UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

MARIANNA DE PINHO SEVERO

IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO DE UM SISTEMA *SINGLE BOARD* PARA MONITORAMENTO AÉREO

QUIXADÁ

2019

MARIANNA DE PINHO SEVERO

IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO DE UM SISTEMA *SINGLE BOARD* PARA MONITORAMENTO AÉREO

Monografia apresentada ao Curso de Bacharelado em Engenharia de Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação. Área de concentração: Computação.

Orientador: Prof. Dr. Arthur de Castro Callado.

QUIXADÁ

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S525i Severo, Marianna de Pinho.
Implementação e análise de desempenho de um sistema single board para monitoramento aéreo / Marianna de Pinho Severo. – 2019.
86 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Computação, Quixadá, 2019.
Orientação: Prof. Dr. Arthur de Castro Callado.

1. Sistemas de controle de tráfego aéreo. 2. Sistemas embarcados. 3. Desempenho-Avaliação. I. Título.
CDD 621.39

MARIANNA DE PINHO SEVERO

IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO DE UM SISTEMA *SINGLE BOARD* PARA MONITORAMENTO AÉREO

Monografia apresentada ao Curso de Bacharelado em Engenharia de Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de Bacharel em Engenharia de Computação. Área de concentração: Computação.

Aprovada em: ___/___/_____.

BANCA EXAMINADORA

_____

Prof. Dr. Arthur de Castro Callado (Orientador)

Universidade Federal do Ceará (UFC)

_____

Prof. Me. Michel Sales Bonfim

Universidade Federal do Ceará (UFC)

_____

Prof. Me. Jeandro de Mesquita Bezerra

Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Sidartha Azevedo Lôbo de Carvalho

Universidade Federal do Ceará (UFC)

A todos que me ajudaram ao longo dessa jornada chamada Vida.

AGRADECIMENTOS

Agradeço a Deus pela minha vida, família, amigos e por todas as experiências, aprendizados, erros e acertos que já vivi e viverei.

Agradeço à minha família por sempre estar ao meu lado e me apoiar, nos momentos felizes, tristes, de vitória e de dificuldades.

Agradeço a todos os meus amigos, por todas as vezes que viramos noites fazendo trabalho juntos, pelos momentos em que nos reunimos fosse para conversar, fazer almoços, tocar músicas ou fazer trilhas para o Cedro. Seria muito mais difícil chegar até esse momento sem eles.

Agradeço a todos os meus professores, tanto os da época da escola como os da universidade, por compartilharem seus conhecimentos, por todos os dias contribuírem para a transformação de nossas vidas, por nos ajudarem na busca de nossos sonhos e por nos mostrarem que, com esforço, podemos vencer qualquer desafio.

Agradeço ao meu orientador, o professor Arthur Callado, e à professora Paulyne Jucá, por todas as conversas, ensinamentos e conselhos, que contribuíram tanto para o meu amadurecimento acadêmico, como profissional e pessoal.

“Love is still the answer.”

(Jason Mraz)

RESUMO

O tráfego aéreo mundial tem crescido cada vez mais e a tecnologia ADS-B tem se mostrado como uma importante ferramenta para garantir o funcionamento desse setor. Entretanto, ainda são empregados, em sua maioria, dispositivos proprietários e de elevado custo. Esse cenário impõe limitações à adoção da tecnologia ADS-B por diferentes tipos de usuários, como aeroportos de pequeno e médio porte, grupos de pesquisadores e outros cidadãos interessados em atividades de monitoramento aéreo. Assim, com o objetivo de fornecer uma solução mais acessível a diversos tipos de usuários, este trabalho realizou a implementação de um sistema de monitoramento aéreo baseado em tecnologia ADS-B, de código aberto e aplicado a uma plataforma de *hardware single board*, a qual possui custos mais baixos, embora apresente maiores limitações de recursos computacionais. Também realizou-se uma análise de desempenho do sistema desenvolvido, comparando seus resultados com os de outro *software* de monitoramento aéreo e com os de uma plataforma de *hardware* de propósito geral. Foi observado que o sistema implementado apresentou resultados tão bons na plataforma *single board* como na de propósito geral.

Palavras-chave: Sistemas de controle de tráfego aéreo. Sistemas embarcados. Desempenho - Avaliação.

ABSTRACT

Worldwide air traffic is growing and ADS-B technology has proven to be an important tool to ensure the operation of this sector. However, most devices are still proprietary and very expensive. This scenario imposes limitations on the adoption of ADS-B technology by different types of users, such as small and medium sized airports, research groups, and other citizens interested in aerial monitoring activities. Thus, in order to provide a more accessible solution to different types of users, this work implemented an open source ADS-B technology-based aerial monitoring system applied to a single board hardware platform, which has lower costs, although it has higher limitations of computational resources. Also, a performance analysis of the developed system was performed, comparing its results with those of other aerial monitoring software and those of a general purpose hardware platform. It was observed that the implemented system presented as good results in the single board platform as in the general purpose.

Keywords: Air Traffic Control Systems. Embedded systems. Performance - Evaluation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACARS	*Aircraft Communications Addressing and Reporting System*
ACC	*Area Control Center*
ADC	*Analog-to-Digital Converter*
ADS-B	*Automatic Dependent Surveillance - Broadcast*
AMD	*Advanced Micro Devices*
APP	*Approach*
ARM	*Advanced RISC Machine*
ATC	*Air Traffic Control*
CA	*Capacity*
CAN	*Controller Area Network*
CB	*Controller Board*
CPU	*Central Processing Unit*
CR	*Carriage Return*
CSI	*Camera Serial Interface*
DF	*Downlink Format*
DSI	*Display Serial Interface*
eMMC	*Embedded Multimedia Card*
GNSS	*Global Navigation Satellite System*
GPIO	*General Purpose Input/Output*
GPS	*Global Positioning System*
GPU	*Graphics Processing Unit*
HD	*Hard Disk*
HDMI	*High-Definition Multimedia Interface*

HTTP *Hypertext Transfer Protocol*

ICAO *International Civil Aviation Organization*

IR *Infrared*

I2C *Inter-Integrated Circuit*

I2S *Inter-IC Sound*

LAN *Local Area Network*

LF *Line Feed*

ORM *Object-Relational Mapping*

OTG *On The Go*

PC *Personal Computer*

PI *Parity*

PSR *Primary Surveillance Radar*

RAM *Random Access Memory*

ROM *Read Only Memory*

RTC *Real Time Clock*

SATA *Serial Advanced Technology Attachment*

SBC *Single Board Computer*

SD *Secure Digital*

SOC *System On Chip*

SPI *Serial Peripheral Interface*

SSR *Secondary Surveillance Radar*

TB *Tera Byte*

TC *Type Code*

TWR *Tower*

UAT *Universal Access Transceiver*

UART *Universal Asynchronous Receiver/Transmitter*

USB *Universal Serial Bus*

SUMÁRIO

1 INTRODUÇÃO

O número de aeronaves ocupando o espaço aéreo simultaneamente tem crescido bastante nos últimos anos, principalmente em decorrência da maior procura da população pelos serviços fornecidos pelas companhias aéreas comerciais. Tendo em vista esse crescimento e as limitações dos tradicionais sistemas de radares empregados para o monitoramento e controle do espaço aéreo, novas tecnologias começaram a ser desenvolvidas tanto para melhorar como para garantir o funcionamento dos sistemas de transporte aéreo (PARK; TOMLIN, 2014).

Nesse cenário, uma nova tecnologia que se destaca e vem sendo gradualmente adotada pelos países ao redor do globo é a ADS-B (*Automatic Dependent Surveillance - Broadcast*). Isso se deve a sua capacidade de fornecer maior acurácia na obtenção do posicionamento das aeronaves; a uma taxa mais rápida de atualização dos dados; a um menor custo de aquisição, instalação e manutenção de equipamentos, se comparada aos radares tradicionais; e a uma melhor operação, mesmo em regiões de baixa altitude (SU et al., 2018).

Uma das grandes diferenças entre a tecnologia ADS-B e os sistemas de radares é que, para que estes últimos consigam informações mais detalhadas sobre as aeronaves, eles têm que enviar requisições para elas. Ao passo que aviões equipados com a tecnologia ADS-B podem compartilhar, periodicamente, mensagens contendo suas informações e qualquer dispositivo capaz de capturar e decodificar os sinais na banda de frequência em que essas mensagens são enviadas pode obtê-las (PARK; TOMLIN, 2014).

Apesar de suas vantagens, tanto a tecnologia ADS-B como os sistemas de radares ainda vêm sendo implementados em computadores de alto custo utilizando *softwares* proprietários. Esse cenário é viável para médios e grandes aeroportos, mas torna-se de difícil acesso para os aeroportos de pequeno porte, como os presentes em cidades pequenas e em propriedades privadas, devido às limitações de recursos para sua implementação e manutenção (FEITOSA et al., 2015). Ademais, isso dificulta os trabalhos de diferentes comunidades de pesquisadores que buscam melhorar a tecnologia ADS-B (SCHÄFER et al., 2014).

Com o avanço tecnológico que vem sendo alcançado, tanto nas áreas de eletrônica e computação como na de comunicações, tem-se conseguido produzir dispositivos cada vez menores, mais baratos e de maior poder computacional. Um exemplo disso é a plataforma Raspberry Pi[1], assim como suas semelhantes, que consiste em um pequeno computador

[1] https://www.raspberrypi.org/

programável, barato, customizável e com suporte a um grande número de periféricos e à comunicação em rede (VUJOVIĆ; MAKSIMOVIĆ, 2015), podendo ser usada para diversos projetos de sistemas embarcados ou de propósito geral.

A tecnologia ADS-B é de grande importância para a melhoria e manutenção do funcionamento do espaço aéreo, mas existem algumas limitações para a sua adoção por todos os níveis de aeroportos e por pesquisadores da área. Assim, novas plataformas de grande potencial e menores custos vêm ganhando cada vez mais espaço. Baseado nisso, este trabalho tem como foco a implementação de um *software* de código aberto para o monitoramento de aeronaves equipadas com a tecnologia ADS-B e a análise de desempenho de sua aplicação em uma plataforma *single board*, semelhante à Raspberry Pi. Para isso, experimentos foram realizados com o objetivo de mostrar a eficiência de um sistema de mais baixo custo - tanto em relação aos sistemas de radares como com respeito aos dispositivos proprietários empregados atualmente, baseados na tecnologia ADS-B - para monitoramento aéreo. Os resultados mostraram que o sistema na plataforma *single board* pode apresentar resultados tão bons como os obtidos em plataformas de maior poder computacional, como o computador pessoal empregado neste trabalho.

Assim, espera-se contribuir para a melhoria do monitoramento do espaço aéreo, tornando mais fácil a utilização da tecnologia ADS-B por pessoas ou instituições de menor poder aquisitivo - como pequenos e médios aeroportos e outros cidadãos interessados - e contribuindo para a comunidade de desenvolvedores e pesquisadores através da implementação de um *software* de código aberto.

Alguns trabalhos vêm sendo desenvolvidos nesse mesmo contexto. Dentre eles, a implementação e avaliação de um coletor de mensagens ADS-B em dois tipos de plataforma, uma embarcada e outra de propósito geral (FEITOSA et al., 2015); o desenvolvimento de um nano satélite para captura de mensagens ADS-B (PAHLEVY et al., 2018); a análise de desempenho de receptores e decodificadores de mensagens ADS-B (ABDULAZIZ et al., 2015); além de trabalhos que fornecem um *software* de monitoramento, uma plataforma embarcada para execução desse *software* e/ou uma plataforma de visualização das informações capturadas, como em (SCHÄFER et al., 2014).

Este trabalho tem como base trabalhos realizados no projeto “Atualização da Monitoração Aeronáutica e Auto-Sustentabilidade”, desenvolvido por professores e alunos da Universidade Federal do Ceará (UFC) no Campus de Quixadá.

1.1 Objetivos

Dessa forma, o objetivo geral deste trabalho é a implementação e análise de desempenho de um sistema *single board* de baixo custo empregado para monitoramento aéreo, baseando-se na tecnologia ADS-B. Como objetivos específicos estão:

a) Desenvolver um *software* em linguagem de programação C para a decodificação de mensagens ADS-B enviadas por aeronaves, de forma a obter informações como identificação, velocidade e posicionamento dessas aeronaves;
b) Realizar a análise de desempenho, tanto deste *software* como de outro - implementado em linguagem de programação Python[2], em etapas passadas do projeto "Atualização da Monitoração Aeronáutica e Auto-Sustentabilidade" -, em duas plataformas de *hardware* diferentes, uma delas de um computador pessoal e a outra, *single board*, de maneira a observar as diferenças de desempenho dos dois tipos de *hardware*;
c) Propor melhorias para o desenvolvimento do sistema de monitoramento aéreo implementado, baseando-se nas análises de desempenho realizadas.

O restante deste trabalho está organizado da seguinte maneira: no capítulo 2 são apresentados os principais conceitos para o desenvolvimento deste trabalho, que são a tecnologia ADS-B, as plataformas *single board*, a análise de desempenho e algumas das técnicas da Estatística Inferencial; no capítulo 3 realiza-se a apresentação e discussão dos trabalhos relacionados, com suas semelhanças e diferenças do trabalho aqui proposto; no capítulo 4, a metodologia empregada para o desenvolvimento deste trabalho é explicada; no capítulo 5 são apresentados os resultados e discussões da análise de desempenho feita sobre o sistema; e, por fim, no capítulo 6, são realizadas as considerações finais.

[2] https://github.com/RadarLivre/RadarLivreCollector

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, serão apresentados alguns dos conceitos principais necessários para o entendimento e desenvolvimento do projeto proposto neste trabalho.

2.1 Sistemas de controle e monitoramento aéreo

O monitoramento do espaço aéreo é realizado por diferentes sistemas de controle, de acordo com a etapa do voo em que a aeronave se encontra. Esses sistemas são citados a seguir. O primeiro são as Torres de Controle Aéreo (TWR, do inglês *Tower*), as quais são responsáveis pelo monitoramento das aeronaves dentro do escopo dos aeroportos, durante as fases de pouso e decolagem. Existem também os Controles de Aproximação (APP, do inglês *Approach*), que ficam encarregados do controle do tráfego aéreo em regiões com vários aeroportos, possuindo um alcance maior que as torres de controle. Há ainda os Controles de Área (ACC, do inglês *Area Control Center*), os quais são responsáveis por regiões fora do alcance dos outros dois tipos de sistemas de monitoramento.

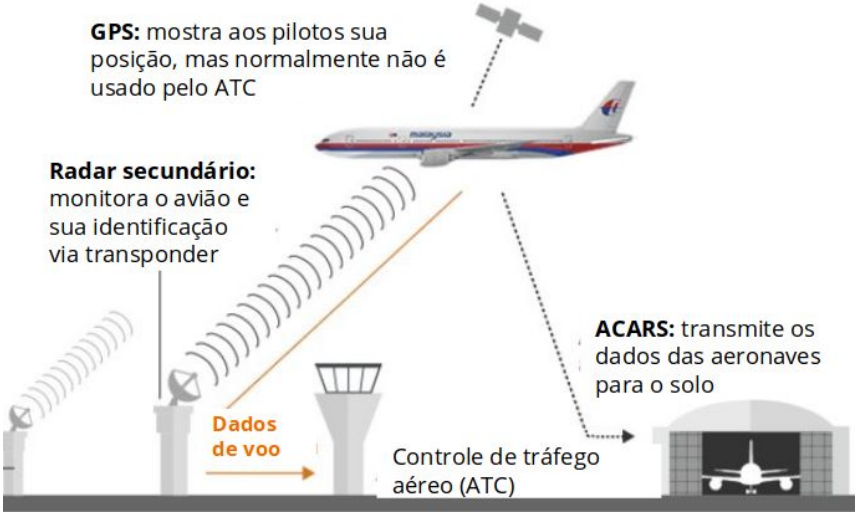
2.1.1 Sistema de radares

Para tornar as atividades de controle e monitoramento aéreo possíveis, uma das principais tecnologias que têm sido adotada ao longo dos anos é a de radares. Dois tipos principais de radares são empregados: os Radares de Monitoramento Primários (PSR, do inglês *Primary Surveillance Radar*), os quais obtêm informações sobre a distância de uma aeronave baseando-se nos princípios de reflexão de ondas eletromagnéticas, não sendo necessária a comunicação com os equipamentos presentes nela; e os Radares de Monitoramento Secundários (SSR, do inglês *Secondary Surveillance Radar*), os quais se comunicam com os equipamentos de comunicação presentes nas aeronaves, enviando requisições e capturando as respostas, as quais contêm informações de identificação e altitude (ABDULAZIZ et al., 2015). Um exemplo de cenário em que os radares primários e secundários são empregados pode ser observado na Figura 1.

Tendo em vista as limitações dos sistemas de radares - como os elevados custos de equipamentos, instalação, operação e manutenção, a impossibilidade de fornecer cobertura em algumas regiões, uma baixa resolução para as informações de posicionamento, diminuição de desempenho devido à presença de ruídos e uma menor velocidade de atualização dos dados, no caso dos radares secundários -, organizações de todo o mundo se empenharam para o

desenvolvimento de uma nova tecnologia de monitoramento aéreo. Assim, surgiu a tecnologia ADS-B (ABDULAZIZ et al., 2015)(SCHÄFER et al., 2014).

Figura 1 - Exemplo de cenário empregando os sistemas de radares.



Fonte: Adaptado de BBC (2014).

2.1.2 Tecnologia ADS-B

A tecnologia ADS-B está sendo desenvolvida de maneira a fornecer uma maior capacidade, eficiência e segurança no controle do tráfego aéreo (PARK; TOMLIN, 2014). Dentre as suas vantagens, que a faz ser adotada por cada vez mais países, estão: maior acurácia da determinação do posicionamento das aeronaves, maior velocidade de atualização dos dados, menores custos se comparada aos radares e melhor desempenho no monitoramento de baixas altitudes (SU et al., 2018).

Essa tecnologia é dita dependente porque, para determinar a posição da aeronave que a abriga, ela utiliza as informações obtidas pelos sistemas de GNSS (*Global Navigation Satellite System*) presentes na aeronave (ABDULAZIZ et al., 2015). Além disso, ela é chamada de automática porque não precisa que nenhuma ação seja tomada pelos pilotos nem que nenhuma requisição seja enviada pelos sistemas de controle terrestres para que ela envie suas informações. Dessa forma, ela envia suas informações automaticamente, algumas delas de maneira periódica e outras, de acordo com eventos (SCHÄFER et al., 2014). Por fim, um importante detalhe sobre essa tecnologia de monitoramento aéreo é que, como ela compartilha suas informações sem a necessidade de requisições, qualquer dispositivo capaz de captar esses sinais pode recebê-los (PARK; TOMLIN, 2014), já que não há criptografia. Na Figura 2, pode-se observar um cenário em que a tecnologia ADS-B está sendo aplicada.

Figura 2 - Exemplo de cenário da tecnologia ADS-B.

Fonte: Adaptado de Shah (2017).

Os equipamentos ADS-B podem ser divididos em sistemas de transmissão, chamados de ADS-B *Out*, e de recepção, quando são chamados de ADS-B *In*. Nas aeronaves, eles podem estar presentes em dois principais tipos de dispositivos: um deles, chamado de *Universal Access Transceiver* (UAT), é especialmente adaptado para a tecnologia ADS-B e precisa ser instalado nas aeronaves, tornando-se caro e às vezes inviável para aplicação em aviões comerciais, apesar de ser utilizado em algumas aeronaves civis não comerciais; o outro tipo de dispositivo são os *transponders* que operam no Modo S, os quais já estão presentes na maioria das aeronaves e podem começar a fornecer serviços ADS-B após algumas pequenas atualizações (SCHÄFER et al., 2014).

Transponders são equipamentos utilizados pelas aeronaves para os processos de comunicação. No caso do modo de operação S, duas frequências de ondas de rádio são empregadas: a de 1030 MHz, para o recebimento de requisições e outros tipos de sinais; e a de 1090 MHz para a transmissão de informações e respostas a requisições. O ICAO (*International Civil Aviation Organization*) determina dois formatos para as mensagens enviadas pelos transponders no modo S: o formato curto, de 56 bits, e o formato longo, de 112 bits. O formato longo pode ser usado para mandar mensagens cujo campo de dados pode conter qualquer tipo de informação especificado em outro de seus campos, chamado de *type code*. Quando isso ocorre, esse formato é chamado de *Extended Squitter* e as mensagens são enviadas sem a necessidade de interrogações (SCHÄFER et al., 2014). Dessa forma, a

tecnologia ADS-B pode utilizar equipamentos que operam no modo S, compartilhando mensagens do tipo *Extended Squitter* e utilizando a radiofrequência de 1090 MHz, dando origem ao termo *link* de dados 1090 ES (JUNZI, 2018).

2.1.2.1 Mensagens ADS-B

As aeronaves enviam suas informações através de mensagens do tipo ADS-B, as quais são compostas por cinco campos principais, conforme pode ser visto na Tabela 1.

Tabela 1 - Campos que compõem uma mensagem ADS-B.

Formato de Downlink (DF)	Capacidade (CA)	Endereço da Aeronave (ICAO)	Dados	Paridade (PI)

Fonte: Adaptado de Junzi (2018).

O campo Formato de *Downlink* indica se a mensagem recebida é do tipo ADS-B ou não. No caso das aeronaves que compartilham mensagens ADS-B utilizando *transponders* modo S, através de *broadcast*, DF = 17. Já a Capacidade é utilizada para fornecer informações mais detalhadas sobre os *transponders* presentes nas aeronaves, indicando o tipo de informações que eles podem trocar e os formatos de mensagem que eles suportam. O ICAO corresponde a um endereço único que é atribuído ao *transponder* presente na aeronave pela instituição ICAO. "Dados" representa o campo de dados, o qual contém informações sobre o avião que enviou a mensagem, como por exemplo, a identificação do voo (*callsign*), as velocidades vertical e horizontal, a direção de deslocamento (*heading*), a altitude, a latitude e a longitude. Por fim, o campo Paridade fornece um conjunto de bits utilizados para a detecção de erros na mensagem recebida (JUNZI, 2018).

O projeto apresentado neste trabalho utilizou um receptor de mensagens ADS-B, que consegue capturar sinais enviados na frequência de 1090 MHz. Uma vez recebida uma mensagem, seus campos são decodificados, conforme a estrutura mostrada na Tabela 1, para a extração das informações descritas no parágrafo anterior.

2.2 Computadores *single board* (SBC)

As informações contidas nesta seção são provenientes dos trabalhos apresentados em Johnston et al. (2018) e Qureshi; Koubâa (2019).

Um novo tipo de plataforma que tem alcançado um destaque cada vez maior são os computadores *single board*. Isso se deve, dentre outros fatores, ao seu baixo custo, ao pequeno consumo de energia e ao considerável poder computacional. Dentre os exemplos desse tipo de tecnologia, estão as plataformas *Raspberry Pi* e *Beaglebone Black*.

Um computador *single board* pode ser definido como um computador completo construído em uma única placa. Aliada a essa definição, é importante destacar três características principais que os diferem dos computadores pessoais: a disponibilidade de portas de entrada e saída de propósito geral, o baixo consumo de energia e o baixo custo. Na Tabela 2, pode-se observar uma comparação entre diferentes tipos de plataformas computacionais e as *single board*.

Tabela 2 - Comparação entre Computador Pessoal (PC), Placa Controladora (CB), *Smartphone* e Computador *Single Board* (SBC).

Componente	**PC**	**CB**	***Smartphone***	***Single Board***
CPU	Sim, placa-filha	Sim	Sim	Sim
GPU	Sim, placa-filha	Sim	Sim	Sim
Memória	Sim, placa-filha	Sim	Sim	Sim
LAN	Sim, placa-filha	Indisponível	Indisponível	Sim
Saída de vídeo	Sim, placa-filha	Indisponível	Indisponível	Sim
Armazenamento	ROM, armazenamento externo de leitura e escrita	ROM	ROM, armazenamento externo de leitura e escrita	ROM, armazenamento externo de leitura e escrita
GPIO	Sim, (USB)	Sim	Indisponível	Sim

Fonte: Adaptado de Johnston et al. (2018).

Outra importante diferença entre as plataformas SBC e os computadores pessoais é que, em sua maioria, elas utilizam processadores ARM, ao passo que no mercado de computadores pessoais se destacam outros tipos de processadores, como Intel e AMD.

Ao longo dos anos, uma variedade de plataformas *single board* têm sido desenvolvidas, cada uma com suas vantagens e desvantagens. Algumas das principais vantagens são o baixo custo, o pequeno tamanho, o baixo consumo de energia e a portabilidade. Dentre as desvantagens estão a limitação de recursos computacionais, como processamento, memória e armazenamento. Na Tabela 3, é possível observar alguns exemplos de computadores *single board* desenvolvidos atualmente e algumas de suas principais características.

Tabela 3 - Exemplos de *single boards*, com preços de abril de 2018 em dólar.

Placa	**SoC**	**RAM**	**Preço**	**Entrada/Saída**

Raspberry Pi 1 B +	BCM2835	512 MB	$30	Áudio, vídeo composto, CSI, DSI, Ethernet, GPIO, HDMI, I2C, I2S, MicroSD, SPI, USB2
Raspberry Pi 2 B	BCM2836	1 GB	$40	Áudio, vídeo composto, CSI, DSI, Ethernet, GPIO, HDMI, I2C, I2S, MicroSD, SPI, USB2
Raspberry Pi 3 B	BCM2837	1 GB	$35	Áudio, Bluetooth, vídeo composto, CSI, DSI, Ethernet, GPIO, HDMI, I2C, I2S, MicroSD, SPI, USB2, WiFi
Raspberry Pi 3 B +	BCM2837B0	1 GB	$35	Áudio, Bluetooth, vídeo composto, CSI, DSI, Gigabit Ethernet, HDMI, I2C, I2C, MicroSD, PoE Header, SPI, USB2, WiFi
Raspberry Pi Zero W	BCM2835	512 MB	$10	Bluetooth, vídeo composto, CSI, GPIO, HDMI, I2C, I2S, MicroSD, SPI, USB2, WiFi
Odroid C2	S905	2 GB	$46	ADC, eMMC/MicroSD, I2S, Gigabit Ethernet, GPIO, HDMI, IR, UART, USB
Odroid XU4	5422	2 GB	$59	ADC, eMMC/MicroSD, I2C, Gigabit Ethernet, GPIO,HDMI, I2S, SPI, UART, USB, USB3.0
Pine A64	R18	≤ 2 GB	$32	CSI, DSI, Ethernet, Euler, EXP, GPIO, MicroSD, RTC, TP, USB
Orange Pi Plus 2	H7	2 GB	$49	Áudio, CSI, eMMC, Gigabit Ethernet, GPIO, HDMI, I2C, IR, SATA 2.0, SPI, TF, USB, WiFi
BeagleBone Black	AM335	512 MB	$55	ADC, CANbus, Ethernet, GPIO, HDMI, I2C, SPI, eMMC/MicroSD, UART
UP Squared	N4200 e MAX 10	≤ 8 GB	$289	ADC, Gigabit Ethernet (x2), GPIO, HDMI, MIPI (x2), mini-PCIe/m-SATA, RTC, UART, USB2, USB3
Xilinx Z-turn	Zynq-7010	1 GB	$119	CANbus, Gigabit Ethernet, HDMI, TF Card, USB2-OTG, USB_UART

Fonte: Adaptado de Johnston et al. (2018).

Apesar de suas características, as *single boards* são suficientemente poderosas para executarem uma variedade de sistemas operacionais, sendo o mais popular o Linux, e para lidarem com cargas de trabalho convencionais.

É esperado que novas e melhores *single boards* sejam lançadas, conforme novas organizações entram no mercado e novas personalizações dos dispositivos se tornem necessárias para aplicações específicas. Atualmente, uma das maiores aplicações desse tipo de tecnologia é na criação de *clusters* para a construção de *datacenters* descentralizados, que vêm sendo utilizados desde em projetos e aulas em universidades, até no processamento de redes neurais.

Tendo em vista o potencial dos computadores *single board*, este trabalho utilizou um dos tipos desses dispositivos, especificamente a plataforma Orange Pi Pc Plus[3], para a execução do *software* de decodificação de mensagens ADS-B, empregando um sistema operacional Linux. Além disso, realizou-se uma análise do desempenho dessa plataforma durante a execução das atividades de monitoramento. Na Tabela 4, pode-se observar as principais configurações da *single board* a ser usada neste projeto.

Tabela 4 - Especificações de *hardware* da Orange Pi Pc Plus.

Placa	**SoC**	**RAM**	**Preço**	**Entradas/Saídas**
Orange Pi PC Plus	AllWinner H3	1 GB	R$ 100,00	Áudio, Vídeo, CSI, Ethernet, WIFI, GPIO, HDMI, USB2.0 e OTG, MicroSD, eMMC

Fonte: Elaborada pela autora (2019).

2.3 Análise de desempenho

As informações fornecidas nesta seção são provenientes do livro de Jain (1990). A análise de desempenho consiste em um conjunto de atividades empregadas para avaliar um sistema, de maneira que os resultados dessa avaliação contribuam para os processos de tomada de decisão envolvidos. Existem três principais métodos adotados para se realizar esse tipo de análise: a modelagem analítica, a simulação e a medição do sistema real.

Em uma atividade de análise de desempenho, diversos conceitos importantes precisam ser levados em consideração, alguns deles são: as métricas, que são os critérios utilizados para se determinar o desempenho de um sistema; a carga de trabalho, que consiste no volume de requisições ou atividades que o sistema precisa atender; os parâmetros, que são as características, tanto do sistema como da carga de trabalho, que afetam o desempenho do

[3] http://www.orangepi.org/orangepipcplus/

sistema; os fatores, que são parâmetros cujos valores podem variar; os níveis, que são os valores que os fatores podem assumir; os serviços que o sistema fornece, os quais podem contribuir para a escolha adequada dos conceitos já descritos; e os resultados desses serviços, que podem indicar sucesso, erro ou indisponibilidade dos mesmos.

Cada contexto que gera a necessidade de uma avaliação de desempenho é único e a avaliação está sujeita a erros, geralmente devido à não aplicação e escolha correta das informações descritas no parágrafo anterior, à análise ineficiente dos resultados, e à não descrição clara de todas as suposições e limitações assumidas. Entretanto, existe um conjunto de etapas comuns que podem ser empregadas em todas as análises de desempenho que ajudam a evitar os erros geralmente cometidos. Esse conjunto de etapas compõem o que se chama de Abordagem Sistemática (JAIN, 1990), e uma visão geral sobre cada uma dessas etapas é fornecida a seguir:

1. Definir os objetivos da análise e da constituição do sistema a ser analisado. Os objetivos contribuem para a forma como o sistema será definido e esta, por sua vez, influencia a escolha das métricas e cargas de trabalho;
2. Construir uma lista de serviços que o sistema oferece e as possíveis saídas desses serviços, geralmente classificadas em sucesso, erro e indisponibilidade. Essas definições também auxiliam na escolha das métricas e carga de trabalho;
3. Escolher as métricas a serem empregadas para a avaliação do sistema, as quais frequentemente tratam de questões como acurácia, velocidade e disponibilidade de serviços;
4. Criar uma lista de parâmetros que influenciam o sistema, a qual contribui para a discussão sobre o impacto de cada parâmetro sobre o desempenho e para a definição dos dados que precisam ser coletados;
5. Definir, dentre os parâmetros listados, aqueles que são considerados fatores e seus respectivos níveis;
6. Escolher a técnica de análise que melhor se encaixa aos objetivos propostos e aos recursos disponíveis, as quais podem ser: modelagem, simulação ou medição;
7. Selecionar a carga de trabalho, que deve ser o mais semelhante possível à observada na operação real do sistema;

8. Planejar os experimentos, que devem ser executados de forma a se obter o máximo de informações com o mínimo de esforço para isso;
9. Analisar e interpretar os dados gerados, os quais devem levar em consideração, dentre outros fatores, a possível variabilidade dos resultados e o entendimento de que a análise gera resultados e, não, conclusões;
10. Apresentar os resultados de uma maneira que facilite o entendimento do que está sendo mostrado.

É importante destacar que, durante o processo de análise de desempenho, as etapas descritas acima podem precisar serem executadas diversas vezes, constituindo ciclos de análise.

Além disso, mais uma informação importante que precisa ser levada em consideração em uma análise de desempenho é a diferença entre amostra e população. Uma amostra consiste em um subconjunto de uma população e as afirmações feitas sobre uma amostra não são suficientes para que se possa fazer afirmações definitivas sobre a população a qual ela pertence. De mesma forma, não se pode tomar como base os resultados de experimentos em um determinado sistema, os quais foram realizados sob certas condições, que possuíam determinadas características e que empregaram determinadas métricas, para se afirmar que todos os sistemas semelhantes ao avaliado devem apresentar exatamente os mesmos resultados.

Apesar disso, pode-se utilizar os resultados obtidos ao se avaliar as amostras para a construção de definições probabilísticas sobre a população. Dessa forma, é possível se calcular a probabilidade de um sistema apresentar determinado desempenho, dado que uma análise de desempenho sobre um sistema semelhante apresentou as probabilidades de os resultados pertencerem a um determinado intervalo de valores, chamado de intervalo de confiança.

Dessa forma, este trabalho empregou os conhecimentos descritos nos parágrafos acima para a realização de uma análise de desempenho de um sistema *single board* aplicado ao monitoramento aéreo.

2.4 Técnicas estatísticas e métodos não paramétricos

A Estatística é uma área de conhecimento que visa auxiliar no entendimento e na tomada de decisões, principalmente quando existe variabilidade, ou seja, quando várias observações de um sistema ou fenômeno geram resultados possivelmente diferentes (MONTGOMERY;

RUNGER, 2016). Nela, um dos campos que mais se destaca é a Estatística Inferencial, que permite a inferência de informações sobre uma população, a partir de observações feitas sobre amostras aleatórias (subconjuntos dessa população). Geralmente, essas informações consistem em um ou mais parâmetros estatísticos, como a média, a variância, o desvio padrão e a proporção (CAÇÃO, 2010). Duas importantes ferramentas podem ser utilizadas para esse fim: os testes de hipótese e os intervalos de confiança.

Os testes de hipótese permitem verificar uma determinada suposição sobre uma população, baseando-se nas evidências obtidas a partir das amostras dessa população (MONTGOMERY; RUNGER, 2016). Cada teste é construído levando-se em consideração duas hipóteses: uma nula, a qual se deseja rejeitar; e uma alternativa, a qual pretende-se testar e é sempre oposta à hipótese nula (GUIMARÃES, 2018). Cada teste também está sujeito a dois tipos de erro: o erro do tipo I, quando rejeita-se a hipótese nula e ela é verdadeira; e o erro do tipo II, quando não se rejeita a hipótese nula e ela é falsa. À probabilidade de se cometer um erro do tipo I, chama-se de nível de significância e é representada pela letra grega minúscula α (alfa).

Para determinar se a hipótese nula deve ser rejeitada ou não, define-se uma região crítica, que é um conjunto de valores para os quais ela deve ser rejeitada. Assim, ao se realizar um teste de hipótese, obtém-se um valor a ser comparado a um valor crítico, o qual depende do nível de significância desejado para o teste e da distribuição dos dados. Além desse valor a ser comparado, também pode-se obter um parâmetro chamado de Valor-P, o qual indica o menor nível de significância para o qual pode-se rejeitar a hipótese nula (MONTGOMERY; RUNGER, 2016).

Já os intervalos de confiança permitem a criação de intervalos de estimadores para um determinado parâmetro populacional. Assim, ao invés de obter-se um único valor, que dificilmente será igual ao da população, determina-se um intervalo de valores que possivelmente contém o parâmetro estimado, com um determinado grau de confiança (MONTGOMERY; RUNGER, 2016). Esse grau de confiança (GC), por sua vez, é obtido a partir da equação $GC = 100*(1- \alpha)$, em que α é o nível de significância desejado. Valores comuns para os intervalos de confiança são 85%, 90%, 95% e 99%. É importante destacar que, à medida que o nível de confiança aumenta, o intervalo também torna-se maior, diminuindo-se a precisão na determinação do parâmetro populacional a ser estimado, uma vez que a quantidade de valores possíveis torna-se maior (MONTGOMERY; RUNGER, 2016).

Dessa forma, deve existir um equilíbrio entre o grau de confiança e a precisão desejada para o intervalo a ser construído.

2.4.1 Testes não paramétricos

No mundo da Estatística Inferencial, dois tipos de métodos podem ser empregados para a tomada de decisões sobre uma população: os testes paramétricos e os testes não paramétricos. Os primeiros envolvem parâmetros populacionais e consideram que as amostras possuem uma distribuição normal ou podem se aproximar dessa distribuição à medida que aumentam seu tamanho (Teorema do Limite Central) (CAÇÃO, 2010). Já os testes não paramétricos são empregados sobre conjuntos de dados que não seguem os critérios de normalidade necessários aos testes paramétricos, sendo geralmente empregados para testes de aleatoriedade, independência, relacionamentos e de modelos.

Para se determinar qual tipo de teste deve ser empregado sobre uma amostra com determinada distribuição, emprega-se o Teste de Normalidade, o qual verifica se a distribuição amostral é semelhante à Distribuição Normal (Gaussiana). Diversos testes podem ser empregados para essa finalidade, os quais variam de acordo com as características que consideram sobre os dados e as suposições que fazem. Apesar disso, todos podem gerar dois valores, que são uma estatística - normalmente comparada a um valor crítico, de acordo com o tipo de teste empregado - e um Valor-P. Assim, essas informações podem ser utilizadas em um teste de hipótese cuja hipótese nula é a afirmação de que a amostra possui uma distribuição normal e a hipótese alternativa é a negação dessa afirmação. Exemplos de testes de normalidade geralmente adotados são: o teste de Shapiro-Wilk, o teste de Anderson-Darling e o teste de Kolmogorov-Smirnov (BROWNLEE, 2018).

Uma vez determinado que o conjunto de dados (amostra) não possui uma distribuição normal, os métodos estatísticos a serem empregados, de maneira a se obter os melhores resultados, devem ser não paramétricos. Neles, os tipos de variáveis normalmente usadas são ordinais e nominais, embora também possa-se usar variáveis intervalares e de razão. Além disso, não se restringe o tipo de relacionamento que pode existir entre os dados, ou seja, eles podem ser dependentes ou independentes. Outro detalhe importante é que, diferentemente dos testes paramétricos, que utilizam a média, os testes não paramétricos geralmente empregam a mediana como medida de tendência central (CAÇÃO, 2010).

Nesse cenário, diversos tipos de testes estatísticos podem ser empregados, considerando-se as características dos dados, como, por exemplo, o seu tipo - ordinal,

nominal, intervalar, de razão - e o seu relacionamento - dependentes ou independentes (GUIMARÃES, 2018). Exemplos desses testes são: teste de Wilcoxon, teste de Friedman, teste de Kruskal-Wallis, teste de Dunn e teste de Nemenyi (GUIMARÃES, 2018)(PINHEIRO et al., 2019).

Por fim, é importante destacar que intervalos de confiança também podem ser calculados para populações cujas amostras não possuem uma distribuição normal. Para isso, uma técnica amplamente empregada é a *Bootstrap*. Ela aplica o método de reamostragem com reposição sobre a própria amostra, gerando novos conjuntos de dados com o mesmo tamanho do original. Para cada nova amostra gerada, o parâmetro de interesse é calculado e a diferença entre ele e o parâmetro gerado pela amostra original é armazenada. Então, repete-se esse processo um número *n* de vezes. Ao final, as diferenças armazenadas são ordenadas, os erros do intervalo são obtidos dessas diferenças de acordo com o grau de confiança e os limites inferiores e superiores são calculados, utilizando-se como base o parâmetro calculado para a distribuição original e os erros obtidos (ORLOFF; BLOOM , 2014).

Dessa maneira, este trabalho visa empregar testes de hipótese e intervalos de confiança para a análise dos experimentos realizados. Um teste de normalidade foi aplicado a cada conjunto de dados e, de acordo com seu resultado, técnicas não paramétricas foram abordadas, uma vez que a grande maioria dos dados apresentou distribuição diferente da normal.

3 TRABALHOS RELACIONADOS

Nesta seção, alguns trabalhos relacionados com o projeto realizado neste trabalho são apresentados.

3.1 Coletor de dados para monitoramento aéreo usando ADS-B e mini PC Android

Em Feitosa et al. (2015), um coletor de mensagens ADS-B foi implementado em Java para execução em uma plataforma embarcada com sistema operacional Android, cujas configurações de *hardware* podem ser observadas na Tabela 5. O sistema implementado se comunica, através de uma porta USB, com um receptor de sinais ADS-B, o qual fornece as mensagens ADS-B capturadas; então, ele as decodifica, utilizando uma API fornecida pelo projeto Open-Sky[4]; depois, ele salva as informações obtidas em um banco de dados, ainda na plataforma embarcada; por último, o sistema as envia para um servidor remoto.

O objetivo do trabalho apresentado foi a realização de uma análise de desempenho do sistema coletor (*hardware* e *software*) desenvolvido por Feitosa et al. (2015) e de um sistema coletor desenvolvido em outra fase do projeto que o autor utiliza como base. Para isso, ele realizou o teste dos dois *softwares* em duas plataformas de *hardware* diferentes, sendo uma delas de propósito geral e a outra, embarcada. Ele também analisou a rede de comunicação usada para trocar mensagens entre os sistemas coletores e um servidor remoto de armazenamento das informações.

As principais diferenças entre o trabalho de Feitosa et al. (2015) e o apresentado nesta monografia são as linguagens de programação empregadas para o desenvolvimento do *software*, que podem exigir maiores ou menores recursos da plataforma de *hardware* empregada; e o fato deste trabalho implementar a decodificação de mensagens ADS-B nativamente, ao passo que Feitosa et al. (2015) utiliza uma API de terceiros para essa finalidade.

Tabela 5 - Especificações de *hardware* do MK802.

CPU	AllWinner A10 @ 1.5GHz + Mali 400 GPU
Memória	512MB RAM
Armazenamento	4GB Flash microSD slot (Up to 32GB)
USB	micro USB 2.0/OTG port USB 2.0 Host port
Teclado	Teclado virtual do Android

[4] https://opensky-network.org/

Saída de Vídeo	HDMI (1080p)
Dimensões	8.8 x 3.5 x 1.2cm
Peso	200g

Fonte: CNXSoft (2015[5]apud FEITOSA et al., 2015).

3.2 SDR reception and analysis of civil aviation ADS-B signals

Em Su et al. (2018) é desenvolvido um sistema para recepção dos sinais de radiofrequência ADS-B enviados pelas aeronaves, sua transformação em uma *string* de dados (mensagem ADS-B), a decodificação das *strings* geradas para obtenção das informações e a apresentação dessas informações em uma interface gráfica.

A principal diferença entre o trabalho de Su et al. (2018) e este é que este utiliza uma plataforma de *hardware* de menor poder computacional, ao passo que Su et al. (2018) emprega uma plataforma de elevado desempenho. Além disso, Su et al. (2018) realiza a análise de seu sistema com objetivos de avaliar características diferentes das propostas neste trabalho, exceto por uma, que é a quantidade de mensagens recebidas por unidade de tempo. Ademais, Su et al. (2018) realiza uma implementação em Python e C++, ao passo que o sistema proposto por este trabalho será implementado em C e também analisará um código em Python.

3.3 Nanosatellite ADS-B receiver prototype for commercial aircraft detection

Em Pahlevy et al. (2018) foi realizado o desenvolvimento de um protótipo de um nano satélite para monitoramento aéreo baseado em tecnologia ADS-B. Para a construção do protótipo, foram utilizados como auxiliares uma antena e um receptor de mensagens ADS-B capaz de tratar as ondas de radiofrequência recebidas. Eles foram conectados a uma plataforma Raspberry Pi, que ficou responsável pelo processamento e armazenamento das mensagens recebidas. Por sua vez, a Raspberry foi conectada a um computador central localizado dentro do próprio nano satélite, o qual também estava conectado a uma interface de transmissão de dados.

Para a avaliação do projeto, alguns testes de desempenho foram realizados, buscando mensurar a acurácia do sistema na identificação das aeronaves, a velocidade de processamento dos dados recebidos para decodificação, a distância máxima até a qual uma aeronave poderia ser percebida e a potência do sinal necessária para isso.

[5] CNXSoft http://www.cnx-software.com

As principais diferenças entre o trabalho de Pahlevy et al. (2018) e este são que neste trabalho a *single board* utilizada na decodificação e armazenamento das mensagens deve se comunicar com um servidor remoto, enviando as informações extraídas para ele, ao passo que Pahlevy et al. (2018) envia os dados para um outro computador local, o que pode diminuir os esforços do *hardware* para a comunicação. Além disso, Pahlevy et al. (2018) analisa diversas questões relacionadas ao funcionamento do sistema, como quantidade de aeronaves identificadas, acurácia das informações, distância máxima até onde uma aeronave pode ser detectada e potência do sinal; ao passo que este trabalho se preocupa em verificar as características do sistema que podem influenciar o seu funcionamento, e o seu desempenho dadas essas características, como o uso de recursos computacionais e a quantidade de mensagens decodificadas.

3.4 Optimum receiver for decoding automatic dependent surveillance broadcast (ADS-B) signals

Em Abdulaziz et al. (2015) foi realizada a construção de um receptor de sinais de radiofrequência e de um decodificador de mensagens ADS-B. Para analisar o desempenho do sistema desenvolvido, foram utilizados dados reais, obtidos a partir de conjuntos de dados de controladores de tráfego aéreo. Eles foram empregados em simulações de ambientes reais, sendo transmitidos no formato de mensagens ADS-B e com a adição de ruídos. Com isso, o sistema de recepção e decodificação foi testado, avaliando-se sua capacidade de detecção de erros na checagem de paridade das mensagens ADS-B e seu desempenho através de indicadores de taxa de erro de bit e taxa de erro de pacote.

Um dos pontos que diferenciam o trabalho de Abdulaziz et al. (2015) do trabalho apresentado nesta monografia é que esta pretende enviar as informações extraídas para um servidor remoto, ao passo que Abdulaziz et al. (2015) não menciona se isso foi realizado ou não em seu trabalho. Isso é importante, pois a comunicação com outros sistemas, e a forma como ela é realizada, podem implicar em maiores ou menores esforços para o sistema sob análise. Outro ponto importante é que as métricas a serem usadas neste trabalho diferem-se das aplicadas por Abdulaziz et al. (2015). Este trabalho pretende analisar o consumo de recursos das plataformas de *hardware* empregadas e a taxa de mensagens processadas de acordo com o uso desses recursos. Já Abdulaziz et al. (2015) analisa a capacidade de seu sistema de detectar erros de paridade e realiza medições de taxa de erro de *bit* e taxa de erro de pacote.

3.5 Bringing up OpenSky: a large-scale ADS-B sensor network for research

Em Schäfer et al. (2014) é mostrado e discutido um trabalho já em operação chamado OpenSky. Este é um projeto em expansão que permite a visualização de aeronaves em tempo real, além de fornecer *software* e *hardware* coletor de mensagens ADS-B para os interessados em contribuir para o projeto ou em realizar pesquisas sobre essa tecnologia. Seu principal objetivo é fornecer uma rede de sensores aberta que possa contribuir para as pesquisas desenvolvidas sobre a tecnologia ADS-B, por diferentes comunidades de pesquisadores.

Para isso, o OpenSky possui uma grande base de dados brutos recolhidos por sensores (receptores) de mensagens ADS-B espalhados por várias regiões do globo. Esses dados são fornecidos para os voluntários do projeto - pessoas ou instituições que utilizam receptores ADS-B e enviam as informações coletadas para a plataforma do OpenSky - ou qualquer outra pessoa que realizar a requisição.

Em Schäfer et al. (2014), foi apresentada a arquitetura do sistema e um conjunto de análises, tanto sobre o sistema como um todo, como sobre a técnica de multilateração de ampla área (*wide-area multilateration*) realizada por esse sistema e uma avaliação de todos os problemas e aprendizados obtidos ao longo do tempo em que o projeto está em operação.

As diferenças encontradas entre este trabalho e o de Schäfer et al. (2014) dizem respeito aos objetivos da análise de desempenho realizadas sobre os sistemas. Este trabalho pretende analisar o consumo de recursos de *hardware* e a taxa de recepção e tratamento das mensagens, de maneira a determinar o desempenho de plataformas *single board* para aplicação em monitoramento aéreo. Já Schäfer et al. (2014) faz uma análise do desempenho de seu sistema ao longo dos dois anos de funcionamento, medindo a quantidade de mensagens ADS-B já recebidas durante esse período; verifica as taxas de envio das mensagens ADS-B, observando quantas mensagens de cada tipo foram enviadas por unidade de tempo; também analisa as métricas sobre o canal de comunicação, que é o *link* de dados de 1090MHz; e analisa seu sistema de multilateração, que consiste em uma técnica para se estimar a posição de uma aeronave, baseando-se não no tipo de informações enviadas por ela, mas na diferença do tempo de chegada dos sinais que ela envia, em quatro ou mais receptores.

3.6 Análise comparativa

Assim como feito em Feitosa et al. (2015), este trabalho tem como objetivo analisar o desempenho de dois sistemas coletores. Entretanto, o trabalho citado desenvolveu um

software em Java, ao passo que este implementa um *software* em linguagem de programação C, que é uma linguagem mais amplamente empregada no desenvolvimento de sistemas embarcados. O sistema implementado neste trabalho também deve se comunicar com um servidor remoto e receber as mensagens ADS-B através de um sistema receptor intermediário. Entretanto, diferentemente do projeto descrito em Feitosa et al. (2015), a tarefa de decodificação das mensagens ADS-B também foi implementada como parte do sistema desenvolvido, não sendo necessária a utilização de APIs de terceiros, possuindo um código próprio deste trabalho.

Já a relação entre este trabalho e o realizado em Su et al. (2018) encontra-se na implementação de um *software* para decodificação de mensagens ADS-B. Entretanto, neste trabalho não é proposta a implementação de um sistema receptor, capaz de tratar os sinais de radiofrequência enviados. Por outro lado, este trabalho visa analisar o desempenho do *software* implementado em uma plataforma de baixo custo e menor poder computacional; ao passo que o trabalho apresentado por Su et al. (2018) é implementado para execução em uma plataforma de elevado poder computacional, baseada na arquitetura de plataforma de processamento escalável Xilinx. Ademais, além da observação em comum empregada por Su et al. (2018) e este trabalho, que é a quantidade de mensagens por unidade de tempo, este trabalho também propõe a observação do uso de recursos das plataformas de *hardware*, como processador e memória.

Assim como em Pahlevy et al. (2018), esse projeto visa a utilização de um sistema receptor auxiliar para a captura das ondas de radiofrequência. Também propõe-se o desenvolvimento de um software para decodificação e armazenamento das informações extraídas a partir de mensagens ADS-B, o qual também deve ser executado em uma plataforma de menor poder computacional, semelhante à Raspberry Pi. Entretanto, ao invés de enviar as informações obtidas para um computador local, este trabalho propõe o envio das informações para um servidor remoto, através do emprego de uma ou mais interfaces de comunicação em rede.

Em Abdulaziz et al. (2015) foi realizada a implementação tanto de um receptor de sinais ADS-B como de um decodificador das mensagens capturadas. Já, neste trabalho, também propõe-se o desenvolvimento e análise de desempenho de um sistema capaz de realizar a decodificação de mensagens ADS-B. Entretanto, não será desenvolvido um receptor dos sinais eletromagnéticos e as métricas de desempenho a serem observadas são diferentes:

Abdulaziz et al. (2015) analisa a taxa de erro de *bit*, a taxa de erro de pacote e a capacidade do sistema de detectar erros de paridade; ao passo que este trabalho pretende analisar o consumo de recursos de *hardware* e a taxa de recepção e tratamento das mensagens. Além disso, implementamos a comunicação com um servidor remoto para envio das informações extraídas das mensagens, o que não foi apresentado por Abdulaziz et al. (2015).

Por fim, este trabalho propõe uma arquitetura semelhante à empregada por Schäfer et al. (2014), a qual será baseada na utilização de um receptor para captura das mensagens ADS-B, na execução de um software em uma plataforma computacional de baixo custo e no envio das informações obtidas para um servidor remoto. Schäfer et al. (2014) também realiza uma análise de desempenho sobre seu sistema, entretanto, os objetivos diferem-se dos propostos por este trabalho. Este trabalho pretende analisar o desempenho da plataforma *single board* nas atividades de monitoramento aéreo, observando consumo de recursos de *hardware* e taxas de tratamento das mensagens recebidas. Já, Schäfer et al. (2014) faz uma análise sobre seu sistema de multilateração, sobre o desempenho do canal de comunicação entre aeronaves e estações terrestres e sobre as mensagens já tratadas pelo seu sistema ao longo dos dois anos de funcionamento.

Na Tabela 6, é possível observar um resumo das características encontradas nos trabalhos relacionados descritos e a comparação com o projeto apresentado neste trabalho.

Tabela 6 - Análise comparativa entre os trabalhos relacionados e este trabalho.

Trabalho	**Implementa receptor ADS-B**	**Implementa decodificador ADS-B**	**Utiliza uma *single board***	**Envia dados para um servidor remoto**	**Analisa o *hardware* que executa o decodificador**
Feitosa et al. (2015)	Não	Não	Sim	Sim	Sim
Su et al. (2018)	Sim	Sim	Não	Não informa	Não
Pahlevy et al. (2018)	Não	Sim	Sim	Não	Não
Abdulaziz et al. (2015)	Sim	Sim	Não informa	Não informa	Não
Schäfer et al. (2014)	Não	Sim	Sim	Sim	Não

Este trabalho	Não	Sim	Sim	Sim	Sim

Fonte: Elaborada pela autora (2019).

Conforme explicado na Introdução, este trabalho faz parte do projeto "Atualização da Monitoração Aeronáutica e Auto-Sustentabilidade", que possui três principais vertentes: o desenvolvimento de um sistema embarcado para recepção de mensagens ADS-B, como o apresentado nesta monografia; a implementação de um sistema para prevenção de colisões entre aeronaves; e a criação de um sistema WEB em que as aeronaves monitoradas possam ser observadas.

Dessa forma, um dos motivos que levaram à construção, a partir do zero, do *software* apresentado neste trabalho foi a importância de se ter um sistema de recepção pertencente ao projeto citado. Além disso, apesar de outros coletores de mensagens ADS-B já terem sido desenvolvidos para o projeto, como o de Feitosa et al. (2015) e o Coletor em Python, teve-se como motivação para a implementação de um Coletor em C o fato de esta ser uma linguagem amplamente empregada para o desenvolvimento de sistemas embarcados, pois possibilita o emprego de abordagens que podem utilizar limitados recursos de maneira mais eficiente (SEBESTA, 2009).

4 METODOLOGIA

Para que os objetivos propostos neste trabalho fossem alcançados, um conjunto de etapas foi adotado, conforme descrito nas próximas subseções e enumerado a seguir:

a) Implementação do coletor de mensagens ADS-B em C.
b) Adaptação do coletor de mensagens ADS-B em Python.
c) Realização de experimentos preliminares.
d) Análise de desempenho do sistema desenvolvido.

4.1 Implementação do coletor de mensagens ADS-B em C

A linguagem de programação C é uma das mais utilizadas para a construção de sistemas e é a escolhida para o desenvolvimento de diversos projetos de sistemas embarcados (SEBESTA, 2009). Dessa maneira, neste trabalho foi desenvolvido um *software* para monitoramento aéreo em C, o qual teve sua implementação segmentada em quatro etapas, conforme a seguir.

4.1.1 Implementação da comunicação serial

As mensagens ADS-B, utilizadas para obtenção de informações de aeronaves, são enviadas por estas através de sinais eletromagnéticos que são captados por antenas receptoras. Para que esses sinais possam ser tratados por sistemas computacionais, eles precisam ser convertidos em sequências de *bits*. Entretanto, como o sistema desenvolvido neste trabalho não realiza essa conversão, necessitou-se empregar um sistema auxiliar, chamado de Receptor microADSB[6] (microADSB-USB *receiver* v2.0), que transforma os sinais recebidos pela antena em sequências de caracteres hexadecimais, as quais são enviadas por uma porta de comunicação USB (*Universal Serial Bus*).

Assim, para que o Coletor tenha acesso às mensagens enviadas pelas aeronaves, ele precisa estar conectado ao receptor através de uma das portas USB do *hardware* em que está sendo executado, realizando a leitura dessa porta serial. Na Figura 3, é possível observar a configuração desse sistema, em que a elipse A representa a antena e o receptor microADSB utilizados para a captura das mensagens ADS-B, a elipse B mostra a plataforma *single-board* em que o software Coletor é executado e a elipse C representa as informações sendo enviadas ao servidor remoto.

Durante essa etapa de desenvolvimento do Coletor, um importante problema precisou ser resolvido: a inicialização da comunicação com o receptor. Após a comunicação com a

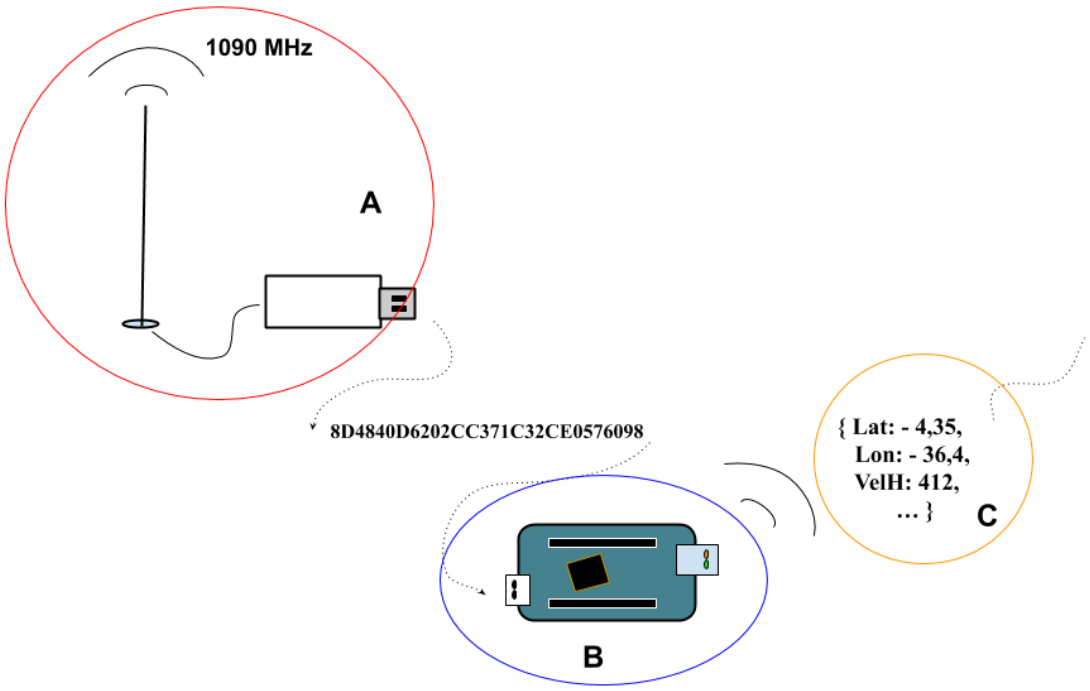
[6] http://www.microadsb.com/

porta serial ter sido implementada, tentou-se diversas vezes obter as informações enviadas pelo receptor, mas nada era recebido. Após algumas pesquisas e testes, foi descoberto que é necessário enviar ao receptor a *string "#43-02\n"* para que a troca de mensagens através da porta USB seja iniciada. Dessa forma, sempre que o Coletor é executado, ele envia essa *string* ao receptor, de maneira a inicializar a comunicação.

4.1.2 Implementação da decodificação das mensagens ADS-B

Ao comunicar-se com o receptor microADSB, o Coletor recebe uma sequência de caracteres hexadecimais (*string*) que representa a mensagem ADS-B enviada por uma aeronave. Entretanto, dependendo do receptor empregado, essa *string* pode possuir informações adicionais incluídas por ele e pode ser apresentada em diferentes formatos (A SIMPLE, 2010).

Figura 3 - Visão geral do sistema proposto.



Fonte: Elaborada pela autora (2019).

Dessa forma, ao iniciar-se o desenvolvimento do algoritmo de decodificação do Coletor, foi necessário determinar o formato das sequências que estavam sendo compartilhadas pelo receptor, o qual é mostrado na Tabela 7.

Tabela 7 - Formato das sequências de caracteres compartilhadas pelo receptor.

@	*Tag* de Tempo	Dados	;	<CR>	<LF>

Fonte: Adaptado de A Simple (2010).

A *Tag* de Tempo é uma informação de *timestamp* adicionada pelo próprio receptor e o campo de Dados compreende os 112 *bits* da mensagem ADS-B que foi recebida, conforme apresentado na Seção 2.1.2.1 da Fundamentação Teórica. Assim, o primeiro passo para a decodificação das mensagens ADS-B diz respeito à obtenção dos 112 bits de interesse a partir das sequências de caracteres enviadas pelo receptor. Na Figura 4, é possível observar uma *string* de caracteres compartilhada pelo receptor e a mensagem ADS-B extraída dela.

Figura 4 - Mensagem ADS-B obtida a partir de dados compartilhados pelo receptor.

Sequência de caracteres:
@1942231157618DE48413587B74CFAD67C260B143;
Mensagem ADS-B:
8DE48413587B74CFAD67C260B143

Fonte: Elaborada pela autora (2019).

Uma vez que a mensagem ADS-B é obtida, ela pode ser decodificada. Para isso, pode-se adotar os seguintes passos:

1. Verificar se a *string* recebida corresponde a uma mensagem ADS-B e, se ela corresponder, determinar se ela é do tipo ADS-B tratado neste trabalho. Para isso, verifica-se o campo *Downlink Format* (DF) da mensagem. Se DF = 17, o Coletor é capaz de tratar essa mensagem e prossegue-se para o próximo passo. Caso contrário, a mensagem é descartada e volta-se para a leitura da porta serial.
2. Verificar o tipo de informação que a mensagem carrega, o qual pode ser obtido através da verificação do campo *Type Code* (TC).
3. Se TC estiver entre 1 e 4, a mensagem carrega informações de identificação da aeronave e pode-se verificar o seu *callsign*, que é a identificação atribuída aquele voo.
4. Caso o *type code* esteja entre 9 e 18, a mensagem contém informações de posição, como a latitude, a longitude e a altitude da aeronave. É importante destacar-se que o Coletor emprega o método de decodificação global de mensagens ADS-B (JUNZI, 2018) para a obtenção da latitude e longitude de aeronaves. Isso significa que são necessárias duas mensagens, uma do tipo *Even* e outra do tipo *Odd*, enviadas pela mesma aeronave dentro de um determinado intervalo de tempo, para que sua latitude e sua longitude possam ser calculadas.

5. Se o TC for igual a 19, a mensagem possui informações de velocidade, as quais incluem a velocidade horizontal, a velocidade vertical e o *heading* (direção) da aeronave.

Por fim, um importante detalhe sobre a implementação da decodificação é que cada aeronave é representada em memória como um nó de uma lista encadeada e cada nó armazena todas as informações que são obtidas das mensagens ADS-B enviadas pela aeronave que ele representa. Para que essa associação ocorra, emprega-se o ICAO da aeronave, já que esta é uma informação que a identifica unicamente e está presente em todas as mensagens ADS-B.

Entretanto, a lista encadeada só deve armazenar nós de aviões que estão sob a área de cobertura do Coletor, evitando-se desperdiçar memória com dados de aeronaves que há muito tempo deixaram de ser monitoradas. Para isso, duas funções foram criadas: uma delas ordena a lista encadeada, de maneira que os nós que acabaram de ser criados ou atualizados sejam colocados para o final da lista; já, a outra desaloca todos os nós cujo tempo de vida - indicado pela diferença de tempo entre o instante atual e o instante em que os nós foram criados ou atualizados - seja maior do que um valor pré-determinado, dado em segundos. A ordenação facilita o processo de desalocação, pois pode-se percorrer a lista em ordem, do nó mais antigo ao mais recente.

4.1.3 Implementação das operações de banco de dados

Embora todas as informações ADS-B possam ser salvas em memória, algumas questões de projeto do Coletor conduziram à necessidade de utilização de um banco de dados para o seu armazenamento. Essas questões são:

a) A criação de um histórico de informações sobre as aeronaves, que pode ser empregado para desenvolvimentos futuros do projeto;
b) A facilitação do envio de informações para o servidor remoto, em caso de perda de conexão;
c) O emprego do banco de dados como uma interface de comunicação entre o código de decodificação de mensagens, atualmente desenvolvido, e um *software* de gerenciamento do sistema embarcado e de comunicação com o servidor remoto, que será desenvolvido em versões futuras do Coletor.

O banco de dados escolhido para este trabalho foi o SQLite, o qual é bastante empregado em aplicações de sistemas embarcados, devido a sua simplicidade e economia de recursos (KREIBICH, 2010). Além disso, foram implementadas tanto operações de escrita

como de leitura do banco de dados e apenas uma tabela foi criada, contendo todas as informações presentes no nó descrito anteriormente, além de dados sobre o Coletor. A estrutura dessa tabela pode ser observada na Figura 5, em que:

a) **id** é uma chave primária atribuída automaticamente pelo banco de dados;
b) **collectorKey** é uma sequência de caracteres que identifica unicamente o Coletor no servidor, sendo gerada por este no momento do cadastro do Coletor;
c) ***modeSCode*** é o ICAO da aeronave;
d) ***callsign*****, latitude, longitude, altitude,** ***verticalVelocity*****,** ***horizontalVelocity*** **e** ***groundTRackHeading*** são as informações obtidas a partir das mensagens ADS-B, conforme explicado na Seção 4.1.2;
e) ***timestamp*** e ***timestampSent*** representam os instantes de tempo em que as informações são salvas no banco de dados e enviadas para o servidor remoto, respectivamente;
f) ***messageDataId*** é a mensagem de onde o *callsign* é obtido;
g) ***messageDataPositionEven*** e ***messageDataPositionOdd*** são as mensagens utilizadas para a obtenção das informações de posição;
h) ***messageDataVelocity*** é a mensagem de onde as informações de velocidade e de *heading* são obtidas.

Figura 5 - Tabela do banco de dados que armazena as informações ADS-B.

```
CREATE TABLE "radarlivre_api_adsbinfo"
(
    "id" integer NOT NULL PRIMARY KEY AUTOINCREMENT,
    "collectorKey" varchar(64) NULL,
    "modeSCode" varchar(16) NULL,
    "callsign" varchar(16) NULL,
    "latitude" decimal NOT NULL,
    "longitude" decimal NOT NULL,
    "altitude" decimal NOT NULL,
    "verticalVelocity" decimal NOT NULL,
    "horizontalVelocity" decimal NOT NULL,
    "groundTrackHeading" decimal NOT NULL,
    "timestamp" bigint NOT NULL,
    "timestampSent" bigint NOT NULL,
    "messageDataId" varchar(100) NOT NULL,
    "messageDataPositionEven" varchar(100) NOT NULL,
    "messageDataPositionOdd" varchar(100) NOT NULL,
    "messageDataVelocity" varchar(100) NOT NULL
);
```

Fonte: Elaborada pela autora (2019).

Para que as informações de um nó possam ser salvas no banco, esse nó deve estar completo. Isso significa que ele deve ter um conjunto mínimo de informações consideradas

suficientes para que possam ser enviadas ao servidor. Neste trabalho, elas são *messageDataPositionEven*, *messageDataPositionOdd*, *callsign* e *messageDataVelocity*.

Dessa forma, sempre que uma nova mensagem é decodificada, verifica-se se o nó em que suas informações são armazenadas está completo e, se estiver, todas as informações desse nó são salvas no banco de dados. Uma vez que elas tenham sido salvas, as mensagens *messageDataPositionEven*, *messageDataPositionOdd* e *messageDataVelocity* são apagadas do nó, pois elas mudam com frequência. Ao passo que a informação de *callsign* é mantida, dado que ela não muda no decorrer do intervalo de monitoramento.

Além disso, sempre que um novo conjunto de informações é armazenado no banco de dados, realiza-se a sua leitura e as informações são armazenadas em uma outra lista destinada a guardar os dados que devem ser enviados ao servidor, conforme descrito na próxima subseção. É interessante destacar que os nós dessa lista possuem os mesmos campos que os nós da lista anteriormente mostrada. Também é importante ressaltar que a leitura do banco de dados pode ser feita concomitantemente à escrita, de maneira a se aproximar do comportamento esperado para o Coletor quando o serviço de gerenciamento do sistema embarcado estiver implementado.

4.1.4 Implementação da comunicação com o servidor remoto

O envio das informações para o servidor remoto passou por dois tipos de configurações. Em um primeiro momento, ele foi implementado de maneira sequencial à leitura do banco de dados, ou seja, sempre que informações eram lidas do banco, elas eram enviadas ao servidor. Entretanto, observou-se que o Coletor desenvolvido em Python cria uma *thread* exclusivamente para a comunicação com o servidor. Dessa forma, com o objetivo de tornar os dois algoritmos (C e Python) o mais semelhantes possível, e devido à observação de que essa segunda configuração pode tornar o código mais eficiente, pois deixa as três primeiras etapas independentes desta, o Coletor em C foi reestruturado de maneira a também adotar uma *thread* apenas para a comunicação com o servidor.

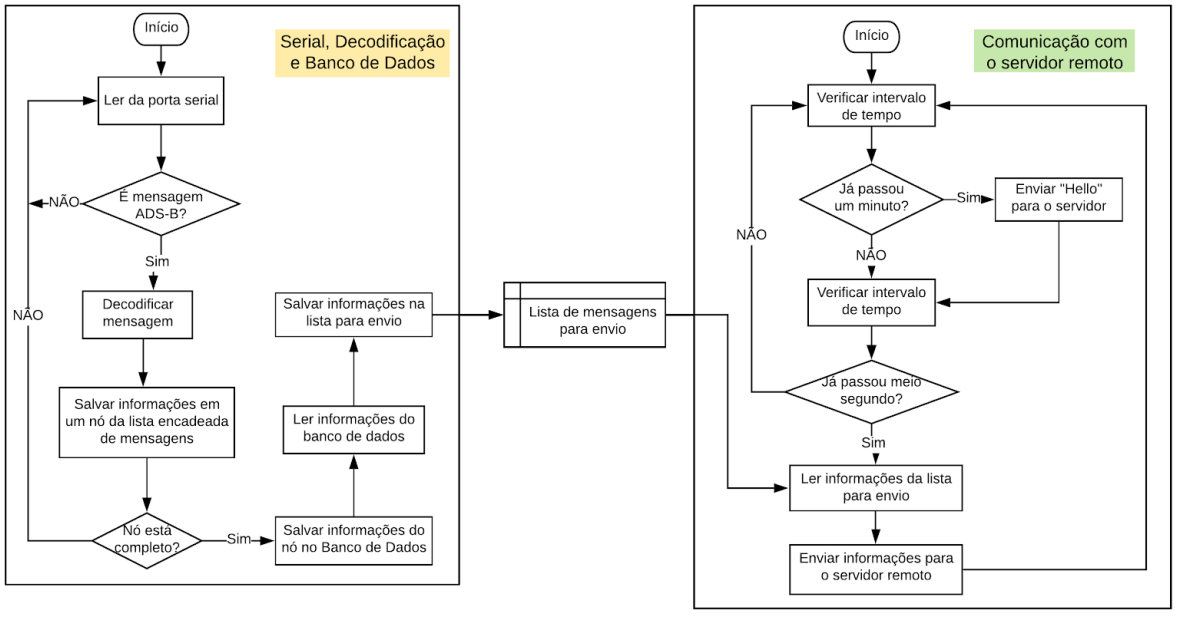
A interação entre essa *thread* e o restante do código do Coletor é feita através da lista que guarda as informações a serem enviadas ao servidor. Nessa *thread*, dois processos são periodicamente executados: um deles envia uma requisição HTTP *PUT* para o servidor, indicando que o Coletor permanece ativo, a cada um minuto; ao passo que o outro envia uma requisição HTTP *POST*, contendo as informações armazenadas na lista, a cada 500 ms.

Para que as informações da lista possam ser enviadas, elas precisam ser convertidas para um formato de representação que o servidor entenda. Assim, cria-se uma *string* no formato JSON para cada nó da lista, utilizando-se a biblioteca json-c[7]. Essas *strings* são concatenadas e dois colchetes são acrescentados, um no início e outro no final da sequência de caracteres resultante. Isso é feito porque o servidor espera receber uma lista de mensagens no formato JSON e ele entende que tudo o que está entre colchetes corresponde a elementos de uma lista. Uma vez que as informações tenham sido enviadas, todos os nós já utilizados são desalocados.

Já para a comunicação com o servidor remoto, este trabalho emprega a biblioteca Libcurl, que é bastante utilizada em aplicações de sistemas embarcados e facilita o desenvolvimento de *softwares* que se comunicam através da Internet (STENBERG, 2017).

Dessa forma, a estrutura do Coletor ADS-B implementado em C pode ser observada na Figura 6, em que se distinguem as duas *threads* do sistema: a primeira, responsável pela leitura da porta de comunicação serial, pela decodificação das mensagens ADS-B e pelas operações de banco de dados; e a segunda, encarregada da comunicação com o servidor remoto. Além disso, pode-se observar que a interação entre as duas ocorre através da lista que armazena as informações a serem enviadas.

Figura 6 - Estrutura do Coletor ADS-B desenvolvido em C.



Fonte: Elaborada pela autora (2019).

[7] https://ubuntu.pkgs.org/18.04/ubuntu-main-i386/libjson-c-dev_0.12.1-1.3_i386.deb.html

4.2 Implementação do coletor de mensagens ADS-B em Python

O Coletor em Python foi desenvolvido em etapas anteriores do projeto do qual este trabalho faz parte. Entretanto, para que ele e o Coletor em C apresentassem algoritmos semelhantes, de maneira a tornar mais justa a análise de desempenho, ele foi adaptado, adicionando-se as operações de escrita e leitura em banco de dados.

Inicialmente, cada uma das operações foi implementada utilizando-se a biblioteca do SQLite para Python. Todavia, foi observado que o código em Python vem com uma API (*Application Programming Interface*) para lidar com banco de dados, chamada de Peewee[8], que é uma ORM (*Object-Relational Mapping*) empregada para o mapeamento entre classes e tabelas de banco de dados.

Assim, a classe do Coletor em Python que armazena as informações ADS-B é utilizada para a criação da tabela. Além disso, existe uma classe que possui uma lista para cada tipo de mensagem ADS-B. Dessa forma, para que as informações obtidas das mensagens sejam salvas no banco de dados, a instância criada para essa classe precisa ser considerada completa e, para isso, utiliza-se como critério as mesmas informações adotadas pelos nós do Coletor em C, exceto que, ao invés do *callsign*, emprega-se o campo *messageDataId*.

Uma vez que a instância é considerada completa, as mensagens contidas nela são decodificadas e utilizadas para a criação de uma instância da classe que gera a tabela, a qual é armazenada no banco de dados. Após serem salvas, as informações são novamente lidas e armazenadas na lista que guarda os dados a serem enviados ao servidor.

Uma das principais diferenças entre os fluxogramas dos algoritmos em C e em Python é que, em C, sempre que uma nova mensagem ADS-B é recebida, ela é decodificada e suas informações são armazenadas em um nó de uma lista encadeada. Ao passo que o código em Python salva as mensagens recebidas e apenas as decodifica quando a instância da classe que as armazena é considerada completa. Na Figura 7, é possível observar a estrutura do algoritmo em Python.

4.3 Experimentos preliminares

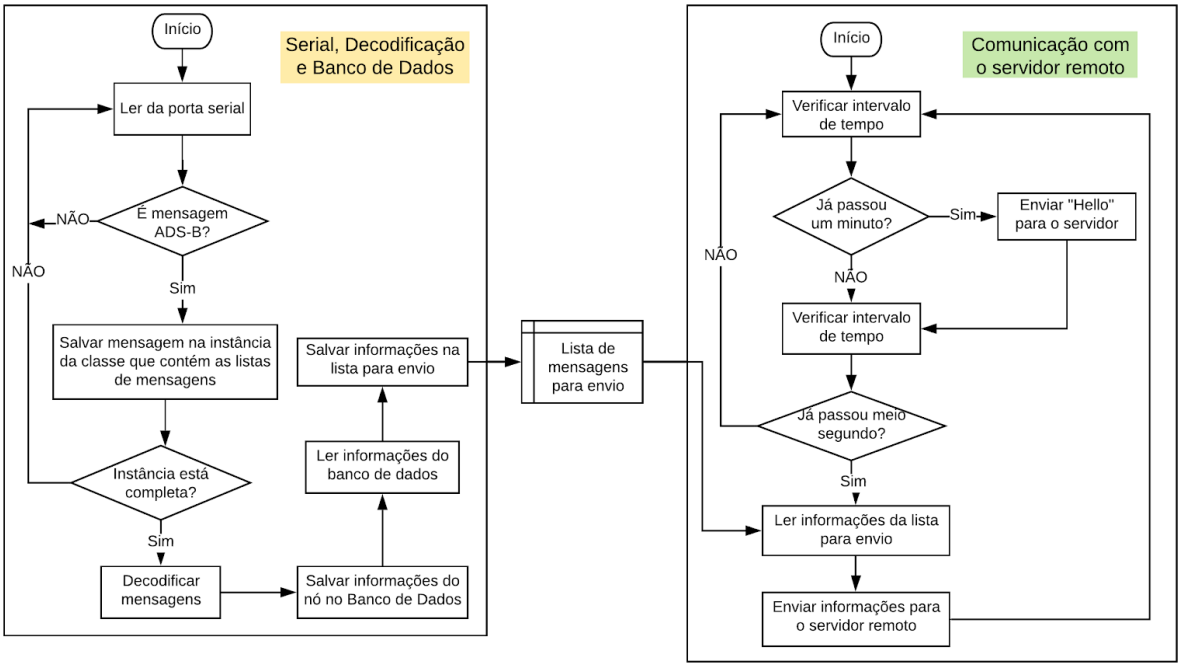
Os experimentos preliminares tiveram como objetivo testar os equipamentos a serem utilizados durantes os experimentos de avaliação de desempenho. Esses equipamentos foram quatro antenas usadas para a captura dos sinais enviados pelas aeronaves, dois receptores

[8] http://docs.peewee-orm.com/en/latest/

microADSB empregados para a conversão dos sinais em sequências de *bits* e duas plataformas *single-board (Orange Pi Pc Plus*) nas quais os coletores são executados.

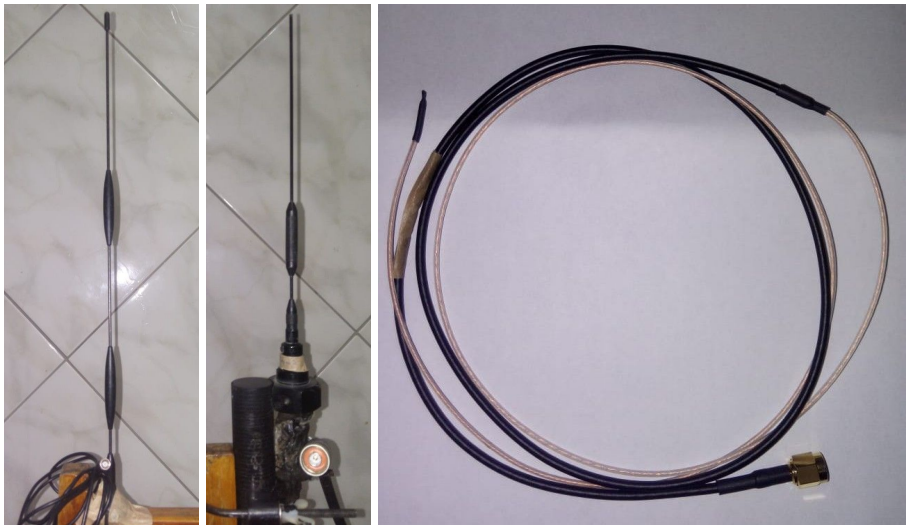
Nas Figuras 8a, 8b e 8c é possível observar as antenas 1, 2 e 3, respectivamente. A antena 4 é semelhante à antena 3, por isso não foi mostrada. Já na Figura 9, é possível observar o receptor microADSB 1. Como o receptor 2 é semelhante ao 1, apenas o primeiro foi mostrado. Por fim, na Figura 10 é possível observar a *single board* 1, que possui as mesmas configurações que a *single board* 2.

Figura 7 - Estrutura do Coletor ADS-B desenvolvido em Python.



Fonte: Elaborada pela autora (2019).

Figura 8 - Antenas usadas nos experimentos preliminares.



(a) (b) (c)

Fonte: Elaborada pela autora (2019).

Legenda: (a): Antena 1, (b) Antena 2, (c) Antena 3.

Em um primeiro momento, desejava-se testar se as duas antenas a serem empregadas, 1 e 2, tinham o mesmo desempenho. Entretanto, nenhuma mensagem foi apresentada na saída do receptor microADSB ao empregar-se Antena 2. Por se acreditar que o problema poderia estar na antena, passou-se a verificar quais antenas estavam funcionando, e as antenas 3 e 4 também foram testadas. Todavia, não houve saída em um dos receptores que as utilizavam.

Figura 9 - Receptor microADSB.

Fonte: Adaptado de Anteni.net ltd (2012).

Figura 10 - *Single board Orange Pi PC Plus.*

Fonte: Elaborada pela autora (2019).

Buscando-se entender os resultados desses primeiros experimentos, as antenas 3 e 4 foram testadas novamente, mas dessa vez os receptores microADSB foram trocados, ou seja, o receptor que estava com a Antena 3 foi colocado na Antena 4 e vice-versa. Então, foi observado que a antena que antes parecia não ter capturado mensagens, passou a capturar, e que a que tinha captado pareceu parar de captá-las.

Logo, o problema mostrou-se estar em um dos receptores. Para testá-los mais uma vez, o receptor que possivelmente estava com defeito foi colocado em uma antena que sabia-se estar funcionando (Antena 1) e o receptor que estava funcionando foi colocado em uma antena que parecia estar com defeito (Antena 2). Cada experimento teve um período

mínimo de captura de duas horas e suas configurações e resultados são apresentados na Tabela 8.

Tabela 8 - Configurações e resultados dos experimentos preliminares.

Experimento	Configuração	Resultado
1	Antena 1 + receptor 1 + *single-board* 1	Apresentou mensagens
	Antena 2 + receptor 2 + *single-board* 2	Não apresentou mensagens
2	Antena 3 + **receptor 1** + *single-board* 1	**Apresentou mensagens**
	Antena 4 + receptor 2 + *single-board* 2	Não apresentou mensagens
3	Antena 3 + **receptor 2** + *single-board* 1	**Não apresentou mensagens**
	Antena 4 + receptor 1 + *single-board* 2	Apresentou mensagens
4	Antena 1 + receptor 2 + *single-board* 1	Não apresentou mensagens
	Antena 2 + receptor 1 + *single-board* 2	Apresentou mensagens

Fonte: Elaborada pela autora (2019).

De acordo com os resultados dos experimentos, é possível observar que todas as antenas estavam funcionando e que o receptor microADSB 2 estava com defeito, pois todas as vezes em que ele foi usado, nenhuma mensagem foi apresentada. Por outro lado, todas as vezes em que o receptor microADSB 1 foi empregado, as mensagens foram apresentadas, mostrando que ele está funcionando.

4.4 Análise de desempenho

Conforme descrito na Seção 2.3 sobre análise de desempenho, é de grande importância o emprego de uma abordagem sistemática para se diminuir a probabilidade de erros durante a execução de uma análise. Dessa forma, nos próximos parágrafos, cada uma das etapas da abordagem sistemática será apresentada, de acordo com o trabalho proposto.

A análise de desempenho realizada tem como objetivos a avaliação do uso de recursos computacionais durante a execução dos *softwares* decodificadores de mensagens ADS-B e da capacidade do sistema de continuar a fornecer seus serviços, de acordo com a utilização desses recursos.

Dessa maneira, o sistema a ser considerado consiste em um *hardware* e um *software* empregados para as atividades de monitoramento aéreo. Dois *hardwares* foram empregados neste trabalho, sendo um deles uma plataforma *single board* (Tabelas 4 e 9) e o outro, um computador pessoal (Tabela 10). Por sua vez, dois softwares também foram utilizados, um desenvolvido em linguagem de programação C e o outro, em Python. Eles executam cinco

funções principais: a leitura de dados de uma porta de comunicação serial, a decodificação dos dados obtidos, o armazenamento das informações extraídas em um banco de dados, a leitura dessas informações do banco de dados e o seu envio para um servidor remoto.

Portanto, os serviços fornecidos pelo sistema compreendem a comunicação com um sistema externo - micro receptor ADS-B - para obtenção das mensagens ADS-B; a decodificação das mensagens recebidas, para obtenção da identificação, velocidade e posicionamento das aeronaves; essas informações, por sua vez, servem de base para os serviços de armazenamento e requisição ao banco de dados local; e, por último, a atualização das informações do servidor através do envio das mensagens armazenadas no sistema.

Tabela 9 - Especificações mais detalhadas do *hardware single board*.

Single Board	Orange Pi PC Plus
CPU	H3 Quad-core ARM Cortex-A7 H.265/HEVC 4K
GPU	Mali400MP2 GPU @600MHz suporta OpenGL ES 2.0
Memória	1GB DDR3 (compartilhada com GPU)
USB	Três USB 2.0 HOST, uma USB 2.0 OTG
Armazenamento	TF card (Max. 32GB) / MMC card slot 8GB EMMC Flash
Sistema operacional	Armbian Xenial 5.38 kernel 3.4.113 Orangepipcplus.

Fonte: Adaptado de Orange Pi (2016).

Tabela 10 - Especificações do computador pessoal.

CPU	Intel® Core™ i5-5200U CPU@ 2.20GHz×4
Memória	8GB, compartilhada com a placa de vídeo integrada.
Armazenamento	1 TB HD disco
Sistema operacional	Ubuntu 18.04.3 LTS

Fonte: Elaborada pela autora (2019).

Como métricas para análise de desempenho, empregou-se o uso de processador e memória durante a execução dos serviços; o tempo para o tratamento de cada mensagem; a quantidade total de mensagens recebidas por minuto, enviadas pela micro receptor ADS-B para a plataforma de hardware; dessas, a proporção das que são do tipo ADS-B, uma vez que a antena pode captar outros sinais que não sejam os enviados pelas aeronaves; e, dessa

proporção de mensagens ADS-B, a taxa de mensagens que o sistema consegue decodificar, já que ele não está tratando todos os tipos mensagens ADS-B.

Já em relação aos parâmetros que podem influenciar o desempenho do sistema, alguns deles são: os *softwares* que foram empregados, a localização das antenas e as plataformas de *hardware* utilizadas. Dentre esses parâmetros, os considerados fatores e seus respectivos níveis são: os *softwares*, os quais foram implementados em C e em Python; e os recursos computacionais, que variam dependendo se está sendo usada uma plataforma *single board* ou um computador pessoal. Por fim, a técnica de análise de desempenho empregada foi a de medição do sistema real.

4.4.1 Realização dos experimentos

Inicialmente, planejou-se a execução dos sistemas de monitoramento, simultaneamente, nas duas plataformas de *hardware* utilizadas, de maneira que eles tivessem acesso ao mesmo conjunto de dados. Entretanto, devido ao problema identificado nos experimentos preliminares - um dos receptores com defeito -, apenas uma plataforma pôde ser usada por vez. Assim, quatro cenários foram determinados, de maneira a testar os dois *softwares* coletores nas duas plataformas de *hardware*:

a) Cenário 1: *hardware single board* e Coletor em C.
b) Cenário 2: *hardware single board* e Coletor em Python.
c) Cenário 3: computador pessoal e Coletor em C.
d) Cenário 4: computador pessoal e Coletor em Python.

No total, foram realizados doze experimentos (capturas), cada um com duração mínima de oito horas e máxima de 12 horas e 35 minutos, alternando-se entre os quatro tipos de cenários da seguinte maneira: primeiro, empregou-se o Cenário 1, depois o Cenário 3, a seguir o Cenário 2 e, por fim, o Cenário 4. Então, repetiu-se essa ordem, até que os doze experimentos tivessem sido realizados, obtendo-se um total de três capturas por cenário. É importante destacar que todos os experimentos foram realizados na residência da autora e a condição de parada foi já ter passado o mínimo de oito horas.

Durante os experimentos, sete tipos de informações foram capturadas: o uso de processador e de memória, os quais foram medidos a cada cinco segundos; o tempo para o tratamento de uma mensagem, em segundos; todas as mensagens recebidas; dessas, as mensagens que eram do tipo ADS-B; dessas, as mensagens que foram decodificadas; e as mensagens que não foram decodificadas. É importante destacar que a soma da quantidade de

mensagens decodificadas e não decodificadas deve ser igual ao total de mensagens ADS-B recebidas.

Para a medição do tempo de tratamento de uma mensagem, foi considerado o tempo gasto para as atividades de decodificação, salvamento no banco de dados, leitura do banco de dados e salvamento na lista de mensagens para envio ao servidor.

Cada uma dessas informações foi armazenada em arquivos .csv, contendo os seguintes campos:

a) Coletor: pode assumir os valores 0, quando é empregado o Coletor em C, ou 1, quando utiliza-se o Coletor em Python.
b) *Hardware*: pode valer 0, quando usa-se a plataforma *single board*, ou 1, ao adotar-se o computador pessoal.
c) Dados: guarda a informação que está sendo capturada, a qual corresponde um dos sete tipos citados em parágrafos anteriores.
d) *Timestamp*: armazena o instante em que a informação foi obtida, sendo dado em milissegundos.

Um exemplo de um desses arquivos pode ser observado na Figura 11, em que apresenta-se as mensagens ADS-B recebidas quando emprega-se o Cenário 1 - plataforma *single board* e Coletor em C.

Figura 11 - Exemplo de arquivo com dados de um experimento.

```
0,0,8DE4956E588300637D2C760B052B,1574173597732
0,0,8DE4956E9914033000083D7AD139,1574173598266
0,0,8DE4956E201CC3F2C77DA058238C,1574173601533
0,0,8DE4956E9914033028043D195518,1574173615223
0,0,8DE4956E9914033028043D195518,1574173617229
0,0,8DE4956E5883006A3D2C73F5D284,1574173619895
0,0,8DE4956E5881F4836964159EC3DC,1574173652853
0,0,8DE4956E5881F483AD6415C96E27,1574173653781
0,0,8DE4956E9914033028043D195518,1574173654976
```

Fonte: Elaborada pela autora (2019).

4.4.2 Análise dos dados

O primeiro passo para a análise dos dados obtidos durante os experimentos foi a determinação de que recursos empregar para essa finalidade. Assim, escolheu-se utilizar o ambiente de desenvolvimento Jupyter Notebook[9], amplamente adotado em projetos de ciência de dados, devido às facilidades que fornece à aplicação de uma variedade de ferramentas para

[9] https://jupyter.org/

tratamento de dados. Além disso, optou-se por utilizar a linguagem de programação Python, devido à diversidade de recursos que ela disponibiliza para o emprego de métodos estatísticos e à facilidade para o manuseio e representação das informações.

Em cada experimento, um conjunto de arquivos foi gerado para cada tipo de informação, conforme pode ser visto na Tabela 11. Assim, como cada cenário possui três experimentos, ao todo, foram gerados 21 arquivos por cenário. Para um mesmo cenário, os experimentos diferenciam-se pelo instante em que foram realizados.

Tabela 11 - Experimentos, cenários e arquivos gerados.

Experimento	**Cenário**	**Arquivos gerados**
1	*Single board* e Coletor em C	exp1_cpu_usage_c.csv; exp1_mem_usage_c.csv; exp1_decoding_time_c.csv; exp1_adsb_received_msgs_c.csv; exp1_all_received_msgs_c.csv; exp1_decoded_msgs_c.csv; exp1_not_decoded_adsb_msgs_c.csv;
2	Computador pessoal e Coletor em C	exp2_cpu_usage_c.csv; exp2_mem_usage_c.csv; exp2_decoding_time_c.csv; exp2_adsb_received_msgs_c.csv; exp2_all_received_msgs_c.csv; exp2_decoded_msgs_c.csv; exp2_not_decoded_adsb_msgs_c.csv;
3	*Single board* e Coletor em Python	exp3_cpu_usage_python.csv; exp3_mem_usage_python.csv; exp3_decoding_time_python.csv;exp3_decoded_msgs_python.csv; exp3_adsb_received_msgs_python.csv; exp3_all_received_msgs_python.csv; exp3_not_decoded_adsb_msgs_python.csv;
4	Computador pessoal e Coletor em Python	exp4_cpu_usage_python.csv; exp4_mem_usage_python.csv; exp4_decoding_time_python.csv;exp4_decoded_msgs_python.csv; exp4_adsb_received_msgs_python.csv; exp4_all_received_msgs_python.csv; exp4_not_decoded_adsb_msgs_python.csv;
5	*Single board* e Coletor em C	exp5_cpu_usage_c.csv; exp5_mem_usage_c.csv; exp5_decoding_time_c.csv;exp5_adsb_received_msgs_c.csv; exp5_all_received_msgs_c.csv;exp5_decoded_msgs_c.csv; exp5_not_decoded_adsb_msgs_c.csv;
6	Computador pessoal e Coletor em C	exp6_cpu_usage_c.csv; exp6_mem_usage_c.csv; exp6_decoding_time_c.csv;exp6_adsb_received_msgs_c.csv; exp6_all_received_msgs_c.csv;exp6_decoded_msgs_c.csv; exp6_not_decoded_adsb_msgs_c.csv;
7	*Single board* e	exp7_cpu_usage_python.csv; exp7_mem_usage_python.csv; exp7_decoding_time_python.csv;exp7_decoded_msgs_python.

	Coletor em Python	csv; exp7_adsb_received_msgs_python.csv; exp7_all_received_msgs_python.csv; exp7_not_decoded_adsb_msgs_python.csv;
8	Computador pessoal e Coletor em Python	exp8_cpu_usage_python.csv; exp8_mem_usage_python.csv; exp8_decoding_time_python.csv;exp8_decoded_msgs_python.csv; exp8_adsb_received_msgs_python.csv; exp8_all_received_msgs_python.csv; exp8_not_decoded_adsb_msgs_python.csv;
9	*Single board* e Coletor em C	exp9_cpu_usage_c.csv; exp9_mem_usage_c.csv; exp9_decoding_time_c.csv;exp9_adsb_received_msgs_c.csv; exp9_all_received_msgs_c.csv;exp9_decoded_msgs_c.csv; exp9_not_decoded_adsb_msgs_c.csv;
10	Computador pessoal e Coletor em C	exp10_cpu_usage_c.csv; exp10_mem_usage_c.csv; exp10_decoding_time_c.csv;exp10_adsb_received_msgs_c.csv; exp10_all_received_msgs_c.csv;exp10_decoded_msgs_c.csv; exp10_not_decoded_adsb_msgs_c.csv;
11	*Single board* e Coletor em Python	exp11_cpu_usage_python.csv; exp11_mem_usage_python.csv; exp11_decoding_time_python.csv;exp11_decoded_msgs_python.csv; exp11_adsb_received_msgs_python.csv; exp11_all_received_msgs_python.csv; exp11_not_decoded_adsb_msgs_python.csv;
12	Computador pessoal e Coletor em Python	exp12_cpu_usage_python.csv; exp12_mem_usage_python.csv; exp12_decoding_time_python.csv;exp12_decoded_msgs_python.csv; exp12_adsb_received_msgs_python.csv; exp12_all_received_msgs_python.csv; exp12_not_decoded_adsb_msgs_python.csv;

Fonte: Elaborada pela autora (2019).

Entretanto, para as análises dos usos de processador e memória, e dos tempos de tratamento de mensagens, os dados de cada experimento foram reunidos em um único arquivo, de acordo com o tipo de informação e o cenário, gerando três arquivos para cada cenário: um com os dados de todos os experimentos de uso de processador; outro com os dados de todos os experimentos de uso de memória; e o último, para os tempos de tratamento de mensagens. Estes foram os principais arquivos empregados para a análise desses três tipos de informações, embora os arquivos de captura individuais também tenham sido usados em um determinado momento da análise.

Essas três informações foram tratadas em *notebooks* diferentes da plataforma Jupyter Notebook. Entretanto, as etapas empregadas foram, aproximadamente, as mesmas: realizou-se uma transformação dos dados, de maneira a facilitar sua visualização; fez-se a sua divisão em amostras menores; verificou-se que medida de tendência central utilizar para estimar um parâmetro populacional; empregou-se testes de normalidade para verificar que métodos estatísticos poderiam ser adotados; e realizou-se a análise dos dados de acordo com as técnicas estatísticas mais adequadas para o seu tipo de distribuição.

Já as informações sobre as mensagens foram tratadas em uma planilha do Google Drive. Registrou-se as seguintes informações, para cada cenário:

a) Quantidade de mensagens recebidas.
b) Quantidade de mensagens ADS-B recebidas.
c) Quantidade de mensagens decodificadas.
d) Quantidade de mensagens não decodificadas.
e) Taxa de mensagens recebidas por minuto.
f) Taxa de mensagens ADS-B recebidas por minuto.
g) Taxa de mensagens decodificadas por minuto.
h) Porcentagem de mensagens ADS-B em relação ao total de mensagens recebidas.
i) Porcentagem de mensagens decodificadas em relação ao total de mensagens ADS-B.

5 RESULTADOS E DISCUSSÕES

Nesta seção, apresenta-se os principais resultados obtidos durante o desenvolvimento deste trabalho e realiza-se discussões sobre eles.

5.1 Análise sobre o consumo de processador

O consumo de processador fornece um indicativo do esforço que o *hardware* está fazendo para que consiga realizar as diversas atividades requisitadas pelos *softwares* que estão sendo executados nele. Neste trabalho, realizou-se a medição da porcentagem de uso do processador nos quatro cenários descritos anteriormente, a cada cinco segundos, ao longo de cada captura. É importante destacar que os valores mensurados levam em consideração todos os processos sendo executados na plataforma e não apenas aqueles criados para a execução dos *softwares* de monitoramento aéreo. Além disso, para a análise desses dados foi empregada a plataforma Jupyter Notebook[10] e as bibliotecas Pandas[11], Numpy[12], Matplotlib[13], Seaborn[14], Scipy[15] e Sklearn[16].

Inicialmente, todos os arquivos de consumo de processador foram lidos e armazenados em um *data frame* (estrutura do Pandas) diferente, tanto os arquivos que contêm as capturas individualmente (por exemplo, exp1_cpu_usage_c.csv), como os que reúnem as capturas por cenário (por exemplo, total_cpu_usage_orangepi_c.csv). Após serem lidos, eles passaram por uma função que converte o valor de uso de processador presente nos arquivos para uma representação em porcentagem, uma vez que eles estavam em uma representação decimal. Um exemplo de *data frame* antes e depois da transformação pode observado nas Figuras 12 e 13.

A seguir, os dados de cada arquivo individual de captura foram plotados em um gráfico, cujo eixo vertical representa o uso de processador (CPU) e o eixo horizontal representa o tempo. Isso foi feito para que se pudesse observar o comportamento desse parâmetro ao longo do tempo. Nas Figuras 14a, 14b e 14c, é possível ver os gráficos para cada captura do Cenário 1; nas Figuras 15a, 15b e 15c, observa-se o uso de processador para o

[10] https://jupyter.org/
[11] https://pandas.pydata.org/
[12] https://numpy.org/
[13] https://matplotlib.org/
[14] https://seaborn.pydata.org/
[15] https://www.scipy.org/
[16] https://scikit-learn.org/stable/

Cenário 2; nas Figuras 16a, 16b e 16c apresenta-se os gráficos para o Cenário 3; e, por fim, nas Figuras 17a, 17b e 17c são apresentados os usos de processador para o Cenário 4.

Figura 12 - Exemplo de *data frame* antes da transformação, para o uso de CPU.

	collector	hardware	cpu_usage	timestamp
0	0	0	0.085451	1574173559958
1	0	0	0.084188	1574173564960
2	0	0	0.082845	1574173569961
3	0	0	0.081842	1574173574963
4	0	0	0.080566	1574173579965

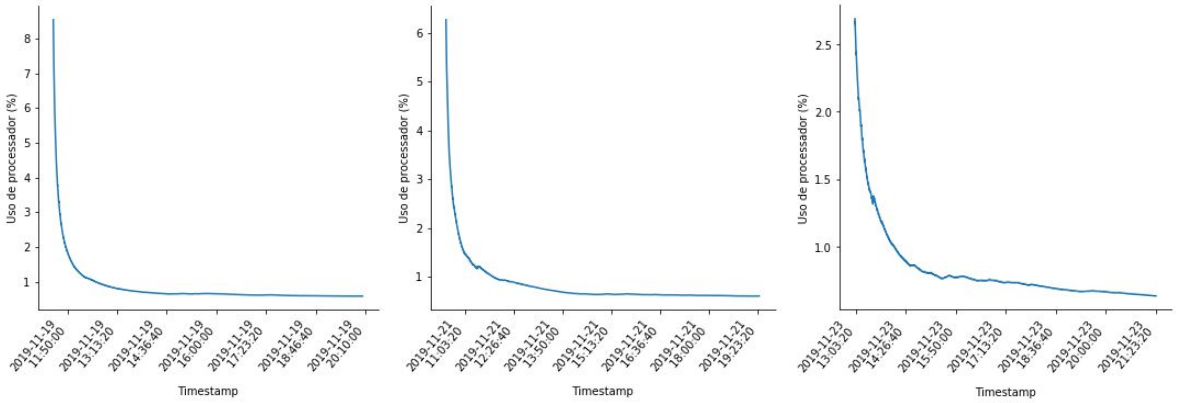
Fonte: Elaborada pela autora (2019).

Figura 13 - Exemplo de *data frame* após a transformação, para o uso de CPU.

	collector	hardware	cpu_usage	timestamp
0	0	0	8.5451	1574173559958
1	0	0	8.4188	1574173564960
2	0	0	8.2845	1574173569961
3	0	0	8.1842	1574173574963
4	0	0	8.0566	1574173579965

Fonte: Elaborada pela autora (2019).

Figura 14 - Uso de processador ao longo do tempo para o Cenário 1 (*single board* + C).



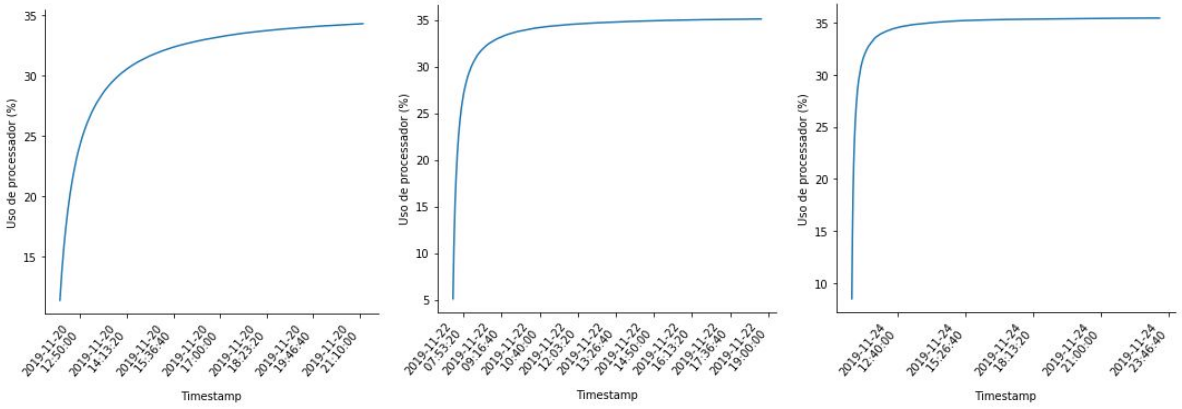
(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Conforme pode-se observar, nos cenários em que o Coletor em C foi executado, o uso de processador diminuiu com o decorrer do tempo. Ao passo que ele aumentou nos cenários em que o Coletor em Python foi executado. Isso pode sugerir que, ao longo do tempo, os

recursos de processamento exigidos pelo código em Python são maiores que aqueles impostos pelo código em C. Apesar disso, é importante destacar que, embora nos cenários 1 e 3 o uso de CPU tenha diminuído, ambos apresentam usos diferentes, sendo os do Cenário 3 maiores do que os do Cenário 1. Isso pode decorrer, principalmente, da existência de outros processos sendo executados no computador pessoal e que não existem na plataforma *single board*. Outra observação importante, que pode ser obtida da análise dos gráficos, é que o consumo de processamento no Cenário 2 torna-se maior do que aquele do Cenário 4. Ou seja, embora ambos cresçam com o decorrer do tempo, os recursos exigidos da plataforma *single board* tornam-se maiores do que aqueles requisitados do computador pessoal.

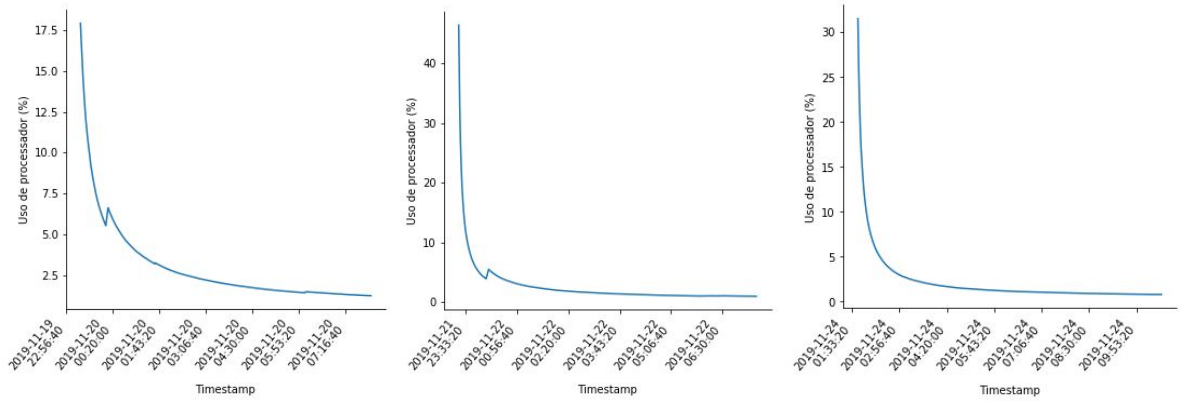
Figura 15 - Uso de processador ao longo do tempo para o Cenário 2 (*single board* + Python).



(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Figura 16 - Uso de processador ao longo do tempo para o Cenário 3 (PC + C).



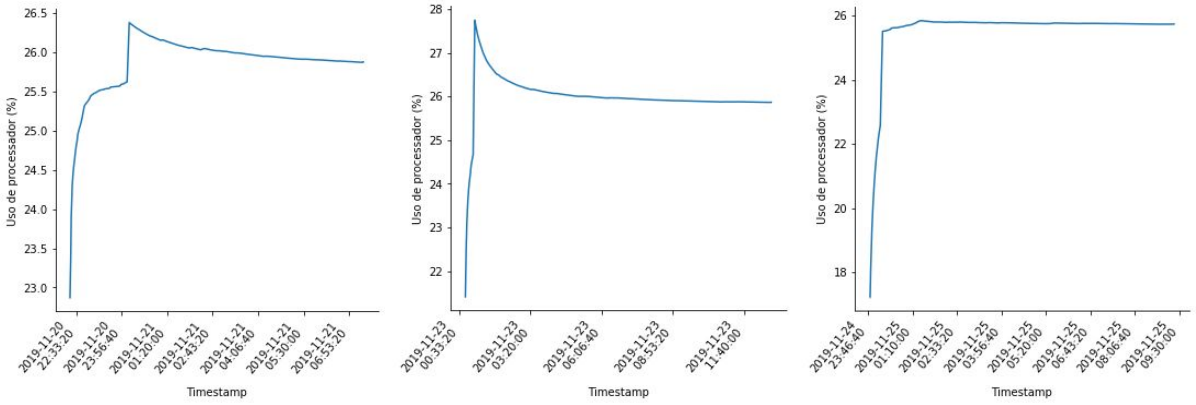
(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Uma vez verificadas as distribuições ao longo do tempo, o próximo passo foi a subdivisão dos dados contendo todas as capturas em amostras menores, para que pudessem

ser melhor analisados. Conforme explicado em seções anteriores, cada captura, ou experimento, teve uma duração mínima de oito horas. Dessa forma, foi realizada uma reamostragem desses dados, criando-se amostras com 120 elementos, a partir dos dados originais.

Figura 17 - Uso de processador ao longo do tempo para o Cenário 4 (PC + Python).



(a) (b) (c)

Fonte: Elaborada pela autora (2019).

Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Para cada uma dessas amostras, objetivou-se calcular um parâmetro para servir de estimador para a população. Assim, ao final de cada reamostragem, esse parâmetro foi calculado para a amostra criada e foi armazenado. Após todas as reamostragens terem sido feitas, o conjunto de parâmetros gerados foi utilizado para a criação de uma nova distribuição, a qual foi usada pelos métodos estatísticos que serão explicados em parágrafos posteriores.

Para se determinar o tipo de parâmetro a ser empregado, realizou-se um teste de normalidade sobre cada nova amostra criada. O teste empregado foi o de Anderson-Darling, com um grau de confiança de 95%. Para cada cenário, foi gerada uma estatística mostrando a quantidade de amostras que não rejeitaram a hipótese nula (Normal) - ou seja, foram consideradas normais - e a quantidade das que rejeitaram (Não normal). Essas estatísticas podem ser vistas na Tabela 12.

Tabela 12 - Normalidade das amostras para o uso de processador.

Cenário	Métrica	Normal	Não normal
1	Uso de processador	13	142
2	Uso de processador	0	190
3	Uso de processador	2	151
4	Uso de processador	2	180

Fonte: Elaborada pela autora (2019).

Em uma distribuição próxima à normal, o parâmetro adotado geralmente é a média, pois ela fornece uma boa caracterização dos dados, sendo o ponto médio a partir do qual a probabilidade acumulada em cada um dos lados da distribuição é a mesma. Entretanto, quando os dados possuem uma distribuição assimétrica, ou enviesada, a média deixa de ser um bom estimador. Nesse caso, a mediana pode ser considerada um bom estimador, pois ela preserva a propriedade citada anteriormente sobre o valor das probabilidades acumuladas.

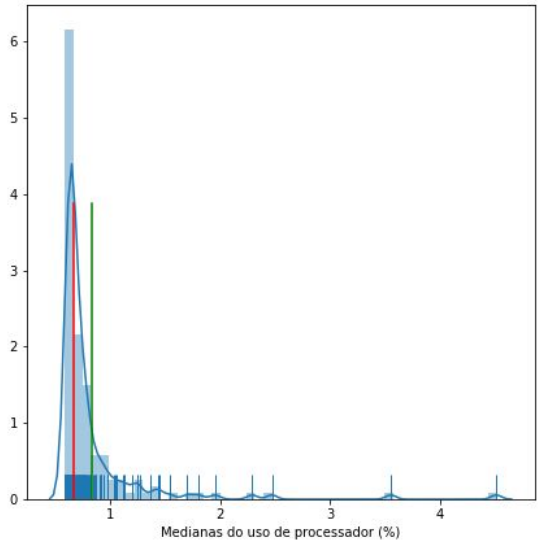
Conforme pode-se observar, o Cenário 2 não apresentou amostras com distribuição normal. Além disso, mesmo outros cenários tendo apresentado algumas amostras com distribuição normal, essa quantidade foi muito pequena se comparada ao total de amostras com distribuição não normal. Dessa forma, uma vez que a média e a mediana possuem, aproximadamente, o mesmo valor em distribuições normais, mas a mediana corresponde a uma representação mais eficiente dos dados quando a distribuição não é normal e tem-se um cenário em que a maioria das distribuições não é normal, escolheu-se a mediana como o parâmetro a ser estimado sobre a população.

Uma vez determinado o parâmetro, novamente realizou-se a reamostragem dos dados, dessa vez armazenando o parâmetro escolhido. Ao final da reamostragem, obteve-se um conjunto de 155 valores para o Cenário 1, 190 para o Cenário 2, 153 valores para o Cenário 3 e 182 para o Cenário 4. Isso decorre do fato de cada cenário possuir uma duração de experimentação diferente. De acordo com o que já foi explicado, a duração mínima foi de 8 horas. Apesar disso, alguns experimentos tiveram durações maiores.

Para que a distribuição de cada cenário tivesse a mesma quantidade de elementos, restringiu-se o número de valores de cada cenário à quantidade mínima encontrada entre todos eles. Nesse caso, o tamanho mínimo entre as distribuições geradas foi de 153. Então, todas as distribuições foram reduzidas a esse número de elementos. Nas Figuras 18, 19, 20 e 21 é possível observar a distribuição das medianas para cada cenário. O eixo horizontal representa os valores de mediana, o eixo vertical demonstra uma densidade para esses valores, a linha vermelha mostra a mediana da distribuição e a linha verde apresenta a média.

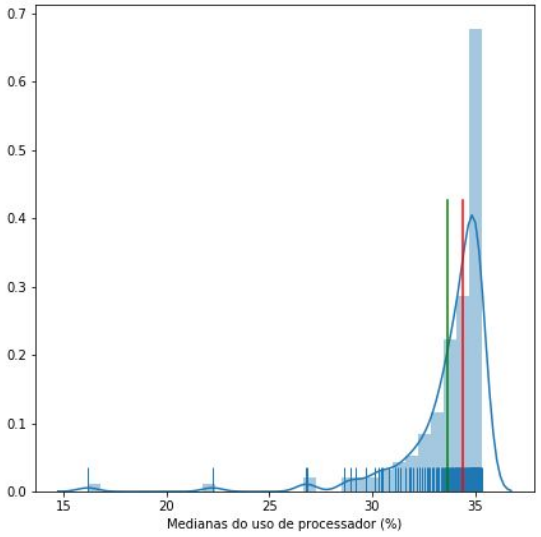
Conforme pode-se observar, as distribuições geradas também diferenciam-se da normal. Os valores de mediana e média para os cenários 1, 2, 3 e 4 foram, respectivamente, (0,6698; 0,83144), (34,41418; 33,67026), (1,54275; 2,51918) e (25,90781; 25,84694).

Figura 18 - Distribuição de medianas sobre o uso de processador, Cenário 1 (*single board* + C).



Fonte: Elaborada pela autora (2019).

Figura 19 - Distribuição de medianas sobre o uso de processador, Cenário 2 (*single board* + Python).



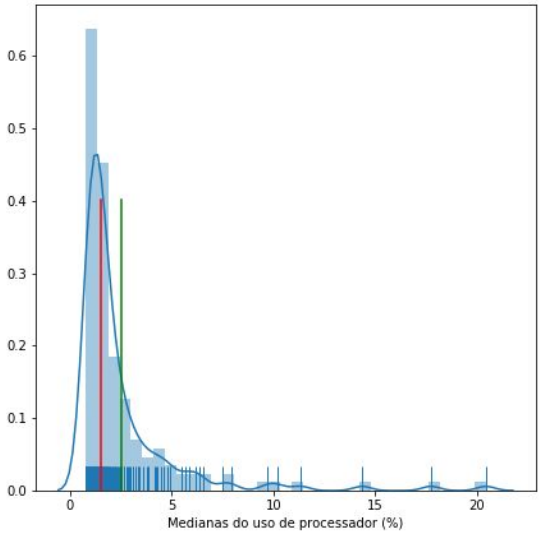
Fonte: Elaborada pela autora (2019).

O próximo passo na análise dos dados de uso de processador foi a verificação da normalidade da distribuição de cada cenário. Para isso, empregou-se novamente o método de Anderson-Darling, com um nível de significância de 5%, ou seja, um grau de confiança de 95%. Verificou-se que, para todas as distribuições, a hipótese nula foi rejeitada. Em outras palavras, elas não possuem uma distribuição normal.

Dado que as amostras dos cenários diferem-se de uma distribuição normal, métodos estatísticos não paramétricos foram empregados para a análise de algumas características desses dados. O primeiro método aplicado foi o teste de Kruskal-Wallis, o qual pode ser utilizado para se testar a semelhança entre três ou mais distribuições independentes. Assim, o

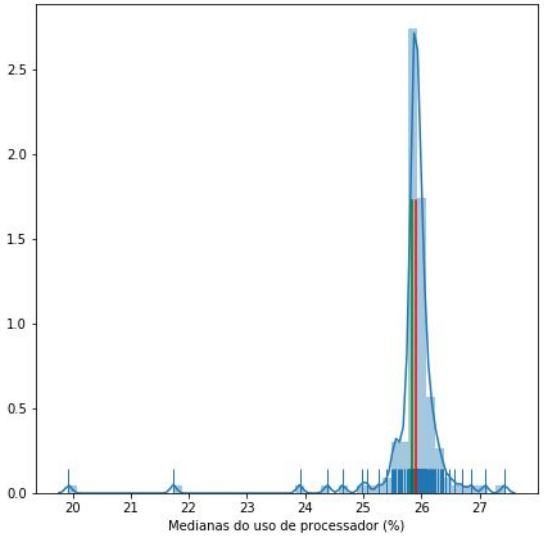
objetivo do emprego desse teste foi a verificação da existência de diferenças significativas entre cada cenário, ou seja, se os usos de processador foram semelhantes ou não para as diferentes configurações de experimentos. Dessa maneira, a hipótese nula afirma que não há diferenças significativas entre as distribuições e a hipótese alternativa afirma que há diferenças significativas. Para isso, um nível de confiança de 95% foi adotado.

Figura 20 - Distribuição de medianas sobre o uso de processador, Cenário 3 (PC + C).



Fonte: Elaborada pela autora (2019).

Figura 21 - Distribuição de medianas sobre o uso de processador, Cenário 4 (PC + Python).



Fonte: Elaborada pela autora (2019).

Como resultado, observou-se que o valor-p gerado foi de 8.886973361211593e-120, mostrando-se muito menor do que o nível de significância (alfa = 0,05) esperado para um grau de confiança de 95%. Assim, pôde-se concluir, com 95% de confiança, que os consumos de processador observados para os quatro cenários possuem diferenças. Isso pode indicar que

um ou outro cenário necessitou consumir mais recursos e também que as formas como esses recursos foram utilizados no decorrer do tempo podem ter distinguido-se. As Figuras 14, 15, 16 e 17, reforçam esse resultado.

Apesar de o teste de Kruskal-Wallis ter demonstrado que há diferenças entre as distribuições, ele não demonstrou quais delas diferenciam-se. Ou seja, ele apenas indicou que há pelo menos dois conjuntos de dados com diferenças significativas, mas ele não disse quais são. Para verificar quais distribuições diferenciam-se, foi aplicado o teste *post-hoc* de Dunn. Esse tipo de teste realiza comparações múltiplas entre pares de conjuntos de dados, retornando estatísticas que permitem determinar se há ou não diferença entre eles. No caso do teste de Dunn aplicado, um *data frame* com os valores-p é retornado. Para um nível de confiança de 95%, se esses valores forem menores do que alfa = 0,05, entende-se que há diferenças significativas entre os pares de conjuntos; caso contrário, entende-se que não há diferenças significativas.

A Figura 22 mostra o data frame gerado para a comparação entre os quatro cenários. Conforme pode-se observar, os valores de p são menores do que alfa ($p < 0,05$) para todas as comparações. Dessa forma, pode-se sugerir, com 95% de confiança, que todos os conjuntos de dados possuem diferenças significativas entre si. Em outras palavras, não houve dois cenários que apresentassem consumos de processador semelhantes.

Figura 22 - Teste de Dunn para o uso de processador.

	1	2	3	4
1	-1.000000e+00	1.033621e-107	1.094120e-10	8.818797e-49
2	1.033621e-107	-1.000000e+00	8.111178e-55	1.736358e-13
3	1.094120e-10	8.111178e-55	-1.000000e+00	1.945063e-16
4	8.818797e-49	1.736358e-13	1.945063e-16	-1.000000e+00

Fonte: Elaborada pela autora (2019).

Por fim, realizou-se o cálculo do intervalo de confiança para a mediana de cada uma das distribuições representativas dos cenários. Como essas distribuições não apresentaram um comportamento normal, nem foi possível determinar o tipo de distribuição observada, empregou-se uma técnica não paramétrica para o cálculo do intervalo de confiança, chamada de *Bootstrap*, com um nível de confiança de 95%. Os resultados desses cálculos são apresentados na Tabela 13. Assim, pode-se afirmar que, se um número grande de amostras aleatórias fossem coletadas da população, para cada tipo de cenário, e um intervalo de

confiança fosse calculado para cada uma delas, 95% desses intervalos conteriam o valor da mediana populacional. Em outras palavras, cada um dos intervalos mostrados na Tabela 13 possui 95% de chances de conter a mediana populacional para o cenário observado.

Tabela 13 - Intervalos de confiança para o uso de processador dos quatro cenários.

Cenário	Limite inferior	Mediana amostral	Limite Superior
1	0,63925	0,6698	0,6812
2	34,13725	34,41418	34,71531
3	1,34735	1,54275	1,67565
4	25,88	25,90781	25,92478

Fonte: Elaborada pela autora (2019).

5.2 Análise sobre o consumo de memória

O uso de memória também fornece indicativos do esforço que o *hardware* está realizando para cumprir as tarefas requisitadas. Neste trabalho, realizou-se a medição da quantidade de memória usada nos quatro cenários já descritos, a cada cinco segundos, no decorrer de cada experimento. Os valores foram medidos em *kilobytes* (KB). Também é importante destacar-se que eles foram mensurados levando-se em consideração não apenas os processos criados para os *softwares* de monitoramento aéreo, mas todas as atividades sendo executadas nas plataformas de *hardware* adotadas. Quase todos os passos empregados para a análise dos dados de uso de processador também foram abordados para a análise do uso de memória, exceto a transformação inicial dos valores, mantendo-os em *kilobytes*.

Inicialmente, todos os arquivos contendo os dados dos experimentos foram lidos, tanto os que contêm as capturas individualmente, como os que reúnem as capturas por cenário. Um exemplo do *data frame* criado para um desses cenários pode ser observado na Figura 23.

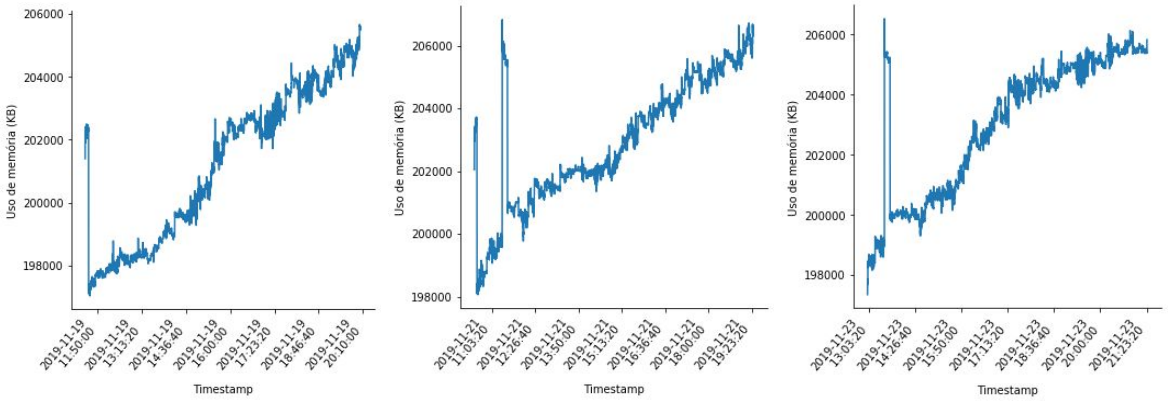
Figura 23 - Exemplo de *data frame* para consumo de memória.

	collector	hardware	mem_usage	timestamp
0	0	0	201400.0	1574173559958
1	0	0	202268.0	1574173564960
2	0	0	201940.0	1574173569961
3	0	0	202064.0	1574173574963
4	0	0	202000.0	1574173579965

Fonte: Elaborada pela autora (2019).

O próximo passo foi a representação dos consumos de memória, de cada captura realizada, em gráficos cujo eixo horizontal representa os instantes de tempo em que os valores foram medidos e o eixo vertical representa os usos de memória. Esses gráficos também foram construídos com o objetivo de observar o consumo de memória ao longo do tempo. Nas Figuras 24a, 24b e 24c pode-se observar os gráficos para o Cenário 1; nas Figuras 25a, 25b e 25c têm-se as representações para o Cenário 2; os gráficos para o Cenário 3 podem ser vistos nas Figuras 26a, 26b e 26c; e as Figuras 27a, 27b e 27c representam os usos de memória para o Cenário 4.

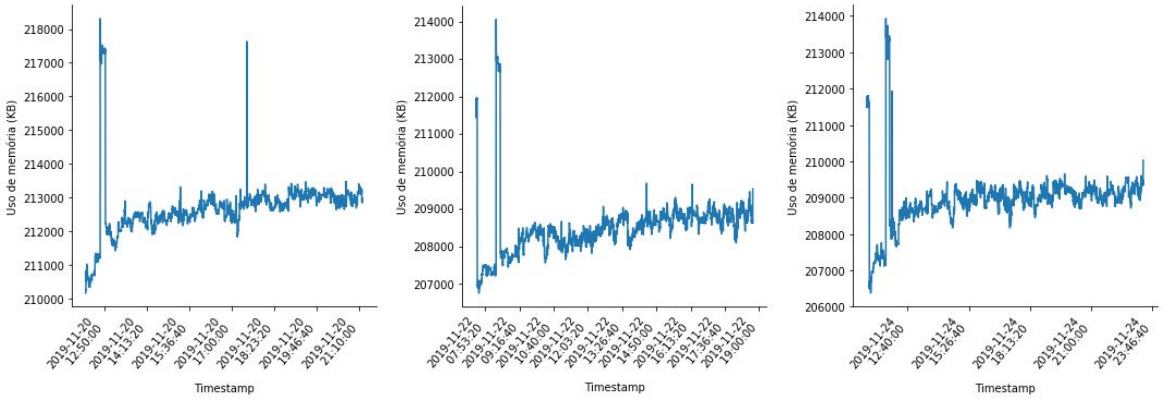
Figura 24 - Uso de memória ao longo do tempo para o Cenário 1 (*single board* + C).



(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Figura 25 - Uso de memória ao longo do tempo para o Cenário 2 (*single board* + Python).



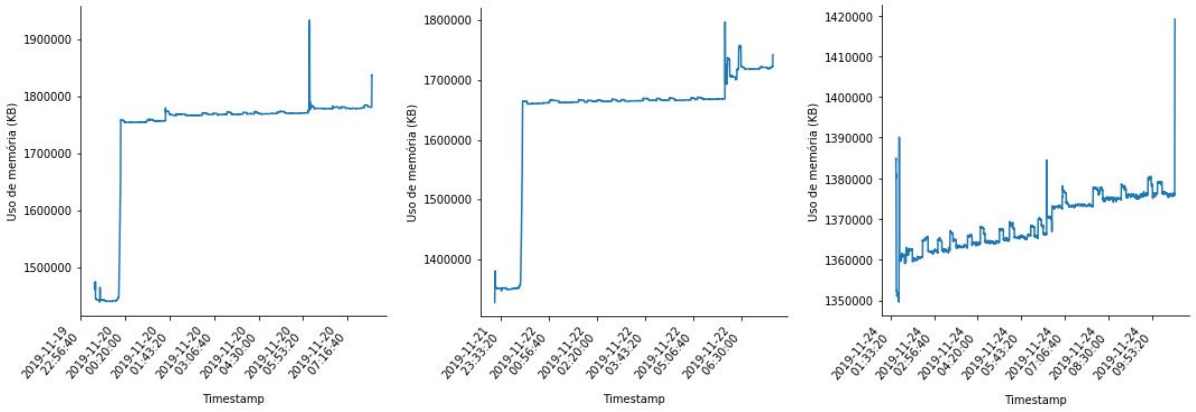
(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Conforme pode-se observar, em todos os cenários o consumo de memória aumenta com o decorrer do tempo. Entretanto, nota-se que para os cenários 1 e 2, em que os experimentos foram realizados na plataforma *single board*, os usos de memória apresentaram

maiores variações, conforme pode ser percebido pelas curvas mais ruidosas. Isso mostra que, mesmo que os valores estivessem crescendo, se um pequeno intervalo de tempo fosse separado, observaria-se a alternância entre valores mais altos e mais baixos. Além disso, pode-se perceber que o consumo de memória, na maior parte das vezes, é maior para o Cenário 2 do que para o Cenário 1. Com relação aos cenários 3 e 4, observou-se que os valores mostram-se mais estáveis, embora também crescentes. Ademais, embora os valores de uso de memória tenham se mostrado próximos para esses dois cenários, pode-se notar que eles diferem-se bastante em suas terceiras capturas.

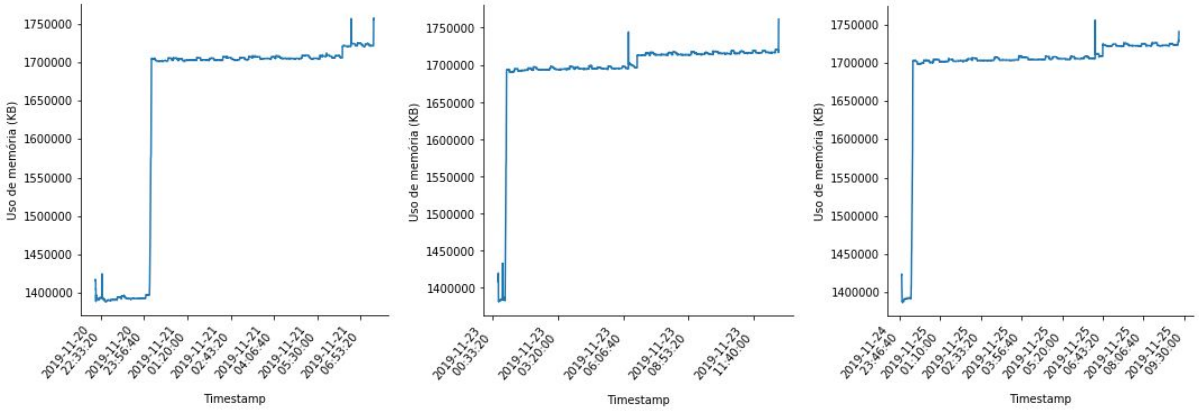
Figura 26 - Uso de memória ao longo do tempo para o Cenário 3 (PC + C).



(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Figura 27 - Uso de memória ao longo do tempo para o Cenário 4 (PC + Python).



(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Da mesma forma como feito para a análise dos consumos de processador, realizou-se a divisão das capturas de todos os cenários em amostras menores de tamanho 120. Essas novas amostras passaram por testes de normalidade, para se verificar o parâmetro estatístico a

ser empregado para a análise dos dados. O teste de normalidade utilizado foi o de Anderson-Darling, com 95% de confiança. Os resultados desses testes podem ser observados na Tabela 14.

Assim como aconteceu para o uso de processador, algumas amostras foram consideradas com distribuições normais. Entretanto, como a maioria das distribuições foram consideradas não normais - hipóteses nulas rejeitadas -, escolheu-se novamente empregar a mediana das distribuições como o parâmetro a ser estimado.

Tabela 14 - Normalidade das amostras para o uso de memória.

Cenário	Métrica	Normal	Não normal
1	Uso de memória	3	152
2	Uso de memória	1	189
3	Uso de memória	6	147
4	Uso de memória	10	172

Fonte: Elaborada pela autora (2019).

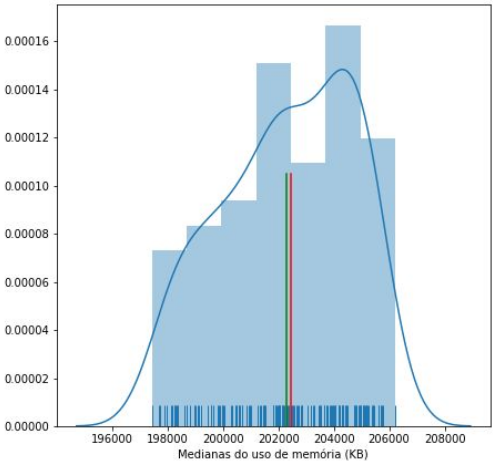
As próximas etapas consistiram na reamostragem dos dados originais para gerar as amostras menores; no cálculo da mediana para cada amostra e na criação da distribuição de medianas para cada cenário, conforme feito para a análise do uso de processamento. Depois, realizou-se a diminuição do número de elementos das amostras de alguns dos cenários, de maneira que todas essas distribuições tivessem a mesma quantidade de elementos. Nesse caso, como a quantidade mínima foi de 153, este foi o tamanho adotado para elas. Nas Figuras 28, 29, 30 e 31 é possível observar a distribuição das medianas para cada cenário. O eixo horizontal representa os valores de mediana, o eixo vertical demonstra uma densidade para esses valores, a linha vermelha mostra a mediana da distribuição e a linha verde apresenta a média.

Uma vez criadas as distribuições, o próximo passo foi a verificação de sua normalidade, de maneira a se determinar o conjunto de métodos estatísticos que poderiam ser empregados para sua análise. Para isso, empregou-se o teste de Anderson-Darling, com um nível de confiança de 95%. Como resultado, todas as distribuições foram consideradas não normais.

Assim, métodos não paramétricos foram empregados para a análise dos dados. Eles foram os mesmos utilizados na seção anterior. Com o teste de Kruskal-Wallis, objetivou-se

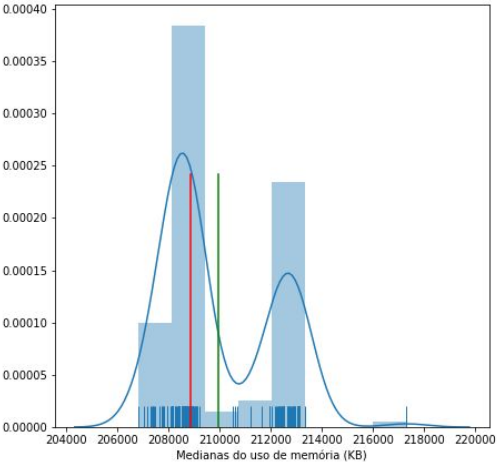
verificar se as distribuições possuíam diferenças significativas entre si. O resultado do teste indicou que devia-se rejeitar a hipótese nula, ou seja, as distribuições possuíam diferenças significativas. De outro modo, existem diferenças entre o consumo de memória para os diferentes cenários.

Figura 28 - Distribuição de medianas sobre o uso de memória, Cenário 1 (*single board* + C).



Fonte: Elaborada pela autora (2019).

Figura 29 - Distribuição de medianas sobre o uso de memória, Cenário 2 (*single board* + Python).



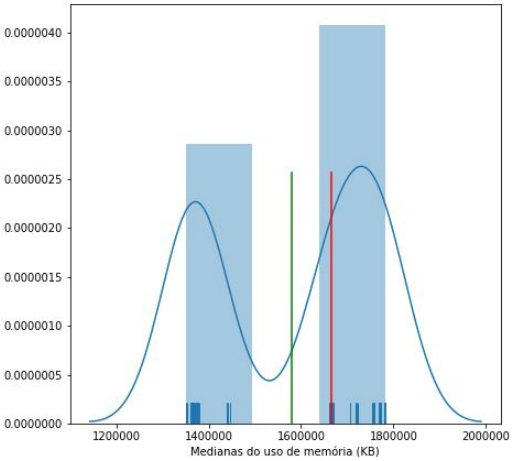
Fonte: Elaborada pela autora (2019).

Além disso, como o teste de Kruskal-Wallis não informa quais distribuições diferem-se, aplicou-se o teste post-hoc de Dunn para se obter comparações entre os pares de cenários, com um nível de significância de 5%. Os resultados desse teste podem ser vistos na Figura 32.

Conforme pode-se observar, quase todos os valores de p são menores do que alfa ($p < 0,05$). Entretanto, na comparação entre os Cenários 3 e 4, o valor de p mostrou-se maior que 0,05 ($0,07 > 0,05$). Dessa forma, pode-se sugerir, com 95% de confiança, que todos os conjuntos de dados, exceto os conjuntos 3 e 4, possuem diferenças significativas entre si. Em

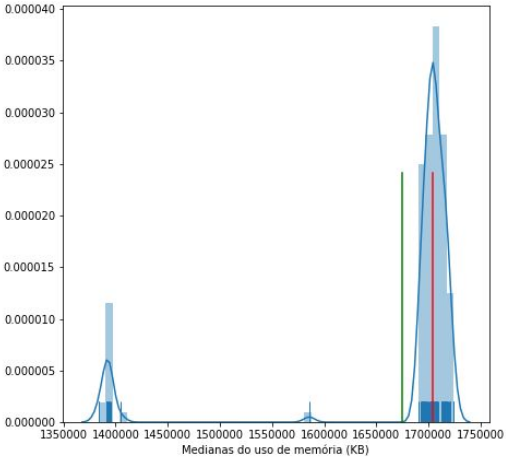
outras palavras, o consumo de memória foi semelhante para os cenários 3 e 4, mas foi diferente considerando-se as outras comparações. Esses resultados podem ser reforçados ao observar-se as Figuras 24, 25, 26 e 27.

Figura 30 - Distribuição de medianas sobre o uso de memória, Cenário 3 (PC + C).



Fonte: Elaborada pela autora (2019).

Figura 31 - Distribuição de medianas sobre o uso de memória, Cenário 4 (PC + Python).



Fonte: Elaborada pela autora (2019).

Figura 32 - Teste de Dunn para o uso de memória.

	1	2	3	4
1	-1.000000e+00	3.777041e-14	1.476140e-72	1.745086e-87
2	3.777041e-14	-1.000000e+00	1.513747e-25	1.513899e-34
3	1.476140e-72	1.513747e-25	-1.000000e+00	7.005939e-02
4	1.745086e-87	1.513899e-34	7.005939e-02	-1.000000e+00

Fonte: Elaborada pela autora (2019).

Por fim, também foi realizado o cálculo do intervalo de confiança para a mediana de cada cenário. Como as distribuições não foram consideradas normais e não se pôde determinar o tipo de distribuição observada, empregou-se novamente a técnica de *Bootstrap*, com nível de confiança de 95%. Os valores obtidos podem ser observados na Tabela 15. É possível concluir que cada intervalo possui 95% de chances de conter a mediana da população, para os cenários considerados.

Tabela 15 - Intervalos de confiança para o uso de memória dos quatro cenários.

Cenário	Limite inferior	Mediana amostral	Limite Superior
1	201604	202438	202868
2	208776	208870	208992
3	1663310	1665400	1669816
4	1703220	1704102	1705188

Fonte: Elaborada pela autora (2019).

5.3 Análise sobre o tempo de tratamento das mensagens

O tempo para tratamento das mensagens fornece indicativos da eficiência do sistema de monitoramento na realização de suas atividades. Quanto maior for esse tempo, mais probabilidade haverá de que algumas mensagens sejam perdidas ou que as informações não sejam entregues a tempo. Neste trabalho, considerou-se o tempo para tratamento das mensagens como sendo todo o intervalo para a decodificação das mensagens, o armazenamento no banco de dados, a leitura do banco de dados e o armazenamento das informações na lista de mensagens para envio ao servidor.

A metodologia empregada para esta métrica foi a mesma utilizada para as duas descritas nas seções anteriores. Assim, primeiramente realizou-se a leitura dos conjuntos de dados, tanto dos arquivos de cada captura, como daqueles que continham todos os dados de captura de acordo com o cenário. Depois, aplicou-se uma função sobre os valores de tempo, de maneira a apresentá-los em microssegundos ao invés de segundos. Isso foi feito devido à grande quantidade de valores próximos de zero. Um exemplo de *data frame* para um dos arquivos lidos, antes e depois das transformações, pode ser observado nas Figura 33 e 34.

A seguir, os dados de cada arquivo individual de captura foram plotados em um gráfico, cujo eixo vertical representa o tempo para tratamento das mensagens e o eixo

horizontal representa os *timestamps* de quando as medições foram realizadas. Isso foi feito para que se pudesse observar o comportamento desse parâmetro ao longo do tempo. Nas Figuras 35a, 35b e 35c, é possível ver os gráficos para cada captura do Cenário 1 (*single board* + C); nas Figuras 36a, 36b e 36c, observa-se os tempos de tratamento para o Cenário 2 (*single board* + Python); nas Figuras 37a, 37b e 37c, apresenta-se os gráficos para o Cenário 3 (PC + C); e, por fim, nas Figuras 38a, 38b e 38c são apresentados os tempos de tratamento para o Cenário 4 (PC + Python).

Figura 33 - Exemplo de *data frame* antes da transformação, para o tempo de tratamento de mensagens.

	collector	hardware	decoding_time	timestamp
0	0	0	0.001205	1574173597733
1	0	0	0.000590	1574173598267
2	0	0	0.000457	1574173601533
3	0	0	0.000205	1574173615223
4	0	0	0.000455	1574173617230

Fonte: Elaborada pela autora (2019).

Figura 34 - Exemplo de *data frame* após a transformação, para o tempo de tratamento de mensagens.

	collector	hardware	decoding_time	timestamp
0	0	0	1205.0	1574173597733
1	0	0	590.0	1574173598267
2	0	0	457.0	1574173601533
3	0	0	205.0	1574173615223
4	0	0	455.0	1574173617230

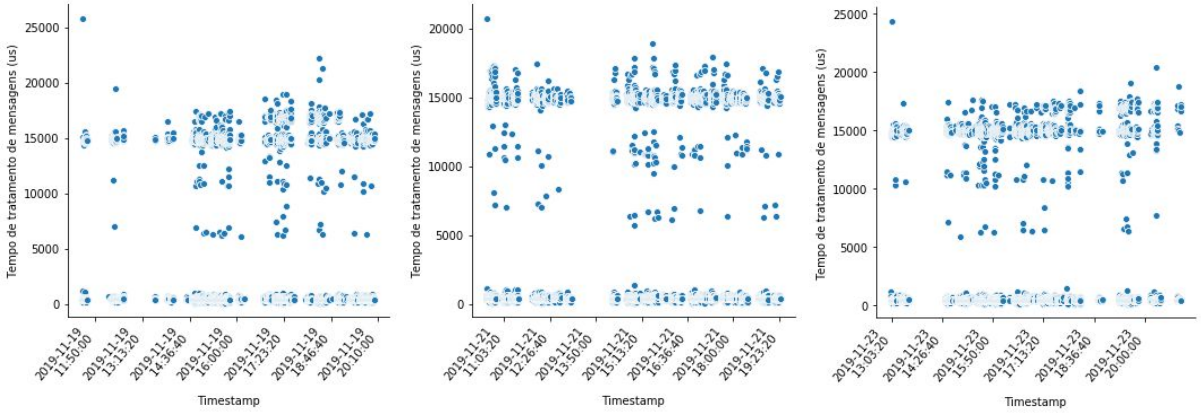
Fonte: Elaborada pela autora (2019).

Um detalhe importante é que, originalmente, o gráfico para o primeiro experimento do Cenário 2 apresentou-se como o mostrado na Figura 39. Pôde-se observar que duas medições apresentaram valores negativos, os quais estavam deslocando a representação do gráfico e atrapalhando a visualização do restante das medições. Não foi possível identificar o problema que gerou esses valores negativos, já que a diferença entre um *timestamp* mais recente e um mais antigo deve ser maior do que zero. Como foram apenas duas medições, considerou-se melhor retirá-las do conjunto de dados, gerando o gráfico visto na Figura 36a. Isso pode indicar possíveis problemas na execução do código em Python na plataforma Orange Pi.

Entretanto, é importante destacar-se que nenhum outro caso como esse foi observado nos outros experimentos nem em outros cenários.

De acordo com os gráficos, pode-se observar que os cenários em que o Coletor em C foi executado (1 e 3) apresentaram menores tempos de tratamento do que os cenários em que o código em Python foi executado (2 e 4). Dessa forma, os dois melhores cenários - aqueles que apresentam os menores tempos de decodificação - foram o 1 e o 3, com o Cenário 3 melhor do que o Cenário 1; e o pior cenário foi o 2, que apresentou muitos tempos de tratamento em segundos.

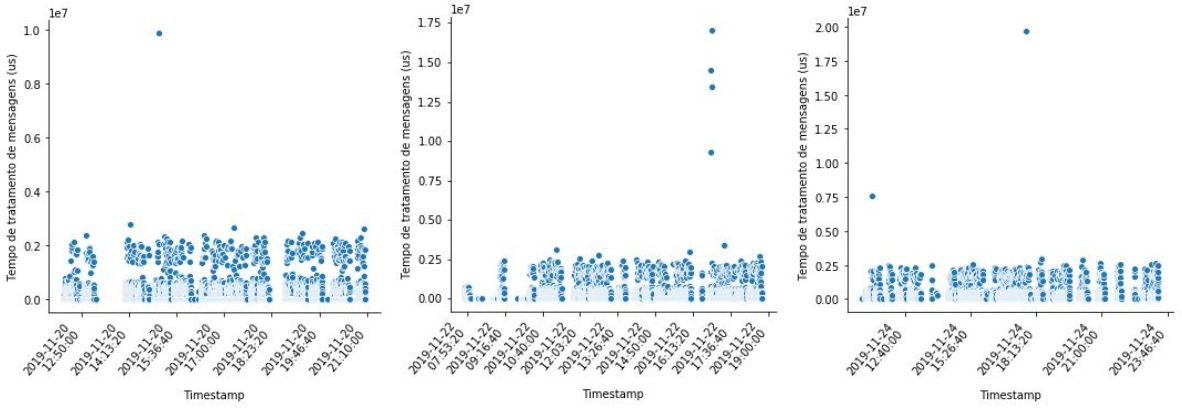
Figura 35 - Tempo para o tratamento de mensagens para o Cenário 1 (*single board* + C).



(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

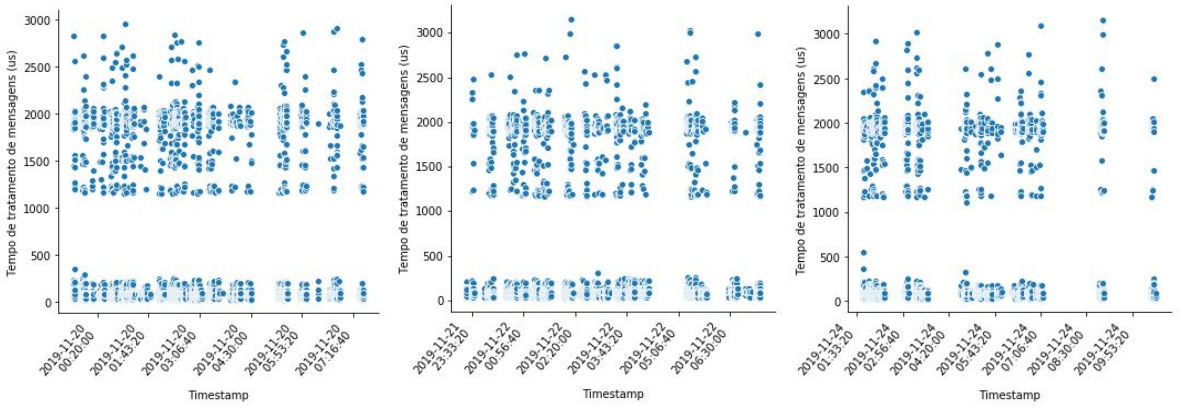
Figura 36 - Tempo para o tratamento de mensagens para o Cenário 2 (*single board* + Python).



(a) (b) (c)

Fonte: Elaborada pela autora (2019).
Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Figura 37 - Tempo para o tratamento de mensagens para o Cenário 3 (PC + C).

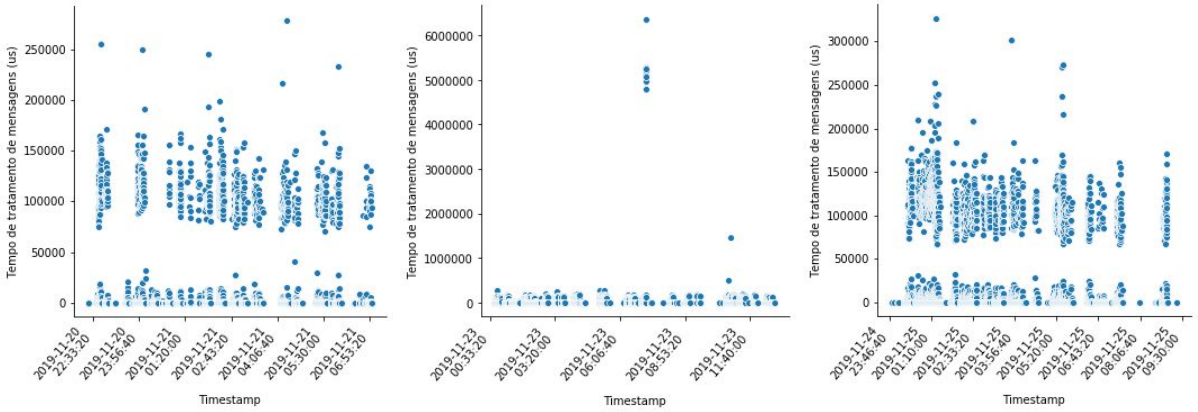


(a) (b) (c)

Fonte: Elaborada pela autora (2019).

Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Figura 38 - Tempo para o tratamento de mensagens para o Cenário 4 (PC + Python).

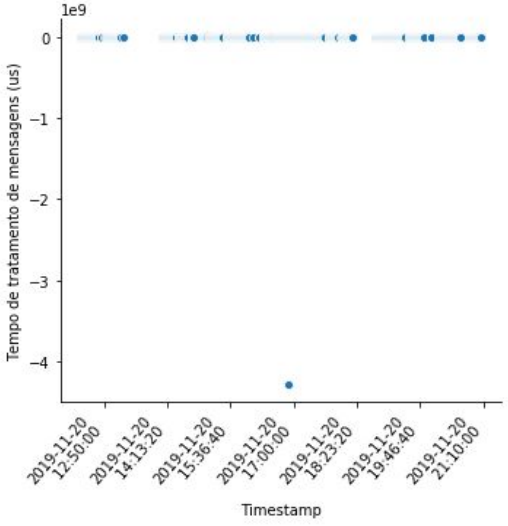


(a) (b) (c)

Fonte: Elaborada pela autora (2019).

Legenda: (a): Captura 1, (b) Captura 2, (c) Captura 3.

Figura 39 - Tempo para o tratamento de mensagens com valores negativos.



Fonte: Elaborada pela autora (2019).

Os próximos passos realizados foram a verificação da normalidade das amostras menores geradas a partir do conjunto de dados de cada cenário; a determinação do parâmetro a ser estimado; o cálculo da distribuição de cada cenário, de acordo com os parâmetros calculados para as amostras menores; e a alteração do comprimento das distribuições de alguns dos cenários, de maneira que todos tivessem o mesmo tamanho.

Os resultados da verificação de normalidade para as amostras menores podem ser vistos na Tabela 16. Para o teste de normalidade, mais uma vez foi empregado o método de Anderson-Darling com nível de significância de 5%. Conforme pode ser visto, nenhuma das distribuições foi considerada normal. Desta forma, escolheu-se utilizar a mediana como parâmetro a ser estimado para a população.

Tabela 16 - Normalidade das amostras para o tempo de tratamento das mensagens.

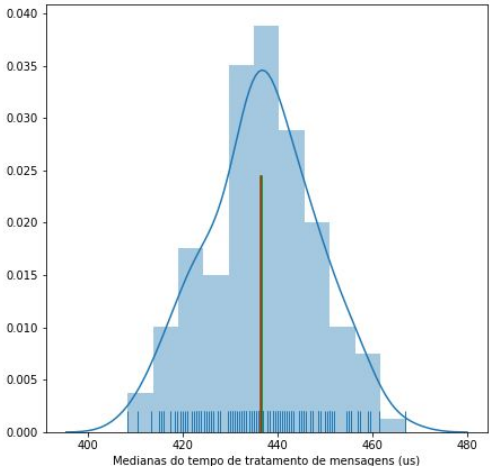
Cenário	**Métrica**	**Normal**	**Não normal**
1	Tempo para tratamento de mensagens	0	454
2	Tempo para tratamento de mensagens	0	150
3	Tempo para tratamento de mensagens	0	199
4	Tempo para tratamento de mensagens	0	464

Fonte: Elaborada pela autora (2019).

Com relação ao tamanho das distribuições geradas para cada cenário, o menor deles foi de 150. Assim, todas as distribuições foram ajustadas para possuírem 150 elementos. Nas Figuras 40, 41, 42 e 43 é possível observar a distribuição das medianas para cada cenário. O eixo horizontal representa os valores de mediana, o eixo vertical demonstra uma densidade para os valores, a linha vermelha mostra a mediana da distribuição e a linha verde apresenta a média.

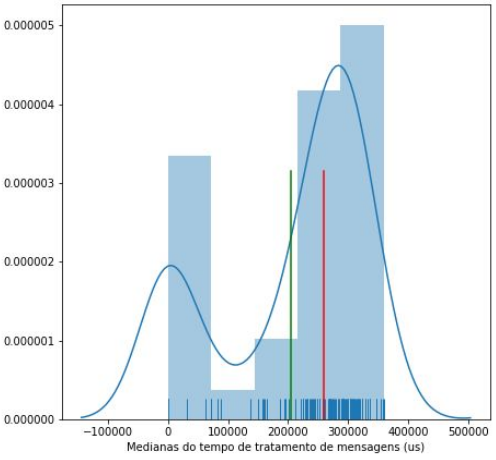
Conforme pode-se observar, as distribuições geradas para os cenários em que o código em C foi executando aproximam-se mais de uma normal, embora ainda não seja possível afirmar que elas sejam normais. Já as distribuições geradas quando o Coletor em Python foi executado mostram-se não simétricas.

Figura 40 - Distribuição de medianas sobre o tempo de tratamento de mensagens, Cenário 1 (*single board* + C).



Fonte: Elaborada pela autora (2019).

Figura 41 - Distribuição de medianas sobre o tempo de tratamento de mensagens, Cenário 2 (*single board* + Python).

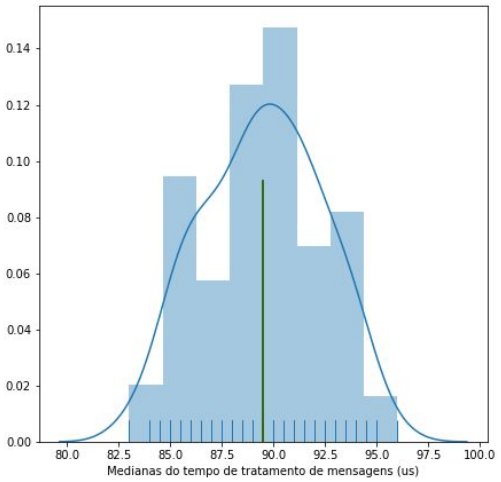


Fonte: Elaborada pela autora (2019).

A próxima etapa consistiu na verificação de normalidade das distribuições calculadas para cada cenário. Mais uma vez, o teste empregado foi o Anderson-Darling, com um nível de significância de 5%. A hipótese nula afirma que os dados possuem uma distribuição normal, ao passo que a hipótese alternativa afirma que eles não possuem. Como resultados, apenas a distribuição do Cenário 1 foi considerada normal, com 95% de confiança. Tendo em vista que o restante das distribuições não apresentou-se como gaussiana; que deseja-se comparar as distribuições entre si; e que os métodos não paramétricos empregados não fazem suposições sobre a distribuição dos dados (embora a aplicação de métodos não paramétricos sobre distribuições paramétricas possa gerar um resultado pior do que a aplicação de técnicas paramétricas); optou-se por aplicar sobre todas as distribuições as mesmas técnicas não

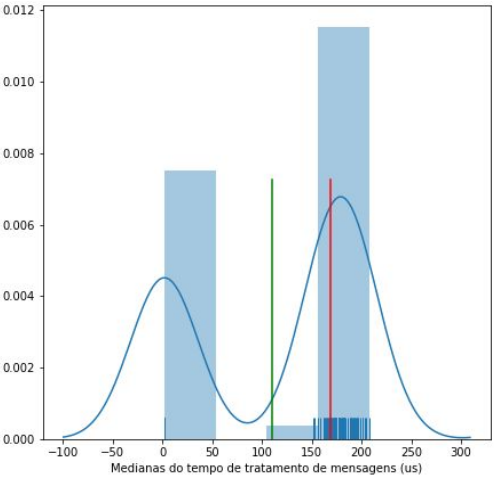
paramétricas aplicadas sobre as outras métricas anteriormente analisadas (uso de processador e memória).

Figura 42 - Distribuição de medianas sobre o tempo de tratamento de mensagens, Cenário 3 (PC + C).



Fonte: Elaborada pela autora (2019).

Figura 43 - Distribuição de medianas sobre o tempo de tratamento de mensagens, Cenário 4 (PC + Python).



Fonte: Elaborada pela autora (2019).

O primeiro método empregado foi o Kruskal-Wallis, com 95% de confiança, para se verificar a existência de diferenças significativas entre as distribuições. De acordo com os resultados, rejeitou-se a hipótese nula, indicando que os dados apresentam diferenças significativas. Então, empregou-se o teste de Dunn, com significância de 5%, para se verificar quais distribuições possuíam diferenças significativas entre si. O resultado desse teste é mostrado na Figura 44.

Figura 44 - Teste de Dunn para o tempo de tratamento de mensagens.

	1	2	3	4
1	-1.000000e+00	2.632447e-02	2.950369e-34	3.674872e-28
2	2.632447e-02	-1.000000e+00	3.569693e-47	6.300884e-40
3	2.950369e-34	3.569693e-47	-1.000000e+00	2.298596e-01
4	3.674872e-28	6.300884e-40	2.298596e-01	-1.000000e+00

Fonte: Elaborada pela autora (2019).

O *data frame* retornado contém o valor de p para todas as comparações entre linhas e colunas. Conforme pode-se observar, quase todos os valores de p são menores do que alfa ($p < 0.05$). Entretanto, na comparação entre as distribuições 3 e 4, o valor de p mostrou-se maior que 0.05 ($0.2 > 0.05$). Dessa forma, pôde-se sugerir, com 95% de confiança, que todos os conjuntos de dados, exceto os conjuntos 3 e 4, possuem diferenças significativas entre si. Em outras palavras, os tempos de tratamento de mensagens foram semelhantes para os cenários 3 e 4, mas foram diferentes considerando-se as outras comparações.

Por último, realizou-se o cálculo dos intervalos de confiança para a mediana das distribuições de cada cenário, aplicando-se a técnica de *Bootstrap*, com 95% de confiança. Os resultados podem ser observados na Tabela 17.

Tabela 17 - Intervalos de confiança para o tempo de tratamento de mensagens dos quatro cenários, em microssegundos.

Cenário	Limite inferior	Mediana amostral	Limite Superior
1	434,25	436,5	438
2	244602	258678,75	278251
3	89	89,5	90
4	164,5	169	176

Fonte: Elaborada pela autora (2019).

5.4 Análise sobre as mensagens recebidas

As métricas sobre as mensagens recebidas fornecem indicativos sobre os serviços fornecidos pelos sistemas coletores de mensagens ADS-B. Entretanto, essas métricas também dependem de características que não puderam ser controladas durante os experimentos, como a densidade do tráfego aéreo no local, que pode fazer com que, em determinados momentos, tenha-se um grande volume de mensagens e, em outros, pouquíssimas mensagens sejam

captadas. Outra questão importante é que não foi possível executar dois coletores ao mesmo tempo, ou seja, cada experimento teve acesso a uma população diferente. Dessa maneira, as estatísticas apresentadas nesta seção fornecem uma visão geral sobre a carga de trabalho a que cada cenário teve acesso, considerando-os independentes uns dos outros.

De acordo com a Tabela 18, pode-se observar que o cenário que teve maior tempo de captura foi o 2, com 32 horas e 51 minutos. Apesar disso, este cenário também foi o que teve acesso ao menor número de mensagens. Além disso, pode-se perceber que o cenário que teve acesso à maior quantidade de mensagens foi o primeiro, com 81847 mensagens recebidas. Assim, pode-se notar uma grande diferença entre as cargas de trabalho aplicadas aos cenários 1 e 2, com relação à quantidade de mensagens. Ou seja, a plataforma *single board* executando o Coletor em C teve acesso a uma maior quantidade de mensagens recebidas do que a mesma plataforma, executando o Coletor em Python.

Embora não se possa determinar o desempenho desses cenários com relação a essa métrica, pois eles não tiveram acesso à mesma população, pode-se investigar o relacionamento entre ela e as outras métricas analisadas. Por exemplo, foi visto que em alguns experimentos, o Cenário 2 levou alguns segundos para tratar as mensagens recebidas. Esse elevado tempo pode ter feito o Coletor em Python perder algumas mensagens.

Tabela 18 - Quantidades totais de mensagens.

Cenário	**Duração (Horas: Minutos)**	**Total de mensagens recebidas**	**Total de mensagens ADS-B**	**Total de mensagens ADS-B decodificadas**	**Total de mensagens ADS-B não decodificadas**
1	25:59	81847	54552	48008	6544
2	32:51	18116	11564	10331	1233
3	25:36	38596	23885	21672	2213
4	30:22	55780	29753	26738	3015

Fonte: Elaborada pela autora (2019).

Por sua vez, ao se observar a Tabela 19, pode-se notar que a quantidade de mensagens recebidas por minuto reflete o total de mensagens recebidas ao longo de todos os experimentos (Tabela 18). Entretanto, a possibilidade de os coletores terem passado certo tempo sem receber nenhuma mensagem e, em outros momentos, terem recebido uma grande quantidade, leva à necessidade de análises mais detalhadas, que observem intervalos menores de captura.

Tabela 19 - Taxas de mensagens.

Cenário	Mensagens recebidas por minuto	Mensagens ADS-B recebidas por minuto	Mensagens decodificadas por minuto	Mensagens ADS-B pelo total de mensagens	Mensagens decodificadas pelo total de mensagens ADS-B
1	52,49967928	34,99166132	30,79409878	66,65 %	88 %
2	9,191273465	5,867072552	5,241501776	63,83 %	89,34 %
3	25,12760417	15,55013021	14,109375	61,88 %	90,73 %
4	30,61470911	16,3298573	14,67508233	53,34 %	89,87 %

Fonte: Elaborada pela autora (2019).

Todavia, ainda na Tabela 19, é possível perceber que, mesmo tendo acesso a uma quantidade de mensagens diferentes, todos os cenários possuem uma taxa semelhante de mensagens ADS-B recebidas em relação ao total de mensagens. Isso significa que, mesmo para conjuntos de diferentes tamanhos, a proporção de mensagens ADS-B dentro deles é aproximada. Além disso, observa-se que as taxas de mensagens que podem ser decodificadas em relação ao total de mensagens ADS-B recebidas também são semelhantes entre os cenários.

Com essas informações, pôde-se verificar que, independentemente das populações às quais tiveram acesso, todos os cenários apresentaram desempenhos semelhantes com relação à quantidade de mensagens que conseguiram decodificar. Além disso, para as populações consideradas, as quantidades de mensagens ADS-B presentes, independente das quantidades totais de mensagens recebidas, também foram aproximadas. Isso pode sugerir que tanto a plataforma de mais baixo custo, como a de propósito geral, apresentaram desempenhos semelhantes.

6 CONSIDERAÇÕES FINAIS

Neste trabalho realizou-se a implementação e análise de desempenho de um sistema *single board* para monitoramento aéreo, baseado na tecnologia ADS-B. Um *software* foi desenvolvido - Coletor em C - e outro foi adaptado - Coletor em Python - para que ambos realizassem as atividades de decodificação de mensagens ADS-B, armazenamento em um banco dados, leitura do banco de dados e envio das informações para um servidor remoto. Além disso, empregou-se uma plataforma *single board* e outra de propósito geral para se comparar o desempenho de *hardware* com diferentes recursos computacionais para as atividades de monitoramento aéreo.

Para isso, sete métricas foram empregadas: o uso de processador, o uso de memória, o tempo para o tratamento das mensagens, a quantidade total de mensagens recebidas, a proporção destas que eram do tipo ADS-B, a total destas que puderam ser decodificadas e a taxa de mensagens recebidas por minuto.

De acordo com o uso de processador, pôde-se verificar que os cenários apresentaram comportamentos de uso diferentes. Aqueles em que o Coletor em Python foi executado exigiram mais consumo de processador ao longo do tempo; ao passo que os cenários que empregaram o Coletor em C passaram a exigir menos processador com o decorrer do tempo. Além disso, o cenário que chegou à maior taxa de consumo de processador foi o 2 (Orange Pi e Coletor em Python), com um intervalo de confiança de 34,13725 <= 34,41418 <= 34,71531 para a mediana estimada.

Com relação ao consumo de memória, todos os cenários apresentaram um comportamento crescente ao longo do tempo. Apesar disso, apenas os cenários 3 e 4 mostraram distribuições sem diferenças significativas, de acordo com os testes de hipótese aplicados. Ademais, novamente cenários em que o Coletor em Python foi aplicado exigiram mais recursos computacionais.

No que diz respeito aos tempos para tratamento das mensagens, observou-se que os cenários em que o Coletor em C foi aplicado exigiram menos tempo do que aqueles em os códigos em Python foram usados. Além disso, notou-se que o Cenário 2 apresentou os piores resultados, possuindo amostras em que o tempo para tratamento de uma mensagem levou alguns segundos. É importante destacar que existem diferenças nos valores entre os intervalos de confiança calculados para essa métrica e as observações a partir dos gráficos apresentados para a métrica no decorrer do tempo (Figuras 36, 37, 38 e 39). Isso pode ser consequência dos

tratamentos aplicados sobre os dados, antes de os intervalos de confiança serem calculados. Apesar disso, em ambas as ferramentas (intervalos de confiança e gráficos), as relações denotadas neste parágrafo são preservadas.

Por fim, levando-se em consideração as métricas calculadas sobre as mensagens recebidas, observou-se que os cenários tiveram acesso a quantidades diferentes de mensagens. Essa diferença se manifesta na taxa de mensagens recebidas por minuto. Entretanto, foi possível observar que as taxas de mensagens ADS-B presentes nos conjuntos de mensagens aos quais os cenários tiveram acesso foram próximas umas das outras. Ademais, notou-se que as taxas de mensagens decodificadas em relação aos totais de mensagens ADS-B recebidas também foram aproximadas, indicando um desempenho semelhante entre os quatro cenários com respeito a essa métrica.

Portanto, pode-se concluir que mesmo plataformas de *hardware* com menores poderes computacionais e mais baixos custos, como a plataforma *single board* aplicada neste trabalho, podem ser empregadas para atividades de monitoramento aéreo, apresentando resultados tão bons como os de um *hardware* com maiores poderes computacionais, como o computador pessoal adotado. Apesar disso, deve-se levar em consideração que, conforme a demanda por recursos aumenta, o desempenho do sistema pode piorar. Isso foi o que possivelmente ocorreu com o Cenário 2, em que o Coletor em Python foi executado na plataforma *single board*, exibindo as maiores taxas de uso de processador; as maiores taxas de uso de memória quando comparado ao Cenário 1, que também empregou a plataforma *single board*; e os maiores tempos para o tratamento das mensagens.

Como trabalhos futuros, propõe-se a realização de experimentos em que todos os cenários tenham acesso à mesma população, de maneira que possa-se avaliar outros aspectos dos sistemas utilizados. Uma abordagem para isso seria empregar-se dados, já coletados, em uma simulação, em que os mesmos dados seriam enviados, garantindo que todos os cenários tivessem acesso à mesma população, assim como feito em Abdulaziz et al. (2015). Também, pode-se aplicar um conjunto mais adequado de métodos estatísticos para a análise dos dados gerados, de forma a se conseguir mais informações úteis sobre eles. Além disso, métricas que levem em consideração outros aspectos dos sistemas também podem ser adotadas, como por exemplo, as que avaliam a comunicação com o servidor remoto. Por fim, pode-se realizar a análise do consumo de energia pelo sistema desenvolvido e a sua adaptação para ser capaz de

se comunicar com qualquer tipo de equipamento de recepção, criando-se, por exemplo, um arquivo que mapeia o formato de mensagens para o tipo de receptor.

REFERÊNCIAS

ABDULAZIZ, Abdulrazaq et al. Optimum receiver for decoding automatic dependent surveillance broadcast (ADS-B) signals. **American Journal of Signal Processing**, v. 5, n. 2, p. 23-31, 2015.

BROWNLEE, Jason. A Gentle Introduction to Normality Tests in Python. *In*: BROWNLEE, Jason. **A Gentle Introduction to Normality Tests in Python**. 2018. Disponível em: https://machinelearningmastery.com/a-gentle-introduction-to-normality-tests-in-python/. Acesso em: 29 nov. 2019.

CAÇÃO, Rosário. **Testes estatísticos**: testes paramétricos e não paramétricos. [S. l.], 2010. 43 slides. Disponível em:https://pt.slideshare.net/rosariocacao/testes-parametricos-e-nao-para metricos-3396639. Acesso em: 29 nov. 2019.

A SIMPLE ADS-B Decoder. [S. l.], 2010. Disponível em: http://rxcontrol.free.fr/PicADSB/index.html. Acesso em: 14 nov. 2019.

FEITOSA, Antônio Guilherme Estevão Soares. **Coletor de dados para monitoramento aéreo usando ADS-B e mini PC Android.** 2016. 37 f. TCC (graduação em Sistemas de Informação) - Universidade Federal do Ceará, Campus Quixadá, Quixadá, CE, 2016. Disponível em: http://www.repositoriobib.ufc.br/000028/00002825.pdf. Acesso em: 2 dez. 2019.

GUIMARÃES, Paulo Ricardo B. **Estatística não paramétrica**. [Paraná], 2018. Disponível em: https://docs.ufpr.br/~lucambio/CE050/20182S/CE050.html. Acesso em: 28 nov. 2019.

JAIN, Raj. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. [New York]: John Wiley & Sons, 1990.

JOHNSTON, Steven J. et al. Commodity single board computer clusters and their applications. **Future Generation Computer Systems**, v. 89, p. 201-212, 2018.

JUNZI, Sun. **The 1090 MHz Riddle:** An open-access book about decoding Mode-S and ADS-B data. [S. l.], 2018. Disponível em: https://mode-s.org/decode/book-the_1090mhz_riddle-junzi_sun.pdf. Acesso em: 2 dez. 2019.

KREIBICH, Jay. **Using SQLite**. [S. l.]: O'Reilly Media, 2010.

ANTENI.NET LTD. **MicroADS-B**. 2012. Disponível em: http://www.anteni.net/adsb/index.html#!/microADSB-USB-receiver/p/15504142/category=3647494. Acesso em: 2 dez. 2019.

MONTGOMERY, DOUGLAS C.; RUNGER, GEORGE C. **Estatística aplicada e probabilidade para engenheiros**. 6. ed. Rio de Janeiro: LTC, 2016.

BBC NEWS. **How do you track a plane?**. 17 mar. 2014. Disponível em: https://www.bbc.com/news/world-asia-pacific-26544554. Acesso em: 2 dez. 2019.

ORLOFF, Jeremy; BLOOM, Jonathan. **Bootstrap confidence intervals**. Massachusetts: MIT, 2014. Disponível em: https://ocw.mit.edu/courses/mathematics/18-05-introduction-to-probability-and-statistics-spring-2014/readings/MIT18_05S14_Reading24.pdf. Acesso em: 2 dez. 2019.

PAHLEVY, Reza Noval et al. Nanosatellite ADS-B Receiver Prototype for Commercial Aircraft Detection. *In*: INTERNATIONAL CONFERENCE ON CONTROL, ELECTRONICS, RENEWABLE ENERGY AND COMMUNICATIONS (ICCEREC). Indonesia. **Anais** [...]. Indonesia: IEEE, 2018. p. 6-12.

PARK, Pangun; TOMLIN, Claire. Performance evaluation and optimization of communication infrastructure for the next generation air transportation system. **IEEE Transactions on Parallel and Distributed Systems**, v. 26, n. 4, p. 1106-1116, 2014.

ORANGE PI. **What's Orange Pi Pc Plus？**. China, 2016. Disponível em: http://www.orangepi.org/orangepipcplus/. Acesso em: 2 dez. 2019.

PINHEIRO, Antônio J. et al. Identifying IoT devices and events based on packet length from encrypted traffic. **Computer Communications**, v. 144, p. 8-17, 2019.

QURESHI, Basit; KOUBÂA, Anis. On energy efficiency and performance evaluation of single board computer based clusters: a hadoop case study. **Electronics**, v. 8, n. 2, p. 182, 2019.

SCHÄFER, Matthias et al. Bringing up OpenSky: a large-scale ADS-B sensor network for research. *In*: INTERNATIONAL SYMPOSIUM ON INFORMATION PROCESSING IN SENSOR NETWORKS, 13., 2014. **Proceedings** [...]. Alemanha: IEEE Press, 2014. p. 83-94.

SEBESTA, Robert W. **Conceitos de linguagens de programação.** 9.ed. Bookman, 2009.

SHAH, Purva. **Reducing Air Traffic congestion with Automatic Dependent Surveillance Broadcast (ADS-B)**. 28 fev. 2017. Disponível em:https://www.linkedin.com/pulse/reducing-air-traffic-congestion-automatic-dependent-broadcast-shah/. Acesso em: 2 dez. 2019.

STENBERG, Daniel. **Everything-curl**. [S. l.]: GitBook, 2017.

SU, Xiangjun et al. SDR reception and analysis of civil aviation ADS-B signals. *In*: INTERNATIONAL CONFERENCE OF SAFETY PRODUCE INFORMATIZATION (IICSPI). **Anais** [...]. China: IEEE, 2018. p. 893-896.

VUJOVIĆ, Vladimir; MAKSIMOVIĆ, Mirjana. Raspberry Pi as a sensor web node for home automation. **Computers & Electrical Engineering**, v. 44, p. 153-171, 2015.