UNIVERSIDADE DO ESTADO DO AMAZONAS - UEA
ESCOLA SUPERIOR DE TECNOLOGIA - EST
ENGENHARIA ELÉTRICA BACHARELADO

MAYKON HENRIQUE FERREIRA DA SILVA

DESENVOLVIMENTO DE SISTEMA IOT PARA MONITORAMENTO DE TRANSPORTE DE CARGA REFRIGERADA

Manaus
2022

MAYKON HENRIQUE FERREIRA DA SILVA

DESENVOLVIMENTO DE SISTEMA IOT PARA MONITORAMENTO DE TRANSPORTE DE CARGA REFRIGERADA

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentado à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Fabio de Sousa Cardoso

Manaus
2022

Universidade do Estado do Amazonas – UEA
Escola Superior de Tecnologia - EST

Reitor:
André Luiz Nunes Zodahib
Vice-Reitor:
Kátia do Nascimento Coureiro
Diretora da Escola Superior de Tecnologia:
Ingrid Sammyne Gadelha Figueiredo
Coordenador do Curso de Engenharia Elétrica:
Israel Gondres Torné

Banca Avaliadora composta por: *Data da defesa: 13/10/2022*
Prof. Fábio de Souza Cardoso, Me *(Orientador)*
Prof. Angilberto Muniz Ferreira Sobrinho, Dr
Prof Israel Gondres Torné, Dr

CIP – Catalogação na Publicação

Da Silva, Maykon Henrique Ferreira

Desenvolvimento de sistema iot para monitoramento de transporte de carga refrigerada / Maykon Henrique Ferreira da Silva ;[orientado por] Fábio de Souza Cardoso, Me . – Manaus: 2022.
60 p.: il.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica). Universidade do Estado do Amazonas, 2022.

1. Internet das Coisas. 2. Gerenciamento de Frotas. 3. Thinger.io. I. Cardoso, Fábio de Souza.

MAYKON HENRIQUE FERREIRA DA SILVA

DESENVOLVIMENTO DE SISTEMA IOT PARA MONITORAMENTO DE TRANSPORTE DE CARGA REFRIGERADA

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro Eletricista.

Nota obtida: 10 (dez)

Aprovada em 13/10/2022.

Área de concentração: Automação

BANCA EXAMINADORA

_____
Orientador: Fabio de Sousa Cardoso, Me.

_____
Avaliador: Angilberto Muniz Ferreira Sobrinho, Dr.

_____
Avaliador: Israel Gomes Torné, Dr.

Manaus
2022

Dedicatória

À Maria Gracirene Ferreira da Silva, mulher, empreendedora e mãe. E à minha irmãzinha Natália Ferreira Mourão.

AGRADECIMENTOS

Agradeço à minha mãe , Maria Gracirene Ferreira da Silva por suportar todas suas noites mal dormidas, todos os dias de fome, todas as jornadas excessivas de trabalho para conceder-me a oportunidade de acesso à educação.

Agradeço a todos meus colegas de faculdade que me apoiaram nessa caminhada, oferecendo-me apoio com seus conhecimentos em diversas áreas.

Agradeço à professora e atual diretora da Escola Superior de Tecnologia Ingrid Gadelha por todo o seu apoio durante meu afastamento e posterior reintegração à UEA.

Agradeço à Igreja de Jesus Cristo dos Santos dos Últimos Dias pela oportunidade de servir como missionário de tempo integral na Missão Argentina Buenos Aires Sul entre os anos de 2017 e 2019.

Agradeço à todas as famílias que conheci como missionário em especial à Família Maidana, Família Martinez, e também à Finance, JeanMax, Mendell, Elizabeth, Rebecca e Pethus por me apresentarem sua cultura e me ensinarem sobre diligência, caridade, amor e fé.

Agradeço à minha falecida avó Maria José que cuidou de mim quando criança, que me alimentou com o pouco que tinha e me ensiou sobre a humildade e perseverança.

Agradeço a toda equipe que compõe a universidade do estado do Amazonas. Em especial: Araci Freitas que sempre me auxiliou durante todos esses anos e meu orientador, Professor Doutor Fabio de Sousa Cardoso, que aceitou auxiliar-me no desenvolvimento deste projeto.

Agradeço a Wolney Monteiro Vasconcelos por todo o apoio durantes esses últimos anos, apoios esses que foram essenciais à minha conclusão do curso.

RESUMO

Com o advento da internet, as barreiras que antes delimitavam o recebimento e envio de dados em massa foram dissipando-se à medida que a evolução tecnológica trazia consigo uma mudança de perspectiva, desde a finalidade, visto que a "domesticação" da tecnologia forçava a indústria a buscar materiais cada vez mais compactos, até a introdução da nuvem, que maximiza o rendimento das ferramentas pois não utiliza espaço de memória para o processamento de dados. No contexto logístico, para o controle de frotas existem ideias de soluções para otimização como software de desenho da distribuição da carga nas unidades, rastreadores para localização exata das unidades, coleta de dados históricos sobre o percurso das unidades, sensores para identificação do estilo de condução dos operadores, controles de consumo de combustível e detecção de níveis de fluídos e de emissões contaminantes. Tendo em vista o amplo cenário de desenvolvimento de tecnologias para melhoria da gestão de frotas, é proposto um sistema IoT de monitoramento de frotas voltado especificamente para cargas refrigeradas. Para implementação dessa estrutura optou-se pelo Raspberry, que será utilizado como gateway (elemento de conexão entre os módulos responsáveis por captar informações e os servidores de rede); e pelo Esp32, que será utilizado como *end-device*. Utilizou-se a API Thinger.io para visualização dos dados e obteve-se resultados coerentes com aqueles medidos nos sensores.

Palavras-chave: IoT, Raspberry, gestão de frotas,Thinger.io

ABSTRACT

The barriers that previously limited the receipt and sending of mass data were dissipating as technological evolution brought with it a change of perspective in itself since the “domestication” of technology forced the industry to seek materials more and more compact, until the introduction of the cloud, which maximizes the performance of the tools as it does not use memory space for data processing. In the logistical context, for fleet control, there are ideas for solutions for optimization such as load distribution design software in the units, trackers for the exact location of the units, collection of historical data on the route of the units, sensors to identify the driving style, fuel consumption controls and detection of fluid levels and contaminant emissions. In view of the broad scenario of technology development to improve fleet management, an IoT fleet monitoring system is proposed specifically for refrigerated cargo. To implement this structure, Raspberry was chosen, which will be used as a gateway (connection element between the modules responsible for capturing information and the network servers); and by Esp32, which will be used as an end-device. The Thinger.io API was used to visualize the data and results were consistent with those measured by the sensors.

Key-words: IoT, Raspberry, Fleet Management,Thinger.io

LISTA DE FIGURAS

LISTA DE QUADROS E TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
AWS	Amazon Web Services
CSS	Chirp Spread Spectrum
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers – Instituto de Engenheiros Eletricistas e Eletrônicos
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
LORA	Long Range
M2M	Machine to Machine
MAC	Media Access Control
MIT	Massachusetts Institute of Technology
ML	Machine Learning
REST	Representational State Transfer
RFID	Radio-Frequency Identification
TCP	Transmission Control Protocol
TI	Tecnologia da Informação
TLS	Transport Layer Security

SUMÁRIO

1. INTRODUÇÃO

A utilização de dispositivos inteligentes para controle de processos é realidade desde o final do século XX, onde iniciou-se o processo de informatização das indústrias com os primeiros computadores, desde então a tecnologia se incorporou aos mais diversos setores. Com o advento da internet, as barreiras que antes delimitavam o recebimento e envio de dados em massa foram dissipando-se à medida que a evolução tecnológica trazia consigo uma mudança de perspectiva, desde a finalidade, visto que a "domesticação" da tecnologia forçava a indústria a buscar materiais cada vez mais compactos, até a introdução da nuvem, que maximiza o rendimento das ferramentas pois não utiliza espaço de memória para o processamento de dados. Esse vislumbramento de possibilidades deu início ao que hoje se conhece como internet das coisas.

Com o avanço da tecnologia temos cada vez mais dispositivos conectados na internet, com isso a IoT (internet das coisas) é uma das áreas que mais cresce na atualidade, sendo amplamente utilizada por empresas de diversos setores da economia. Especialmente observa-se que o serviço logístico implementou fortemente a IoT nos últimos anos, o que permitiu organizar, automatizar e controlar os processos a distância e desde qualquer dispositivo conectado à internet. Por definição, uma cadeia de abastecimento eficiente se encarrega de entregar as mercadorias, do produtor ao cliente final, no tempo combinado e nas condições especificadas. Mediante o uso da tecnologia da IoT ao longo de todo este processo é possível dar um seguimento em tempo real a cada uma de suas fases, impulsionando a rapidez e a eficiência de processos automatizados que reduzem tempo e economizam custos para que o pessoal envolvido se dedique a oferecer valor à empresa (ARAÚJO,2017).

No contexto logístico, para o controle de frotas existem ideias de soluções para otimização como software de desenho da distribuição da carga nas unidades, rastreadores para localização exata das unidades, coleta de dados históricos sobre o percurso das unidades, sensores para identificação do estilo de condução dos operadores, controles de consumo de combustível e detecção de níveis de fluídos e de emissões contaminantes. Cada variável a ser monitorada dependerá do tipo de carga que se transporta além dos fatores externos que influenciam no balanço logístico. Algumas situações impõem desafios à tecnologias de transmissão de dados mais populares como WiFi e Ethernet devido a limitação de infraestrutura inerente a esses tipos de rede, a tecnologia LoRa é uma solução para situações assim; em outras situações os desafios podem ser

no controle e segurança da carga, assim atuadores de baixa latência e sensores com alta perceptibilidade são necessários.

Tendo em vista o amplo cenário de desenvolvimento de tecnologias para melhoria da gestão de frotas, é proposto um sistema IoT de monitoramento de frotas voltado especificamente para cargas refrigeradas. Para implementação dessa estrutura optou-se pelo Raspberry, que será utilizado como gateway (elemento de conexão entre os módulos responsáveis por captar informações e os servidores de rede); e pelo Esp32, que será utilizado como *end-device*. Com isso, pretende-se somar ao cenário do IoT aplicado à logística, mais uma solução para acompanhamento e controle de frota.

1.1. PROBLEMÁTICA

Dificuldades encontradas no monitoramento de cargas refrigeradas decorrentes da falta de dados sobre o estado das mesmas. Diante deste cenário é proposto um sistema de monitoramento de frota de cargas refrigeradas cuja infraestrutura de rede será composta por:

a) um Raspberry cuja função será de servir de conexão entre os módulos responsáveis por captar informações e os servidores de rede;
b) um Esp32 que será utilizado como end-device;
c) Sensores para monitoramento do veículo.

1.2.OBJETIVOS

1.2.1. OBJETIVOS GERAIS

Projetar uma aplicação IoT para monitoramento de frota de cargas refrigeradas usando Raspberry, ESP32 e sensores além de uma API de controle.

1.2.2. OBJETIVOS ESPECÍFICOS

Visando atingir o objetivo geral citado anteriormente, alguns objetivos específicos são requeridos, entre eles:

a) Conectar o *end-device* e o Raspberry de forma que os dados possam ser transferidos;
b) Integrar Raspberry e algum serviço de Cloud, onde os dados serão coletados, armazenados e processados;
c) Criar uma API para melhor visualização dos dados;
d) Realizar testes sistêmicos.

2. REFERENCIAL TEÓRICO

A Internet tem estado em um caminho constante de desenvolvimento e melhoria, mas sem dúvida não mudou muito. Por exemplo, nos primeiros dias, havia vários protocolos de comunicação, incluindo AppleTalk, Token Ring e IP. Hoje, a Internet é amplamente padronizada em IP. Nesse contexto, a IoT se torna imensamente importante porque é a primeira evolução real da Internet – um salto que levará a aplicativos revolucionários que têm o potencial de melhorar drasticamente a maneira como as pessoas vivem, aprendem, trabalham e se divertem (EZECHINA et al, 2015).

A IoT tornou a Internet sensorial (temperatura, pressão, vibração, luz, umidade, estresse), permitindo que informações de fontes remotas fossem disponibilizadas na rede. Sensores extremamente pequenos conectados à Internet podem ser colocados em plantas, animais e no solo. Na outra extremidade do espectro, a Internet está encaminhando-se para o espaço por meio do programa *Internet Routing in Space* (IRIS) da Cisco (EVANS, 2011).

2.1. IoT

Kevin Ashton propôs pela primeira vez o conceito de IoT em 1999, e ele se referiu à IoT como objetos conectados exclusivamente identificáveis com tecnologia de identificação por radiofrequência (RFID). No entanto, a definição exata de IoT ainda está em processo de formação que está sujeita às perspectivas adotadas. A IoT foi geralmente definida como infraestrutura de rede global dinâmica com recursos de auto configuração baseados em padrões e protocolos de comunicação (GOKHALE, 2018).

Tal como acontece com novos conceitos, a raiz da IoT pode ser remetida ao MIT, a partir do trabalho no Auto-ID Center. Fundado em 1999, este grupo trabalhava na área de identificação por radiofrequência em rede (RFID) e tecnologias de detecção emergentes. Os laboratórios consistiam em sete universidades de pesquisa localizadas em quatro continentes. Essas instituições foram escolhidas pelo Auto-ID Center para projetar a arquitetura para IoT .

Em 2003, havia aproximadamente 6,3 bilhões de pessoas vivendo no planeta e 500 milhões de dispositivos conectados à Internet. Ao dividir o número de dispositivos conectados pela população mundial, descobriu-se que havia menos de um (0,08) dispositivo para cada pessoa. Com base na definição do Cisco, a IoT ainda não existia em 2003 porque o número de objetos conectados

era relativamente pequeno, uma vez que dispositivos onipresentes, como smartphones, estavam apenas sendo introduzidos no mercado (EZECHINA et al, 2015).

Objetos físicos e virtuais em um sistema de IoT têm suas próprias identidades e atributos, além disso, são capazes de usar interfaces inteligentes e serem integradas como uma rede de informações. Em termos simples, a IoT pode ser tratada como um conjunto de dispositivos conectados que são exclusivamente identificáveis (LI et al, 2015). As palavras “Internet” e “Coisas” significam uma rede mundial interconectada baseada em sensores, comunicação, redes e tecnologias de processamento de informações, que podem ser a nova versão da tecnologia da informação e comunicação (TIC), a figura 1 ilustra a enorme possibilidade de implementação de endpoints IoT.

Figura 1 - Visão geral da implementação da internet das coisas

Fonte: GOKHALE (2018)

Até o momento, várias tecnologias estão envolvidas na IoT, como redes de sensores sem fio, códigos de barras, detecção inteligente, RFID, NFCs, comunicações sem fio de baixa energia, computação em nuvem e assim por diante. A IoT descreve a próxima geração da Internet, onde as coisas físicas podem ser acessadas e identificadas através da Internet. Dependendo de várias tecnologias para a implementação, a definição da IoT varia. No entanto, o fundamental da IoT implica que os objetos em uma IoT possam ser identificados exclusivamente nas representações virtuais. Dentro de uma IoT, todas as coisas são capazes de trocar dados e, se necessário, processar dados de acordo com esquemas predefinidos.

Com o avanço do 5G a implementação de sistemas de monitoramento utilizando-se da IoT será potencializada. Isso deve-se ao fato desta rede especificar uma parte de sua estrutura exclusivamente para processar, enviar e receber dados de equipamentos IoT. Quando fala-se de

IoT é inerente especificar sobre qual protocolo de comunicação os dados captados serão transferidos, pois cada caso de uso demanda algum requisito específico de rede.

2.1.1. IoT no Brasil

O desenvolvimento de infraestruturas interconectadas de objetos inteligentes foi instituído pelo Decreto nº 9.854, de 25 de junho de 2019, conhecido como Plano Nacional de Internet das Coisas cuja finalidade é implementar e desenvolver o IoT no País e, com base na livre concorrência e na livre circulação de dados, observadas as diretrizes de segurança da informação e de proteção de dados pessoais**.**

Os objetivos do Plano Nacional de Internet das Coisas são:

I - melhorar a qualidade de vida das pessoas e promover ganhos de eficiência nos serviços, por meio da implementação de soluções de IoT;

II - promover a capacitação profissional relacionada ao desenvolvimento de aplicações de IoT e a geração de empregos na economia digital;

III - incrementar a produtividade e fomentar a competitividade das empresas brasileiras desenvolvedoras de IoT, por meio da promoção de um ecossistema de inovação neste setor;

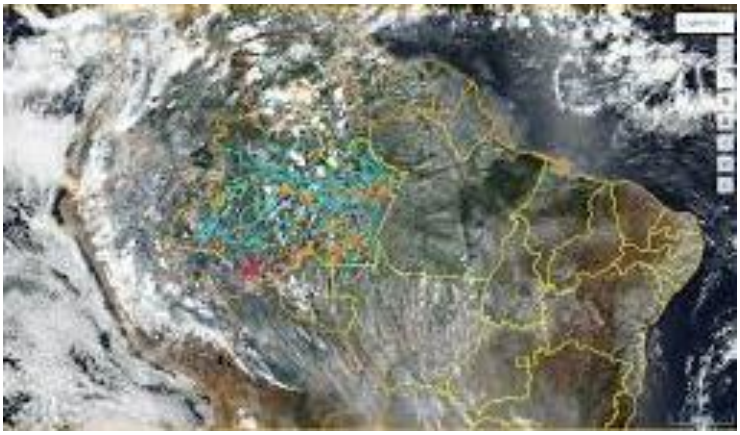
IV - buscar parcerias com os setores público e privado para a implementação da IoT; e

V - aumentar a integração do País no cenário internacional, por meio da participação em fóruns de padronização, da cooperação internacional em pesquisa, desenvolvimento e inovação e da internacionalização de soluções de IoT desenvolvidas no País.

2.1.2. IoT na Amazônia

A principal solução para monitoramento de dados presente na Amazônia Legal é o sistema de combate a queimadas desenvolvido pelo Instituto Nacional de Pesquisas Espaciais - INPE (da Cruz et al, 2018). Através do sensoriamento remoto de imagens via satélite, o sistema consegue identificar possíveis incêndios na região como visualizado na figura 2, frequentemente, provocados pelo desmatamento ilegal. Porém não é somente em questões ambientais que desenvolve-se tecnologia na Amazônia, em espaços físicos controlados, como em empresas instaladas no Pólo industrial de Manaus, observa-se contínuo uso de dispositivos IoT.

Figura 2 - Monitoramento de queimadas INPE



Fonte: DA CRUZ et al (2018)

Projetos de IoT são desenvolvidos na Amazônia, principalmente impulsionados pela indústria e universidades. Exemplos podem ser encontrados em dissertações com “Análise da implantação de um sistema de monitoramento da produção de fita adesiva contemplando os conceitos da Indústria 4.0” cuja autoria é de Lucas, Á. M. (2020), vinculado à Universidade do Estado do Amazonas. Nesta ocasião implantou-se uma estrutura tecnológica e um sistema de monitoramento da produção utilizando dispositivos inteligentes.

Um outro caso é o projeto de Silva, A. S. D. (2021) , vinculado à Universidade do Estado do Amazonas, autor de “Desenvolvimento de sistema para controle e monitoramento remoto de um forno industrial de uma empresa do polo de duas rodas do distrito industrial ”. Neste trabalho foi elaborado um sistema de controle e monitoramento remoto para um forno industrial usando o IoT, a solução em questão foi aplicada em um equipamento de uma empresa do Polo de Duas Rodas do Distrito Industrial de Manaus, auxiliando a evitar perdas de recursos, produtividade e eventuais acidentes de trabalho.

2.2. GERENCIAMENTO DE FROTAS

O trabalho desempenhado pela gestão da cadeia de suprimentos é classificado como sendo uma área importante para a inovação e investimento de Tecnologia da Informação (TI). Um crescente número de ferramentas para essa área da gestão foram e estão sendo desenvolvidas afim de otimizar as decisões logísticas das empresas, como observado na figura 3. Os investimentos aplicados nos setores de TI fazem parte de um conceito estratégico onde as empresas que buscam

ganhar vantagem competitiva buscam por um ambiente mais dinâmico para execução e interação das atividades (MAÇADA et al, 2007).

Figura 3 - Gestão de frotas

Fonte: TRINDADE et al (2021)

O transporte de cargas faz parte de um conceito básico da logística para a empresa devido a sua responsabilidade em transportar matéria prima para a linha de produção e o produto final até o seu destino. Sendo capaz até mesmo de ser classificado como uma das atividades mais relevantes devido a sua importância para o gerenciamento de custos logísticos. Nos dias atuais, mesmo com o acesso à informação sendo captada e transferida em tempo real, a atividade dos serviços de transporte é fundamental para o desempenho e atuação dos objetos da logística, pois graças aos seus serviços o produto é adquirido em tempo, quantidade e qualidade adequadas ao menor custo possível (CALAZANS et al, 2014)

Em seu trabalho “Como a tecnologia impacta na melhoria do gerenciamento de frotas de uma empresa de transporte", FEITOSA e PEZZIN (2019) discorrem sobre a cadeia de transporte e o uso em escala da Tecnologia da Informação (TI). Essa discussão é emergente no contexto dos dispositivos inteligentes pois integra o baixo custo e disponibilidade de plataformas hardwares visto nos últimos anos, os conceitos de Big Data e a administração da cadeia produtiva, interligando assim partes antes desconexas nas companhias. Contudo, é importante salientar que não são todas as empresas que estão informatizando-se, por isso, eventos que aproximem empresários de pesquisadores nas áreas de IoT são essenciais, visto que influenciam na adesão da cultura da inovação das companhias e, por conseguinte, na integração de ferramentas inteligentes em seus setores como nas linhas de produção, na administração e no transporte de cargas.

2.2.1. Soluções tecnológicas implementadas no gerenciamento de frotas

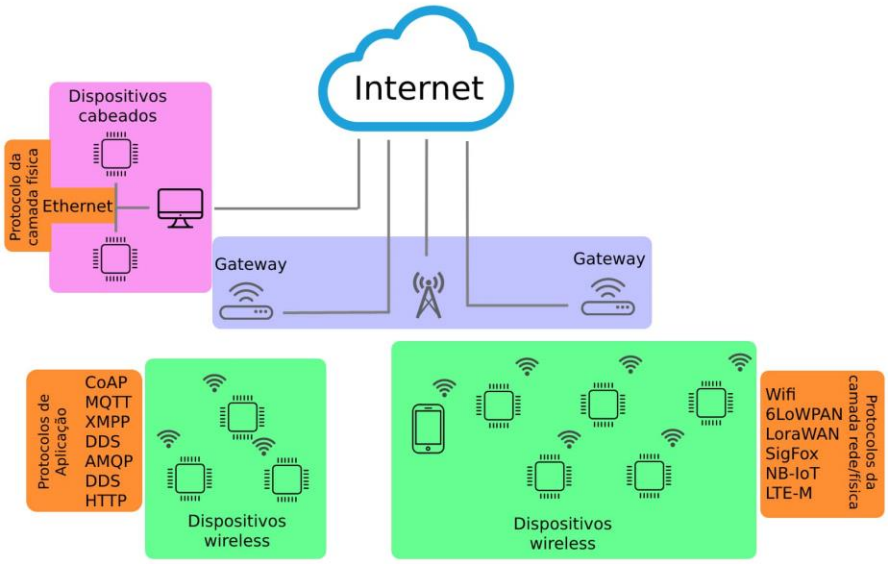
Os novos processos que são desenvolvidos para a linha de produção e gestão de negócio, são formados de acordo com a busca por eficiência por intermédio da otimização de ganhos e custos, proporcionando assim maiores conquistas competitivas nas empresas. O processo desta nova ótica de gestão permitiu o desempenho das funções logísticas que passaram de uma questão operacional para uma atividade estratégica (AGUILERA et al, 2003).

No âmbito de tecnologias aplicadas ao gerenciamento de transporte, encontra-se considerável interesse do meio acadêmico na produção de dissertações sobre o tema. Em "Desenvolvimento de Aplicação IoT utilizando a Plataforma Integrada *Softway4IoT* e *Fiware*", TRINDADE et al (2021) apresentam um caso de uso na Universidade de Brasília onde o IoT foi utilizado para gerenciar frotas em áreas do campus universitário. Seguindo a mesma temática, em seu trabalho intitulado "Monitoramento de caminhões usando internet das coisas", EGER (2017) desenvolveu um software denominado *middleware* e instalou sensores no módulo de um caminhão para obtenção de dados e controle de frota. Ambos trabalhos usam o conceito de compartilhamento de dados e rede fechada para embasar suas escolhas de infraestrutura de rede em relação aos dispositivos alocados remotamente.

2.3. ESTRUTURA DE REDE IOT

Quando se trata de transferência de dados em dispositivos IoT, a possibilidade de escolhas é grande (vide figura 4). Devido ao baixo custo, encontram-se no mercado ainda dispositivos que usam a tecnologia GSM (2G) para transferência de dados, porém com o avanço das telecomunicações e robustez das redes, os padrões 3GPP (3G), LTE e NR são encontrados em hardwares para desenvolvimento, pesquisa e até mesmo em produtos já comercializados. Tendo em vista os recursos escolhidos para o desenvolvimento do projeto, nesta seção serão abordados o protocolo da camada física Ethernet, o protocolo de aplicação HTTP, atécnica de modulação LORA e seu protocolo de camada de rede LORAWAN.

Figura 4 - Estrutura IoT



Fonte: EVANS (2022)

2.3.1. Ethernet

As Redes Ethernet se consolidaram como padrão de comunicação entre computadores desde sua invenção, como a Automação Industrial se convergiu ao longo dos últimos anos com a Tecnologia da Informação (TI), as Redes Ethernet se desenvolveram dentro do universo da Tecnologia da Automação (TA), ganhando características que delinearam um cenário de total aderência aos novos projetos e atualização de sistemas legados de rede para automação e controle.

A rede Ethernet é um protocolo para redes locais e foi desenvolvida na década de 1970 nos laboratórios do Xerox Palo Alto Research Center, posteriormente, foi revisado e melhorado em conjunto pelas empresas Intel, Xerox e Digital Equipment Corporation (DEC). Em 1985, o publicou o padrão IEEE 802.3 para estas redes locais de baixo custo.

A Ethernet foi desenvolvida na década de 1970, porém, a origem da ideia veio de Abramson quando desenvolveu a rede denominada Aloha, mostrando que múltiplos nós na rede podiam usar o mesmo canal para comunicação e enviar dados a qualquer momento. A principal diferença entre a Aloha e a Ethernet é que, a primeira permitia que qualquer nó transmitisse dados a qualquer momento e não permitia que um nó detectasse que outro nó estava transmitindo dados e não lidava com as colisões (colisões ocorrem quando dois dados tentam transmitir dados simultaneamente), exigindo assim, muitas retransmissões; a segunda, Ethernet, foi projetada com monitoramento de

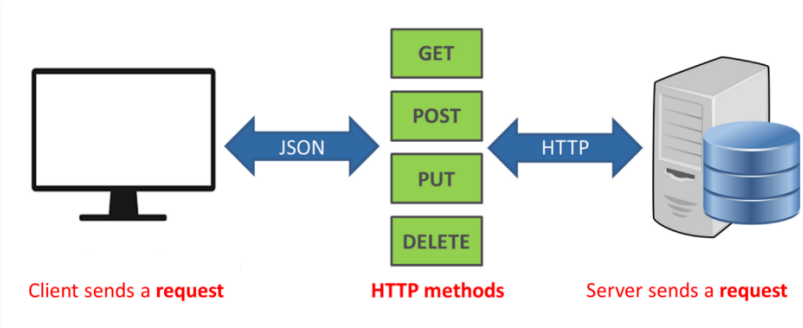
transporte e detecção de colisões. A Ethernet possui várias versões – 10 Mbps, 100Mbps, 1 Gbps e 10 Gbps– diferenciadas principalmente pelo comprimento máximo que o sinal será propagado. Ela define o meio físico de conexão do cabeamento, define o controle de acesso do dado na rede e define o quadro (frame) de informação, tudo isso baseado na norma IEEE 802-2 e IEEE 802-3, que em suas subdivisões, estabelece características técnicas dos padrões de rede .

2.3.2. HTTP

HTTP é um protocolo para buscar recursos como documentos HTML. É a base de qualquer troca de dados na Web e é um protocolo cliente-servidor, o que significa que as solicitações são iniciadas pelo destinatário, geralmente o navegador da Web. Um documento completo é reconstruído a partir dos diferentes subdocumentos obtidos, por exemplo, texto, descrição do layout, imagens, vídeos, scripts e muito mais.

Clientes e servidores se comunicam trocando mensagens individuais (em oposição a um fluxo de dados). As mensagens enviadas pelo cliente, geralmente um navegador da Web, são chamadas de solicitações e as mensagens enviadas pelo servidor como resposta são chamadas de respostas, como observado na figura 5.

Figura 5 - Protocolo HTTP



Fonte: EVANS (2022)

Projetado no início da década de 1990, o HTTP é um protocolo extensível que evoluiu ao longo do tempo. É um protocolo de camada de aplicativo que é enviado por TCP ou por uma conexão TCP criptografada por TLS, embora teoricamente qualquer protocolo de transporte confiável possa ser usado. Devido à sua extensibilidade, ele é usado não apenas para buscar documentos de hipertexto, mas também imagens e vídeos ou para postar conteúdo em servidores, como resultados de formulários HTML. O HTTP também pode ser usado para buscar partes de documentos para atualizar páginas da Web sob demanda.

HTTP é um protocolo cliente-servidor: as solicitações são enviadas por uma entidade, o agente-usuário (ou um proxy em nome dele). Na maioria das vezes, o agente do usuário é um navegador da Web, mas pode ser qualquer coisa, por exemplo, um robô que rastreia a Web para preencher e manter um índice de mecanismo de pesquisa.

2.3.2.1. Mensagens HTTP

As mensagens HTTP, conforme definido em HTTP/1.1 e anteriores, são legíveis por humanos. No HTTP/2, essas mensagens são embutidas em uma estrutura binária, um frame, permitindo otimizações como compressão de cabeçalhos e multiplexação. Mesmo que apenas parte da mensagem HTTP original seja enviada nesta versão do HTTP, a semântica de cada mensagem permanece inalterada e o cliente reconstitui (virtualmente) a solicitação HTTP/1.1 original. Existem dois tipos de mensagens HTTP, *Requests* e *Responses*, cada uma com seu próprio formato.

As *Requests* possuem os seguintes elementos:

- Um **método** HTTP, geralmente um verbo como GET, POST ou um substantivo como OPTIONS ou HEAD que define a operação que o cliente deseja executar. Normalmente, um cliente deseja buscar um recurso (usando GET) ou postar o valor de um formulário HTML (usando POST), embora mais operações possam ser necessárias em outros casos;
- O caminho do recurso a ser buscado; a **URL** do recurso retirada de elementos óbvios do contexto, por exemplo, sem o protocolo (http://), o domínio ou a porta TCP;
- A **versão** do protocolo HTTP;
- **Cabeçalhos** (*headers*) opcionais que transmitem informações adicionais para os servidores;
- Um **corpo** (*body*), para alguns métodos como POST, semelhantes aos das respostas, que contêm o recurso enviado.

As *Responses* possuem os seguintes elementos:

- A **versão** do protocolo HTTP que eles seguem;
- Um **código de status**, indicando se a solicitação foi bem-sucedida ou não e por quê;
- Uma **mensagem de status**, uma breve descrição não autorizada (*non-authoritative*) do código de status;
- **Cabeçalhos** (*headers*) HTTP, como aqueles para solicitações;
- Opcionalmente, um **corpo** (*body*) contendo o recurso buscado.

2.3.2.2. APIs baseadas em HTTP

A API mais comumente usada baseada em HTTP é a API *XMLHttpRequest*, que pode ser usada para trocar dados entre um agente de usuário e um servidor. A *API Fetch modern* fornece os mesmos recursos com um conjunto de recursos mais poderoso e flexível.

Outra API, server-sent events, é um serviço unidirecional que permite que um servidor envie eventos para o cliente, usando HTTP como mecanismo de transporte. Usando a interface *EventSource,* o cliente abre uma conexão e estabelece manipuladores de eventos. O navegador cliente converte automaticamente as mensagens que chegam no fluxo HTTP em objetos *Event* apropriados. Em seguida, ele os entrega aos manipuladores de eventos que foram registrados para o tipo dos eventos, se conhecido, ou ao manipulador de eventos onmessage, se nenhum manipulador de eventos específico do tipo foi estabelecido.

2.3.3. LoRA

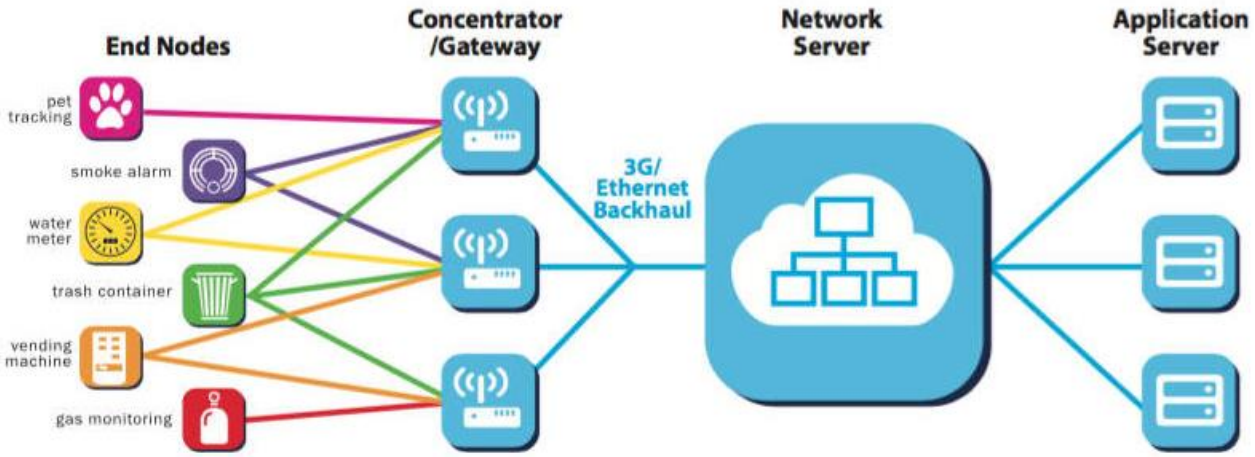
LoRa é uma técnica de modulação de espectro desenvolvida pela empresa Semtech. A camada física LoRa pode ser usada com qualquer camada MAC, entretanto, o LoRaWAN é o MAC proposto que opera este tipo de rede. Usar um rádio LoRa em uma rede de sensores tem alguns aspectos interessantes. Em primeiro lugar, como o alcance é relativamente grande (centenas de metros em ambientes fechados, quilômetros em ambientes externos), as redes podem abranger grandes áreas. Em segundo lugar, a transmissão na mesma frequência portadora, mas com fator de espalhamento diferente, é ortogonal. Isso cria a oportunidade de dividir o canal em subcanais virtuais, facilitando a transferência de dados. Em terceiro lugar, quando as transmissões ocorrem ao mesmo tempo com os mesmos parâmetros, a transmissão mais forte será recebida com alta probabilidade, ou seja, transmissões simultâneas não são destrutivas, mesmo quando seus conteúdos são diferentes (BOR et. al, 2016).

2.3.4. LoRAWAN

LoRaWAN é um padrão de rede de área e baixa potência (LPWAN) gerenciado pela LoRa Alliance, uma empresa de tecnologia sem fins lucrativos. Foi projetado para conectar dispositivos operados por bateria à Internet em redes regionais, nacionais ou globais. Além disso, o padrão LoRaWAN atende aos principais requisitos da Internet das Coisas (IoT), como comunicação bidirecional, segurança de ponta a ponta, mobilidade e serviços de geolocalização.

O LoRaWAN aproveita o espectro de rádio não licenciado na banda ISM. A especificação define os parâmetros da camada física LoRa e o padrão LoRaWAN e fornece interoperabilidade perfeita entre os fabricantes. O padrão LoRaWAN é reconhecido pela União Internacional de Telecomunicações (UIT), a agência especializada das Nações Unidas para tecnologias de informação e comunicação (TICs), como um padrão para LPWANs. Enquanto a Semtech fornece chipsets LoRa, a LoRa Alliance impulsiona a padronização e harmonização global do padrão LoRaWAN para o vasto ecossistema de dispositivos conectados à internet (LORAWAN STANDARD, 2022). A estrutura de uma rede que utiliza LoRaWAN é dividida em: *End-devices*, *Gateway*, *Network Server* e *Application Server*, tal como pode ser observado na figura 6.

Figura 6 - Estrutura de Rede LORAWAN



Fonte: LORAWAN STANDARD (2022)

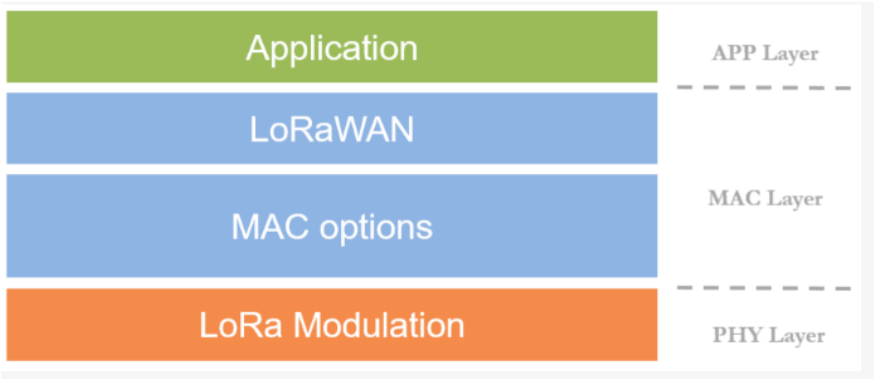
Os *End Nodes*, ou *End Devices*, representam os dispositivos LoRaWAN remotos exemplificados pelos equipamentos IoT tão amplamente utilizados. Para garantir a conexão de maneira eficiente é necessário utilizar antena apropriada e implementar um código que seja capaz de encapsular e receber as informações dos sensores e atuadores. Com esses elementos incorporados ao projeto, o envio de dados pela parte remota da rede é garantido.

Os *Gateways*, ou *Concentrators* como também são conhecidos, representam basicamente as antenas de conexão por onde os dados são encaminhados para a internet ou servidor local. O Módulo Gateway é uma das ferramentas mais importantes em uma rede LoRa pois é por meio dele que os dispositivos finais de uma rede LoRa se comunicam com um transceiver, que recebe e envia pacotes nesse mesmo protocolo. Como o objetivo de uma rede de sensores IoT é a captura de dados

para controle externo de alguma variável, fica imprescindível o transporte desses dados pela internet. Desta forma, é necessário que exista uma interface de processamento para identificar um pacote do protocolo LoRaWAN, capturar seu *payload* e informações importantes, reencapsular em um pacote do protocolo IP e enviá-lo para um servidor de internet, que, futuramente, o encaminha para a camada de aplicação. Essa interface de processamento é o módulo *gateway*, que conta com um transceiver para a comunicação LoRa e um módulo de conexão com internet como Wi-fi ou via cabo *ethernet*.

Da antena de conexão, as informações trocadas com os dispositivos são enviadas pela internet para o *Network Server*, que é responsável por encaminhar corretamente as mensagens de um certo dispositivo para sua respectiva aplicação final, ou no sentido inverso. O servidor de rede permite conectividade, gerenciamento e monitoramento de dispositivos, gateways e aplicativos de usuário final. Seus principais objetivos são garantir a segurança, escalabilidade e confiabilidade do roteamento de dados em toda a rede. Por fim, as informações chegam ao *Application Server*, que representa a plataforma onde são finalmente exibidas, ou de onde partem, as informações da comunicação.

Figura 7 - Frame LoRaWAN



Fonte: SANCHEZ-IBORRA et al (2018)

O *frame* LoRaWAN define duas camadas bem diferenciadas conforme observado na figura 7, a camada PHY, definida pela modulação LoRa, e a camada MAC, definida pelo protocolo LoRaWAN. Em relação ao primeiro, LoRa é um esquema de modulação de espectro espalhado proprietário que deriva da modulação *Chirp Spread Spectrum* (CSS) que se concentra em fornecer robustez a transmissões de longo alcance. LoRa apresenta três parâmetros configuráveis diferentes, a saber, *Spreading Factor* (SF), *Coding Rate* (CR) e *Bandwidth* (BW). Ao ajustar esses fatores,

alguns dos recursos de comunicação de ponta a ponta podem ser ajustados, por exemplo, taxa de dados, capacidade de correção de erros e faixa de transmissão, entre outros.

Em relação ao SF, a modulação do espectro espalhado é realizada representando cada bit de informação por múltiplos chips. Assim, o SF representa a razão entre a taxa de chip e a taxa de informação de banda base. Os valores usuais de SF variam de 7 a 12, portanto, valores maiores de SF aumentam a robustez do link de transmissão, aumentando a sensibilidade do equipamento receptor em detrimento da taxa de transmissão de dados. Por outro lado, ao diminuir o SF, a taxa de dados é notavelmente aumentada, mas o sinal transmitido precisa ser recebido em um nível de potência mais alto para sua decodificação adequada.

Com o objetivo de melhorar ainda mais a robustez do link, o LoRa emprega codificação de erros cíclicos para realizar a detecção e correção de erros. Essa codificação de erro incorre em uma sobrecarga de transmissão que é determinada pelo parâmetro CR, que pode ser definido para os seguintes valores, 4/5, 4/6, 4/7 e 4/8. Por fim, o BW mais empregado na transmissão LoRa é o de 125 kHz, embora também sejam suportadas larguras de banda de 250 kHz e 500 kHz (SANCHEZ-IBORRA et al, 2018).

2.4. HARDWARE

Para implementação dessa estrutura de rede na camada física optou-se pelo Raspberry Pi 4 Modelo B, que será utilizado como *gateway* e pelo ESP32 LoRa WiFi, que será utilizado como *end-device*.

2.4.1. Raspberry Pi 4 Modelo B

Em 2006, os membros do Laboratório de Computação da Universidade de Cambridge, no Reino Unido, Eben Upton, Rob Mullins, Jack Lang e Alan Mycroft pensaram em como interromper o declínio gradual do interesse dos estudantes pela ciência da computação no país. Depois de analisar o cenário, chegaram à conclusão que o problema estava nas grades curriculares dos cursos de computação e também em um novo panorama tecnológico com o qual os jovens passaram a ter contato (RASPBERRY FOUNDATION, 2022).

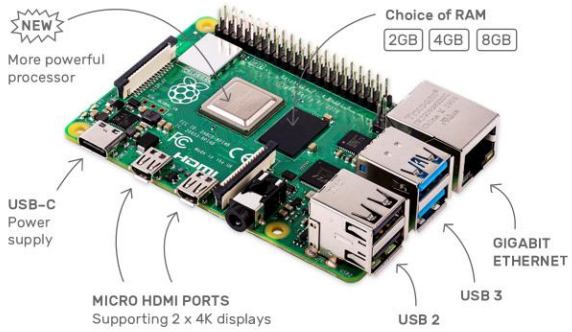
Buscando mudar esse comportamento, entre 2006 e 2008, os cientistas anteriormente citados desenvolveram diversos protótipos baseados no microcontrolador Atmel ATmega644, que serviria de base para o Raspberry Pi. A expansão dos dispositivos móveis também ajudou, pois, a

partir de 2008 já foi possível ter acesso a processadores menores, mais potentes e completos do que se tinha até então (RASPBERRY FOUNDATION, 2022)

Atualmente, está disponível no mercado a versão Raspberry Pi 4, a velocidade e o desempenho é um avanço em relação aos modelos anteriores. De acordo com o fabricante, pela primeira vez, construiu-se uma experiência de desktop completa. Com este modelo é possível editar documentos, navegar na *web* com várias guias abertas, fazer diversas planilhas ou editar uma apresentação desfrutando da mesma experiência comparado a computadores tradicionais, mas em um formato físico menor, mais eficiente em termos de energia e muito mais econômico.

O Raspberry Pi 4 Modelo B possui duas portas USB 2 e duas portas USB 3, que podem transferir dados até dez vezes mais rápido. Além disso, vem com Gigabit Ethernet, possibilidade de se conectar à rede Wifi e Bluetooth. Pode-se observar na figura 8 que o modelo tem micro entradas HDMI e diferentes tamanhos de memória RAM disponibilizados.

Figura 8 - Raspberry Pi 4 Modelo B



Fonte: RASPBERRY FOUNDATION (2022)

A alimentação elétrica do Raspberry Pi é feita por meio de uma fonte de cinco volts e três amperes ligada a um conector MicroUSB, idêntico ao encontrado nos celulares, levando a um baixo consumo energético em comparação a um dispositivo maior como um computador *desktop* tradicional (Eberman et al, 2017).

Em relação ao sistema operacional, o Raspberry possui diferentes modelos sendo o mais avançado e comumente usado o Raspbian. Este é uma derivação do sistema operacional Linux que foi projetado e compilado para a plataforma Raspberry Pi. O Raspbian oferece mais de 35.000 pacotes de software, que estão pré-compilados para serem facilmente instalados nesta distribuição de sistema operacional. O Raspbian também é otimizado para obter o desempenho máximo do Raspberry Pi. Então, o Raspbian basicamente contém o ambiente de *desktop*, o gerenciador de

janelas OpenBox, o navegador Midori, ferramentas para desenvolvimento de software e código fonte de exemplo (RASPBERRY FOUNDATION, 2022).

2.4.2. ESP32 LoRa WiFi

Esta placa de tamanho 25x52x10mm, tem como base WI-FI Kit 32 com chip SX1276, que é o modem remoto LoRA, 915 MHz frequência, com cerca de -148dBm de alta sensibilidade, saída de potência de +20 dBm, sua distância de transmissão, ou seja, distância de comunicação de área aberta, é de aproximadamente 3 Km). A Placa WiFi LoRa 32 (figura 9) é uma placa de desenvolvimento voltada a aplicações da Internet das Coisas (IOT). Ela foi desenvolvida pela empresa Heltec Automation e possui um ESP32, tecnologia LoRa, display OLED de 0,96 polegadas com resolução de 128x64 capaz de mostrar informações em tempo real ao projetista, além de contar com um controlador de carga e conector JST na parte inferior para uma bateria de lítio (Li-Ion ou Li-Po) de 3.7V e até 1000mAh para alimentação remota, sem a necessidade de fios.

Figura 9 - ESP32 LORA WIFI

Fonte: LILYGO (2022)

O Hardware é equipado com ESP32 WiFi, Bluetooth Low Energy e processador Tensilica LX6 Dual Core operando com até 240Mhz, trabalhando junto ao transceptor LoRa SX1276 e Display OLED. A interface USB/Serial se dá através do chip CP2102 da Silicon LAB. Por essas características, a placa é ideal para desenvolver inúmeras aplicações de automação residencial, industrial, rural, sistemas de localização, infraestrutura de serviços, controle remoto de sensores e atuadores, entre outras muitas aplicações voltadas à Internet das Coisas e M2M.

2.4.3. GPS NEO 6M

O NEO-6M é um pequeno módulo capaz de se comunicar com um Sistema Global de Navegação por Satélite (GNSS). Esse módulo GPS é capaz de informar a localização exata do objeto em que está instalado, fornecendo dados com a latitude e longitude, data, hora e velocidade de deslocamento.

Este módulo é construído para ser utilizado com interface Serial de 3,3V, não sendo compatível com sinais 5V, que podem danificar o módulo, para utilização com 5V é necessário utilizar corretamente algum tipo de conversor de nível lógico.

2.5. CLOUD

O termo computação em nuvem, segundo Taurion (2009), surgiu em 2006 em uma palestra de Eric Schmidt, da Google, sobre como sua empresa gerenciava seus data centers. Hoje, a computação em nuvem, se apresenta como o cerne de um movimento de profundas transformações do mundo da tecnologia. A nuvem é uma representação para a internet ou infra-estrutura de comunicação entre componentes arquiteturais, baseada em uma abstração que oculta a complexidade da infra-estrutura. Cada parte desta infraestrutura é provida como um serviço, e estes serviços são normalmente alocados em data centers, utilizando hardware compartilhado para computação e armazenamento (SOUSA, 2009).

Para Buyya (2008) uma nuvem é um modelo de sistema paralelo e distribuído que consiste de uma coleção de computadores virtualizados e interconectados que são provisionados de forma dinâmica e apresentados como um ou mais recursos computacionais unificados. Estes recursos são disponibilizados e controlados através de acordos relacionados aos serviços que são estabelecidos entre um prestador e um consumidor sendo definidos a partir de negociações entre as partes.

Entre as propriedades da nuvem tem-se: eficiência de recursos, elasticidade, serviços de autogestão, acessibilidade e alta disponibilidade. A eficiência de recursos refere-se ao agrupamento da computação e rede para fornecer serviços a vários usuários. A alocação de recursos é adaptada dinamicamente de acordo com a demanda. A elasticidade é relativa aos recursos computacionais que podem ser rapidamente provisionados para escalar verticalmente ou reduzir de acordo com a demanda do consumidor. Os serviços de autogestão são a possibilidade do consumidor fornecer serviços em nuvem, como aplicativos da web, servidores tempo, processamento, armazenamento e rede conforme necessário e automaticamente sem a necessidade de humanos interação com o

provedor de cada serviço (TAURION, 2009). Acessibilidade e alta disponibilidade significa que os recursos de nuvem estão disponíveis na rede a qualquer momento e em qualquer lugar e são acessados por meio de mecanismos que promovem o uso por diferentes tipos de plataforma (por exemplo, telefones celulares, laptops e PDAs).

2.5.1. Exemplos de Cloud

A escolha da *cloud* depende da finalidade e escalabilidade do projeto. Itens como tipos de dados a serem enviados e disponibilidade de desenvolvimento de aplicações devem ser considerados. No quadro 1 é possível observar as 8 melhores ferramentas *cloud* para IoT em 2022 segundo MUTS et al:

Quadro 1 - Melhores plataformas para usar em projetos IoT.

Plataformas	**Interoperabilidade (protocolos de comunicação)**	**Função principal**	**Principais casos de uso**
Google IoT	HTTP MQTT	• Google Cloud IoT Core • Conectividade • Gerenciamento de dispositivo	• Energia • Transporte e logística
Amazon Web Services IoT Platform (AWS)	HTTP MQTT WebSockets	• AWS IoT Core • Conectividade • Autenticação • Mecanismos de regras • Ambiente de desenvolvimento	• Cidades inteligentes • Casas conectadas • Agricultura
Microsoft Azure IoT	MQTT AMQP WebSockets HTTPS	• Azure IoT Hub • Conectividade • Autenticação • Gerenciamento de dispositivo • IoT Edge	• Saúde • Manufatura
Oracle IoT Intelligent Applications	HTTP MQTT	• Oracle IoT Cloud Service • Conectividade • Monitoramente de ativos • Análise de dados • Integração de dados	• Automação industrial • Manutenção preventiva • Logística

Fonte: Adaptado de MUTS et al (2021)

2.5.2. Amazon web services IoT platform (AWS IoT)

A Amazon oferece um extenso pacote de serviços de IoT em nuvem, onde as principais vantagens são infraestrutura comprovada e escalável que suporta bilhões de dispositivos e trilhões de mensagens. Os usuários podem usar a plataforma Amazon IoT para diferentes fins: industrial, doméstico e comercial.

Os principais recursos desta plataforma são:

- Amplo conjunto de ferramentas de IoT, desde o end-device até a nuvem;
- Segurança em várias camadas, incluindo criptografia e controle de acesso;
- Excepcionais integrações de IA para soluções 2x mais rápidas;
- Uma plataforma escalável.

O AWS IoT Core permite conexões fáceis e seguras entre dispositivos e sua interação com aplicativos em nuvem. O suporte a dispositivos IoT Core pode processar mensagens e ajudar a acompanhar todos os dispositivos sem a necessidade de gerenciar qualquer infraestrutura.

O AWS IoT Device Management simplifica a organização e o monitoramento de uma grande frota de dispositivos de forma segura e em escala com a capacidade de solucionar problemas de funcionalidade do dispositivo.

O AWS IoT Device Defender usa as melhores práticas de segurança para controlar a autenticação e autorização do dispositivo, garantir a identidade do dispositivo e criptografar seus dados. Com o sistema operacional FreeRTOS para microcontroladores, é possível controlar e gerenciar os *end-devices* pequenos e de baixo consumo de energia e conectá-los a outros serviços de nuvem da AWS ou localmente a outros dispositivos.

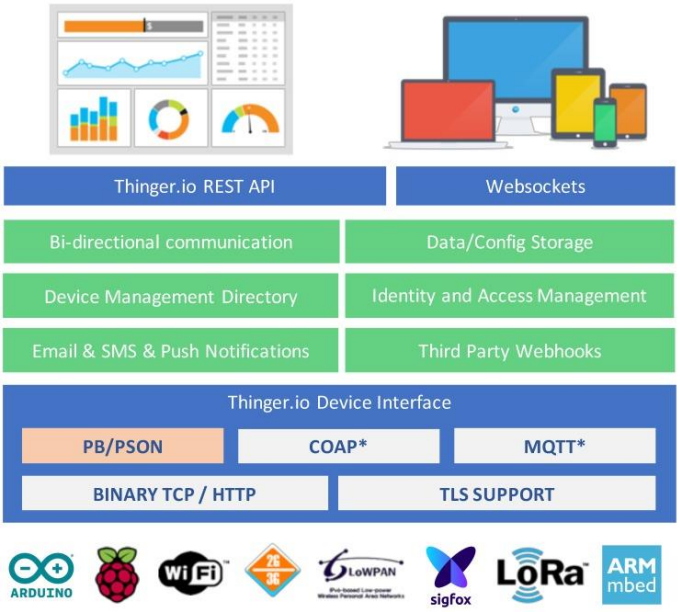
O AWS IoT Greengrass ajuda no armazenamento local, sincronização e gerenciamento de dispositivos conectados com recursos de ML . O AWS IoT Analytics permite a análise automática de uma grande quantidade de dados de diferentes tipos de dispositivos para insights mais sofisticados e precisos e casos de uso de ML.

Em relação a custos de utilização da ferramenta não há taxas mínimas, o processo de cobrança depende das mensagens enviadas e recebidas. Quanto mais mensagens, mais barato é o plano. Uma avaliação gratuita está disponível por 12 meses e permite 2.250.000 minutos de conexão e 500.000 mensagens.

2.5.3. Thinger.io

Nesta seção, descreve-se a arquitetura geral de uma plataforma Open Source para implantação de aplicativos de fusão de dados integrando tecnologias de Big Data, Cloud e IoT, o Thinger.io. Esta é uma nova plataforma que está despertando o interesse da comunidade científica/tecnológica e já encontram-se projetos em que esta plataforma tem sido utilizada com sucesso inclusive na área da educação. O Thinger.io fornece um serviço de nuvem pronto para conectar dispositivos à Internet e realizar qualquer sensoriamento remoto ou atuação pela Internet. Além disso, oferece um nível gratuito para conectar um número limitado de dispositivos, porém também é possível instalar o software fora da nuvem para um gerenciamento privado dos dados e dispositivos conectados à plataforma, sem qualquer limitação. Uma visão geral pode ser vista na figura 10.

Figura 10 - Visão geral Thinger.io



Fonte: Thinger.io (2022)

O principal benefício da utilização desta plataforma, além de ser open source, é a possibilidade de obter uma comunicação bidirecional com os dispositivos, em tempo real, utilizando interfaces padrão REST-API. Como tal, é possível desenvolver qualquer aplicação de análise de dados, ou seja, desktop, mobile, Webservice, que interaja com dispositivos usando uma interface bem conhecida e comprovada baseada em REST. Enquanto isso, os dispositivos podem

usar protocolos binários mais eficientes (em termos de largura de banda ou espaço de memória) para se comunicar com a nuvem.

Na abordagem HTTP, cada dispositivo que envia dados para a nuvem emite uma solicitação HTTP com uma carga útil personalizada. A plataforma IoT recebe essas informações, assim como um servidor web comum, e armazena essas informações em um banco de dados. Essa forma de comunicação resulta em uma forma ineficiente de transmissão de informações, em termos de largura de banda, latência e consumo de energia, que são restrições para dispositivos IoT .

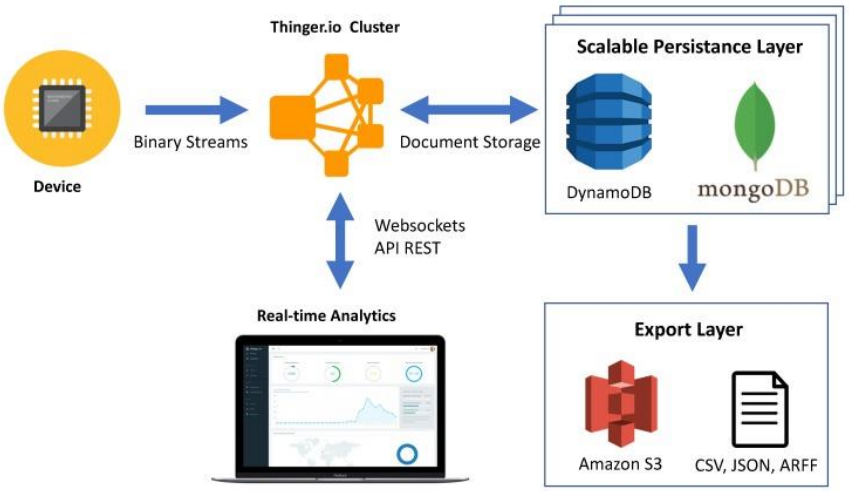
As plataformas baseadas em HTTP podem ser fáceis de desenvolver, implantar e manter, pois operam como servidores HTTP regulares recebendo dados de sensores usando tecnologia bem conhecida. No entanto, tais abordagens não podem fornecer um mecanismo eficiente para uma comunicação bidirecional, ou seja, quando é necessário atuar sobre um dispositivo ou configurá-lo em tempo real. As plataformas baseadas em HTTP contam com um mecanismo de pesquisa em que os dispositivos verificam o servidor periodicamente em busca de novos comandos ou atualizações. Enquanto isso, o Thinger.io fornece um canal de comunicação bidirecional entre o dispositivo e o servidor em nuvem, para que qualquer aplicativo possa interagir com o dispositivo em tempo real, aumentando o número de casos de uso que podem ser desenvolvidos com a plataforma.

2.5.3.1. Gerenciamento de armazenamento

Uma das funcionalidades desta plataforma está relacionada com a gestão de armazenamento, a que se dá o nome de “Data Buckets”. Um bucket é um recurso de nuvem para armazenar dados de séries temporais. Um bucket de dados é um armazenamento de série temporal em que os dispositivos podem enviar informações quando necessário.

Cada ponto de dados é automaticamente marcado na nuvem no momento da recepção, pois os dispositivos IoT não lidam com um relógio em tempo real (RTC) sozinhos. Essas informações são armazenadas na nuvem em soluções seguras, eficientes e escaláveis (principalmente DynamoDB da Amazon Web Services). As informações armazenadas em um bucket podem ser exibidas em um painel dentro da interface do console ou exportadas em armazenamento escalável (como Amazon S3) em ARFF, JSON ou CSV para sua análise offline, conforme ilustrado na figura 11.

Figura 11 - Estrutura de dados Thinger.io



Fonte: Thinger.io, 2022

O armazenamento de informações em uma infraestrutura de nuvem escalável como o Thinger.io pode ser dividido nas seguintes etapas:

- **Recursos de dados de modelo no dispositivo:** Um dispositivo pode enviar um conjunto de recursos, como por exemplo temperatura ou umidade. Os recursos não estão vinculados a nenhuma estrutura de dados, pois são gerenciados como documentos não estruturados no formato JSON;
- **Conectar o dispositivo à infraestrutura do Thinger.io:** O dispositivo expõe os recursos disponíveis, que ficam imediatamente visíveis tanto para alimentação de dashboards em tempo real, quanto para armazenamento em buckets de dados;
- **Configuração de um bucket no console:** Uma vez que o dispositivo esteja conectado à plataforma, é possível definir um novo bucket de dados que é alimentado por um recurso do dispositivo. O usuário normalmente seleciona aqui o intervalo de amostragem necessário. Neste momento, o dispositivo começará a transmitir as informações na frequência necessária.

3. METODOLOGIA

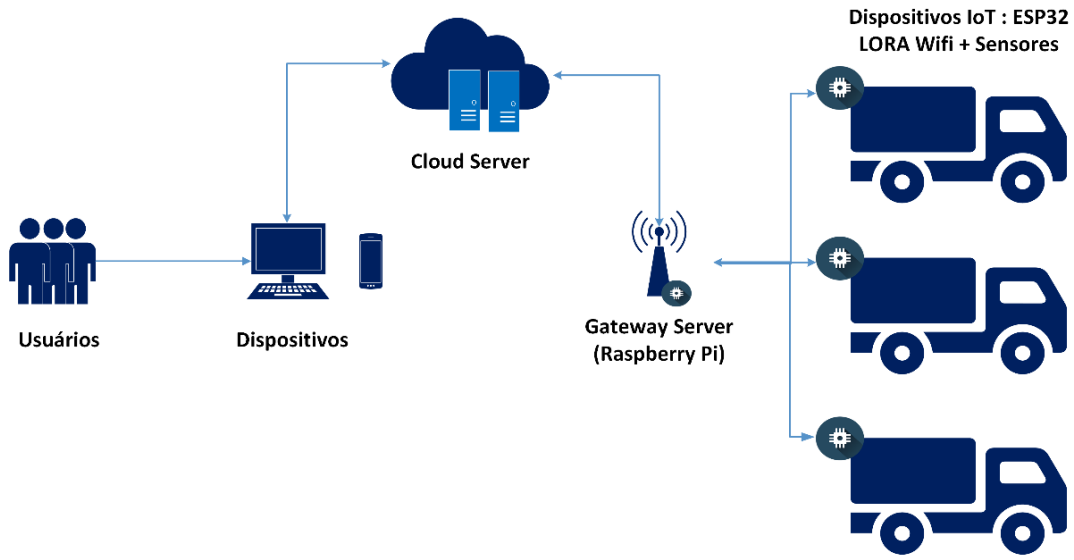
O trabalho apresentado é uma pesquisa aplicada visando a realização de Pesquisa Exploratória em base do material bibliográfico referente ao assunto. Foi utilizado o procedimento técnico de pesquisa bibliográfica e aplicada. O método de abordagem utilizado foi o hipotético-dedutivo e o método de procedimento de elaboração foi o monográfico. A coleta de dados foi feita através da observação direta intensiva e documentação indireta, sendo estes dados qualitativos e interpretados de forma global.

Foram realizadas pesquisas bibliográficas na integração das tecnologias para transmissão e manipulação de dados coletados. As pesquisas bibliográficas foram realizadas para compilar informações sobre o estado da arte e as técnicas mais recentes e que sejam viáveis para a elaboração do projeto. Estes dados servirão de embasamento para futuros testes de performance e prototipação do projeto.

O referido projeto consiste na implementação de uma estrutura de monitoramento de frotas de cargas refrigeradas. Para tanto, escolheram-se o Raspberry, que será utilizado como *gateway* e Esp32, que foi utilizado como *end-device* por isso a primeira fase foi implementar um sistema de conexão entre os hardwares. Inicialmente tentar-se-á utilizando-se do LORAWAN, porém caso seja observado dificuldades no que diz respeito ao envio de dados, o projeto se encaminhará para outra forma de comunicação para que os dados possam ser transferidos e processados no Raspberry, os scripts desenvolvidos serão em C no ESP e em Python no Raspberry. Os dados citados foram randomizados somente para teste inicial, não necessitando de sensores conectados fisicamente no ESP32. Durante esta fase foram definidas todas as variáveis de controle a serem enviadas.

As partes do sistema foram desenvolvidas separadamente (vide figura 12), porém com sucessivos testes de integração a fim de facilitar o andamento do projeto. A segunda etapa foi a integração entre Raspberry e *Cloud*, onde os dados foram coletados, armazenados e processados. Nesta fase foi definido o tipo de conexão do Raspberry com a Internet (Wi-fi ou através de cabo *Ethernet*), a alimentação dos hardwares e a padronização de alguns aspectos como o intervalo de envio além da modelagem do armazenamento de dados.

Figura 12 - Esquema de funcionamento do projeto



Fonte: própria (2022)

A terceira etapa consiste na criação de uma API para melhor visualização dos dados. Como seguramente uma das variáveis transmitidas é a localização, uma aplicação com mapas interativos que mostre onde encontra-se um dos veículos quando requisitado será desenvolvida. O modelo de API será o de API REST que usa objetos JSON para passar dados.

A última etapa foi a de testes sistêmicos. Neste cenário o projeto será avaliado em termos de precisão, em diversas situações. Os resultados foram discutidos e analisados entre si, com o objetivo de encontrar as circunstâncias mais favoráveis e identificar oportunidades de melhoria.

3.1. MATERIAIS UTILIZADOS

Os materiais utilizados para o experimento foram:

- Um ESP32 LORA WIFI + NEO 6M;
- Um Raspberry Pi 4;
- Sensor de temperatura e umidade DHT11;
- Jumpers e cabo USB.

Em relação aos softwares empregados têm-se:

- IDE Arduino para programação do ESP 32;
- Ferramenta de texto do Raspberry para edição dos códigos em pyton;
- Compilador python3 do Raspberry (pré-instalado);

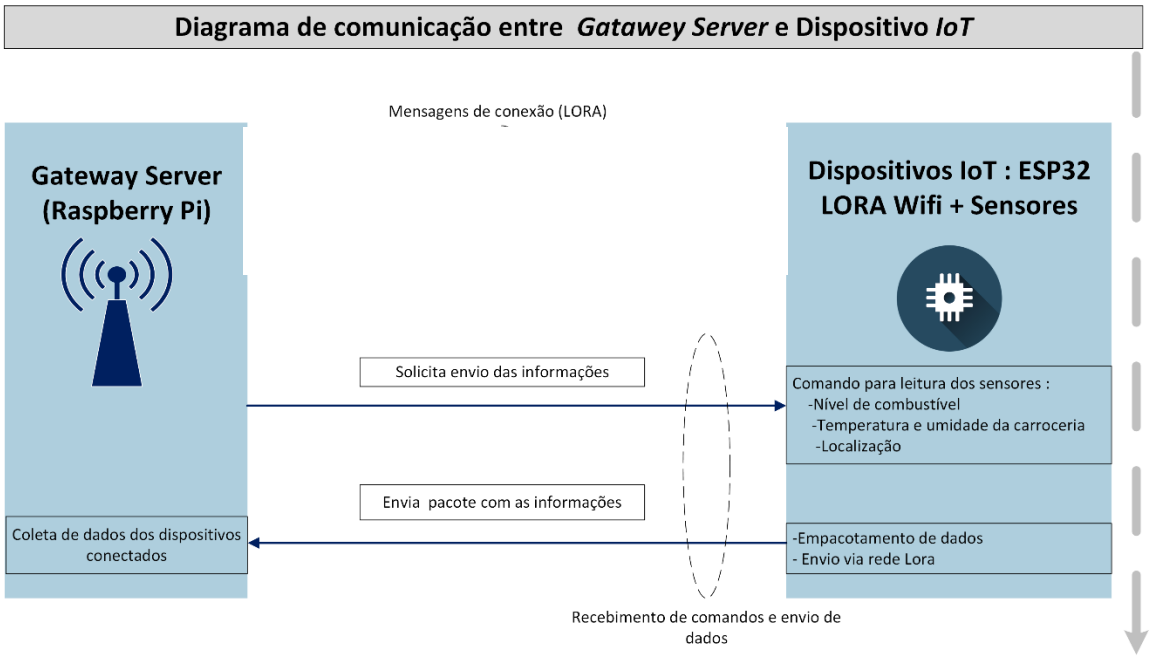
- API Thinger.io .

3.2.CÓDIGO ESP 32

Pode-se dizer que o código do ESP32 LORA WIFI divide-se em duas partes. A primeira refere-se à obtenção de dados do GPS e a segunda se refere a leitura do sensor DHT11. Em relação ao GPS utilizou-se uma biblioteca padrão chamada *TinyGPS++* (vide ANEXO A) que possui recursos para obtenção de forma otimizada dos parâmetros do NEO 6M como latitude, longitude e velocidade terrestre. A utilização não somente desta biblioteca, mas também para leitura de dados de GPS de quaisquer outra forma nesta placa, presupõe o uso dos pinos 34 e 12 como RX/TX além de *baudrate* de 9600 (vide ANEXO A) . O DHT11 por sua vez não possui complexidade em seu uso porém deve-se atentar ao pino utilizado como *signal* para que as informações cheguem à placa. É importante destacar que estes códigos foram desenvolvidos utilizando a IDE do Arduíno.

A comunicação entre o endpoint e o Raspberry pretende seguir o esquema apresentado na da figura 13:

Figura 13 - Diagrama de comunicação entre Raspberry e ESP32



Fonte própria (2022)

3.3. CÓDIGO RASPBERRY PI

De todas as funções utilizadas a mais importante e crucial para o funcionamento do sistema é a *post_request* (vide ANEXO B). Sua chamada é feita indicando os valores recebidos do *end-*

device além das métricas de sistema do Raspberry Pi como temperatura de cpu, uso da cpu, memória RAM, etc.

Utilizou-se o método *subprocess* para realizar o POST dos valores obtidos (vide ANEXO B). Este método escreve o comando preterido no prompt de comando do linux. Como o programa está em loop e por dificuldades de iteração que serão explicados a seguir, uma variável auxiliar json foi criada para que se obtivesse os dados de maneira correta.

As outras funções são de cunho de análise de processamento e sistema local que utilizam-se de informações armazenadas pelo próprio Raspbian em arquivos específicos. Então resumidamente cada uma das funções de análise do Raspberry acessa um arquivo (vide ANEXO B) e coleta informações sobre o uso da cpu, temperatura da placa, etc e guarda em uma vari9ável que depois é alocada no método POST e enviada à API.

3.4. ESPECIFICAÇÃO DE AMBIENTE DE TESTE

Inicialmente prospectou-se a utilização do LORA para comunicação entre Raspberry e ESP32 porém o módulo apresentava erro na captação de dados do GPS quando utilizado em concomitância com o LORA, chegando a reiniciar inúmeras vezes durante os testes de maneira não controlada. Depois de extensa pesquisa descobriu-se que erros semelhantes foram encontrados tanto em dispositivos da Heltech quanto da Ttgo e geralmente eram associados à alimentação insuficiente ou má fabricação da placa. Devido ao tempo disponível para entrega da dissertação optou-se por comunicação serial entre Raspberry e ESP32.

Os testes foram realizados em ambiente controlado onde tanto ESP32 quanto o Raspberry usufruíam de alimentação constante e proteção contra agentes externos. A API Thinger.io foi utilizada para servir de *front-end* do sistema. A forma de realizar a integração entre o device e a API é facilmente encontrada na documentação do Thinger.io e não será explorada profundamente neste trabalho.

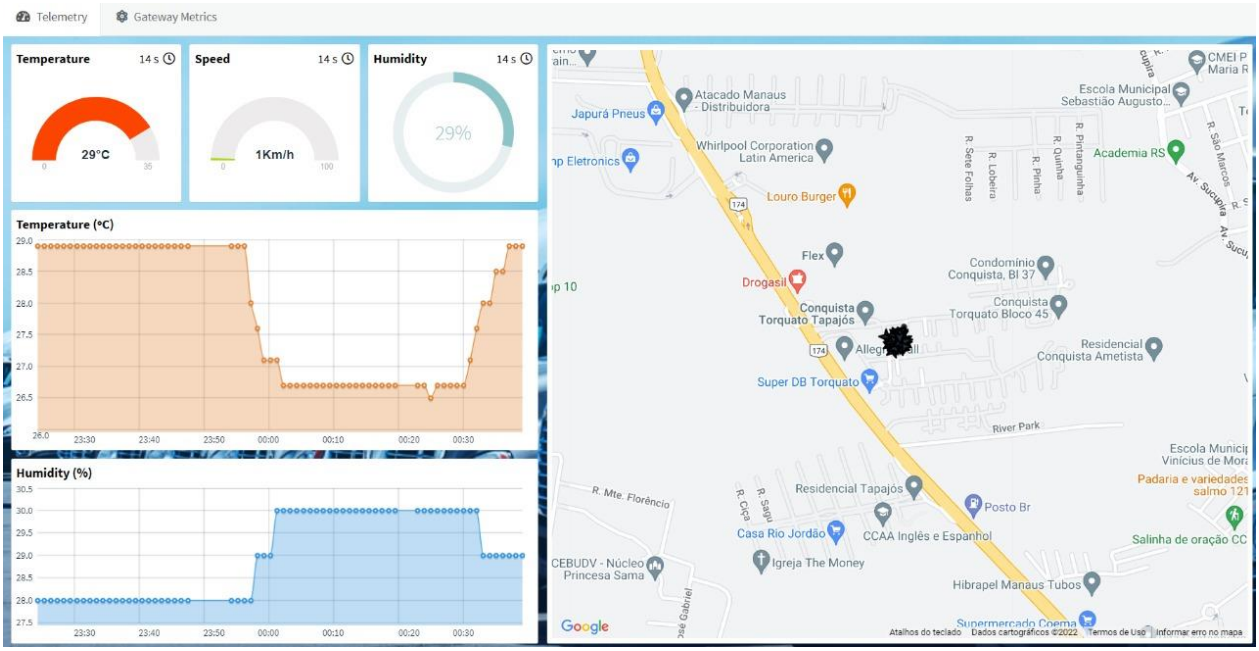
4. RESULTADOS E DISCUSSÕES

Nesta seção serão apresentados os resultados da proposta apresentada. Ela está dividida em duas partes: a primeira denominada Dashboard desenvolvido mostra a solução a nível de front-end ao qual chegou-se, explicando objetivamente os aspectos do mesmo como por exemplo as variáveis a serem visualizadas; a segunda parte é mais extensa e aborda análises do desempenho do sistemacomo um todo, apontando falhas e sugestões de melhorias além de relatar os sucessos obtidos.

4.1.DASHBOARD DESENVOLVIDO

Parte da solução é mostrar de forma simplicada os dados coletados pelos sensores para que aqueles com acesso a esses dados possam tomar decisões a respeito do transporte de carga. Como mencionado anteriormente os dados coletados são: temperatura e umidade da parte traseira do caminhão; velocidade aproximada do caminhão; dados de GPS como latitude e longitude. O dashboard desenvolvido pode ser observado na figura 14.

Figura 14 - Dashboard inicial do sistema



Fonte própria (2022)

Para melhor visualização optou-se por um diagrama em que temperatura e umidade pudessem ser vistas no seu estado atual e sua variação com o tempo, pois essas são as variáveis mais importantes se tratando do controle do ambiente da carga; além disso, a localização e

tragetória também são mostrados no mapa de forma destacada; a velocidade também é mostrada porém o seu histórico não é de tanta relevância quanto as outras variáveis.

4.2.TESTE DE DESEMPENHO DO SISTEMA

O teste de desempenho do sistema avalia funcionalidade do sistema de maneira integral passando e identificando possíveis melhorias no processo como um todo. Para maior entendimento dividiu-se a análise em quatro partes: análise do end-point, análise do Gateway Raspberry, análise de Cloud e análise geral.

4.2.1. Análise do end-device

Como esperado, o ESP32 LORA WIFI mostrou-se de fácil aplicabilidade e integração com a plataforma Arduino necessitando apensas de algumas bibliotecas para a utilização de alguns de seus componentes. Uma característica negativa encontrada foi a não possibilidade de funcionamento do LORA e NEO-6M ao mesmo tempo, depois de sucessivas tentativas percebeu-se que o problema era devido à alimentação. Foi providenciada bateria externa para alimentação do mesmo porém a concomitância destas duas funcionalidades não foi possível, por esse motivo o projeto inicial que envolveria comunicação LORA entre o ESP e o Raspberry foi modificado. Problemas parecidos a esse foram relatados tanto em dispositivos da Heltec quanto da Ttgo contudo não é um episódio recorrente, e sim uma excepcionalidade.

Os dados enviados do *end-device* conforme ANEXO A contém a seguinte forma : latitude, longitude, velocidade, umidade, temperatura, b.

É explícito quais informações cada variável guarda porém é visto ao final o char “b”. Esse char foi utilizado para demarcar o final da leitura de dados. Além disso as informações só são enviadas ao Raspberry se e somente se a latitude e longitude forem valores válidos. Essa escolha foi determinada durante os testes de integração e fez-se necessária pois observou-se demora na coleta dos dados de geolocalização, visto que o NEO-6M tem que se conectar com os satélites mais próximos e iniciar uma sessão de envio e recebimento de dados, e geralmente esse processo demora de alguns segundos a minutos e ocorre a cada reinicialização do módulo.

4.2.2. Análise Raspberry Gateway

O termo *gateway* pode ser classificado como “portal” ou “portão”. Ele é considerado uma passagem entre dois ambientes distintos. Ou ainda, em outras palavras, um sistema ou equipamento encarregado de estabelecer a comunicação entre duas redes. Tendo este conceito em vista, o Raspberry ao funcionar como ponte entre os *endpoints* e a rede atua como um gateway porém um visão sistemática mais ampla classificaria como a ponta de um nó. Os testes realizados utilizaram um Raspberry Pi 4B conectado à rede via cabo Ethernet, as pré-configurações do Raspberry podem ser modificadas de modo que o mesmo conecte-se à rede deste que a mesma possibilite tal configuração.

O código foi desenvolvido para comunicar o ESP32 com o Raspberry Pi 4 via serial. O ESP32 como *endpoint* que envia dados de latitude, longitude, velocidade, umidade e temperatura. O raspberry por conseguinte processa esses dados internamente e adiciona as métricas de desempenho do hardware ao pacote.

Em termos práticos não observou-se dificuldades na comunicação serial entre os *hardwares.* Em relação à comunicação com a API, o método *subprocess* não suportou as iterações no loop, sendo necessário criar uma variável json para que a mesma pudesse ser alterada a cada envio de novos dados. Isso tornou a requisição “verbosa” e caso replicado em escala pode levar à perda de desempenho por parte do Raspberry e erros na requisição POST. Para tornar o projeto escalonável deve-se buscar outras maneiras de solicitar o envio de variáveis como o uso da biblioteca *requests* .

4.2.3. Análise Cloud

A ferramenta Thinger.io provou ser uma poderosa ferramenta no monitoramento de dispositivos IoT. A sua interface gráfica é de fácil usabilidade além de contar com ferramentas de edição que propiciam a utlização do Google Maps, gráficos em tempo real, botões virtuais, etc. Como foi utilizada a versão livre, a atualização dos dados ocorreu a cada 60s o que, para uma proposta de informações em tempo real, não seria suficiente não obstante, a provedora disponibiliza pacotes de uso em que os dados podem ser atualizados em tempo real.

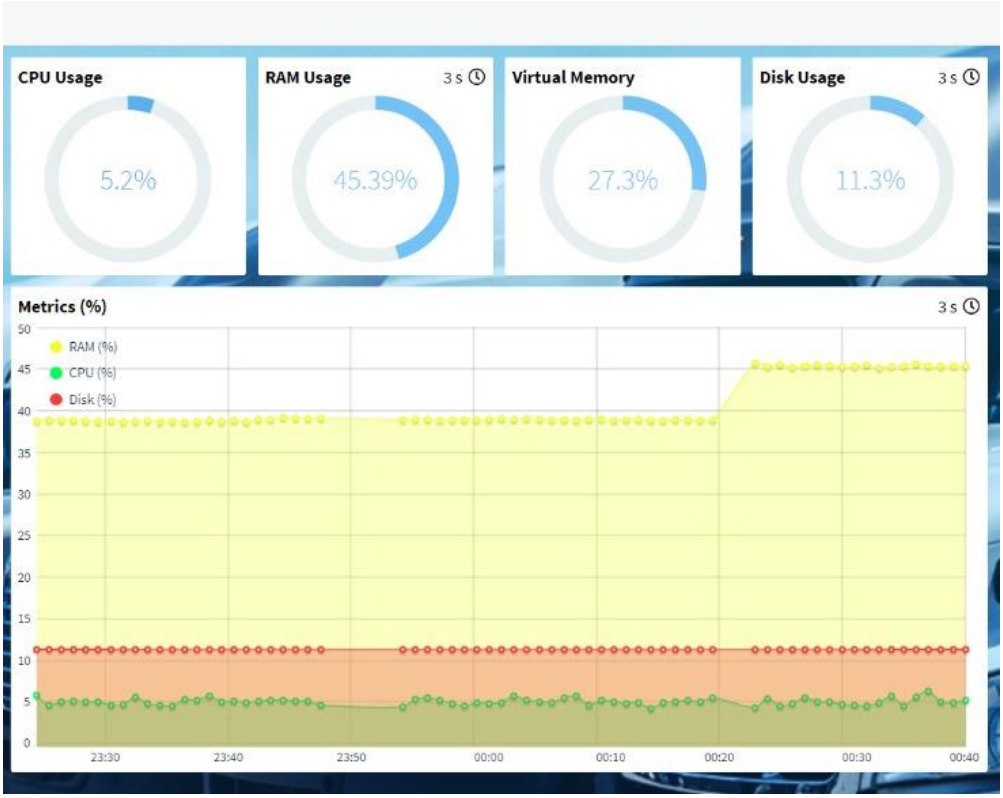
O banco de dados de fácil acesso auxiliou notoriamente no desenvolvimento da solução proposta. Um dos testes de integração realizados gerou dados randomizados durante 6 hrs e não

houve queda na comunicação com a API devido a servidor e, quando a queda foi estimulada pelo cliente,ou seja, deixou-se de alimentar o dispositivo IoT, o mesmo volta a se comunicar com a API ao estar *online* . Porém, caso essa queda excedesse o tempo de 2hrs a conexão era finalizada pelo servidor, isso deve-se ao fato da conta utilizada ser uma versão livre pois a versão paga não possui tempo limite quando o device está no modo ocioso.

4.2.4. Análise Geral

Como visto no tópico 4.2.3, o tempo estipulado pela *cloud* para o envio de dados para API foi de 60s, dentro deste tempo o end-device pode enviar dados porém somente o último enviado será lido pela API. Vários testes de integração foram realizados, a seguir será descricionado um daqueles feitos em *thread*, onde duas ou mais tarefas realizam-se ao mesmo tempo, no caso abordado o Raspberry recebeu dados de dois *endpoints* e enviou os dados para a API. As métricas do Raspberry podem ser observadas na figura 15.

Figura 15 - Métricas do rendimento do Raspberry

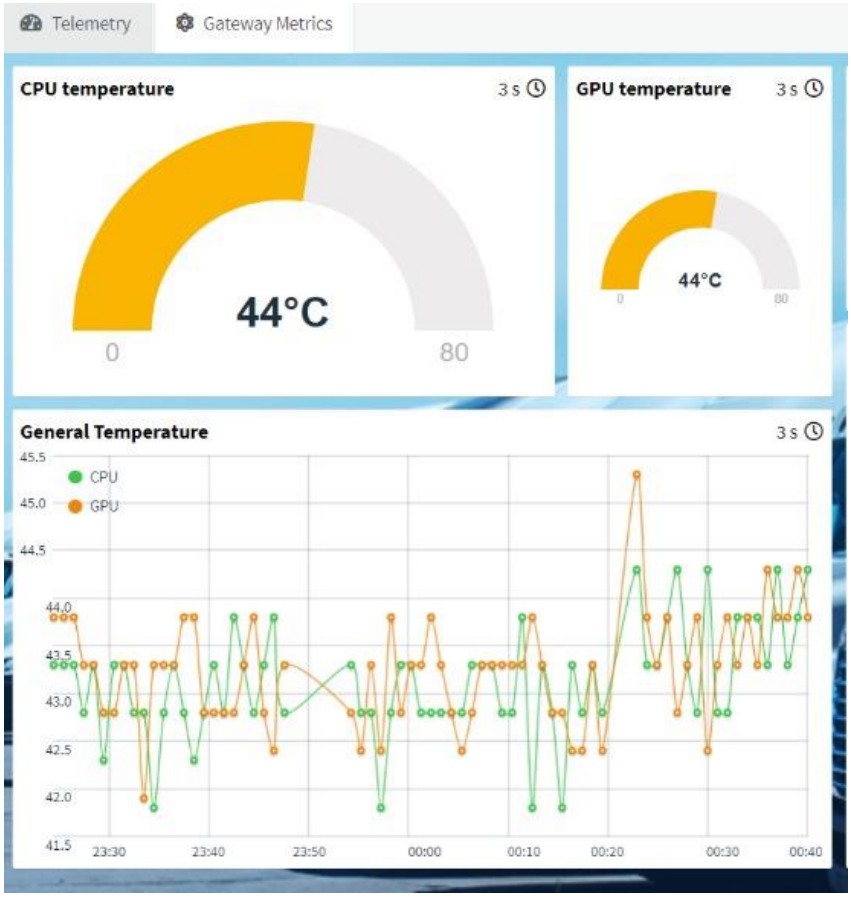


Fonte própria (2022)

Entre 00:20 e 00:30 foram randomizados dados para que fosse simulado o envio de dois pacotes de *end-devices* diferentes, esse comportamento influenciou em alguns parâmetros de funcionalidade. Notou-se que o envio de dados para API consome em torno de 5% da capacidade de CPU do Raspberry independentemente se um ou dois *end-devices* estão conectados a ele. Além disso, o consumo de memória RAM subiu de aproximadamente 39% para 45,39% mostrando aumento na quantidade de dados armazenados.

Em relação à temperatura da GPU e CPU, a inclusão de um *end-device* faz a mesma variar de aproximadamente 42,75°C para 44,5°C (vide figura 16), indicando aumento de atividade no processador sendo necessário a inclusão de um *cooler* e dissipadores para não danificar a placa por superaquecimento.

Figura 16 - Métricas de temperatura



Fonte próporia (2022)

5. CONCLUSÃO

O presente projeto de pesquisa se justifica pois sua implementação torna uma cadeia de abastecimento mais eficiente, assim, a entrega de mercadorias, do produtor ao cliente final, acontecerá no tempo combinado e nas condições de carga especificadas. A utilização de uma plataforma de código aberto para modelagem de aplicativos baseados em tecnologias de IoT facilitará a implantação de novas funcionalidades em diferentes campos, incluindo casas e edifícios inteligentes, sistemas veiculares e transporte, sistemas médicos e de saúde, infraestruturas inteligentes, redes elétricas e, no caso deste projeto, no sistema logístico.

Ao desenvolver deste trabalho, elencou-se quatro objetivos diferentes. O primeiro foi conectar o *end-device* e o Raspberry de forma que os dados possam ser transferidos. Isso foi possível pois a conexão entre ambos deu-se via cabo serial.

O segundo tratava-se de integrar o Raspberry em algum serviço de *Cloud.* Com o auxílio da documentação Thinger.io os dados referentes aos sensores e estado do próprio Raspberry foram coletados, armazenados e processados na nuvem.

O terceiro objetivo era criar uma API para melhor visualização dos dados. A ferramenta Thinger.io forneceu uma gama de ferramentas gráficas para melhor visualização dos dados armazenadois no banco dedados, graças a tais ferramentas foi possível criar dashboards de monitoramento não somente dos *endpoints* (que simbolizam os caminhões) mas também do Raspberry.

O último objetivo era realizar testes sistêmicos. Como mencionado no tópico 4.2.4 foi realizado um levantamento geral da funcionalidade do Raspberry quando houve o envio de dados (neste teste foram dados randomizados) de duas fontes de informação. Assim, ao percorrer os objetivos específicos percebe-se que os mesmos foram gradualmente aferidos de acordo com o avanço do projeto.

Utilizaram-se os conhecimentos adquiridos na universidade para que, mediante o uso da tecnologia da IoT ao longo de todo o processo, dar seguimento em tempo real a cada fase logística do transporte de cargas refrigeradas, seja medindo o estado da temperatura da carga, seja informando a localização da mesma em tempo real. Portanto, o método sugerido continuará impulsionando a rapidez e a eficiência através de processos automatizados que reduzem tempo e custos de distribuição.

REFERÊNCIAS BIBLIOGRÁFICAS

AGUILERA, Luiz Manoel; GIMENEZ, Claudemir; BACIC, Miguel Juan. **Sistemas de gerenciamento de transportes–estudo de caso**. In: Simpósio de Engenharia de Produção, v. 10, Campinas-SP, 2003.

ARAÚJO, Alan Kilson Ribeiro. **Internet das coisas: análise das questões chave de aplicação pública no Brasil**. Dissertação de Mestrado Apresentado ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Paulista, São Paulo, 2017.

BOR, Martin; VIDLER, John Edward; ROEDIG, Utz. **LoRa for the Internet of Things**. Departamento de Computação e Comunicações da Universida de Lancaster, Inglaterra, 2016.

BUYYA, R., YEO, C. VNUGOPAL, S. **Market-oriented cloud 41 computing: Vision, hype, and reality for delivering it services as computing utilities**. Universidade de Melbourne, Australia 2008.

CALAZANS, Flávio Mendonça et al. **Gestão de Frotas no Transporte Rodoviário de Carga**. In: Simpósio de excelência em gestão e tecnologia, v. 11, p. 1-16, 2014.

CHAGAS, E. **A influência da internet das coisas no desenvolvimento de produtos e serviços das operadoras de telecomunicações no Brasil** . Dissertação de doutorado apresentado à Fundação Getulio Vargas, Escola de Administração de Empresas, São Paulo, 2017.

DA CRUZ, J. W. V., OLIVEIRA, M. V. R., DA COSTA SILVA, J. I., DA COSTA SILVA, R., SOUSA, R. C., DA SILVA, H. J., e JÚNIOR, J. J. H. F. **Uma Solução Internet das Coisas para Monitoramento de Gases Poluentes na Amazônia Legal.** In: Anais SULCOMP, 9, 2018.

EBERMAN, E., PESENTE, G., RIOS, R. O. e PULINI, I. C. **Programação para leigos com Raspberry Pi** . João Pessoa: Editora IFPB, 2017.

EGER, Fabiano. **Monitoramento de caminhões usando Internet das Coisas**. Trabalho de conclusão de curso apresentado ao Centro Universitário da Católica de Santa Catarina. Joinville-SC, 2017.

EVANS, Dave. **The Internet of Things How the Next Evolution of the Internet Is Changing Everything**. Cisco IBSG, p. 5-9, 2011. Disponível em: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. Acessado em: 27/03/2022.

EZECHINA, M. A., OKWARA, K. K. e UGBOAJA, C. A. U. **The Internet of Things (Iot): a scalable approach to connecting everything**. In: The International Journal of Engineering And Science, 4(1), 09-12, 2015.

FEITOSA, Ana Paula Pratti; PEZZIN, Matiele Oliveira. **Como a tecnologia impacta na melhoria do gerenciamento de frotas de uma empresa de transporte**. Trabalho de conclusão de curso apresentado à Faculdade Doctum de Serra. Serra-ES, 2019.

GOKHALE, Pradyumna; BHAT, Omkar; BHAT, Sagar. **Introduction to IOT. In:** International Advanced Research Journal in Science, Engineering and Technology, v. 5, n. 1, 2018.

LI, S., XU, L. e ZHAO, S. **The internet of things: a survey**, v. 17, n. 2, 2015.

LILYGO. **LILYGO® TTGO T-Beam V1.1 ESP32 433/868/915/923Mhz WiFi Wireless Bluetooth Module GPS NEO-6M SMA LORA 32 18650 Battery Holder With SoftRF**. Acesso em: 18/09/2022.

LORAWAN STANDARD. In: SEMTECH. Disponível em: https://www.semtech.com/lora/lorawan-standard. Acesso em : 03/05/2022.

LUCAS, Á. M. **Análise da implantação de um sistema de monitoramento da produção de fita adesiva contemplando os conceitos da Indústria 4.0**. Trabalho de conclusão de curso apresentado à Universidade do Estado do Amazonas. Manaus, 2020.

MAÇADA, Antonio Carlos Gastaud; FELDENS, Luis Felipe; SANTOS, Andre Moraes dos. **Impacto da tecnologia da informação na gestão das cadeias de suprimentos: um estudo de casos múltiplos.** In: Revista Gestão e produção, v. 14, n. 1, p. 1-12, São Carlos -SP 2007.

MUTS, Ivan; ROSIL, Maria. **8 Best IoT Cloud Platforms in 2022**. Euristiq, 2021. Disponível em: https://euristiq.com/8-best-iot-cloud-platforms/. Acesso em: 02 /05/2022.

RASPBERRY FOUNDATION. **Raspberry Pi**. Disponível em: https://www.raspberrypi.org. Acesso em: 05/02/2022.

SANCHEZ-IBORRA, Ramon, SANCHEZ-GOMEZ, J., BALLESTA-VIÑAS, J., CANO, M. D., e SKARMETA, A. F. **Performance evaluation of LoRa considering scenario conditions**. **Sensors**, v. 18, n. 3, p. 772, Espanha, 2018.

SILVA, A. S. D. **Desenvolvimento de sistema para controle e monitoramento remoto de um forno industrial de uma empresa do polo de duas rodas do Distrito Industrial**. Trabalho de conclusão de curso apresentado à Universidade do Estado do Amazonas. Manaus, 2021.

SOUSA, F., MOREIRA, L., MACHADO, J. **Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios**. In: Antônio Costa de Oliveira;Raimundo Santos Moura;Francisco Vieira de Souza. (Org.). III Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI). 1 ed. Teresina: SBC, 2009, v. 1, p. 150-175.

SURESH, Priya, et al. **A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment**. 2014 International conference on science engineering and management research (ICSEMR). Chennai, India, 2014.

TAURION, Cezar. **Cloud computing - computação em nuvem**. Brasport, 2009.

THINGER.IO. **Overview**. Acesso em: 18/09/2022.

TRINDADE, Yan et al. **Desenvolvimento de Aplicação IoT utilizando a Plataforma Integrada Softway4IoT e Fiware**. In: Anais do VI Workshop do testbed FIBRE. SBC, 2021. p. 47-56.

ANEXO A – CÓDIGO FONTE DO ESP 32 LORA WIFI

```
/*
        Amazonas State University
        Final project: Development of IoT System for Refrigerated Cargo Transport Monitoring
        Code: ESP 32 LORA wifi module code
        Created By  : Maykon Henrique Ferreira da Silva
        Created Date: 11/09/2022
        version ='1.0'
        ---------------------------------------------------------------------
        UNIVERSIDADE DO ESTADO DO AMAZONAS
        TCC2: Desenvolvimento de Sistema IoT para Monitoramento de Transporte de Carga
Refrigerada.
        Código: Módulo ESP 32 LORA wifi
        Autor  : Maykon Henrique Ferreira da Silva
        Data de criação: 11/09/2022
        versão ='1.0'
        --------------------------------------------------------------------- */

//ESP 32 LORA WIFI. Wifi doesn't work with serial usb communication.
//--------------------DHT11--------------
#include "DHT.h"
#define DHTPIN 14     // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11   // DHT 11

DHT dht(DHTPIN, DHTTYPE);

//--------------------GPS--------------
/*
   9600-baud serial GPS device hooked up on pins 34(rx) and 12(tx). */

static const int RXPin = 34, TXPin = 12;
static const uint32_t GPSBaud = 9600;

#include <TinyGPS++.h>
#include <SoftwareSerial.h>
// The TinyGPS++ object
TinyGPSPlus gps;

// The serial connection to the GPS device
SoftwareSerial ss(RXPin, TXPin);

void setup()
{
  Serial.begin(115200);
  ss.begin(GPSBaud);
 // Serial.println(F("Arduino.ino"));
```

```
 // Serial.println(TinyGPSPlus::libraryVersion());

  dht.begin();
}
void loop()
{

// This sketch displays information every time a new sentence is correctly encoded.
  while (ss.available() > 0)
    if (gps.encode(ss.read()))
      displayInfo();

  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
   // Serial.println(F("No GPS detected: check wiring."));
    while(true);
  }
 //ss.end();

}
void displayInfo()
{
  delay(5000);
  /*Serial.print(F("  Date/Time: "));
  if (gps.date.isValid())
  {
    Serial.print(gps.date.month());
    Serial.print(F("/"));
    Serial.print(gps.date.day());
    Serial.print(F("/"));
    Serial.print(gps.date.year());
    Serial.print(F("  "));
  }

  if (gps.time.isValid())
  {
    if (gps.time.hour() < 10) Serial.print(F("0"));
    Serial.print(gps.time.hour());
    Serial.print(F(":"));
    if (gps.time.minute() < 10) Serial.print(F("0"));
    Serial.print(gps.time.minute());
    Serial.print(F(":"));
    if (gps.time.second() < 10) Serial.print(F("0"));
    Serial.print(gps.time.second());
    Serial.print(F("."));
    if (gps.time.centisecond() < 10) Serial.print(F("0"));
```

```
    Serial.print(gps.time.centisecond());
   }
   Serial.print(F("  Location: "));
   */
   if (gps.location.isValid())
   {
    Serial.print("b,");
    //latitude,longitude,velocidade,umidade,temperatura,b
    //Serial.print(F("   Latitude: "));
    Serial.print(gps.location.lat(), 6);
    Serial.print(F(","));
    //Serial.print(F("   Longitude: "));
    Serial.print(gps.location.lng(), 6);
    Serial.print(F(","));
    //Serial.print(F("   Speed: "));
    Serial.print(gps.speed.kmph(), 6);
    Serial.print(F(","));
    control_temp_hum();
   }

   else
   {
    //Serial.print(F("INVALID"));
   }

   Serial.println();
}

void control_temp_hum(){
   // Reading temperature or humidity takes about 250 milliseconds!
   // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
   float h = dht.readHumidity();
   // Read temperature as Celsius (the default)
   float t = dht.readTemperature();
   // Read temperature as Fahrenheit (isFahrenheit = true)
   float f = dht.readTemperature(true);

   // Check if any reads failed and exit early (to try again).
   if (isnan(h) || isnan(t) || isnan(f)) {
    //Serial.println(F("Failed to read from DHT sensor!"));
    return;
   }

   // Compute heat index in Fahrenheit (the default)
   //float hif = dht.computeHeatIndex(f, h);
   // Compute heat index in Celsius (isFahreheit = false)
```

```
  float hic = dht.computeHeatIndex(t, h, false);

  //Serial.print(F("  Humidity: "));
  Serial.print(h);
  //Serial.print(F("%    Temperature: "));
  Serial.print(F(","));
  Serial.print(t);
  Serial.print(",b");
  //Serial.print(F("°C "));
  // Serial.print(f);
  // Serial.print(F("°F  Heat index: "));
  // Serial.print(hic);
  //Serial.print(F("°C "));
  //Serial.print(hif);
  // Serial.println(F("°F"));
 }
/*
```

ANEXO B – CÓDIGO FONTE DO RASPBERRY PI 4B

```
# Amazonas State University
# Final project: Development of IoT System for Refrigerated Cargo Transport Monitoring
# Code: Raspberry Pi 4B
# Created By  : Maykon Henrique Ferreira da Silva
# Created Date: 11/09/2022
# version ='1.0'
# ---------------------------------------------------------------------------
# UNIVERSIDADE DO ESTADO DO AMAZONAS
# TCC2: Desenvolvimento de Sistema IoT para Monitoramento de Transporte de Carga Refrigerada.
# Código: Raspberry Pi 4B
# Autor  : Maykon Henrique Ferreira da Silva
# Data de criação: 11/09/2022
# versão ='1.0'
# ---------------------------------------------------------------------------
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
#install python 3
#   $sudo apt update
#   $sudo apt install python3 idle3
#all the lib used was installed with python3 by default
#execute the program:
#   $ python3 serial_data.py
#---------------------------------------------------------------------------
"""
   This code was made to communicate ESP32 with Raspberry Pi 4 via serial
   ESP32 as a Endpoint sends a couple of information to Raspberry Pi as a temperature, humidity, gps data and speed data
   ESP32 send the datas as a list
   This datas was send to API using REST requests
   The API used was Thinger.io


  _____          _____          __________
|       |       USB     |       |   HTTP   |                |
|ESP32| →→→→→→ |RASP | ))))))))    | Thinger.io  |
|_____ |                |_____ |            |__________|

"""
# ---------------------------------------------------------------------------
# Imports

from datetime import datetime
import serial
import requests
```

```
import subprocess
#import sys
import json
import os
import time
import psutil

#Serial comunnication with ESP 32
Serial_Rasp = serial.Serial('/dev/ttyACM0', 115200)

#function to REST API POST request

def get_rpi_power_status():
    with open(r"/sys/class/hwmon/hwmon1/in0_lcrit_alarm") as StatusFile:
        _throttled = int(StatusFile.readline())
        os.close

    if _throttled == 0:
        return 70 #'Everything is working as intended'
    elif _throttled == 1000:
        return 60 #'Under-voltage was detected, consider getting a uninterruptible power supply for
your Raspberry Pi.'
    elif _throttled == 2000:
        return 50 #'Your Raspberry Pi is limited due to a bad powersupply, replace the power supply
cable or power supply itself.'
    elif _throttled == 3000:
        return 40 #'Your Raspberry Pi is limited due to a bad powersupply, replace the power supply
cable or power supply itself.'
    elif _throttled == 4000:
        return 30 #'The Raspberry Pi is throttled due to a bad power supply this can lead to
corruption and instability, please replace your changer and cables.'
    elif _throttled == 5000:
        return 20 #'The Raspberry Pi is throttled due to a bad power supply this can lead to
corruption and instability, please replace your changer and cables.'
    elif _throttled == 8000:
        return 10 #'Your Raspberry Pi is overheating, consider getting a fan or heat sinks.'
    else:
        return 'There is a problem with your power supply or system.'

def getFree():
    free = os.popen("free -h")
    i = 0
    while True:
        i = i + 1
        line = free.readline()
        if i==2:
```

```
        return(line.split()[0:7])


def temp_gpu():
    gpu_temp = os.popen("vcgencmd measure_temp").readline()
    os.close
    gpu_temp=gpu_temp.replace("'C", "")
    gpu_temp=gpu_temp.replace("temp=", "")
    return gpu_temp

def temp_cpu():

    with open(r"/sys/class/thermal/thermal_zone0/temp") as File:
        cpu_temp = File.readline()
        os.close

    cpu_temp=round(float(cpu_temp)/1000, 1)

    return cpu_temp

def post_request(latitude,
                 longitude,
                 speed,
                 humidity,
                 temperature,
                 temperature_gpu,
                 temperature_cpu,
                 cpu_usage_percent,
                 disk_usage_percent,
                 virtual_memory_percent,
                 power_suply,
                 memory_usage_percent):
    try:
        #creating a json var
        data='{}'
        #adding new statements to data
        y = {
            "lat":latitude,
            "lng":longitude,
            "speed":speed,
            "humidity":humidity,
            "temperature":temperature,
            "temperature_gpu":temperature_gpu,
            "temperature_cpu":temperature_cpu,
            "cpu_usage_percent":cpu_usage_percent,
            "disk_usage_percent":disk_usage_percent,
```

```
            "virtual_memory_percent":virtual_memory_percent,
            "power_suply":power_suply,
            "memory_usage_percent":memory_usage_percent

            }
        # parsing JSON string:
        z = json.loads(data)

        # appending the data
        z.update(y)

        #subproces makes a terminal command to post newest data
        # -H -> header
        # -X -> request type
        # -d -> data

        subprocess.run([
                "curl",
                "-H", "Content-Type: application/json;charset=UTF-8",
                "-H", "Authorization: Bearer xxxxxxxxxxxxxxxxxxxxxxx", #change this authorization
bearer to your code to your code
                "-H", "Accept: application/json, text/plain, */*",  #change this authentication code to
your auth code
                "-X", "POST",
                #.json file to String
                "-d",json.dumps(z),
                #API host
                "https://backend.thinger.io/v3/users/mferrero/devices/Rasphttp/callback/data"   #change
this url to your path API
                ])
    #if there are any Exception in this function it is reported
    except Exception as request_error:
        print(request_error)

#function to generate and writing logs in a .txt file
def write_log():
    try:
        #get current datetime and converts to string
        current_datetime = datetime.now()
        str_current_datetime = str(current_datetime)
        #open control.txt file with append proprety
        file = open('control.txt', "a")
        #adding read values in the log's line
        file.write("\n"
                    +str_current_datetime+","
                    +s_lat+","
```

```
                +s_long+","
                +s_speed+","
                +s_hum+","
                +s_temp)
    #if there are any path os error a exception is called
    except OSError as exception:
        print(exception)
    #if file is not found a exception is called
    except FileNotFoundError:
        msg = "Sorry, the file "+ arquivo + "does not exist."
        print(msg)
    #this code snippet is always compiled even though an exception has been called
    finally:
        #close file and print current_datetime in serial for a real time analisys in linux terminal
        file.close()
        print("")
        print(current_datetime)

#LOOP BEGIN

#always execute
while True :
    try:
        #read the data received from ESP32 in RASP serial and save each other in a string var
        answer=str(Serial_Rasp.readline())
        #  print(len(answer))
        #print(answer)
        s_lat=answer.split(',')[1]
        s_long=answer.split(',')[2]
        s_speed=answer.split(',')[3]
        s_hum=answer.split(',')[4]
        s_temp=answer.split(',')[5]
    #if there are any exception in this part an exception is called as a Serial.Connection Exception
    except Exception:
        print("Serial.Conection Excepition")
        #writing this values in log file

    #if none exception is called save each var as a float type
    else:
        write_log()
        lat_f=float(s_lat)
        long_f=float(s_long)
        speed_f=float(s_speed)
        hum_f=float(s_hum)
        temp_f=float(s_temp)
```

```
temp_gpu_f=float(temp_gpu())
time.sleep(0.10)
temp_cpu_f=float(temp_cpu())
time.sleep(0.10)

cpu_usage_percent = psutil.cpu_percent()
disk_usage_percent = psutil.disk_usage('/').percent
virtual_memory_percent = psutil.virtual_memory().percent
power_suply = get_rpi_power_status()

mem = getFree()
mem[2]=mem[2].replace("Mi","")
mem[3]=mem[3].replace("Gi","")
memory_usage_percent=round((float(mem[2])/1000)/float(mem[3])*100,2)

#print all var for a real time analisys in linux terminal
print("latidude: {}" .format(lat_f))
print("longitude: {} ".format(long_f))
print("speed :{} ".format(speed_f))
print("hum :{} ".format(hum_f))
print("temp :{} ".format(temp_f))

print("disk_usage_percent: {}".format(disk_usage_percent))
#print(disk_usage_percent)
print("virtual_memory_percent: {}".format(virtual_memory_percent))
#print(virtual_memory_percent)
#print(power_suply)
time.sleep(0.1)
print("temp_gpu: {}".format(temp_gpu_f))
print("temp_cpu: {}".format(temp_cpu_f))
time.sleep(0.1)
print("cpu_usage_percent: {}".format(cpu_usage_percent))
#print(cpu_usage_percent)
print("memory_usage_percent: {}".format(memory_usage_percent))
#print(memory_usage_percent)

print("power_suply: {}".format(power_suply))
print("-----------------")
time.sleep(5)

#after read all the serial values, make a POST request to API
post_request(lat_f,
        long_f,
        speed_f,
        hum_f,
        temp_f,
```

```
            temp_gpu_f,
            temp_cpu_f,
            cpu_usage_percent,
            disk_usage_percent,
            virtual_memory_percent,
            power_suply,
            memory_usage_percent)

#LOOP END
```