JONG HWAN HONG, PEDRO PAULO TELES DE SOUZA, VICTOR SCAVAZIN DA SILVA SIQUEIRA

SmartBus: Aplicação de um Sistema Conexo por Redes Sem Fio para Rastreamento de Ônibus

São Paulo, SP

2023

JONG HWAN HONG, PEDRO PAULO TELES DE SOUZA, VICTOR SCAVAZIN DA SILVA SIQUEIRA

SmartBus: Aplicação de um Sistema Conexo por Redes Sem Fio para Rastreamento de Ônibus

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: Prof. Dr. Carlos Eduardo Cugnasca

São Paulo, SP

2023

Gerar a ficha catalográfica em https://www.poli.usp.br/bibliotecas/servicos/catalogacao-na-publicacao
Salvar o pdf e incluir na monografia

Resumo

Observa-se na atualidade um avanço tecnológico em tempo real em diversas áreas, incluindo a comunicação. O advento de tecnologias como 4G/5G tem impulsionado o crescimento de soluções para Internet das Coisas e Cidades Inteligentes, abordando demandas em diversas áreas. Essa tendência surge da crescente possibilidade de conectar 'coisas' à rede para monitoramento e controle remotos, visando a exploração de lucros diretos ou o desenvolvimento de aplicações que melhoram a qualidade de vida. O Sistema de Rastreamento de Ônibus é uma dessas aplicações de Smart City, já existente parcialmente no Brasil. Segundo levantamento da Associação Nacional das Empresas de Transportes Urbanos, 85.7% das pessoas que utilizam transporte coletivo apontaram o ônibus como o principal meio de locomoção. Esse dado ressalta a relevância do serviço de ônibus no Brasil, e ter acesso a informações sobre a localização dos veículos pode melhorar a qualidade de vida dos usuários, além de fornecer dados úteis para o estudo do trânsito local e do fluxo geral da cidade. O presente projeto propõe a exploração de conceitos de Internet das Coisas para implementar dispositivos de Sistema Embarcado nos pontos de ônibus, utilizando uma abordagem de Redes de Sensores Sem Fio. A motivação principal vem da necessidade identificada de um sistema conectado e inteligente, independente de Wi-Fi, para o agendamento de ônibus dentro da USP, visando planejar horários de forma mais eficiente e tentar amenizar o fluxo nos horários de pico.

Palavras-chave: IoT. Smart City. Rede de Sensores Sem Fio. Transporte coletivo.

Abstract

Currently, there is a real-time technological advancement in various areas, including communication. The emergence of technologies like 4G/5G has been driving the growth of solutions for the Internet of Things and Smart Cities, addressing demands in various fields. This trend arises from the growing possibility of connecting 'things' to the network for remote monitoring and control, aiming for direct profit exploitation or the development of applications that enhance the quality of life. The Bus Tracking System is one such application of a Smart City, already partially existing in Brazil. According to a survey by the NTU, 85.7% of people using public transportation identified the bus as their primary means of transportation. This data underscores the relevance of the bus service in Brazil, and having access to information about the location of vehicles can improve the quality of life for users, as well as provide useful data for the study of local traffic and the overall flow of the city. The present project proposes the exploration of Internet of Things concepts to implement Embedded System devices at bus stops, using a Wireless Sensor Networks approach. The main motivation comes from the identified need for a connected and intelligent system, independent of Wi-Fi, for bus scheduling within the USP (University of São Paulo), aiming to plan schedules more efficiently and try to alleviate congestion during peak hours.

Keywords: Iot. Smart City. Public Transportation.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Lista de abreviaturas e siglas

IoT	Internet of Things
ITS	Intelligent Transportation System
ATMS	Sistemas Avançados de Gerenciamento de Tráfego
ATIS	Sistemas Avançados de Informação ao Viajante
AVCS	Sistemas Avançados de Controle Veicular
ETC	Coleta Eletrônica de Pedágio
CVO	Operação de Veícuos Comerciais
NTU	Associação Nacional das Empresas de Transportes Urbanos
GPS	Global Positioning System
SE	Sistema Embarcado
USP	Universidade de São Paulo
BLE	Bluetooth Low Energy
RSSI	Received Signal Strength Indicator
CRC	Cyclic Redundancy error Check
LoRaWAN	Long Range Wide Area Network
SF	Spreading Factor
BW	Bandwidth
MQTT	Message Queuing Telemetry Transport
IDE	Integrated Development Environment
SPI	Serial Peripheral Interface
NTP	Network Time Protocol
RTC	Real Time Clock
MAC	Media Access Control
JSON	JavaScript Object Notation

Lista de símbolos

m	Metro
$Mbps$	Megabit por segundo

Sumário

1 Introdução

Neste capítulo será apresentado o contexto do projeto e as soluções existentes do problema proposto

1.1 Motivação

Em 2023, não é possível pensar num ambiente urbano sem trânsito. Em muitos países, os problemas de trânsitos são levantados em perspectivas tanto do meio ambiente quanto da saúde pública de uma forma alarmante, tamanho é o seu impacto na qualidade da vida das pessoas (PSICOLOGIA, 2000).

A mobilidade urbana é, de uma forma resumida, a capacidade de deslocamento de pessoas no espaço urbano para a realização de suas atividades cotidianas, num tempo considerado ideal, de modo confortável e seguro. Independente dessa atividade cotidiana a ser realizada, os indivíduos buscam o meio de transporte mais adequado de acordo com fatores como distâncias que terá que percorrer, do tempo a disposição para a locomoção, dos meios de transportes e vias de acesso disponíveis e do custo e qualidade deste deslocamento. "Pensar a mobilidade urbana é, portanto, pensar sobre como se organizam os fluxos na cidade e a melhor forma de garantir o acesso das pessoas ao que a cidade oferece, de modo mais eficiente em termos sócio-econômicos e ambientais."(VARGAS, 2008).

No cenário de transporte urbano brasileiro, com mais ênfase no chamado transporte público, as cidades ainda apresentam várias brechas para melhoria da qualidade, como a infraestrutura das vias, as condições dos veículos, a acessibilidade a deficientes, a segurança interna e nos pontos de embarque e desembarque, os congestionamentos no trânsito etc. (ARAÚJO et al., 2011). Em termos relacionados à disponibilidade de informações para os usuários do transporte público, percebe-se que os desafios são principalmente em torno da divulgação exata e em tempo real de informações sobre a frota, itinerários e horários dos ônibus. Isso inclui detalhes sobre horários e locais de maior fluxo de passageiros, linhas que operam em determinados pontos e terminais de ônibus, atrasos, congestionamentos e outros problemas semelhantes.

Neste cenário, é levantado então uma distribuição percentual de números de viagens, distâncias médias percorridas, e tempo médio gasto por modo de transporte.

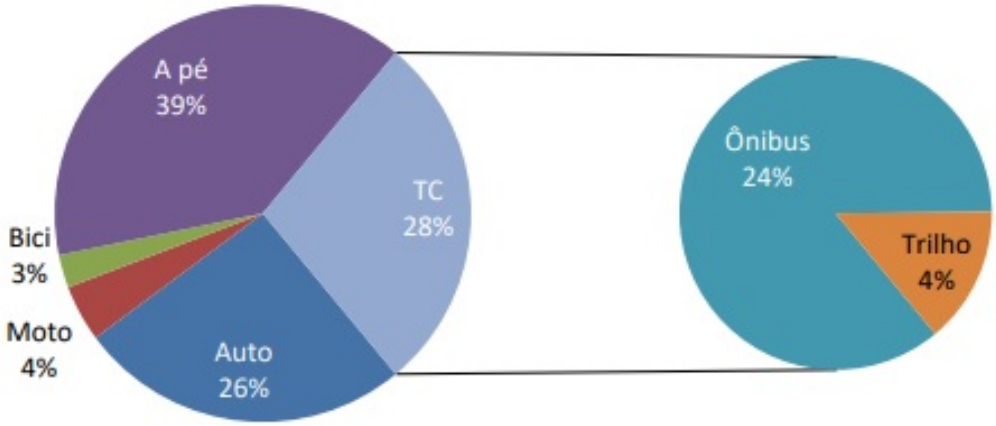


Figura 1 – Distribuição percentual de viagens por modo de transporte

Fonte: Extraído de (ANTP, 2018).

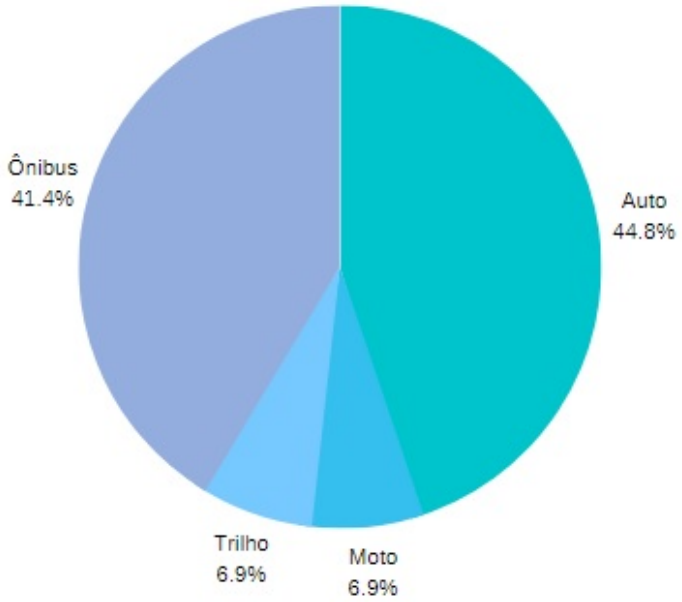


Figura 2 – Distribuição percentual de viagens por modo de transporte motorizados

Fonte: Adaptado de (ANTP, 2018).

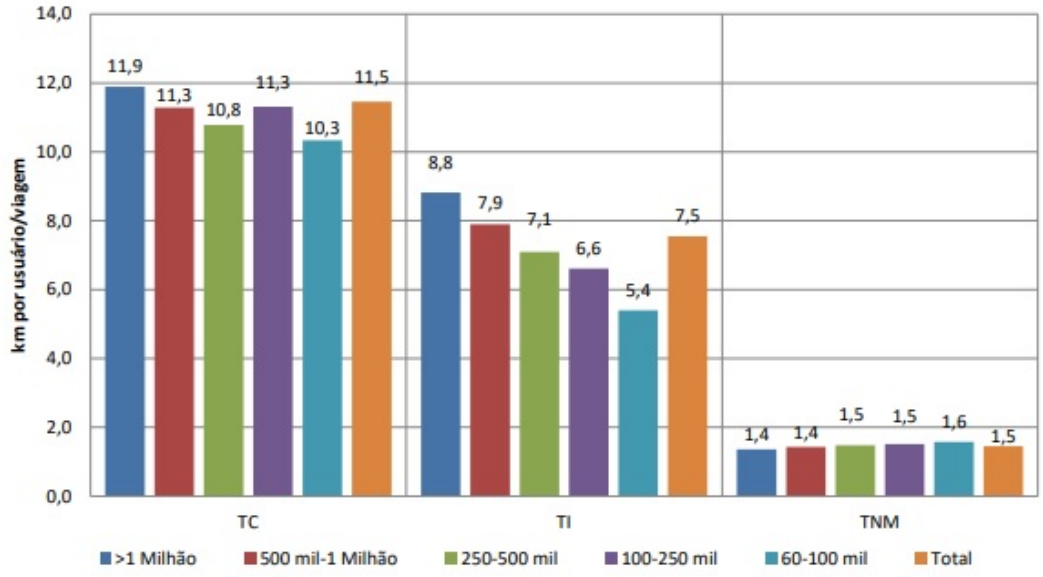


Figura 3 – Distâncias médias das viagens por modo de transporte e porte do município

Fonte: Extraído de (ANTP, 2018).

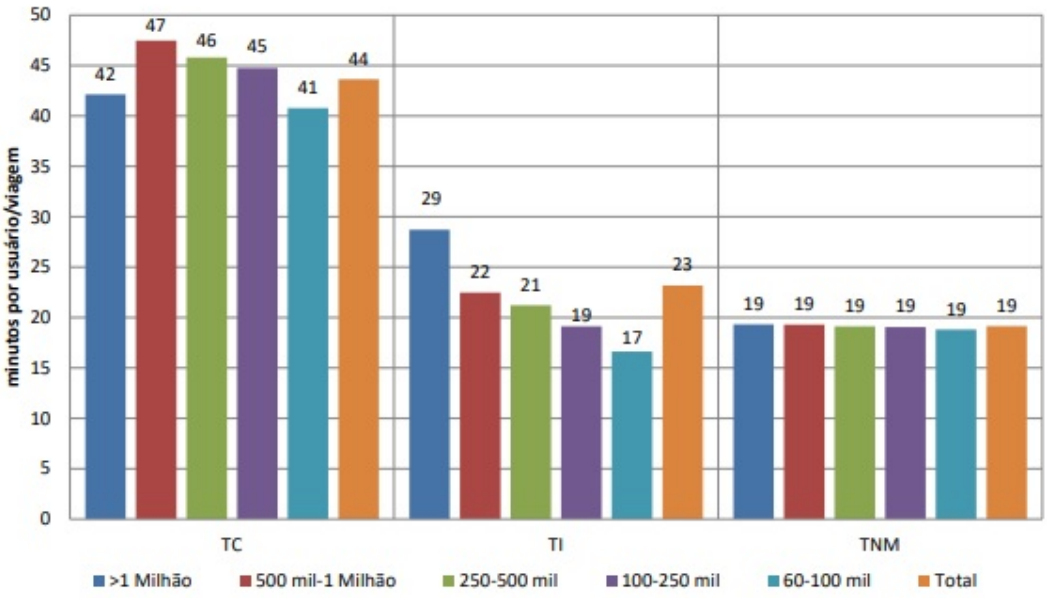


Figura 4 – Tempo médio de viagem por modo agregado e porte do município

Fonte: Extraído de (ANTP, 2018).

Com os gráficos acima percebe-se que o transporte coletivo (TC) ocupa uma grande parte das viagens urbanas das pessoas. Pesquisas feitas pela Confederação Nacional dos Dirigentes Lojistas (CNDL) apontam que o tempo gasto no trânsito por brasileiros que vivem nas capitais era, em média, de 147.9 minutos em 2017 antes da pandemia. Isso significa que em outras palavras, cerca de 15% do tempo útil de atividade do dia - faixa

de horário calculado a partir de pesquisas sobre tempo de sono recomendado, que seria de 7 a 8 horas para adultos (HIRSHKOWITZ et al., 2015) - é utilizado apenas para a locomoção das pessoas nas capitais no cotidiano.

1.2 Objetivos

O grande objetivo do projeto é conceber, implementar e validar um sistema completo para a identificação, agendamento e exposição de informações através de uma ou várias interfaces (app, tela, etc) para ônibus com a finalidade de promover uma melhoria na qualidade de vida dos usuários, mantendo requisitos de usabilidade e custos.

1.3 Justificativa

Segundo um artigo publicado na Revista de Economia Contemporanea, foi calculada uma perda total de 2.6% do PIB nacional (R$ 99 bilhões) no ano de 2010 com o tempo gasto em trânsito no Brasil. A pesquisa aponta também que caso as regiões metropolitanas do país perdessem o mesmo tempo que o interior de seus devidos estados, poderiam reduzir 27.6% (R$ 26.73 bilhões) das perdas totais, resultando numa perda total de 1.8% do PIB (VIANNA; YOUNG, 2015).

Além do capital direto perdido com o tempo gasto no trânsito, é importante ressaltar também que a falta de informação pode levar a desperdícios desnecessários, tanto de recursos materiais quanto de tempo. Nos pontos de ônibus, é comum encontrar painéis ou telas que contêm informações importantes para auxiliar o usuário a se deslocar na cidade de maneira mais independente, rápida e eficiente. Essas informações fazem parte de uma camada básica da infraestrutura do sistema de transporte urbano e têm como objetivo promover uma boa mobilidade na cidade. Segundo uma pesquisa que comparou sistemas de informação em paradas de ônibus, concluiu-se que o Brasil ainda apresenta um desempenho inferior em comparação a outros países (TAVARES et al., 2015).

O desenvolvimento de sistemas inteligentes em transportes (*Intelligent Transportation System*, ITS) já é uma realidade absoluta em países desenvolvidos. Os principais centros urbanos fazem uso das tecnologias mais avançadas para aprimorar o funcionamento das redes de transporte. O objetivo é simplificar cada vez mais a rotina de motoristas e usuários em geral com o auxílio na busca das melhores rotas, e também vários países já mostram um grande interesse na aplicação do ITS direcionado ao transporte público e aos usuários finais destes (SILVA, 2000).

1.4 Organização do Trabalho

Este trabalho apresenta 6 capítulos:

O capítulo 1 apresenta a introdução contendo a motivação mostrando o cenário do problema a ser trabalhado, o objetivo como uma proposta de solução do problema, e por fim a justificativa do trabalho realizado.

O capítulo 2 apresenta os conceitos empregados e o resumo da literatura.

O capítulo 3 apresenta a metodologia do trabalho adotada, os passos que serão tomados na frente etc.

O capítulo 4 apresenta a especificação da solução SmartBus, incluindo os requisitos funcionais e não-funcionais, as tecnologias utilizadas e os procedimentos dos testes.

O capítulo 5 apresenta um relatório da elaboração do produto, os passos que foram seguidos e os resultados dos testes realizados.

O capítulo 6 discute-se os resultados obtidos e as conclusões finais do trabalho

2 Aspectos Conceituais

O objetivo deste capítulo é levantar os principais conceitos que são relevantes sobre o tema a ser abordado. Os conceitos levantados são de referências externas de documentos acadêmicos, pesquisas e notícias, além dos conhecimentos dos próprios alunos obtidos internamente na USP.

2.1 IoT e Cidades Inteligentes

2.1.1 Introdução

A Internet das coisas (*Internet of Things*, IoT) desempenha um papel essencial impulsionando a integração de serviços por meio de múltiplas aplicações. Permitindo a comunicação de objetos do dia a dia através da internet. Uma arquitetura popular, Ilustrado na figura 5, é composta de três camadas (WANG et al., 2016).

1. Percepção, como sensores, atuadores e outros dispositivos inteligentes.
2. Rede, responsável pela comunicação entre as duas outras camadas, i.e. entre os dispositivo e o ser humano utilizando a aplicação.
3. Aplicação, objeto final, desde aplicativos a sistemas autônomos.

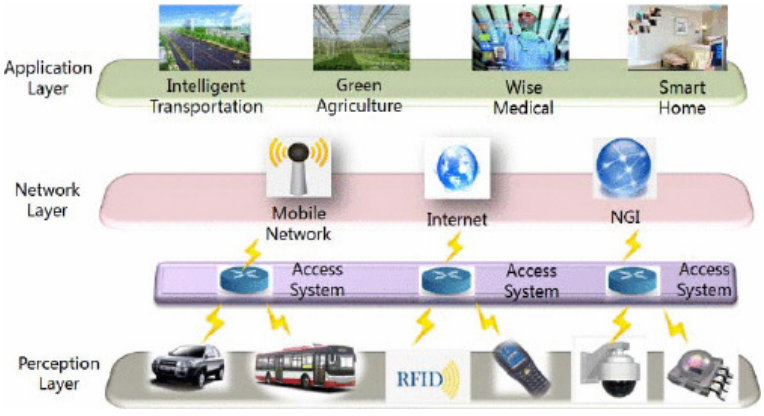


Figura 5 – Arquitetura de três camadas do IoT

Fonte: Extraído de (WANG et al., 2016).

O conceito de Cidades Inteligentes surgiu nos primórdios da década de 1990 com propósito de descrever o fenômeno do crescimento urbano que se baseia na tecnologia, inovação e globalização. Especialmente sob um perspectiva econômica (GIBSON; KOZMETSKY;

SMILOR, 1992). E no estudo realizado por (GIFFINGER et al., 2007), foi apresentado um modelo de referência para as Cidades Inteligentes, que são compreendidas como cidades que se caracterizam por seis elementos essenciais. Esses elementos são construídos por meio da combinação inteligente de recursos, autogerenciamento, participação ativa dos cidadãos e uma abordagem para avaliar e comparar a inteligência urbana. Os seis elementos fundamentais identificados pelos autores são: economia inteligente, cidadãos inteligentes, governança inteligente, mobilidade inteligente, ambiente inteligente e qualidade de vida aprimorada.

2.1.2 Estado da Arte

IoT: Atualmente, existem muitos sistemas e equipamentos de interface no mercado. No entanto, a maioria deles trabalha em ambiente especializado e só pode acessar um número muito limitado de dispositivos com interfaces especializadas. Dada a grande diversidade de dispositivos existentes, promover o acesso unificado para diversos dispositivos é a questão chave das aplicações IoT (WANG et al., 2016).

2.2 Sistema de Transporte Inteligente

2.2.1 Introdução

Desenvolvido a partir do conceito proposto pelos Estados Unidos no século vinte e motivado pela crescente demanda do avanço no setor de transporte, o sistema de transporte inteligente é capaz de gerar sistemas de controle de transportes eficientes e precisos, utilizando tecnologias avançadas de comunicação de dados, integrando informações, comunicações, computadores e outras tecnologias no campo do transporte, criando uma integração entre pessoas, vias e veículos (AN; LEE; SHIN, 2011).

Os Sistemas Inteligentes de Transportes estão presentes em diversos setores, variando suas tecnologias, podendo assim, serem categorizados em seis categorias (JENSEN, 1996).

- Sistemas Avançados de Gerenciamento de Tráfego (ATMS) - compreendem o gerenciamento global do tráfego. Empregam tecnologias em projetos que tentam reduzir o congestionamento das vias urbanas ou rurais e garantir segurança. Tecnologias avançadas são aplicadas em sistemas de sinalização (semáforos), segurança no trânsito e gerenciamento de congestionamentos e rotas.

- Sistemas Avançados de Informação ao Viajante (ATIS) - empregam tecnologias avançadas para melhor informar o viajante sobre a via, sobre as condições ambientais

e o trânsito. Incorporam o uso de sistemas de navegação e informação para garantir segurança ao motorista e para minimizar os congestionamentos.

- Sistemas Avançados de Controle Veicular (AVCS) - garantem melhoria na segurança viária, permitindo que os veículos auxiliem os motoristas (veículos inteligentes). Os veículos são equipados com tecnologias que permitem monitorar as condições de dirigibilidade e tomar medidas necessárias para evitar acidentes .

- Coleta Eletrônica de Pedágio (ETC) - utilizam tecnologias avançadas para prover os mais adequados e eficientes métodos de cobrança de pedágio, trabalhando para minimizar tempos perdidos e reduzir os congestionamentos.

- Operação de Veícuos Comerciais (CVO) - envolvem o gerenciamento e a operação de veículos comerciais. Empregam tecnologias para melhorar a gerência e o serviço dos transportes de carga e para minimizar as interferências com relação às rotas e aos tempos perdidos, procurando manter um alto nível de segurança. E devem ser projetados de forma a não onerar os custos do sistema como um todo.

- Sistemas Avançados de Transporte Púbico (APTS) - representam o uso de tecnologias avançadas para melhorar a segurança, eficiência e efetividade dos sistemas de transporte público. Os benefícios para os usuários incluem a minimização dos tempos de espera, segurança e facilidade para o pagamento da tarifa, bem como informações precisas e atualizadas sobre itinerários e horários.

  Neste trabalho o foco será apenas no APTS.

2.3 Conclusão/Relação

Considerando os tópicos abordados anteriormente, nota-se que as Cidades Inteligents utilizam os equipamentos IoT para implementar soluções inovadoras (TOPPETA, 2010). Com o aumento expressivo do uso de eletrônicos(como smartphones e tablets) integrados a redes de trocas de dados (como GPS, *Bluetooth* e Wi-Fi), a tecnologia passou a ser imprescindivel na sociedade (ATZORI; IERA; MORABITO, 2010). Dado que IoT pode ser compreendida como uma rede de dispostivos inteligados, trocando dados com diversos intuitos possíveis, como rastreametno, monitoramento, gerenciamento de rede e localização. Nesse contexto, os ITS's utilizam estes conceitos para desenvolver soluções especifícas de suas áreas.

3 Método do trabalho

Neste capítulo será apresentado brevemente a metodologia adotado para este projeto.

3.1 Metodologia

Sobre o conteúdo desenvolvido em si, foi decidido adotar uma metodologia de design *Top-Down* para Sistemas Embarcados (SE). Uma metodologia de *design* bem definido é importante por três motivos. Primeiramente, ajuda a manter uma tabela de desempenho atualizado sobre o andamento do projeto e garantir que estamos fazendo tudo que precisamos fazer. Em segundo lugar, aliviamos o trabalho grande e maçante ao separar o processo em etapas mais administráveis. E por último, uma metodologia bem definida facilita a comunicação entre os membros do projeto. Definindo o processo como um todo, fica mais fácil os membros da equipe saber o que precisam fazer, o que eles terão de outros membros e o que eles possuem em mão para realizar os próximos passos (WOLF, 2023).

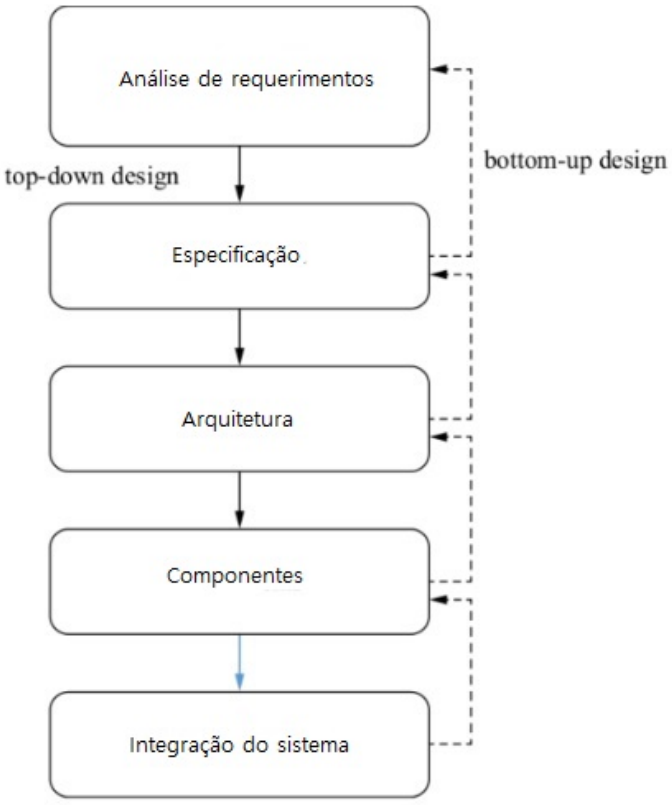


Figura 6 – Metodologia *top-down* (e *bottom-up*) para *design* de SE

Fonte: Adaptado de (WOLF, 2023).

1º Etapa - Proposta detalhada da solução.

2º Etapa - Levantamento dos requisitos funcionais e não-funcionais.

3º Etapa - Especificação das tecnologias e abordagens selecionadas para atender os requisitos levantados anteriormente.

4º Etapa - Levantamento da arquitetura geral do sistema, mostrando como cada parte levantada nas especificações vão formar o produto final.

5º Etapa - Informações dos componentes a serem utilizados, verificação se atende os requisitos levantados e testes unitários seguindo a especificação.

6º Etapa - Integração dos componentes respeitando a arquitetura definida, testes funcionais do produto final.

3.2 Prototipagem

Depois de seguir as etapas descritas, o objetivo é desenvolver um protótipo para conduzir testes e usá-lo como uma demonstração prática da solução proposta. O plano inicial é simular um ambiente real em menor escala para realizar uma prova de conceito.

O passo seguinte seria adaptar a solução para realizar testes no ambiente real da USP - Butantã, se possível. Esta fase será uma oportunidade para observar e analisar como o protótipo funciona em um cenário prático e, a partir disso, realizar ajustes e otimizações conforme necessário.

4 Concepção do Sistema SmartBus

Neste capítulo apresenta-se a concepção do SmartBus. Nele consta o levantamento da proposta detalhada, requisitos e a arquitetura do sistema.

4.1 Proposta Detalhada

Conforme comentado anteriormente, o objetivo deste projeto é implementar um Sistema Embarcado (SE) que registra a passagem dos ônibus nos pontos. Com essas informações, é gerado um registro dos ônibus que já passaram, bem como uma previsão de quando o próximo ônibus chegará. Para que isso seja possível, é necessário que os nós (os pontos de ônibus equipados com o SE) estejam interconectados e se comuniquem entre si. Estas informações serão então transmitidas para o mundo externo por meio de uma ou mais interfaces.

4.2 Requisitos Funcionais

4.2.1 Deteccção de Ônibus

O sistema tem que ser capaz de realizar a detecção da passagem dos veículos e identificá-los.

4.2.2 Processamento de Dados nos pontos

Um microcontrolador deverá ser capaz de realizar a leitura do veículo e extrair dados úteis como o tempo de chegada, ID do veículo e a linha. Tendo esses dados em mãos, será capaz de exibir informações relevantes para aquele ponto, e passar adiante a nós que precisam dessa informação.

4.2.3 Conexão da Rede

O SE deve ser capaz de realizar uma conexão entre os nós (pontos de ônibus) e um *gateway*, compartilhando os dados de tempo e posição dos ônibus coletados.

4.2.4 Servidor Central

Um nó denominado de sorvedouro (*gateway*) terá conexão com a internet. Por esse nó será passada as informações coletadas a uma nuvem (ou qualquer outro sistema centralizado) para processamento e análise adicional.

4.3 Requisitos Não Funcionais

4.3.1 Confiabilidade

O sistema deve garantir um certo nível de confiabilidade e disponibilidade. Nesse projeto, os pontos devem garantir uma leitura confiável dos veículos que passam por eles, as informações mostradas ao exterior devem ser corretas e os dados passados a outros nós devem ser adequados.

4.3.2 Escalabilidade

O sistema deve garantir escalabilidade para acomodar um crescimento no número de pontos e veículos sem afetar a sua performance.

4.3.3 Latência

O sistema deve manter uma latência baixa no processamento e transmissão de dados para garantir um serviço funcional de rastreamento dos ônibus de algo perto de "tempo real".

4.3.4 Eficiência energética

Como um SE qualquer, é necessário levar em consideração uma boa eficiência energética caso o sistema não esteja conectado a uma fonte de energia.

4.3.5 Segurança

O sistema deve garantir alguns serviços de segurança em relação a interferências externas. Listando alguns deles:

- Disponibilidade: além da disponibilidade do sistema em si, deve existir medidas de segurança para que um agente externo não consiga interferir no funcionamento dele.

- Integridade: o sistema deve garantir que as informações enviadas e mostradas para os usuários sejam consistentes e verídicas. Caso alguma alteração não autorizada seja realizada garantir que ela seja detectada.

- Autenticidade: deve haver uma garantia de que a informação recebida seja de um remetente confiável e conhecido.

4.3.6 Facilidade de manutenção

O sistema deve ser projetado de uma forma que a sua manutenção e controle sejam simples, de modo que uma atualização ou mudança tanto no Hardware quanto no Software seja possível quando necessário.

4.4 Especificação

4.4.1 Detecção de Passagem dos Ônibus via *Bluetooth Low Energy (BLE)*

BLE é uma tecnologia de comunicação sem fio projetada para aplicações de baixo consumo de energia. Representa uma versão de baixa potência do *Bluetooth* clássico e foi desenvolvida para permitir a comunicação entre dispositivos que requerem uma vida útil prolongada da bateria. Embora o BLE seja um constituinte do padrão Bluetooth, ele difere do Bluetooth clássico em vários aspectos fundamentais. Por exemplo, o BLE emprega um esquema de modulação distinto na camada física em contraste com o Bluetooth clássico. No entanto, o BLE compartilha algumas características e componentes com o Bluetooth clássico, como a camada L2CAP. (WOLF, 2023).

O *BLE* apresenta diversas vantagens em comparação ao Bluetooth Clássico, o que o torna mais "inteligente"ao oferecer funcionalidades aprimoradas para desenvolvedores, fabricantes e usuários, resultando em uma experiência melhor com a Internet das Coisas (IoT) fornecida pelo BLE. De acordo com a (BLUETOOTH SPECIAL INTEREST GROUP, 2010), essas vantagens tornam o BLE uma das tecnologias mais recomendadas para uso na IoT, conforme listado a seguir:

- Facilita a IoT ao oferecer suporte a várias opções de conexão, como *Internet Protocol version* 6 (IPv6), 6LoWPAN (IPv6 *over Low power Wireless Personal Area Networks*) e *Bluetooth Smart Gateways*.

- Possui baixo consumo de energia, permitindo que funcione por quase um ano com uma pequena célula de bateria do tamanho de uma moeda.

- Permite conexões múltiplas, onde um dispositivo BLE central, conhecido como BLE *master*, aceita, gerencia e controla as conexões de vários outros dispositivos, chamados de BLE *slaves*, que solicitam a conexão.

- Permite o envio de arquivos para vários dispositivos simultaneamente, graças à capacidade de enviar apenas mensagens pequenas, com taxa de transferência máxima de 1 Mbps (Megabit por segundo), o que permite o envio dessas mensagens relativamente pequenas em *broadcast* para todos os BLE slaves conectados.

- Capacidade de enviar pequenas quantidades de bytes em *broadcast* sem a necessidade de estabelecer uma conexão.

- Fornece informações dos sensores dos dispositivos que o utilizam, como porcentagem da bateria, eixo de rotação e indicador de força do sinal recebido, conhecido como RSSI (*Received Signal Strength Indicator*), por meio do serviço de resposta à mensagem em *broadcast* enviada.

- Apresenta baixo custo, com produtos com essa tecnologia disponíveis no mercado por menos de 10,00 dólares.

Um Beacon BLE é descrito como um dispositivo que emprega a Tecnologia BLE para transmitir uma mensagem em *broadcast* para todos os dispositivos dentro de um alcance de aproximadamente 100 metros, desde que possuam a função BLE ativada (PAPAPANAGIOTOU, 2015). Normalmente, esses dispositivos consistem em smartphones e tablets, que respondem a essa mensagem executando ação(s) relacionada(s) à mensagem transmitida. Na figura 7, tem-se um exemplo de como é estruturada uma mensagem enviada por um Beacon BLE:

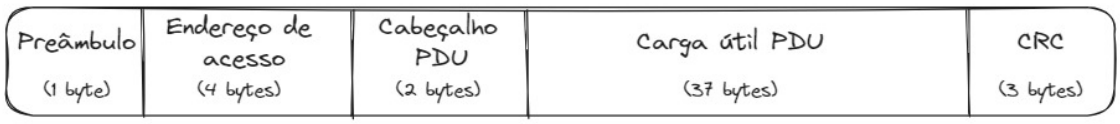


Figura 7 – Estrutura do pacote de uma mensagem BLE

Fonte: Adaptado de (BLUETOOTH SPECIAL INTEREST GROUP, 2010).

O tamanho de um pacote de mensagem BLE pode variar de 8 a 47 bytes, contendo os seguintes parâmetros (BLUETOOTH SPECIAL INTEREST GROUP, 2010):

- **Preâmbulo**: Cada pacote da camada de enlace inclui um preâmbulo, que desempenha o papel de sincronização de frequência, ajuste automático de ganho e estimativa de tempo de símbolo no receptor. O preâmbulo consiste em uma sequência constante de bits alternados entre 0 e 1 e é o primeiro a ser transmitido.

- **Endereço de acesso**: sempre será 0x8E89BED6 para as mensagens enviadas em *broadcast*

- **Cabeçalho PDU**: Carrega informações importantes sobre os dados que serão enviados, como o tipo e o comprimento.

- **Carga útil PDU**: Contém as informações desejadas (dados).

- **CRC(*Cyclic Redundancy error Check*)**: CRC - 24 bits - é utilizada para garantir a integridade do conteúdo do pacote. Se a verificação CRC detectar um erro na carga útil do pacote, o receptor não reconhecerá o pacote e o remetente retransmitirá o pacote para garantir uma transmissão confiável.

4.4.2 Rede LoRaWAN

Uma das partes essenciais do projeto é, sem dúvidas, a comunicação entre os nós. LoRa é uma tecnologia e protocolo de comunicação especializado para aplicações IoT, cuja suas principais características são:

- Como o nome implica, Longas Distâncias (*Long Range*).
- Baixo consumo de energia.
- Força de penetração de obstáculos relativamente alto (boa para ambientes *outdoors*).

Essas características foram os principais motivos da escolha deste protocolo para a solução. Mas como qualquer outra tecnologia, LoRa possui algumas desvantagens, e a ideia seria tentar contorná-los com estratégias de projetos mais adequadas. Algumas dessas desvantagens são:

- Baixa taxa de transmissão.
- Latência relativamente alta.
- Transmissão nem sempre garantida.
- Regulamentação de frequências.

Todas as desvantagens acima influenciam e são influenciadas pelo tamanho da mensagem a ser transmitida.

4.4.2.1 Arquitetura LoRa

Entrando um pouco mais no detalhe do contexto do projeto, foi verificado que os pontos de ônibus dentro do ambiente USP butantã tem uma distância média de 300m entre eles. Por esse motivo, a abordagem de uma rede descentralizada ad-hoc foi deixado de lado e LoRa foi escolhida. LoRa é uma tecnologia de rede caracterizada principalmente pelo seu grande alcance de alguns quilômetros, ele funciona de uma forma centralizada com a existência de um *Gateway* que serve como nó central, onde todos os nós periféricos se comunicam apenas com este Gateway, o que caracteriza a topologia estrela. Um nó, estando dentro do alcance deste *Gateway*, consegue realizar a conexão e fazer parte da

rede, essa característica garante uma grande escalabilidade sabendo que cada *Gateway* suporta até milhares de dispositivos finais e um alcance grande.

Figura 8 – Diagrama da arquitetura LoRa

Fonte: adaptado de (LoRa Alliance, 2020).

4.4.2.2 Consumo

A comunicação em LoRa é assíncrona, dados são transmitidos apenas quando os dados estão prontos a serem transmitidos seguindo um evento ou agendamento específico. Essa característica implica num grande aprimoramento no consumo energético, uma vez que a maioria do tempo o módulo LoRa se mantém "dormindo".

4.4.2.3 Alcance x Taxa de transmissão

Em projetos de redes LoRa, os principais pontos que precisam ser bem planejados são *Spreading Factor* (SF) e *Bandwidth* (BW). Eles são configurações em LoRa que causam influências diretas em fatores importantes como data de transmissão, sensibilidade e alcance.

De uma forma resumida, quanto maior o SF (e.g. SF12), maior será o alcance e sensibilidade, porém menor será a taxa de transmissão. E no caso do BW, quanto maior o BW, maior será a taxa de transmissão, e menor será o alcance.

A decisão dessas configurações são influenciadas por outros diversos fatores como o alcance desejado para este projeto em específico, mas principalmente a taxa de transmissão que vamos precisar. Ela, por sua vez, depende de outros fatores importantes que precisam ser bem planejados.

Tabela 1 – Tabela das caracteristicas do SF.

Spreading Factor (For UL at 125KHz)	Bit Rate	Range (Depends on terrain)	Time on Air for an 11-byte payload
SF10	980 bps	8 km	371 ms
SF9	1760 bps	6 km	185 ms
SF8	3125 bps	4 km	103 ms
SF7	5470 bps	2 km	61 ms

Fonte: adaptado de (Semtech Corporation, 2022).

4.4.2.4 Payload da mensagem

Um dos principais fatores a ser considerado para a taxa de transferência é o payload (tamanho da mensagem) que será transmitida.

Primeiramente, é necessário listar as principais informações que serão transmitidas:

- ID do veículo: os veículos precisam ter um identificador único. O número de bytes necessário para este problema depende de quantos veículos vão ser implementados no sistema. De uma forma simplificada, cada byte consegue representar 256 IDs diferentes.

- Timestamp: quando um veículo é encontrado, precisamos colocar o tempo exato de quando ele chegou. Para isso, o Unix Timestamp pode ser utilizado (número de segundos a partir de 1970-01-01 00:00:00 UTC), dependendo da sua precisão, ocuparia até 4 bytes.

- ID do ponto: do mesmo jeito que identificamos os veículos, os pontos de ônibus também precisam ser identificados.

- dados adicionais: outros dados podem ser considerados dependendo das necessidades.

Quanto menor o payload da mensagem, maior poderá ser o alcance da rede e maior poderá ser a taxa de transmissão, além de possibilitar soluções de segurança e confiabilidade (garantia de entrega).

Aumentar a taxa de transmissão é uma tarefa extremamente importante pois procura-se um sistema mais “em tempo real” possível, o que entra como uma desvantagem da rede LoRa (baixa latência). Esse problema de atraso de informação ocorre uma parte por causa da baixa taxa de transmissão em si, mas também por causa de regulamentações como o *Duty Cycle*.

4.4.2.5 *Duty Cycle*

Duty cycle é uma regra que limita o tempo que um dispositivo ocupa uma certa faixa de frequência na comunicação, por exemplo, se for um duty cycle de 1% e um dispositivo transmitiu uma mensagem por 1 segundo (*Time On Air* : 1s), a próxima transmissão só poderá ser realizada daqui a 100 segundos. Portanto, é importante reduzir o tempo de transmissão dos nós para que as próximas comunicações sejam realizadas sem atrasos. Essas regulamentações dependem de cada organização responsável do país.

4.5 Arquitetura do sistema

A arquitetura do sistema segue o padrão de uma rede centralizada com topologia de estrela, limitação vinda da tecnologia LoRaWAN que foi escolhida como o protocolo de comunicação dos nós. Os nós (ou dispositivos finais) são os próprios sistemas embarcados localizados nos pontos de ônibus que irão atender os requisitos funcionais (4.2) 1 a 3 colocados anteriormente. Os pontos estarão conectados por um nó central denominado *Gateway*, que por sua vez, estará conectado à internet para serviços de nuvens e outros.

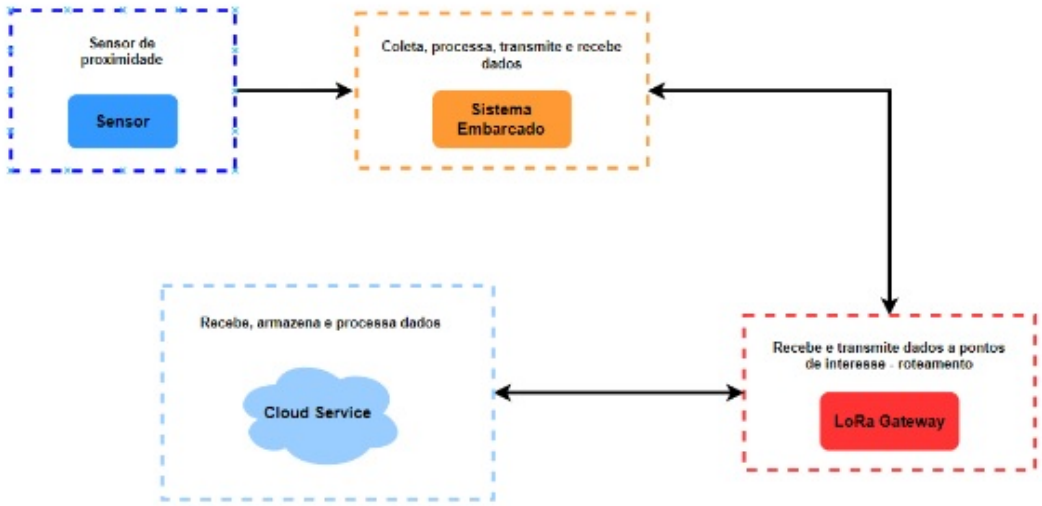


Figura 9 – Diagrama da arquitetura geral do sistema SmartBus

Fonte: autores.

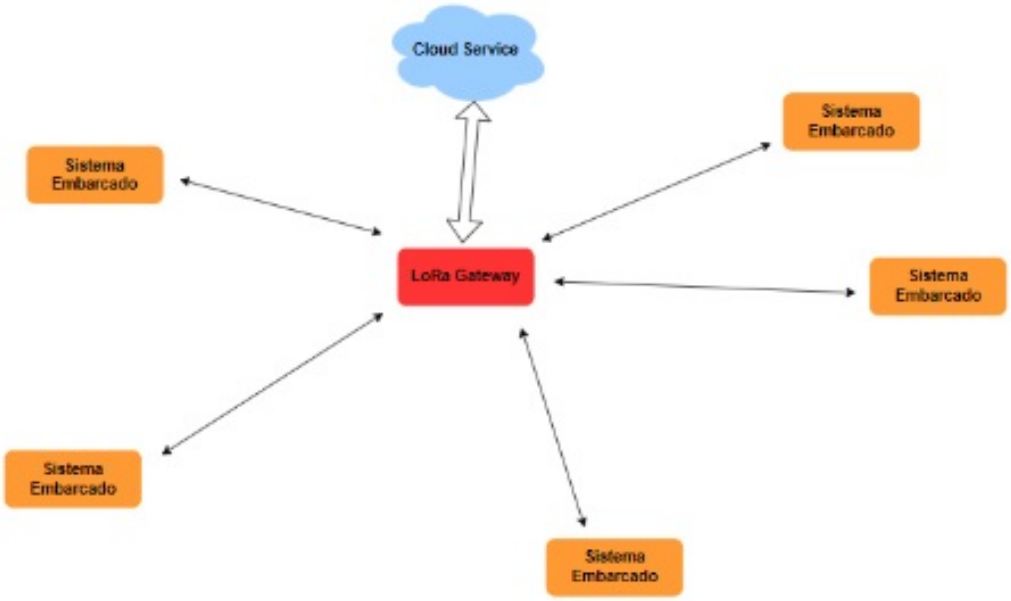


Figura 10 – Diagrama simplificado da topologia rede LoRa

Fonte: autores.

5 Desenvolvimento do Trabalho

Nesta seção são apresentados os experimentos/testes realizados e os resultados obtidos, a fim de validar as soluções propostas para o sistema.

5.1 Experimentos Realizados em Laboratório

Com os módulos ESP32 disponíveis, foram realizados experimentos visando avaliar cada componente de software e hardware.

5.1.1 Etapas do Experimento

Foram realizadas algumas etapas para realização dos experimentos, acrescentando em cada uma, uma nova rotina, um novo módulo de hardware ou um novo aplicativo, conforme arquitetura da figura 11.

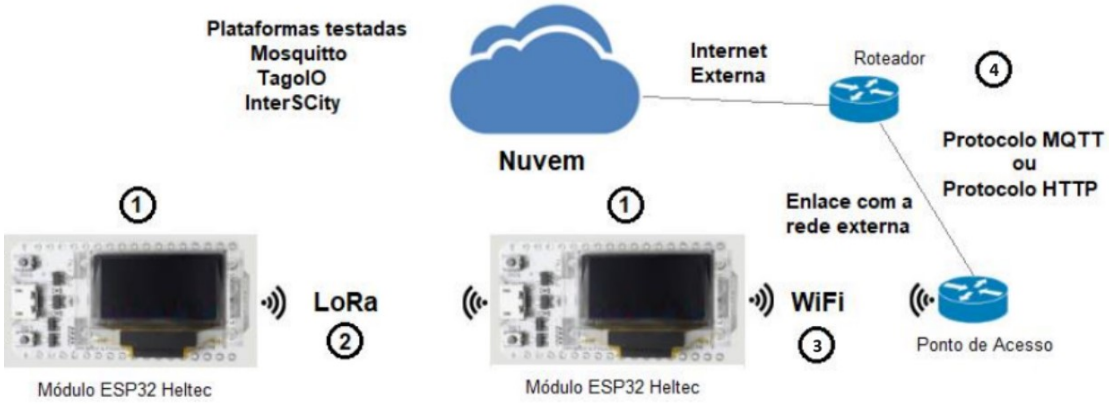


Figura 11 – Arquitetura utilizada para experimento em laboratório

Fonte: autores.

- 1ª etapa: Os módulos ESP32 da Heltec foram preparados e programados, incluindo um teste das funcionalidades do display integrado nesses controladores. Isso permitirá que os resultados dos experimentos sejam visualizados em tempo real.
- 2ª etapa: Dois módulos ESP32 da Heltec foram configurados e programados para estabelecer comunicação usando a tecnologia LoRa na camada física. Um deles atua como transmissor (emissor) e o outro como receptor. A transferência de dados ocorre por meio de pacotes de informação. As mensagens exibidas em seus displays indicam a confirmação de que as operações estão ocorrendo sem problemas.

- 3ª etapa: Após verificar que a transferência de pacotes de dados pelo link LoRa está funcionando corretamente, foi configurada e programada a conexão Wi-Fi disponível no módulo ESP32, permitindo que os dados sejam enviados para uma rede externa da Internet por meio de um ponto de acesso.

- 4ª etapa: Após estabelecer a conexão Wi-Fi com uma rede externa à Internet, foram criadas, configuradas e programadas mensagens contendo os dados recebidos no segundo módulo ESP32. Essas mensagens foram destinadas a serem enviadas em utilizando o protocolo Message Queuing Telemetry Transport (MQTT). Com a troca de mensagens ocorrendo conforme planejado, foram realizados testes em uma plataforma Cloud, TagoIO.

5.1.2 Experimento da Validação do Módulo ESP32

A seguir são apresentados os experimentos realizados para a validação do funcionamento do módulo ESP32-LoRa.

5.1.2.1 Comunicação LoRa entre módulos

O módulo ESP32 Heltec utiliza a plataforma ArduinoIDE para os seus programas, assim o seu código é escrito na linguagem C/C++. Começamos incluindo a principal biblioteca "#include <LoRa.h>"para a comunição LoRa com o módulo, em seguida são definidas algumas variáveis para a configuração inicial da comunicação *Serial Peripheral Interface* (SPI) e LoRa, e seus respectivos significados:

```
/* Definicoes para comunicação com radio LoRa */
#define SCK_LORA          5
#define MISO_LORA         19
#define MOSI_LORA         27
#define RESET_PIN_LORA    14
#define SS_PIN_LORA       18

#define HIGH_GAIN_LORA    20  /* dBm */
#define BAND              915E6  /* 915MHz de frequencia */
```

Figura 12 – Definição de variáveis para configuração inicial LoRa

Fonte: autores.

Quadro 1 – Exemplo de quadro

Variável	Descrição
SCK	Pino para o "*Serial Clock*"do SPI
MISO	Pino para o "*Master In, Slave Out*", envio de dados do escravo para mestre
MOSI	Pino para o "*Master Out, Slave In*", envio de dados do mestre para o escravo
RESET_PIN	Pino para o sinal de reset do LoRa
SS_PIN	Pino de seleção de escravo (*Slave Select*), usado para ativar ou desativar um dispositivo no barramento SPI
HIGH_GAIN	Potência de saída, ganho em dBm
BAND	Frequência de operação do módulo LoRa

Fonte: Autor.

5.1.2.2 Estrutura de dados para comunicação entre módulos

A definição dos dados mínimos que precisam ser trafegados para alimentar o sistema, deu origem ao seguinte diagrama de classes:

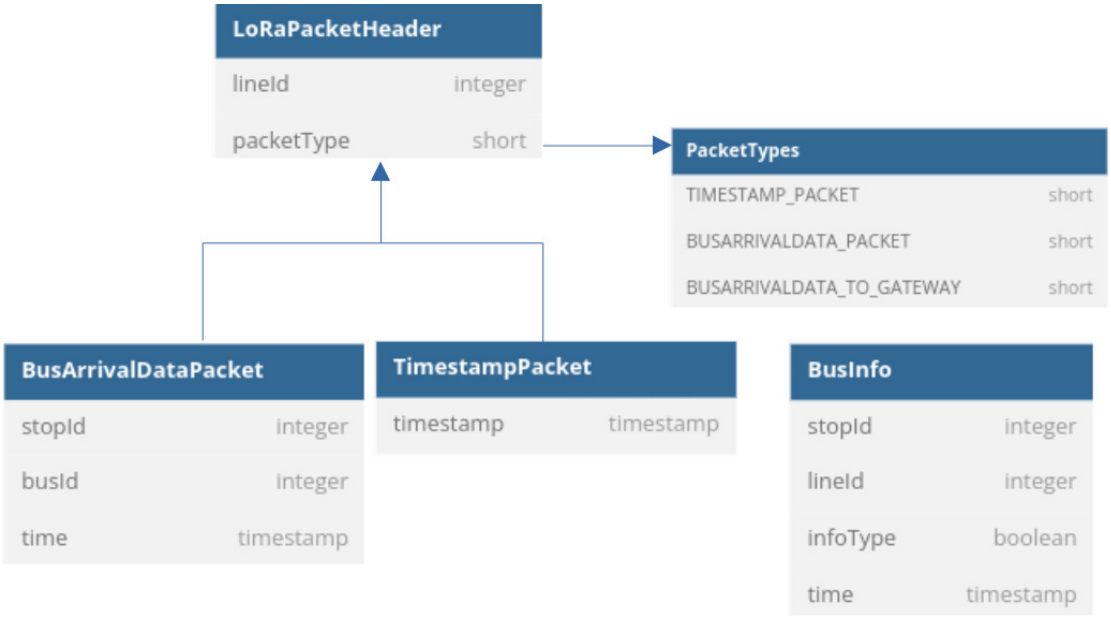


Figura 13 – Diagrama de Classes

Fonte: autores.

A lógica de comunicação da rede foi definida de forma que cada pacote transmitido possua duas principais partes: *header* e *packet*. Desse modo, cada mensagem contém essas duas informações e cada entidade presente na rede processa os dados de acordo com o tipo de dado recebido. Todos os pacotes tem o *header* como uma classe comum, ele serve como um primeiro identificador para verificar quais pontos vão processar o pacote. Sabendo que a comunicação LoRa é realizado através de um *broadcast*, o módulo identifica um pacote e

decide se irá processá-lo ou não dependendo do valor do lineId.

O packetType por sua vez serve para identificar qual é o método que irá tratar o dado, possibilitando assim que múltiplos tipos de dados sejam devidamente reconhecidos e processados. A linha horizontal indica quais os possíveis valores do packetType, isto é, quais são os tipos de pacotes trafegados dentro da rede, sendo:

- TIMESTAMP_PACKET - pacote de configuração dos nós, responsável por sincronizar o temporizador dos módulos da rede.
- BUSARRIVALDATA_PACKET - pacote gerado a partir de um evento (chegada de ônibus no ponto) no nó, o qual será enviado para o gateway que fará o broadcast deste pacote.
- BUSARRIVALDATA_TO_GATEWAY - pacote de dados de chegada de ônibus enviado apenas para o Gateway.

A seta vertical indica a herança das classes, isto é, BusArrivalDataPacket e TimestampPacket herdam de LoRaPacketHeader. Por último a classe BusInfo, foi feita unicamente para facilitar o print dos dados finais (úteis para o usuário final).

A imagem abaixo explicita o processo que ocorre na chegada de um ônibus.

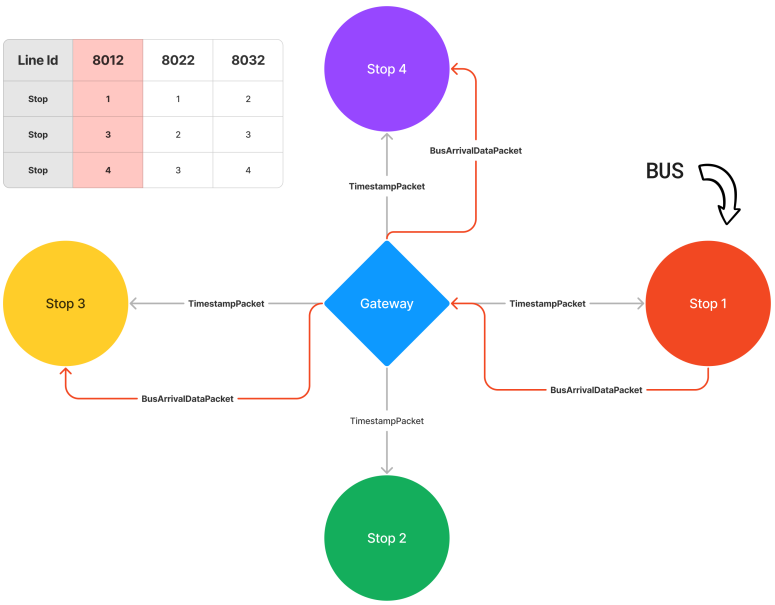


Figura 14 – Fluxograma

Fonte: autores.

No exemplo, temos um diagrama com um gateway central e 4 nós (paradas de ônibus). A tabela descreve três linhas exemplo e, em destaque, temos a linha 8012 cujo trajeto é representado paradas 1, 3 e 4. Inicialmente, o gateway envia o pacote de configuração dos nós (TIMESTAMP_PACKET). Em seguida, um ônibus passa pela parada 1. O nó responsável envia os dados referentes a chegada do ônibus para o gateway, que repassa para os demais nós da linha (3 e 4).

5.1.2.3 Configuração do Timestamp Unix e RTC

A obtenção da informação do horário atual no projeto é um passo crucial. No caso do *gateway* que possui conexão com a internet a princípio, consegue obter o tempo atual preciso pelo protolo NTP. Já os nós não conseguem sincronizar o seu tempo interno com o tempo real preciso da internet, logo surge uma nova tarefa da parte do *gateway* de enviar dado de *timestamp* em formato de uint32 com a finalidade de sincronizar os nós. O nó por sua vez precisa utilizar essa configuração para acionar o contador interno a partir de um tempo correto. Vale ressaltar também que o módulo ESP32 possui um *Real Time Clock* (RTC) interno, responsável por realizar contar o tempo a partir da configuração recebida. O formato para a obtenção do horário utilizado foi o Unix, que de uma forma simplificada, funciona contando cada segundo (ou milisegundo dependendo da precisão) a partir do dia 1 de janeiro de 1970 - 0h 0m 0s. Assim, considerando que o RTC de todos dispositivos do mundo estejam sincronizados, o tempo que estes dispositivos vão mostrar sempre serão iguais.

5.1.3 Comunicação com Beacon BLE

A leitura BLE em si tem um funcionamento extremamente simples. O módulo BLE faz um *scan* de dispositivos BLE que estão realizando um *advertising*, o que significa que este dispositivo, no nosso caso o *beacon*, está constantemente "anunciando"a sua presença ao redor. Cada módulo deve conhecer o identificador do *beacon* do ônibus que irá passar por ele, estes identificadores ficarão salvos numa lista dentro de cada nó. Quando um ônibus passar no ponto a uma certa distância, o *software* registra a passagem do veículo.

5.1.3.1 Considerações e experimentos sobre o nível RSSI

Além do identificador do *beacon* outra caracterísca que deve ser considerado é o valor do *Received Signal Strength Indication* (RSSI). De uma forma simplificada, RSSI é uma medida usada para estimar a potência do sinal recebido em comunicações sem fio. No caso do BLE *beacon* ele pode ser utilizado para conseguir uma estimativa da proximidade relativa. No escopo do projeto, a configuração do nível RSSI é interessante devido à limitação de distância entre o ônibus e o seu ponto, uma vez que um veículo do outro lado da rua, por exemplo, não deve ser lido. Essa lógica está brevemente descrita no

```
class MyAdvertisedDeviceCallbacks: public BLEAdvertisedDeviceCallbacks {
    void onResult(BLEAdvertisedDevice advertisedDevice) {
      String dispositivosEncontrados = advertisedDevice.getAddress().toString().c_str();
      Serial.println(dispositivosEncontrados);
      if (dispositivosEncontrados == dispositivosAutorizados
                  /*&& advertisedDevice.getRSSI() > nivelRSSI*/) {
        dispositivoPresente = true;
        Serial.println(dispositivoPresente);
      } else {

      }
    }
};
```

Figura 15 – Função de leitura BLE

Fonte: autores.

código observado na figura 15. A distância entre os dispositivos pode ser calculada de uma forma aproximada pela fórmula (JIANYONG et al., 2014):

$$P = A - 10 * n * \log_{10}(d)$$

Aqui, P representa a potência em que o sinal é recebido quando o dispositivo se localiza a d de distância; A representa a potência quando o dispositivo se localiza a 1m de distância e n representa o fator de atenuação do ambiente. Trocando as variáveis de lugar, chegamos na seguinte fórmula para distância:

$$d = 10^{((A-P)/(10*n))}$$

Vale ressaltar que a fórmula acima da um resultado aproximado da distância, além do fato que num caso real existem muitos outros fatores que baixam ainda mais a precisão da fórmula, logo ela serve apenas para conseguir a grandeza do valor aproximado de RSSI que queremos testar.

5.1.3.2 Considerações e experimentos sobre o tempo de leitura

Um ponto importante a considerar na leitura BLE é o tempo de *scan* ótimo para a aplicação. Essa consideração foi levantada devido ao modo que a leitura BLE funciona. Quando realiza-se a leitura BLE no módulo, todo o seu tempo de processamento fica voltada apenas para a leitura. O que significa que outros processos assíncronos ficam prejudicados durante a leitura. A solução para este problema foi realizar a leitura por um intervalo pequeno o suficiente para que outros processos assíncronos, como o recebimento de pacotes LoRa, não sejam prejudicados. Mas que também não seja pequeno ao ponto da leitura falhar em alguns casos.

5.2 Projeto e Implementação

Descrever os resultados do projeto e da implementação do trabalho, com justificativas das decisões tomadas.

Após a validação de cada componente, foi possível o desenvolvimento da implementação e integração do projeto.

5.2.1 Implementação dos Dispositivos LoRa

A implantação da rede LoRa consistiu em fazer upload dos códigos desenvolvidos para os nós nos seus respectivos módulos ESP32, cada um representando uma parada de ônibus. É importante destacar que o sistema foi concebido visando escalabilidade, ou seja, aumentar o número de pontos da rede (ou acrescentar novas linhas de ônibus) requer somente a adição dos endereços dos novos veículos identificadores das novas linhas, de modo que a rede se adapta automaticamente.

5.2.1.1 Configurações e Funcionamento dos nós

O desenvolvimento do nó foi realizado em volta de 3 problemas principais, a leitura do beacon, envio de dados ao Gateway, e recebimento de dados do Gateway.

Como comentado anteriormente, a leitura do *beacon* precisa de uma atenção extra devido ao seu caráter bloqueante durante a leitura. Durantes os testes, utilizamos uma janela de 100 mili segundos entre leituras BLE, o que resultou em um tempo bom para ambas as operações de comunicação BLE e LoRa. Para criar este intervalo de leitura, foi utilizado o método **millis()** do contador interno do módulo, junto com a seguinte condição:

$$if(currentMillis - previousMillis >= interval)$$

Sendo o *currentMillis* o tempo atual medido do contador, *previousMillis* o tempo medido na última vez que o código dentro do *if* foi acionado, e intervalo o tempo de *delay* desejado. Assim, cria-se um atraso particular para uma certa parte do código sem bloquear o restante do *main loop*.

Para o controle da distância de leitura do *beacon*, foi realizado um experimento de leitura do valor RSSI captado pelos módulos dos nós, relacionando com a distância do momento da medição, e foram obtidos as seguintes leituras:

```
Prediction Time: 7 : 25 : 50
ff:ff:c1:0e:1e:60
-80
Onibus chegou!
60ff:ff:c1:0e:1e:60
-77
Onibus chegou!
60ff:ff:c1:0e:1e:60
-71
Onibus chegou!
60ff:ff:c1:0e:1e:60
-71
Onibus chegou!
60ff:ff:c1:0e:1e:60
-71
Onibus chegou!
60ff:ff:c1:0e:1e:60
-91
Onibus chegou!
60ff:ff:c1:0e:1e:60
-77
Onibus chegou!
60ff:ff:c1:0e:1e:60
-77
Onibus chegou!
60ff:ff:c1:0e:1e:60
-80
Onibus chegou!
60ff:ff:c1:0e:1e:60
-80
Onibus chegou!
```

Figura 16 – Nível RSSI a 1m de distância

```
Onibus chegou!
60ff:ff:c1:0e:1e:60
-93
Onibus chegou!
60ff:ff:c1:0e:1e:60
-87
Onibus chegou!
60ff:ff:c1:0e:1e:60
-87
Onibus chegou!
60ff:ff:c1:0e:1e:60
-90
Onibus chegou!
60ff:ff:c1:0e:1e:60
-91
Onibus chegou!
60ff:ff:c1:0e:1e:60
-89
Onibus chegou!
60ff:ff:c1:0e:1e:60
-90
Onibus chegou!
60ff:ff:c1:0e:1e:60
-89
Onibus chegou!
60ff:ff:c1:0e:1e:60
-90
Onibus chegou!
60ff:ff:c1:0e:1e:60
-88
Onibus chegou!
```

Figura 17 – Nível RSSI a 6m de distância

Como observado na imagem, quanto maior a distância, menor é o valor do RSSI. Observa-se também que este valor varia bastante em cada leitura. Logo foi decidido que, apesar do limite de máxima distância ser importante em alguns casos, uma leitura com margem de distância maior seria mais necessária. Em outras palavras, uma leitura equivocada seria melhor que a ausência da leitura. Para isso, foi colocado uma configuração de -90 para nível RSSI.

Por questão de simplicidade e facilidade de leitura e identificação, foi criado um *mapping* de identificador por MAC x Id personalizado.

```
struct MacToId {
    String mac;
    uint32_t id;
};

MacToId macMappings[] = {
    {"ff:ff:c1:0e:1e:60", 60},
    {"ff:ff:c1:0e:22:ec", 80},
    // Adicione mais mapeamentos conforme necessário
};

std::map<int, int> busIdToLineId = {
    {60, 8012},
    {80, 8022},
    // outros mapeamentos
};
```

Figura 18 – Mapeamento de IDs

Fonte: autores.

A figura mostra um exemplo com 2 endereços MAC de *beacons* BLE ligado com um ID personalizado. Utilizando a mesma ideia foi criado também um mapeamento entre o ID do ônibus e a sua linha correspondente.

Após uma leitura bem sucedidade de veículos nos nós, são realizadas 2 tarefas. A primeira é uma atualização no histórico local de veículos passados, este histórico é utilizado para mostrar dados localmente, mas também tem um papel importante de salvar o último horário que um certo veículo passou para que não aconteça múltiplas leituras dentro de um intervalo pequeno. Como por exemplo em casos de paradas longas num ponto.

A segunda tarefa é referente ao envio dos dados via LoRa para o *Gateway*, os dados são empacotados no formato do BUSARRIVALDATA_PACKET e então enviado.

```
void SendBusArrivalData(uint32_t busId)
{
    struct tm timeinfo;
    if (!getLocalTime(&timeinfo))
    {
      Serial.println("Falha ao obter o tempo local");
      return;
    }

    time_t unix_timestamp = mktime(&timeinfo);

    BusArrivalDataPacket arrivalData;
    arrivalData.packetType = BUSARRIVALDATA_TO_GATEWAY;
    arrivalData.lineId = busIdToLineId[busId];
    arrivalData.busId = busId;
    arrivalData.stopId = StopID;
    arrivalData.time = static_cast<uint32_t>(unix_timestamp);

    UpdateBusHistory(arrivalData, true);

    LoRa.beginPacket();
    LoRa.write((unsigned char *)&arrivalData, sizeof(BusArrivalDataPacket));
    Serial.println(arrivalData.lineId);
    Serial.println(arrivalData.time);
    LoRa.endPacket();
}
```

Figura 19 – Função de envio de dados de chegada de ônibus

Fonte: autores.

Vale ressaltar observando a imagem que o dado de tempo não é enviado no seu

modo formatado via LoRa, com o propósito de diminuir o tamanho do pacote a ser enviado. Logo toda formatação do *timestamp* ocorre apenas nos momentos que um usuário precisa do seu formato legível.

5.2.1.2 Configuração do Gateway

O *Gateway* - nó central - foi projetado para receber e processar os dados de todos os nós periféricos da rede, e fazer o envio para uma plataforma em nuvem, o TagoIO.

Para lidar com os dados recebidos de todos os nós da rede, o *gateway* precisa ser capaz de armazenar os registros de passagem dos veículos. Nesta etapa, optou-se por uma estratégia/arquitetura de *edge computing*, a fim de melhorar o tempo de resposta e reduzir largura de banda. Sendo assim, a lógica de predição de horários com base no histórico foi desenvolvida no nó central, e a camada de aplicação (TagoIO) é responsável apenas pela exibição dos dados.

Para armazenar os registros de cada veículo, implementou-se uma estrutura de dados que permitisse individualizar os registros tanto por linha como por ponto - um array bidimensional (matriz). Desse modo, define-se quantos registros de cada par (ponto, linha) serão armazenados no histórico - que é utilizado para gerar as previsões de horário

```
const int NUM_STOPS = 2;
const int NUM_LINES = 2;
const int NUM_RECORDS = 2;  // Para armazenar os últimos 2 ônibus

struct BusStopData {
  BusArrivalDataPacket records[NUM_RECORDS];
};

BusStopData stops[NUM_STOPS][NUM_LINES];
```

Figura 20 – Estrutura para salvar registros dos ônibus

Fonte: autores.

Em seguida, como boa prática, optou-se por inicializar os registros da rede toda antes da inicialização da rede, a fim de garantir que os dados na memória são nulos - nesse caso, preechendo-os com zeros.

O fluxo da rede é ditado pela passagem dos veículos através dos pontos: Sempre que um veículo é identificado, o *gateway* recebe essa informação e transmite, via MQTT, para o TagoIO. Nesse ponto, tem-se o pacote do tipo "Real Time"sendo transmitido, que, por questões de simplificação utilizou-se uma nomenclatura por concatenação de informações - por exemplo R_time_101_8022 se refere ao tempo real de chegada do onibûs da linha 8022 no ponto A (101). Além disso, há o processamento e atualização dos dados históricos.

```
void initRecords()
{
  for(int i = 0; i < NUM_STOPS; i++)
  {
    for(int j = 0; j < NUM_LINES; j++)
    {
      for(int k = 0; k < NUM_RECORDS; k++)
      {
        stops[i][j].records[k].lineId = 0;
        stops[i][j].records[k].packetType = 0;
        stops[i][j].records[k].busId = 0;
        stops[i][j].records[k].stopId = 0;
        stops[i][j].records[k].time = 0;
      }
    }
  }
  Serial.println(stops[0][0].records[0].time);
}
```

Figura 21 – Função que inicializa os registros

Fonte: autores.

```
// Função que atualiza os arrays de cada parada, mantendo sempre 2 itens em cada array
void updateRecords(int stopId, int lineId, const BusArrivalDataPacket& newRecord) {

  // Desloca os registros existentes para abrir espaço para o novo registro
  for (int i = NUM_RECORDS - 1; i > 0; --i) {
    stops[stopId][lineId].records[i] = stops[stopId][lineId].records[i - 1];
  }

    // Insere o novo registro no início do array
    // Sempre o último ônibus que passou estará na posição 0 do Array
    Serial.println(newRecord.time);
    stops[stopId][lineId].records[0] = newRecord;
}
```

Figura 22 – Função que atualiza os dados históricos

Fonte: autores.

```
// Função para gerar uma previsão de tempo com base nos últimos registros
uint32_t getPredictTime(int stopId, int lineId) {

  if(stops[stopId][lineId].records[1].time != 0 &&
      stops[stopId + 1][lineId].records[0].time != 0)
      {
        return stops[stopId + 1][lineId].records[0].time - stops[stopId][lineId].records[1].time;
      }
  else if(stopId == 0){
    return 1200; // 60 s*20 m = 1200s
  }
  else{
    return 0;
  }
}
```

Figura 23 – Função que obtém a previsão de horário

Fonte: autores.

A função *updateRecords* atualiza os registros, deslocando o array *records* de cada par (ponto, linha), de modo a manter sempre *NUM_RECORDS* registros. Já a função *getPredictTime* obtém a previsão de horário de uma linha específica para um determinado ponto. Após as duas serem executadas, é possível montar o pacote de predict a ser enviado para os pontos desejados - pontos que fazem parte daquela linha - e para o TagoIO. Nesta etapa, o mesmo padrão de concatenação de informações para envio a nuvem, ou seja, o

pacote de predict é da forma "P_time_101_8022", indicando que se trata de um horário previsto de chegada para a linha 8022 no ponto com id 101.

Nos métodos de tratamento de pacotes de dados, é realizado também a criação de pacotes JSON utilizando a biblioteca *ArduinoJson.h* enviados para o TagoIO. Neste empacotamento, utiliza-se uma estratégia de nomeação de variáveis JSON com contaneção de nome + diversas informações do dado chegado em formato de String. Assim, passa a ser possível diferenciar as variáveis no TagoIO por ponto, linha e, se necessário, por ônibus.

5.2.2 TagoIO

Era necessário desenvolver um método para visualizar os dados relativos à chegada dos ônibus. A ideia inicial era a criação de um aplicativo que pudesse se conectar via Bluetooth ao ponto em questão para acessar as informações desejadas. No entanto, devido a restrições temporais, optou-se por criar uma representação visual por meio da plataforma TagoIO. Devido sua grande varieadade e versatilidade para outras funções além da visualização final dos dados.

Nesta plataforma é possível criar um dashboard para exposição dos dados enviados pelo ESP32, de diferentes maneiras, com algumas estruturas prontas (dispaly, tabelas dinâmicas, tabela estática, cards, gráficos variados), como pode ser visto na imagem abaixo:

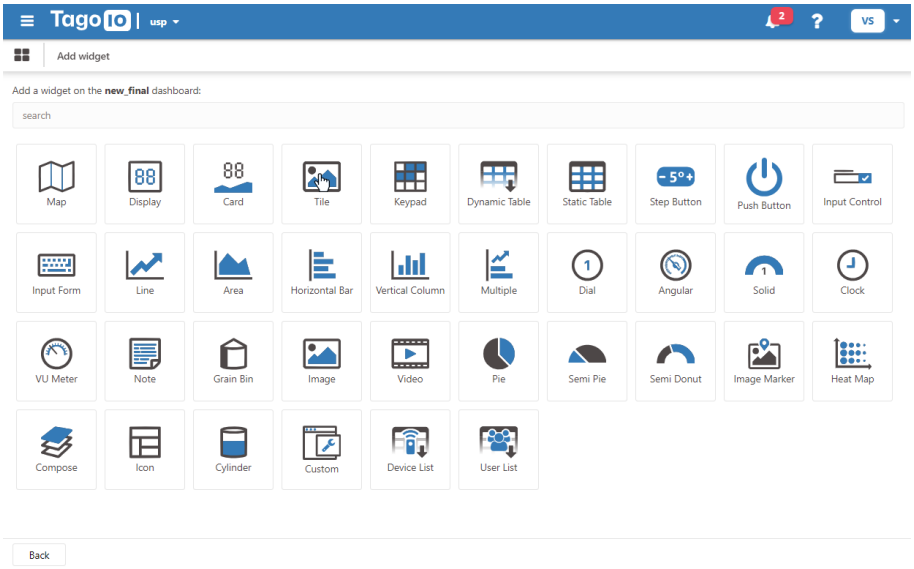


Figura 24 – Templates para dashboard

Fonte: autores.

Portanto foi necessário estabelecer a comunicação entre o dispositivo e a plataforma, de maneira que os dados fiquem salvos em um *bucket*, e assim seja possível utilizar os dados para exibição, e possíveis tratamentos de dados para aprimorar a técnica de *predict* utilizada.

Neste processo, foi feito um JSON *parser* para a correta leitura dos dados enviados pelo ESP32, isto foi necessário devido como a biblioteca *PubSubClient* monta e trasmite dados, com campos inúteis para o funcionamento do sistema. Porém este se demonstrou desnecessário e foi descartado pois a própria plataforma parseia o pacote. Analisando apenas o *payload* montado no código através da biblioteca ArduinoJson, que monta o JSON somente com os dados úteis pro sistema, através da marcação deste pacote com uma tag que será lida e a plataforma fica encarregada de fazer o *parse*.

Com os dados já prontos para uso, era necessário escolher a melhor maneira de expor as informações. Iniciamente optou-se por utilizar a tabela dinâmica, assim cada evento gerava uma linha nova, o que não era adequado para o projeto, considerando que deveria ser algo prático e intuitivo. Visado otimizar o *front-end* para este tipo de dado o *template* de tabela estática foi escolhido, como pode ser visto na imagem abaixo (contendo dados aleatórios). Assim a cada chegada de um ônibus a tabela é atualizada na célula correta.

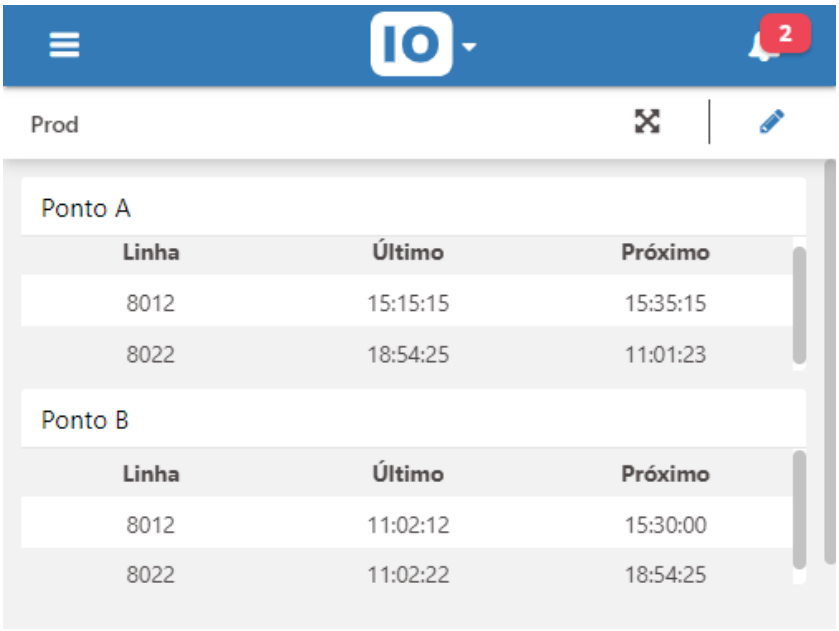


Figura 25 – Front-end montado pelo TagoIo (dados aleatórios)

Fonte: autores.

E como pode ser visto na imagem abaixo foi decidido que utilizaría-se a tabela dinâmica para registar todos os dados enviados a cada chegada de ônibus para controle. Exemplo de alguns dados na imagem abaixo.

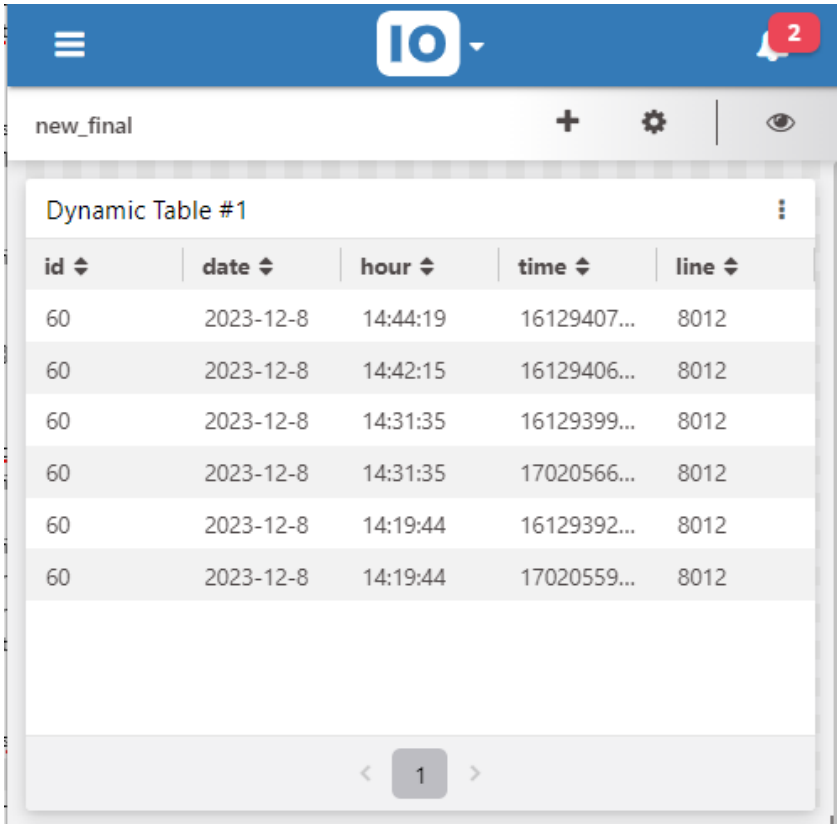


Figura 26 – Tabela Dinâmica para controle (dados aleatórios)

Fonte: autores.

Este gatilho de acionamento (ônibus chegando em um ponto), é monitorado e implementado atráves da ferrementa de *Action* do TagoIO. Relacionando um evento a uma ação, que pode ser uma notificação por algum canal (e-mail, SMS, na própria plataforma), enviar uma mensagem MQTT, enviar os dados recebidos para algum endpoint através do protocolo HTTP, rodar um análise de dados, ou por fim, inserir os dados recebidos em um *bucket* para armazenamento. Esta última, foi a opção que foi utilizada, todos os eventos são salvos neste *bucket* criado automaticamente com a criação de um dispositivo. Outra função da *Action* é fazer a marcação do pacote com tag (como já mencionado) para o TagoIO parsear corretamente a mensagem e salvá-la adequadamente. Essas tags são mais formalmente as *MQTT Topics*, sendo que cada uma distingue um pacote. Os nomes desses *topics* devem ser idênticos no código e nestes espaços (como pode ser visto na imagem abaixo) na *Action*.

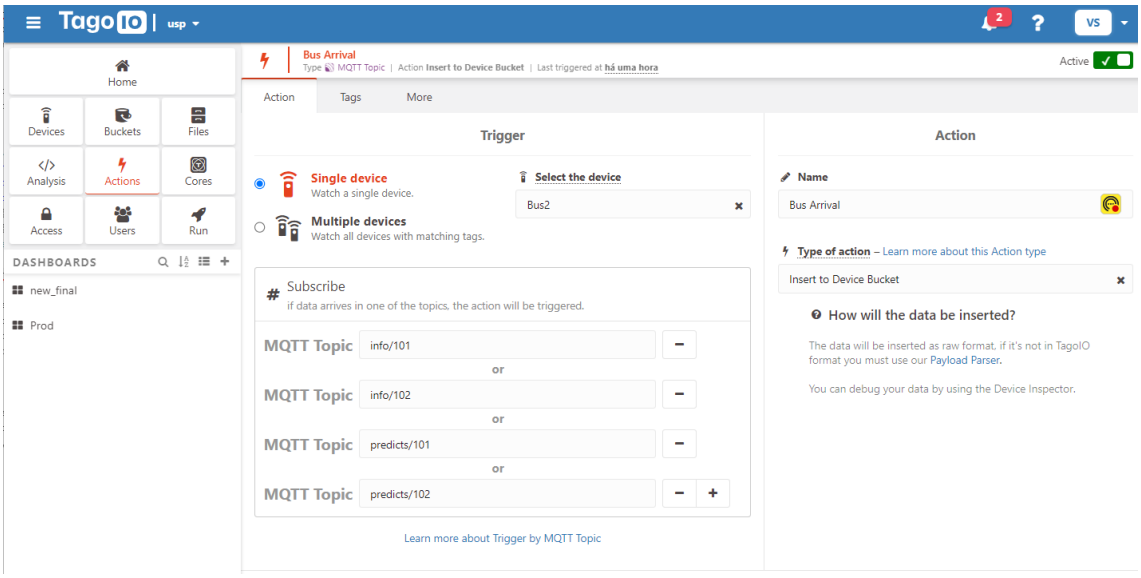


Figura 27 – Ferramenta Action

Fonte: autores.

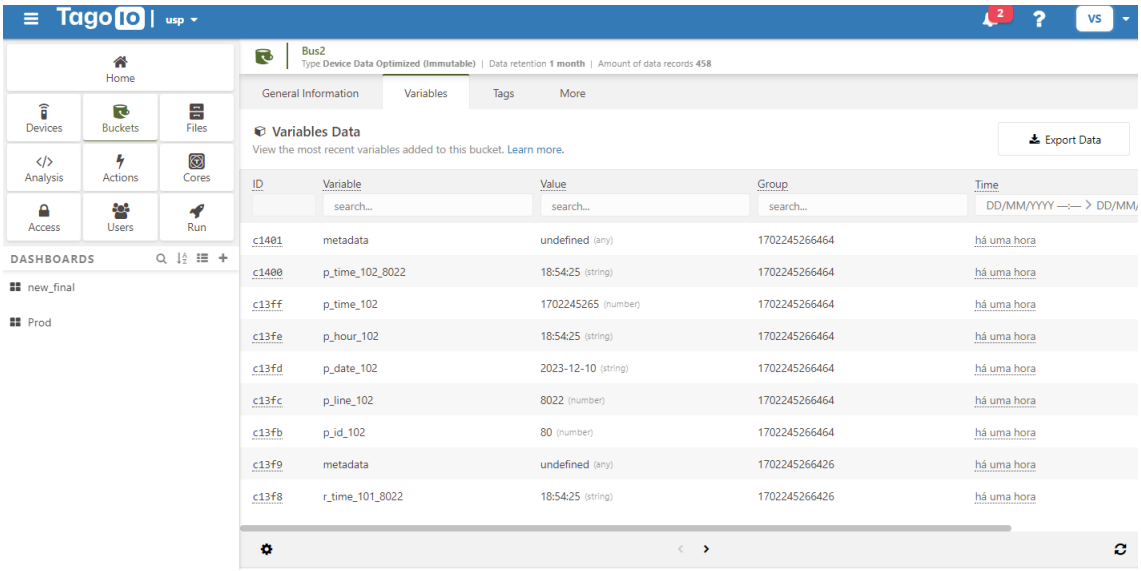


Figura 28 – Ferramenta Bucket

Fonte: autores.

Por fim, para a adaptação ao *template*, houve a necessidade de modelar os dados de uma maneira específica, para que a lógica de atualização de pontos distintos (pontos A e B) e das colunas necessárias fosse estabelecida. Assim os campos dentro do pacote enviado depende do ponto em questão e da linha, de modo que a informação da mensagem é em parte transmitida no próprio nome do campo, através da concantenação da informação e do nome do campo separados por underlines (_). Por exemplo R_time_101_8022 se refere ao tempo de chegada do onibûs da linha 8022 no ponto A (101), e R_time_102_8022 a mesma coisa porém no ponto B (102).

5.3 Testes e Avaliação

Essa seção contém uma breve descrição do procedimento da avaliação e validação do trabalho, através de testes e resultados de hardware, software, integração e outros.

5.3.1 Experimentos e Validação dos Nós

Os nós contém o papel de coletar dados via BLE, enviar e receber dados via LoRa e mostrar os dados localmente através do *display*. Nessa seção terá testes e problemas encontradas durante o experimento da parte dos nós (pontos de ônibus).

5.3.1.1 Experimentos com Beacons BLE

Leitura de *beacons* BLE tem um funcionamento relativamente simples. Módulos leitores ficam a espera de um sinal BLE anunciada por um *beacon*, e só do módulo reconhecer a aproximação deste *beacon* através de um identificador único como endereço MAC significa que ele já está funcionando como um sensor de proximidade.

O primeiro passo foi identificar os endereços MAC dos *beacons* disponíveis, este passo pode ser feito por um aplicativo de identificação de sinais BLE, ou até mesmo pelo próprio aparelho celular caso ele suporte BLE. Conhecendo estes endereços, cria-se uma lista de dispositivos permitidos para o método de leitura sempre buscar apenas por *beacons* contidos nessa lista.

Na solução de leitura passada especificada na figura 15 existia um intervalo de tempo dedicado para o *scan* de dispositivos BLE, o que causava um bloqueio do restante do programa durante a leitura. Esperava-se contornar este problema colocando uma janela pequena o suficiente para que outros processos não fossem prejudicados. Porém logo percebeu-se que o tempo mínimo configurável de janela para este método era de 1 segundo, um tempo suficiente para bloquear a leitura de um pacote LoRa com uma alta probabilidade.

Uma nova solução encontrada para este problema foi a utilização da biblioteca *ArduinoBLE*. O método de leitura dessa biblioteca se diferenciava do antigo no seu jeito de procurar por dispositivos permitidos - ao invés de se prender num *loop* tentando encontrar algum dispositivo da lista, a cada iteração do *loop* principal o leitor percorre a lista procurando se algum dispositivo da lista está próximo.

Para a validação do *beacon*, foi colocado um *print* no monitor serial da plataforma arduino avisando a leitura bem sucedida do dispositivo. Durante a validação, foi também registrado os níveis RSSI em relação a distância conforme as figuras 16 e 17.

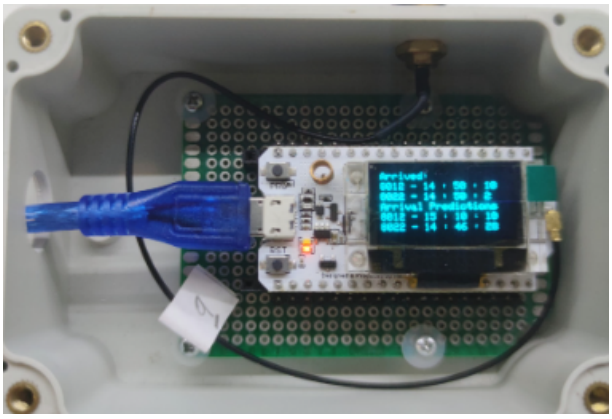


Figura 29 – *Display* do nó após leitura de ônibus pelo *beacon*

Fonte: autores.

5.3.1.2 Experimentos com contador interno configurado e formatação do *timestamp*

Uma funcionalidade imprescindível no projeto é o registro correto e preciso do tempo na chegada do veículo. Portanto todas as funcionalidades vitais dos nós passaram a ter uma condição de "timestampConfigured = true", além de que na parte do *Gateway* existe também uma verificação que checa se o *timestamp* coletado da internet é valida ou não. Depois de ter certeza que o contador do nó está devidamente configurado, coleta-se dados do *beacon* e converte-se o tempo coletado num formato legível para humanos.

A formatação do *timestamp* é realizada utilizando uma biblioteca nativa do arduino, através de um tipo de variável de formatação de tempo chamado "*struct tm*"mostrado nas figuras baixo:

```
uint32_t unixTime = busHistory[temp].time;
struct tm timeStruct;
gmtime_r((const time_t *)&unixTime, &timeStruct);

int currentHour = timeStruct.tm_hour - 3;
if(currentHour < 0)
{
  currentHour = currentHour + 24;
}

Serial.println("Arrival Time: " + String(currentHour) + " : " + String(timeStruct.tm_min) + " : " + String(timeStruct.tm_sec));
```

Figura 30 – Formatação de *unix timestamp* em *struct tm*

Fonte: autores.

```
Bus arriving simulation: BusId = 19
Bus ID: 19
Line ID: 8032
Arrival Time: 21 : 21 : 11
Bus ID: 18
Line ID: 8032
Arrival Time: 21 : 20 : 41
Bus ID: 17
Line ID: 8032
Arrival Time: 21 : 20 : 11
Bus ID: 16
Line ID: 8032
Arrival Time: 21 : 19 : 40
Bus ID: 15
Line ID: 8032
Arrival Time: 21 : 19 : 10
Bus ID: 14
Line ID: 8032
Arrival Time: 21 : 18 : 40
Bus ID: 13
Line ID: 8032
Arrival Time: 21 : 18 : 10
Bus ID: 12
Line ID: 8032
Arrival Time: 21 : 17 : 40
8032
1697415671
```

Figura 31 – Teste de coleta e formatação de *timestamps*

Fonte: autores.

5.3.1.3 Experimentos da lógica interna para envio e recebimento de pacotes

Encontra-se ao longo do código diversas condições para a leitura e processamento de pacotes, isso ocorre devido à necessidade de filtrar pacotes que estão chegando no módulo que não são do interesse dele, afinal de contas comunicação LoRa é *broadcast*. Esses filtros são realizados por 3 diferentes informações, cada um com seu propósito:

- *Packet Type* - O tipo de pacote define qual será o tratamento que o pacote irá receber.
- ID da linha - Primeira barreira do filtro, pontos que não fazem parte da linha do dado enviado nunca ficarão interessados no pacote.

- ID do ponto - Mesmo sendo da mesma linha do dado, em alguns casos não faz sentido receber o pacote, como por exemplo um ponto da frente mandar seu dado de chegada a pontos anteriores.

Foram criado listas de objetos que guardam dados de chegadas de veículos e dados de predição enviadas por outros pontos, com a finalidade de verificar se os dados estão sendo devidamente lidos, tratados e enviados. A lista de dado de chegada tem um outro papel muito importante de inibir a leitura do mesmo veículo múltiplas vezes pelo BLE. Logo a leitura de um ônibus é realizado somente se:

- O ID do veículo não se encontra na lista de histórico
- O ID do veículo foi encontrado na lista, porém o horário da última chegada daquele veículo passou um intervalo de tempo grande o suficiente

Os filtros e condições apresentados acima são essenciais para o funcionamento do nó, sendo que a validação dele ocorre através do seu próprio bom funcionamento.

5.3.2 Experimentos e Validação do *Gateway*

Nessa seção será apresentado alguns experimentos realizados para validação do *gateway*

5.3.2.1 Experimentos de coleta do timestamp via internet e envio da configuração

Um papel fundamental do *Gateway* é o envio do timestamp sincronizado com a internet para os nós dos pontos que não tem acesso à internet. Para isso, foram utilizados os métodos "configTime()"e "getLocalTime()", onde o configTime() recebe como parâmetro links de *Network Time Protocol* (NTP) e getLocalTime() devolve o valor de *unix timestamp* do momento. A validação deste valore pode ser facilmente realizado comparando com o valor do *timestamp* atual de um site de tempo no fuso 0. Este dado então é empacatado como um tipo *TIMESTAMP_PACKET* e enviado a todos os nós do sistema.

```
Connecting to MQTT...
Connected to MQTT
[LoRa Gateway] Tentando iniciar comunicacao com o radio LoRa...
[LoRa Gateway] Comunicacao com o radio LoRa ok
1697415588
entrou
Sending Timestamp Configuration
12
8032
1697415588
Pacote enviado com sucesso!
```

Figura 32 – Envio de configuração *timestamp*

Fonte: autores.

5.3.2.2 Experimentos de envio de dados para a nuvem e outros nós

O comportamento do *gateway* quando recebe um pacote de algum nó é

- Gerar previsões para os próximos nós.
- Empacota estes dados e envia-os para nós alvos.
- Gera objetos JSON para enviar para a nuvem.

Para testes de geração de previsões e o seu envio para nós e nuvem, foi realizado uma simulação de um caso real de circulação de ônibus com 2 pontos, A e B. Seguindo o funcionamento descrito na seção: 5.2.1.2, o experimento se constitui de 3 passos:

- Veículo chega no ponto A.
  - Grava chegada do veículo no ponto A.
  - Gera uma previsão nula para o ponto B.
  - Gera objetos JSON e pacotes para a nuvem e ponto B, respectivamente.
- Veículos chega no ponto B.
  - Grava chegada do veículo no ponto B.
  - Gera objetos JSON para a nuvem.
- O mesmo veículo passa no ponto A novamente.
  - Grava chegada do veículo no ponto A.
  - Gera uma **previsão não nula a partir do histórico** para o ponto B.
  - Gera objetos JSON e pacotes para a nuvem e ponto B, respectivamente.

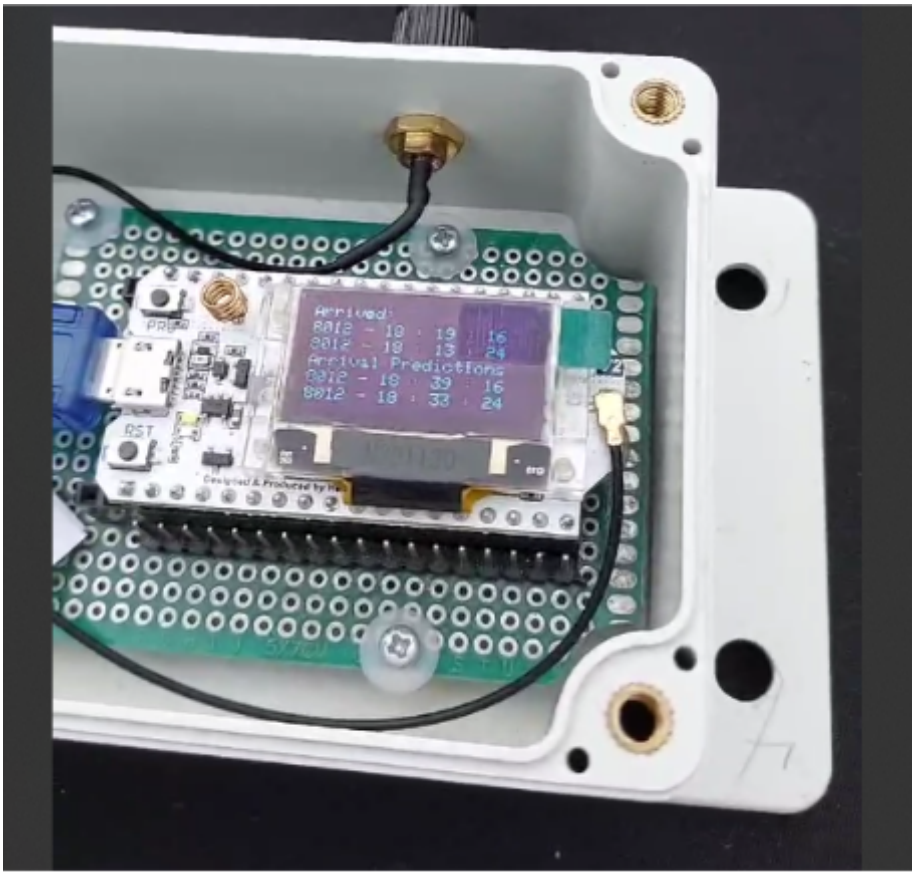


Figura 33 – Resultado do teste no ponto A

Fonte: autores.

A figura acima ilustra o resultado do ponto A após a segunda passagem do veículo. Na imagem mostra os tempos de chegada 18:13:24 e 18:19:16. As previsões de chegada gerada no ponto A são valores tabelados de 20 minutos, considerando que o ponto A é o primeiro ponto da linha, e pela limitação vinda da ausência do ponto anterior a ela.

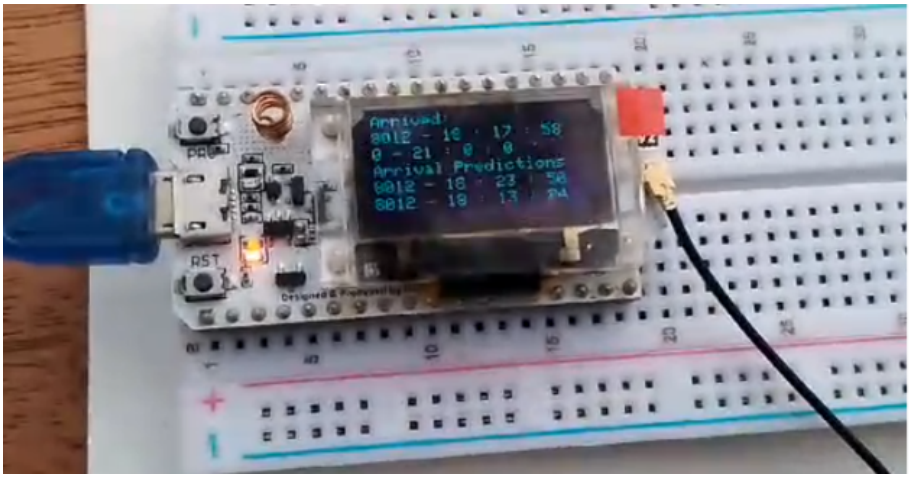
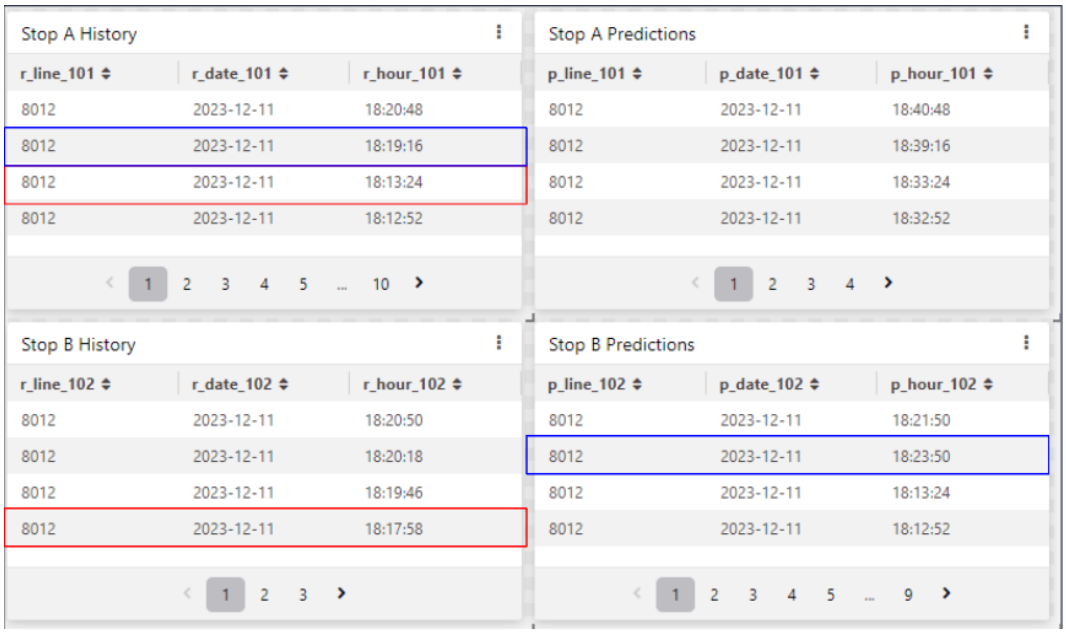


Figura 34 – Resultado do teste no ponto B

Fonte: autores.

A figura acima ilustra o resultado no ponto B, no mesmo momento que o veículo passa no ponto A pela segunda vez. Na imagem mostra o dado de chegada 18:17:58 e 18:23:50.



Stop A History

r_line_101	r_date_101	r_hour_101
8012	2023-12-11	18:20:48
8012	2023-12-11	18:19:16
8012	2023-12-11	18:13:24
8012	2023-12-11	18:12:52

1 2 3 4 5 ... 10

Stop A Predictions

p_line_101	p_date_101	p_hour_101
8012	2023-12-11	18:40:48
8012	2023-12-11	18:39:16
8012	2023-12-11	18:33:24
8012	2023-12-11	18:32:52

1 2 3 4

Stop B History

r_line_102	r_date_102	r_hour_102
8012	2023-12-11	18:20:50
8012	2023-12-11	18:20:18
8012	2023-12-11	18:19:46
8012	2023-12-11	18:17:58

1 2 3

Stop B Predictions

p_line_102	p_date_102	p_hour_102
8012	2023-12-11	18:21:50
8012	2023-12-11	18:23:50
8012	2023-12-11	18:13:24
8012	2023-12-11	18:12:52

1 2 3 4 5 ... 9

Figura 35 – Resultado do teste no TagoIO

Fonte: autores.

Essa figura mostra os mesmos resultados gravados numa tabela na plataforma nuvem TagoIO, pode-se perceber que a previsão de chegada do ponto B destacado em azul é resultado de:

$$18:19:16 + (18:17:58 - 18:13:24)$$

Onde o destaque azul mostra dados da segunda passagem, e os desques vermelhos são da primeira passagem.

6 Considerações Finais

6.1 Conclusões do Projeto de Formatura

Durante o desenvolvimento dos projetos algumas ideias foram se adaptando às restrições temporais e a erros ou obstáculos encontrados durante o processo. Foi possível atingir o propósito inicial da concepção do projeto com pequenas mudanças nos detalhes dos requisitos funcionais: O microntrolador, originalmente responsável pela recepção dos dados "crús"(sem nenhum processamento) e pelo envio destes, também ficou responsável pelo processamento de dados, o que inicialmente seria feito no servidor central atráves da plataforma TagoIO. Porém devido a limitações da plataforma para lidar com os dados de tempo e para processar os dados da maneira desejada, foi decidido que os dados seriam enviados para plataforma já processados pelo microntrolador. Os outros requisitos também foram atingidos, de modo que foi possível atingir uma latência baixa no processamento e transmissão dos dados, e uma facilidade de manutenção do sistema - que consiste basicamente em atualizar os códigos de cada módulo.

Conseguimos alcançar os diferenciais que tornam o projeto vantajoso e útil, como uma arquitetura que não depende de GPS nem conexão internet por parte do usuário final, que podera acessar esses dados apenas com Bluetooh. Dados os quais são de grande ajuda para quem precisa utilizar o serviço de transporte público, reduzindo tempos de espera inúteis, oriundos da falta de conhecimento do tráfego local de um ônibus, seja por falta de Internet ou pela própria ausência da informação.Locais rurais são exemplos que podem faltar Internet ou apresentar conexões muito instáveis, e a utilização da rede Lora permite a conexão dos nós mesmo muito distantes e o tráfego desejado das informações. Da maneira que foi pensando, o projeto é facilmente escalável, podendo ser expandido para quantos nós e ônibus fossem necessários, demandando apenas algumas configurações para cada objeto (nós e ônibus) novo adicionado.

6.2 Perspectivas de Continuidade

Possíveis extensões do projeto poderiam ser feitas, a partir do aprimoramento do método/técnica utilizada para obter a previsão de chegada de um ônibus de uma linha em determinado ponto. Tendo espaço para desenvolvimento tanto no método de processamento, quanto na abordagem de quando e como essa informação é originida. Outro aspecto interessante seria o desenvolvimento um aplicativo *mobile* que receba as informações do ponto por *Bluetooh* e exiba-as numa tela mais intuitiva e com desing mais sofisticado e atraente. A expansão da quantidade de dispositivos envolvidos seria

interessante para visualizar uma operação mais realistica e complexa, apesar de não apresentar tantos desafios devido a escalibilidade natural do projeto. Explorar outro tipo de rede, como a rede difusa, onde deixa de existir um nó central aumentado disponibilidade, escalabilidade e confiabilidade da rede devido a independência do sistema a um ponto único, além de possibilidar um alcance muito maior dado que os nós não precisam estar próximos de um outro em específico, permitindo inclusive novas abordagens para redes e esquema de maior complexidade.

Referências

AN, S.-h.; LEE, B.-H.; SHIN, D.-R. A survey of intelligent transportation systems. In: *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks*. [S.l.: s.n.], 2011. p. 332–337. Citado na página 26.

ANTP. *Relatório geral 2018*. [S.l.], 2018. Citado 2 vezes nas páginas 20 e 21.

ARAÚJO, M. R. M. de et al. Transporte público coletivo: discutindo acessibilidade, mobilidade e qualidade de vida. *Psicologia & Sociedade*, v. 23, p. 574–582, 2011. Citado na página 19.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, v. 54, n. 15, p. 2787–2805, 2010. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2010.05.010>. Citado na página 27.

BLUETOOTH SPECIAL INTEREST GROUP. *Bluetooth Core Specification*. 2010. Disponível em: <https://www.bluetooth.com/specifications/specs/core-specification-4-0/>. Citado 2 vezes nas páginas 33 e 34.

GIBSON, D. V.; KOZMETSKY, G.; SMILOR, R. W. *The Technopolis Phenomenon: Smart Cities, Fast Systems, Global Networks*. New York: Rowman & Littlefield, 1992. Citado na página 26.

GIFFINGER, R. et al. *Smart Cities: Ranking of European Medium-Sized Cities*. Vienna, Austria, 2007. Retrieved September 25, 2016. Citado na página 26.

HIRSHKOWITZ, M. et al. National sleep foundation's sleep time duration recommendations: methodology and results summary. *Sleep Health*, v. 1, n. 1, p. 40–43, 2015. ISSN 2352-7218. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2352721815000157>. Citado na página 22.

JENSEN, C. Its in australia. 1996. Disponível em: <[URLdoartigo]>. Citado na página 26.

JIANYONG, Z. et al. Rssi based bluetooth low energy indoor positioning. In: *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*. [S.l.: s.n.], 2014. p. 526–533. Citado na página 46.

LoRa Alliance. A technical overview of LoRa ® and LoRaWAN TM What is it? 2020. Disponível em: <https://hz137b.p3cdn1.secureserver.net/wpcontent/uploads/2020/11/what-is-lorawan.pdf?time=1671456125>. Citado na página 36.

PAPAPANAGIOTOU, I. Enhancing iBeacon Based Micro-Location with Particle Filtering. *Globecom*, 2015. Citado na página 34.

PSICOLOGIA, C. F. de. *Caderno de Psicologia do Trânsito e Compromisso Social*. Brasília, DF: CFP, 2000. Citado na página 19.

Semtech Corporation. Lora® and lorawan®: A technical overview. 2022. Disponível em: <https://loradevelopers.semtech.com/uploads/documents/files/LoRa_and_LoRaWANA_Tech_Overview-Downloadable.pdf>. Citado na página 37.

SILVA, D. M. d. *Sistemas inteligentes no transporte público coletivo por ônibus*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, Brasil, 2000. Disponível em: <http://hdl.handle.net/10183/3134>. Citado na página 22.

TAVARES, A. et al. Information on public transport: A comparison between information systems at bus stops. *Procedia Manufacturing*, v. 3, p. 6353–6360, 2015. ISSN 2351-9789. 6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2351978915009592>. Citado na página 22.

TOPPETA, D. *The Smart City Vision: How Innovation and ICT Can Build Smart, "Livable", Sustainable Cities*. Milan: The Innovation Knowledge Foundation, 2010. Citado na página 27.

VARGAS, H. C. (I)Mobilidade urbana nas grandes cidades. *URBS*, São Paulo, n. 47, 2008. Citado na página 19.

VIANNA, G. S. B.; YOUNG, C. E. F. In search of lost time: An estimate of the production losses in traffic congestion in brazil; [em busca do tempo perdido: Uma estimativa do produto perdido em trânsito no brasil]. *Revista de Economia Contemporanea*, v. 19, n. 3, p. 403 – 416, 2015. Cited by: 1; All Open Access, Gold Open Access, Green Open Access. Citado na página 22.

WANG, S. et al. A novel iot access architecture for vehicle monitoring system. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2016. p. 639–642. Citado 2 vezes nas páginas 25 e 26.

WOLF, M. Chapter 1 - embedded computing. In: WOLF, M. (Ed.). *Computers as Components (Fifth Edition)*. Fifth edition. Morgan Kaufmann, 2023, (The Morgan Kaufmann Series in Computer Architecture and Design). p. 1–52. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780323851282000013>. Citado 2 vezes nas páginas 29 e 33.