UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
MESTRADO EM ENGENHARIA ELÉTRICA

Jéssica Bárbara da Silva Soares

Desenvolvimento de conexões para comunicação entre microcomputadores e microcontroladores do tipo ESP32 usando linguagem do Arduino e MicroPython

MOSSORÓ
2024

Jéssica Bárbara da Silva Soares

Desenvolvimento de conexões para comunicação entre microcomputadores e microcontroladores do tipo ESP32 usando linguagem do Arduino e MicroPython

Exame de dissertação de mestrado apresentado ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Mestre em Engenharia Elétrica.

Linha de Pesquisa: SISTEMAS DE CONTROLE E AUTOMAÇÃO
Orientador: Prof. Dr. Idalmir de Souza Queiroz Júnior
Co-orientador: Prof. Dr. Paulo Henrique Lopes Silva

MOSSORÓ
2024

S676d Soares, Jessica Barbara da Silva.
Desenvolvimento de conexões para comunicação entre microcomputadores e microcontroladores do tipo ESP32 usando linguagem do Arduino e MicroPython / Jessica Barbara da Silva Soares. - 2024.
60 f. : il.

Orientador: Idalmir de Souza Queiroz Júnior.
Coorientador: Paulo Henrique Lopes Silva.
Dissertação (Mestrado) - Universidade Federal Rural do Semi-árido, Programa de Pós-graduação em --Selecione um Curso ou Programa--, 2024.

1. Internet das coisas. 2. sistemas embarcados. 3. MicroPython. 4. análise de dados. I. Júnior, Idalmir de Souza Queiroz, orient. II. Silva, Paulo Henrique Lopes, co-orient. III. Título.

Ficha catalográfica elaborada por sistema gerador automáto em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Mossoró / Setor de Informação e Referência
Bibliotecária: Keina Cristina Santos Sousa e Silva
CRB: 15/120

Jéssica Bárbara da Silva Soares

Desenvolvimento de conexões para comunicação entre microcomputadores e microcontroladores do tipo ESP32 usando linguagem do Arduino e MicroPython

Exame de dissertação de mestrado apresentado ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Mestre em Engenharia Elétrica.

Linha de Pesquisa: SISTEMAS DE CONTROLE E AUTOMAÇÃO

Defendida em: 22 / 02 / 2024.

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br IDALMIR DE SOUZA QUEIROZ JUNIOR
Data: 08/05/2024 14:23:49-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Idalmir de Souza Queiroz Júnior (UFERSA)
Presidente

Documento assinado digitalmente
gov.br PAULO HENRIQUE LOPES SILVA
Data: 08/05/2024 14:36:08-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Paulo Henrique Lopes Silva (UFERSA)
Co- orientador

Documento assinado digitalmente
gov.br DANIELLE SIMONE DA SILVA CASILLO
Data: 09/05/2024 14:12:30-0300
Verifique em https://validar.iti.gov.br

Profª. Dra. Danielle Simone da Silva Casillo (UFERSA)
Membro Examinador

Documento assinado digitalmente
gov.br FRANCISCO DE ASSIS BRITO FILHO
Data: 12/05/2024 19:22:05-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Francisco de Assis Brito Filho (UFERSA)
Membro Examinador

Documento assinado digitalmente
gov.br JOAO BATISTA BORGES NETO
Data: 10/05/2024 09:27:53-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. João Batista Borges Neto (UFRN)
Membro Examinador

AGRADECIMENTOS

Agradeço a minha mãe e em nome dela, a toda minha família, que fizeram incontáveis esforços para que eu trilhasse meu caminho acadêmico e profissional.

Agradeço ao meu marido, Wagner Gomes, que é um grande incentivador dos meus projetos profissionais e que acredita fortemente no meu potencial. Sua escuta foi e é um grande apoio para as minhas ideias saírem do papel. Além de ser o ouvinte fiel das minhas reflexões, reclamações e aflições (risos).

Agradeço a Lucas Ribeiro, amigo com quem puder discutir temas pertinentes acerca do experimento. As dicas trocadas sobre estatística, as nossas conversas sobre temas computacionais e o material fornecido sobre as ferramentas de Ciência de Dados foram fundamentais a execução do trabalho e até mesmo, foram elementos tranquilizadores.

Agradeço a minha amiga Ingrid, por ter sido apoio emocional em diversos momentos, além da dissertação, inclusive. E por ser essa companhia que me inspira e me inclui em projetos acadêmicos desde que nos conhecemos.

Agradeço ao meu Orientador Idalmir Queiroz pelos lembretes, por esclarecer dúvidas e ajudar sempre que foi preciso. A minha admiração pelo profissional e pessoa que você é já existe desde a minha adolescência, quando eu era aluna do CEFET-Mossoró.

Agradeço ao meu co-orientador Paulo Henrique por toda paciência, compreensão e apoio. Sempre disposto a tirar dúvidas teóricas e até mesmo a me auxiliar na pesquisa, aquisição e utilização de materiais. Migrar para uma área computacional foi um desafio para mim e sem a sua ajuda teria sido muito mais difícil.

Agradeço a Thomas Maikon, pela parceria na criação dos códigos na execução do experimento, o desenvolvimento desse trabalho também é mérito seu.

Agradeço por fim a Banca Examinadora por terem aceitado o convite e destinado tempo e atenção para o trabalho, acredito que receberei contribuições importantes para melhoria da versão final.

ÉPIGRAFE

“Desistir... eu já pensei seriamente nisso, mas nunca me levei realmente a sério; é que tem mais chão nos meus olhos do que o cansaço nas minhas pernas, mais esperança nos meus passos, do que tristeza nos meus ombros, mais estrada no meu coração do que medo na minha cabeça.”

Cora Coralina

RESUMO

O avanço da tecnologia é algo indiscutível e, apesar de ser nítido o quanto as atividades e objetos estão cada vez mais integrados à tecnologia, essa presença também é invisível, pois os sistemas computadorizados estão embutidos em carros, casas e até nas pessoas. Com a existência da Internet móvel e dos Sistemas Distribuídos houve um aumento de importância e utilização da Internet das Coisas (do inglês *Internet of Things*, ou apenas *IoT*). A *IoT* está associada à capacidade de os objetos estarem conectados e trocando informações/ações em rede, contando com sensores, microprocessadores e microcontroladores distribuídos pelos ambientes que captam dados e são capazes de executar tarefas de forma automatizada. Para se atender ao aumento da necessidade de aplicações distribuídas e suas especificidades, é necessário o contínuo desenvolvimento dos sistemas embarcados e de tecnologias de comunicação, arquiteturas e linguagens de programação para os referidos sistemas. O objetivo do trabalho apresentado é verificar o desempenho ao se utilizar dois cenários distintos para uma aplicação distribuída. Um cenário através da linguagem do Arduino – plataforma baseada em C/C++ já consolidada e muito utilizada pelos desenvolvedores – e o outro cenário com a utilização do MicroPython – derivação da linguagem Python, que vem ganhando notoriedade e muitos adeptos devido à sua relativa facilidade de aprendizado e disponibilidade de recursos, como diversas bibliotecas. A placa de desenvolvimento ESP32 foi o ambiente no qual a aplicação foi executada. O teste consistiu no envio de mensagens de diferentes tamanhos seguindo uma ordem crescente, para os quais se verificou a velocidade de comunicação (latência). A arquitetura em ambos os cenários ocorreu através do protocolo de comunicação cliente/servidor. Após a obtenção dos dados, foram utilizadas ferramentas de análises de dados para se extrair os valores de latência médios válidos para os dois cenários e pôde-se verificar que o desempenho através da aplicação na linguagem do Arduino apresentou melhor resultado, pois obteve menor latência ao ser comparada com a aplicação em MicroPython.

Palavras-chave: Internet das coisas; sistemas embarcados; MicroPython; análise de dados.

ABSTRACT

The advancement of technology is indisputable and, although it is clear how activities and objects are increasingly integrated with technology, this presence is also invisible, as computerized systems are embedded in cars, homes and even people. With the existence of the mobile Internet and Distributed Systems there has been an increase in the importance and use of the Internet of Things (from the English Internet of Things, or just IoT). IoT is associated with the ability of objects to be connected and exchange information/actions in a network, relying on sensors, microprocessors and microcontrollers distributed throughout environments that capture data and are capable of performing tasks in an automated manner. To meet the increased need for distributed applications and their specificities, the continuous development of embedded systems and communication technologies, architectures and programming languages for these systems is necessary. The objective of the work presented is to verify the performance when using two different scenarios for a distributed application. One scenario using the Arduino language – a platform based on C/C++ already consolidated and widely used by developers – and the other scenario using MicroPython – a derivative of the Python language, which has been gaining notoriety and many followers due to its relative ease of use. learning and availability of resources, such as various libraries. The ESP32 development board was the environment in which the application was run. The test consisted of sending messages of different sizes in ascending order, for which the communication speed (latency) was checked. The architecture in both scenarios occurred through the client/server communication protocol. After obtaining the data, data analysis tools were used to extract the average latency values valid for the two scenarios and it was possible to verify that the performance through the application in the Arduino language presented better results, as it obtained lower latency when be compared with the MicroPython application.

Keywords: Internet of things; embedded systems; Arduino; MicroPython; data analysis.

LISTA DE FIGURAS

Figura 1. Representação dos objetivos específicos 15
Figura 2. Representação da arquitetura cliente/servidor 18
Figura 3. Representação visual da abrangência da IoT nas diversas áreas 20
Figura 4. Diagrama simplificado linguagem compilada 23
Figura 5. Diagrama simplificado linguagem interpretada 25
Figura 6. Representação da configuração do experimento 32
Figura 7. Representação da comunicação executada 33
Figura 8. ESP32 utilizado 36
Figura 9. Exemplo no Jupyter Notebook com destaque na linha de comentário 40
Figura 10. Exemplo no Jupyter Notebook com destaque na linha de código 40
Figura 11. Extraindo os arquivos com dados a partir da pasta original 42
Figura 12. Juntando os dados numa tabela única com seus respectivos rótulos 42
Figura 13. Amostra exibindo primeiras e últimas linhas da lista gerada 43
Figura 14. Conversão de tempo para milissegundos 43
Figura 15. Exemplo básico de uma distribuição em boxplot 46
Figura 16. Código para identificação e remoção dos outliers 46
Figura 17. Resumo do tratamento dos dados 48
Figura 18. Gráfico de dispersão dos dados antes da extração dos outliers 49
Figura 19. Histograma para mensagem de 1 byte antes de extração de outliers 49
Figura 20. Gráfico de dispersão dos dados após a extração dos outliers 50
Figura 21. Histograma para mensagem de 1 byte após de extração de outliers 51
Figura 22. Tempo médio de envio de mensagens por tamanho e linguagem 52
Figura 23. Tempo médio de envio de mensagens com valores acima de 1024 54

LISTA DE TABELAS

Tabela 1. Arquivo gerado com mensagens de 4 bytes no cenário 1 34
Tabela 2. Especificação técnica ESP32 37
Tabela 3. Média dos valores de tempo para cada teste. 52
Tabela 4. Média dos valores de tempo com valores maiores 53

SUMÁRIO

1 INTRODUÇÃO 12

1.1 Justificativa 12

1.2 Objetivos geral e específicos 15

1.2.1 Objetivo Geral 15

1.2.2 Objetivos específicos 15

1.3 Organização da Dissertação 16

2 REFERENCIAL TEÓRICO E TRABALHOS RELACIONADOS 17

2.1 Sistemas distribuídos 17

2.2 Internet das coisas 19

2.3 A Linguagem do Arduino 21

2.4 MicroPython 24

2.5 Integrated Development Environment – IDE 26

2.6 ESP 32 27

2.7 Ciência de dados 28

2.8 Trabalhos relacionados 29

3 CARACTERIZAÇÃO DO EXPERIMENTO 31

3.1 Configuração do experimento 31

3.2 Especificação técnica dos dispositivos 36

3.3 Ferramentas para análise de dados 38

3.4 Análise e tratamento dos dados 41

4 ANÁLISE DOS RESULTADOS 47

4.1 Observações adicionais levantadas 53

5 CONCLUSÕES E TRABALHOS FUTUROS 56

REFERÊNCIAS 58

1 INTRODUÇÃO

Como a ideia apresentada em Facionni Filho (2016), a relação da sociedade com a Internet vem se aprofundando num ritmo acelerado e, a cada dia, não só pessoas, como também mais e mais dispositivos estão inseridos de forma quase simbiótica na rotina da humanidade. Durante atividades totalmente costumeiras, bilhões de pessoas no mundo estão conectadas e imersas na rede, obtendo e enviando informações através dos mais variados dispositivos sem nem sequer tomar consciência da quantidade de dados que envia/recebe.

No mundo pós Internet, é nítido o aumento da velocidade na qual foram se transformando as relações de trabalho, de aprendizado e de relacionamento interpessoal e [...] "consolida-se uma rede digital universal de relações sociais, intelectuais, artísticas e éticas, além de ideias e práticas" (PATRICIO et al., 2018, apud MARTINO, 2015, p. 27).

Os indivíduos estão acompanhando tudo que ocorre no mundo em tempo real (ou quase) e realizando as mais diversas atividades pessoais e profissionais, através de seus *smartphones*, *tablets* e *notebooks*.

Como exposto em Antunes (2020), essa realidade foi se tornando possível através da miniaturização dos dispositivos tecnológicos e redução de custos de produção dos microprocessadores. Além da evolução constante de sistemas embarcados, possibilitando a criação dos mais diversos *gadgets*[1].

Esse cenário também se deve ao desenvolvimento das redes de computadores e sistemas distribuídos que permitiu a implementação prática de diversas soluções para interconexão de dispositivos.

Assim, como comentado por Ferreira (2019), sensores distribuídos nos ambientes captam informações, como dados de presença, temperatura, reconhecimento digital e facial, variáveis de processo industrial ou informações comerciais e residenciais e, a partir dessas informações, executam ações previamente programadas.

1.1 Justificativa

Para se estabelecer a comunicação de dispositivos em rede, existem protocolos de comunicação que permitem a interação dos componentes nesse sistema. Todo protocolo é formado por um conjunto de regras e características de modo a garantir melhor desempenho,

[1] Termo utilizado para se referir a dispositivos eletrônicos conectados e portáteis, como por exemplo relógios inteligentes.

levando em conta diversos parâmetros. Como exemplos desses protocolos, podemos citar: o padrão IEEE 802.15.4, 6LoWPAN, Zigbee, Thread, *Message QueuingTelemetry Transport* (MQTT) e *Constrained Application Protocol* (CoAP).

No intuito de se avançar e obter cada vez mais comodidade e segurança, é necessário que se investigue quais são as melhores alternativas para o desenvolvimento de aplicações que atendam às especificidades desses sistemas, buscando corresponder às necessidades à medida que vão surgindo, além do aprimoramento dos protocolos propriamente ditos.

Na busca por uma configuração mais eficiente possível, com baixo consumo de energia para os dispositivos, é preciso se pensar além dos protocolos utilizados, é preciso pensar nos sensores utilizados, na rede de conexão sem fio, nas linguagens a se desenvolver como base da comunicação entre os equipamentos, buscando otimizar todos os fatores envolvidos.

Não é possível definir de forma simplista qual linguagem ou ferramenta é a melhor para todos os casos, até porque não existe uma solução única para casos distintos. O que de fato importa é pontuar quais são os aspectos positivos e negativos de cada uma, para que se possa fazer a melhor escolha caso a caso e buscar melhorias ao passo que se conhecem os gargalos.

No processo de desenvolvimento de novas linguagens é muito comum umas linguagens surgirem a partir de outras, gerando derivações capazes de suprir uma ou outra necessidade importante para as quais foram desenvolvidas. Não é preciso e nem desejado que seja criada uma linguagem totalmente do zero, nem radicalmente distinta das antecessoras e é natural que se derive de uma base conhecida, até mesmo contribuindo com a difusão e o aprendizado dos usuários.

No que se referem a linguagens de programação criadas/utilizadas, a linguagem C/C++ é sem dúvidas uma das mais populares, mantendo seu destaque até hoje, conforme podemos perceber no acompanhamento do índice Tiobe[2](2023). Se olharmos para os últimos anos, ela está se mantendo no pódio de utilização de forma geral. Esse sucesso também se estende à sua utilização em sistemas embarcados, devido alguns fatores, dentre eles, ser uma linguagem que possui um bom acesso a nível de *hardware*.

Ela é base para diversas ferramentas de eletrônica embarcada, como a plataforma *wiring*, que também serviu como base para o Arduino, que é uma das mais famosas plataformas

[2] O índice TIOBE Programming Community é um indicador da popularidade das linguagens de programação. O índice é atualizado uma vez por mês. As classificações são baseadas no número de engenheiros qualificados em todo o mundo, cursos e fornecedores terceirizados. Mecanismos de busca populares como Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube e Baidu são usados para calcular as classificações. Fonte: https://www.tiobe.com/tiobe-index/(adaptado).

eletrônicas para desenvolvimento de sistemas embarcados, com diversas aplicações para desenvolvedores e para aprendizado em eletrônica.

Contudo, existem também outras linguagens sendo criadas e/ou aprimoradas e crescendo em número de adeptos. Dentre as linguagens que vêm ganhando preferência de diversos desenvolvedores é possível citar a linguagem Python.

Segundo classificação anual das principais linguagens de programação do *IEEE Spectrum*[3] para o ano de 2022, Python está no topo em adeptos, e é seguido de perto pela linguagem C, se considerarmos apenas C, segregando da linguagem C++.

O Python possui sua versão para sistemas embarcados, o MicroPython, que, conforme informado no site oficial, MicroPython.org (2023), é uma implementação baseada em Python 3, fazendo com que programadores optem por desenvolver seus projetos nessa linguagem.

Dentre os diversos parâmetros que fazem se escolher uma configuração de componentes em um sistema em detrimento a outro estão, por exemplo: velocidade, confiabilidade, latência e segurança.

Em relação ao desempenho de uma rede, pode-se destacar a latência como um parâmetro importante na avaliação de desempenho de uma comunicação. “Latência é o tempo decorrido após uma operação de envio ser executada e antes que os dados comecem a chegar em seu destino” (COULOURIS, 2013, p. 83).

Em acordo com o que diz Cerda (2018), pode-se entender a latência como o tempo entre a saída de um pacote de dados da sua máquina e o início da resposta no servidor de destino. Existem estudos que utilizam o termo latência bidirecional, chamado RTT (*Round Time Trip*) ou simplesmente *ping*, que é o tempo que um pacote leva para chegar ao destino e voltar à origem.

Pode-se dizer que a latência está relacionada ao atraso na comunicação e, por isso, que, para mesmas situações, quanto menor a latência, melhor o sistema.

Com o intuito de disponibilizar mais informações para subsidiar pesquisadores e desenvolvedores, este trabalho busca ser um instrumento de observação do comportamento da latência entre duas formas de se estabelecer uma comunicação cliente/servidor sem fio em aplicações *IoT*. Para isso, foi desenvolvida uma aplicação distribuída que consistiu em troca de mensagens entre um computador cliente e um sistema embarcado servidor, no caso, um microcontrolador ESP32.

[3] IEEE Spectrum é uma revista de tecnologia premiada e a principal publicação do IEEE, organização profissional dedicada à engenharia e às ciências aplicadas. Fonte: https://spectrum.ieee.org/st/about(adaptado)

No primeiro cenário, a comunicação do cliente ocorreu na linguagem do Arduino, que é baseada na linguagem C/C++ e no segundo cenário a comunicação do cliente ocorreu na liguagem MicroPython, baseado na linguagem Python. Os tamanhos das mensagens trocadas seguiram uma progressão crescente.

1.2 Objetivos geral e específicos

Nesta seção estão descrito o objetivo geral do trabalho e os objetivos que se buscam em cada tarefa de forma a atingir o objetivo geral.

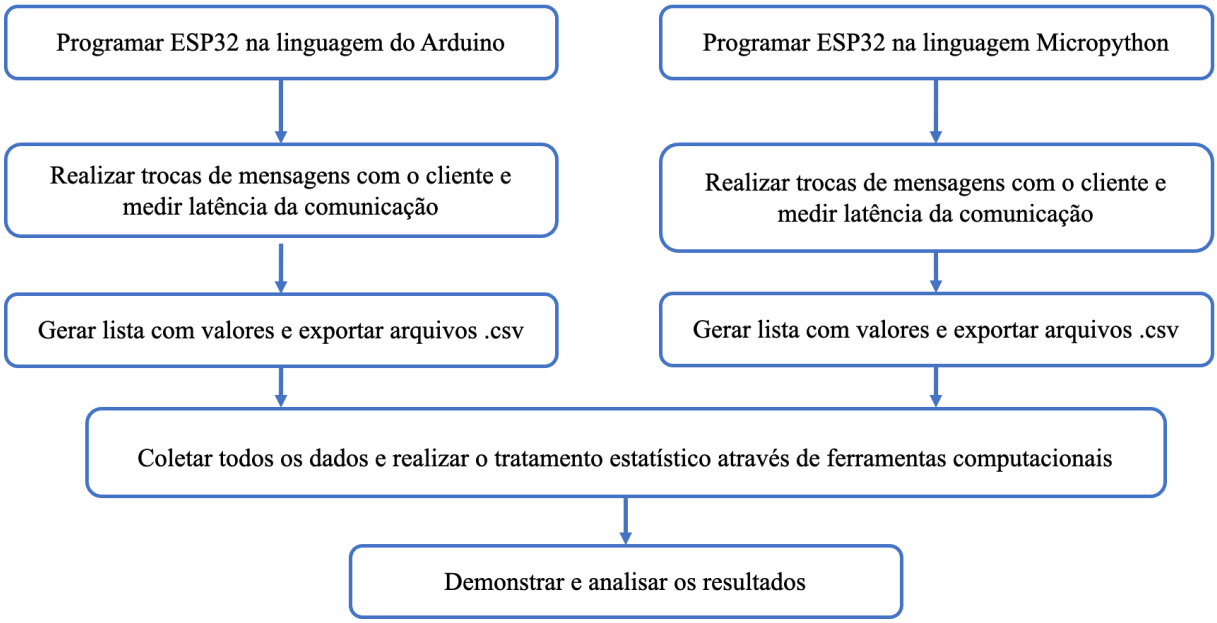
1.2.1 Objetivo Geral

Verificar os valores de latência de comunicação entre microcomputador e microcontrolador em uma aplicação cliente/servidor sem fio.

1.2.2 Objetivos específicos

O diagrama representado na Figura 1 busca expressar quais são os objetivos específicos traçados para se atingir o objetivo geral.

Figura 1. Representação dos objetivos específicos



Fonte: Autoria própria

Os objetivos descritos na Figura 1 indicam as etapas planejadas, que consistem em:

- Estabelecer a comunicação cliente/servidor com servidor na linguagem do Arduino e cliente em C++ (fluxo que inicia do lado esquerdo da figura);
- Estabelecer a comunicação cliente/servidor com servidor na linguagem MicroPython e cliente em Python (fluxo que inicia do lado direito da figura);
- Realizar a troca de mensagens em diferentes tamanhos, realizando 60 vezes a comunicação para cada tamanho de mensagem;
- Medir a latência em todos os envios/recebimentos da comunicação;
- Realizar o tratamento dos dados através de ferramenta computacional para calcular as médias da latência em cada tamanho de mensagem.

As etapas acima descritas nos objetivos específicos foram elaboradas em um formato que se assemelha a um plano de ataque para que fosse alcançado o objetivo geral, também descrito nesse item.

1.3 Organização da Dissertação

Este trabalho está dividido em cinco capítulos, organizados da seguinte forma: no primeiro capítulo está presente a Introdução, contendo uma visão inicial, justificativa e objetivos; o segundo capítulo é composto pelo referencial teórico, abordando conceitos teóricos importantes para o entendimento e apresentando uma sessão contendo alguns trabalhos relacionados que trazem análises com certa similaridade ao trabalho desenvolvido e que de alguma forma inspiraram o trabalho realizado; o terceiro capítulo traz a configuração do experimento, conceitos que envolveram as etapas de análise e de tratamento dos dados, o quarto capítulo é composto pelos resultados e suas análises e as observações adicionais obtidas durante o experimento e o quinto e último capítulo traz as conclusões do trabalho desenvolvido levantando propostas e considerações do trabalho e possibilidades futuras.

2 REFERENCIAL TEÓRICO E TRABALHOS RELACIONADOS

Neste capítulo é possível entender alguns conceitos chave acerca do trabalho desenvolvido bem como elucidar termos que serão citados no decorrer do experimento.

2.1 Sistemas distribuídos

Os sistemas distribuídos são definidos, segundo Tanenbaum (2023), como um conjunto de computadores independentes que se apresentam a seus usuários como um sistema único e coerente. Para Coulouris (2013), um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações passando mensagens.

Das definições existentes na literatura, é possível perceber que um sistema distribuído é um conjunto formado por computadores que interagem e se "comunicam" para execução de tarefas de forma integrada, como se fossem um dispositivo único.

A propriedade que um Sistema Distribuído tem de se apresentar como um único computador para o usuário final é definida como transparência. "A transparência é definida como a ocultação, para um usuário final ou para um programador de aplicativos, da separação dos componentes em um sistema distribuído, de modo que o sistema seja percebido como um todo, em vez de como uma coleção de componentes independentes." (Coulouris, 2013, p. 23).

Sendo assim, um sistema distribuído é uma boa solução para facilitar aos usuários acesso a recursos e serviços, que podem ser, por exemplo: impressoras, espaço de armazenamento, programas, dados, páginas da Web. Consequentemente, uma das razões de se compartilhar recursos está na economia.

Outra vantagem econômica está na fragmentação e distribuição de tarefas em um grupo de máquinas. Isso faz com que seja possível executar aplicações mais complexas que exigem maior grau de processamento através da agregação de um grupo de máquinas com *hardwares* relativamente simples ao invés de um único supercomputador com alto poder computacional.

Pois, devido ao aumento das tecnologias, os equipamentos têm limitações de capacidade e desempenho que não conseguem acompanhar o mesmo ritmo da complexidade com que as necessidades computacionais crescem. Além disso, desenvolver um supercomputador que consiga agregar todas as necessidades pode inviabilizar algum projeto devido aos custos.

A existência de sistemas distribuídos possibilita também conectar usuários, o que facilita a colaboração entre os elementos, a tolerância a falhas e a troca de informações. Essa

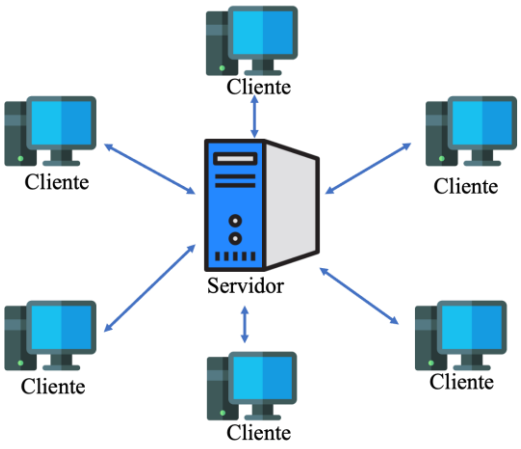
colaboração pode ser ilustrada pelo sucesso da Internet com seus protocolos para trocar arquivos, correio, documentos, áudio e vídeo.

Os componentes independentes dentro de um sistema são nomeados como nós e a forma de comunicar os elementos dentro de um sistema distribuído é definida como conector. O termo não significa que há uma conexão física propriamente dita como o nome leva a pensar. Ele está mais relacionado a um componente de *software*, uma forma de enlace, através do qual os diversos componentes interagem, se interligam. Neste contexto, o conector é um mediador de comunicação entre os componentes e define a forma de comunicação entre os componentes dentro de um Sistema Distribuído.

Esses componentes e conectores podem apresentar formas diferentes de configuração e comunicação entre os nós. Se há um elemento centralizador da comunicação ou não, se está focado em dados ou em objetos etc. A forma como essa configuração está estabelecida é definida como arquitetura. E uma das mais conhecidas é a arquitetura cliente/servidor.

Na arquitetura cliente/servidor, como explicado em Coulouris (2013), existe um computador ou grupo de computadores chamados servidores, atuando como fornecedores de serviço e os chamados clientes, que solicitam serviços/ recursos dos servidores. A comunicação entre eles se baseiam na troca de mensagens, como demonstrado na Figura 2.

Figura 2. Representação da arquitetura cliente/servidor



Fonte: Autoria própria

Nesse entendimento conceitual mais abrangente, é coerente incluir a *IoT* como uma forma ou ramo dentro da área dos Sistemas Distribuídos. Já que é uma maneira de integração entre dispositivos na execução de tarefas se comportando como um “ecossistema” integrado, permitindo objetos “inteligentes” estarem conectados, compartilhando e processando dados.

2.2 Internet das coisas

De acordo com matéria do Conselho Consultivo da Associação Brasileira de Internet das Coisas (ABINC, 2023), pesquisas da *IoT Analytics*[4] estimam que até 2025 cerca de 27 bilhões de dispositivos estarão conectados.

Como visto no capítulo 1 neste trabalho e exposto em Santos (2018), os dispositivos eletrônicos são empregados para atividades cotidianas inseridos de forma tão rotineira que se tornam quase imperceptíveis. Mas, além de dados constantemente trocados entre as pessoas e consumidos por estas, o que vem se tornando cada vez mais crescente também é a troca de informações e conexão entre dispositivos, utilizados para auxiliar em tarefas industriais, na área da saúde, na coleta de dados acadêmicos e até mesmo em tarefas residenciais.

A forma que esses objetos permeiam a rotina das pessoas foi idealizada e conceituada em 1991 por Mark Weiser, definida com o termo “computação ubíqua”, esse conceito explora justamente a forma quase que onipresente e invisível que os dispositivos estão conectados na vida humana.

Como citado no trabalho de Cirilo (2008), a palavra “ubíquo" vem do latim “*obiquu*” e significa “estar em todos os lugares ao mesmo tempo”. Marc Weiser projetou que, no futuro, computadores estariam presentes nos mais comuns objetos do dia a dia - etiquetas de roupas, xícaras de café, interruptores de luz, canetas etc. A computação se move para fora das estações de trabalho e PCs e torna-se pervasiva no cotidiano dos indivíduos.

Essa previsão feita há mais de 30 anos encontra correlação com outros conceitos computacionais que foram se inserindo na rotina e sendo observados durante esses anos, como computação pervasiva, computação móvel e, sem dúvidas, uma realidade que permitiu a materialização dessa ideia através do avanço tecnológico da chamada Internet das Coisas, do inglês *Internet of Things*, ou, em sua forma abreviada, *IoT*.

Segundo Sônego et al (2016), o conceito primordial associado à *IoT* relaciona-se à capacidade que os objetos possuem de se comunicar, reportando informações acerca de seu estado e funcionamento. O termo Internet das Coisas, apresentada nesse trabalho doravante através da sigla *IoT*, foi apresentado por Kevin Ashton no final dos anos 90 e pode ser definido, de maneira geral, como a capacidade de os objetos estarem conectados e serem capazes de fornecer informações e executarem tarefas de forma “autônoma”, disponibilizando diagnósticos e notificações de seu estado.

[4] É fornecedora líder global de insights de mercado e inteligência estratégica de negócios para IoT, IA, nuvem, Edge e Indústria 4.0. Disponível em < https://iot-analytics.com/about/>

Não existe um conceito único para definir *IoT*. Nas palavras de Magrani (2021) está relacionado a um ambiente de objetos físicos interconectados com a Internet através de sensores embutidos criando um ecossistema de computação ubíqua visando facilitar o cotidiano das pessoas. Para Ferreira et al (2019), contempla a conexão lógica dos dispositivos e meios relacionados ao ambiente produtivo como os sensores, controladores, computadores, células de produção, sistema de planejamento produtivo, sendo as informações compartilhadas através de bancos de dados.

Independente de uma ou outra abordagem mais específica em sua conceituação, fato é que se trata de uma nova fronteira onde a internet está aprofundando e, ao se vincular entre si e com as pessoas, lugares e dados, controles em automação e diagnósticos, podemos obter respostas que aumentem a qualidade e a praticidade nas mais variadas aplicações.

As soluções em *IoT* se fazem presentes nas mais diversas áreas, conforme representada de forma simplificada na Figura 3.

Figura 3. Representação visual da abrangência da *IoT* nas diversas áreas

Fonte: Autoria própria

A Figura 3 representa que a grande abrangência da área de *IoT* mostrando que, desde dispositivos para controle residencial até sistemas de segurança empresarial, transporte e indústria e saúde podem lançar mão da tecnologia e da troca de dados para desenvolvimento e operacionalização de suas tarefas.

Mas não há apenas esses elementos mais aparentes integrando o estudo Internet das Coisas, componentes de “baixo nível”, tais como sensores, medidores de energia, coletores de dados também são elementos que compõe a *IoT*. Conforme apresentado em Faccione Filho (2016), há um conjunto de softwares que integra tais componentes, passando por sistemas operacionais, protocolos de comunicação, aplicações, interfaces, bancos de dados e sistemas em nuvem. Essa camada de software deve ser capaz de gerir os inúmeros componentes em uma aplicação.

Segundo Faccioni Filho (2017), no que se refere aos protocolos de identificação, para que seja possível o endereçamento IP para o número gigantesco de dispositivos, o protocolo de IPv6 é o padrão mais apropriado para identificação dos objetos inteligentes na rede. De acordo com Santos (2016), na IoT os elementos da rede são endereçados usando o IPv6 e, geralmente, têm o objetivo de enviar pequenas quantidades de dados obtidos pelos dispositivos.

Neste cenário, é possível conectar uma quantidade extremamente elevada de dispositivos heterogêneos na rede e a IoT pode ser vista como a combinação de diversas tecnologias para viabilizar a integração físico/ virtual.

De maneira geral, os objetos não têm grande poder de processamento, então além de prover um endereçamento na rede, é necessário encontrar formas eficientes de comunicação que se adaptem aos recursos limitados dos objetos.

O desenvolvimento da comunicação entre computadores, bem como a interação entre os mais diversos sensores e equipamentos necessita de um ambiente hiperconectado e as soluções exigem um modelo de internet distribuída, o que faz com que a área de *IoT* seja um ramo integrante dos estudos em Sistemas Distribuídos.

2.3 A Linguagem do Arduino

Como explicado por Antunes (2020), a linguagem C surgiu em 1972 nos laboratórios Bell, criada pelo cientista da computação Dennis Ritchie, a partir da linguagem B. O propósito inicial era que fosse uma linguagem usada no desenvolvimento do sistema operacional Unix, com a finalidade de evitar futuras reescritas do sistema quando um novo *hardware* fosse criado, ou seja, facilitar a compatibilidade do Unix com diversos computadores.

A popularização do sistema UNIX foi um dos fatores que contribuiu com a popularização da linguagem C que está presente em sistemas operacionais atuais como LINUX,

Windows e MAC OS. Nos anos 80 foi criada sua extensão: C++ para lidar com problemas maiores e de maior complexidade.

A linguagem C/C++ é empregada também, e não somente, na criação de *softwares*, em banco de dados, em jogos, aplicativos, inclusive na programação em sistemas embarcados.

Pode ser classificada como uma linguagem de alto nível, o que facilita seu aprendizado em relação à linguagem de máquina, mas apesar de ser de alto nível, algumas fontes a classificam coloquialmente como "médio nível", pois possui características que facilitam comunicação com *hardware* – acesso a escrita de memória e a entradas e saídas.

Por ter sido criada pensando em uma aplicação que lidaria diretamente com o *hardware*, ela também é muito popular para uso com microcontroladores, pois possui recursos que fazem com que acesso aos espaços de memória e dados dessas máquinas seja facilitado. Existem diversas ferramentas que surgiram baseadas em C/C++ e Wiring é uma delas.

Apesar de definida em algumas literaturas como linguagem, não é exatamente uma linguagem de programação em seu conceito mais tradicional. Na verdade, a rigor seria um conjunto composto por uma linguagem, IDE e um microcontrolador em uma plataforma de prototipagem de *hardware* livre.

De acordo com o site oficial wiring.org (2023), Wiring é uma estrutura de programação de código aberto para microcontroladores e foi iniciado por Hernando Barragán (*Universidad de Los Andes* | Escola de Arquitetura e Design), no *Interaction Design Institute Ivrea*, na Itália.

De acordo com sua página oficial, wiring.org (2023), permite escrever *software* de plataforma cruzada para controlar dispositivos conectados a uma ampla gama de placas de microcontroladores para criar todos os tipos de codificação criativa, objetos interativos, espaços ou experiências físicas.

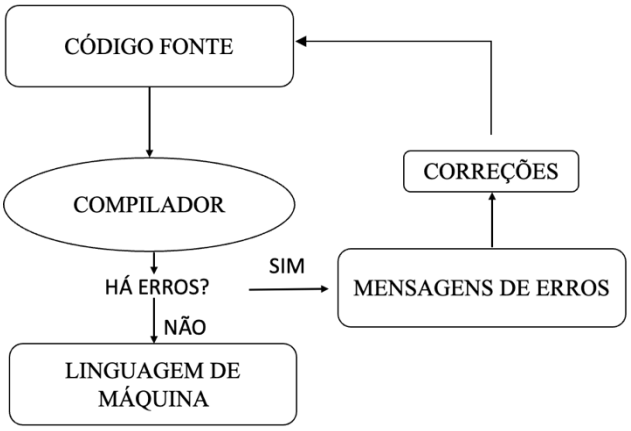
Segundo Kenshima (2021), Wiring serviu como base para criação do Arduino e desde sua criação, o Arduino tem sido usado em milhares de implementações, variando desde projetos simples até os mais complexos. Segundo Oliveira (2020), o Arduino conta com uma grande comunidade mundial, envolvendo os próprios criadores, estudantes, programadores e profissionais, que reuniram suas contribuições a esta plataforma de código aberto e disponibilizaram uma grande quantidade de conhecimento e projetos acessíveis a todos, sejam iniciantes ou profissionais já em atuação.

A linguagem do Arduino, assim como a linguagem C/C++ na qual se baseia, no que se refere a forma de conversão de código para máquina é do tipo compilada, ou seja, para ser executada pela máquina, seu código fonte é convertido através de um compilador para que seus comandos possam ser executados corretamente.

De acordo Sebesta (2018), um compilador é um programa de *software* que segue a regra de sintaxe da linguagem de programação para converter um código-fonte em código de máquina (binário). O compilador converterá todo o seu código-fonte em código de máquina de uma só vez. E, em seguida, seu programa é executado.

Caso haja erros presentes em um programa; gera uma mensagem de erro e após o usuário corrigí-la na sintaxe do programa ele fará novamente a conversão total para ser executado, conforme ilustrado simplificadamente na Figura 4.

Figura 4. Diagrama simplificado linguagem compilada



Fonte: Autoria Própria

A Figura 4 tenta mostrar essa ideia de maneira resumida, na qual todo o código fonte é convertido antes de ser executado na máquina e caso haja erros presentes no código-fonte, esse deve ser corrigido e assim sim todo o código será executado.

Conforme apresentado em Sebesta (2018), os compiladores oferecem vantagens como execução de código otimizada e eficiente, pois o código é previamente traduzido em código de máquina. Isso leva a uma execução mais rápida do programa e à redução da sobrecarga do tempo de execução. Além disso, programas compilados podem ser distribuídos sem revelar o código-fonte.

2.4 MicroPython

Como aponta Borges (2014), Guido Van Rossum, informático que trabalhava num Centro de Pesquisa Holandês (CWI) criou o Python em 1989. Essa linguagem foi lançada em 1991 e teve como motivação atender a uma necessidade de otimização do projeto Amoeba o qual fazia parte. Amoeba era um sistema distribuído desenvolvido conjuntamente pelo CWI e uma Universidade em Amsterdã.

O objetivo de Guido era desenvolver uma linguagem de alto nível que fosse capaz de otimizar o tempo de desenvolvimento do projeto.

Portando, Python é uma linguagem de alto nível, modular, multiplataforma e orientada a objeto, gratuita, com linguagem de sintaxe relativamente simples e de fácil compreensão, agilizando o desenvolvimento de projetos. Python é desenvolvido sob uma licença de código aberto aprovada pela OSI, tornando-o livremente utilizável e distribuível, mesmo para uso comercial. A licença do Python é administrada pela *Python Software Foundation*, como informa seu site oficia python.org (2023).

Como apontado por Kriger (2022), um de seus maiores atrativos é possuir muitas bibliotecas, nativas e de terceiros, tornando-a muito difundida e útil em uma grande variedade de setores dentro de desenvolvimento web, também em áreas como análise de dados, aprendizado de máquina e inteligência artificial. Para trabalhar com ciência de dados conta com diversas bibliotecas e ferramentas, o que também colaborou fortemente com sua expansão e utilização.

Pela sua alta popularização, desde 2019, a linguagem se mantém situada entre as 3 principais linguagens em buscas mensais no índice Tiobe, como é possível acompanhar em tiobe.org (2022) tendo sido a linguagem com maior crescimento de buscas para os anos de 2018, 2020 e 2021.

O crescimento de uso e visibilidade da linguagem permitiu seu desenvolvimento e criação de bibliotecas e ferramentas para diversas aplicações, inclusive para sistemas embarcados, este último contando com uma linguagem derivada com esse propósito, através da criação do MicroPython.

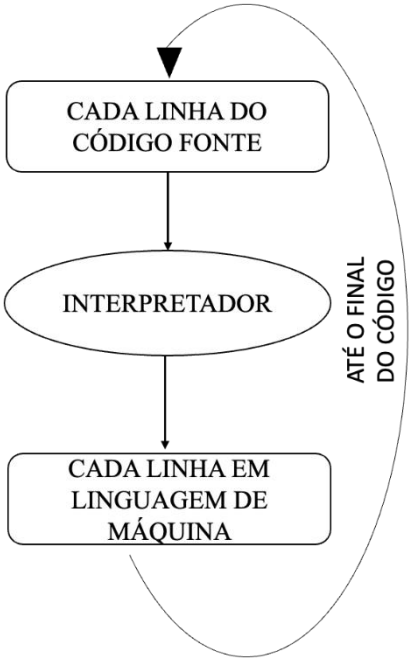
Segundo site oficial micropython.org (2023), o MicroPython é uma implementação enxuta e eficiente da linguagem de programação Python 3 que inclui um pequeno subconjunto da biblioteca padrão Python e é otimizada para execução em microcontroladores e em ambientes restritos.

Está repleto de recursos avançados, como *prompt* interativo, números inteiros de precisão arbitrários, fechamentos, compreensão de lista, geradores, tratamento de exceções e muito mais. No entanto, é compacto o suficiente para caber e funcionar em apenas 256k de espaço de código e 16k de RAM.

Pretende ser o mais compatível possível com o Python normal para permitir a transferência de código com facilidade do desktop para um microcontrolador ou sistema embarcado.

Quanto ao que se refere à forma de conversão de código para máquina, ela, assim como Python, é do tipo interpretada. Um interpretador, assim como um compilador também é um programa de *software* que traduz um código-fonte em uma linguagem de máquina. No entanto, o interpretador converte a linguagem de programação de alto nível em linguagem de máquina linha por linha enquanto interpreta e executa o programa, como apresentado em Sebesta (2018) e representado na Figura 5 de forma resumida.

Figura 5. Diagrama simplificado linguagem interpretada



Fonte: Autoria própria

De acordo com lido em Sebesta (2018), os interpretadores oferecem vantagens como depuração fácil, pois os erros são detectados durante o tempo de execução e a capacidade de fornecer *feedback* imediato em ambientes interativos. Eles também são mais flexíveis, pois as

alterações no código entram em vigor sem a necessidade de recompilação. No entanto, os programas interpretados, de regra geral, tendem a ser mais lentos em comparação com os compilados devido à sobrecarga de tradução do código durante a execução.

2.5 *Integrated Development Environment* – IDE

O termo IDE significa *Integrated Development Environment*, ou, em uma tradução livre para o português: Ambiente de Desenvolvimento Integrado.

Segundo Lima (2022), uma IDE é um programa repleto de funcionalidades que podem ser usadas por muitos aspectos no desenvolvimento de *software*, que inclui ferramentas de preenchimento de código, plugins, e muitos outros recursos para facilitar o processo de desenvolvimento de *software*. Porém, atualmente existem diversas IDE's disponíveis no mercado, algumas desenvolvidas para linguagens de programação específicas.

Trata-se de uma ferramenta que reúne recursos que visam facilitar o processo de desenvolvimento de programas. Consiste, de forma geral de um editor de código-fonte, de utilitários que automatizam tarefas simples e são responsáveis pela compilação/interpretação do código e *debugger*, responsável por localizar e indicar erros.

Existem algumas IDE's que podem ser utilizadas para programar um ESP32. Para o trabalho foram utilizadas duas, a IDE do Arduino, ao se programar o primeiro cenário, através da linguagem da própria plataforma Arduino, e a IDE Thonny para se programar em MicroPython.

Desde o ano de 2022 também é possível usar a linguagem MicroPython na IDE do Arduino, contudo, pela maior quantidade de informações na literatura e conhecimento prévio, foi definido que a IDE usada para a programação em MicroPython seria Thonny, já que essa implementação na plataforma Arduino é mais recente.

Segundo Oliveira (2020). a IDE do Arduino é responsável pela implementação dos códigos e transferência desse código para o microcontrolador, que é responsável pelo processamento dos dados, controlando o envio e recebimento de dados das portas digitais e analógicas de programação, ou seja, será responsável por gerenciar e dar funcionalidade ao circuito, é implementada e executada em um computador (programação), conhecida como *sketch*, que após concluída será transferida via *upload* para a placa de prototipagem

através de uma comunicação serial. O *sketch* implementado pelo analista dirá ao sistema ou à placa o que deve ser executado durante o seu funcionamento.

A IDE Thonny é uma das opções para se programar em Python/MicroPython. Segundo site oficial, thonny.org (2023), é uma IDE simples e intuitiva. Ela já possui o Python 3 instalado, e considerada pelo próprio site oficial, uma IDE para iniciantes, para qual existe muito material disponível.

2.6 ESP 32

Como já comentado nesse trabalho, o desenvolvimento de sistemas embarcados é um dos fatores que permite a materialização da computação ubíqua e da *IoT*. Esse termo, mesmo sendo conhecido no universo dos desenvolvedores, ainda pode trazer dúvidas e é preciso ter em mente que o termo "sistema embarcado" não define apenas um único tipo de dispositivo, mas é possível perceber características gerais comuns a grande parte desses equipamentos eletrônicos.

Segundo Antunes (2020), são equipamentos com algumas características comuns, como por exemplo: utilizados para aplicação para um propósito específico; a descentralização; o uso de microprocessadores; capacidade limitada de memória e armazenamento e alta conectividade e disponibilidade quase que integral, para os quais existem sistemas operacionais específicos e forma de programar atividades específicas.

O ESP32 é um microcontrolador com conectividade sem fio integrada, adequado para diversas aplicações *IoT*, como exposto em Plauska (2022).

Como exposto em Maier (2017), o ESP32 é um dispositivo de baixo custo e baixo consumo de energia, compostos de chips de microcontroladores com Wi-Fi e recursos *bluetooth* e uma estrutura altamente integrada alimentada por um microprocessador dual-core Tensilica Xtensa LX6.

"Um microcontrolador trata-se de um sistema microprocessado com vários periféricos dentro de um único chip." (KENSHIMA, 2021, s.n.) e um microprocessador "é um circuito integrado com muitas portas lógicas, organizadas de modo a realizar cálculos computacionais." (KENSHIMA, 2021, s.n.).

O ESP32 é "projetado para dispositivos móveis, eletrônicos vestíveis e aplicações *IoT*, o ESP32 atinge consumo de energia ultrabaixo com uma combinação de vários tipos de *software* proprietário." (ESPRESSIF, 2023, s.n.) sua aquisição é de baixo custo e relativamente

alto poder de processamento. O sistema embarcado é um microcontrolador projetado pela fabricante Espressif Systems, uma empresa chinesa sediada em Xangai. E no site da própria Espressif, é possível ver mais de um modelo de ESP32 no mercado.

Pode ser considerado uma versão melhorada de outro modelo de microcontrolador, o ESP8266, pois o ESP32 além de *hardware* superior, possui módulo *bluetooth* integrado, permitindo sua conexão sem fio seja tanto através do Wifi ou do próprio sistema *bluetoot*h, conferindo ao EPS32 capacidade de aplicação em projetos mais avançados e possui total compatibilidade com a IDE do Arduino.

Esse dispositivo é o escolhido para o desenvolvimento da análise proposta nesse trabalho e no capítulo 3 há mais informações sobre a utilização dele.

2.7 Ciência de dados

Apesar do termo "Ciência de Dados" ser recente como especialidade formalmente definida, também chamada em seu termo inglês *Data Science*, não é novo o fato de as pessoas e organizações sempre lançarem mão de dados para tomada de decisões, contudo, nem todos possuem um nível de amadurecimento de análise de dados satisfatório. É um ramo que vem ganhando cada vez mais destaque e as empresas estão investindo em ferramentas de pesquisas, tratamentos e estatísticas para formalização de análise e delineamento de estratégias de negócios baseado em dados.

Quando se conhece os dados é possível entender como alguns parâmetros estão relacionados de forma clara e fazer correlações que antes não se poderiam e, algumas vezes, nem ao menos havia sido percebida.

Em Comarela (2019) é apresentada a ideia de que para diversos especialistas já consideram que o grande volume de dados gerados a todo momento já ultrapassou a capacidade humana de processamento, fazendo com que seja necessário lançar mão de algoritmos para tratar esse grande volume de dados existentes.

A "crescente necessidade de analisar dados fez surgir a demanda por profissionais bem versados em estatística, aprendizado de máquina, mineração de dados, inteligência artificial, econometria e várias outras áreas correlatas." (COMARELA, 2019, pg 247). Esses profissionais são chamados de cientistas de dados.

O Python foi a linguagem escolhida para tratar os dados de latência para os dados gerados pelos algoritmos nesse trabalho, pois está entre as linguagens mais populares para análise de dados possuindo bibliotecas robustas para essa finalidade. Segundo Mckinney

(2018), Python desenvolveu uma imensa comunidade que fez com que ela evoluísse de uma linguagem inovadora para uma das linguagens mais importantes em ciência de dados, aprendizado de máquina e desenvolvimento de *softwares* em geral, no meio acadêmico e de mercado.

2.8 Trabalhos relacionados

Esta seção será dedicada a citar resumidamente alguns trabalhos relacionados a dissertação aqui desenvolvida considerados pertinentes.

De Sales (2000) em seu trabalho analisa a latência entre clusters pertencentes à Universidade Federal de Santa Catarina (UFSC), que possuem diferentes sistemas operacionais e interligado a diferentes redes. A latência foi medida através da ferramenta *Netperf*[5], um programa para medir a latência nesses ambientes. Como resultado, o trabalho obteve que o cluster com o sistema operacional Linux interligado por uma rede Fast-Ethemet possui a menor latência, seguido pelo cluster com o sistema operacional AIX interligados por uma rede ATM, e pelo SP 2, com o sistema operacional AIX interligado por uma rede Ethernet. O trabalho também obteve como conclusões que, para ambientes que não possuem colisão, a variabilidade da latência pode ser explicada pelo tamanho do segmento com um alto coeficiente de determinação, porém, para as redes que possuem colisão, a variabilidade da latência também dependerá da taxa de colisão.

Dinari (2020) estuda o desempenho em sistemas distribuídos para diferentes tecnologias de *middleware*, avaliando o impacto em se utilizar alguns tipos distintos de comunicação entre processos. No trabalho, os tipos analisados foram: CORBA, RMI, Socket, SOAP, REST, XML-RPC e MQ. No experimento são enviadas mensagens em 6 tamanhos diferentes escolhidos - 3, 8, 15,7, 26,6, 60,2, 341 MB – essas mensagens foram enviadas por um cliente para um servidor gerando curvas de comportamento em diferentes formas de comunicação. No trabalho são analisadas a velocidade dessa comunicação e o consumo de memória. Esse experimento foi executado 5 vezes e, no trabalho em questão, a comunicação via sockets teve o melhor desempenho encontrado.

[5] Netperf, segundo De Sales (2000) é O Netperf é uma ferramenta de benchmark para performance de rede de computadores, desenvolvida pela Divisão de Informação de Redes da Hewlett- Packard Company. É composta de dois programas básicos - netperf e netserver. Baseado no modelo cliente-servidor, onde o tráfego é gerado de uma estação monitora, com o programa netperf, até a estação refletora, com o netserver, no qual o tráfego é refletido e retomado à estação monitora.

Rysak (2023) estuda as diferenças de velocidade de execução de código escrito na linguagem C e Python. Para isso foram executados três algoritmos diferentes em ambas as linguagens e medido o tempo de execução. No trabalho ele encontrou uma maior velocidade nas execuções rodadas em C.

No trabalho de Venkataraman (2015) são realizados testes com alguns fatores que afetam o desempenho na comunicação entre processos, tais como mensagem, tamanho, *caches* de *hardware* e agendamento de processos e feitos experimentos para três configurações com tipos de comunicação entre processos distintas. No trabalho em questão, os melhores desempenhos, respectivamente, foram: memória compartilhada fornece a menor latência e maior rendimento, seguido por kernel pipes e, por último, TCP/IP.

Plauska (2022) traz em seu artigo a avaliação da eficiência das linguagens C/C++, MicroPython, Rust e TinyGo, comparando seu desempenho de execução no microcontrolador ESP32. Vários algoritmos populares de processamento de dados e sinais foram implementados nessas linguagens, e seus tempos de execução foram comparados. Os resultados mostraram que as implementações C/C++ foram mais rápidas na maioria dos casos, seguidas de perto por TinyGo e Rust, enquanto os programas em MicroPython foram muitas vezes mais lentos do que as implementações em outras linguagens de programação.

Nos trabalhos acima citados, os cenários comparam desempenhos em sistemas operacionais de acordo com a mudança de alguns parâmetros, como os tipos de sistemas operacionais, linguagens de programação e comunicação entre processos.

Já no trabalho aqui desenvolvido, a comunicação entre processos não varia, ela se dá através do mesmo mecanismo de comunicação via sockets nos dois cenários desenvolvido, numa aplicação distribuída cliente/servidor. O trabalho aqui desenvolvido mede a latência, observando o desempenho entre dois cenários de teste, cada um com respectiva linguagem e IDE utilizadas para executar a aplicação num microcontrolador ESP32 com um computador.

3 CARACTERIZAÇÃO DO EXPERIMENTO

Entendendo que diversos fatores podem influenciar a performance de uma aplicação num sistema distribuído e na busca permanente em adquirir mais informações para contribuir em se buscar melhores resultados, o parâmetro a ser observado nesse trabalho é a latência.

Vale ressaltar que esse não é único fator a ser considerado quando se escolhem os parâmetros de uma aplicação, porém, esse parâmetro escolhido a se medir consegue expressar de forma direta um dos importantes parâmetros de desempenho.

Como descrito nas seções anteriores, o objetivo do trabalho é comparar dois cenários que se distinguem pela linguagem e a IDE utilizadas. Um com a utilização da linguagem do Arduino – já pontuado que é baseada em Wiring, consequentemente na linguagem C/C++ – comparativamente à linguagem MicroPython, derivada de Python, respectivamente uma linguagem compilada e uma linguagem interpretada.

A aplicação é uma comunicação cliente/servidor sem fio utilizando um microcontrolador ESP32, buscando, assim, simular uma comunicação típica de sistema distribuído em Internet das Coisas.

Partindo do que diz a literatura e foi encontrado em outros trabalhos lidos, é esperado como hipótese inicial que o desenvolvimento em Arduino obtenha menores valores de latência que as aplicações em MicroPython. Contudo essa hipótese inicial não esvazia ou invalida a fase de experimentação, pois o experimento científico é uma ferramenta útil e importante quando possível de ser utilizada, através do qual é possível negar ou afirmar as hipóteses iniciais.

Em certos casos, o ganho obtido em se realizar testes pode ultrapassar a confirmação/ negação de uma hipótese inicial e trazer informações e novos comportamentos relevantes que nem se esperavam ou estavam em análise, pois os resultados podem ser muito diversos, trazendo à tona situações inesperadas.

Além disso, quanto mais testes distintos são realizados acerca de um problema, mais informações e metodologias são descobertas e disponibilizadas para outros pesquisadores, possibilitando também o refinamento do método científico.

3.1 Configuração do experimento

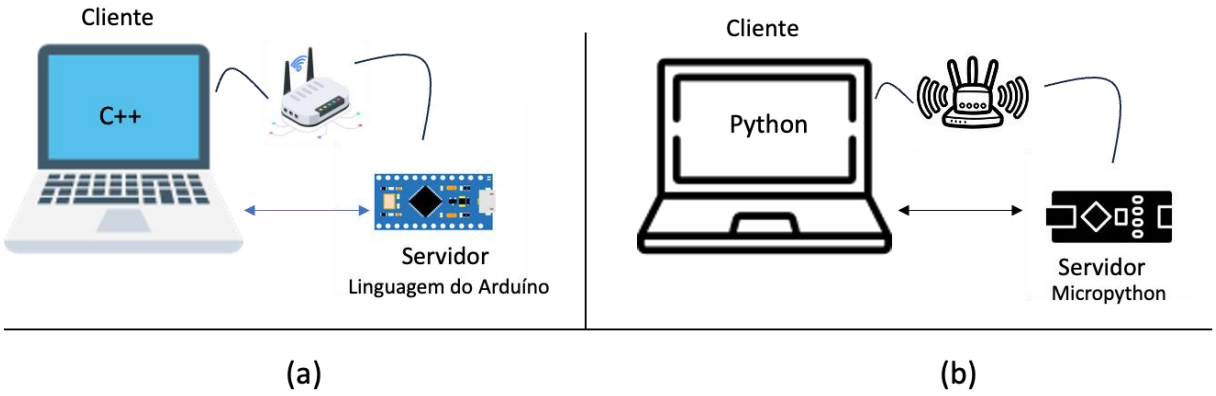
A montagem física do experimento é formada pelos seguintes elementos:

- ESP-32, que irá se comportar como servidor da aplicação.

- Um computador, que irá se comportar como cliente.
- Um dispositivo residencial de roteamento de sinal de internet.

A Figura 6 ilustra os cenários testados, na Figura 6(a) a comunicação no Arduino IDE utilizando a linguagem do Arduino e na Figura 6(b) com IDE Thonny utilizando MicroPython.

Figura 6. Representação da configuração do experimento



Fonte: Autoria própria

A comunicação acontece através de uma rede wi-fi residencial. A comunicação entre os processos é do tipo cliente/servidor, estabelecida por meio de *sockets* e o protocolo de camada de transporte de dados utilizado foi o TCP.

Um *socket* pode ser definido, segundo Manziero (2008) como uma interface de comunicação bidirecional entre processos, que permitem a comunicação entre processos distintos na mesma máquina ou em máquinas distintas, através de uma rede. Pode-se considerar os sockets como a base da comunicação em redes TCP/IP.

O TCP é um protocolo que apresenta um serviço confiável para transporte de dados, garante a conexão uma vez que a comunicação é estabelecida. “O TCP é considerado confiável por oferecer alguns recursos como: sequenciamento, controle de fluxo, retransmissão, identificação de duplicidade, entre outros.” (ALVES, 2008, pg 27).

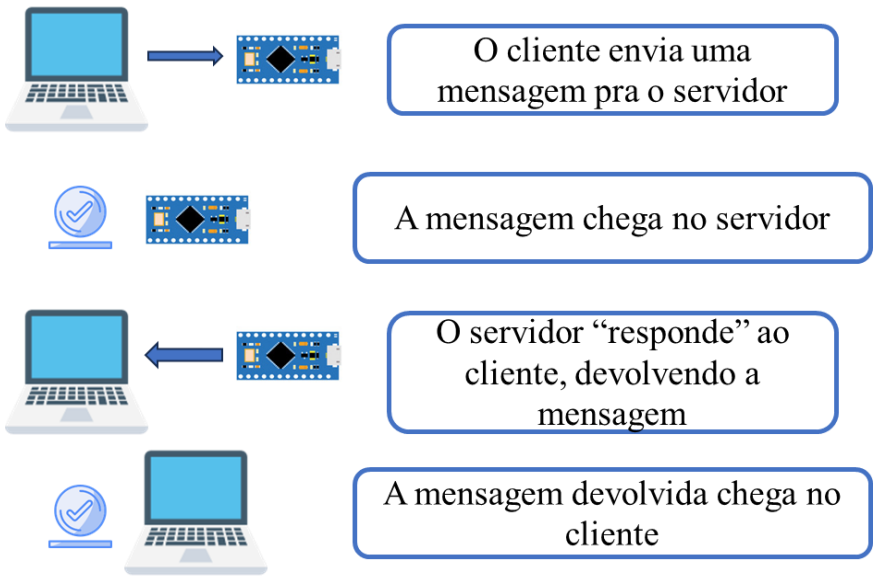
Foram criados 4 programas para o estudo em questão. Dois pares cliente/servidor. Da seguinte maneira:

- Um programa cliente em C++ para realizar comunicação com um programa servidor na linguagem do Arduino. Essa configuração será definida e nomeada doravante como “cenário 1”.
- Um programa cliente desenvolvido em Python para realizar comunicação com um programa servidor na linguagem do Mycropython. Essa configuração será definida e nomeada doravante como “cenário 2”.

O experimento em si consiste no envio de mensagens entre o cliente e o servidor de 11 tamanhos diferentes para cada cenário. Os tamanhos das mensagens foram definidos (não randômicos) e seguiram a seguinte sequência: 1, 2, 4, 8, 16, 32, 64, 128, 256, 512 e 1024 bytes.

O conteúdo da mensagem não é relevante, apenas o tamanho da cadeia de caracteres para análise. A Figura 7 demonstra a sequência ocorrida.

Figura 7. Representação da comunicação executada



Fonte: Autoria Própria

A latência foi medida através de bibliotecas nativas das linguagens, na programação para o cenário 1 foi a “<chrono>” do C++ e para o cenário 2, a biblioteca "time" do Python. A contagem do tempo iniciou com o recebimento da mensagem no servidor até a resposta chegar no cliente, no caso, da segunda até a quarta etapa representadas na Figura 7.

Para cada tamanho de mensagem, essa comunicação de requisição e resposta foi feita 60 vezes seguidas buscando a melhor repetibilidade de forma a obter mais confiabilidade no cálculo do tempo médio, gerando uma população amostral total de 1320 valores.

Para aumento de campo de análise, houve tentativa de aumentar essa faixa de valores, com o envio de mensagens de 2048, 4096 e 8192 bytes, que foram descartados para fins de análise do experimento, pois não se mostraram confiáveis. Essas informações serão relatadas de forma mais detalhada no item de problemas encontrados no capítulo da análise de dados.

Para cada execução de tamanho em cada cenário foi gerado um arquivo de duas colunas e 60 linhas (nos 60 envios consecutivos) e extensão .csv conforme ilustrado na Tabela 1, para exemplificar.

Tabela 1. Arquivo gerado com mensagens de 4 bytes no cenário 1

tamanho texto em Bytes	tempo decorrido
4	0.0061419
4	0.004418
4	0.0033832
4	0.0076897
4	0.0054663
4	0.0053424
4	0.0034764
4	0.0033248
4	0.0043886
4	0.0076718
4	0.0185419
4	0.0035551
4	0.0034648
4	0.0214088
4	0.0076145
4	0.0037563
4	0.0034559
4	0.0034749
4	0.0086847
4	0.0035518
4	0.0098264
4	0.0038076
4	0.0038864
4	0.005141
4	0.0070769
4	0.0168902

4	0.0036771
4	0.0041525
4	0.0066907
4	0.0041265
4	0.0067681
4	0.0033857
4	0.0034358
4	0.0036388
4	0.0036277
4	0.0066383
4	0.0035135
4	0.0041219
4	0.0036078
4	0.0034251
4	0.0035092
4	0.004523
4	0.0038647
4	0.0089568
4	0.0036087
4	0.0033273
4	0.019888
4	0.0059825
4	0.0035789
4	0.0046924
4	0.0034332
4	0.0033133
4	0.0036303
4	0.0033586
4	0.0035423
4	0.0068545
4	0.0035306
4	0.013656
4	0.0033749
4	0.0033276

Fonte: Autoria Própria

Através da execução dos programas, foram gerados e gravados ao todo 28 arquivos, todos de mesmo formato que o exemplificado da Tabela 1. E, utilizados 22 para análise dos valores. Tendo em vista que os arquivos das mensagens de 2048, 4096 e 8192 bytes foram descartados tanto para o cenário 1 quanto para o cenário 2, totalizando 6 arquivos .csv.

Os arquivos gerados no cenário 1 foram nomeadas com o termo “cpp” nos arquivos exportados em suas tabelas de valores, esse termo é relativo à extensão dos arquivos na linguagem C++. Assim como, os arquivos gerados no cenário 2 apresentam o termo “python” nos arquivos. Para se evitar confusão na apresentação dos gráficos no capítulo 4, será adotado o termo cenário para uniformizar a apresentação dos dados, contudo é possível ver os termos “cpp” e “python” em trechos do algoritmo de análise de dados que serão exibidos no decorrer das explicações. Esse esclarecimento é interessante para que não surjam dúvidas na leitura do capítulo 4.

3.2 Especificação técnica dos dispositivos

Para fins de conhecimento, seguem as especificações técnicas dos equipamentos utilizados no experimento.

O Módulo ESP32, mostrado na Figura 8 é do tipo Controlador ESP-WROOM-32 e foi escolhido por se tratar de um *hardware* livre que já possui Wi-Fi de forma nativa, dispensando a necessidade de grande estrutura para conexão, além de possuir grande bibliografia de aplicações em *IoT*.

Figura 8. ESP32 utilizado

Fonte: Autoria própria

Algumas características estão informadas na Tabela 2 abaixo.

Tabela 2. Especificação técnica ESP32

BLUETOOTH	v4.2 BR / EDR e BLE
CLOCK	até 240MHz
CORRENTE DE OPERAÇÃO (TÍPICA)	80mA
FAIXA DE FREQUÊNCIA DO WIFI	2,4 a 2,5GHz
FLASH	4 MB;
MICROPROCESSADOR	Tensilica Xtensa 32-bit LX6
NÍVEL LÓGICO	3.3V
PORTAS GPIO	30
ROM	448KB
SOC (SYSTEM ON CHIP)	ESP32-D0WDQ6
SRAM	520KB
SUPORTE A REDES WIFI	802.11 b/g/n
TEMPERATURA DE OPERAÇÃO	-40º a 85º celsius
TENSÃO DE OPERAÇÃO	2,2V - 3,6VDC

Fonte: Espressif, adaptado. Disponível em: https://www.espressif.com/en/products/socs/esp32

O computador utilizado segue como cliente possui processador intel i5-10300H, 16GB de memória RAM gDDR4 2933mhz, armazenamento de disco com SSD placa de rede Intel(R) Wireless-AC 9560.

O roteador possui as seguintes especificações técnicas:

- Alimentação: 100-240 V a 50/60 Hz.
- Padrões: ieee 802.11 a/b/g/n/ac.
- Velocidade Wireless: 867Mbps em 5 GHz e 300Mbps em 2,4 GHz.
- Frequência de Operação: 2,4 GHz e 5 GHz.
- Antena: 4 antenas fixas de 5 dBi.
- Potência máxima: 2,4 GHz - 158 mW (22 dBm) e 5 GHz - 158 mW (22 dBm).

3.3 Ferramentas para análise de dados

A linguagem que se utilizou para tratamento dos dados coletados no trabalho foi o Python. Essa linguagem, como foi pontuado em Comarela (2019), é bastante utilizada em ciência de dados, pois é versátil, simples e conta com uma vasta quantidade de recursos, funções e bibliotecas para essa finalidade.

Para a análise de dados existem ferramentas disponíveis e poderosas pertencentes e/ou compatíveis ao Python. Podemos destacar algumas bibliotecas, inclusive, utilizadas nesse trabalho, que são: Numpy, Pandas e Matplotlib, além de ferramentas interativas como Jupyter Notebook.

"*Numerical Python* (Python Numérico) ou NumPy é a biblioteca em Python padrão para o suporte à utilização de matrizes e arrays[6] multidimensionais de grande porte, e vem com uma vasta coleção de funções matemáticas de alto nível para operar nestas arrays." (numpy.org, 2023. s.n.).

Como escrito em Mckinney (2018), Numpy oferece já há muito tempo, a maioria das aplicações científicas que envolvam dados numéricos em Python. Além disso:

> Além dos recursos de processamento rápido de arrays que a NumPy acrescenta ao Python, um de seus principais usos em análise de dados é como um contêiner para que dados sejam passados entre algoritmos e bibliotecas. Para dados numéricos, os arrays NumPy são mais eficientes para armazenar e manipular dados do que as outras estruturas de dados embutidas (built-in) de Python. Além do mais, as bibliotecas escritas em uma linguagem de mais baixo nível, como C ou Fortran, podem operar em dados armazenados em um array NumPy sem copiar dados para outra representação em memória. Assim, muitas ferramentas de processamento numérico para Python supõem os arrays NumPy como uma estrutura de dados principal ou têm como meta uma interoperabilidade suave com a NumPy. (MCKINNEY, 2018, p.27)

Conforme Mckinney (2018), a biblioteca Pandas é uma biblioteca mais recente, criada em 2010. Foi escrita em Python, Cython e C e tem a biblioteca Numpy como base de sua construção, sendo assim, uma outra ferramenta poderosa com dados numéricos.

[6] Array é uma estrutura de dados que consegue guardar vários elementos. Os elementos em um array são acessados através de um índice. Fonte: Almeida (2023).

O Pandas é definido como “uma ferramenta de análise e manipulação de dados de código aberto rápida, poderosa, flexível e fácil de usar, construído sobre a linguagem de programação Python.” (pandas.pydata.org, 2023. s.n., adaptado).

Segundo descrito por Almeida (2023), o Pandas trabalha com Series e Dataframes. Series são objetos arrays unidimensionais, com rótulos. O Dataframe é “uma estrutura de dados tabular, orientada a colunas, com rótulos (labels) tanto para linhas quanto para colunas” (Mckinney, 2018, p.27), ou seja, são objetos bidimensionais.

Segundo Mckinney (2018, p.28) “combina as ideias de processamento de alto desempenho de arrays da NumPy com os recursos flexíveis de manipulação de dados das planilhas e dos bancos de dados relacionais (como o SQL).”

A terceira biblioteca mencionada, a Matplotlib, “é uma biblioteca abrangente para criar visualizações estáticas, animadas e interativas em Python.” (matplotlib.org, 2023. s.n., adaptado).

De acordo com Mckinney (2018), é dedicada a geração de gráficos é a mais famosa biblioteca Python para gerar gráficos bidimensionais, não é a única capaz de gerar esse tipo de visualizações no Python, mas é a mais utilizada para plotagens pelos programadores da linguagem.

De acordo com informações de Almeida (2023), o Jupyter Notebook é um *notebook* interativo, bastante utilizado junto com o Pandas, pode ser classificada como uma espécie de IDE apresentando grande utilidade tanto para desenvolvedores como para profissionais de Ciência de Dados. “Jupyter ganhou força em muitos campos como um ambiente de código aberto compatível com inúmeras linguagens de programação. O nome Jupyter é uma referência às três linguagens principais suportadas pelo projeto (Julia, Python e R).” Dombrowski (2023, s.n.).

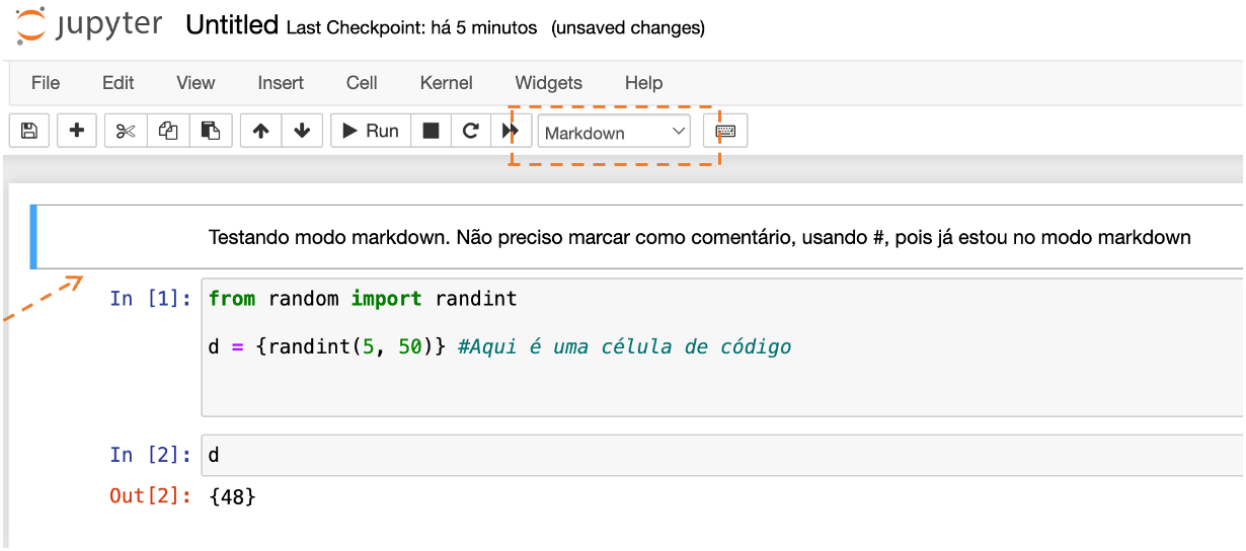
É possível inserir textos formatados, imagens, gráficos e equações matemáticas na criação, explorando a praticidade do modo interativo, conforme informado no site oficial Jupyter.org.

Segundo Miranda (2023), temos a opção de criar blocos de texto e blocos de código no *notebook*. Cada bloco é conhecido como uma célula do *notebook* e é possível mesclar células de texto e células de códigos de forma organizada, ou seja, é um ambiente altamente interativo de experimentação. Enquanto se escreve código e já é possível visualizar prontamente a sua saída, dispensando a necessidade de executar todo o código do início quando se quer visualizar

e/ou testar apenas uma célula especificamente, pois o processo fica armazenado na memória e os blocos vão funcionar de forma parcialmente independente.

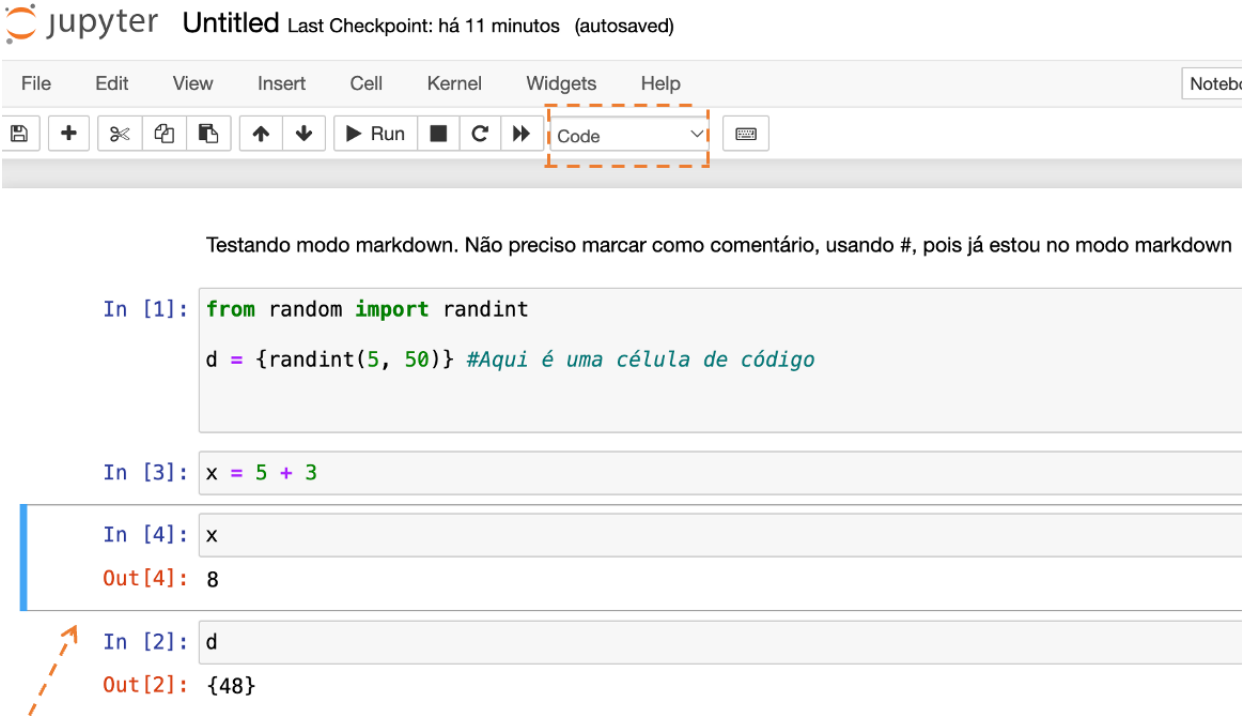
Na Figura 9 e na Figura 10 é possível ver algumas das informações comentadas.

Figura 9. Exemplo no Jupyter Notebook com destaque na linha de comentário



Fonte: Autoria própria

Figura 10. Exemplo no Jupyter Notebook com destaque na linha de código



Fonte: Autoria própria

Na Figura 9, a célula do tipo *markdown* está destacada com uma seta. Nesse tipo de célula podemos escrever observações de texto livremente, sem a necessidade de destacar com a cerquilha (#).

Na Figura 10, uma célula de código está destacada. É possível notar também que essa célula foi executada e inserida posteriormente a variável d gerada randomicamente no código, que está presente na Figura 9. A inserção da equação de "x" não fez com que o sorteio do valor "d" fosse executado novamente. Executar o código por completo ou apenas células específicas é possível, a depender do desejo do programador quando está manipulando o Jupyter Notebook.

3.4 Análise e tratamento dos dados

Os dados passaram por um processo de tratamento antes da geração dos gráficos com as médias de latência, pois não seria interessante simplesmente executar uma média aritmética dos 60 valores em cada caso. É preciso estudar os dados para que seja possível extrair deles informações relevantes, desconsiderando informações que poderiam apenas "poluir" os valores relevantes.

Esse processo de preparação e tratamento dos dados antes de fato extrair os valores que são o foco da análise é muito comum quando se manipulam dados, principalmente um volume alto. Muitas vezes os cientistas de dados gastam mais tempo em entender e preparar os dados que propriamente executando os algoritmos específicos do objetivo final.

Para conseguir trabalhar os dados da análise em questão, foram feitas manipulações que podem ser enquadradas como a técnica ETL. Como definido em Tiesse (2023) APUD Vida et. al. (2021), ETL, do inglês *extract, transform and load*, ou seja: extrair, transformar e carregar. Essas etapas representam extrair os dados de onde estejam, realizar as transformações necessárias para que possam ser manipulados e carregá-los, ou seja colocá-los numa base consolidada.

O que foi realizado em cada etapa foi:

- Extração: Os arquivos .csv gerados pelos programas executados nos dois cenários descritos acima foram carregados a partir da pasta indicada e importados para o Jupyter Notebook, conforme apresentado na Figura 11. Um dataframe vazio já foi criado para que todos os valores oriundos de um arquivo .csv sejam adicionados dentro dele, conforme descrito na próxima etapa da manipulação dos dados.

Figura 11. Extraindo os arquivos com dados a partir da pasta original

```
# Identificando a pasta que contém os arquivos gerados csv
directory_paths = [r"/Users/macbookjessica/Documents/Mestrado/Tratamento/Dados"]
# Inicia uma lista vazia para armazenar DataFrames com nomes de arquivos
dfs = []

for directory_path in directory_paths:
    # Lista todos os arquivos na pasta
    files = os.listdir(directory_path)
    # Filtrando apenas os arquivos csv dentro da pasta
    csv_files = [file for file in files if file.endswith('.csv')]
    # Percorrendo arquivos CSV e lendo-os em DataFrames
```

Fonte: Autoria própria

- Carregamento: Foi gerada uma tabela unificando todas as tabelas dos arquivos gerados formando uma tabela única padronizada com todos os tamanhos de mensagens e seus respectivos tempos com os rótulos das colunas padronizados. A Figura 12 demonstra essa etapa.

Figura 12. Juntando os dados numa tabela única com seus respectivos rótulos

```
for directory_path in directory_paths:
    # Lista todos os arquivos na pasta
    files = os.listdir(directory_path)
    # Filtrando apenas os arquivos csv dentro da pasta
    csv_files = [file for file in files if file.endswith('.csv')]
    # Percorrendo arquivos CSV e lendo-os em DataFrames
    for csv_file in csv_files:
        file_path = os.path.join(directory_path, csv_file)
        # Lendo o arquivo csv dentro DataFrame
        df_temp = pd.read_csv(file_path, sep=";", skiprows=1, names=["Tamanho","Tempo"])
        # Adicionando uma nova coluna com o nome do arquivo, agregando os cpp em uma lista e python outra
        if csv_file.endswith('cpp.csv'):
            df_temp['Linguagem'] = 'cpp'
            # Adicionando um DataFrame, um vetor a lista
            dfs.append(df_temp)
        elif csv_file.endswith('python.csv'):
            df_temp['Linguagem'] = 'python'
            # Adicionando um DataFrame, um vetor a lista
            dfs.append(df_temp)
# Concatenando os data frames em uma lista unica com todos
df = pd.concat(dfs, ignore_index=True)

# Grava o DataFrame combinado em um novo arquivo CSV
df.to_csv('combined_data_with_filenames.csv', sep=";", index=False)
```

Fonte: Autoria própria

Após essa concatenação, uma lista com 1320 valores (22 arquivos com 60 valores cada) devidamente rotulado com seus respectivos linguagem e tamanho é criada, conforme pode ser visto na Figura 13, onde foi requerida uma amostra com as primeiras e últimas linhas dessa lista.

Figura 13. Amostra exibindo primeiras e últimas linhas da lista gerada

```
In [2]: df.head()
```

Out[2]:

	Tamanho	Tempo	Linguagem
0	1	0.105351	cpp
1	1	0.004858	cpp
2	1	0.004189	cpp
3	1	0.005768	cpp
4	1	0.015764	cpp

```
In [3]: df.tail()
```

Out[3]:

	Tamanho	Tempo	Linguagem
1315	2	0.008967	python
1316	2	0.005563	python
1317	2	0.010144	python
1318	2	0.008007	python
1319	2	0.007772	python

Fonte: Autoria própria

- Transformação: Esses valores passaram por algumas alterações: Os valores de tempo gerados pelo programa estão em segundos, esses valores foram multiplicados por 1000 gerando valores em milissegundos, que no caso em questão fica numa ordem de grandeza mais representativa. Figura 14.

Figura 14. Conversão de tempo para milissegundos

```
In [6]: #Converts Tempo to milliseconds
        df["Tempo"] = df["Tempo"] * 1000
```

Fonte: Autoria própria

Outro tratamento essencial antes que se calculasse a média dos valores de latência foi a extração de outliers, essa etapa merece um melhor detalhamento, que seguirá abaixo.

Conforme visto em Prates (2017), os *outliers* são dados que se diferenciam drasticamente de todos os outros em uma amostra. Em outras palavras, um outlier é um valor que foge da normalidade do comportamento analisado e que pode (e provavelmente irá) causar anomalias nos resultados obtidos por meio de algoritmos e sistemas de análise.

Há casos em que os valores incomuns podem sim fazer parte da amostra considerada, ser um dado real apesar da discrepância. Por exemplo, ao elencar salários em uma empresa de produção fabril com muitos funcionários e poucos diretores, é de se esperar que haja poucos valores acima da média dos demais, e que esses valores fazem sim parte da amostra, corretamente coletados.

Já numa amostra de cadastro de idades de pessoas de uma cidade, caso apareça algum valor muito elevado, como por exemplo, 901, podemos afirmar que ocorreu um erro de digitação de quem realizou o cadastro do dado e a idade seria provavelmente 91, ou seja, aquele valor provavelmente não é verdadeiro na amostra estudada.

Nem sempre a escolha é descartar esse valor incomum, pois há casos que o outliers são justamente os valores procurados. Como citado em Valadares (2012), empresas que buscam proteção a fraudes financeiras, os outliers são justamente o foco da busca, pois aqueles comportamentos de compras extremamente atípicos do cliente, pode indicar situações de fraudes. De qualquer forma, o outlier é um ponto de atenção e é preciso decidir o que se fazer com ele.

Para os casos em que esses dados analisados são entendidos como dados espúrios, que não representam a realidade da situação analisada, é boa prática fazer o descarte desses para evitar que eles venham a comprometer a representatividade dos dados extraídos. Essa escolha vai depender de alguns fatores, dentre eles, da natureza dos dados e da experiência do analista.

No caso da análise tratada, o tempo médio é o valor a ser extraído, que é obtido através da média do tempo dos 60 “disparos” de mensagens enviadas seguidamente. Entende-se que os valores para a mesma configuração e mesmos tamanhos deveriam ser teoricamente iguais, ou pelo menos navegar em torno de uma faixa estreita. Então, assim sendo, os valores de outliers provavelmente são pontos fora da curva resultados de problemas de qualidade ou interferências não identificadas na transmissão ou atrasos dos demais equipamentos que acessam o mesmo roteador. Após encontrados, os outliers desse trabalho foram descartados, de forma a não falsear o valor médio real de envios, já que a média é altamente influenciada pelos outliers.

Segundo Valadares (2012), existem várias técnicas para se encontrar os outliers em uma amostra, algumas técnicas com maior rigor matemático, mas também há formas mais subjetivas para se identificar pontos discrepantes, auxiliadas por análises gráficas e conhecimentos da natureza dos dados em questão. Existem métodos baseados em visualização de dados, métodos

gráficos, métodos estatísticos e eles todos podem inclusive serem implementados em aprendizagem de máquina.

A técnica adotada nesse trabalho foi um método estatístico que considera o intervalo interquartil, também conhecido como *IQR*, do inglês *Interquartile Range*.

O método de intervalo interquartil, mostra como os dados estão espalhados sobre a mediana. É menos suscetível do que o intervalo a valores discrepantes e pode, portanto, ser mais útil.

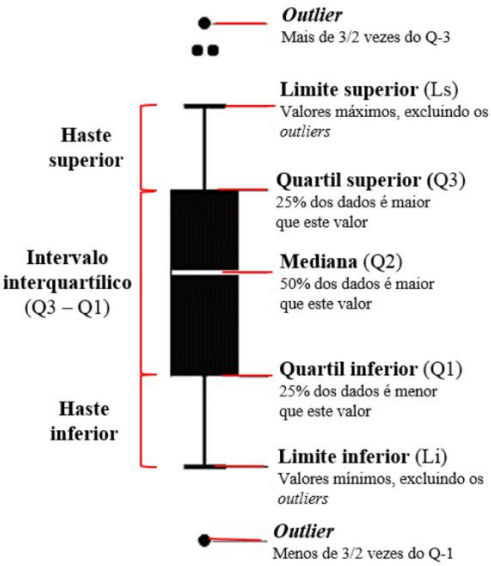
Como estudado em Neto (2017), esse método é baseado nos quartis, que são valores que dividem um conjunto de dados analisados ordenados em quatro partes iguais. Existem três quartis principais: o primeiro quartil (Q1), o segundo quartil (Q2) e terceiro quartil (Q3). O segundo quartil, ou mediana, divide o conjunto de dados em duas partes iguais, ou seja, metade dos dados está abaixo da mediana e a outra metade está acima.

O primeiro quartil (Q1) marca o limite inferior do segundo quarto dos dados, ou seja, 25% dos dados estão abaixo do primeiro quartil e 75% estão acima. O terceiro quartil (Q3) marca o limite superior do terceiro quarto dos dados, ou seja, 75% dos dados estão abaixo do terceiro quartil e 25% estão acima.

O IQR é a diferença entre o terceiro quartil e o primeiro quartil: IQR = Q3 - Q1. Ele fornece uma medida de variabilidade robusta em relação a outliers, sendo útil para a detecção de valores extremos.

Os quartis são amplamente usados em estatística descritiva e na construção de gráficos de caixa (*box plots*), representado na Figura 15, onde o retângulo central representa o intervalo interquartil. Valores acima ou abaixo dos quartis podem indicar a presença de outliers ou informações sobre a distribuição dos dados.

Figura 15. Exemplo básico de uma distribuição em boxplot



Fonte: Neto, 2017

Para calcular os limites superiores e inferiores e consequentemente os outliers, a regra padrão é considerar uma vez e meia para baixo e para cima do intervalo interquartil, ou seja, o limite inferior é dado por Q1 - 1,5 X IQR, e o limite superior é dado por Q3 + 1,5 * IQR e valores abaixo do limite inferior e acima do superior são considerados outliers. Essa regra foi aplicada, conforme exibido na Figura 16.

Figura 16. Código para identificação e remoção dos outliers

```
for l in Linguagens:
    for t in Tamanhos:
        # Essa parte do tratamento é para retirar os outliers.
        data = df[(df["Linguagem"]==l)&(df["Tamanho"]==t)]["Tempo"]

        # Calculando (Q1 and Q3)
        Q1 = np.percentile(data, 25)
        Q3 = np.percentile(data, 75)

        # Calculando the IQR (Interquartile Range)
        IQR = Q3 - Q1

        # Definindo os limites inferiores e superiores
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Identificando outliers
        outliers = data[(data < lower_bound) | (data > upper_bound)]

        # Removendo outliers
        df.drop(outliers.keys(), inplace=True)
```

Fonte: Autoria própria

Apesar do arquivo ser unificado, o processo de remoção de outliers foi executado em laço, como visto na Figura 16. Portanto, para cada grupo de 60 valores, olhando separadamente cada caso de acordo com o tamanho de mensagem e configuração de cenário (no algoritmo gerado definido pela coluna “linguagem”) foi extraído os seus outliers específicos encontrados.

Só após os dados terem passado por esse processo, ou seja, extraídos os valores que podem foram entendidos como não válidos, as médias dos valores de latência foram calculadas. No próximo capítulo é possível ver esse comportamento de forma gráfica, assim como os valores obtidos.

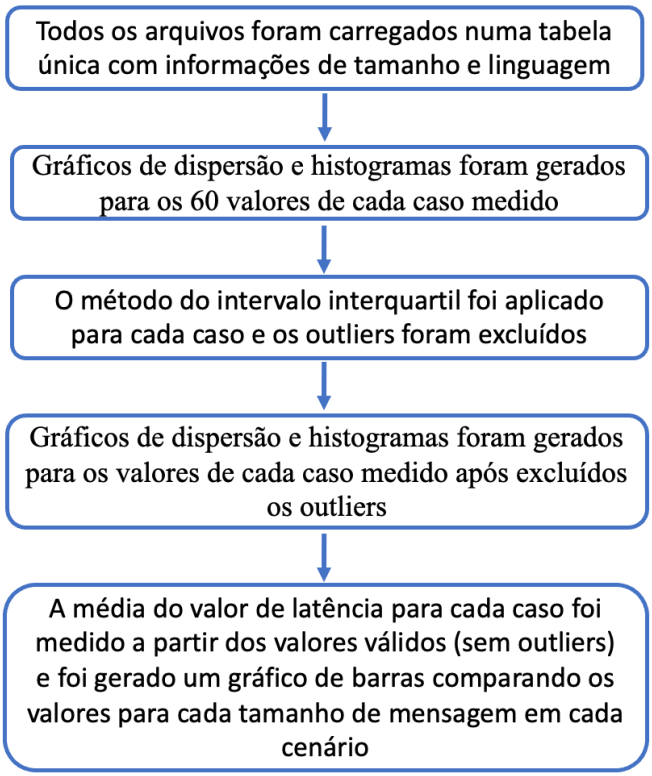
4 ANÁLISE DOS RESULTADOS

Pelo exposto no capítulo anterior, sempre que for possível, a etapa de testes é um recurso muito importante para ajudar a verificar hipóteses, bem como aumentar a quantidade de opções de ferramentas e informações no desenvolvimento de tecnologias e soluções para esse e outros problemas.

Após reunir todos os dados medidos nos programas em uma lista única, foram gerados gráficos através de laços no código, segregando os 60 envios de cada mensagem para cada cenário de teste. Foram gerados gráficos de dispersão e histogramas para auxiliar na observação dos comportamentos dos valores e a forma de distribuição. Assim, visualizando essa distribuição antes e depois de serem retirados os outliers.

Para recapitular as etapas desenvolvidas no Jupyter Notebook no tratamento dos dados, segue representado na Figura 17 os passos de forma resumida:

Figura 17. Resumo do tratamento dos dados

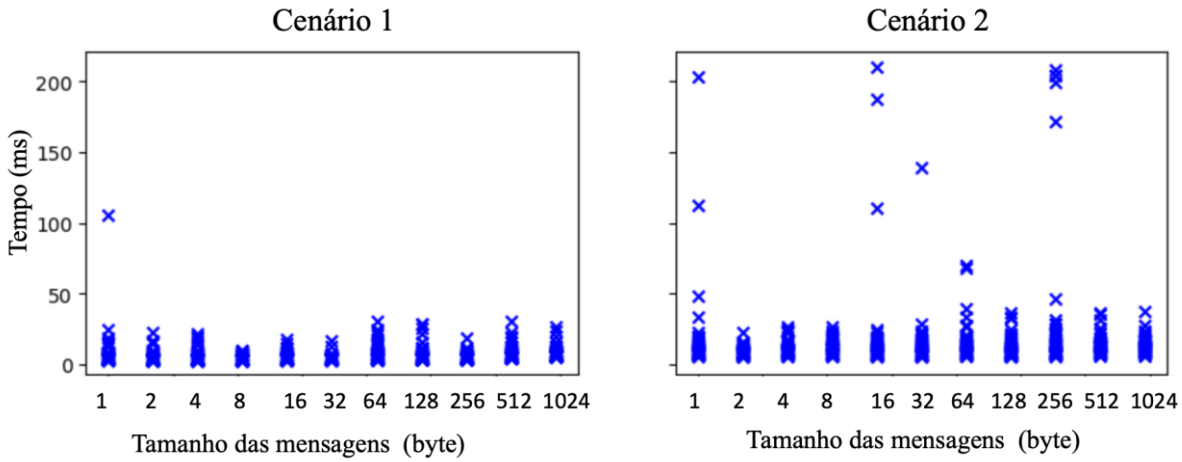


Fonte: Autoria própria

Observando inicialmente os gráficos de dispersão apresentados na Figura 18 no qual cada “x” representa um valor de medida de tempo, é possível identificar que há alguns valores bem diferentes dos demais.

Há maior concentração de numa faixa bem próximos, fazendo os “x” ficarem densos como uma barra próxima a valores abaixo de 30 ms e alguns poucos pontos dispersos em valores afastados, próximos a 200 milissegundos. Esses valores são outliers.

Figura 18. Gráfico de dispersão dos dados antes da extração dos outliers

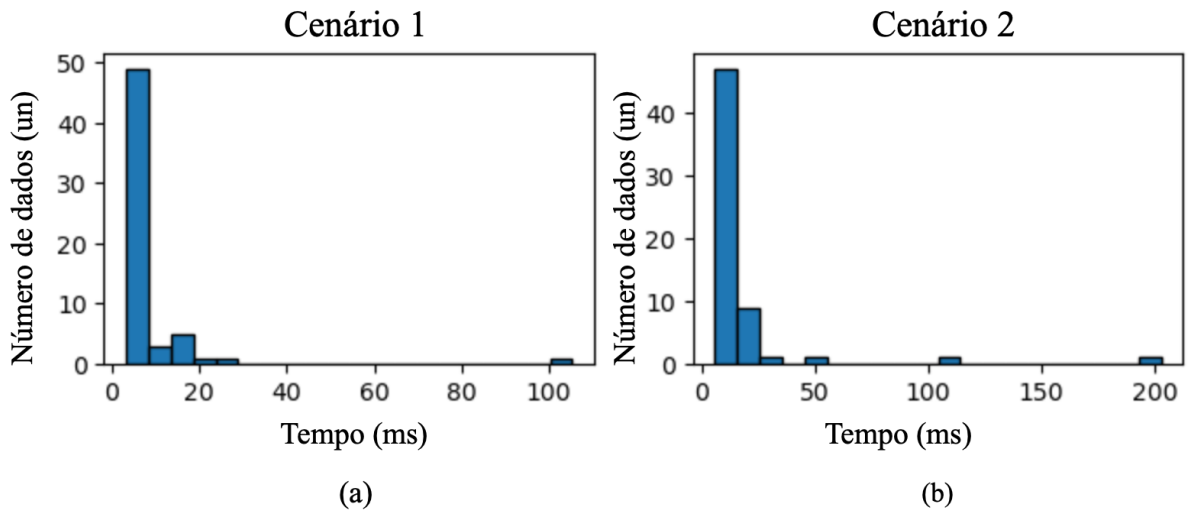


Fonte: Autoria própria

Outra forma de perceber que há pequena quantidade de valores fora das faixas de tempo maiores que 30 ms, é visualizando os histogramas gerados. Não há necessidade de colocar todos os 22 histogramas gerados, pois o gráfico de dispersão já representa bem esse comportamento.

Para fins de exemplificação, a Figura 19 mostra um histograma para o valor de 1 byte. Na Figura 19 (a) está o histograma para o cenário 1 e na Figura 19 (b) para o cenário 2. O eixo da vertical expressa a quantidade de valores existentes e no eixo horizontal a faixa dos valores de tempo, em milissegundos. É possível notar que maior parte dos valores, praticamente 50 pontos dentre os 60 existentes estão concentrados em faixas de menores valores de tempo, enquanto uma minoria encontra-se para valores em centenas de milissegundos.

Figura 19. Histograma para mensagem de 1 byte antes de extração de outliers

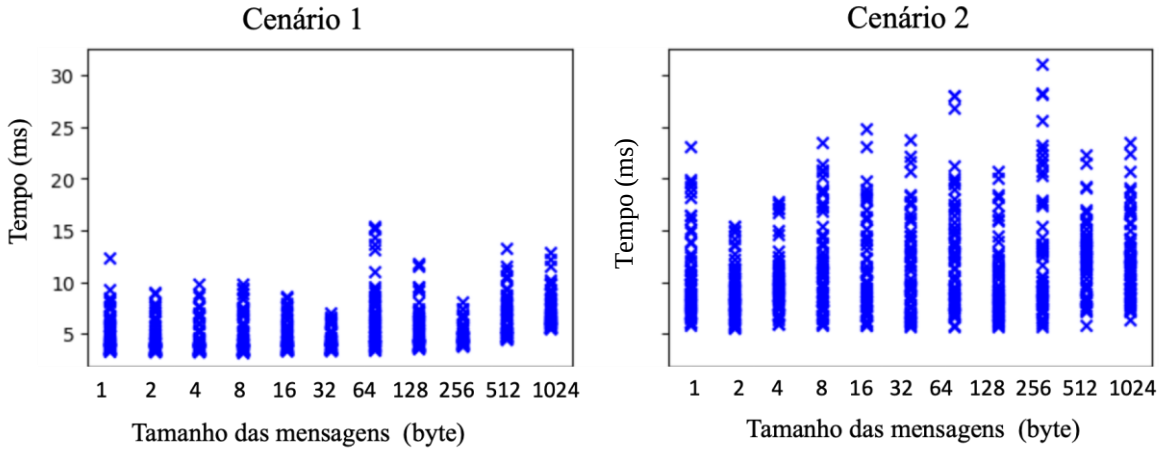


Fonte: Autoria própria

Nem todos os valores possuem outliers tão óbvios e perceptíveis no histograma, por isso para detecção foi escolhida a regra dos *IQR*.

Após realizados os cálculos e a extração dos outliers, o gráfico de dispersão dos dados, ficou conforme Figura 20.

Figura 20. Gráfico de dispersão dos dados após a extração dos outliers

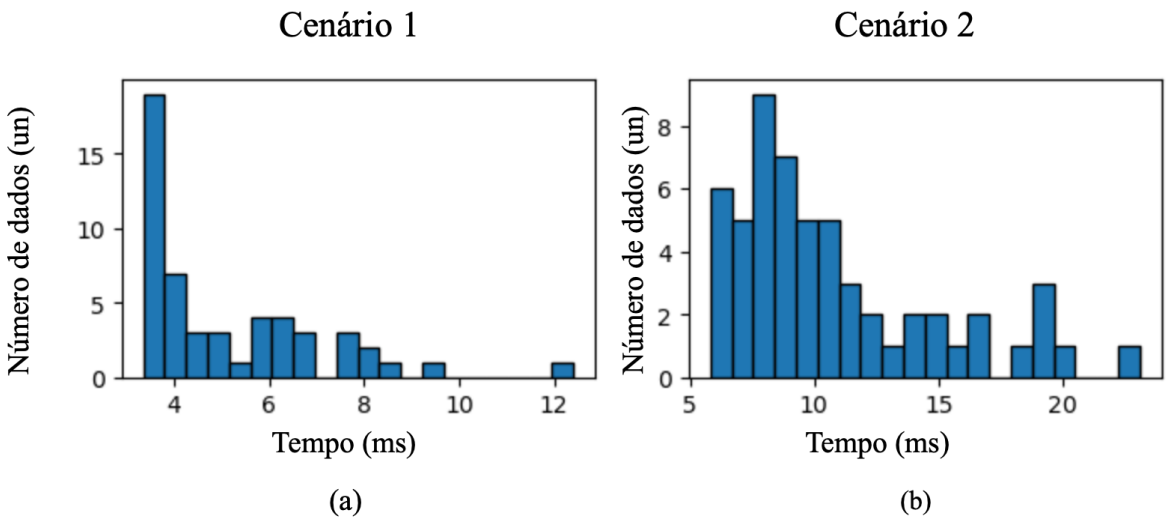


Fonte: Autoria própria

Como visto na Figura 20, os dados após extração de outliers estão situados em uma faixa bem mais estreita de valores, concentrando-se em uma faixa de valores mais coerentes com o comportamento esperado.

Comparando o Histograma pós extração de outliers para o valor de 1 byte, os valores estão distribuídos claramente numa faixa de tempo menor. Concentrados em valores em torno de 2 a 12 ms para o cenário 1 e 5 a 20ms para o cenário 2, conforme é possível ver na Figura 21.

Figura 21. Histograma para mensagem de 1 byte após de extração de outliers



Fonte: Autoria própria

É interessante notar que o outliers encontrados nesse trabalho eram basicamente valores acima da faixa de tempo predominante, ou seja, outliers superiores, o que nos leva a crer que esses valores acima sejam interferências durante o envio, que tornaram o tempo mais longo para a execução.

Algumas hipóteses prováveis que fizeram com que a entrega ocorresse de forma atrasada podem ser resultado da fila de execução de outras aplicações, já que a rede era compartilhada com outras requisições além do experimento, existiam outros equipamentos conectados a rede. Outra possibilidade seria decorrente do atraso no gerenciador das aplicações do próprio computador. Além de algum possível atraso do roteador, o que também pode ter gerado impacto nesses envios/resposta mais demorados.

Todas essas questões são levantadas e podem servir de inspiração para análises futuras. Mas o fato de a aplicação não ter como ser rodada em um ambiente “isolado” não invalidam a análise, pois todo os testes executados aconteceram sob as mesmas condições, mantendo o padrão. Além de que, numa aplicação real de um usuário, também existem fatores não controláveis num uso prático de desenvolvedores de aplicações, seja *home office*, seja num ambiente de escritório.

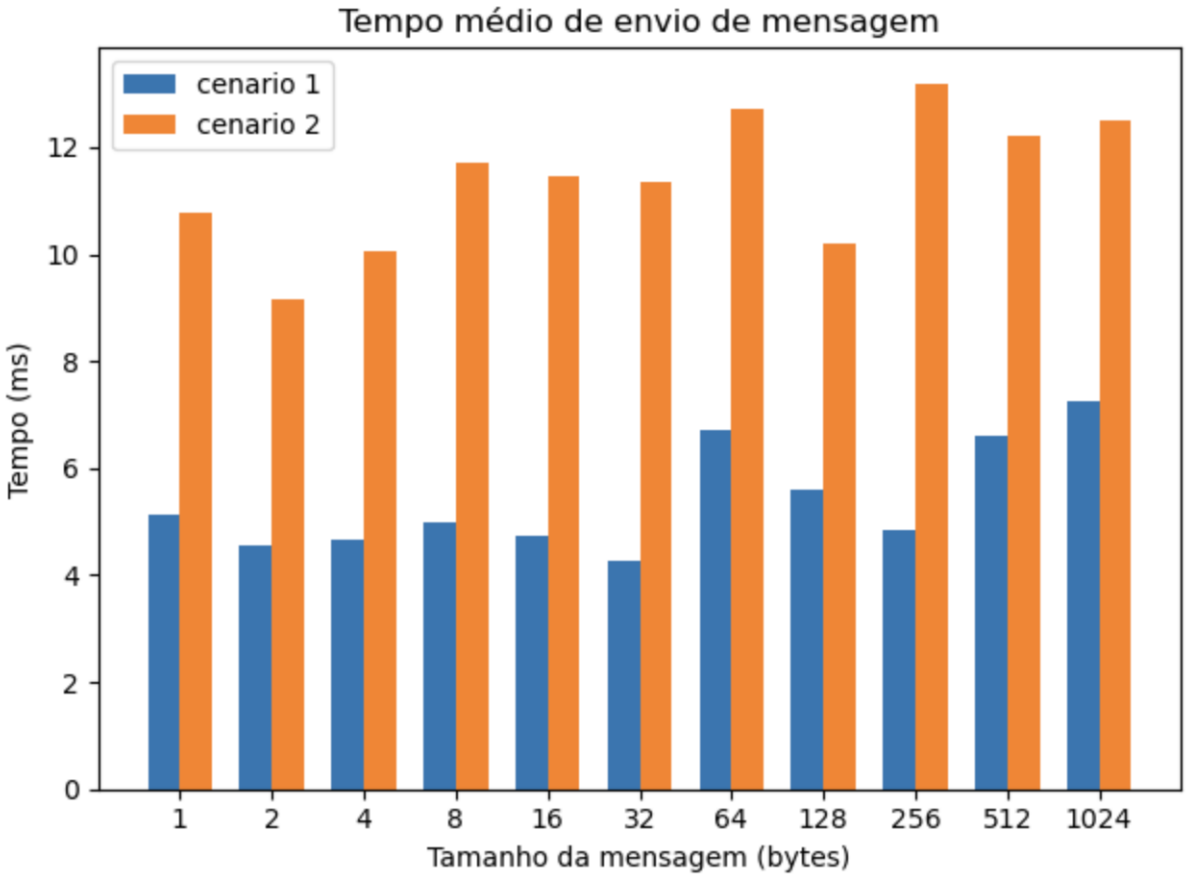
Com os dados considerados válidos, foram extraída as médias de tempo medidas, apresentadas na Tabela 3 e gerado gráficos para visualização dos resultados, como demonstrado no gráfico da Figura 22.

Tabela 3. Média dos valores de tempo para cada teste.

Tamanho (bytes)	Tempo médio (ms)		Diferença absoluta entre o cenário 2 e o 1 (ms)	Diferença % de tempo cenário 2 em relação ao 1
	Cenário 1	Cenário 2		
1	5,147	10,763	5,616	109%
2	4,560	9,146	4,586	101%
4	4,651	10,057	5,406	116%
8	4,991	11,710	6,719	135%
16	4,738	11,442	6,705	142%
32	4,279	11,344	7,065	165%
64	6,721	12,715	5,995	89%
128	5,607	10,188	4,581	82%
256	4,843	13,193	8,350	172%
512	6,614	12,228	5,615	85%
1024	7,254	12,489	5,235	72%

Fonte: Autoria própria

Figura 22. Tempo médio de envio de mensagens por tamanho e linguagem



Fonte: Autoria própria

Como visto na tabela e gráficos acima, o desempenho mais rápido obtido foi através do cenário 1, que utilizou a linguagem do Arduíno, que é uma linguagem compilada, conforme hipótese inicial baseadas na literatura pesquisada, chegando a demorar muitas vezes, menos da metade do tempo equivalente no cenário 2. Foi observado também que ela apresentou menor dispersão dos valores, possuindo maior repetibilidade.

4.1 Observações adicionais levantadas

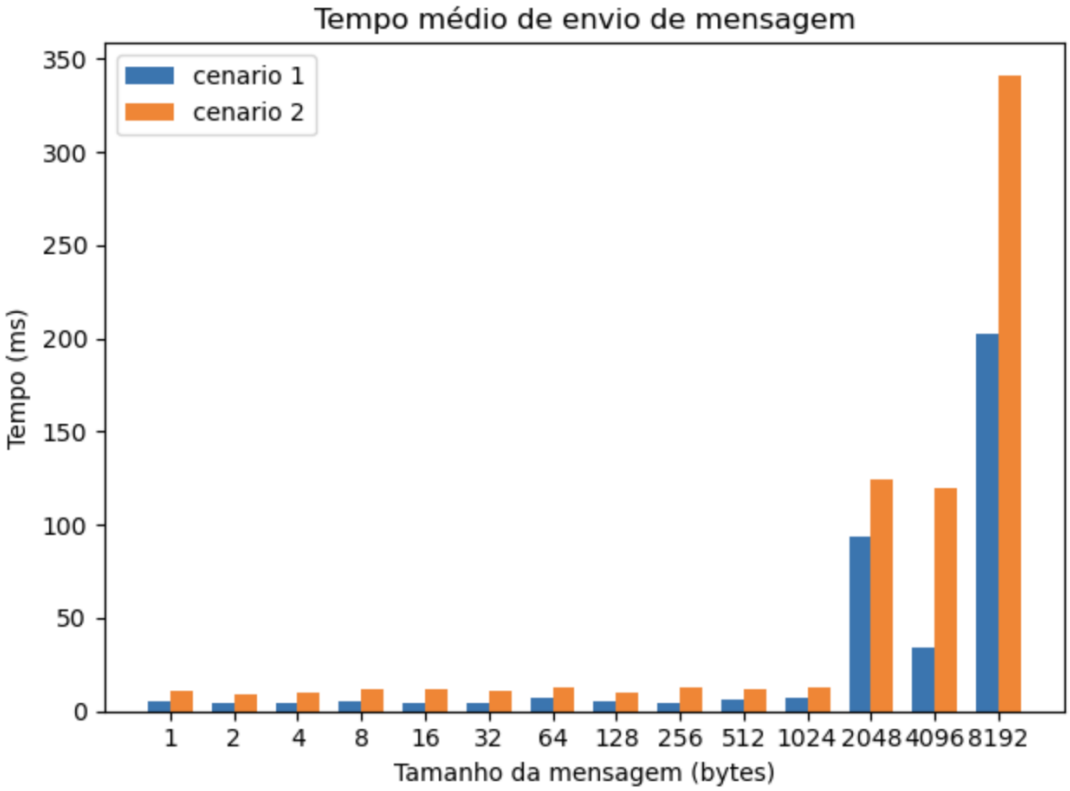
Durante a montagem, mensagens testadas de tamanho 2048, 4096 e 8192 apresentaram valores dispersos e muito fora da ordem de grandeza dos demais, como demonstrado na Tabela 4 e no gráfico da Figura 23.

Esses dados não trouxeram uma regularidade que permitisse sua confiabilidade estatística, sendo decidido que esses dados seriam desprezados para a comparação, pois não evidenciavam o comportamento confiável.

Tabela 4. Média dos valores de tempo com valores maiores

Tamanho (bytes)	**Tempo médio (ms)**		**Diferença absoluta entre o cenário 2 e o 1 (ms)**	**Diferença % de tempo cenário 2 em relação ao 1**
	Cenário 1	**Cenário 2**		
1	5,147	10,763	5,616	109%
2	4,560	9,146	4,586	101%
4	4,651	10,057	5,406	116%
8	4,991	11,710	6,719	135%
16	4,738	11,442	6,705	142%
32	4,279	11,344	7,065	165%
64	6,721	12,715	5,995	89%
128	5,607	10,188	4,581	82%
256	4,843	13,193	8,350	172%
512	6,614	12,228	5,615	85%
1024	7,254	12,489	5,235	72%
2048	93,874	124,113	30,239	32%
4096	33,752	119,805	86,053	255%
8192	202,682	341,452	138,769	68%

Figura 23. Tempo médio de envio de mensagens com valores acima de 1024



Fonte: Autoria própria

Houve mais tentativas na realização de mais testes e tentativas de envio completo das mensagens, contudo sem sucesso.

Após isso, foram levantadas algumas hipóteses do porquê desse comportamento.

Um pensamento inicial seria que esses tempos podem ter sido causados pelo “estouro de pilha”, que ocorre quando um programa tenta alocar mais memória na pilha de execução do que está disponível, que pode ter ocorrido porque os experimentos foram implementados em um loop que ia aumentando o tamanho da mensagem ao final das amostras de cada tamanho.

Depois, uma hipótese levantada seria que esses de valores de tempo muito maiores podem ter surgido devido a quebra de mensagens nos envios realizadas pelo próprio protocolo TCP, para controle de envio, de processamento. Ou seja, uma estratégia do próprio protocolo, que ao encontrar a lista com as demais tarefas no gerenciador, demandou um maior tempo devido atrasos.

Foi percebido que durante a execução de mensagens maiores havia uma quebra em tamanhos maiores que 1044 bytes. Comportamento que conforma a própria natureza da

comunicação em *IoT*, utilizada para pequenos tamanhos de mensagens trocadas a todo momento.

Mesmo com essa alteração no código, os valores permaneceram discrepantes. Uma hipótese discutida foi a possibilidade de essas funções de concatenação estivessem agregando algum valor ao total, então isso impossibilitou esses dados para comparação com o demais.

Procurando informações acerca desse comportamento, foram encontrados fóruns de discussão de desenvolvedores relatando situações semelhantes a essas no Github[7]. Trazendo relatos que convergiam para o comportamento aqui observado, que após 1024 bytes, ocorria uma quebra nos dados, o que corrobora com o funcionamento padrão dos dispositivos em comunicação para aplicações *IoT*.

[7] O GitHub é um repositório de código onde os desenvolvedores compartilharam soluções e problemas nos mais diversos cenários. Geralmente o código é aberto, o que permite realizar projetos. Fonte: Curso em vídeo. Disponível em: < https://www.youtube.com/watch?v=GDGMf2bnHlE&t=10s >.

5 CONCLUSÕES E TRABALHOS FUTUROS

Os testes em questão nesse trabalho tiveram como finalidade observar e quantificar, na forma da metodologia utilizada, a diferença de desempenho quanto a latência entre os dois cenários de comunicação desenvolvidos e nesse aspecto, os testes cumpriram seu papel.

Partindo da informação inicial obtida a partir de bibliografias existentes que indicam que linguagens compiladas possuem um melhor desempenho em detrimento a linguagens interpretadas, a hipótese foi confirmada pelo trabalho experimental presente, no qual o modelo utilizando a linguagem do Arduino apresentou menor latência que o modelo utilizando MicroPython.

Entretanto, diante dos valores e dos estudos realizados, concluiu-se que esse resultado não inviabiliza o uso de uma nem outra opção, como dito em outros momentos desse trabalho, pois esse parâmetro não é o único a se considerar.

É preciso tomar posse dos dados experimentais e olhar além deles, realizar uma análise mais profunda, respondendo perguntas como “essa diferença é relevante para todas as aplicações que usam este microcontrolador?”, “quais melhorias podem ser feitas no cenário onde o MicroPython foi utilizado?”, “quais aplicações podem tolerar essa diferença de comunicação em prol da produtividade em seu desenvolvimento?”, “em quais situações uma linguagem ou outra poderia apresentar maiores vantagens ou é uma escolha particular de cada desenvolvedor?”.

As diferenças, apesar de percentualmente altas, apresentam diferenças absolutas na ordem de milissegundos, o que não são relevantes em aplicações comerciais e residências de um modo geral.

Em sistemas que envolvam segurança em plantas industriais, por exemplo, nas quais seja importante atuar quase que instantaneamente ou que utilizem diagnósticos de falhas com precisão em décimos de segundos, essa diferença vai se tornar relevante. Para esses casos, o tempo é um critério de grande peso na escolha, sendo, portanto, mais indicado o cenário baseado em C/C++.

Já em aplicações como sistemas inteligentes residenciais, diferenças tão pequenas podem não ter impacto degradante e a escolha em utilizar o MicroPython pode ser a melhor opção, a depender de outros fatores como da disponibilidade de desenvolvedores treinados, a velocidade de aprendizado da mesma e de preparação de equipes, levando em conta ser linguagem de baixa complexidade e a relativa facilidade de aprendizado.

Como discutido em demais pontos do texto, o objetivo aqui foi subsidiar o desenvolvedor de situações para que se possa tomar a melhor decisão baseado nos fatores mais relevantes de cada aplicação.

Por isso, como trabalhos futuros, há a intenção de aumentar a diversidade de cenários, utilizando outras combinações, com outras linguagens que estão ganhando espaço, como por exemplo a linguagem Go[8], e outras tradicionais como Java e C# e utilizar também outros microcontroladores como o Raspeberry Pi Pico[9]. Realizando inclusive testes híbridos com o servidor em linguagem do Arduino e o cliente em Python e nas outras linguagens citadas, tendo assim mais combinações para subsidiar escolhas de configurações.

[8] Go é uma linguagem de programação de código aberto suportada pelo Google. Fonte: Site oficial. Diponível em: https://go.dev/

[9] A série Raspberry Pi Pico é uma linha de placas minúsculas, rápidas e versáteis construídas usando RP2040, o principal chip microcontrolador projetado pela Raspberry Pi no Reino Unido. Fonte: Site oficial. Disponível em: https://www.raspberrypi.com/products/raspberry-pi-pico/

REFERÊNCIAS

ABINC. Associação Brasileira de Internet das Coisas. **IOT em 2024: novas tendências apontam para uma revolução conectada**. Disponível em: <https://abinc.org.br/iot-em-2024-novas-tendencias-apontam-para-uma-revolucao- conectada/>. Acesso em: 10 de dez. de 2023.

ALMEIDA, Marcos. **Pandas Python: o que é, para que serve e como instalar.** Alura,2023. Disponível em < https://www.alura.com.br/artigos/pandas-o-que-e-para-que-serve-como-instalar>. Acesso em 26 de outubro de 2023.

ALVES, Maicon Melo. **Sockets Linux**. Brasport, 2008.

ANTUNES, Rodrigo M.. **Sistemas Embarcados e Linguagem C**, 2020.UNIFEI. 1 vídeo (1h e 10min). Publicado pelo canal Rodrigo Maximiano Antunes de Almeida. Disponível em: https://www.youtube.com/watch?v=azv20pt6fCM. Acesso em: 10 mar. 2023.

BORGES, Luiz Eduardo. **Python para desenvolvedores: aborda Python 3.3**. Novatec Editora, 2014.

CERDA, Diogo Busanello**. Implementação de Dead Reckoning para mitigar o efeito da latência em jogos multiplayer.** Monografia (Graduação em Ciências da Computação).Universidade Federal de Santa Maria, 2018.

CIRILO, Carlos Eduardo. **Computação Ubíqua: definição, princípios e tecnologias**. São Carlos, 2008.

COMARELA, Giovanni et al. Introdução à Ciência de Dados: Uma Visão Pragmática utilizando Python, Aplicações e Oportunidades em Redes de Computadores. **Sociedade Brasileira de Computação**, 2019.

COULOURIS, George et al. **Sistemas Distribuídos-: Conceitos e Projeto**. Bookman Editora, 2013.

DE SALES, André Barros. **Medidas de latência em ambientes de processamento de alto desempenho**. Dissertação (Mestrado em Ciência da Computação). Universidade Federal de Santa Catarina, Centro Tecnológico, 2000.

DINARI, Hamed. Inter-Process Communication (IPC) in Distributed Environments: An Investigation and Performance Analysis of Some Middleware Technologies. **International Journal of Modern Education & Computer Science**, v. 12, n. 2, 2020.

DOMBROWSKI, Quinn; GNIADY, Tassie; KLOSTER, David. **Introdução ao Jupyter Notebook.** The Programming Historian em Português, 2023.

FACCIONI FILHO, Mauro. **Internet das coisas**. Unisul Virtual, 2016.

FERREIRA, Israel Vieira; BIGHETI, Jeferson André; GODOY, Eduardo Paciencia. Development of a wireless gateway for industrial internet of things applications. **IEEE Latin America Transactions**, v. 17, n. 10, p. 1637-1644, 2019.

IEEE Spectrum. Disponível em https://spectrum.ieee.org/st/about. Acesso em 10 de jan de 2024.

KENSHIMA, Gedeane. **Nas Linhas do Arduino Plus: Wiring, Hardware e Possibilidades**. Novatec Editora; 1ª edição, 2021.

KRIGER, Daniel. **O que é Python, para que serve e por que aprender?** Kenzie, 2022. Disponível em: < https://kenzie.com.br/blog/o-que-e-python/ >. Acesso em 21 de jan de 2023.

LIMA, Guilherme. **IDE. Integrated Development Environment**. Alura, 2022. Disponível em < https://www.alura.com.br/artigos/o-que-e-uma-ide>. Acesso em 24 de ago de 2023.

MAGRANI, Eduardo. **A internet das coisas**. BOD GmbH DE, 2021.

MAIER, Alexander et al. **Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things**. Glyndwr University, UK, 2017.

MANZIEIRO, Carlos. **Notas de aula**, 2008. Comunicação em rede. Disponível em < https://wiki.inf.ufpr.br/maziero/doku.php?id=pua:comunicacao_em_rede >. Acesso em 11 de janeiro de 2024.

MICROPYTHON. Página Inicial. Disponível em < https://micropython.org/>. Acesso em 11 de agosto de 2023.

MCKINNEY, Wes. **Python para análise de dados: Tratamento de dados com Pandas, NumPy e IPython.** Novatec Editora, 2018.

MIRANDA, João Vitor de. **Jupyter Notebook: Exemplos de Códigos e Como Usar** Alura,2023. Disponível em < https://www.alura.com.br/artigos/conhecendo-o-jupyter-notebook>. Acesso em 30 de outubro de 2023.

NETO, José Valladares et al. **Boxplot: um recurso gráfico para a análise e interpretação de dados quantitativos.** Revista Odontológica do Brasil Central, v. 26, n. 76, 2017.

NUMPY.org. Disponível em <numpy.org>. Acesso em 11 de agosto de 2023.

OLIVEIRA, Marcelo Eduardo de et al. **Introdução à robótica educacional com Arduino**– hands on! Iniciante. Pirassununga. SP. 2020.

PANDAS.org. Disponível em < https://pandas.pydata.org/>. Acesso em 11 de agosto de 2023

PATRICIO, Thiago Seti et al. **Internet Das Coisas (Iot): As Consequências Da Computação Ubíqua Na Sociedade.** In: Colloquium Humanarum. ISSN: 1809-8207. 2018. p. 83-93.

PYTHON. Página Inicial. Disponível em < https://www.python.org/>. Acesso em 11 de agosto de 2023.

PLAUSKA, Ignas; LIUTKEVIČIUS, Agnius; JANAVIČIŪTĖ, Audronė. **Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller.** Electronics, v. 12, n. 1, p. 143, 2022.

RYSAK, Paweł. **Comparative analysis of code execution time by C and Python based on selected algorithms.** Journal of Computer Sciences Institute, v. 26, p. 93-99, 2023.

SANTOS, Bruno P. et al. Internet das coisas: da teoria à prática. **Minicursos SBRC-Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuıdos**, v. 31, p. 16, 2016.

SANTOS, Sandro. **Introdução à IoT: desvendando a internet das coisas**. SS Trader Editor, 2018.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação-11**. Bookman Editora, 2018.

SÔNEGO, Arildo Antônio; MARCELINO, Roderval; GRUBER, Vilson. **A Internet das Coisas aplicada ao conceito de eficiência energética: uma análise quantitativo-qualitativa do estado da arte da literatura.** Atoz: novas práticas em informação e conhecimento, v. 5, n. 2, p. 80-90, 2016.

TANENBAUM, Andrew S e STEEN, Maarten Van. **Sistemas distribuídos: princípios e paradigmas**. 4ª edição. São Paulo, 2023.

TIESSI, Ana; FREITAS, Nicksson. **Técnicas para processamento de big data**. Editora Senac São Paulo, 2023.

TIOBE. Disponível em < https://www.tiobe.com/tiobe-index/ >. Acesso em 02 de fev de 2023.

THONNY. Página Inicial. Disponível em < https://thonny.org/>. Acesso em 11 de agosto de 2023.

VALADARES, F. G. Detecção de outliers multivariados em redes de sensores. 2012. 53 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Ouro Preto, Ouro Preto, 2012.

VENKATARAMAN, Aditya; JAGADEESHA, Kishore Kumar. **Evaluation of inter-process communication mechanisms.** Architecture, v. 86, p. 64, 2015.

WIRING. Página Inicial. Disponível em < https://wiring.org.co/ >. Acesso em 01 de agosto de 2023.