway*. Essa taxa de perda está relacionada principalmente aos obstáculos entre os *end-devices* e o *gateway*, e ao uso do espectro eletromagnético por outro dispositivos alheios à aplicação.

Testou-se o alerta de perigo, configurado para que fosse enviado após 2 minutos em que uma entidade se mantivesse entre os sensores. Um dos membros da equipe se posicionou entre os sensores por 5 minutos, ocasionando o envio de 2 alertas para o banco de dados.

Entre o local de teste e o *gateway*, existem edifícios, paredes, árvores e outros obstáculos, o que afeta a transmissão da mensagem LoRaWAN. Além disso, foi constatado também que a taxa de perda de pacotes variou dependendo dia em que o sistema estava funcionando, o que pode estar relacionado à ocupação do canal de comunicação. A depender do dia, também foi verificado que a quantidade de pacotes alheios recebidos pela aplicação se alterava. Portanto, para uma solução comercial pode ser necessário alterar a banda de comunicação LoRaWAN do sistema no local de implantação de maneira dinâmica, a depender do uso do espectro por outros sistemas na região.

Também ocorreu o envio duplicado de 2 eventos. Como explicado na seção de construção dos sensores, estes enviam o evento de detecção e só passam a considerar novas quedas nas amplitudes como novos eventos assim que a amplitude das subportadoras suba acima do limiar estipulado. Entretanto, isto não foi suficiente para impedir a duplicação de eventos, tendo em vista que em alguns casos a amplitude subiu acima do limiar mesmo enquanto haviam entidades atravessando entre os sensores. Isso se deu provavelmente pelo fato de as antenas WiFi utilizadas não serem direcionais. Sendo assim, objetos atrás dos sensores poderiam causar uma reflexão do sinal, resultando num aumento da amplitude média do CSI medido entre os sensores. Uma abordagem que poderia ser considerada é descartar eventos que possuam *timestamps* muito próximos, na camada de aplicação ou a utilização de antenas unidirecionais de rádio WiFi.

A respeito do tempo de resposta, a medição do tempo exato é complexa, devido a atrasos ocorrerem em diversas etapas da solução completa. Entretanto, por meio de cronômetro, verificou-se que o tempo decorrido entre a ocorrência dos eventos e seu registro no banco de dados foi em torno de 5 segundos.

Estes resultados apresentados foram centralizados na tabela 3.

	Verdadeiro positivo	**Falso positivo**	**Evento falso**	**Pacote perdido**	**Duplicado**
Carro	15	0	0	0	0
Pessoa	6	1	3	3	1

Tabela 3 – Centralização de resultados de teste realizado no dia 29/11/2023.

Teste em 06/12/2023

Os dispositivos foram posicionados de maneira a replicar a mesma disposição tanto da coleta quanto do teste anterior. Apesar disso, os resultados foram divergentes. Na tabela 4 eles são apresentados. Nota-se que de 9 eventos de identificação, 8 foram perdidos, totalizando 89% de perda de mensagens. Além dos eventos de identificação, houve um evento de alerta de perigo que foi devidamente recebido pelo gateway. Este evento foi ocasionado por um carro que parou na linha do sensor para que um passageiro desembarcasse.

	Verdadeiro positivo	**Falso positivo**	**Evento falso**	**Mensagem perdida**	**Duplicado**
Carro	0	0		4	0
Pessoa	1	0		4	0

Tabela 4 – Centralização de resultados de teste realizado no dia 06/12/2023.

Para entender melhor o que estava causando a perda de mensagens, o *end-device* foi levado para o laboratório onde se encontrava o *gateway* e realizou-se eventos forçados para testar o envio de mensagens LoRaWAN para o *gateway* no caso de os dispositivos estarem em um raio de alcance menor (aproximadamente, 4 metros). Foram enviados 5 eventos de testes e todos foram recebidos pelo *gateway*. Sendo assim, possivelmente neste dia haviam outros elementos operando na mesma faixa de frequência que o *end-device*, 915 MHz, nas imediações de onde se realizou os testes. Além disso, também haviam diversos automóveis estacionados próximo aos sensores, o que também poderia vir a acarretar em uma perda maior de mensagens, porém não tão grande quanto a observada. Para remediar tal questão, seria possível a utilização do *Adaptive Data Rate*(ADR), mecanismo de otimização da tecnologia LoRaWAN capaz de alterar a largura de banda, SF e potencia de transmissão de maneira a garantir um menor consumo energético e a entrega das mensagens (ALLIANCE, 2017). Isto não foi implementado pois foge do escopo do projeto, tendo em vista que tornaria o programa que roda na ESP32 mais complexo.

5.2.3 Distância máxima entre *end-device* e *gateway*

Para avaliar o alcance da rede, configurou-se a ESP32 para utilizar fator de espalhamento espectral SF7 na transmissão LoRaWAN, que garante a maior taxa de transmissão, porém menor alcance, e aumentou-se gradualmente a distância entre *end-device* e *gateway*. Na figura 41 tem-se as posições de ambos onde ocorreu a distância máxima. O *gateway* estava localizado dentro do laboratório, enquanto o *end-device* foi movido para o exterior do prédio. Próximo aos 240 metros, houve gradual aumento na perda de pacotes, até que a transmissão foi interrompida.

É importante notar que entre os dois dispositivos havia diversos obstáculos, como várias camadas de parede, edifícios e árvores. Dessa forma, mesmo no cenário de menor alcance e com interferência no sinal, foi possível obter distância na ordem de centenas de metros. Assim, com valores de SF mais altos e linha de visada, em vias remotas, é possível garantir alcance ainda maior.

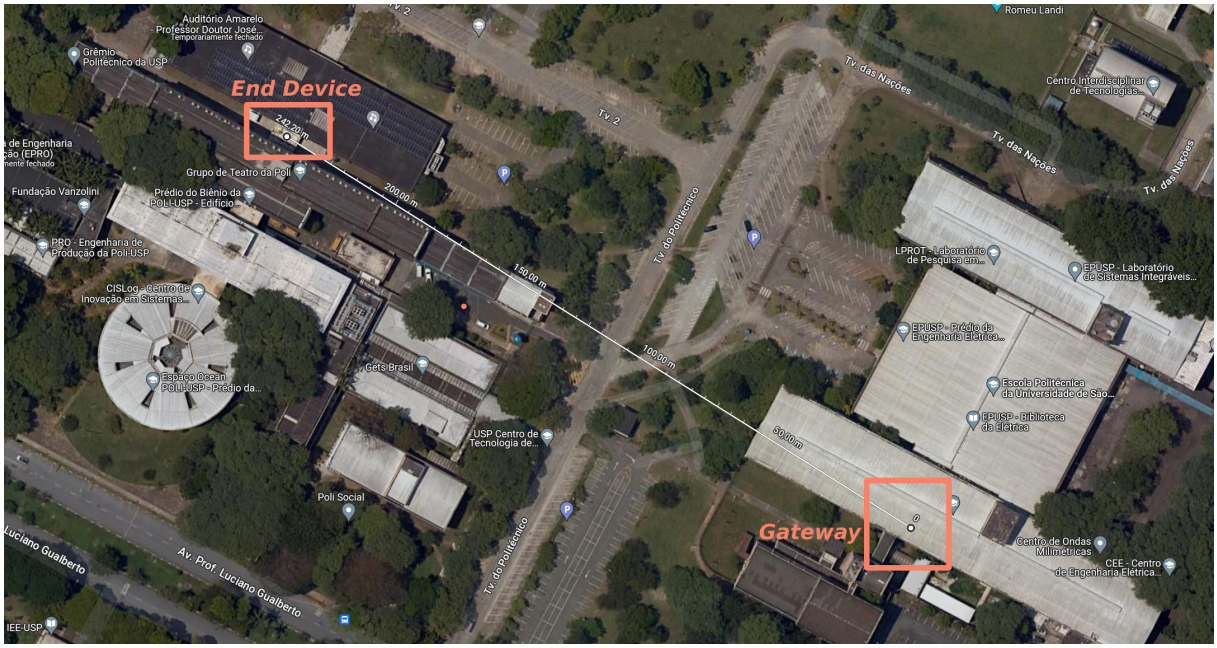

Figura 41 – Vista aérea da posição do *gateway* e *end-device* em seu alcance máximo no cenário de teste. A distância é de aproximadamente 240 metros.

6 Considerações Finais

6.1 Conclusões

O sistema de monitoramento de tráfego em vias remotas, baseado em sensoriamento WiFi e LPWAN, foi implementado de modo a atender os requisitos de baixo custo, baixo consumo energético e longo alcance de rede. Os dispositivos ESP32 são de baixo custo e baixo consumo energético, e podem ser implantados nas áreas remotas por meio do dimensionamento de um sistema de bateria e painel solar. A rede LoRaWAN é capaz de enviar os dados a longas distâncias mantendo o consumo baixo de energia. A computação em nuvem permite a identificação dos eventos e sua exibição em *dashboard* para análise. O sistema foi testado e avaliado, e foi comprovado seu funcionamento e viabilidade prática.

A aplicação é capaz de detectar e identificar a passagem de pessoas, carros, vacas e cachorros em vias. Também é possível identificar possíveis perigos, ao detectar objetos parados obstruindo a linha de medição dos sensores. Os eventos de detecção e de obstrução das vias são coletados e centralizados por meio da rede LoRaWAN, processados em nuvem e disponibilizados para análise.

6.1.1 Sensoriamento

Modelo em ambiente rural

A partir de dados coletados em ambiente rural, demonstrou-se possível o desenvolvimento de um modelo de *Machine Learning* capaz de identificar a passagem de entidades por meio de medições CSI do canal WiFi IEEE 802.11. O algoritmo KNN foi escolhido, no qual *features* extraídas das medições CSI são utilizadas para a classificação de carros, pessoas, vacas e cachorros. Além disso, as *features* foram selecionadas de modo a diminuir a quantidade necessária de dados para o funcionamento do modelo, o que viabilizou o envio das variáveis preditoras por meio da rede LPWAN, e por consequência a execução da identificação em nuvem.

A acurácia do modelo no *dataset* de teste foi de até 98%, e, por meio da matriz de confusão (figura 32), demonstrou-se a capacidade do modelo de discernir as classes. Além disso, é possível diminuir a quantidade de *features* enviadas, realizando um *tradeoff* entre quantidade de dados e performance da classificação.

Modelo em ambiente urbano

Foram coletados dados em ambiente urbano, correspondentes a passagens de pessoas e carros em uma via (35). A intenção foi a de retreinar o modelo, anteriormente ajustado para ambiente rural, para uma localidade em que fosse possível realizar testes práticos.

O KNN, ajustado para os novos dados, foi capaz de discernir as classes no conjunto de testes, obtendo 100% de acurácia na classificação de 10 passagens de carro e 6 passagens de pessoa, ao se utilizar 16 *features*.

Quando a aplicação de monitoramento foi avaliada em campo, ao utilizar-se do modelo para ambiente urbano, foi constatado uma acurácia de 95,5% para 22 observações. Somente uma passagem de pessoa foi classificada de forma incorreta. Dessa forma, comprovou-se na prática a validade da classificação feita pelo modelo. Em um segundo teste, em um outro dia com condições de ambiente diferente, houve uma perda grande de mensagens. Neste cenário, ao testar-se os sensores mais próximo ao *gateway*, as perdas foram mitigadas. Sendo assim, constatou-se uma possível interferência na largura de banda na qual o nó sensor estava operando.

6.1.2 Comunicação

Como forma de comunicação, o sistema adota a tecnologia LoRaWAN, de baixo consumo de energia e de longo alcance. Dessa forma, a solução de monitoramento pode ser implantada em vias onde há falta de infraestrutura elétrica e de rede, longe de centros urbanos e grandes rodovias. O baixo consumo de energia faz com que não seja necessário conexão com rede elétrica para transmissão dos dados. O longo alcance da rede permite centralizar os dados em servidores distantes dos pontos monitorados.

O uso de uma LPWAN para envio das informações medidas, impôs limitações ao projeto em relação à vazão de dados. Foi necessário desenvolver o modelo de identificação com variáveis preditoras que pudessem ser enviadas por meio da rede LoRaWAN, de forma que fosse possível se realizar a identificação dos eventos em nuvem.

6.1.3 Processamento de dados

O sistema desenvolvido adota *cloud computing* como infraestrutura para armazenamento de dados, execução do modelo de *Machine Learning* , hospedagem da *dashboard*. O uso da tecnologia de nuvem permite flexibilidade na alocação de recursos computacionais, em caso de necessidade de maior armazenamento ou capacidade de processamento. Para armazenamento de dados, foi adotado PostgreSQL, executando no serviço AWS RDS. O banco de dados relacional permite agregação dos dados para o desenvolvimento de análises. A execução do modelo de *Machine Learning* se dá no serviço AWS Lambda, que aloca os recursos necessários de computação e memória apenas na ocorrência de

eventos na aplicação. Dessa forma, na ausência de eventos de detecção, não há gastos em recursos computacionais na etapa de identificação. A *dashboard* foi construída utilizando a linguagem Python, por meio da biblioteca Ploty, e hospedada em uma instância virtual EC2 na AWS. O uso de Python permite acesso ao seu ecossistema de bibliotecas de manipulação e análise de dados, como a biblioteca Pandas.

6.2 Contribuições

Um dos aspectos centrais do projeto foi o desenvolvimento de um modelo de aprendizado de máquina que fosse capaz de identificar pessoas, carros e animais em vias, por meio de sensoriamento WiFi. Os resultados do projeto demonstraram que é possível a criação de tal modelo, por meio das medições das perturbações do CSI. Além disso, demonstrou-se a viabilidade de utilizar dispositivos IoT, como a ESP32, para a coleta de *features* necessárias para a execução do modelo, e transmití-las via LPWAN. Isto possibilita a implementação de sensores sem o uso de câmeras ou equipamentos de maior custo, e que também iriam necessitar de redes de maior vazão para seu devido funcionamento.

A principal contribuição do trabalho foi uma solução que atenda aos requisitos de vias remotas, já que utiliza equipamentos de baixo custo com alta independência da rede elétrica e com capacidade de envio de dados para grandes distâncias. Este sistema disponibiliza os dados coletados por meio de *dashboard*, permitindo que planejadores e autoridades tomem decisões informadas baseadas em dados de tráfego nas vias. Dessa forma, é possível a implementação de medidas de controle de tráfego e estratégias de segurança em áreas onde antes não era possível.

6.3 Perspectivas de Continuidade

No que diz respeito ao desenvolvimento futuro, identifica-se várias áreas para aprimoramento e expansão. Uma dessas áreas é a recalibragem dinâmica do sensor. Os sensores, ao serem ligados, calibram um valor de referência de amplitude das ondas subportadoras do WiFi, com o qual atenuações nas amplitudes são comparadas. Efeitos no ambiente, como movimentação de objetos ao redor dos sensores ou mudanças no tempo ou umidade relativa do ar, podem levar à alteração do valor real de referência, induzindo o sistema a falhas. Um mecanismo de recalibragem, portanto, aumentaria a confiabilidade e disponibilidade dos sensores.

Para detecção de eventos, é estabelecido nos sensores um limiar de queda de amplitude do sinal, e um tamanho de janela de medições CSI para calcular a amplitude média em cada instante. Estes dois parâmetros devem ser determinados empiricamente para o ambiente onde o sensor está sendo implantado. Um ponto de melhoria, seria o

desenvolvimento de métodos automáticos para determinar o limiar de queda de amplitude e a janela de tempo ideal em diferentes ambientes. Este desenvolvimento tem como objetivo reduzir a dependência de testes manuais, que pode ser um processo demorado e sujeito a erros. Automatiza-los melhoraria a eficiência na implementação do sistema em novos locais.

Por fim, destacasse a possível melhoria da modelagem de identificação, tanto em termos de acurácia quanto a de identificação de mais classes, como caminhões, ônibus e outros animais além de vacas e cachorros. Isto poderia ser feito por meio da coleta de novos dados, engenharia de novas *features*, e do uso de diferentes algoritmos de *Machine Learning*. A inclusão desses novos elementos visa ampliar a aplicabilidade do sistema em diferentes cenários, melhorando assim a coleta e análise de dados para uma variedade de aplicações.

Referências

ADELANTADO, F. et al. Understanding the limits of lorawan. *IEEE Communications magazine*, IEEE, v. 55, n. 9, p. 34–40, 2017. Citado 2 vezes nas páginas 12 e 32.

Agência Nacional de Telecomunicações (ANATEL). *Ato nº 14448*. 2017. <https://informacoes.anatel.gov.br/legislacao/atos-de-certificacao-de-produtos/2017/1139-ato-14448>. Accessed: date-of-access. Citado na página 30.

ALLIANCE, L. *LoRaWAN 1.1 Specification*. [S.l.]: LoRa Alliance, 2017. <https://hz137b.p3cdn1.secureserver.net/wp-content/uploads/2020/11/lorawantm_specification_-v1.1.pdf?time=1680293580>. Accessed: 2023-04-03. Citado 5 vezes nas páginas 5, 27, 29, 50 e 73.

ALLIANCE, L. *What is LoRaWAN Specification*. [S.l.]: LoRa Alliance, 2022. <https://lora-alliance.org/about-lorawan/>. Accessed: 2023-02-25. Citado 2 vezes nas páginas 5 e 28.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Citado na página 16.

BLISS, T.; RAFFO, V. Improving global road safety: Towards equitable and sustainable development, guidelines for country road safety engagement. World Bank, Washington, DC, 2013. Citado na página 11.

CHEN, X.; VENOSA, E.; HARRIS, F. Synchronization steps for low complexity chirp spread spectrum (css) receivers. In: IEEE. *MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM)*. [S.l.], 2021. p. 127–132. Citado 2 vezes nas páginas 26 e 27.

CHIRPSTACK. *Chirpstack Architecture*. [S.l.]: ChirpStack, 2023. <https://www.chirpstack.io/docs/architecture.html>. Accessed: 2023-02-25. Citado 2 vezes nas páginas 5 e 31.

COTRIM, J. R.; KLEINSCHMIDT, J. H. Lorawan mesh networks: A review and classification of multihop communication. *Sensors*, v. 20, n. 15, 2020. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/20/15/4273>. Citado na página 29.

CUNNINGHAM, P.; DELANY, S. J. k-nearest neighbour classifiers: 2nd edition (with python examples). *CoRR*, abs/2004.04523, 2020. Disponível em: <https://arxiv.org/abs/2004.04523>. Citado na página 59.

Espressif Systems. *ESP32 Datasheet*. [S.l.], 2017. Accessed: 2023-11-17. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Citado na página 44.

FreeRTOS. *xSemaphoreCreateBinary - FreeRTOS API*. 2023. Accessed: 2023-11-15. Disponível em: <https://www.freertos.org/xSemaphoreCreateBinary.html>. Citado na página 49.

HAXHIBEQIRI, J. et al. A survey of lorawan for iot: From technology to application. *Sensors*, Mdpi, v. 18, n. 11, p. 3995, 2018. Citado na página 31.

HERNANDEZ, S. M.; BULUT, E. Wifi sensing on the edge: Signal processing techniques and challenges for real-world systems. *IEEE Communications Surveys Tutorials*, v. 25, n. 1, p. 46–76, 2023. Citado 6 vezes nas páginas 12, 14, 17, 21, 22 e 40.

HO, T. K. Random decision forests. In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. [S.l.: s.n.], 1995. v. 1, p. 278–282 vol.1. Citado na página 64.

INFSO, D. Networked enterprise & rfid infso g. 2 micro & nanosystems, in co-operation with the working group rfid of the etp eposs, internet of things in 2020, roadmap for the future [r]. *Information Society and Media, Tech. Rep*, v. 10, 2008. Citado na página 16.

JAMES, G. et al. *An Introduction to Statistical Learning with Applications in R*. [S.l.]: Springer, 2021. Citado na página 64.

JIANG, J.-g. et al. Cs-dict: Accurate indoor localization with csi selective amplitude and phase based regularized dictionary learning. In: SPRINGER. *Algorithms and Architectures for Parallel Processing: 20th International Conference, ICA3PP 2020, New York City, NY, USA, October 2–4, 2020, Proceedings, Part II 20*. [S.l.], 2020. p. 677–689. Citado 2 vezes nas páginas 12 e 23.

KOOIJMAN, M. *Arduino-LMIC library*. [S.l.]: GitHub, 2023. <https://github.com/matthijskooijman/arduino-lmic>. Citado na página 48.

LEHMANN, M. et al. Accuracy and performance of the lattice boltzmann method with 64-bit, 32-bit, and customized 16-bit number formats. *Physical Review E*, v. 106, 07 2022. Citado na página 48.

LI, S.; XU, L. D.; ZHAO, S. The internet of things: a survey. *Information systems frontiers*, Springer, v. 17, p. 243–259, 2015. Citado 2 vezes nas páginas 16 e 17.

LIMA, T. F. de et al. Análise epidemiológica dos acidentes de trânsito no brasil. *Encontro de Extensão, Docência e Iniciação Científica (EEDIC)*, v. 5, n. 1, 2019. Citado na página 11.

MA, Y.; ZHOU, G.; WANG, S. Wifi sensing with channel state information: A survey. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 52, n. 3, p. 1–36, 2019. Citado na página 12.

MELEK, M.; MELEK, N.; KAYIKCIOGLU, T. A novel simple method to select optimal k in k-nearest neighbor classifier. *International Journal of Computer Science and Information Security,*, v. 15, p. 464–469, 12 2017. Citado na página 60.

PAULTRE, A. *How Open-Source is the LoRaWAN IoT Community?* [S.l.]: Embedded Computing Design, 2015. <https://embeddedcomputing.com/technology/open-source/linux-freertos-related/how-open-source-is-the-lorawan-iot-community>. Accessed: 2023-04-15. Citado na página 32.

PRIYANGA, M.; VIMALRAJ, S. L. S.; LYDIA, J. Energy aware multiuser & multi-hop hierarchical–based routing protocol for energy management in wsn-assisted iot. In: IEEE. *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*. [S.l.], 2018. p. 701–705. Citado na página 17.

QUETÉ, B. et al. Understanding the tradeoffs of lorawan for iot-based smart irrigation. In: IEEE. *2020 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*. [S.l.], 2020. p. 73–77. Citado na página 12.

RADCLIFFE, P. J. et al. Usability of lorawan technology in a central business district. In: IEEE. *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*. [S.l.], 2017. p. 1–5. Citado na página 12.

ROMAN, R.; LOPEZ, J. Integrating wireless sensor networks and the internet: a security analysis. *Internet Research*, Emerald Group Publishing Limited, 2009. Citado na página 17.

SEMTECH. *LoRa Modulation Basics AN1200.22*. [S.l.]: Semtech Corporation, 2015. Accessed: 2023-04-03. Citado 4 vezes nas páginas 5, 23, 25 e 30.

SEMTECH. *SX1276 Datasheet*. [S.l.], 2016. Accessed: 2023-11-17. Disponível em: <https://www.mouser.com/datasheet/2/761/sx1276-1278113.pdf>. Citado na página 44.

SHEN, J. et al. An efficient centroid-based routing protocol for energy management in wsn-assisted iot. *Ieee Access*, IEEE, v. 5, p. 18469–18479, 2017. Citado na página 17.

SCIKIT-LEARN. *Permutation Feature Importance*. [S.l.]: scikit-learn, 2023. <https://scikit-learn.org/stable/modules/permutation_importance.html#id2>. Accessed: 2023-08-01. Citado 2 vezes nas páginas 20 e 62.

SCIKIT-LEARN. *Preprocessing data*. [S.l.]: scikit-learn, 2023. <https://scikit-learn.org/stable/modules/preprocessing.html>. Accessed: 2023-08-01. Citado na página 19.

SCIKIT-LEARN. *Underfitting vs. Overfitting*. [S.l.]: scikit-learn, 2023. <https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html>. Accessed: 2023-08-01. Citado na página 20.

The Things Network. *Regional Parameters*. 2023. Accessed: 2023-11-12. Disponível em: <https://thethingsnetwork.org/docs/lorawan/regional-parameters/>. Citado na página 48.

VARSIER, N.; SCHWOERER, J. Capacity limits of lorawan technology for smart metering applications. In: IEEE. *2017 IEEE international conference on communications (ICC)*. [S.l.], 2017. p. 1–6. Citado na página 12.

VASSEUR, J.-P.; DUNKELS, A. Ip for smart objects. *White Paper*, IPSO Alliance, v. 1, p. 1–6, 2008. Citado na página 16.

VERMESAN, O.; FRIESS, P. *Internet of things applications-from research and innovation to market deployment*. [S.l.]: Taylor & Francis, 2014. Citado na página 16.

WIXTED, A. J. et al. Evaluation of lora and lorawan for wireless sensor networks. In: IEEE. *2016 IEEE SENSORS*. [S.l.], 2016. p. 1–3. Citado na página 32.

ZHOU, R. et al. Device-free localization based on csi fingerprints and deep neural networks. In: IEEE. *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. [S.l.], 2018. p. 1–9. Citado 2 vezes nas páginas 12 e 23.