CEFET/RJ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

UM SISTEMA PARA DETECÇÃO DE VAZAMENTO DE ÁGUA EM RESIDÊNCIAS BASEADO EM INTERNET DAS COISAS

Luis Barbosa de Assis Junior

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca CEFET/RJ, como parte dos requisitos necessários à obtenção do grau de mestre.

Orientador: Diego Nunes Brandão
Co-orientador: Helga Dolorico Balbi

Rio de Janeiro, Março de 2023

Um Sistema para Detecção de Vazamento de Água em Residências baseado em Internet das Coisas

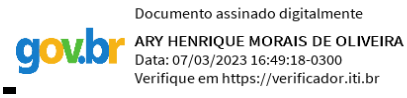
Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação, Centro Federal de Educação Tecnológica Celso Suckow da Fonseca CEFET/RJ, como parte dos requisitos necessários à obtenção do grau de mestre.

Luis Barbosa de Assis Junior

Banca Examinadora:

Presidente, Professor D.Sc. Diego Nunes Brandão (CEFET/RJ)

Professora D.Sc. Helga Dolorico Balbi (CEFET/RJ)



Professor D.Sc. Ary Henrique Morais de Oliveira (UFT)

Professor D.Sc. André Chaves Mendes (IME)

Professor D.Sc. Felipe da Rocha Henriques (CEFET/RJ)

Rio de Janeiro, Março de 2023

Ficha catalográfica elaborada pela Biblioteca Central do CEFET/RJ

A848 Assis Junior, Luis Barbosa de
Um sistema para detecção de vazamento de água em residências baseado em internet das coisas / Luis Barbosa de Assis Junior – 2023.
75f. il (algumas color.). + anexos , enc.

Dissertação (Mestrado). Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, 2023.
Bibliografia : f. 71-75.
Orientador: Diego Nunes Brandão
Coorientadora: Helga Dolorico Balbi

1. Internet das coisas. 2. Dispositivos de internet embarcados. 3. Água – Consumo – Monitoramento. 4. Sensores. 5. Microcontroladores. I. Brandão, Diego Nunes (Orient.). II. Balbi, Helga Dolorico (Co-orient.). III. Título.

CDD 006

Elaborada pelo bibliotecário Leandro Mota de Menezes – CRB/7 nº 5281

DEDICATÓRIA

À minha mãe que se dedicou tanto para que eu pudesse estudar.

AGRADECIMENTOS

..

Agradeço ao professor Diego Nunes Brandão por não me deixar desistir, pela preocupação em contornar minhas dificuldades e guiar meu caminho quando não entendia de nada.
Aos professores do PPCIC pelo interesse nos alunos e por compartilharem seus conhecimentos. Ao empenho dos profissionais do CEFET-RJ.
Ao prof. Felipe Henriques por dedicar o pouco tempo que teve disponível para nortear os trabalhos da dissertação e a profa. Helga Balbi por aceitar o desafio de auxiliar na conclusão deste trabalho. Aos professores Fabrício Lopes e Silva, Cristiano de Souza de Carvalho e Raphael Pereira de Oliveira Guerra que contribuíram na concepção deste trabalho.
Aos funcionários da Biblioteca Central do CEFET-RJ e da Biblioteca Municipal João do Rio, no Rio de Janeiro, pelo empenho em manter esses equipamentos abertos onde pude ter tranquilidade e foco para desenvolver meus estudos e pesquisa.
À CAPES e ao povo brasileiro que me permitiram dedicação exclusiva aos estudos, pois sem ela não seria capaz de trabalhar e também estudar.
Agradeço a minha equipe que se revesa diariamente para me proteger no ir e vir dos caminhos da vida.

RESUMO

Um Sistema para Detecção de Vazamento de Água em Residências baseado em Internet das Coisas

O uso racional da água é essencial para o desenvolvimento e crescimento econômico de uma sociedade. Dados da Organização das Nações Unidas (ONU) indicam que até 2050 ocorrerá uma maior escassez de água devido ao aumento da sua procura em países emergentes, principalmente com o aumento populacional destes locais (ONU, 2018). Um consumo consciente por parte da sociedade pode amenizar tal efeito com a preservação das reservas naturais e aumento da segurança hídrica. Esta premissa serve de motivação para o desenvolvimento deste trabalho, que propõe a criação de um sistema computacional de baixo custo que detecte vazamentos de água em residências utilizando conceitos de Internet of Things (IoT). Tal sistema abrange a captação de dados por meio de sensoriamento, transmissão, processamento e, por fim, uma interface com o usuário. Uma avaliação de uma arquitetura de processamento híbrida (borda e névoa) para o problema de detecção de vazamentos em residências é apresentada. Além disso, duas técnicas conhecidas para o problema - *Consume Non Zero* (CNZ) e *Minimun Night Flow* (MNF) são avaliadas. As principais contribuições consistem nas avaliações das técnicas de detecção de vazamento, na análise de consumo usando dados estatísticos brasileiros e no estudo sobre o impacto do consumo e vazamento na pressão interna de uma residência. Por fim, um algoritmo clássico de aprendizado de máquina foi implementado para estimar o consumo de água e demonstrar a viabilidade do uso desse tipo de algoritmo.

Palavras-chave: Internet das Coisas, Rede de Sensores, Sistemas embarcados, Detecção de vazamento, Água

ABSTRACT

An Internet of Things-based System for Water Leakage Detection in Households

The rational use of water is essential to the development and economic growth of a society. Data from United Nations (UN) indicates that by 2050 will occur a bigger water scarcity due to the increasing demand for it in emerging countries, mainly with the populational growth of these locals (ONU,2018). A conscious consumption by society can softy this effect preserving natural reserves and increasing water security. This premise is used to motivate the development of this work, which proposes the creation of a low-cost computational system that detects water leakages in residences using Internet of Things concepts. The system includes data capture, transmission, processing, and finally displays the information to the user. Empirical assessments of the data transmitting process, as well as a comparison of different processing architectures and leakage detection techniques, are in development to demonstrate proposal viability. Actually, the project presents at first prototype tested in a real environment. An evaluation of a hybrid (edge and fog) processing architecture for the residential leak detection problem is presented. In addition, two known techniques for the problem - *Consume Non-Zero* (CNZ) and *Minimun Night Flow* (MNF) are evaluated. The main contributions consist of evaluations of leak detection techniques, consumption analysis using Brazilian statistical data, and a study on the impact of consumption and leakage on the internal pressure of a residence. Finally, a classical machine learning algorithm was implemented to estimate water consumption and demonstrate the feasibility of using this type of algorithm.

Key-words: IoT, Wireless sensor network, Embedded system

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

BI	Business Intelligence
CNZ	Consume Non Zero
CSV	Comma-separated Values
HTTP	Hypertext Transfer Protocol
INPI	Instituto Nacional Da Propriedade Industrial
IOT	Internet Of Things
IP	Internet Protocol
KNN	K-Nearest Neighbors
LORA	Long Range
LPWAN	Low Power Wide Area Network
MDR	Ministério Do Desenvolvimento Regional
MNF	Minimum Night Flow
MQTT	Message Queue Telemetry Transport
NBIOT	Narrowband Internet Of Things
NFC	Near-Field Communication
NTP	Network Time Protocol
ODS	Objetivos De Desenvolvimento Sustentável
ONU	Organização Das Nações Unidas
OSI	Open System Interconnection
RSSF	Rede De Sensores Sem Fio
RSSI	Received Signal Strength Indication
SNIS	Sistema Nacional De Informações Sanitárias
TCP	Transmission Control Protocol
WDN	Water Distribution Network
WLAN	Wireless Local Area Network
WM-BUS	Wireless Meter Bus
WPAN	Wireless Personal Area Network
WSN	Wireless Sensor Network

Sumário

Capítulo 1

Introdução

A segurança hídrica é importante para o desenvolvimento e manutenção da vida. Segundo a ONU (Organização das Nações Unidas), os próximos 30 anos podem apresentar uma escassez significativa de água devido ao aumento populacional em países emergentes [ONU, 2018]. Tal fenômeno pode afetar significativamente o desenvolvimento econômico e social destes países.

O Brasil é um dos signatários dos Objetivos de Desenvolvimento Sustentável (ODS) da ONU. Tal tratado possui em seu objetivo de número seis, como uma das metas,7 tornar o acesso à água potável e ao saneamento básico universais, além de requerer que seja realizado o tratamento do esgoto evitando assim que ele seja lançado *in natura* nos corpos hídricos. Outra meta importante é aumentar substancialmente a eficiência do uso da água e, neste quesito, o país tem grandes desafios pela frente. O relatório do Ministério do Desenvolvimento Regional (MDR) mostra que 38,5% da água potável produzida no Brasil é perdida na distribuição, seja por vazamentos, aparentes ou não, seja por ligações ilegais [MDR, 2018].

O decreto nº 9.854, de 25 de junho de 2019, instituiu o Plano Nacional de Internet das Coisas que dentre sua atribuições define o que é IoT, quais são as diretrizes de desenvolvimento do tema e ministérios e atores governamentais envolvidos [Brasileiro, 2019]. Enquanto que a Lei nº 14.108, de 16 de dezembro de 2020, conhecida como Lei de Incentivo a IoT, especifica incentivos fiscais para desenvolvimento da política de IoT no Brasil [Brasileiro, 2020]. O Ministério de Ciência e Tecnologia possui uma área dedicada em seu site[1] para todas as ações, políticas e relatórios referentes ao assunto.

1.1 Motivação

Apesar das estimativas da ONU para a escassez hídrica em 2050, o Brasil vem sofrendo inúmeras crises de abastecimento de água desde 2013 [Cruz, 2014]. Naquele ano, cidades como Rio de Janeiro e São Paulo tiveram seus reservatórios de água quase exauridos.

Em 2021, o problema da falta de água, devido à ausência de chuvas, voltou a ocorrer reforçado pelo desmatamento da Floresta Amazônica que afeta a umidade do ar transportado desta região

[1] https://www.gov.br/mcti/pt-br/acompanhe-o-mcti/transformacaodigital/internet-das-coisas

para o sudeste e sul do país. Os chamados "rios voadores" têm impactado na menor precipitação nos reservatórios das hidroelétricas que também são utilizadas para armazenar água para consumo humano [Vieira, 2021]. O reflorestamento, tanto da Floresta Amazônica, quanto das nascentes e matas ciliares, é uma maneira de auxiliar a aumentar o volume de água armazenado. Porém, devido ao longo tempo necessário para reconstituição da vegetação, o abastecimento humano no curto prazo se torna um problema. O consumo consciente é capaz de amenizar esse problema. Nesse sentido, a proposta deste trabalho é explorar o uso de tecnologias de baixo custo para auxiliar na redução de consumo de água residencial na zona urbana dos municípios, através da detecção de vazamentos e análise de hábitos de consumo.

1.2 Pontos Favoráveis

Em muitas cidades, o consumo de água residencial não é mensurado em tempo real. Uma desvantagem se dá na descoberta de vazamentos, principalmente os não aparentes, apenas após a recepção da conta de água, que ocorre com frequência mensal [Fuentes and Mauricio, 2020]. Apesar disso, a detecção de vazamentos no pós-medição ainda traz benefícios que não se limitam somente à economia quantitativa da água, mas também incluem redução dos custos com energia elétrica usada no bombeamento e nos custos de tratamento de esgoto [Britton et al., 2013].

Dentro do contexto da eficiência do uso da água, podem ser verificadas algumas pesquisas desenvolvidas na parte tecnológica que indicam quais ferramentas de *software* e *hardware* estão em maior uso e suas tendências. Ficou constatado, através de busca na base científica Scopus com a expressão "*water and leakage and detection*", o crescente interesse pelo assunto. Na atual década, apesar de decorridos apenas dois anos dela, já existem mais artigos publicados sobre o assunto do que em toda a década de 2000, por exemplo. Os dados referentes ao número de publicações no decorrer dessas e de outras décadas podem ser visualizados na Figura 1.1, que indica também a distribuição das publicações na primeira (azul) a na segunda (laranja) metade de cada década.

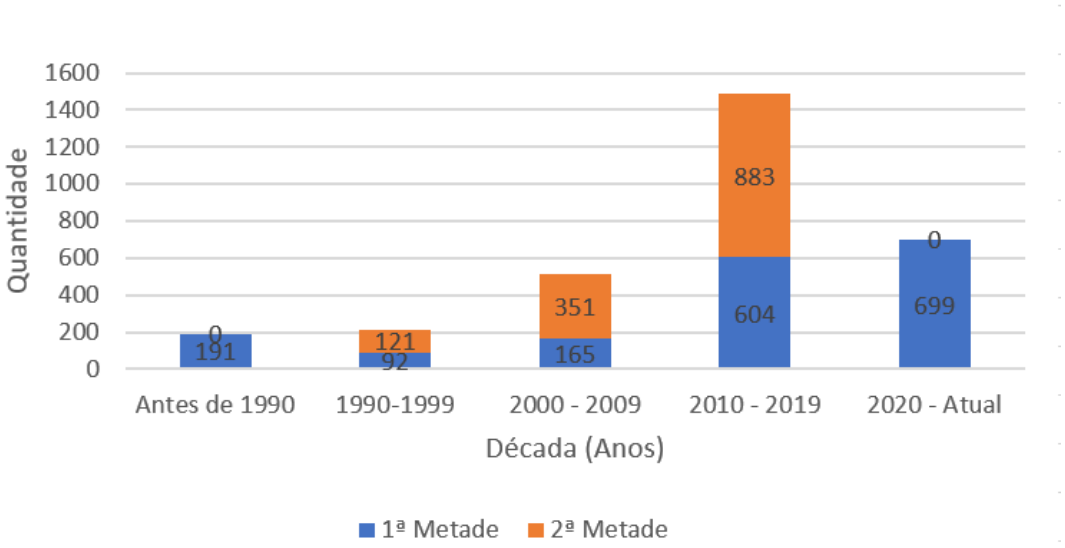


Figura 1.1: Artigos sobre detecção de vazamento de água encontrados na base científica Scopus

O levantamento também demonstrou que inúmeras pesquisas vêm sendo desenvolvidas visando auxiliar na redução do desperdício de água. Boa parte destas pesquisas abordam o problema dos vazamentos em redes de distribuição de água (WDN, do inglês *Water Distribution Network*). No entanto, pesquisas relativas à detecção de vazamento diretamente nos consumidores — ambientes residenciais, escolas, universidades, prédios comerciais e etc. — estão se tornando cada vez mais frequentes.

O desenvolvimento deste trabalho baseou-se em estudos como o de Sánchez et al. [2020], que demonstrou o impacto no consumo de água ao realizar o monitoramento antes e depois da instalação de arejadores [2] nas torneiras em um prédio universitário. Os resultados mostraram uma redução no consumo de água de 45,6% em dias úteis, e de 68,1% nos fins de semana com esta iniciativa. Para mensurar o consumo de água foi instalado um hidrômetro inteligente que realizou medições na entrada de água da cisterna.

Outro trabalho que reforça a importância de desenvolvimento de sistemas de monitoramento de prevenção a vazamentos é o de Britton et al. [2013] que, em ambiente real, acompanhou a substituição de hidrômetros antigos por hidrômetros inteligentes. Estes passaram a avaliar se houve fluxo ininterrupto nas últimas 48 horas, além de registrar por telemetria o consumo a cada hora e enviar os dados via transmissão sem fio para dispositivos da concessionária de água, podendo estes serem um computador de mão ou computador instalado em um automóvel.

De acordo com os resultados do estudo de Britton et al. [2013], 3,8% das casas avaliadas (308 residências) tiveram suspeita de vazamento detectada. A pesquisa consistiu na detecção de vazamentos no pós-medição dos hidrômetros e na avaliação de duas formas de notificação de vazamento

[2] Arejadores são dispositivos capazes direcionar e misturar ar à água, diminuindo o fluxo, porém mantendo a sensação de volume de água para o usuário

aos moradores. Os dados foram estratificados por bairro e valor do m^2 de onde uma amostra de 372 casas foi dividida em dois grupos de maneira aleatória. No grupo A, ao final de 3 meses, o consumo foi reduzido em expressivos 87,68% (de 4009L/h para 494L/h). Outro fato interessante foi o baixo custo para o conserto do vazamento. De forma global, em ambos os grupos, 70% dos vazamentos foram solucionados com investimentos de até BRL R$ 722 (AUD$ 200) enquanto que 50% foram sanados com até BRL R$ 360 (AUD$ 100).

Retratando a realidade brasileira, o trabalho de Dantas and Moraes [2006] visitou prédios residenciais nos bairros da orla da cidade de Salvador, estado da Bahia. Constatou que 238 apartamentos vistoriados (14,22% do total) possuíam algum tipo de vazamento e identificou o vaso sanitário como maior vilão.

1.3 Nicho Escolhido

Ao pesquisar os tipos de consumo de água, descobriu-se por meio do relatório da ANA [2019] que o maior uso se dá na irrigação/agricultura como mostra a Figura 1.2. Porém, segundo o mesmo relatório, os dois maiores municípios brasileiros em volume de extração de água bruta da natureza o fazem para consumo humano urbano conforme Tabela 1.1. Assim, optou-se explorar o eixo de consumo humano urbano.

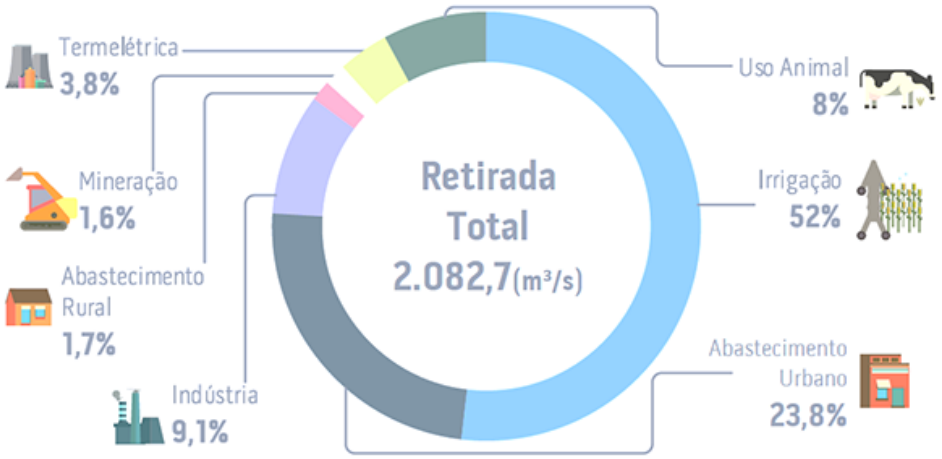


Figura 1.2: Principais usos da água no Brasil. Fonte ANA (2019).

1.3.1 Diferenciais desta Pesquisa

Algumas propostas de detecção de vazamentos nos consumidores visam a instalação de hidrômetros inteligentes (*smart meters*) que medem o consumo e enviam os dados via rede de telefonia móvel para o consumidor ou à concessionária de água [Muhammetoglu et al., 2020]. O estudo de vazamento pós-medição se dá depois do hidrômetro da concessionária de água, ou seja, avalia o

Tabela 1.1: Maiores Vazões Retiradas(m3/s) por Município (2017) – Usos Consumitivos no Brasil

Raking	**UF**	**Município**	**Vazão Retirada (m3/s)**	**Uso Predominante**
1º	SP	São Paulo	46,026	Abastecimento Humano Urbano
2º	RJ	Rio de Janeiro	45,283	Abastecimento Humano Urbano
3º	RS	Uruguaiana	24,405	Irrigação
4º	RS	Sta Vitória do Palmar	24,376	Irrigação
5º	RS	Alegrete	22,03	Irrigação
6º	RS	Itaqui	20,874	Irrigação
7º	BA	Juazeiro	18,261	Irrigação
8º	RS	São Borja	16,771	Irrigação
9º	PE	Petrolina	16,009	Irrigação
10º	RS	Mostardas	15,794	Irrigação

Fonte: ANA(2019)

consumo contabilizado para residência.

Foram utilizados conceitos de Rede de Sensores Sem Fio (RSSF) e IoT a fim de encontrar vazamentos de água em menos tempo e com localização mais precisa ao propor o monitoramento do uso da água por cômodos da residência. Soma-se a isso o estudo dos hábitos de consumo dos residentes para propor mudanças de comportamento, resultando, assim, na redução do consumo. Para alcançar o objetivo foram utilizados sensores de pressão e vazão de baixo custo para medição do consumo de água e detecção de vazamentos.

A opção por incluir sensores, principalmente, após a saída de água da caixa d'água é motivado pela realidade brasileira onde o fornecimento não é constante. Geralmente determinados dias da semana há abastecimento. É comum o uso de cisternas ligadas diretamente na entrada do hidrômetro, logo, não seria possível, em muitos casos, a detecção de vazamento instalando hidrômetros inteligentes.

Os sensores foram ligados a uma placa de prototipagem NodeMCU responsável por recepcionar os dados do sensoriamento, processar e transmitir para a plataforma IoT ThingsBoard onde são armazenados, reprocessados e exibidos para o usuário. Foram exploradas técnicas de detecção de vazamentos conhecidas na literatura: Mínimo Fluxo Noturno e Consumo Não Zero nas Últimas x Horas [Fuentes and Mauricio, 2020; Muhammetoglu et al., 2020], mas também foram geradas contribuições, sendo estas: uso de dados brasileiros de consumo doméstico para criação de regras de associação e estudo do comportamento da pressão durante o consumo.

1.4 Objetivos e estrutura

O objetivo deste trabalho foi criar um protótipo de baixo custo para detecção de vazamento em residência utilizando IoT. Optou-se por explorar processamento em borda (*Edge Computing*) e em névoa (*Fog Computing*) que proporcionam menores latências de resposta e menor custo de

implantação. Estas soluções não necessitam de assinatura de serviços de nuvem (*Cloud Computing*). Para sua viabilidade foi necessária a prospecção de tecnologias de baixo custo, o sensoriamento em ambiente real, a escolha de tecnologias e protocolos de comunicação, processamento dos dados e, por fim, aplicação de algoritmo de aprendizado de máquina.

Este trabalho encontra-se dividido em mais 4 capítulos: o referencial teórico, onde são apresentados alguns conceitos básicos sobre IoT e revisão bibliográfica, é apresentado no Capítulo 2. Em seguida, o Capítulo 3 apresenta a metodologia com as técnicas, ferramentas e conceitos utilizados na solução proposta ao problema. Já o Capítulo 4 apresenta os resultados obtidos aplicando-se a solução proposta em um ambiente residencial. As considerações finais e propostas de continuação do trabalho são apresentadas no Capítulo 5.

Capítulo 2

Referencial teórico

Este capítulo apresenta alguns conceitos básicos que são necessários para compreensão do trabalho desenvolvido: IoT, Arduino, arquitetura de processamento e redes. Além disso, ele apresenta uma revisão da literatura na qual são discutidos os trabalhos encontrados que nortearam o desenvolvimento desta pesquisa. Para facilitar a compreensão e auxiliar na discussão, tais trabalhos foram divididos nos seguintes grupos: artigos sobre sensores, artigos sobre técnicas de detecção de vazamento, artigos sobre análise de pressão, artigos sobre arquitetura de processamento, artigos sobre arquitetura de software e, por fim, artigos sobre técnicas de comunicação/transmissão de dados.

2.1 Conceitos fundamentais

Esta seção apresenta uma breve introdução aos conceitos de sensores, redes sem fio, arquitetura de dispositivos, que são essenciais para a compreensão da ideia de Internet das Coisas.

2.1.1 Internet das Coisas

O termo IoT se refere à interconexão em rede de objetos do cotidiano geralmente equipados com algum tipo de inteligência [Xia et al., 2012]. O advento das plataformas de prototipagem de baixo custo permitiu que a IoT se tornasse cada vez mais popular com aplicações em domótica, agricultura, indústria e muitas outras. Dentre tais plataformas, o Arduino[1] é uma das mais famosas, permitindo o desenvolvimento de soluções de forma simples e com baixo custo econômico [Prasojo et al., 2020].

Apesar das inúmeras vantagens, o Arduino necessita que sejam incorporadas novas placas (*shields*) para que seja possível utilizar novas funções. Alguns submodelos possuem comunicação sem fio embarcada, porém a um custo maior. Por isso, um dispositivo similar e de custo menor foi escolhido para este projeto. O NodeMCU[2] (Figura 2.1) também é uma plataforma de prototipagem que já possui módulos de comunicação embarcados, como módulo WiFi e Bluetooth, garantindo

[1] https://www.arduino.cc/
[2] http://www.espressif.com/

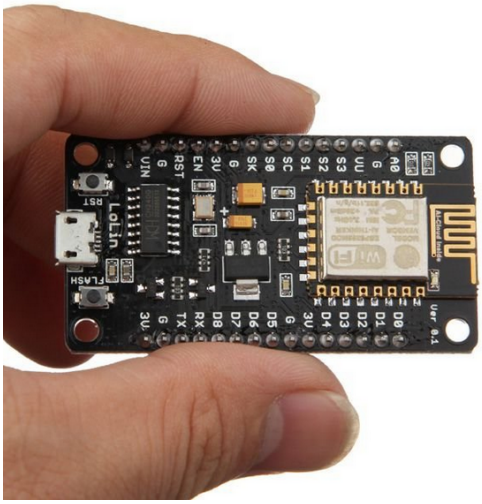


Figura 2.1: NodeMCU [Fonte: br.travelsearchexpert.com]

um menor custo ao projeto a ser desenvolvido. Além disso, ele possui compatibilidade com o ambiente de desenvolvimento do Arduino e mais portas compatíveis com sinais analógicos, facilitando a adição de mais sensores em uma mesma placa [Fuentes and Mauricio, 2020; Deepa et al., 2021].

Outra de suas vantagens é maior capacidade de memória tanto para armazenar instruções e dados quanto para execução de programas mais robustos e complexos. Além destas, o microcontrolador do NodeMCU ESP32 possui maior frequência de processamento A Tabela 2.1 apresenta uma comparação resumida dos dispositivos.

Tabela 2.1: Comparativo Arduino x NodeMCU

Característica	**Arduino Uno**	**NodeMCU ESP32**
Microcontrolador	ATmega328 Single Core 16MHz	LX6 Dual Core 240MHz
Conexão	USB	USB, Bluetooth, Bluetooth Low Energy, WiFi
Portas	20 GPIO sendo: 6 Analógicas 6 PWM	25 GPIO sendo: 18 Analógicas/digitais 16 PWM
Memória	SRAM: 2KB EEPROM: 1KB	RAM: 520 KBytes ROM: 448 KBytes Flash: 4 MB

2.1.2 Sensores

Sensores agregam funcionalidades às plataformas de prototipagem. Estes são dispositivos que respondem a estímulos físicos ou químicos, transformando a ação sofrida em um sinal analógico ou digital, que podem ser transformados em grandeza física para fins de medição ou monitoramento

[Moris, 2001]. Dentre diferentes sensores, algumas de suas funções podem ser citadas, como medição de temperatura, quantidade de CO_2, luminosidade, velocidade, pressão e etc.

Individualmente, cada sensor tem limitação de banda de transmissão, energia, processamento e armazenamento. No entanto, juntos, inúmeros nós sensores coletam dados do meio e os transmitem a um nó coletor, formando uma RSSF (do inglês, Wireless Sensor Network (WSN)) capaz de abranger uma ampla área e gerar grande quantidade de dados. Este nó coletor recebe os dados, transmitindo-os posteriormente ao usuário final ou à aplicação desenvolvida. RSSF são úteis em diversas aplicações tais como: monitoramento do meio ambiente, monitoramento de consumo (de água, de energia, dentre outros), monitoramento de idosos ou pacientes em hospitais, segurança, enxame de drones, robôs cooperativos, sendo essenciais para o desenvolvimento de cidades inteligentes e agricultura de precisão [Patnaik et al., 2020].

2.1.3 Arquiteturas de processamento

Os dados coletados pelos sensores precisam ser processados para serem transformados em informação. De acordo com o local de processamento, pode-se classificar a arquitetura do sistema desenvolvido de três maneiras: em borda (*Edge Computing*), em névoa (*Fog Computing*) ou em nuvem (*Cloud Computing*)[Shi and Dustdar, 2016; Sulieman et al., 2022; Laroui et al., 2021]. A mais conhecida, computação em nuvem, é aquela na qual os dados captados são transmitidos para serviços em servidores de Internet onde são processados.

Na opção de computação em névoa, os dados são processados em dispositivos de borda de rede como roteadores, *gateways* e computadores ligados a essa rede. Neste caso, o objetivo é trazer o processamento para a arquitetura nativa ganhando desempenho ao reduzir a latência no envio dos dados, enviando-os para a nuvem central apenas para cálculos mais custosos [Mohan and Kangasharju, 2017]. Assim, a computação em névoa consiste em adicionar poder computacional próximo aos geradores de dados brutos [Amaxilatis et al., 2020] em uma camada intermediária entre os sensores IoT e os tradicionais *data centers*. Por fim, a arquitetura mais próxima aos nós é a computação de borda, na qual é explorado parte do processamento nos próprios nós sensores [Mohan and Kangasharju, 2017].

2.1.4 Redes de Sensores Sem Fio

Nas RSSF, em inglês WSN, inúmeros nós sensores captam dados que são transmitidos para um ou mais nós sorvedouros que, por sua vez, os repassam ao usuário final ou para ambientes de armazenamento.

As RSSFs são utilizadas em aplicações, tais como: monitoramento do meio ambiente; monitoramento de consumo de água e de energia; monitoramento de pessoas idosas ou pacientes em hospitais;

segurança; navegação de forma autônoma, como ocorre em enxames de drones; e execução de ações de forma coordenada, como as realizadas por robôs cooperativos, dentre outras. Elas são essenciais para o desenvolvimento de cidades inteligentes e agricultura de precisão Patnaik et al. [2020], formando uma infraestrutura de comunicação para tais aplicações.

Essas redes podem utilizar diferentes tecnologias de comunicação. Especificamente para o caso de RSSFs dentro do contexto de IoT, as tecnologias mais utilizadas são as que se enquadram nas classes: Low Power Wide Area Network (LPWAN); Wireless Local Area Network (WLAN); e Wireless Personal Area Network (WPAN)[de Assis Jr et al., 2022].

As redes LPWAN focam na cobertura de grandes áreas com baixa taxa de transmissão e baixo consumo de energia pelos dispositivos. A Long Range (LoRa) é um exemplo de tecnologia desta classe. Já as redes WLAN e WPAN são exemplificadas pelas tecnologias Wi-Fi e Zigbee, respectivamente. Elas se baseiam nos protocolos IEEE 802.11 e IEEE 802.15.4, respectivamente. A tecnologia Zigbee permite a comunicação de diversos sensores e atuadores em redes pessoais com baixas taxas de transmissão, mas com a flexibilidade de implementar diferentes tipos de topologia de rede. Por outro lado, o Wi-Fi é a tecnologia mais utilizada para a criação de redes locais sem fio, presente em *smartphones*, *notebooks*, *tablets* e etc. Outra tecnologia que pode ser classificada como WPAN é a Near-Field Communication (NFC).

LoRa[3] é uma tecnologia de transmissão de dados de baixa largura de banda, variando de 0,3 kbps até 50 kbps, porém com grande alcance. No Brasil, ela está liberada para uso na faixa de 915MHz e é considerada uma tecnologia de radiação restrita [Anatel, 2017].

Ao utilizar uma faixa de frequências abaixo dos giga-hertz (sub-GHz), consegue-se um maior o alcance devido à facilidade de as ondas transporem os obstáculos em comparação aos sinais de alta frequência [Amaxilatis et al., 2020]. Por exemplo, utilizando-se um chip de modelo SX1276 da Semtech[4] operando na frequência de 868MHz, o alcance em ambiente aberto chega a 2,8Km [Khutsoane et al., 2017]. Entre suas vantagens, estão o baixo consumo de energia e o menor custo de implantação se comparado ao padrão ZigBee. Essas características têm atraído desenvolvedores a usarem o LoRa em aplicações IoT.

Um comportamento citado por Bertoleti [2019] é que numa conexão LoRa ponto-a-ponto, se houver três ou mais dispositivos em um enlace, todos receberão as informações transmitidas, pois não há endereçamento de rede em uma conexão LoRa pura, além de ela não oferecer segurança dos dados, ficando a cargo do projetista implementar a criptografia. Para solucionar essa questão criou-se o LoraWan. "LoraWan é um protocolo de rede definido em *software* livre/aberto que permite fazer uma rede completa, com endereçamento de dispositivos, *gateway*, mecanismos de anti-colisão de pacotes, em outra palavras, LoraWan implementa detalhes de funcionamento, segurança, qualidade

[3]https://lora-alliance.org/
[4]https://www.semtech.com/products/wireless-rf/lora-connect/sx1276

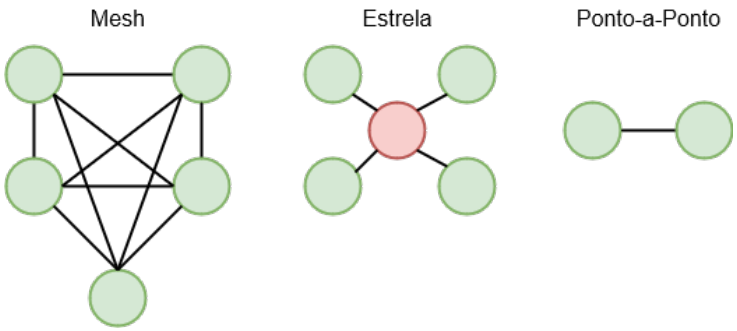


Figura 2.2: Topologias de rede

de serviço e ajuste de potência visando maximizar a duração da bateria" [Bertoleti, 2019].

A tecnologia NFC é a menos utilizada no contexto IoT, pois garante a comunicação em espaços muito curtos, cerca de poucos centímetros (de Assis Jr et al., 2022). Outras tecnologias que podem ser citadas com finalidade de criação de redes IoT são: GSM-GPRS, 4G-LTE (*Long Term Evolution*) e a mais recente, Narrowband Internet of Things (NbIoT) [Muhammetoglu et al., 2020].

Dentre as topologias de rede, podem ser destacadas, conforme mostra a Figura 2.2, as seguintes: estrela, na qual um dispositivo fica encarregado de gerenciar as comunicações recebendo todas as mensagens e as distribuindo; ponto-a-ponto, na qual dois dispositivos se comunicam diretamente; e em malha (*mesh*), na qual existem enlaces de comunicação redundantes entre os diversos nós da rede, possibilitando a adaptação das rotas de acordo com as disponibilidades de caminhos, sendo essa, uma rede descentralizada. Já os protocolos de comunicação servem para dar forma aos dados, adicionando cabeçalhos, capacidade de dividir os dados, segurança, confirmação de entrega, entre outros recursos. Por fim, é importante decidir se a rede considerada em um projeto de comunicações será uma rede síncrona ou assíncrona, definindo em que instantes de tempo serão feitas as comunicações [de Assis Jr et al., 2022].

2.1.5 Middleware de comunicação

Para a transmissão dos dados, é necessário escolher, além da tecnologia e meio físico, protocolos de comunicação. Em redes de maior capacidade, tais como as redes sem fio padrão IEEE 802.11, é comum o uso de protocolos como o Hypertext Transfer Protocol (HTTP) utilizado por Fuentes and Mauricio [2020] na comunicação entre o nó de pré-processamento, que no caso é o Raspberry Pi, e o servidor em nuvem. Porém, em redes de menor capacidade, o uso do HTTP se torna difícil devido ao tamanho do cabeçalho. Por isso, existem protocolos específicos para soluções IoT.

Manter a comunicação sem fio entre os dispositivos mencionados, além de aspectos de segurança e outras funcionalidades em uma RSSF é uma tarefa difícil dada as interações diretamente no *hardware*. Para amenizar a necessidade dessas interações e facilitar o desenvolvimento de sistemas, surge o conceito de *Middleware*. Um *middleware* consiste em um *software* que implementa diversas

funcionalidades de maneira transparente ao usuário. No caso de RSSF, um *middleware* fornece um serviço de troca de mensagens que garante, além da entrega das mensagens entre os participantes da rede, características como a tolerância a falhas, aumento de capacidade no volume de dados trafegados, a transparência de localização, dentre outras [Pradeep and Krishnamoorthy, 2019].

Segundo Van Steen and Tanenbaum [2017], *Middlewares* Orientados a Mensagem (MOMs) garantem uma comunicação persistente assíncrona entre os elementos do sistema, sem que estes elementos (emissor e receptor) estejam necessariamente ativos durante a transmissão das mensagens. Um MOM implementa uma abstração de filas de mensagens que podem ser acessadas via rede. Assim, ele garante que os nós da rede não precisam necessariamente estar o tempo todo conectados a ela, garantindo com isso economia de recursos como bateria. Além disso, os nós da rede não precisam conhecer os endereços uns dos outros, garantindo também economia de memória. Todavia, o MOM necessita de um elemento central que é responsável por gerenciar as filas de mensagens. Tal limitação pode ser superada com a replicação de tal elemento [de Assis Jr et al., 2022].

Diferentes políticas podem ser implementadas para garantir um melhor gerenciamento das filas de mensagens e do acesso as essas mensagens. Em uma das mais utilizadas, a *publish-subscriber*, as mensagens são associadas a um determinado tópico, sendo lidas somente por assinantes que optaram por receber mensagens dos tópicos desejados. Nesse tipo de política, um terceiro elemento é necessário, o *broker*. Ele é um servidor responsável por armazenar as mensagens enviadas, além de filtrá-las, associando cada mensagem a um tópico e garantindo que somente os assinantes daquele tópico tenham acesso a ele [de Assis Jr et al., 2022].

MQTT

Neste contexto, surge o protocolo de comunicação MQTT[5], que funciona com dois tipos de atores: clientes e o *broker*. Os clientes que enviam mensagens são os já mencionados *publishers* e os que recebem são os *subscribers*. Os tópicos são como se fossem canais onde as mensagens são publicadas. É importante salientar que um cliente pode ser, simultaneamente, *publisher* e *subscriber*, além de ser permitido estar inscrito e um ou mais tópicos [Kodali and Soratkal, 2017]. A Figura 2.3 sintetiza a comunicação entre clientes e *broker*.

Na Figura 2.3 pode-se ver o dispositivo 2 que publica informações no Tópico 1 e recebe informações do Tópico 2. Também há dispositivos que só publicam e outro que só consome, que são os dispositivos 3 e 4 respectivamente. O *broker* gerencia o recebimento e envio das mensagens para todos conectados a ele.

O MQTT foi desenvolvido na década de 1990 pela IBM para gerenciar a transmissão de dados assíncronos através de redes intermitentes (Silva Júnior, 2017). Suas principais vantagens são: cli-

[5]http://mqtt.org/

entes MQTT são muito pequenos, requerendo poucos espaço na memória dos microcontroladores, seus cabeçalhos são pequenos e otimizados para redes com pouca largura de banda, suporta protocolos de segurança e é escalável permitindo adição de novos nós com facilidade. Tanto Fuentes and Mauricio [2020] quanto Silva Júnior [2017] implementaram este protocolo em seus trabalhos.

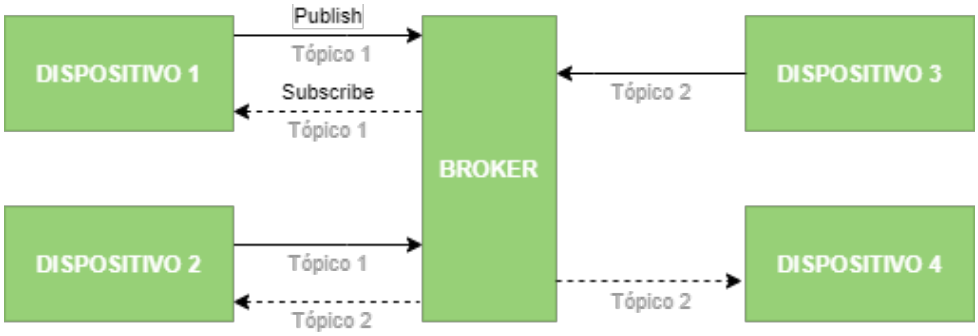


Figura 2.3: Diagrama de fluxo da comunicação cliente-broker no MQTT

A sessão MQTT é estabelecida após quatro passos serem executados de forma bem sucedida: conexão, autenticação, comunicação e desconexão. O cliente se conecta ao *broker* via Transmission Control Protocol (TCP)/Internet Protocol (IP) por meio da porta 1883 (*default*) ou personalizada utilizando SSL/TLS para segurança. Ele, então, valida o servidor usando o certificado provido pelo servidor e, por fim, o cliente fornece seu certificado ao *broker* para autenticação (Wukkadada et al., 2018).

Apache Kafka

Ainda no contexto de *publisher-subscriber*, é apresentado o Apache Kafka[6], que foi concebido inicialmente para atuar como uma fila de mensagens. Sua aplicação é voltada para o contexto de sistemas distribuídos, permitindo que mais de um sistema acesse suas filas para processar os dados. É escalável e permite a distribuição de carga dos processos para diferentes computadores. O Kafka precisa de um servidor Zookeeper[7], que é um serviço centralizado para manutenção de informações de configuração, nomes, fornecimento de sincronização distribuída e fornecimento de serviços de grupo. Os serviços Kafka permitem que sejam desenvolvidos aplicativos em diversas linguagens como Java e Python.

2.2 Trabalhos relacionados

Nesta seção, são apresentados alguns dos trabalhos estudados ao longo da realização desta pesquisa, assim como o processo de busca por tais artigos. Pelo foco de pesquisa da comunidade científica ser em Water Distribution Network (WDN), houve dificuldade de se encontrar trabalhos

[6] www.confluent.io/lp/apache-kafka
[7] https://zookeeper.apache.org/

cujo objetivo fosse detecção de vazamento de água em residências. Por este motivo, pesquisas exploratórias no Google Acadêmico foram realizadas para conhecer palavras-chave mais relacionadas ao problema. Foram utilizadas *strings* simples, tais como: *water leakage*, *vazamento de água* e *water monitoring*.

A partir da leitura de artigos que serviram para fundamentação, novos termos foram identificados na seção de palavras-chave (*keywords*) de cada artigo. Para a realização de nova pesquisa, foi escolhido o site Periódicos CAPES[8] devido ao acesso irrestrito a diversas bases científicas. A *string* de busca com melhores resultados foi: *smart water monitoring leak iot* resultando em 130 artigos, em novembro de 2020, com 4 artigos de interesse relevantes e um deles abrangendo alguns aspectos deste projeto. A partir deste trabalho, foi utilizada a técnica de *snowballing* para seleção de novos artigos de interesse. Está técnica refere-se à leitura de artigos e fontes citadas no artigo de interesse.

Strings mais elaboradas tal como: *smart and ("meter" or "metering" or "water")* resultaram em poucos artigos, mesmo realizando buscas diretamente em redes como Scopus e IEEE. Também foram adicionados artigos ao conjunto de trabalhos selecionados conforme alguns deles eram considerados potencialmente interessantes. Conforme mencionado anteriormente, os artigos avaliados foram classificados em 5 grupos que são descritos a seguir.

2.2.1 Sensores

Nos trabalhos encontrados, foram usados sensores de umidade, temperatura, som e imagem, porém, os mais comumente encontrados foram os de pressão e vazão. Este último possui alguns diferentes mecanismos. Ambos serão melhores abordados a seguir.

Sensores de Vazão

Foram encontrados alguns tipos de medidores de vazão: ultrassom, multijatos, magnético e de efeito Hall (Tabela 2.2). Enquanto alguns trabalhos focam na medição por hidrômetros, inteligentes ou não, outros vão ao encontro com uso de IoT com RSSF. Nota-se que soluções mais precisas também têm um custo mais elevado. Os artigos que mencionam o uso dos diferentes tipos de sensores estão resumidos na Tabela 2.2, bem como o modelo/fabricante do sensor, a precisão, a sensibilidade, o tipo e o custo em Reais (BRL) e dólares americanos (USD).

Alarefi and Walker [2017] sentiram a necessidade de utilizar um filtro para remover ruídos dos sinais captados pelo sensor de vazão ultrassônico e usaram um hidrogerador na tubulação para ajudar na alimentação de energia do sensor de fluxo, aumentando a vida útil da bateria.

Um ponto ressaltado por Sánchez et al. [2015] é a dificuldade de se detectar pequenos vazamentos. Para amenizar o problema, utilizaram uma válvula de impulsos hidráulicos junto aos hidrômetros.

[8] periodicos.capes.gov.br

Tabela 2.2: Sensores de vazão utilizados nos artigos

Artigo	Fabricante Modelo	Precisão	Sensibilidade	Tipo	Valor
Fuentes and Mauricio (2020)	Seeed G3&4	4,63%*	1-60L/min	Hall	BRL 84,90 USD 15,16
Romeiro (2019), Silva Junior (2017)	Seeed YF- S201	5,00%			BRL 50,00 USD 8,92
Britton et al. (2013)	Elster V100	2,00%	3L/h	-	BRL 369,60 USD 66.00
Muhammetoglu et al. (2020)	MANAS Smart meter	-	-	-	-
Coronado (2019)	Endress + Huaser (Proline Promag 10P)	± 0,5%	4L/min a 9,6 x 10^5 L/h	Magnético	BRL 23.828,55 USD 4232.20
Alarefi (2017)	Titan Atrato	± 1,5%	2mL/min 20 L/min	Ultrassom	BRL 5.088,76 USD 908.70
Pietrosanto et al. (2021)	Maddalena DS TRP	± 0,5%*	7,8L/h*	Multijatos	BRL 1.336,44 USD 238.64
	Maddalena MVM	± 0,5%*	2,8L/h*	Volumétrico	BRL 397,70 USD 71.00
	Ultrassônico	± 0,5%*	4L/h*	Ultrassom	-

*Resultados de experimentos dos autores, demais características técnicas foram retiradas dos sites dos fabricantes. Cotação: USD $1 = R$ 5,60 em 04/01/2022

Essa válvula muda o regime de vazão para um que esteja dentro da sensibilidade do hidrômetro. Para este fim optaram pelo uso do modelo UFR fabricada em Israel pela A.R.I. Flow Control Accessories Ltd, igual a da Figura 2.4.

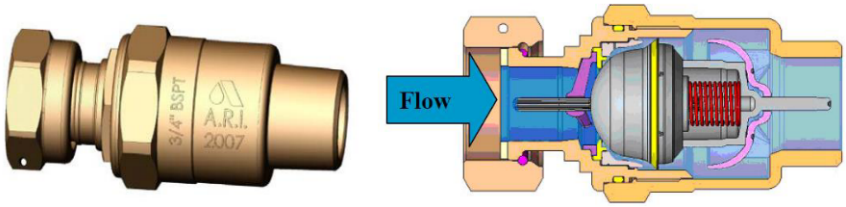


Figura 2.4: Válvula de impulsos hidráulicos A.R.I. UFR [Sánchez et al., 2015]

Pequenos vazamentos, como gotejar ou filete na torneira, consomem a água na tubulação lentamente, não sendo fortes o suficiente para girar as pás internas que medem a vazão de água no hidrômetro. Num sistema com a válvula e sem vazamento, a diferença de pressão antes e depois do êmbolo é zero. O consumo faz a pressão e quantidade de água no pós-válvula reduzir, porém, o êmbolo só se move quando a diferença é significativa. Ao se mover, permite a passagem de água, reequilibrando a pressão na tubulação de maneira rápida. A rapidez com que a água preenche a tubulação faz aumentar a vazão, permitindo ser detectada pelo hidrômetro. Durante o consumo, o embolo se move permitindo a passagem da água.

Sensores de Pressão

Apesar de usar o sensor de vazão para detecção de vazamento, o trabalho de Gama-Moreno et al. [2010] se destacou pela forma do uso do sensor de pressão para o monitoramento do reservatório de água. Os autores mediram a pressão por meio de um sensor instalado em uma câmara de ar de um tubo em U ligado ao reservatório. O nível de água é medido pelo nível de pressão na câmara. Nesta solução, mostrada na Figura 2.5, o sensor não tem contato com o líquido, contribuindo para uma vida útil maior. Por outro lado, a solução de Sánchez et al. [2015] utilizou uma sonda de nível submersa (Solinst Levelogger 3001) que registra, a cada instante, o nível de água no reservatório e envia a leitura por telemetria.

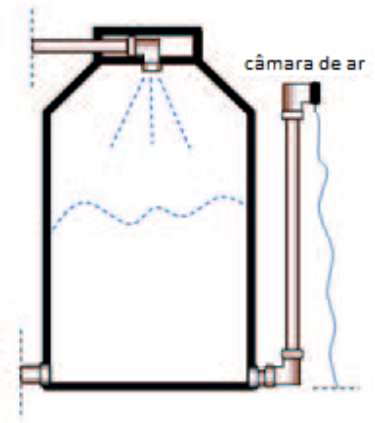


Figura 2.5: Sensor no reservatório usando câmara de ar (Traduzida de Gama-Moreno et al., 2010).

Astharini [2012] utilizou sensores de pressão espalhados ao longo da tubulação de água, porém, destacou-se na transmissão dos dados ao transformar as leituras em ondas sonoras que são recebidas através de fios por uma placa de áudio ligada a um computador. É comum a utilização de sensores de pressão para detecção de vazamento, principalmente, em redes distribuidoras de água. Porém, são mais suscetíveis a ruídos e sua precisão melhora conforme a distância entre o sensor e o ponto de vazamento é menor [Sánchez et al., 2020].

2.2.2 Impacto da pressão no consumo interno

Os resultados obtidos nas buscas demonstraram considerável número de trabalhos realizando análise de pressão em WDN, o que levou à busca de trabalhos que estudassem a relação de pressão e vazamento dentro de residências. Foi possível encontrar o estudo de Zhao et al. [2018], que foram a campo e fizeram diversas medições para avaliar o quanto a pressão na tubulação interna de arranha-céus impactava na vazão de água/consumo. A pressão da água em edifícios altos sempre excede a pressão nominal recomendada pelas normas Zhao et al. [2018] e, consequentemente, esta discrepância faz com que a vazão de água da torneira exceda o fluxo nominal reduzindo o conforto

de uso devido a severos respingos, uma vez que a taxa de fluxo é proporcional à pressão nos pontos de distribuição de água [Zhao et al., 2018].

Zhao et al. [2018] realizaram 195 testes em 23 arranha-céus chineses no período de julho à dezembro de 2014 e utilizaram válvulas de pressão Amico Y22X-16T nos pontos de teste (torneiras novas e antigas, descargas, chuveiros e mictórios) e medidores de água da mesma marca a fim de calcular o sobre-fluxo de água decorrente do excesso de pressão na tubulação, e mensurar, de maneira teórica, a economia de luz com a redução do acionamento de bombas. Constataram ainda que 64,5% das torneiras estudadas estavam com pressão acima de 0,2MPa, recomendada pela norma GB50555-2010. Em suas medições de campo, eles relacionaram o fluxo com a pressão nos pontos de consumo como denota a Figura 2.6.

Zhao et al. [2018] alegaram ainda que esse experimento não seria suficiente para correlacionar pressão e vazão devido a fatores não avaliados, como diâmetro de tubulação de distribuição, idade e corrosão dos canos. Por estes motivos, eles simularam consumo de diferentes dispositivos em laboratório usando: torneira sem e com arejador, torneira misturadora e chuveiro. O resultado da simulação pode ser observado na Figura 2.7, demonstrando de forma mais clara o aumento da vazão para maiores pressões. Constataram, por exemplo, que uma torneira sanitária de alavanca sem aerador tinha uma vazão de 0,510 ± 0,33 L/s a 0,5MPa, sendo que a mesma torneira, a uma pressão de 0,2MPa, teve vazão de 0,213 ± 0,22 L/s possibilitando a redução de 58,2% na vazão.

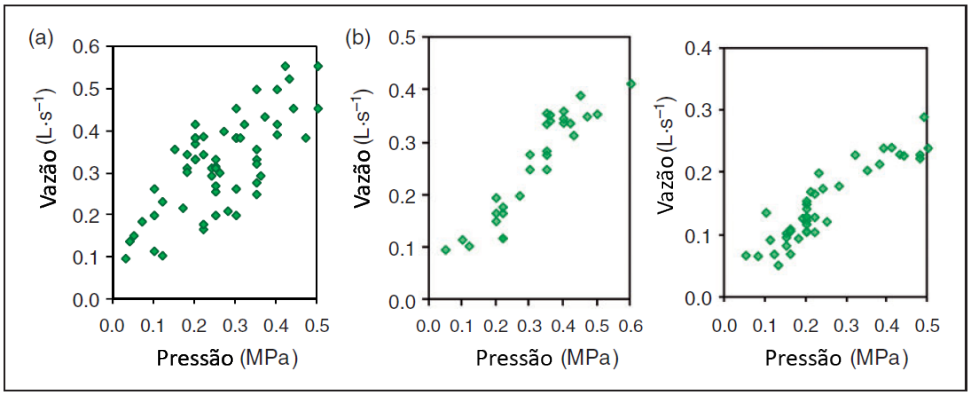


Figura 2.6: Teste de campo das características de pressão-vazão de diferentes torneiras. (a) Torneira sanitária de alavanca única sem aerador; (b) Torneira sanitária de alavanca única com aerador; e (c) Torneira misturadora de água. [Traduzida de Zhao et al. [2018]]

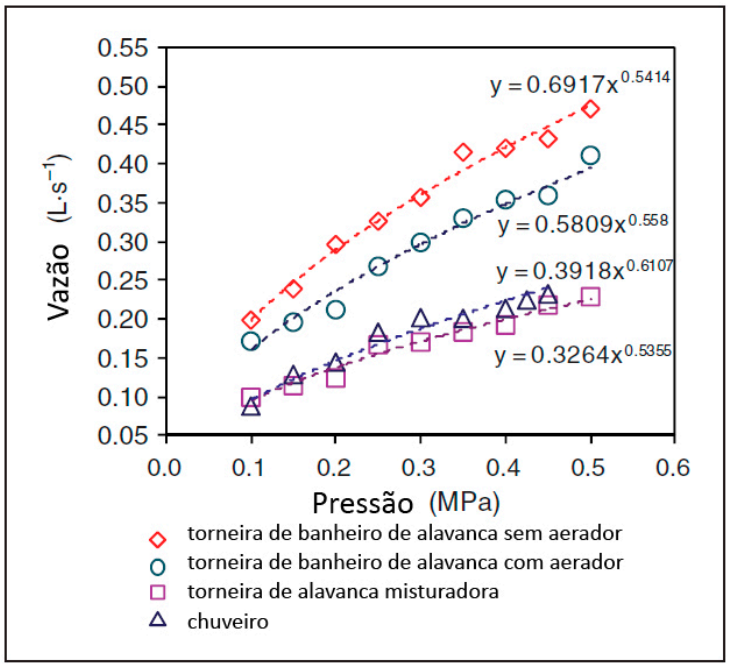


Figura 2.7: Características de pressão-vazão de torneiras e chuveiro obtidas através de simulações realizadas em laboratório. [Traduzida de Zhao et al., 2018]

2.2.3 Técnicas de detecção de vazamento

Há uma diversidade de técnicas que podem ser utilizadas para detecção de vazamento. Foram encontradas as seguintes: regras de associação; aprendizado de máquina; modelos físico-matemáticos; lógica Fuzzy; processamento de imagem; e ferramentas matemáticas e estatísticas. Com base nos casos de consumo brasileiros (Figura 1.2), foi possível agrupar as diferentes soluções apresentadas nos artigos segundo sua finalidade, sensor, e técnica de detecção por meio da taxonomia proposta na Figura 2.8. Destaca-se que um mesmo tipo de sensor pode ser utilizado de forma invasiva, quando é preciso estar dentro das instalações hidráulicas para sua utilização, ou não-invasiva, quando este pode ser acoplado à tubulação externamente.

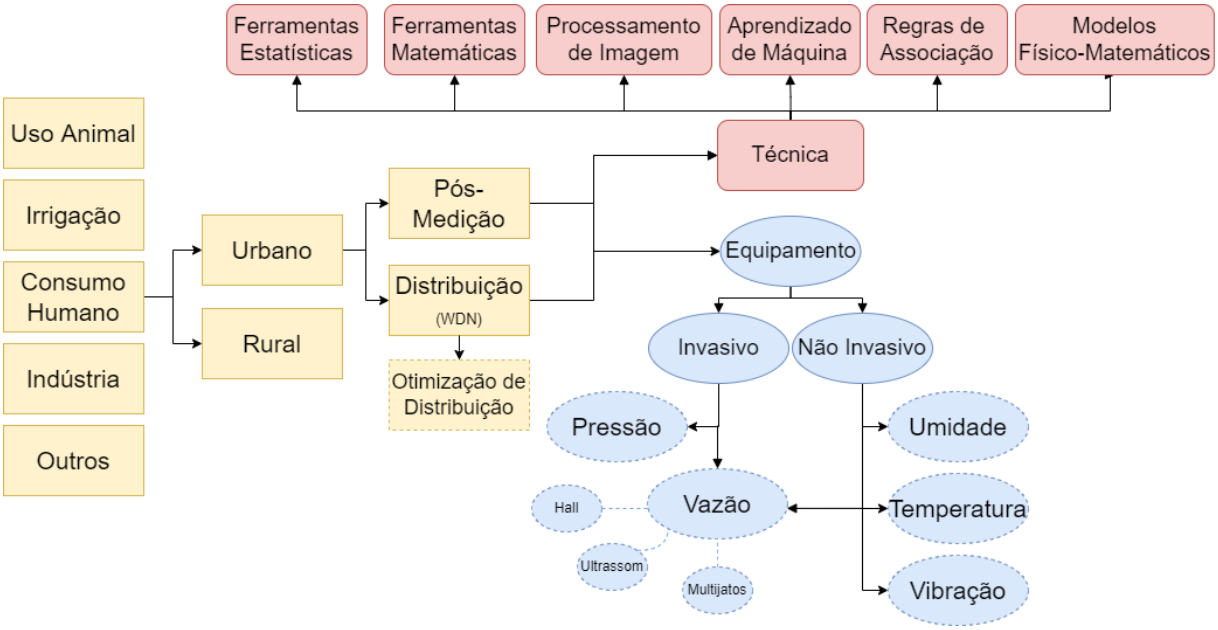


Figura 2.8: Taxonomia para classificação dos artigos

A Figura 2.8, que deve ser lida da esquerda para a direita, é composta de: retângulos amarelos representando os principais usos da água conforme ANA [2019]; os retângulos arredondados vermelhos representam a categoria das técnicas de detecção de vazamento encontradas na literatura; as elipses azuis contínuas representam a forma como se aplica o sensor, invasivo ou não invasivo; e , por fim, os sensores podem ser encontrados dentro das elipses azuis pontilhadas. A taxonomia, que poderá ser expandida no futuro, deu ênfase ao detalhamento do consumo humano urbano devido ao foco deste trabalho.

Regras de associação

A definição de regras para detecção de vazamentos, ou seja, a programação por regras de associação, foi utilizada em alguns trabalhos na literatura. Essas estruturas remetem a sistemas especialistas, nos quais um humano com grande conhecimento e experiência na área de atuação do novo programa era contratado para criar uma série de estruturas condicionais que analisariam os dados (Araújo, 2015). Estes sistemas eram muito utilizados antes das técnicas de aprendizado de máquina (*machine learning*).

Dentre as regras de associação mais conhecidas para detecção de vazamentos em residências podem-se destacar: *Non-Zero Water Consumption* (Consume Non Zero (CNZ)), implementada por Britton et al. [2013] e *Minimum Night Flow* (MNF) implementada por Muhammetoglu et al. [2020]. Ambas foram usadas por Fuentes and Mauricio [2020], além de citar diversas outras. A ideia da técnica CNZ é verificar, dentro de um determinado espaço de tempo, se houve pelo menos um consumo zero. Caso isso não ocorra, é um indicativo de vazamento. Enquanto a MNF baseia-se na

inferência de que, em residências e prédios comerciais, é esperado que, no período da madrugada, a vazão seja zero,devido ao fato de as pessoas estarem dormindo. Neste período do dia, os vazamentos ficam mais evidentes nas leituras.

Aprendizado de máquina

Apesar de não ser uma solução nova, atualmente tem-se verificado uma quantidade maior de trabalhos que implementam algoritmos de aprendizado de máquina supervisionado para solução de diversos problemas. O conceito se baseia em utilização de dados previamente conhecidos, incluindo seus resultados, em modelos que aprendem com estes dados e possibilitam prever, com algum grau de certeza, o resultado da análise de novos dados não utilizados durante o aprendizado. Há também modelos que aprendem sozinhos de acordo com suas interações com o problema, como as redes neurais não-supervisionadas.

Fuentes and Mauricio [2020] implementaram o algoritmo de aprendizado de máquina supervisionado K-Nearest Neighbors (KNN) comparando os k-vizinhos mais próximos com a medição suspeita de vazamento, alcançando uma acurácia de 74% ao detectar vazamentos. Já Amaxilatis et al. [2020] implementaram as chamadas funções lambda, usando as técnicas de regressão linear e aprendizado não supervisionado Clustering KNN, além de funções baseadas em séries de Fourier.

Modelos físico-matemáticos e ferramentas matemáticas

Um modelo matemático é uma interpretação ou representação simplificada da realidade, ou uma interpretação de um fragmento de um sistema, mas características essenciais do mundo real devem aparecer no modelo de modo que o seu comportamento seja igual ou semelhante à realidade. Outra maneira possível para se detectar vazamentos é a utilização de modelo matemático que represente o comportamento do fluxo de água dentro do encanamento.

O modelo matemático proposto por Coronado et al. [2019] utiliza os seguintes parâmetros: pressão na entrada e saída; fluxo na entrada e saída; gravidade; diâmetro da tubulação; e coeficiente de fricção da água. Os autores realizaram experimentos em laboratório a céu aberto, onde foi possível detectar e localizar vazamentos. O modelo foi utilizado em adutoras de abastecimento onde não há consumo, apenas fluxo. Baseando-se no mesmo princípio de modelagem matemática utilizado por Coronado et al. [2019], porém explicando de forma mais detalhada, Razvarz et al. [2020] se diferenciam ao utilizar lógica Fuzzy, “fuzzificando” as entradas com funções triangulares, gaussianas ou exponenciais, para indicar as chances de haver vazamento e identificar o ponto de ruptura baseando-se em simulações.

Processamento de imagem

Pietrosanto et al. [2021] inovaram ao propor detecção de vazamento pela observação do hidrômetro de forma automatizada. Primeiro, em seu laboratório de metrologia, avaliaram o limite de sensibilidade de três hidrômetros domésticos. Usando a diretiva 2014/32/EU, definiram o limiar TH que é o fluxo máximo que não gere qualquer rotação no digito ou ponteiro menos significativo. Os resultados de sensibilidade encontrados podem ser vistos na Tabela 2.2.

A primeira medição do TH foi feita por olho humano. Para automatização do processo, os pesquisadores desenvolveram um protótipo utilizando uma câmera de sensor CMOS modelo OV7670 ligada por cabo ao microcontrolador STM32F407VGT6 da fabricante STMicroeletronics. Diversas fotos são tiradas e enviadas via cabo para um computador onde o algoritmo desenvolvido pela equipe em MatLab analisa pequenas variações nos dígitos do marcador do hidrômetro. Essa parte da imagem é conhecida como Região de Interesse (ROI, do inglês *Region of Interest*). Entretanto, somente a leitura das imagens não é suficiente para detectar vazamento. Por este motivo, os autores sugerem a utilização em conjunto com regras de associação CMZ ou PWNC (*Period Without Null Consumption*). Como projeto futuro, pretendem implementar o mesmo algoritmo em processador de execução em tempo real.

Ferramentas estatísticas

Silva Júnior [2017] optou por processar o fluxo de água tanto no nó sensor, quanto no nó coordenador, tomando-se a média aritmética e desvio padrão de 32 amostras, utilizando o erro quadrático médio (EQM) sobre 1024 amostras como método de verificação dos cálculos dos sensores; ao conjunto do *hardware* deu o nome LWC (*Local Water Control*). "O cálculo da média de leitura acrescida do desvio padrão, deveria ser maior que a média de leitura acrescida do EQM, e o cálculo da média de leitura subtraída do desvio padrão, deveria ser menor que a média de leitura subtraída do EQM, caso fossem encontradas médias divergentes em 3 ciclos consecutivos de 1.024 amostras de medições, era então identificado um evento na rede de distribuição que provavelmente indicasse a ocorrência de distúrbios indesejados, como a exemplo de vazamentos." [Silva Júnior, 2017].

Intervalo de medição

Uma dúvida pertinente refere-se a qual seria o melhor tempo de obtenção de dados para melhor visualização do problema de vazamento. Para um acompanhamento macro, Muhammetoglu et al. [2020] realizaram a medição da vazão a cada 15 minutos, porém para detecção de vazamentos sugeriu-se a realização de medições a cada 2 minutos durante a execução de rotina específica de busca. Por outro lado, Fuentes and Mauricio [2020] sugerem intervalos de 1 minuto entre as medições

no geral.

Muhammetoglu et al. [2020] ressaltam que a análise por sensores de vazão não é capaz de diferenciar mal uso, como, por exemplo, uma torneira aberta sem presença de vazamento aparente, além de não levar em consideração a quantidade de dispositivos consumidores no ramal. Também chamam a atenção para a importância do uso de medidor de vazão com diâmetro adequado ao consumo, pois há perda de sensibilidade. Os sensores mediram o fluxo de água ao longo dos dias durante todo o dia em intervalos de 15 minutos e, a partir destes dados, foram identificados padrões indicativos de vazamentos nas instalações hidráulicas, assim como foram identificados comportamentos de consumo.

Sánchez et al. [2015] utilizaram uma sonda de nível submersa no reservatório superior onde registraram, a cada 5 minutos, o nível de água. Esses valores foram agrupados em conjuntos de uma hora e convertidos posteriormente em quantidade de água consumida. Utilizaram um programa de computador que realizou operações algébricas relacionando variação de nível/volume, área do reservatório e períodos de acionamento da bomba. Desta maneira, traçaram perfis de consumo para cada hora de todos os dias da semana e feriados.

Otimização de rede

Apesar de não detectar vazamento em si, a otimização de distribuição de água em WDN proporciona um melhor uso, racionamento e economia de recursos hídricos e financeiro. Além disso, permite que sejam reduzidos os espaços de busca de vazamento somente para um determinado segmento ou região da rede de distribuição. Por isso, são citados alguns trabalhos interessantes neste seguimento e que podem servir de introdução ao assunto para os interessados.

Mala-Jetmarova et al. [2018] apresentam uma revisão da literatura sobre o problema. Os autores demonstram que a área de otimização para distribuição de água é fonte de inúmeros estudos desde a década de 1980. Os autores apresentam uma tabela com inúmeros trabalhos sobre o assunto.

Já Martinho et al. [2021] apresentam um modelo para o design de rede de distribuição de água considerando efeitos da gravidade, além de uma nova técnica de solução. Percebe-se que apesar de uma área bem estabelecida ainda existem desafios e fontes interessantes de pesquisa.

2.2.4 Arquitetura de processamento

Avaliando as diferentes arquiteturas apresentadas por cada trabalho encontrado, foram observadas as três abordagens de arquiteturas de IoT existentes: borda (*edge computing*), névoa (*fog computing*) e nuvem (*cloud computing*), além de abordagens híbridas.

Computação em borda

A arquitetura de computação na borda foi explorada opor Gama-Moreno et al. [2010]. Neste artigo, os autores utilizaram parte do processamento do microcontrolador PIC 18F4550A, sem uma placa de interface, para verificar fluxo por meio de sensores de vazão de água em horários em que não deveria ter consumo, *i.e.*, nos horários da madrugada. A mesma técnica foi utilizada por Muhammetoglu et al. [2020], porém com análise feita por humanos e não por meio de algoritmos, identificando o intervalo de 00:00 às 05:00 como alvo devido ao direcionamento do estudo em locais públicos tais como escolas, banheiros públicos e parques.

Computação em nuvem

Já o processamento em nuvem foi a solução escolhida por Muhammetoglu et al. [2020] que, além de mostrar dados visualmente, avisa quando encontra um vazamento por meio de mensagem de texto em celulares, tendo também se diferenciado por usar um medidor inteligente/hidrômetro (*smart meter*) proprietário da empresa MANAS. Sua classificação quanto a engenharia foi de "Não Destrutivo" por prever a substituição do hidrômetro antigo pelo novo sem requerer nenhuma obra.

Computação em névoa

Amaxilatis et al. [2020] citam que o processamento em nuvem tem como grande vantagem o maior poder computacional, homogeneidade dos dispositivos e acesso de qualquer local do planeta, porém costumam ser serviços pagos, são centralizados para reduzir custos, passivos de indesejados atrasos e congestionamentos na rede. Uma forma de reduzir o tráfego de dados para nuvem e ter redução no tempo de latência é explorar o poder computacional de dispositivos espalhados pela rede interna do usuário. Por estes motivos, os autores escolheram utilizar a computação em névoa, o que reduziu significativamente a quantidade de dados a serem enviados para a nuvem, restringindo o envio a apenas 5% do tráfego gerado na rede interna além de prover melhores recursos gerais de computação, segurança e privacidade para a solução desenvolvida. Os autores também citam que a escolha por essa solução de arquitetura de processamento gera ganhos no tempo de resposta, que é mais rápido já que as requisições não precisam ir para um sistema de processamento distante, sendo processado em dispositivos em névoa tais como: servidores, *notebooks*, roteadores de borda e *smartphones*.

Quando Amaxilatis et al. [2020] optaram por computação em névoa, realizaram testes de desempenho nos nós de processamento chamados de Tergo e Tethys onde abordou-se o uso de um Raspberry Pi, um *notebook* equipado com processador Intel Core i3-3120M e um servidor Intel E5-2630V4, com diferentes configurações de RAM e armazenamento SD ou SSD. Porém, considerou-se

uma limitação do uso de apenas 1 núcleo de processamento. Em suas conclusões, os autores indicam o Raspberry Pi como solução de baixo consumo e processamento aceitável para projeto de até 30 medidores inteligentes, atingindo latência de processamento de 1,62 milissegundos, e conseguindo lidar com operações de criptografia sem oferecer nenhum atraso significativo na comunicação.

Coronado et al. [2019] também optaram por computação em névoa e usaram um Arduino UNO para processamento. A arquitetura proposta não só detecta se há vazamento, mas também estima o local da ruptura na tubulação.

Computação híbrida

Fuentes and Mauricio [2020] detalharam as ferramentas usadas para o processamento híbrido névoa e nuvem. Optaram pelo pré-processamento dos dados em névoa, implantando o sistema IoT Node-Red da IBM com banco de dados No-SQL Couch DB num computador Raspberry Pi. Os dados pré-processados são transferidos para a nuvem IBM Cloud. Recepcionados pela plataforma IBM Watson, ficam disponíveis para o algoritmo de aprendizado de máquina via IBM Stream Flow Analysis. Então, o resultado é recepcionado pela Azure Cloud e enviado ao usuário por meio de protocolo Web.

Além da conhecida Node-Red, há outras plataformas IoT: WSo2, Kaa IoT Platform, DeviceHive. SiteWhere, ThingSpeak, Zetta, e Thinger.io. Porém, estas últimas não possuem visualização de dados. Outra plataforma de destaque é a ThingsBoard que possui recursos de segurança como algoritmos de criptografia padrão SSL, certificados X.509 ou *tokens* de acesso; visualização de dados; e suporte a REST APIs [Henschke et al., 2020].

Silva Júnior [2017] optou pelo processamento de uma quantidade menor de dados na borda e, posteriormente, agrupar maior quantidade de dados em névoa para realizar outros cálculos.

2.2.5 Sistemas computacionais

O desenvolvimento de sistemas computacionais próprios também foi foco de alguns trabalhos, por exemplo: Gama-Moreno et al. [2010] desenvolveu o *Application for water leak detection* (AIDA), e, sem placa de interface, utilizou microcontroladores PIC que a partir de sensores de vazão e pressão foram capazes de analisar a vazão e nível do reservatório para detectar vazamentos usando regras de associação, além de alertar o usuário por meio de mensagens de texto em seus celulares; Coronado et al. [2019] desenvolveram o Wireless Monitoring System (WMS) que utiliza a plataforma Arduino tanto nos nós coletores quanto no concentrador para processamento de dados por meio de regras de associação em computação em névoa; enquanto Romeiro [2019] criou um algoritmo para transmitir dados de uma placa NodeMCU para a nuvem Dropbox[9] patenteando seu trabalho junto ao Instituto

[9] www.dropbox.com

Nacional da Propriedade Industrial (INPI).

Já Fuentes and Mauricio [2020] optaram por uma solução que mistura regras de associação e aprendizado de máquina. Os algoritmos detectam o vazamento e perguntam ao usuário se o resultado é verdadeiro. Com isso, aos poucos, eles estudaram os hábitos de consumo da residência e obtiveram melhorias na detecção de vazamentos obtidas pelo algoritmo de aprendizado de máquina. Este trabalho optou por utilizar o método KNN e chegou a uma acurácia de 74% em uma arquitetura em nuvem, porém, com alguns dados pré-processados em dispositivos de borda. Enquanto Silva Júnior [2017] utilizou a ferramenta de Business Intelligence (BI) QlikView Personal Edition para criar painéis com representações gráficas dos dados obtidos pelos sensores.

Apesar de não ser uma abordagem para detecção de vazamento, e sim para IoT de forma geral, o trabalho de Pantoja et al. [2019] também foi avaliado. Os autores propõem a disponibilização dos dados coletados por meio de uma arquitetura, nomeada de *Research Management Architecture* (RMA), de tal maneira que transponha as dificuldades de interpolação e comunicação decorrentes da heterogeneidade dos fabricantes e tipos de sensores utilizando uma camada acessível a todos via *webservice*. A arquitetura é dividida em três camadas: *Device*, para aquisição e envio de dados dos sensores; *Cloud*, responsável por manter os dados atualizados e disponíveis para todos via RESTful *web services*; e por fim *Client*, que implementa um mecanismo de *pusblish* e *subscribe* para que desenvolvedores possam obter dados de sensores reais sem possuí-los. *Webservice* é um conjunto de funções invocadas por aplicações de outros sistemas por meio da Web permitindo que novos sistemas interajam com outros existentes mesmo quando escritos em linguagens de programação distintas.

2.2.6 Transmissão de dados

Durante a pesquisa exploratória, foi possível identificar tanto o uso de cabos quanto a utilização de redes sem fio para a transmissão de dados. Nesta temática, há duas áreas relacionadas de importância: Tecnologia e Procolo.

Tecnologia de transmissão

Em trabalhos que utilizam a plataforma de prototipagem Arduino, é comum o uso de placa de extensão, conhecidas como *shields*. XBee (padrão IEEE 802.15.4) é bem difundido para a comunicação sem fio com Arduino e foi utilizado por Coronado et al. [2019], porém existem outras tecnologias emergindo junto com os dispositivos IoT. Trabalhos que usaram NodeMCU optaram pelo uso do Wifi, por ser uma tecnologia embarcada nesse dispositivo e de menor custo [Fuentes and Mauricio, 2020; de Assis Jr et al., 2022; Romeiro, 2019].

Amaxilatis et al. [2020] utilizaram outra tecnologia, em frequência sub-GHz, a Wireless Meter Bus (wM-Bus)[10], para comunicações sem fio entre os medidores inteligentes e os nós que recebem a telemetria, e usaram LoRa na comunicação entre esses receptores e os nós de processamento. Verificaram que taxa de entrega de pacotes do LoRa oscilou entre 89-98% enquanto que a taxa de entrega do wM-Bus ficou entre 42-70%.

Amaxilatis et al. [2020] relatam que perceberam interferência nas transmissões LoRa de equipamentos emissores de RF tais como controles de abertura de garagem. Também constataram que ,em dias de grande movimentação de humanos, houve maior taxa de retransmissões de pacotes se comparado aos finais de semana e feriados onde praticamente houve sucesso já na primeira tentativa de envio dos pacotes.

Protocolos

Para a transmissão dos dados, é necessário escolher, além da tecnologia e meio físico, protocolos de comunicação. Em redes de maior capacidade, tais como as redes sem fio padrão IEEE 802.11, é comum o uso de protocolos como o HTTP utilizado por Fuentes and Mauricio [2020] na comunicação entre o nó de pré-processamento (Raspberry Pi) e o servidor em nuvem. Porém, em redes de menor capacidade, o uso do HTTP se torna difícil devido ao tamanho do cabeçalho. Por isso, existem protocolos específicos para soluções IoT.

A tecnologia de rede ZigBee, que inclui um protocolo, é muito difundido na área [Coronado et al., 2019; Adsul et al., 2017; Gama-Moreno et al., 2010]. Porém, outro candidato que tem emergido para a tarefa é o MQTT. Desenvolvido na década de 1990 pela IBM para gerenciar a transmissão de dados assíncronos através de redes intermitentes [Silva Júnior, 2017], o MQTT possui as seguintes características: clientes MQTT são muito pequenos, requerendo poucos espaços na memória dos microcontroladores; seus cabeçalhos são pequenos e otimizados para redes com pouca largura de banda; suporta protocolos de segurança e é escalável permitindo adição de novos nós com facilidade (de Assis Jr et al., 2022).

Wukkadada et al. [2018] comparam o funcionamento do HTTP e MQTT para aplicações IoT. Ambos os protocolos atuam na camada de aplicação do modelo Open System Interconnection (OSI). Verificou-se que, para estabelecer uma comunicação HTTP, são necessários nove pacotes a cada envio de mensagem, enquanto que no MQTT, são necessários apenas dois pacotes. Desta forma, o uso do HTTP dificulta as várias transmissões de telemetria devido à possibilidade de sobrecarga da rede. Como as mensagens MQTT possuem tamanho de 2 bytes, ou seja, um tamanho pequeno quando comparado à mensagem HTTP, o consumo de energia no envio de mensagens MQTT são sempre muito menores que o consumo no envio de mensagens feito em HTTP, contribuindo também para

[10] st.com/en/applications/connectivity/wm-bus.html

uma duração maior da bateria. Por fim, a recepção de mensagens MQTT possui ínfima taxa de perda de pacotes.

A seguir, pode-se verificar um resumo dos artigos nas Tabelas 2.3 e 2.4 compostas pelas colunas: Autor; Finalidade, onde é mostrado local de implementação da solução e qual propósito; Hardware, que mostra ,primeiramente, se a solução necessita de obras e, em segundo, qual o sensor usado e, por fim, qual microcontrolador ou placa de prototipagem foi usada; a coluna Software é composta pela estrutura de processamento (névoa, borda ou nuvem), se foi utilizado algum software existente ou se foi criado, e neste caso, qual nome e ou a linguagem utilizada; enquanto que a coluna Transmissão é composta pela tecnologia empregada seguida do protocolo de aplicação; por fim, mostra-se a quantidade de citações do artigo no Google Acadêmico[11] em agosto de 2022.

Após a análise dos artigos, notou-se que poucos fazem estudo do comportamento de consumo (Tabela 2.5), geralmente feito por humanos. Neste quesito, Fuentes and Mauricio [2020] se destacaram por validar os vazamentos detectados pelos algoritmos junto aos usuários, além de armazenar esses dados e os utilizar como base de comparação para um método de aprendizado de máquina, de modo a reforçar o sistema de detecção de vazamentos. Já Muhammetoglu et al. [2020] fez inspeções visuais durante alguns vazamentos detectados pelo sistema e identificou uma torneira aberta no local de estudo, por exemplo.

[11] scholar.google.com.br

Tabela 2.3: Lista de artigos com informações gerais resumidas

Autor	Finalidade	Equipamento	Software	Transmissão	Cit.
	Ambiente de uso; Objetivo	Instalação Invasiva ou não; Tipo do Sensor; Placa/ Microcontrolador	Arquitetura de Software; Ambiente tecnológico; Linguagem	Meio; Protocolo	
Alarefi and Walker (2017)	Residência; Detecção de Vazamento	Não Invasivo; Ultrassom(Vazão); Não tem	Fog (Computador); Não diz o software usado	Cabo USB (Não diz o protocolo)	5
Astharini (2012)	Indústria; Apenas Monitorar	Invasivo; Pressão; Placa de Áudio (PC)	Fog (Computador); MS Viewer; .Net C#	Cabo (Sem protocolo)	0
Gama-Moreno et al. (2010)	Residência; Detecção de Vazamento	Ambas; Pressão (MPX10D) e Vazão; PIC 18F4550A	Edge (PIC) e Fog (Gráficos); Regra Associação (AIDA)	Xbee (ZigBee) GSM (RS232)	5
Coronado et al. (2019)	Redes de Distribuição; Detecção de Vazamento	Invasivo; Pressão (Promag PMP41) e Vazão (Promag Proline 10P); Arduino Uno	Edge(Arduino); Regra Associação (WMS)	Xbee (ZigBee)	0
Adsul et al. (2017)	Não define; Apenas Monitorar	Não Invasivo; Pressão, Temperatura, Som e Ultrassom; Arduino Uno	Fog (Computador); LabView	Xbee (ZigBee)	16
Silva Júnior (2017)	Redes de Distribuição; Apenas Monitorar	Invasivo; Vazão (Seeed YF-S201); Arduino Mega 2560	Edge e Fog(Computador); Dashboard (QlikView)	Xbee (MQTT)	3
Araújo (2015)	Residência; Detecção de Vazamento	Invasivo; Simula Vazão; Não possui	Fog(Computador); Regra Associação (C#)	Não tem	0
Muhammetoglu et al. (2020)	Residência e Locais Públicos; Detecção de Vazamento	Invasivo; Vazão; Hidrômetro inteligente MANAS	Cloud; Website	GSM	6
Sánchez-Rivero et al. (2020)	Redes de Distribuição; Detecção de Vazamento	Não implementou; Pressão; Não se aplica	Fog(Computador); EPANET; Meta-heurística	Não se aplica	2
Romeiro (2019)	Residência; Segregação de Consumo e Monitoramento	Invasivo; Vazão (Seeed YF-S201); NodeMCU (ESP8266EX)	Edge e Cloud; Dropbox; Arduino IDE (C)	Wifi (HTTP)	1
Fuentes and Mauricio (2020)	Residência; Detecção de Vazamento	Invasivo; Vazão (Seeed G3&4); NodeMCU (ESP8266EX)	Fog (Raspberry Pi) e Cloud; Node-Red, IBM Cloud e Azure Cloud; Condicional, Machine Learning (KNN) e Python	Wifi (MQTT, HTTP)	25

Tabela 2.4: Lista de artigos com informações gerais resumidas (Continuação)

Autor	Finalidade	Hardware	Software	Transmissão	Cit.
	Ambiente de uso; Objetivo	Instalação Invasiva ou não; Tipo do Sensor; Placa/ Microcontrolador	Arquitetura de Software; Ambiente tecnológico; Linguagem	Meio; Protocolo	
Amaxilatis et al. (2020)	Redes de Distribuição; Apenas Monitora	Invasivo; Vazão e pressão; Hidrômetro inteligente	Fog (Raspberry Pi Servidor Desktop); Série de Fourrier, Clustering KNN, Regressão Linear; Java	wM-Bus e LoRa (MQTT)	14
Britton et al. (2013)	Residência; Detecção de Vazamento	Invasivo; Vazão; Hidrômetro inteligente Elster V100	Fog; Não informado	Sem Fio Não informada	198
Pietrosanto et al. (2021)	Residência; Detecção de Vazamento	Não Invasivo; Vazão (Hidrômetros) e Imagem (OV7670); STMicroeletronics (STM32F407VGT6)	Fog (Computador); MatLab MathWorks 2019a	Cabo	5
Razvarz et al. (2020)	Rede de Distribuição; Detecção de Vazamento	Invasivo; Vazão e Pressão;	Por simulação; Modelo Matemático (Lógica Fuzzy)	Não se aplica	0
de Assis Jr et al. (2022)	Residência; Detecção de Vazamento	Invasivo; Vazão (Seeed YF-S201) e Pressão; NodeMCU ESP32	EDGE e FOG (Raspberry Pi e Notebook); Regras de Associação e Thingsboard; Arduino IDE (C)	Wifi (MQTT e HTTP)	0

Tabela 2.5: Análise dos artigos quanto a estudo do consumo

Autor	**Estudo do Consumo**
Araújo (2015)	Mede consumo 15 dias antes do início de operação do sistema
Muhammetoglu et al. (2020)	Análise realizada por humano
Romeiro (2019)	Análise realizada por humano
Fuentes and Mauricio (2020)	Análise feita por Humano e Aprendizado de Máquina
Britton et al. (2013)	Análise feita por hidrômetro inteligente e humanos

Capítulo 3

Sistema de detecção de vazamento de água

Este capítulo apresenta a arquitetura desenvolvida neste trabalho utilizada para a detecção de vazamento e identificação de hábitos de consumo de água em ambientes residenciais. Também são apresentados os algoritmos utilizados para medir o fluxo de água, a pressão e para detecção de vazamento.

3.1 Arquitetura Proposta

Após a realização de comparações entre algumas plataformas de prototipagem, o NodeMCU ESP32 foi escolhido para compor a arquitetura do sistema por possuir mais memória do que o Arduino Uno, dois núcleos para processamento, além de conexão Wifi e Bluetooth embarcados. O objetivo deste trabalho é encontrar uma solução tecnológica de menor custo para detecção de vazamentos em residências, explorando o poder computacional dos dispositivos de borda e névoa como diferencial. Hoje, as aplicações focam mais em processamento na nuvem, porém há um crescente interesse nas demais arquiteturas.

Informações coletadas por sensores e dispositivos IoT em tempo real geram dados que podem auxiliar no desenvolvimento de projetos que promovam a mudança de comportamento das pessoas quanto ao consumo de água [Amaxilatis et al., 2020]. Conforme apresentado anteriormente, neste trabalho foram escolhidos os sensores de vazão e pressão que medirão o consumo, vazão média e pressão.

O sensor de vazão utilizado foi de efeito Hall devido ao seu baixo custo. Este possui, internamente, seis pás em formato de cata-vento que são movidas com o fluxo de água passante. Esse movimento gira o ímã acoplado. A rotação do campo magnético aciona o sensor Hall, que então emite ondas quadradas (pulsos) [Seeed, 2020]. Esses pulsos elétricos são contabilizados pelo NodeMCU que os transforma em valores mensuráveis.

Outra forma de detecção de vazamento se dá pelo estudo da pressão na tubulação, geralmente usado em WDN. Este sensor tem contato direto com o líquido sofrendo ação da pressão da água. De acordo com a força sofrida, ele gera um valor de tensão que será interpretado e convertido para

pascal (Pa) pelo NodeMCU. O sinal gerado é analógico. Na Figura 3.1 é demonstrada a estrutura interna dos sensores, seu funcionamento e o formato de saída dos dados. Uma vez convertidos em valores mensuráveis, os dados podem ser mostrados de forma gráfica para o usuário.

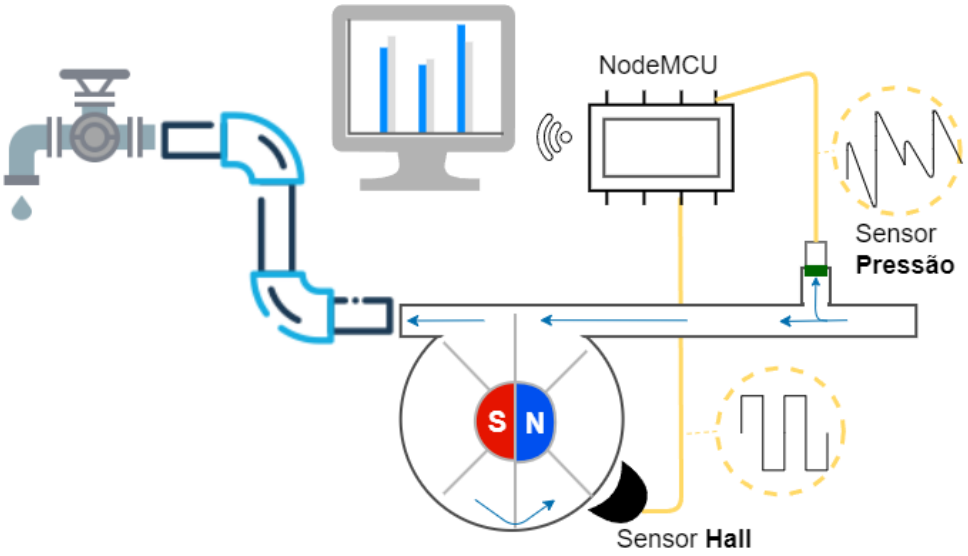


Figura 3.1: Funcionamento dos sensores

A fim de determinar valores para o estudo da pressão em redes internas Zhao et al. [2018] usaram a norma chinesa GB50015-2003 (edição de 2009) que determina uma pressão máxima de 0,45 MPa para evitar danos tais como fissura na tubulação e gotejamento. Por outro lado, a norma GB50555-2010 foca em redução de pressão na rede interna para economia de água, e para tal sugere que a pressão esteja abaixo de 0,2 MPa. Considerando a realidade brasileira, a norma 5626 [ABNT, 2020] sugere uma pressão máxima nos pontos de consumo internos em 0,4 MPa e uma pressão mínima aceitável de 1 kPa em residências e de 5 kPa em prédios. O objetivo de avaliar a pressão é oferecer ferramentas para redução do consumo e evitar vazamentos devido a alta pressão, enquanto que o medidor de vazão permitirá avaliar o consumo em si, permitindo também a identificação de áreas de vazamento ou maior consumo.

A proposta deste trabalho é direcionada a perímetros urbanos onde a densidade populacional é maior e a implementação do protótipo desenvolvido foi realizada na segunda maior metrópole brasileira, a cidade do Rio de Janeiro. Foram exploradas tecnologias de comunicação sem fio IEEE 802.11 e LoRa. Ambas são interessantes, pois estão disponíveis em certas versões do NodeMCU ESP32, sendo a primeira bem mais suscetível a poluição eletromagnética em edifício residencial, onde cada apartamento possui sua própria rede sem fio. Por outro lado, dispositivos LoRa operam em frequência sub-GHz, proporcionando maior alcance e tornando-se uma boa alternativa para prédios comerciais como shoppings ou universidades onde a cobertura precisa ser maior. Diferentemente de Amaxilatis et al. [2020] optou-se por uma conexão LoRa ponto-a-ponto devido ao custo ainda

elevado de um roteador LoRaWan. Propõe-se aqui uma solução híbrida com Wi-Fi em locais mais próximos do roteador.

Optou-se pelos seguintes *softwares*, arquiteturas de *software* e protocolos para criação do *core* de processamento: para criação dos programas a serem executados nas placas NodeMCU, foi escolhida a Arduino IDE[1] que se baseia na linguagem de programação C++. Esta IDE foi escolhida devido a sua simplicidade e vasta documentação disponível. Foram executados testes para averiguar compatibilidade do NodeMCU com a Arduino IDE; para detecção dos vazamentos, foram usadas regras de associação no nó sensor e também em dispositivos na rede interna, explorando os conceitos de computação na névoa e em borda.

Para auxiliar na reunião e recebimento dos dados dos vários sensores, o sistema IoT ThingsBoard[2] foi escolhido. ThingsBoard é uma plataforma de código aberto, portanto gratuita, voltada para desenvolvimento de soluções IoT para coleta de dados, processamento, visualização e gerenciamento de dispositivos, podendo ser rodada localmente ou em nuvem. As telemetrias foram repassadas do Thingsboard para algoritmos de aprendizado de máquina via serviço de mensagem para sistemas distribuídos Apache Kafka. Foi utilizada regressão linear para estimar o consumo da residência e valor da conta de água.

O protocolo de aplicação MQTT foi utilizado na comunicação entre NodeMCU e ThingsBoard. Já para comunicação entre ThingsBoard e o usuário, optou-se por utilizar a interface gráfica da plataforma via protocolo HTTP. As Figuras 3.2 e 3.3 ilustram as opções tecnológicas tanto em *hardware* quanto em *software* implementadas.

Os experimentos do projeto foram planejados para realização de teste de pequenas funções separadamente, para então serem agregadas aos poucos de forma a reduzir erros na implementação e facilitar a identificação de falhas. Optou-se por começar os testes pela placa NodeMCU, para verificar se, realmente, códigos feitos para Arduino por meio da plataforma Arduino IDE eram compatíveis com o NodeMCU, o que se mostrou verídico. O próximo passo foi testar o funcionamento dos sensores escolhidos e verificar sua precisão. Também foi realizado um teste de alcance LoRa *indoor*. Por fim, foram realizados testes com o protocolo MQTT, iniciando-se por testes de comunicação interna entre um nó de testes e o Raspberry Pi e, a seguir, entre o Raspiberry Pi e o NodeMCU. Por fim, foi testada a comunicação entre NodeMCU e ThingsBoard.

As Tabelas 3.2 e 3.1 demonstram a descrição das formas de estudo de consumo e das formas de detecção de vazamentos implementadas, respectivamente, ambas incluindo o local de processamento (i.e., borda, névoa ou nuvem). As contribuições deste trabalho são: implementação de técnicas de detecção de vazamentos conhecidas na literatura com processamento em borda (*edge computing*); análise de consumo baseado em dados retirados do Diagnóstico dos Serviços de Água e Esgotos

[1] www.arduino.cc/en/software
[2] thingsboard.io

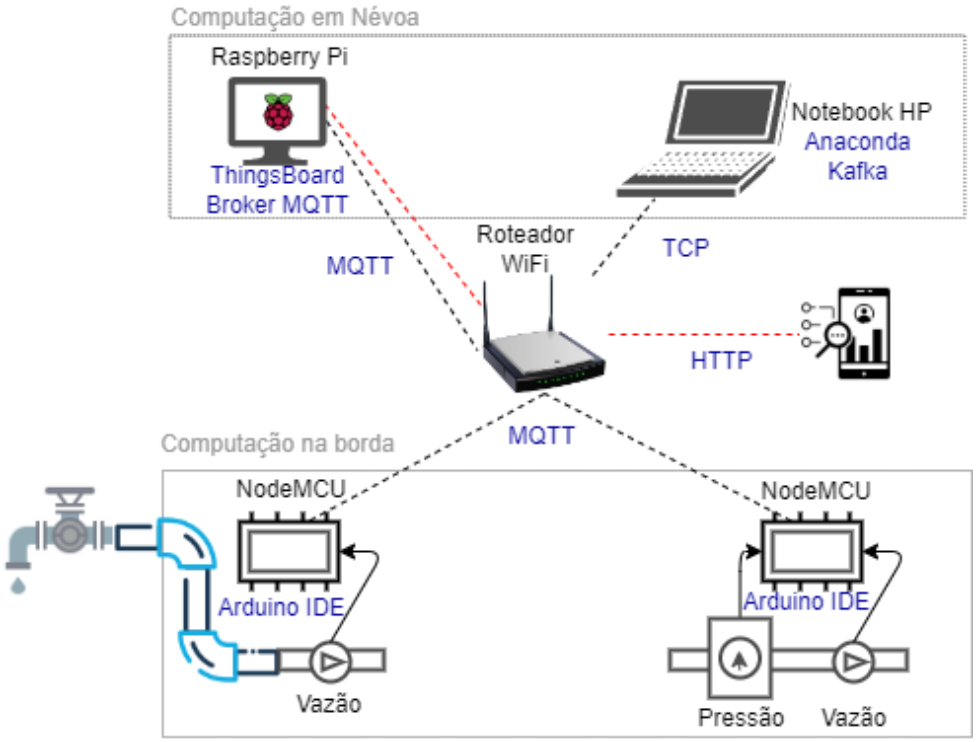


Figura 3.2: Arquitetura implementada

do MDR [2018]; detecção de vazamento analisando fluxo de saída do reservatório e consumo dos cômodos da casa; análise de pressão na tubulação; e análise do comportamento da pressão durante o consumo em residências.

Tabela 3.1: Técnicas de detecção de vazamento a serem exploradas

Formas de Detecção de Vazamento

Descrição	**Local de Processamento**
Mínimo Fluxo Noturno (do inglês MNF)	Borda
Fluxo Zero nos Últimos x tempos (do inglês CNZ)	Borda
Fluxo de saída do reservatório versus fluxo de consumo nos ramais	Névoa
Estimativa de consumo por Regressão Linear	Névoa

Os dados originados dos sensores serão armazenados no banco de dados PostegreSQL por meio do ThingsBoard possuindo a estrutura: *consumo* responsável pela telemetria do consumo de água em litros dentro do intervalo de medição; $vazaoMedia$ que mede a vazão média no mesmo período; *pressao* responsável por armazenar a pressão em kPa (kilo Pascal); *timestamp* que armazena o horário do recebimento da telemetria; o *token*, um conjunto de caracteres que identifica o dispositivo podendo ser palavra composta por letras e números; por fim, *alive* usado para indicar que o dispositivo está funcionando caso não envie telemetria por falta de consumo.

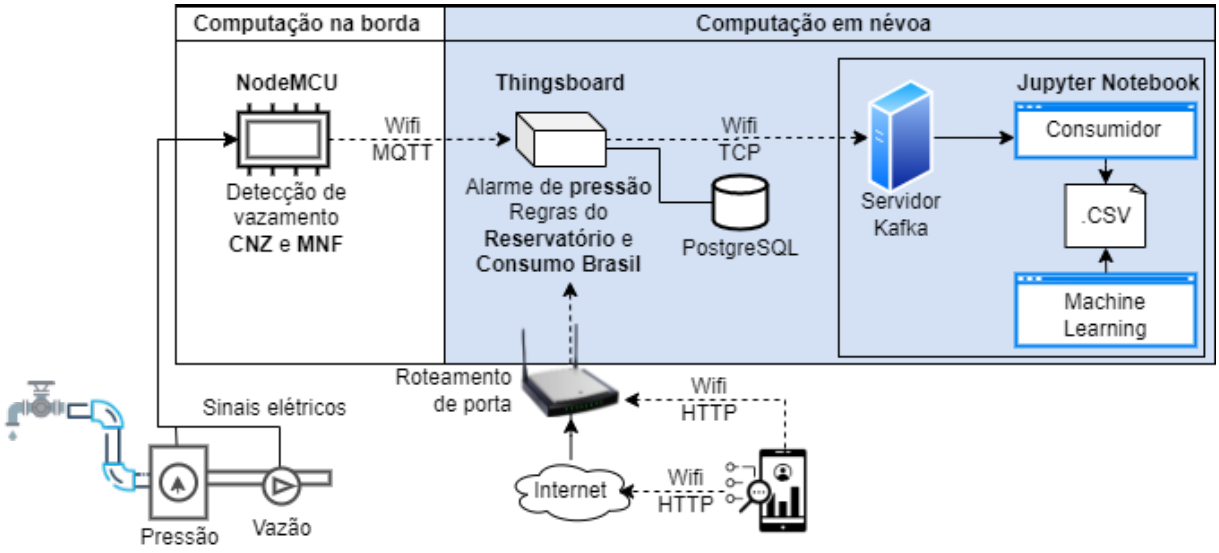


Figura 3.3: Fluxo de processamento

Tabela 3.2: Abordagens para estudo do consumo

Formas de Estudo de Consumo

Descrição	**Local de Processamento**
Comparação de consumo per capita por dia com dados do relatório MDR [2018]	Névoa
Alarme de pressão fora do recomendado	Névoa

3.2 Algoritmos e Implementação

Códigos criados na Arduino IDE para placas de prototipagem Arduino, NodeMCU e demais são compostos basicamente por duas funções: *setup*(), onde são ligadas portas aos componentes externos e também inicializadas variáveis, e *loop*(), onde encontram-se instruções que serão repetidas a todo momento. Podem ser criadas novas funções para ajudar a organizar o código, porém essas só serão executadas quando chamadas dentro das duas funções mencionadas. A seguir, são apresentados os algoritmos utilizados para o monitoramento de fluxo de água, pressão e detecção de vazamentos.

3.2.1 Vazão

O Algoritmo 1 foi desenvolvido para determinação da vazão de água. A inicialização das variáveis, conexão das portas e ativação do sensoriamento por interrupção ocorrem dentro da função *setup*(). Na função *loop*() são realizadas as etapas de cálculo da vazão. A cada um segundo para-se a escuta da interrupção, calcula-se a vazão de água e seu total acumulado, imprime os dados no monitor serial, zera o contador de pulsos e ativa novamente o monitoramento de interrupção. Monitor serial é um recurso da Arduino IDE que permite fazer leitura dos sensores ligados ao NodeMCU. Essas informações chegam ao computador via porta USB.

Algorithm 1 Medição de Vazão/Fluxo de água - 1ª versão

Entrada: $sensorPin$

Função `setup`**:**

```
zerar variáveis
ativarInterrupção(a cada transição decrescente de sensorPin, pulseCount + 1)
```

Função `loop`**:**

```
if tempoAtual − tempoSeg > 1000 then
    pararInterrupção()
    flowRate ← (1000.0*((tempoAtual - tempoAntigo)*pulseCount)) / fatorCalibração
    tempoAntigo ← tempoAtual
    flowMililiters ← (flowRate/60)*1000
    totalMililiters ← totalMililiters + flowMililiters
    imprimir fluxo em litros/min, fluxo total
    pulsecount ← 0
    tempoSeg ← tempoAtual
    ativarInterrupção(a cada transição decrescente de sensorPin, pulseCount + 1)
```

Uma nova versão desse algoritmo foi elaborada. Segundo dados do fabricante do sensor de vazão, para cada litro medido, um sensor de 1/2"gera 450 pulsos; para o sensor de 3/4"são necessários 330 pulsos e para 1/4", 4380 pulsos (Seeed, 2020). São medidos o *consumo* e $vazaoMedia$. A vazão média mede o $\frac{consumido}{intervaloMedicao}$ já que a vazão altera durante o abrir e fechar dos dispositivos consumidores e esse movimento transiente estará dentro de pelo menos um intervalo de medição. O fator de calibração passa a ter um novo sentido mudando seu valor para a quantidade de pulsos necessárias para medição de 1 litro de água.

O Algoritmo 2 demonstra o funcionamento da nova versão. É implementado um *delay* que possui o intervalo de tempo que se deseja medir o consumo. Então são calculados o consumo e vazão média. Por fim, o contador de pulsos é zerado.

Algorithm 2 Medição de Consumo e Vazão Média de Água - 2ª versão

Entrada: $flowPin$, $intervaloMedicao$, $fatorCalibracao$

Função `setup`**:**

```
zerar variáveis
ativarInterrupção(a cada transição decrescente de flowPin, pulseCount + 1)
```

Função `loop`**:**

```
delay(intervaloMedicao) //em minutos
pararInterrupção()
consumo ← pulseCount / fatorCalibracao
vazaoMedia ← (consumo / intervaloMedicao) // Litros/minuto
imprimir consumo e vazaoMedia
ativarInterrupção(a cada transição decrescente de flowPin, pulseCount + 1)
```

A avaliação do funcionamento das duas versões do algoritmo é realizada no próximo capítulo, onde detalhes dos resultados são também apresentados.

3.2.2 Pressão

O Algoritmo 3 apresenta uma descrição de como é realizado o monitoramento da pressão. Neste algoritmo, o $VinSensor$ é a tensão de alimentação do sensor; Offset é o zero referencial que no primeiro momento é configurado como zero; e *leitura* é o valor medido na saída do sensor. Ao executar o código pela primeira vez, o valor de V será o Offset específico para aquele sensor.

Algorithm 3 Medição da pressão

Input: $VinSensor$, $OffSet$, *leitura*

Função `loop`**:**

V ← leitura * VinSensor / $2^{\text{resolucaoPorta}}$
P ← (V - OffSet) * 400
imprima voltagem V em volts
imprima pressão P em kPa

3.3 Comunicação

A comunicação sem fio é essencial para sistemas de monitoramento em tempo real. Diversas tecnologias foram desenvolvidas ao longo dos anos. LoRa é uma tecnologia de transmissão de dados sem fio que provê um alcance maior devido ao uso de uma frequência sub-GHz, sendo no Brasil a frequência permitida de 915Mhz. Neste trabalho, foram necessárias 2 placas NodeMCU com LoRa com códigos diferentes para o emissor e o receptor, conforme descrito nos Algoritmos 4 e 5.

Algorithm 4 Transmissor LoRa

Função `setup`**:**

inicializar módulo LoRa
configurar pinos LoRa
verificar inicialização LoRa
iniciar display

Função `loop`**:**

contador = contador+1
criar mensagem
enviar mensagem
aguardar 1000ms

Algorithm 5 Receptor LoRa

Entrada: *pacote*

Função `loraData`**:**
- escrever no display dados do pacote recebido

Função `cbk`**:**
- transformar pacote recebido em string
- loraData()

Função `setup`**:**
- inicializar módulo LoRa
- configurar pinos LoRa
- verificar inicialização LoRa
- receber pacotes LoRa
- iniciar display

Função `loop`**:**
- tamanho ← LoRa.parsePacket()
- **if** $tamanho > 0$ **then**
  - cbk(tamanho)

3.3.1 Protocolo de Comunicação

Conforme mencionado, este trabalho utiliza o protocolo de envio de mensagens MQTT. Ele foi implementado no NodeMCU com o Algoritmo 6, que tem como entrada as informações da rede Wi-Fi utilizada (*ssid* e *password*), endereço IP e porta do *broker* Raspberry Pi, além do(s) tópico(s) em que está inscrito para receber e mandar mensagens.

Algorithm 6 Envio de Mensagem MQTT

Entrada: $ssid$, $password$, $ipMqttServer$, $mqttPort$, $mqttUser$, $topicSub$, $topicPub$

Função `setup`**:**
- conectarWiFi()
- conectarAoBroker()

Função `loop`**:**
- Enviar mensagem no topicPub
- Receber mensagem do topicSub
- Imprimir mensagem recebida do topicSub no monitor Serial

O Algoritmo 7 é a versão utilizada para comunicação envolvendo os sensores de pressão e vazão integrados ao Thingsboard.

Algorithm 7 Envio de Telemetria de Vazão e Pressão para o ThingsBoard

Entrada: *ssid*, *password*, *ipMqttServer*, *mqttPort*, *topicoPub*, *token*, $flowPin$, *pressurePin*, $fatorCalibracao$

Função `setup`**:**

```
conectarWiFi()
 conectarAoBroker()
 zerar variáveis
 ativarInterrupção(a cada transição decrescente de flowPin, pulseCount + 1)
```

Função `loop`**:**

```
if tempoAtual − tempoSeg > 1000ms then
    pararInterrupção()
     vazaoCoef ← (1000.0*((tempoAtual - tempoAntigo)*pulseCount)) / fatorCalibração
     vazaoMililitros ← (vazaoCoef/60)*1000
     totalMililitros ← totalMililitros + vazaoMililitros

    if tempoAtual − tempoAntigo > 60000ms then
        ler pressao
         calcular pressão
         imprimir pressao e totalMililitros no monitor serial
         enviar pressao e totalMililitros pro ThingsBoard pelo topicoPub
         tempoAntigo ← tempoAtual
         totalMililitros ← 0
    pulseCount ← 0
     tempoSeg ← tempoAtual
     ativarInterrupção(a cada transição decrescente de flowPin, pulseCount + 1)
```

3.4 Abordagens para Detecção de Vazamento

Conforme apresentado, existem diferentes técnicas de detecção de vazamentos conhecidas na literatura. Esta seção apresenta duas abordagens tradicionais conhecidas como *Consume Non Zero* (CNZ) e *Minimun Night Flow* (MNF). Além dessas técnicas, uma abordagem baseada em Aprendizado de Máquina também é apresentada. Nela, espera-se que, baseado em dados de previsão de consumo, seja possível determinar que existe um vazamento, caso o valor obtido seja muito diferente do previsto pelo modelo.

O desenvolvimento deste projeto ocorreu de maneira incremental, partindo de soluções simples e acrescentando-se novas funcionalidades ao sistema. A Figura 3.4 mostra a construção em etapas até a configuração atual. No Algoritmo 8 é mostrada a visão macro do procedimento implementado nos nós sensores (NodeMCU).

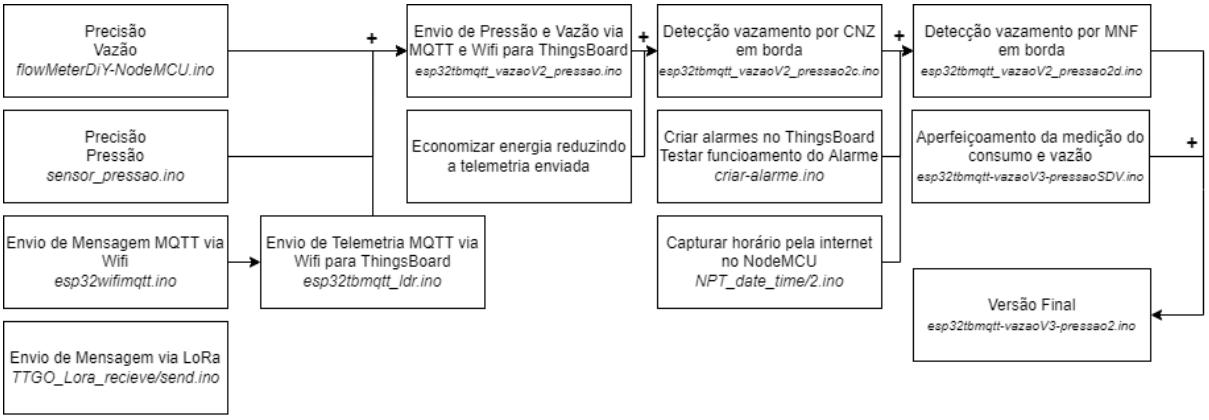


Figura 3.4: Evolução da programação das técnicas de detecção de vazamento do algoritmo rodado em borda

Um detalhe de implementação consiste na implementação da rotina de verificação da conexão com Wi-Fi para restabelecer a conexão com Thingsboard antes do envio da mensagem. Essa necessidade ocorre quando a inatividade de envio de telemetria é superior a 10 minutos, dado que o Thingsboard encerra a conexão com o nó sensor após esse intervalo.

Algorithm 8 Detecção de Vazamento

Entrada: *topicoPub*, $flowPin$, *pressurePin*

Função `setup`**:**

```
conectarWiFi()
 conectarBroker()
 zerar variáveis
 ativarInterrupção(a cada transição decrescente de flowPin, pulseCount + 1)
```

Função `loop`**:**

```
delay (intervaloTelemetria)
 pararInterrupção()
 ler pressão
 consumoLitros ← pulseCount / fatorCalibracao
 vazaoMedia ← consumoLitros /intervaloTelemetria
 if pulseCount > 0 then
    verificar status Wifi e conexão com Broker
     enviar telemetria com consumo, vazaoMedia, pressao ao Thingsboard
     atualizar telemetryTime
     zerar consumoLitros
 else if tempoAtual - telemetryTime > telemetryHours then
    verificar status Wifi e conexão com Broker
     envia telemetria de alive
 else
     zeroFlowTime ← tempoAtual
 pulseCount ← 0
 deteccaoCNZ()
 deteccaoMNF(vazaoMedia, consumoLitros)
 ativarInterrupção(a cada transição decrescente de flowPin, pulseCount + 1)
```

Função `deteccaoMNF(`*consumoLitros, vazao*`):`

```
Detectar vazamento pela técnica MNF
```

Função `deteccaoCNZ(`*zeroFlowTime*`):`

```
Detectar vazamento pela técnica CNZ
```

Função `conectarWiFi(`*ssid,password*`):`

```
Conectar ao Wifi e mostrar dados de rede no monitor serial
```

Função `conectarBroker(`*mqttServer, mqttPort*`):`

```
Conectar ao Thingsboard
```

3.4.1 Algoritmos Tradicionais

Esta seção apresenta as duas abordagens tradicionais utilizadas, os algoritmos *Consume Non Zero* (CNZ) e o *Minimun Night Flow* (MNF).

Consume Non Zero

CNZ é uma técnica que avalia se houve consumo nulo em algum momento nas últimas horas. O tempo de análise pode ser configurado conforme a necessidade. Em locais de maior movimento, como shoppings, espera-se muito uso dos banheiros, por exemplo, e, por isso, tempos maiores de análise são recomendados. Porém, numa residência, uma hora parece ser um intervalo razoável. Esse intervalo carece de validação em ambiente real.

O algoritmo principal verifica se há telemetria não nula de consumo para enviar. No caso de consumo nulo, atualiza o tempo da variável que sinaliza a última medição nula. A cada intervalo de tempo *intervaloTelemetria*, o algoritmo principal chama a função de detectar vazamento por CNZ. Caso verifique um tempo maior que esperado sem consumo nulo, sinaliza vazamento por CNZ.

Assim que o algoritmo encontra um consumo nulo, desfaz o alerta de vazamento, enviando uma informação via telemetria. Seu pseudo-código pode ser verificado no Algoritmo 9.

Algorithm 9 Detecção de Vazamento por CNZ

```
Entrada: zeroFlowTime
Função setup:
    vazamento ← false
    zeroFlowTime ← 0
  Função deteccaoCNZ():
    if tempoAtual - zeroFlowTime > zeroFlowHours e vazamento==False then
        vazamento ← true
        ativar Wifi e conectar ao broker
        enviar telemetria vazamento:CNZ_true
    else if tempoAtual - zeroFlowTime < zeroFlowHours e vazamento==True then
        vazamento ← False
        ativar Wifi e conectar ao broker
        enviar telemetria vazamento:CNZ_false
```

Na Tabela 3.3, encontram-se alguns trabalhos que utilizaram essa técnica e como a implementaram, além do custo dos sensores. No Algoritmo 9, tem-se o pseudo-código da implementação em borda.

Autor	**Hardware (Sensor)**	**Análise de dados realizada em**
Fuentes and Mauricio (2020)	NodeMCU ESP8266 + Sensor Seeed G3&4 R$ 124,00	Névoa (Node-RED), dados e gráficos fornecidos pela plataforma. Análise automatizada.
Britton et al. (2013)	Hidrômetro Inteligente Elster V100 R$ 369,60	Borda com flag indicativo de vazamento. Análise automatizada, porém não utilizada no trabalho.
Autor	NodeMCU ESP32 + Sensor YF-201 R$ 103,00	Borda com geração de alerta. Dados gráficos mostrados via ThingsBoard em névoa.

Tabela 3.3: Trabalhos que usaram CNZ

Minimun Night Flow

Observando que os hábitos noturnos, na maioria das residências, é de não ocorrer consumo de água devido a ser o momento dos moradores estarem dormindo, a técnica MNF entende que neste horário não há consumo. Assim, neste período do dia, vazamentos ficam mais aparentes nas medições. Um fluxo contínuo entre meia-noite e cinco da manhã é forte indicativo de vazamento.

Esse intervalo foi escolhido por Muhammetoglu et al. [2020] que analisou ambientes públicos tais como: banheiros públicos, parques, faculdade e templos religiosos. Essa técnica foi empregada por alguns autores de maneira automatizada ou por análise feita por humanos. Na Tabela 3.4 encontram-se alguns trabalhos que a utilizaram.

Autor	**Hardware (Sensor)**	**Análise de dados realizada em**
Fuentes and Mauricio (2020)	NodeMCU ESP8266 + Sensor Seeed G3&4 R$ 124,00	Névoa (Node-RED), dados e gráficos fornecidos pela plataforma. Análise automatizada.
Muhammetoglu et al. (2020)	Hidrômetro Inteligente MANAS Modelo não citado	Cloud (Website). Análise feita por humanos.
Britton et al. [2013] (2013)	Hidrômetro Inteligente Elster V100 R$ 369,60	Dados armazenados no datalogger do hidrômetro e enviado sem fio ao tablet ou carro equipado para leitura. Análise feita por humanos.
Autor	NodeMCU ESP32 + Sensor YF-201 R$ 103,00	Borda com geração de alerta. Dados gráficos mostrados via ThingsBoard em névoa.

Tabela 3.4: Trabalhos que usaram MNF

O Algoritmo 10 demonstra o pseudo-código implementado. A cada instante de tempo, cujo nome no código é *intervaloTelemetria*, é chamada a função *vazamentoMNF*. O momento de início e término deste algoritmo é determinado por uma consulta à Internet para determinar o horário local. Caso o horário esteja no intervalo de meia noite às cinco da manhã, são contabilizadas a quantidade total de telemetrias e telemetrias nulas. Às seis da manhã, ele analisa os resultados e envia telemetria ao sistema de monitoramento, no caso deste trabalho, ao Thingsboard. Caso tenha sido detectado um vazamento, o sistema mostra visualmente uma lista de alarmes com dados do sensor de origem e a regra que detectou o vazamento. Sua implementação leva em conta esporádicos usos durante a madrugada ao verificar se 80% das medições realizadas são zero.

Algorithm 10 Detecção de Vazamento por MNF

Entrada: *vazao*, *consumo*
Função `setup`**:**
  flagMNF ← false
  vazaoMNF ← 0
Função `deteccaoMNF`(*vazao, consumo*)**:**
  a cada 20 min verificar a hora local via NTP
  if *hora>=0 ou hora<=5* **then**
    if *flagMNF==False* **then**
      acender LED indicativo e avisar via monitor serial início do funcionamento do algoritmo MNF
    flagMNF ← true
    if *vazaoMNF==0* **then**
      telemetriaNull++
    else if *vazaoMNF == 0 ou vazaoMNF > vazao* **then**
      vazaoMNF ← vazao
    telemetriaTotal++
  else if *hora==6 e flagMNF==true* **then**
    if *telemetriaNull/telemetriaTotal < 0.2* **then**
      ativar Wifi e conectar ao broker
      enviar "vazamento: MNF_true, vazaoMNF: vazaoMNF"ao ThingsBoard via topicPub
    zerar variáveis MNF e desligar LED

3.4.2 Aprendizado de Máquina

Também chamado de *Machine Learning* (ML) é um conjunto de diferentes técnicas que envolvem programação, matemática e estatística, buscando dar ao computador a capacidade de encontrar padrões em dados ou prever comportamento nos dados a partir de dados conhecidos [Mitchell, 1997]. Dentre as abordagens existentes de ML, o aprendizado supervisionado consiste naquele onde dados de entrada e dados de saída são conhecidos. A partir de muitas entradas gera-se um modelo adequado para a predição de futuros dados.

A Regressão Linear é uma das técnicas mais simples de aprendizado supervisionado. Ela consiste na minimização de um funcional quadrático dado pelo somatório do erro entre valores de entrada e saída. Assim, obtém-se uma reta que melhor aproxima os valores conhecidos, ou melhor dizendo, essa reta fornece o menor valor para o erro entre todos os pares de dados conhecidos. Uma visão pictórica consiste na reta que, plotando todos os pares (entrada,saída), consegue passar na menor distância entre todos os pontos ao mesmo tempo.

A soma das distâncias da reta aos pontos gera a função custo ou erro ($J(\theta)$) como mostra a Figura 3.5. É importante salientar que nem sempre o menor custo implica no melhor modelo. É preciso balancear a relação para que não tenha-se o chamado *overfiting* que ocorre quando o modelo está muito ajustado ao conjunto de treino, não permitindo uma generalização para dados desconhecidos. Técnicas mais sofisticadas de aprendizado de máquina podem ser utilizadas, mas a regressão linear atende ao propósito de estimar o custo da conta de água e esgoto.

$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Figura 3.5: Função custo $J(\theta)$ da regressão linear

O capítulo a seguir apresenta os experimentos realizados. Neles, são avaliados os diferentes algoritmos apresentados, bem como é feita uma análise sobre os resultados encontrados.

Capítulo 4

Resultados

Neste capítulo, são apresentados o ambiente de desenvolvimento, a validação do sistema desenvolvido, além da descrição dos experimentos realizados e uma discussão sobre os resultados obtidos.

4.1 Ambientes de desenvolvimento

Esta seção descreve os componentes eletrônicos, materiais hidráulicos, programas e bibliotecas que foram utilizados em diferentes experimentos. A Tabela 4.1 apresenta a especificação dos componentes eletrônicos utilizados, além dos custos desses componentes em reais (BRL).

Este trabalho foi desenvolvido utilizando os sistemas operacionais, Linux Raspibian OS[1] versão 3.6 32-bits e Microsoft Windows[2] 10 Pro versão 20H2 64-bits. A Tabela 4.2 apresenta detalhes das bibliotecas utilizadas na IDE do Arduino (versão 1.8.13) para programação dos componentes eletrônicos. Para o gerenciamento da arquitetura IoT foi utilizada a plataforma ThingsBoard (versão 3.4.1) instalada no Linux com o banco de dados PostgreSQL versão 11. A abordagem de aprendizado de máquina desenvolvida utilizou o Apache Kafka 3.2 com o Anaconda e o Python 3.9. Para maiores informações do ambiente de desenvolvimento consulte o Apêndice (**??**).

[1] raspberrypi.org/software/operating-systems/
[2] microsoft.com/pt-br/windows/

Tabela 4.1: Equipamentos eletrônicos utilizados

Fabricante Modelo	**Versão**	**Especificações**	**Valor Unid.**	**Qtd.**
Espressif NodeMCU ESP32	1.0	Microcontrolador Xtensa® Dual-Core 32-bit LX6 @ 240MHz ROM: 448 KBytes RAM: 520 Kbytes Flash: 4 MB Wireless 2.4GHz 802.11 b/g/n Conector micro-usb Bluetooth BLE 4.2	50,00	2
LillyGo NodeMCU c/ LoRa 915Mhz	2.0	Mesmo do NodeMCU + Antena LoRa Móvel (SDX1276) Slot SD card Display OLED Bateria externa	170,00	2
Raspberry Pi	3B	Processador Broadcom Quad-Core 64-bit BCM2837 1GB RAM BCM43438 wireless LAN 4x USB 2.0 HDMI Micro SD 64Gb Class 10 Power source Raspberry 3.0A 5V	315,00	1
Power Bank TP-Link	TL-PB10400	Bateria portátil com 2,5 anos de uso Capacidade: 10.400mAh Saída: 2A 5V	94,00	1
Sensor Vazão Seeed	YF−S201	Funcionamento: DC 4,5V a 18V Tensão de trabalho: DC 4,5V Corrente Máxima: 15mA (DC 5V) Vazão de água: 1 a 30L/min Temperatura de operação: 80°C Pressão da água: $\leq$2 MPa Diâmetro entrada/saída: 1/2"	53,10	1
Sensor Pressão	-	Conexão: Rosca 1/4 BSP Tensão de trabalho: DC 5V Pressão de trabalho: 0 a 1.2 MPa Máx. pressão: 2.4 MPa Temperatura: 0 a 85°C	143,89	1
Notebook Toshiba Tecra	R940	Processador Intel Core i3 3110M @ 2,4GHz Chipset Ivy Bridge (HM76) Intel(R) HD Graphics 4000 256GB SSD Sandisk Sata III 2x4GB DDR3-1333 Kingstone CL9 Dual	-	1

Tabela 4.2: Bibliotecas utilizadas durante a programação

Nome	**Versão**	**Autor**	**Descrição**	**Plataf.**
PubSubClient	2.8.0	Nick O'Leary	Cliente para mensagens MQTT	Arduino
WiFiManager	2.0.3a	Tzapu, Tablatronix	Gerenciador de configuração WiFi com interface web.	Arduino
LoRa	0.8.0	Sandeep Mistry	Envio e recepção de dados via LoRa	Arduino
OLED Driver	4.2.0	ThingsPulse, Fabrice Weinberg	ESP8266 e ESP32 OLED driver para displays conectados ao NodeMCU	Arduino
WiFi	1.2.7	Arduino	Comunicação WiFi	Arduino
esp32	1.0.4	Espressif	Driver para placas NodeMCU	Arduino
Mosquitto	1.5.7-1		Framework para cliente MQTT (mosquitto-clients)	Linux
Mosquitto	3.1.1		MQTT message broker	Linux
Confluent-Kafka	1.7.0	Confluent	Serviço de mensageria Kafka	Anaconda

4.2 Precisão do sensor de vazão

O primeiro experimento consistiu na avaliação do comportamento do sensor de vazão, também chamado de sensor de fluxo. As especificações do fabricante[3] pedem que a tensão de operação deva ser maior ou igual a 5V, no entanto, o sensor funcionou com 3,3V fornecidos pelo NodeMCU associado a um resistor $pull-up$ na entrada do sinal do sensor para estabilização das leituras. A montagem do experimento seguiu as orientações de Romeiro [2019] e também a indicação do sentido de fluxo de água definido no próprio sensor.

Para aferir a precisão do sensor, foram executadas 20 medições utilizando 500ml de água e comparando o valor medido pelo sensor e o obtido por equipamento de medição experimental (Figura 4.1). O sensor apresentou erro de 6,79%. Fuentes and Mauricio [2020] realizaram um experimento similar, obtendo 4,63% de erro nas medições. Para o Algoritmo 1 o fatorCalibracao, segundo o fabricante, é 7.5 para sensores de 1/2"e 5.5 para sensores 3/4".

Para aferir a precisão do Algoritmo, 2 o sensor foi acoplado diretamente na saída de água da parede e o dispositivo consumidor ligado a ele por meio de um T (Figura 4.6), diferentemente do teste da primeira versão do algoritmo, onde foram utilizadas mangueiras (Figura 4.1). Foram realizadas 17 medições de 500ml. Inicialmente foi utilizado como o $fatorCalibracao$ a quantidade de pulsos indicada pelo fabricante, ou seja, 450 pulsos.

O erro encontrado usando o $fatorCalibracao$ do fabricante foi de 6,33%, muito próximo ao de 6,79% encontrado na primeira versão do algoritmo. Na prática, constatou-se que, para cada 500ml, ocorriam em média 210,75 pulsos e mediana de 211 pulsos. Com o resultado, configurou-se o $fatorCalibracao$ como 422 pulsos por litro, aumentando-se a precisão das medições.

Outra avaliação foi a da sensibilidade do sensor de vazão, que se mostrou insuficiente para detec-

[3] wiki.seeedstudio.com/G1_and_2_inch_Water_Flow_Sensor

Figura 4.1: Experimento de aferição da precisão sensor de vazão

tar pequenos vazamentos, como obter 500 ml ao longo de um minuto e treze segundos, resultando numa vazão de 0,41L/min obtidos pela Equação 4.1. Nas especificações do fabricante, o mínimo detectável é 1L/min (Tabela 4.1). A telemetria a seguir demonstra leituras mínimas muito próximas ao limite de sensibilidade do sensor, porém deve-se desconsiderar o primeiro e o último segundo da telemetria, pois nestes não há garantias de medição no iniciar e no finalizar do dispositivo fornecedor de água, resultando em vazão detectada inferior à especificada pelo fabricante.

Este teste também permitiu demonstrar que um vazamento no dispositivo consumidor, no exemplo usou-se uma torneira, que possui um consumo na direção e sentido do fluxo laminar de água na tubulação não seria detectado pelos algoritmos CNZ e MNF. Para aumentar a chance de detecção desse tipo de vazamento foi criada a cadeia de regra "Consumo X Reservatório", seção 4.8. Na seção 4.7 verificou-se que um pequeno vazamento perpendicular ao fluxo laminar é detectável.

Britton et al. [2013] também identificou dificuldade no registro do consumo de pequena vazão ao utilizar o hidrômetro inteligente Elster V100 que possui 2% de precisão e uma limitação de leitura de 3L/h. Durante esta pesquisa, encontrou-se o aparelho usado a (USD) $66,00, ou em conversão direta (BRL) R$369,60. Apesar de o equipamento usado por Britton et al. [2013] ser mais preciso que o escolhido neste trabalho, seu custo mais elevado inviabilizou sua utilização aqui.

$$vazao = \frac{quantidade(mL)}{tempo(seg)} \times \frac{60(seg)}{1000(L)} = L/min \tag{4.1}$$

15:13:53.146 − > Flow rate: 0.6L/min 11mL/Sec Output Liquid Quantity: 11mL
15:13:54.149 − > Flow rate: 1.5L/min 26mL/Sec Output Liquid Quantity: 37mL
15:13:55.152 − > Flow rate: 1.4L/min 24mL/Sec Output Liquid Quantity: 61mL
15:13:56.155 − > Flow rate: 1.4L/min 24mL/Sec Output Liquid Quantity: 85mL
15:13:57.158 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 107mL
15:13:58.161 − > Flow rate: 1.4L/min 24mL/Sec Output Liquid Quantity: 131mL
15:13:59.164 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 153mL
15:14:00.168 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 175mL
15:14:01.171 − > Flow rate: 1.4L/min 24mL/Sec Output Liquid Quantity: 199mL
15:14:02.174 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 221mL
15:14:03.177 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 243mL
15:14:04.180 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 265mL
15:14:05.184 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 287mL
15:14:06.187 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 309mL
15:14:07.190 − > Flow rate: 1.1L/min 19mL/Sec Output Liquid Quantity: 328mL
15:14:08.173 − > Flow rate: 1.3L/min 22mL/Sec Output Liquid Quantity: 350mL
15:14:09.192 − > Flow rate: 1.1L/min 19mL/Sec Output Liquid Quantity: 369mL
15:14:10.179 − > Flow rate: 1.0L/min 17mL/Sec Output Liquid Quantity: 386mL
15:14:11.167 − > Flow rate: 0.0L/min 00mL/Sec Output Liquid Quantity: 386mL

4.3 Precisão do sensor de pressão

Este experimento consistiu na avaliação do sensor de pressão. O primeiro passo para sua utilização foi medir a tensão equivalente a zero que ele emite no conector amarelo (dados) utilizando o próprio código que faz a medição da pressão. Nesta etapa, foi ligado o sensor ao NodeMCU, conforme a Figura 4.2, e medida sua tensão, onde encontrou-se 0,192 V.

Para validação experimental dos dados do sensor, foram realizadas medições *in loco* em andares diferentes de um prédio residencial, conforme indicado pela Figura 4.3. O prédio é constituído por uma casa de máquinas, seguido de seis andares habitacionais e três andares de uso coletivo (incluindo o térreo). Foram selecionados dois pontos de água de colunas diferentes de distribuição para medições, sendo eles de mesmo diâmetro. Foram desconsiderados para cálculo o tamanho da laje entre andares (aproximadamente 11cm) e altura de água na caixa d'água (aproximadamente 1,4m). A mangueira foi preenchida com água e teve sua pressão liberada para se fazer a medição inicial em cada ponto. Para validação teórica, foi utilizada a relação de 10 metros de água para 98,04KPa de pressão, levando-se em consideração que a altura da casa de máquina (H_m) é 2,1m,

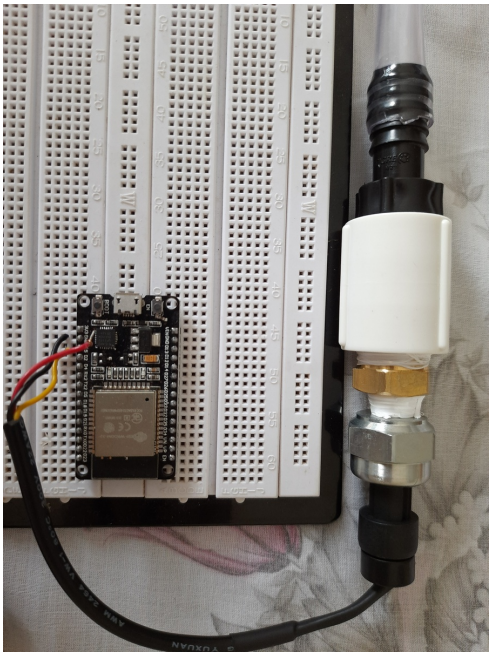


Figura 4.2: Sensor de pressão ligado ao NodeMCU

a dos andares (H_a) é 2,75m, do ponto de medição até o teto (H_p) do 2º andar é 1,6m, e do andar térreo foé 1,95m, conforme a Equação 4.2. A Tabela 4.3 apresenta os resultados, onde se constatou erro de precisão de cerca de 5%.

$$P = \frac{H_m + Andares \cdot H_a + H_p}{10} \cdot 98,04 \tag{4.2}$$

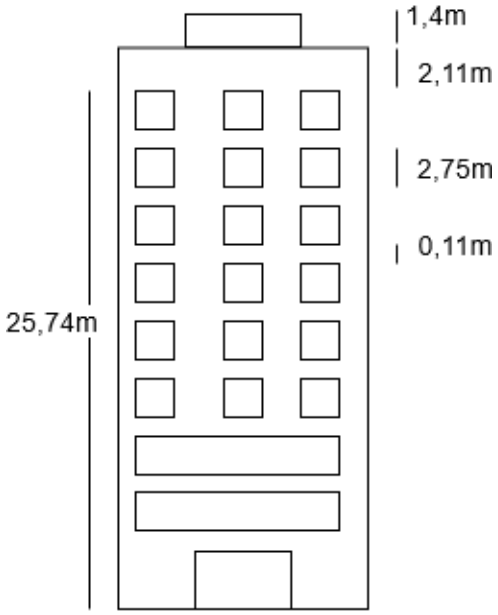


Figura 4.3: Ilustração do prédio residencial dos experimentos

Tabela 4.3: Medição prática e teórica calculada da pressão nas torneiras escolhidas

2º Andar				**Térreo**			
0,200v	3,1KPa	Teórico	Erro	0,206v	5,7KPa	Teórico	Erro
0,606v	165,5KPa	171,11KPa	3,2%	0,865v	268,7KPa	282,41KPa	**4,8**%

Tabela 4.4: Intensidade do sinal LoRa nos andares

Andar	**RSSI** (dBm)
Máquinas	-57
6	-64
5	-70
4	-85
3	-82
2	-91
1	-100
P	-96
G2	-101
T	-105

4.4 Avaliação de alcance da rede LoRa indoor

A experiência consistiu em contabilizar a quantidade de falhas na recepção e a qualidade de sinal para pacotes enviados sequencialmente. Assim como feito no trabalho de Khutsoane et al. [2017], foi verificado, na prática, que a posição horizontal da antena apresenta melhor desempenho. Não foi identificado, no código da fabricante, recursos de segurança na transmissão dos dados. Neste experimento, foram usados apenas os dois NodeMCU com LoRa.

4.4.1 Alcance no prédio

O primeiro ambiente escolhido foi a escada de emergência, onde o emissor ficou no chão do andar da casa de máquinas e medições foram feitas em cada andar imediatamente abaixo do emissor.A Tabela 4.4 mostra a qualidade do sinal recebido, medido pelo Received Signal Strength Indication (RSSI), em dBm. Neste experimento, quanto menor o número, maior é a perda. A sensibilidade máxima é -139dBm [LILYGO, 2023]. Não houve perda de pacotes neste experimento e a distância testada chegou à 25,74 metros. A título de comparação Khutsoane et al. [2017], encontrou resultados similares na frequência de 868Mhz alcançando 60 metros *indoor*.

4.4.2 Alcance em residência

O segundo teste foi realizado dentro de uma unidade habitacional com $96m^2$ do prédio residencial (Figura 4.3). Neste caso, o emissor ficou no extremo do apartamento, no comodo externo (varanda). Posteriormente, ele foi caminhando para dentro do apartamento, fechando janelas de vidro, portas e aumentando a quantidade de obstáculos que poderiam de alguma forma interferir no sinal. Não

Tabela 4.5: Intensidade do sinal LoRa dentro do apartamento

Local	**RSSI (dBm)**
Quarto	-60
Banheiro	-70
Sala	-72
Área Serv.	-81
Quarto 2	-84
Banheiro 2	-85

houve perda de pacotes neste teste. O último item da Tabela 4.5 possui a maior distância entre os pares NodeMCU. Vale ressaltar que o ambiente avaliado é densamente permeado por sinais de rede Wi-Fi 802.11 em 2,4 GHz e 5 GHz implicando em possível incidência de interferências e perda de desempenho para este tipo de rede. Nota-se que, apesar de as redes Wi-Fi operarem em frequências diferentes do LoRa (915MHz), não foi realizado nenhum estudo para verificar se interferências não estão realmente ocorrendo.

Experimentos iniciais de adaptação do código detecção de vazamentos para uso de conexão LoRa ponto-a-ponto foram realizados. A primeira dificuldade verificada foi o acesso à servidores de internet Network Time Protocol (NTP) para obtenção do horário para funcionamento do algoritmo MNF. Também verificou-se o recebimento de telemetrias de dispositivos que não pertenciam ao experimento. Desta maneira, seria necessário grande esforço para recriar sistemas de segurança já existentes em roteadores LoRa com protocolo LoRaWan. Devido ao custo ainda elevado para proposta, optou-se que a implementação dessa tecnologia seja avaliada em um momento futuro. Um roteador LoRaWan Dragino LPS8 uso interno custa BRL R$ 998 por importação direta, novembro de 2022.

4.5 Protocolo de envio de mensagens MQTT

O teste inicial utilizou o *framework* Mosquitto[4] no Linux no Raspberry Pi. As bibliotecas utilizadas estão relacionadas na Tabela 4.2. Foram abertos dois clientes MQTT, um *pusblisher* e um *subscriber*, mais o *broker*. Cada um deles foi executado em um terminal diferente, e foram realizadas as trocas de mensagens.

Na etapa seguinte, o Raspberry Pi (Mosquitto) continuou como *broker*, porém teve-se a adição do NodeMCU à troca de mensagens. Tanto o Raspberry Pi quanto NodeMCU assumiram os papéis de *publisher* quanto de *subscriber*. Os dispositivos foram conectados por meio de um roteador sem fio. O envio de mensagens do Raspberry para NodeMCU e vice-versa foram realizados.

Um experimento em ambiente real foi realizado, a fim de avaliar seu funcionamento com envio contínuo de telemetria. A Figura 4.6 mostra sua instalação na torneira de um tanque na área de

[4]mosquitto.org

serviço de um apartamento residencial. Relembrando, o Algoritmo 7 é o utilizado para o teste envolvendo os sensores de vazão e pressão. Notou-se que a vazão estava com valores não reais. Observou-se que para o funcionamento correto do sensor de vazão no NodeMCU, a porta 2 não pode ser utilizada pelo sensor, visto que ela também é utilizada pelo chip Wi-Fi e isso afeta as medições.

4.6 Algoritmos de detecção de vazamentos

Os algoritmos de detecção de vazamento foram implementados em uma arquitetura em borda, ou seja, próximo dos nós sensores. Essa abordagem vem se mostrando cada vez mais interessante devido à eficiência energética, capacidade de processamento e quantidade de memória de placas eletrônicas cada vez menores.

O Consume Non Zero (CNZ) avalia se houve consumo nulo em uma janela de tempo. A validação desta técnica foi realizada em ambiente real e pode ser vista na telemetria a seguir. O tempo de análise foi reduzido para dois minutos para validação, mas prevê-se uma janela de uma hora para residências.

```
20:06:47.299 -> WiFi connected
20:06:47.352 -> Conectando ao broker MQTT...
20:06:47.453 -> Conectado ao broker!
20:07:46.764 -> Leituras pressão/vazão updated
20:08:47.035 -> Leituras pressão/vazão updated
20:08:47.182 -> Vazamento detectado! Nas últimas 24h você não teve consumo zero
em nenhum momento.
20:09:47.449 -> Leituras pressão/vazão updated
20:10:47.558 -> Vazamento consertado!
20:11:47.636 -> alive = true enviado!
```

Já o MNF, como mencionado, entende que não há consumo durante o período da madrugada, devido às pessoas estarem dormindo. A seguir, apresenta-se a validação dessa técnica e o alerta criado no Thingsboard. Uma vez criado o alerta no Thingsboard, é preciso confirmá-lo ou negá-lo. Se novos alertas do mesmo tipo vierem posteriormente, sem a devida ação de confirmação ou exclusão do anterior, o primeiro alerta permanecerá ativo e não será sobreposto. A Figura 4.4 mostra o alerta criado decorrente da validação. Nesse caso, foi utilizado um intervalo de tempo de 4 minutos durante a madrugada.

```
02:00:04.120 -> WiFi connected
```

```
02:00:04.167 -> Conectando ao broker MQTT...
02:00:04.521 -> Conectado ao broker!

02:01:03.597 -> Wednesday, February 09 2022 01:01:03
02:01:03.650 -> Detecção de Vazamento por MNF iniciada!
02:02:03.760 -> vazaoMNF atualizada! regra 1
02:02:03.965 -> Enviando {"flow_min":2247, "pressure":141.696091}
02:03:04.174 -> vazaoMNF atualizada! regra 2
02:03:04.390 -> Enviando {"flow_min":882, "pressure":166.832809}
02:04:04.624 -> Wednesday, February 09 2022 01:04:04
02:04:04.671 -> Detectado vazamento por MNF. Alerta enviado para ThingsBoard!
```

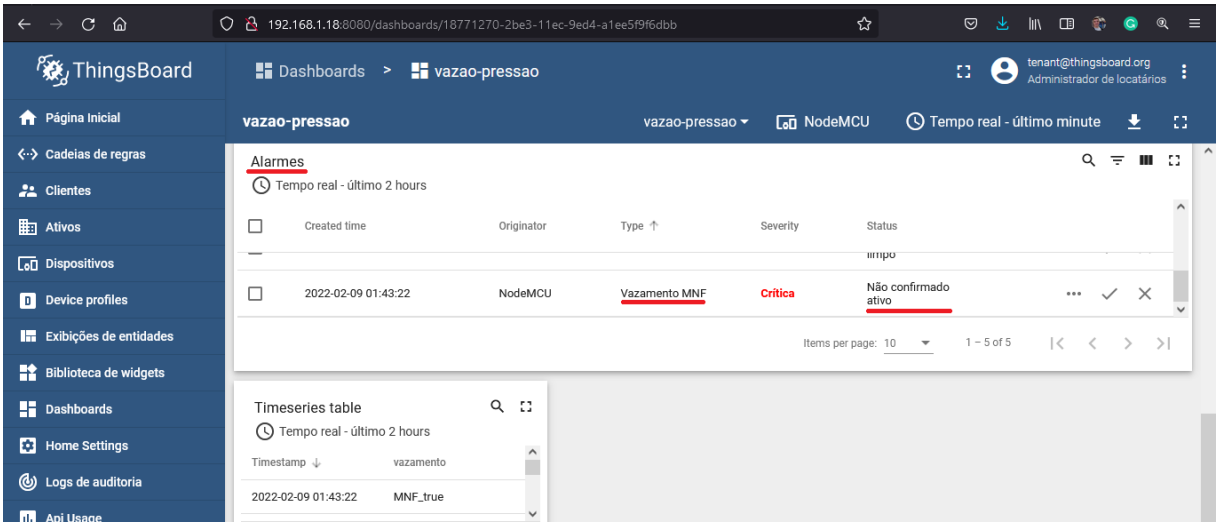


Figura 4.4: Alerta de vazamento por MNF recebido pelo Thingsboard

Com os dois algoritmos validados, passou-se a avaliar o quanto tais algoritmos impactam na solução de computação em borda, visto que tais dispositivos tem uma limitação de fonte de energia. *Dashboards* são representações gráficas dos dados recebidos em tempo real ou armazenados. O Thingsboard permite o uso de diferentes gráficos. Dentre eles o mais comum em IoT é o linha temporal (*time series*) onde os dados são apresentados na grandeza física recebida organizado pela data e hora de recebimento.

Outro tipo de *dashboard* é a hierarquia de dispositivos que mostra ativos; exemplo casa, banheiro, cozinha, etc; e dispositivos associados. Também há a seção de alertas que mostra os alertas criados pelo sistema de acordo com a regras criadas. É importante não mostrar telemetrias de ordens de

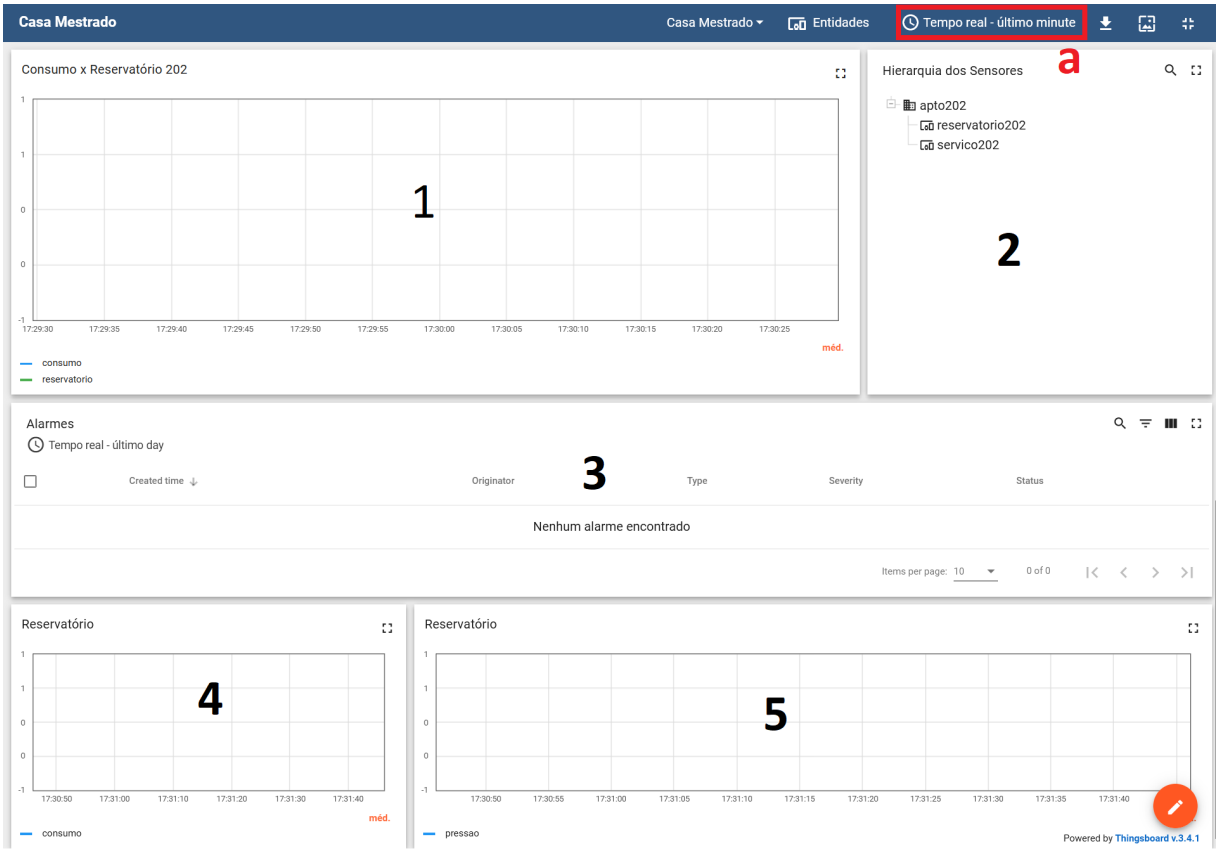


Figura 4.5: Dashboard de monitoramento do consumo de água da residência

grandeza diferentes no mesmo gráfico, pois há risco de uma delas desaparecer. Na Figura 4.5 é possível verificar o *dashboard* criado onde os itens: 1, 4 e 5 são do tipo linha temporal, 3 do tipo histórico de alarme e 2 o de hierarquia de sensores e ativos. O item *a*, em vermelho, permite a escolha do tempo que deseja-se mostrar os dados, em tempo real ou período determinado.

Na Figura 4.5 o item 1 mostra um comparativo de consumo de água com o fornecido pelo reservatório para facilitar a visualização de alguns tipos de vazamento. Item 4 mostra o consumido pelo reservatório e o item 5 a pressão na tubulação próximo ao reservatório.

4.6.1 Eficiência energética

O experimento consistiu em avaliar os algoritmos de detecção de vazamento (CNZ e MNF) em ambiente real para averiguar a eficiência energética dos mesmos. O microcontrolador operou em 80MHz. Foi utilizado um *powerbank* de 10.400 mAh como fonte de energia. O ambiente usado foi a torneira de um tanque de um apartamento residencial (Figura 4.6). Foram usados os sensores de vazão e pressão ligados ao NodeMCU que realizou comunicação Wi-Fi com o Thingsboard usando protocolo MQTT.

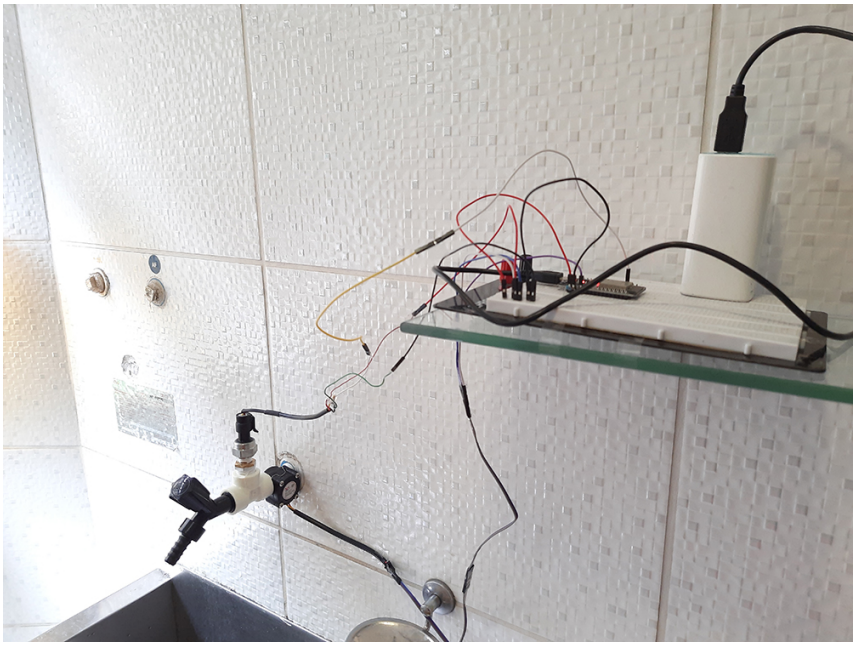


Figura 4.6: Teste de envio contínuo de telemetria em ambiente real

O Algoritmo 7 foi a primeira versão criada. A cada segundo, foi lida a vazão e calculado o consumo. Ao final de 60 segundos acrescentou-se dados de pressão que foram enviados pro ThingsBoard independente do valor, mesmo que fosse zero. Os resultados mostraram que, em poucos momentos, houve consumo no período do experimento como mostra a Figura 4.7, onde os dados de vazão por minuto são vistos em verde e foram gerados nos *dashboards* do Thingsboard.

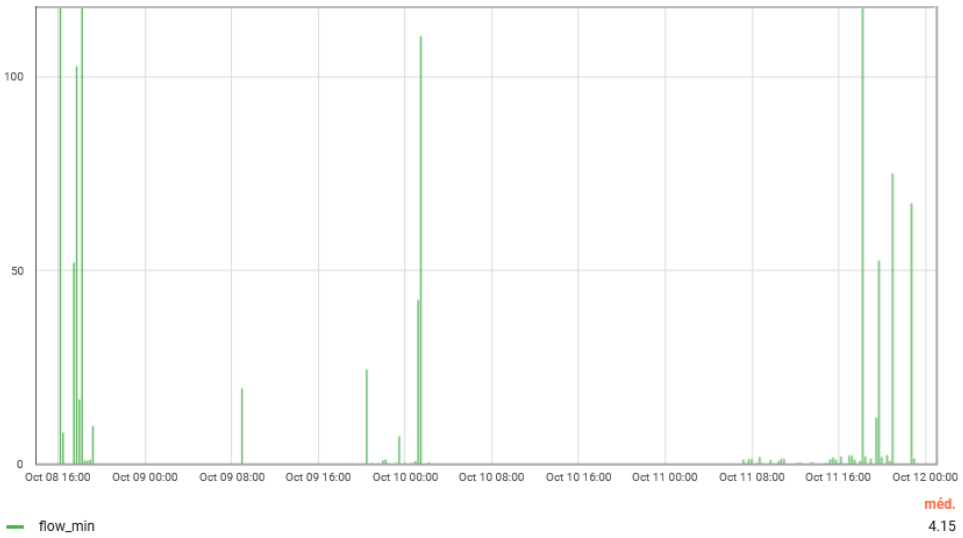


Figura 4.7: Gráfico do envio contínuo de telemetria de vazão em ambiente real

O período de análise se iniciou às 15:52 h de 08/10/2021 e teve sua última telemetria, devido ao esgotamento do *power bank*, enviada às 23:07 do dia 11 do mesmo mês totalizando 3 dias e 7 horas de experimento, onde foram enviadas um pouco mais de 4.740 telemetrias. Com os poucos dados diferentes de zero enviados, buscou-se otimizar o envio de telemetria a fim de poupar energia

do nó sensor, assim como economizar espaço no banco de dados no nó Thingsboard. Para isso, foi criado um sinal de *alive* responsável por enviar mensagem de tempos em tempos, avisando que o dispositivo está funcionando.

Com a simplificação obtida com a segunda versão do algoritmo de medição de consumo e vazão (Algoritmo 2), somados à implementação dos Algoritmos 10 e 9 de detecção de vazamentos, e otimização do envio de mensagens, o teste de eficiência energética foi repetido. Desta vez o mesmo *powerbank* durou até 4 dias e 13 horas, um aumento de 37,97% na vida útil da bateria ou, em média, 4 dias 7 horas e 20 minutos. A Tabela 4.6 mostra o resumo comparativo dos algoritmos, suas funções e eficiência energética.

	Algoritmo 7	Algoritmo 8
Recursos	Envio de telemetria vazão e pressão a todo minuto	Envio de telemetria: vazãoMédia, consumo, pressão, alive e flag vazamento; Detecção MNF e CNZ; LEDs indicativos de funcionamento; Envio apenas de telemetria não nula; Simplificação da medição de vazão
Vida Útil Bateria	3 dias e 7 horas	4 dias e 13 horas
Economia		**37,97%**
Telemetrias	4730	220
Redução percentual da telemetria		**95%**

Tabela 4.6: Comparação de funções implementadas e eficiência energética

Apesar de não explicado anteriormente, no Algoritmo 8 foi possível a aplicação da técnica de economia de energia *Duty Cycle* de janela fixa [Carrano et al., 2014]. Esta técnica consiste em reduzir as operações realizadas pelo microcontrolador, sendo implementada por meio da função *delay()*. Não foi realizada aferição de impacto no aumento da vida da bateria causado unicamente pelo *duty cycle*.

4.7 Detecção de pequenos vazamentos

Objetivou-se analisar se o sistema proposto detectaria um vazamento de um furo perpendicular ao fluxo laminar de água na tubulação. Para tal, um furo de 4mm de diâmetro foi feito como mostra a Figura 4.8. Mantendo-se a pressão original do sistema em média 173 kPa, o fluxo foi tão forte que seria capaz de ser detectado pelos algoritmos implementados na borda, CNZ e MNF.

Para avaliar a detecção em sistemas com pressão menor, a pressão necessitava ser reduzida de alguma forma. Para tanto, o registro de gaveta da tubulação foi aberto com menos de 1/4 de volta. Sucessivas medições foram feitas realizando obstruções no furo, fechando o registro do experimento

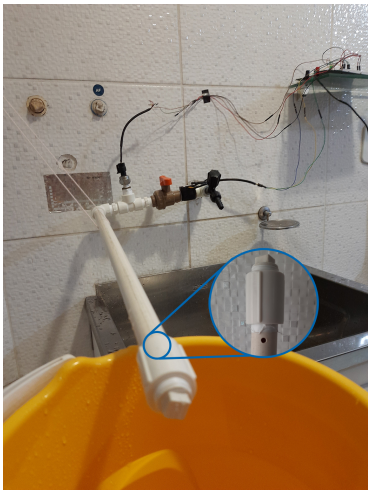


Figura 4.8: Experimento para teste de pequenos vazamentos

Medição da vazão (l/min)	
Manual	0,379
Sensor	0,23
Medição da pressão (kPa)	
Sem consumo	20,2
Durante consumo	3,58

Tabela 4.7: Detecção de pequenos vazamentos

e analisando-se os valores de pressão. A média desses valores resultou em uma pressão de 20,2 kPa.

Foi possível detectar um fluxo contínuo de água, porém houve um erro de 39,31% entre a vazão detectada pelo sensor e a vazão real medida. Em ambos os casos, a vazão detectada ficou abaixo da sensibilidade de 1 l/min especificada no *datasheet* [Seeed, 2020]. Mas, neste caso o resultado da detecção de vazamento não seria prejudicado, tendo em vista que a capacidade de captura de consumo constante é suficiente para ativação de alertas de vazamento. A Tabela 4.7 apresenta os dados do experimento.

4.8 Regra do Reservatório X Consumo

Uma limitação da arquitetura proposta consiste no fato de que cada nó é responsável somente pelos seus dados, não realizando compartilhamento da informação com outros nós. Assim, para avaliar o consumo baseado no reservatório, foi necessária uma implementação dos algoritmos em uma arquitetura em névoa. Sua análise é feita pelo ThingsBoard, pois é o responsável por receber todas as telemetrias. Assim, a ideia desta abordagem é comparar a água fornecida pelo reservatório na última hora com o registrado pelos sensores de consumo no mesmo período, sendo a análise baseada nos valores fornecidos pelos sensores de vazão.

Esta abordagem tem por objetivo mitigar a menor sensibilidade dos sensores de baixo custo nas seguintes situações: vazamentos menores que somados ao consumo dos moradores tornam-se detectáveis; e pequenos vazamentos nos diversos ramais consumidores que, quando são somados, entram na faixa de sensibilidade do sensor do reservatório. Por fim, detectar um vazamento entre o reservatório e ramais consumidores.

O algoritmo, neste caso, é baseado na Regra de Cadeia (*Rule Engine*) do ThingsBoard. Para utilizar essa regra, torna-se necessário configurar o atributo *tipoDispositivo* como *server side* em cada nó sensor que assumirá valor *consumidor* ou *reservatorio* na plataforma IoT. Também é necessário configurar a relação de contido em um *Ativo* (*Asset)*, por exemplo: Casa. A Figura 4.9 mostra seu funcionamento e a explicação é vista a seguir.

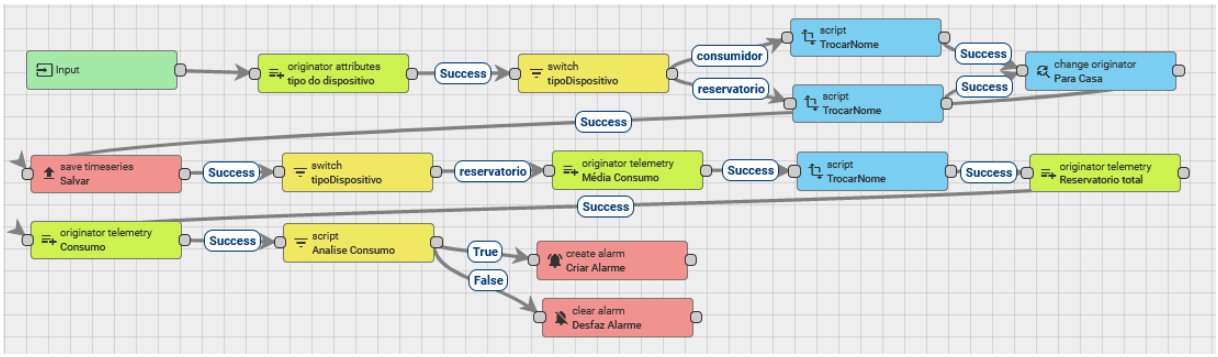


Figura 4.9: Regra de Detecção de Vazamento Reservatório x Consumo

O Thigsboard fornece quatro tipos de nós usados na programação da regra, são eles: *Filtros*, responsáveis por filtrar a telemetria recebida; Enriquecimento, que adicionam atributos dos ativos e dispositivos à telemetria; *Transformação*, que possibilitam cálculos e mudanças na telemetria; *Externo*, para comunicação com sistemas externos ao ThingsBoard; e, por fim, *Ação*, que proporcionará à regra ações decorrentes do processamento da telemetria.

Na Figura 4.9, a telemetria recebida é enriquecida com o atributo tipoDispositivo, e então é segregada para haver a troca do nome do campo da telemetria para ser atribuída ao ativo Apto202. Após salvar a telemetria no Ativo, a cada nova telemetria do reservatório, é calculada a média de consumo do Ativo. Calcula-se o consumo total e o volume fornecido pelo reservatório. Com esses dados reunidos, roda-se um *script* que avalia se o fornecido pelo reservatório menos o consumido é maior que a média do consumo, todos referindo-se à última hora. Teoricamente, o consumido é igual ao fornecido pelo reservatório, porém as telemetrias não são síncronas podendo ocorrer de o Raspberry Pi não receber a telemetria de um consumidor a tempo da análise. Por este motivo, o valor calculado no *script* é comparado à média de consumo ao invés de zero.

4.9 Estudo do Hábito de Consumo

Analisar os hábitos dos moradores é uma outra forma de auxiliá-los na redução do consumo. Estas análises são realizadas pelo ThingsBoard retornando alertas para os moradores. A seguir, encontram-se duas regras baseadas em estudos encontrados durante a pesquisa.

4.9.1 Alarme de Pressão

A partir do trabalho de Zhao et al. [2018], que verificou na prática a implementação da norma chinesa de conservação de água GB 50015-2003 (2009), que impacta diretamente na vazão de água pelos dispositivos consumidores, foi implementado um alerta quando a pressão excede 200 kPa. Automaticamente, esse alerta é desfeito quando a pressão encontra-se em níveis abaixo do valor mencionado. A configuração do alarme foi feita no perfil do dispositivo no ThingsBoard.

4.9.2 Consumo Diário

Esta regra visa comparar o consumo da residência com a média de sua região e alertar quando este está acima. A regra leva em consideração a quantidade de moradores e a região do país. Para sua implementação, é necessário configurar os atributos *server side moradores* e *regiao* no Ativo Casa. Esta é uma contribuição original desta pesquisa. A Figura 4.10 mostra a regra e sua explicação vem a seguir.

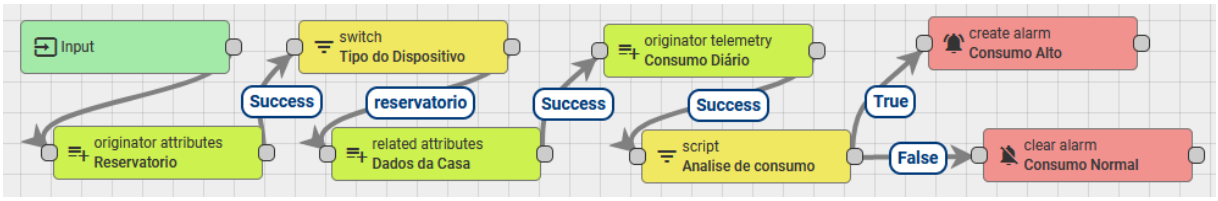


Figura 4.10: Regra do Consumo Diário

Para seu funcionamento, a telemetria é enriquecida com o atributo *server side tipoDispositivo*. Somente quando há um recebimento de dados do sensor do reservatório, a telemetria é enriquecida com dados *moradores* e *regiao* do Ativo Casa. Os dados seguem para cálculo do consumo diário, onde o ThingsBoard soma volume fornecido pelo reservatório nas últimas 24 horas. Então, a telemetria é tratada com o recurso *JSON.parse* e analisada pelo *script* que compara o resultado com dados do Sistema Nacional de Informações Sanitárias (SNIS). Se o consumo for acima do valor de referência, um alarme de aviso é criado e mostrado ao usuário. Ao reduzir o consumo, o alarme é desfeito automaticamente. A Tabela 4.8 mostra o consumo médio por região usado.

Tabela 4.8: Consumo Diário per Capita

Região	**Consumo** (L/dia/hab)
Norte	131,8
Nordeste	115,4
Sudeste	182,6
Sul	146,1
Centro-Oeste	144,9
Brasil	154,9

Fonte: MDR [2018]

4.10 Comportamento da pressão durante o consumo

Neste experimento, buscou-se analisar o comportamento da pressão durante o consumo e investigar possíveis contribuições da analise de pressão na detecção de vazamentos. O teste em ambiente real foi montado com possibilidade de medição de pressão em diferentes pontos com variadas distâncias do ponto de consumo.

Foram realizados consumos nos pontos P2 e P7, ou seja, com o fluxo de água passando pelo sensor de pressão e não passando. Medições de pressão foram feitas nos pontos P2, P4, P5 e P6. A Figura 4.11 mostra o experimento montado e suas especificações são vistas na Figura 4.12. Na primeira rodada, as medições foram realizadas a cada 10 segundos e são apresentadas nas Tabelas 4.11 e 4.12 . Novas medições foram realizadas a cada 5 segundos para melhor captura dos dados e confirmação dos resultados iniciais, os quais são apresentados a seguir.

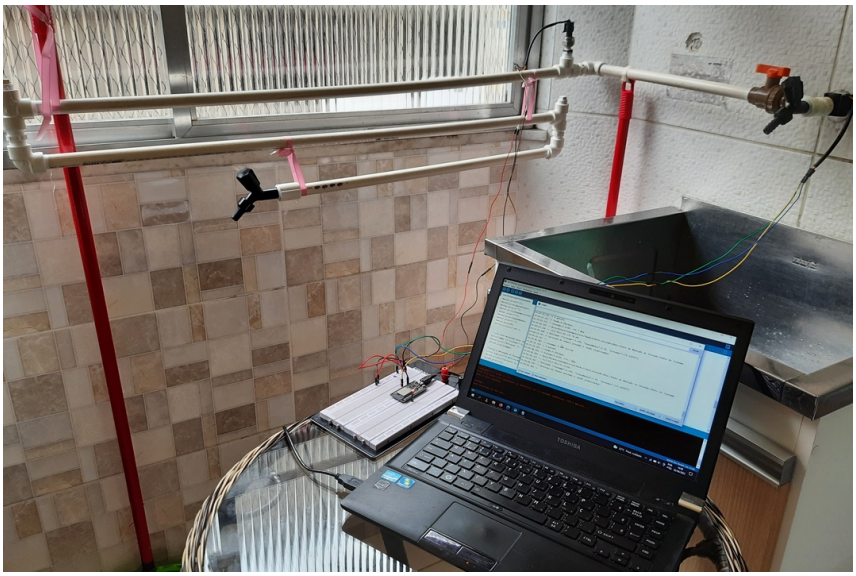


Figura 4.11: Experimento comportamento da pressão durante o consumo

A pressão média inicial sem uso ficou em 172kPa. O experimento visou o teste de diferentes vazões variando a abertura da torneira de consumo começando por vazões menores, fechando e então vazões maiores. A repetição da vazão para melhor comparação foi dificultada dada a baixa

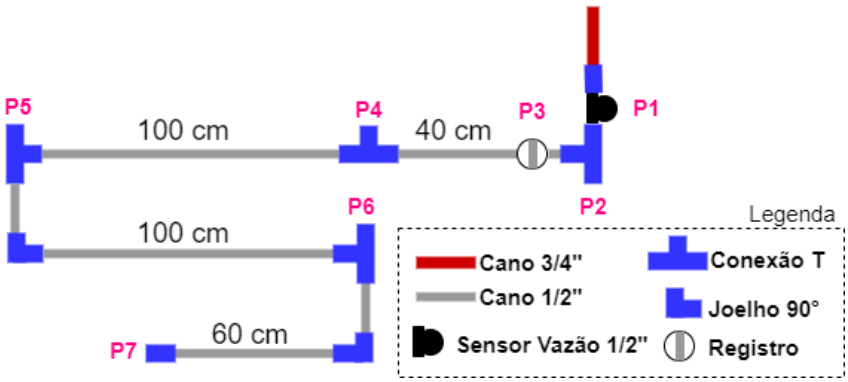


Figura 4.12: Esquema de montagem do experimento do estudo do comportamento da pressão durante o consumo

precisão do dispositivo torneira utilizado.

Dadas as condições do experimento, uma vazão de 3L/min com ponto de consumo antes ou depois do ponto de medição da pressão, não houve variação significativa da pressão. Esse comportamento se repetiu na segunda rodada de experimentos que teve uma vazão de até 4,18 L/min.

Quanto mais próximo o sensor de pressão do dispositivo consumidor e mais distante do ramal distribuidor de maior diâmetro, maior foi a variação de pressão. Esse comportamento é verificado com os dados da Tabela 4.9, extraídos das Tabelas 4.11/4.12. O comportamento se repetiu parcialmente na segunda rodada de experimentos, porém os pontos P5 e P6 tiveram varições parecidas de pressão. Dessa forma, é observado que a variação de pressão devido ao consumo se perde conforme a distância aumenta entre o dispositivo consumidor e o ponto de sensoriamento da pressão.

Pressão					**Consumo**
Inicial (kPa)	**Consumo (kPa)**	**Variação**	**Distância (cm)**	**Ponto**	**Vazão (L/min)**
163,93	146,85	10,41%	300	P2	12,75
175,20	143,3	18,2%	260	P4	7,33
176,17	136,21	23,11%	160	P5	7,41
173,60	125,91	27,47%	60	P6	7,46

Tabela 4.9: Comportamento da pressão (multipontos) x consumo (P7)

Também observou-se, em ambos os experimentos, que o ponto P2 não se mostrou eficiente para medição da variação de pressão. A Tabela 4.9 mostra que mesmo uma vazão bem maior obteve uma variação próxima de 1/3 da pressão em outros pontos com vazão menor. O fato pode ser explicado devido ao sensor estar na direção do fluxo de água e estar próximo ao ramal distribuidor de maior diâmetro capaz de fornecer maior vazão de água. Desta maneira, não é recomendada a medição da pressão nessas condições.

Um fenômeno diferente do esperado ocorreu em algumas poucas medições: o aumento de pressão devido ao consumo. A esse fenômeno dá-se o nome de Golpe de Aríete Bistafa [2010]. Esse fenômeno

foi observado pelo sensor de pressão, assim como visualmente no experimento pelo balançar da tubulação que estava exposta e não fixa.

O Golpe de Aríete é um fenômeno hidráulico onde o fluido dá "marteladas" no interior da tubulação, se chocando contra tubos, válvulas e conexões. Esse comportamento é típico de escoamentos turbulentos. Geralmente, decorre de obstrução do fluido como fechamento abrupto ou rápida abertura do registro ou ainda pelo acúmulo de ar dentro do fluido causando bolsões compressíveis [Bistafa, 2010].

Na primeira rodada de experimentos, o fenômeno foi capturado na fechamento abrupto dos dispositivos fornecedores de água, onde a pressão passou de 169,09kPa para 189,71kPa, aumento de 12,19% e na abertura abrupta dos mesmos dispositivos, com variação de 171,66kPa para 221,30kPa, aumento de 28,91% (Tabela 4.12). O mesmo fenômeno foi capturado uma vez no segundo grupo de experimentos (Tabela 4.10) também no fechamento abrupto da torneira com variação de pressão de 171,02kPa para 203,89kPa, aumento de 19,21%. Interessante notar que a pressão se estabilizou, 15 segundos após o fechamento completo, em 171,02kPa (Tabela 4.10).

		Consumo		**Pressão**		**Consumo**
Hora	**Pulsos**	**Litros**	**Vazão (L/min)**	**Valor (kPa)**	**Ponto de Medição**	**Ponto de Medição**
15:18:50.393	0	0,00	0,00	171,02	P4	P2
15:18:55.459	0	0,00	0,00	154,59		
15:19:00.474	244	0,59	7,04	138,15		
15:19:05.530	358	0,86	10,33	178,76		
15:19:10.593	13	0,03	0,37	203,89		
15:19:15.654	0	0,00	0,00	176,18		
15:19:25.737	0	0,00	0,00	171,02		

Tabela 4.10: Dados brutos de comportamento da pressão durante o consumo - 2ª rodada

	Consumo		Pressão		Consumo
Hora	**Pulsos**	**Litros**	**Vazão (L/min)**	**Valor (kPa)**	**Ponto de Medição**	**Ponto de Consumo**
11:52:38	0	0,00	0,00	169,41	P4	P2
11:52:48	0	0,00	0,00	158,13		
11:52:58	0	0,00	0,00	175,85		
11:53:08	0	0,00	0,00	172,95		
11:53:18	105	0,25	1,51	169,73		
11:53:28	210	0,51	3,03	175,85		
11:53:38	209	0,50	3,01	171,34		
11:53:48	172	0,20	1,18	172,63		
11:54:48	569	1,368	8,21	160,70		
11:54:58	48	0,12	0,69	174,20		
11:57:18	167	0,40	2,41	165,80	P4	P7
11:57:28	324	0,78	4,67	171,70		
11:57:38	119	0,29	1,72	176,50		
11:57:48	0	0,00	0,00	175,20		
11:58:38	43	0,10	0,62	143,30		
11:58:48	508	1,221	7,33	172,31		
11:58:58	0	0,00	0,00	179,07		
12:30:46.220	0	0,00	0,00	175,86	P5	P2
12:31:16.264	17	0,04	0,25	171,67		
12:31:26.328	299	0,72	4,31	175,53		
12:31:36.307	284	0,68	4,10	185,52		
12:31:46.334	14	0,03	0,20	179,08		
12:31:46.381	0	0,00	0,00	181,01		
12:33:26.528	417	1,00	6,01	169,09		
12:33:36.559	134	0,32	1,93	189,39		
12:33:46.591	0	0,00	0,00	189,71		
12:34:26.629	106	0,26	1,53	176,50	P5	P7
12:34:36.659	221	0,53	3,19	174,24		
12:34:46.690	212	0,51	3,06	172,96		
12:34:46.743	61	0,15	0,88	172,96		
12:35:06.717	0	0,00	0,00	176,18		
12:36:06.820	514	1,24	7,41	136,22		
12:36:16.851	126	0,30	1,82	175,86		
12:36:26.882	0	0,00	0,00	172,31		

Tabela 4.11: Dados brutos de comportamento da pressão durante o consumo - 1ª rodada

	Consumo			Pressão		Consumo
Hora	**Pulsos**	**Litros**	**Vazão (L/min)**	**Valor (kPa)**	**Ponto de Medição**	**Ponto de Consumo**
13:53:00.604	0	0,00	0,00	173,60	P6	P2
13:53:10.599	0	0,00	0,00	171,67		
13:53:20.640	101	0,24	1,46	170,70		
13:53:30.624	246	0,59	3,55	168,44		
13:53:40.674	244	0,59	3,52	172,31		
13:53:50.662	7	0,02	0,10	172,63		
13:54:00.694	0	0,00	0,00	171,67		
13:55:00.796	2	0,01	0,03	181,01		
13:55:10.812	217	0,52	3,13	165,54		
13:55:20.855	400	0,96	5,77	176,50		
13:55:30.879	1	0,00	0,01	170,06		
13:55:40.874	0	0,00	0,00	176,82		
13:56:10.934	372	0,89	5,37	160,71		
13:56:20.987	237	0,57	3,42	171,67		
13:56:30.989	0	0,00	0,00	172,96		
13:58:41.211	208	0,50	3,00	171,34	P6	P7
13:58:51.253	171	0,41	2,47	172,96		
13:59:01.287	177	0,43	2,55	173,92		
13:59:11.271	82	0,20	1,18	172,31		
14:00:21.399	566	1,36	8,16	187,14		
14:00:31.453	62	0,15	0,89	176,18		
14:00:41.481	0	0,00	0,00	173,60		
14:01:21.557	517	1,24	7,46	125,91		
14:01:31.538	138	0,33	1,99	172,96		
14:01:41.572	0	0,00	0,00	174,24		
14:13:50.405	0	0,00	0,00	171,67	P2	P7
14:13:55.405	50	0,12	1,44	221,30		
14:14:00.467	89	0,21	2,57	165,54		
14:14:05.483	91	0,22	2,62	170,38		
14:14:10.506	91	0,22	2,62	173,60		
14:14:15.524	95	0,23	2,74	170,38		
14:14:20.539	87	0,21	2,51	176,18		
14:14:25.555	88	0,21	2,54	172,96		
14:14:30.570	26	0,06	0,75	179,40		
14:14:35.586	0	0,00	0,00	163,93		
14:15:05.709	92	0,22	2,65	146,85		
14:15:10.724	442	1,06	12,75	152,33		
14:15:15.743	64	0,15	1,85	175,86		
14:15:25.775	0	0,00	0,00	170,70		

Tabela 4.12: Dados brutos de comportamento da pressão durante o consumo - 1ª rodada

4.11 Impacto do registro no consumo

Neste experimento, buscou-se avaliar o impacto do grau de abertura do registro no consumo de água. Foram realizadas sucessivas medições durante o regime permanente de consumo (ao menos cinco medições) e encontrada a média da vazão de água e pressão de uma torneira que teve sua aber-

tura inalterada. Em outras palavras, foram descartadas as medidas durante abertura e fechamento da torneira e do registro. Foi utilizado um registro do modelo borboleta de 1/4"de volta mantendo-o 100% aberto e metade aberto. A Figura 4.13 mostra o experimento realizado. A torneira usada é a mais à esquerda e o registro mencionado é o de cor marrom com laranja.

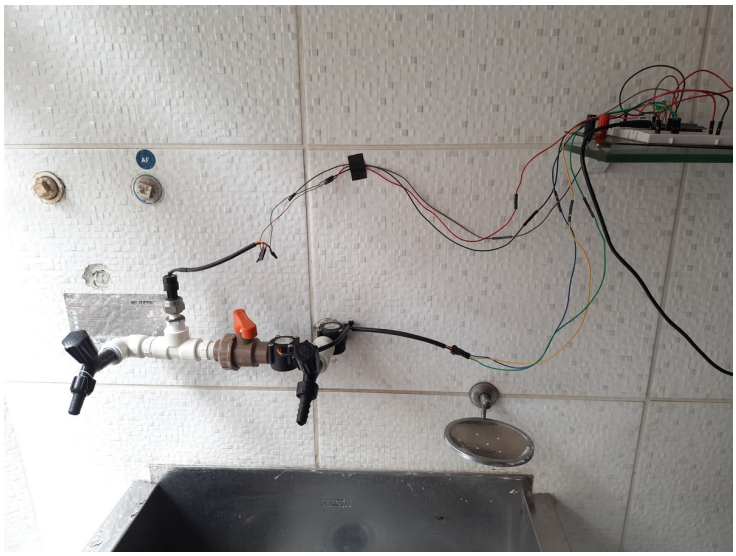


Figura 4.13: Experimento de impacto no consumo com registro aberto pela metade

Os resultados mostraram redução da vazão e pressão durante o consumo com o registro aberto pela metade. O consumo obteve uma redução média de 16,75% e a pressão de 22,45%. A pressão sem consumo ficou igual ao valor da tubulação antes do registro levando menos de cinco segundos para estabilização do valor. Os dados comprovam que uma medida simples para redução do consumo de água é a abertura parcial do registro do cômodo. Os valores medidos são expostos na Tabela 4.13.

	Totalmente Aberto	**Registro Abertura pela metade**	**Diferença**
Pressão inicial	194,54 kPa		
Pressão	173,09 kPa	131,55 kPa	- 24%
Vazão	10,7 l/min	8,67 l/min	- 18,97%
Pressão inicial	194,73 kPa		
Pressão	174,36 kPa	137,93 kPa	- 20,9%
Vazão	9,36 l/min	7,98 l/min	- 14,8%

Tabela 4.13: Experimento análise do impacto da abertura do registro no consumo

4.12 Aprendizado de máquina

Para implementação do algoritmo de Regressão Linear, optou-se pelo uso do programa Anaconda com a linguagem Python. As telemetrias recebidas são tratadas e agrupadas por dia e localização

dos dispositivos, calculando-se o consumo por dia. O modelo de aprendizado de máquina é ajustado levando em consideração os diferentes consumos por dia da semana. A Figura 4.14 ilustra o resultado da aplicação desta técnica sobre os dados. Dia 0 é segunda-feira, dia 1 é terça-feira e assim por diante. Com passar dos dias, em ambiente real, o modelo terá mais pontos de consumo no mesmo dia da semana.

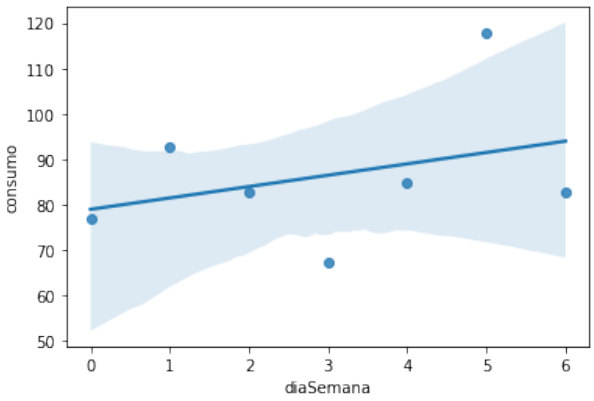


Figura 4.14: Regressão Linear utilizada para predição de consumo

Para o recebimento dos dados pelo algoritmo de aprendizado de máquina, utilizou-se o Kafka, uma plataforma que permite trabalhar de forma simples com *streaming* de dados, permitindo que os dados sejam manipulados, armazenados e processados de acordo com o fluxo temporal.

Um *Jupyter Notebook*, como é chamado a aplicação desenvolvida no ambiente Anaconda, fica responsável por receber a telemetria, tratá-la e salvá-la em um arquivo Comma-separated values (CSV). Este é o *consumidor*. Um segundo *Jupyter Notebook* faz o agrupamento das telemetrias salvas no arquivo CSV e executa o algoritmo de aprendizado de máquina. A aplicação criada é utilizada para prever o consumo futuro. A partir deste valor, é possível comparar com os valores obtidos, Como já mencionado, outras técnicas de ML podem ser implementadas para se detectar vazamento, como técnicas de identificação de anomalias e etc. No entanto, para a estrutura atual do trabalho, a regressão linear já fornece informações suficientes tanto identificar dados que apresentem uma variância muito significativa em relação ao valor previsto pelo modelo.

A telemetria recepcionada passou por um pré-processamento para seleção dos dados de interesse, e, posteriormente, foram apresentados ao algoritmo de aprendizado de máquina para predição do consumo mensal e valor da conta de água/esgoto. Os valores são vigentes pela concessionária de água de cidade do Rio de Janeiro/RJ - Brasil em agosto de 2022.

Comparando-se o custo da solução implementada com o módulo Thingsboard Trendz Analytics[5] oferecido pela fabricante da plataforma, nota-se a obtençao de uma solução mais barata e que pode ser incrementada com novas funcionalidades. Em contra partida, perde-se nos quesitos de integração

[5]https://thingsboard.io/pricing/

e usabilidade gráfica. Todavia, o custo da solução Trendz Analytics é de USD 175 mensais no plano básico.

Capítulo 5

Considerações finais e trabalhos futuros

A manutenção e preservação da água é fundamental para toda a sociedade. O desenvolvimento de técnicas e abordagens que auxiliem na redução do desperdício de água é uma tarefa complexa, que também requer a conscientização da população sobre seus hábitos de consumo. O advento da área de Internet das Coisas e de Ciência de Dados, pode auxiliar em tais desafios.

Essa dissertação apresentou um trabalho que abrange todo o ciclo de uma solução de sensoriamento com Internet das Coisas. Avaliou-se a disponibilidade de sensores no mercado, as tecnologias disponíveis para transmissão, a escolha de protocolos de comunicação além do desenvolvimento de algoritmos para o acompanhamento das variáveis avaliadas. Uma avaliação experimental real em uma residência foi realizada, o que permitiu validar algoritmos clássicos de detecção de vazamento, além da observação de hábitos de comportamento humano.

A proposta de desenvolver um protótipo de um sistema de baixo custo para detecção de vazamento em residências foi alcançada. Foi possível também contribuir com análises não disponíveis na literatura ao relacionar o estudo em tempo real do consumo com informações do SNIS. Os algoritmos de detecção de vazamento foram implementados tanto na borda quanto na névoa (*Fog* e *Edge computing*), permitindo observar as vantagens e limitações de cada abordagem.

Um estudo do comportamento da pressão na tubulação também foi conduzido. Durante o consumo residencial, todavia, observou-se que este monitoramento é mais adequado em redes de distribuição de água, pois a variação da pressão naquelas redes é mais significativa.

Este trabalho permitiu a publicação de um artigo no congresso Computer on the Beach, no qual foram apresentadas a construção e validação do sistema. Outro trabalho, em desenvolvimento, consiste na proposição da taxonomia aqui apresentada sobre a área de detecção de vazamentos. Detalhes sobre os ambientes de implementação e testes realizados podem ser observados nos Anexos A, B e C, enquanto detalhes sobre a montagem dos experimentos podem ser observados no Anexo D.

Como trabalhos futuros, novas tecnologias e técnicas podem ser avaliadas, como outros ambientes de IoT (Node Red, Particle, OpenRemote, Firebase e o AWS IoT). Uma outra vertente, consiste na avaliação de técnicas para melhorar a eficiência energética dos dispositivos utilizados. Com o uso de técnicas de Light-sleep e Deep-sleep do NodeMCU, o consumo de energia do sistema pode ser

reduzido significativamente.

Ambientes residenciais em edifícios apresentam inúmeros dispositivos de comunicação Wi-Fi. Testes avaliando a interferência de comunicação entre os equipamentos utilizados no sistema de monitoramento também podem ser conduzidos, avaliando-se qual melhor tecnologia deveria ser utilizada em relação a perda de pacotes e etc. Também é possível explorar a tecnologia LoRa que mostrou bom alcance *indoor*, mas que não chegou a ser implementada devido ao custo do roteador LoRaWan, uma vez que testes sem o protocolo mencionado impediram adaptação do código de detecção de vazamento.

O uso de algoritmos de aprendizado de máquina para identificação do comportamento de indivíduos dentro de cada residência é uma outra possibilidade. O sistema desenvolvido pode ser associado a modelos de mecânica de fluidos computacional visando fornecer dados que melhorem tais modelos ou auxiliem na compreensão de fenômenos como o "Golpe de Aríete". Esse tipo de fenômeno também é conhecido em inglês como *water hammer* e tem seu comportamento descrito matematicamente no trabalho de Ghidaoui et al. [2005].

Por fim, o desenvolvimento de algoritmos que sejam capazes de identificar anomalias no sistema a partir da identificação do comportamento descrito e que possam interagir com os usuários para identificar o que realmente é um vazamento ou simplesmente uma mudança de comportamento são novas abordagens a serem avaliadas.

Referências Bibliográficas

ABNT (2020). NBR 5626 - Instalação predial de água fria. Technical report, Rio de Janeiro, BR.

Adsul, S., Sharma, A. K., and Mevekari, R. G. (2017). Development of leakage detection system. *International Conference on Automatic Control and Dynamic Optimization Techniques, ICACDOT 2016*, pages 673–677.

Alarefi, S. A. and Walker, S. (2017). Innovative shunt measurement for residential water microleakage detection. *2017 8th International Renewable Energy Congress, IREC 2017*, (Irec).

Amaxilatis, D., Chatzigiannakis, I., and Tselios, C. (2020). applied sciences A Smart Water Metering Deployment Based on the Fog Computing Paradigm. *Applied Sciences*, pages 1–28.

ANA (2019). Manual dos usos consuntivos de Água do brasil. Technical report, Brasília, BR.

Anatel (2017). Ato n 14448. Technical report, Agência Nacional de Telecomunicações, Brasília, BR.

Araújo, B. Q. U. (2015). *Sistema detector de vazamentos em instalações prediais de água fria*. PhD thesis, UFRN.

Astharini, D. (2012). A Straightforward Design and Implementation of Real Time. *Jurnal AL-AZHAR INDONESIA SERI SAINS DAN TEKNOLOGI*, 1(4):182–185.

Bertoleti, P. (2019). *Projetos com ESP32 e LoRa*. Instituto NCB, São Paulo, BR, 1 edition.

Bistafa, S. R. (2010). *Mecânica dos Fluídos*. Bulcher, São Paulo, Brasil, 1 edition.

Brasileiro, G. (2019). Plano nacional de internet das coisas.

Brasileiro, G. (2020). Lei de incentivo a iot.

Britton, T. C., Stewart, R. A., and O'Halloran, K. R. (2013). Smart metering: Enabler for rapid and effective post meter leakage identification and water loss management. *Journal of Cleaner Production*, 54:166–176.

Carrano, R. C., Passos, D., Magalhaes, L. C. S., and Albuquerque, C. V. N. (2014). Survey and taxonomy of duty cycling mechanisms in wireless sensor networks. *IEEE Communications Surveys Tutorials*, 16(1):181–194.

Coronado, J. M., Badillo-Olvera, A., and Begovich, O. (2019). A wireless low-cost monitoring system for leak diagnosis in water transmission pipelines based on arduino-xbee technology. *2019 IEEE International Autumn Meeting on Power, Electronics and Computing, ROPEC 2019*.

Cruz, F. (2014). São Paulo sofreu pior crise de água de sua história em 2014. *Agência Brasil*.

Dantas, A. M. and Moraes, L. R. S. (2006). Desperdício de água tratada por meio de vazamentos nas instalações hidráulico-sanitárias em edifícios residenciais de salvador, brasil. In *XII Silubesa - Simpósio Luso-Brasileiro de Engenharia Sanitária e Ambiental*, page 8, Figueira da Foz, Portugal.

de Assis Jr, L. B., Oliveira, R. P. D., Carvalho, C. D. S. D., Brandão, D. N., and da Rocha Henriques, F. (2022). Construção e validação de um sistema iot de baixo custo para detecção de vazamento de Água em residências. *Proceedings of Computer on the Beach (COTB'22)*, 1:181–187.

Deepa, B., Anusha, C., and Devi, P. C. (2021). Smart agriculture using iot. In *Intelligent System Design*, pages 11–19. Springer.

Fuentes, H. and Mauricio, D. (2020). Smart water consumption measurement system for houses using IoT and cloud computing. *Environmental Monitoring and Assessment*, 192(9).

Gama-Moreno, L., Reyes, J., Sánchez, M., Ochoa-Franco, C., and Noguerón, C. (2010). Instrumentation of a water-leaks detection system controlled via the short message service through the GSM network. *Proceedings - 2010 IEEE Electronics, Robotics and Automotive Mechanics Conference, CERMA 2010*, pages 654–659.

Ghidaoui, M. S., Zhao, M., McInnis, D. A., and Axworthy, D. H. (2005). A review of water hammer theory and practice. *Applied mechanics reviews*, 58(1):49–76.

Henschke, M., Wei, X., and Zhang, X. (2020). Data Visualization for Wireless Sensor Networks Using ThingsBoard. *2020 29th Wireless and Optical Communications Conference, WOCC 2020*.

Khutsoane, O., Isong, B., and Abu-Mahfouz, A. M. (2017). IoT devices and applications based on LoRa/LoRaWAN. *Proceedings IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017-Jan:6107–6112.

Kodali, R. K. and Soratkal, S. R. (2017). MQTT based home automation system using ESP8266. *IEEE Region 10 Humanitarian Technology Conference 2016, R10-HTC 2016 - Proceedings*.

Laroui, M., Nour, B., Moungla, H., Cherif, M. A., Afifi, H., and Guizani, M. (2021). Edge and fog computing for iot: A survey on current research activities and future directions. *Computer Communications*, 180(September):210–231.

LILYGO (2023). Lilygo®ttgo lora32 v2.0 433/868/915mhz esp32 lora oled 0.96 inch sd card display bluetooth wifi esp32 module with antenna.

Mala-Jetmarova, H., Sultanova, N., and Savic, D. (2018). Lost in optimisation of water distribution systems? a literature review of system design. *Water*, 10(3).

Martinho, W., Melo, R., and Sorensen, K. (2021). An enhanced simulation-based iterated local search metaheuristic for gravity fed water distribution network design optimization. *Computers Operations Research*, 135:105429.

MDR (2018). Diagnóstico dos Serviços de Água e Esgotos - 2018. Technical report, Ministério do Desenvolvimento Regional, Brasília, BR.

Mitchell, T. M. (1997). *Machine learning*. McGraw-hill New York.

Mohan, N. and Kangasharju, J. (2017). Edge-Fog cloud: A distributed cloud for Internet of Things computations. *2016 Cloudification of the Internet of Things, CIoT 2016*, pages 1–6.

Moris, A. S. E. (2001). *Measurement and Instrumentation Principles*. Butterworth-Heinemann, Oxford, 3a edition.

Muhammetoglu, A., Albayrak, Y., Bolbol, M., Enderoglu, S., and Muhammetoglu, H. (2020). Detection and Assessment of Post Meter Leakages in Public Places Using Smart Water Metering. *Water Resources Management*, 34(9):2989–3002.

ONU (2018). Relatório trienal – fórum mundial da Água. Technical report, ONU, Brasília, BR.

Pantoja, C. E., Soares, H. D., Viterbo, J., Alexandre, T., Casals, A., and El-Fallah Seghrouchni, A. (2019). A resource management architecture for exposing devices as a service in the Internet of Things. pages 233–238. Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE.

Patnaik, S., Sen, S., and Mahmoud, M. S. (2020). *Smart Village Technology: Concepts and Developments*. Springer, Gewerbestrasse, Suíça, 17 edition.

Pietrosanto, A., Carratu, M., and Liguori, C. (2021). Sensitivity of water meters to small leakage. *Measurement: Journal of the International Measurement Confederation*, 168:108479.

Pradeep, P. and Krishnamoorthy, S. (2019). The mom of context-aware systems: A survey. *Computer Communications*, 137:44–69.

Prasojo, I., Maseleno, A., Shahu, N., et al. (2020). Design of automatic watering system based on arduino. *Journal of Robotics and Control (JRC)*, 1(2):59–63.

Razvarz, S., Raheleh, J., Vargas-Jarillo, C., Gegov, A., and Arabikhan, F. (2020). Pipeline Leak Detection and Location based on Fuzzy Controller. *IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1944–1949.

Romeiro, L. d. A. W. (2019). *Técnicas de Desagregação de Consumo D'Água a partir da Dinâmica de Vazão em Residências*. PhD thesis, UNIVERSIDADE FEDERAL DA BAHIA.

Seeed (2020). How to use Water Flow Sensor / Meter with Arduino.

Shi, W. and Dustdar, S. (2016). The promise of edge computing. *Computer*, 49(5):78–81.

Silva Júnior, J. F. d. (2017). *Detecção De Perdas Em Sistemas De Distribuição De Água Através De Rede De Sensores Sem Fio*. PhD thesis, UFPE.

Sulieman, N. A., Celsi, L. R., Li, W., Zomaya, A., and Villari, M. (2022). Edge-Oriented Computing: A Survey on Research and Use Cases. *Energies*, 15(2):1–28.

Sánchez, A. S., Cohim, E. H. B., and Kalid, R. d. A. (2015). Sistema inteligente para monitoramento do consumo de Água e detecção de vazamentos em prédios. volume RESAG, pages 1–12, Aracaju, Brasil.

Sánchez, A. S. U., Martins, G., and Martins, A. (2020). Smart metering to track the performance of water conservation programs in a small water system : case study of a University Campus. pages 1–3, Belo Horizonte, Brasil. 17th IWA - Small Water and Wastewater Systems Conference.

Sánchez-Rivero, M., Quiñones-Grueiro, M., Cruz Corona, C., J. Silva Neto, A., and Llanes-Santiago, O. (2020). A Proposal of Robust Leak Localization in Water Distribution Networks Using Differential Evolution Maibe. pages 311–320. 4th International Conference on Soft Computing Models in Industrial and Environmental Applications.

Van Steen, M. and Tanenbaum, A. S. (2017). *Distributed systems*. Maarten van Steen Leiden, The Netherlands.

Vieira, B. M. (2021). Entenda por que está chovendo menos no brasil e se há risco de nova crise hídrica em sp.

Wukkadada, B., Wankhede, K., Nambiar, R., and Nair, A. (2018). Comparison with HTTP and MQTT in Internet of Things (IoT). pages 249–253. Proceedings of the International Conference on Inventive Research in Computing Applications, ICIRCA.

Xia, F., Yang, L. T., Wang, L., and Vinel, A. (2012). Internet of things. *International journal of communication systems*, 25(9):1101.

Zhao, J., Xie, X., Liu, R., Sun, Y., Wu, M., and Gu, J. (2018). Water and energy saving potential by adopting pressure-reducing measures in high-rise building: A case analysis. *Building Services Engineering Research and Technology*, 39(5):505–517.

Anexo A

Arduino IDE

A.1 Instalação

Arduino IDE[1] é a ferramenta usada para programar as placas de prototipagem como Arduino e a NodeMCU, utilizada neste trabalho, e está disponivel para Linux, Mac e Windows. A linguagem de programação é baseada em C. Obrigatoriamente você terá duas funções no seu código: *setup* onde você configurará algumas informações de portas e *loop* onde ficam as instruções a serem executadas.

Instalando bibliotecas

Bibliotecas são conjunto de funções (códigos) prontas que facilitam a criação dos programas para Arduino. Você pode baixar uma biblioteca do GitHub[2], do site do desenvolvedor ou, pela maneira mais simples, por meio do Repositório. Optamos pela última opção neste trabalho. Para instalar abra a Arduino IDE e vá no menu Ferramentas > Gerenciador de Bibliotecas e realize a busca pelas bibliotecas, escolha a versão então pressione "Instalar", como mostrado na Figura A.1. As biblioteca utilizadas estão na Tabela 4.2.

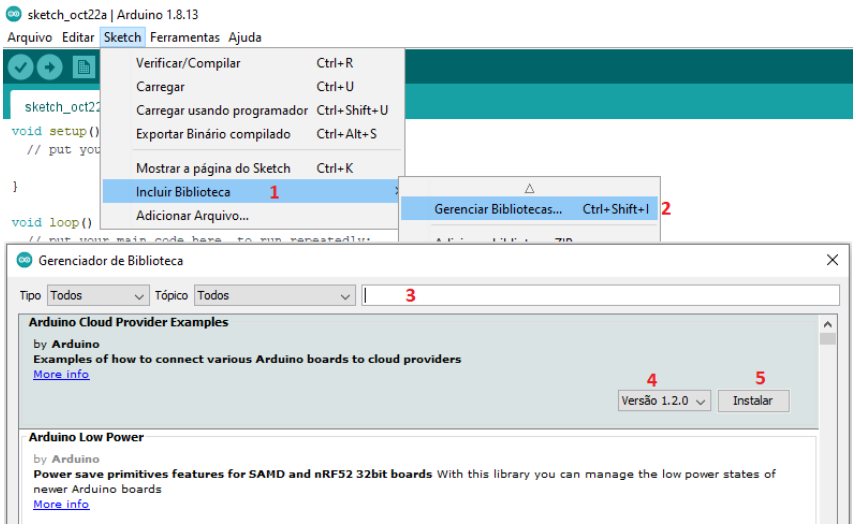


Figura A.1: Instalação de bibliotecas no Arduino IDE

[1] `https://www.arduino.cc/en/software`
[2] github.com

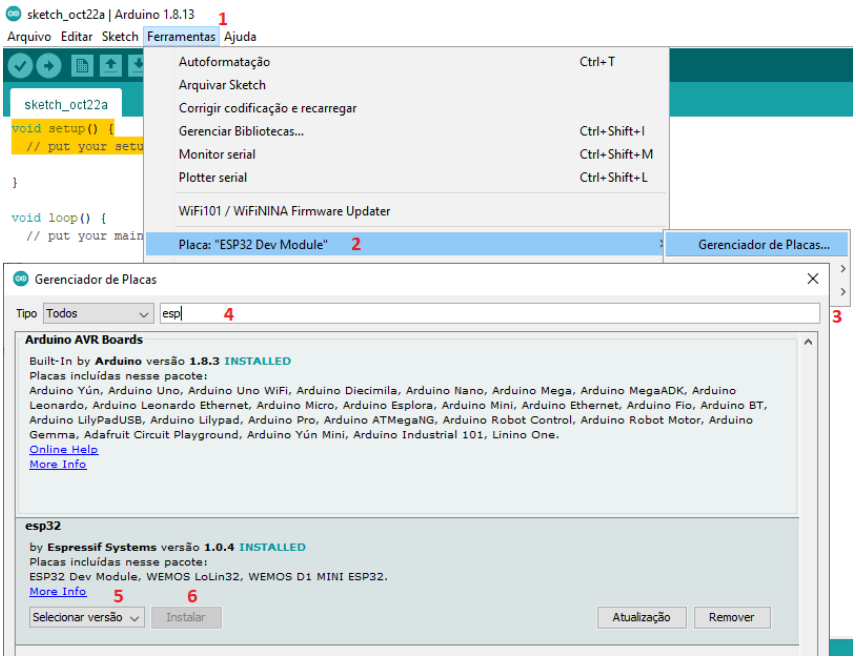


Figura A.2: Instalação de placa no Arduino IDE

Instalando placas

Foram utilizadas placas NodeMCU ESP32 que possui bibliotecas similares, mas não idênticas a NodeMCU ESP8266. Para adicionar a lista de drives necessárias vá em: Arquivo > Preferências > URLs Adicionais para Gerenciadores de Placas e preencha com o repositório da placa [3]. Para a instalação do *driver* da placa NodeMCU (Figura A.2) acesse o menu Ferramentas > Placa > Gerenciador de Placas conforme as versões citadas na Tabela 4.1.

A.2 Soluções de Problemas

Erro ao subir o código para NodeMCU

Se A Arduino IDE apresentar falha ao subir o código para a placa, desligue a energia dos sensores de pressão e vazão.

Valores anômalos de telemetria

Caso a telemetria capturada pelo NodeMCU esteja com valores diferentes do esperado, verifique se o sensor está ligado à porta 2. A porta 2 é compartilhada com Wifi e não deve ser utilizada caso seu programa conecte-se a rede sem fio. Troque de porta fisicamente e também no código a ser subido no NodeMCU.

[3] `https://dl.espressif.com/dl/package_esp32_index.json`

Valores anômalos de telemetria 2

A variável de interrupção só deve ser lida após o comando *detachInterrupt*, pois haverá comportamento anômalo se alterado o valor da variável durante a leitura.

Anexo B

ThingsBoard

Como o ThingsBoard foi instalado num Raspberry Pi 3B, a documentação fornece um passo-a-passo específico [1]. Foi preciso instalar o banco de dados PostgreSQL e o Java via Terminal Linux. Caso tenha dificuldades com o comando de instalação proposto para o PostgreSQL na documentação, tente o comando a seguir:

```
pi@raspberry:~ $ sudo apt install postgresql libpq-dev postgresql-client
postgresql-client-common -y
```

As linhas de criação e configuração do bancos de dados devem ser executadas uma a uma e não em bloco. Anote a senha criada para o PostgreSQL. A seguir estão os passos necessário para importação das configurações do sistema de detecção de vazamentos.

B.1 Criação de Ativo

O Ativo é a estrutura física onde os sensores estão instalados. Serão configuradas dois atributos para este ativo: *moradores* que possui a quantidade de moradores na residência e *regiao* do país em que a casa está localizada. Esses dados são usados na Regra da Cadeia: Consumo Brasil.

Acesse o menu Ativo > + > Adicionar novo ativo. Preencha as informações. Edite o ativo. Vá em Atributos > Atributos do Servidor > +.

```
chave: moradores
tipo:  número inteiro
valor: 2

chave: regiao
tipo:  cadeia de caracteres
valor: sudeste ou nordeste ou norte ou sul ou centro-oeste ou vazio
```

[1] https://thingsboard.io/docs/user-guide/install/rpi/

B.2 Importação de perfil de dispositivo

Importar o arquivo *nodemcu.json* em Device Profiles > + > Importar device profile. Neste arquivo ficam configurados os alarmes de vazamento gerado pelo NodeMCU para visualização no Dashboard e histórico de telemetria do dispositivo.

B.3 Criação e configuração dispositivos

Responsável pela criação dos sensores dentro do Thingsboard usando o perfil de dispositivo: nodemcu. Necessário para as regras de cadeia e *dashboards*. Acessar o menu Dispositivos > + > Adicionar.

```
Nome: reservatorio202
Etiqueta: 202
Select existing device profile: nodemcu
Next: Credencials

Add Credenitials > Access Token >
Token de acesso: Token_Teste_1
```

Este *token* é usado para envio de telemetria do NodeMCU para o Thingsboard. No arquivo *esp32tbmqtt-vazao-pressao.ino* possui uma constante *mqttUser* que armazena este *token*. A seguir está a configuração de atributo que é usado pelas regras de cadeia: Consumo Diário e Regra do Reservatório. Acessar: reservatorio202 > Atributos > Atributos do servidor > +.

```
Chave: tipoDispositivo
Tipo de valor: cadeia de carácter
Valor: reservatorio
```

Configurar um cômodo da casa ou aparelho de consumo. Exemplo: banheiro, cozinha, torneira do jardim, etc.

```
Nome: servico202
Etiqueta: 202
Select existing device profile: nodemcu
Next: Credencials

Add Credenitials > Access Token >
Token de acesso: Token_Teste_2
```

Acessar: reservatorio202 > Atributos > Atributos do servidor > +.

```
Chave: tipoDispositivo
Tipo de valor: cadeia de caracter
Valor: reservatorio
```

B.4 Configuração das relações dos dispositivos com o ativo

É necessário criar a relação entre o ativo apto202 e os dispositivos: reservatorio202 e servico202. Essa configuração é necessária para o funcionamento da cadeia de regra Regra do Reservatório. Percorra o caminho: Ativo > apto202 > Relações > Direção: De > +.

```
Tipo de relação: contains
Dispositivo: servico202 e reservatorio202
Tipo dispositivo: dispositivo
```

B.4.1 Configurações de rede

É necessário configurar um IP fixo para o computador onde será instalado o ThingsBoard e o servidor Kafka. Para tal acesse as configurações do seu roteador.

B.5 Importação das cadeias de regras

As Cadeias de Regras são responsáveis por processar, salvar ou redirecionar os dados recebidos pelo ThingsBoard. Para importação utilize o caminho: Cadeias de Regras > + > Importar.

B.5.1 Kafka

Depois de importar a regra e necessário abri-la. Selecione o bloco Kafka dentro da regra da cadeira e então edite-o. Mude o atributo *bootstrap-server* para o endereço de IP e porta usados pelo servidor Kafka.

O atributo *Topic pattern* configura o tópico Kafka que receberá a mensagem. O ideal que o tópico já esteja criado no servidor Kafka, porém ao receber a primeira mensagem no tópico o próprio servidor o criará dentro de até cinco minutos. Este atributo também é usado pelo consumidor escrito em Python usado no pré-processamento do algoritmo de aprendizado de máquina.

A porta 9092 é a padrão do serviço Kafka, essa configuração deve ser a mesma do arquivo *server.properties* no atributo *advertised.listeners* disponível na pasta onde o Kafka está instalado. Não é preciso se preocupar, a porta 9092 vem configurada por padrão e não é necessário fazer esta verificação.

B.5.2 Demais regras

É necessário importar as regras: Consumo Diário e Regra do Reservatório. Porém estas não requerem configurações extras.

B.5.3 Configuração da Root Rule Chain

A cadeia de regras principal do Thingsboard deve ser alterada. Serão adicionado o redirecionamento de telemetrias do tipo POST TELEMETRY para as regras de cadeia importadas nos passos anteriores (Figura B.1).

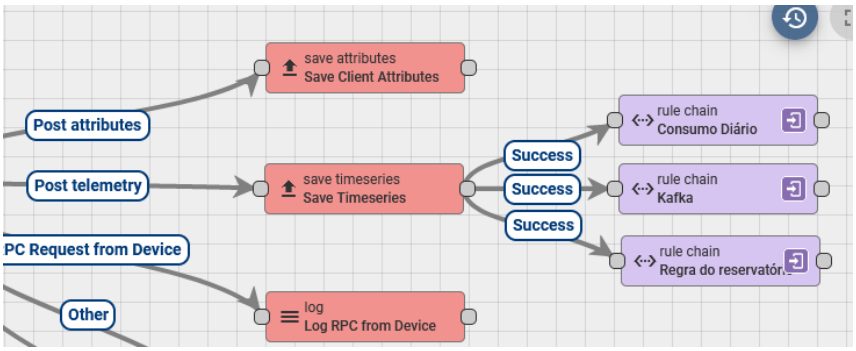


Figura B.1: Adaptação da Root Rule Chain

B.6 Importação do Dashboard

Dashboards são botões de comandos e gráficos mostrados em um painel em tempo real ou de um período de tempo selecionado. Para importação escolha as opções: Dashboards > + > Importar.

B.7 Soluções de problemas

Erro ao subir o Thingsboard

Na versão 3.2.2 durante a verificação do log de notou-se problemas no tomcat. Apache Tomcat que é um servidor JAVA. Na versão 3.4.1 não houve necessidade desta etapa. A ação complementar necessária foi a instalação do tomcat. Essa etapa foi feita pela interface gráfica do Linux em Menu Raspberry > Preferência > Add/Remove Software > tomcat9 (Apache Tomcat9 – Servlet and JSP engine) e tomcat9-user.

Não é possível acessar Thingsboard do navegador ou receber telemetrias

A instalação do ThingsBoard já conta com a implementação do Tomcat e MQTT é necessário interromper esses serviços, caso instalados eternamente, para o funcionamento do ThingsBoard. Utilize os seguintes comandos sempre que reiniciar seu Raspberry:

```
pi@raspberry:~ $ sudo service mosquitto stop
```

```
pi@raspberry:~ $ sudo service tomcat9 stop
pi@raspberry:~ $ sudo service thingsboard restart
```

Problemas de importação de configurações

Podem ocorrer problemas de exportação e importação de configurações do projeto entre versões diferentes do Thingsboard. Ao exportá-las da versão 3.2.2 para 3.4.1 o perfil de dispositivo apresentou falhas impedindo o processamento das telemetrias na Root Rule Chain já no primeiro bloco da cadeia de regra. Esse erro é relatado nas *issues* no GitHub do Thingsboard nº 6421 e 3050. A solução foi recriar o perfil de dispositivo dentro da versão nova de maneira manual e associar o novo perfil aos dispositivos sensores.

Anexo C

Kafka

Para melhor entendimento do Kafka sugere-se leitura de tutoriais na página do Kafka [1], além de assistir o vídeo no YouTube intitulado: Começando com Kafka - Hipsters Ponto Talks 12 [2].

C.1 Instalação

Baixar os arquivos no site da fundação mantenedora do Kafka[3]. Então descompactá-los no diretório desejado (fica a critério do usuário executar esta etapa via interface gráfica ou terminal). O Kafka não é instalado em si, ele é executado, podendo ser salvo em qualquer pasta.

C.2 Configurações

Editar o endereço da pasta de logs é muito importante, pois por padrão está está numa pasta temporária e ao reiniciar o Linux todas as mensagens são perdidas. Edite o arquivo *server.properties* dentro pasta do Kafka a linha *log.dirs*. A seguir como ficou após edição:

```
log.dirs=/home/luis/kafka_2.13-3.1.0/kafka-logs
```

C.3 Comandos úteis

Os comandos a seguir são usados no terminal Linux para utilização de recursos do Kafka. O Zookeeper é responsável por fornecer o ambiente para o funcionamento do Kafka e por isso deve ser executado primeiro. Ambos estão contidos no arquivo compactado do Kafka. O Kafka deve ser executado em um terminal em separado. Para utilizar os comandos Kafka e Zookeeper é necessário navegar pelo terminal Linux até a pasta dos arquivos. As telas do terminal Linux tanto do Zookeeper quanto do Kafka devem permanecer abertas para o funcionamento dos mesmos. Nestes terminais são mostrados o *log* de funcionamento de ambos serviços.

Acesso a pasta do Kafka e Zookeeper quando instalada em *home*

[1] `kafka.apache.org`
[2] `www.youtube.com/watch?v=aaCqcX30pzc`
[3] `kafka.apache.org/downloads`

```
luis@JuniorTOS:$ cd ..
luis@JuniorTOS:$ cd /home/luis/kafka_2.13-3.1.0/kafka-logs
```

Inicialização do Zookeeper

```
luis@JuniorTOS:$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

Inicialização do Kafka

```
luis@JuniorTOS:$ bin/kafka-server-start.sh config/server.properties
```

Encerramento do Kafka Em um terminal separado execute a linha abaixo para fechar o Kafka corretamente.

```
luis@JuniorTOS:$ bin/kafka-server-stop.sh config/server.properties
```

Listar tópicos existentes

```
luis@JuniorTOS:$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092
```

Criar um tópico

```
luis@JuniorTOS:$ bin/kafka-topics.sh --create --bootstrap-server localhost:9092
--replication-factor 1 --partitions 1 --topic nome-do-seu-topico
```

Alterar um tópico

```
luis@JuniorTOS:$ bin/kafka-topics.sh --alter --zookeeper localhost:2181
--topic nome-do-seu-topico --partitions qtd-desejada
```

Ver detalhes de um tópico

```
luis@JuniorTOS:$ bin/kafka-topics.sh --bootstrap-server localhost:9092
--describe --topic nome-do-seu-topico
```

Saber sobre grupos

```
luis@JuniorTOS:$ bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092
--group nome-do-seu-grupo --describe
```

Criar um produtor

```
luis@JuniorTOS:$ bin/kafka-console-producer.sh --broker-list localhost:9092
--topic nome-do-seu-topico
```

Criar um consumidor

```
luis@JuniorTOS:$ bin/kafka-console-consumer.sh --bootstrap-server localhost:9092
--from-beginning --topic nome-do-seu-topico
```

C.4 Solução de problemas

Caso o Kafka encerre repentinamente ou não suba o serviço por falha nos logs, delete o conteúdo da pasta *kafka-logs* e reinicie o serviço.

Anexo D

Experimentos

Esta seção possui detalhes dos experimentos como material utilizado e forma de montagem.

D.1 Precisão do sensor de vazão

Este experimento serviu de base aprender sobre o funcionamento do sensor de vazão e cálculo do erro. O sensor possui três conectores, onde: o amarelo transmite as leituras, o preto é o terra e o vermelho responsável pela alimentação do circuito.

Uma breve descrição do aparato experimental é apresentada. A Figura 4.1 apresenta o esquema do experimento. Resumidamente, o experimento consistiu na ligação de meio metro de mangueira completamente cheia de água à uma torneira por meio de um engate rosqueável. A outra extremidade da mangueira foi conectada ao sensor de vazão por meio de um adaptador de mangueira para rosca interligada a uma luva, que por sua vez, estava ligada o sensor. A outra extremidade do sensor foi ligada a um registro com adaptador para mangueira na ponta.

D.2 Precisão do sensor de pressão

Este experimento serviu de base aprender sobre o funcionamento do sensor de pressão e cálculo de erro. Foram usados os itens relacionados na Tabela D.2para montagem deste experimento. O sensor funcionou com alimentação de 3,3V e não precisou de resistor *pull-up*. Usou-se uma porta analógica no NodeMCU.

Este é um sensor que retorna um sinal analógico e por isso deve ter sua porta configurada para tal. A resolução do NodeMCU ESP32 é de 12 bits, ou seja, aceita valores de 0 à 4095 (2^12). No ESP8266 são 10 bits. No Algoritmo 3 é informação é utilizada na variável *resolucaoPorta*.

D.3 Testes com MQTT

Este experimento buscou a implementação do protocolo de troca de mensagens MQTT que foi utilizado nas comunicações entre dispositivos. Sua implementação se deu por meio do Mosquitto[1]

[1] mosquitto.org

Tabela D.1: Aferição da precisão do sensor de vazão - 1a versão do algoritmo

Valores (ml)		
Real	**Medido via sensor**	
500	633	
500	519	
500	571	
500	434	
500	670	
500	590	
500	706	
500	495	
500	485	
500	566	
500	550	
500	532	
510	532	
520	528	
500	460	
500	522	
500	516	
500	458	
500	466	
500	487	
500	525	Mediana
501,43	535,48	Média
4,67	69,44	Desvio Padrão
	6,79	Erro

Tabela D.2: Materiais utilizados no experimento de precisão do sensor de pressão

Qtd	Descrição
1	NodeMCU ESP32 devkit 1.0
1	Mangueira com 1m e 1,7cm de diâmetro útil, maleável e transparente
1	Sensor de pressão 1,2Mpa e 1/4"para gases, líquido e óleo
1	Adaptador niple de metal de 1/4" para 1/2" rosqueável
1	Fita veda-rosca (branca)
1	Adaptador de 1/2" para mangueira
1	Luva 1/2"

na plataforma Linux, tentativas de instalação na plataforma Windows não obtiveram sucesso.

D.3.1 Instalação do MQTT

Este passo não é necessário para o funcionamento do Thingsboard. Para estudo e testes do protocolo MQTT pode-se instalar o *framework* Mosquitto no Linux seguindo os passos adiante.

Para instalação do *broker* via interface gráfica (Figura D.1) selecione o Menu Raspberry > Preferência > Add/Remove Software > Instalar o pacote: MQTT version 3.1/3.1.1 compatible message broker. No trecho a seguir temos os comandos *sudo* utilizados no terminal Linux para instalação do Mosquitto (cliente).

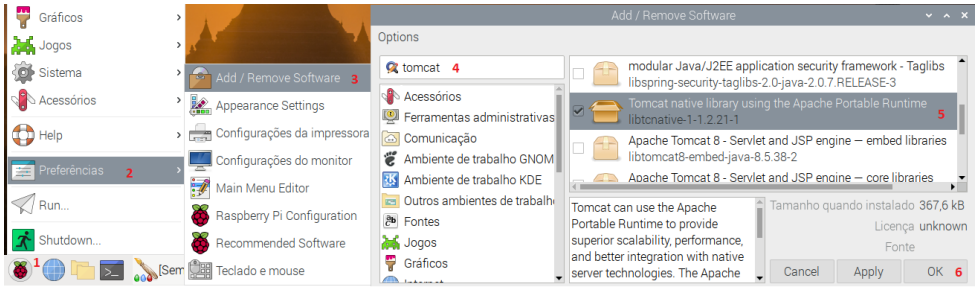


Figura D.1: Instalando programas/bibliotecas no Linux via interface gráfica

```
pi@raspberry:~ $ sudo apt update
pi@raspberry:~ $ sudo apt install -y mosquitto mosquitto-clients
```

D.3.2 Teste no Raspberry Pi

Nesta etapa foi utilizado um Raspberry Pi para troca de mensagens. Foram abertos 2 terminais e executada uma linha de comando em cada terminal. Também foi realizado teste com dois subscribers abertos. Os testes foram bem sucedidos, como pode ser visualizado na Figura D.2.

Terminal 1: Subscriber

```
pi@raspberry:~ $ mosquitto_sub -d -t testTopic
```

Terminal 2: Publisher

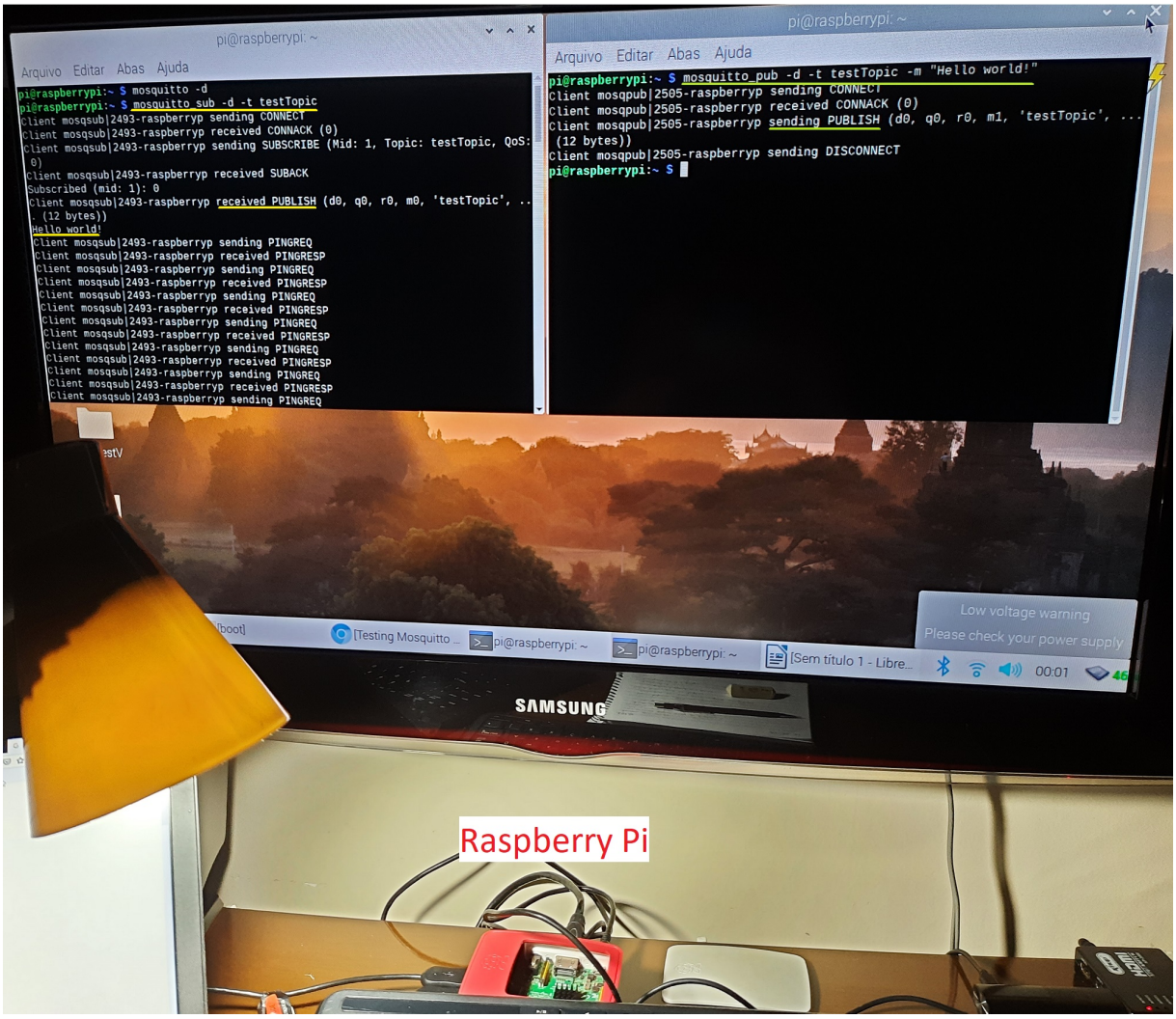


Figura D.2: Envio de mensagem MQTT no Raspberry Pi

```
pi@raspberry:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"
```

D.3.3 Comunicação NodeMCU e Rasberry Pi

Os testes com MQTT continuam nesta etapa com o acréscimo de um NodeMCU que assume a função de *publisher* e *subscriber*, enviando e recebendo mensagens para o Raspberry Pi que tem ambos papéis de *broker* e *subscriber*. O Algoritmo 6 tem como entrada as informações da rede wifi utilizada (*ssid* e *password*), endereço IP e porta do *broker* Raspberry Pi, além do(s) tópico(s) que está inscrito para receber e mandar mensagens. Os dispositivos foram conectados por meio de um roteador sem fio. O comando a seguir foi executado no terminal do Linux e envia uma mensagem do Raspberry para NodeMCU.

```
pi@raspberry:~ $ mosquitto_pub -d -t sub_esp32 -m "Eu sou o Raspberry Pi!"
```

D.3.4 Comunicação NodeMCU e ThingsBoard

Neste experimento, o envio de telemetria do sensor de luminosidade ligado ao NodeMCU em tempo real ao ThingsBoard via protocolo MQTT foi avaliado. Para tanto, o Algoritmo 6 foi utilizado, adicionando apenas o $mqttUser$ (chamado de $Token$ dentro do ThingsBoard) e $mqttPassword$ (não utilizado neste primeiro momento), além de alterar ligeiramente a instrução usada para logar no servidor/broker. Enquanto que no ThingsBoard é necessário ter um dispositivo criado com o token e tópico MQTT utilizados no código do NodeMCU. A Figura D.3 mostra o monitor serial do Arduino IDE exibindo a leitura do sensor diretamente no NodeMCU enquanto que no navegador de internet apresenta a telemetria recebida pelo ThingsBoard. Uma maneira de visualização gráfica dos dados é por meio de *dashboards*, como na Figura D.4. *Dashboards* são gráficos interativos com a telemetria enviada pelos sensores atualizados em tempo real ou por períodos selecionados. A Tabela D.3 apresenta todos os componentes utilizados nas diferentes experiências da subseção D.3.4.

Tabela D.3: Materiais utilizados nos experimentos de comunicação com ThingsBoard da Seção D.3.4

Qtd	**Descrição**
2	Adaptador de 1/2” para mangueira
1	Adaptador niple de metal de 1/4” para 1/2” rosqueável
1	Fita veda-rosca (branca)
2	NodeMCU ESP32 devkit 1.0
1	Power Bank 10.400mAh
1	Mangueira com 1m e 1,7cm de diâmetro útil, maleável e transparente
1	Registro Rosqueável 1/2”
2	Resistor 10KΩ
1	Sensor de luminosidade (LDR)
1	Sensor de pressão 1,2Mpa de 1/4” para gases, líquido e óleo
1	Sensor de Vazão 1/2”
1	Sensor de Vazão 3/4”
1	Luva 1/2”

Envio de Telemetria da Pressão e Vazão

O experimento seguinte teve como objetivo o envio de telemetria do sensor de vazão e pressão para o ThingsBoard. A Figura D.6 apresenta a montagem física do experimento, onde uma mangueira flexível e transparente foi ligada a uma torneira e sua outra extremidade conectada aos sensores de pressão e vazão em um junção T de plástico de 1/2"e após os sensores, um registro

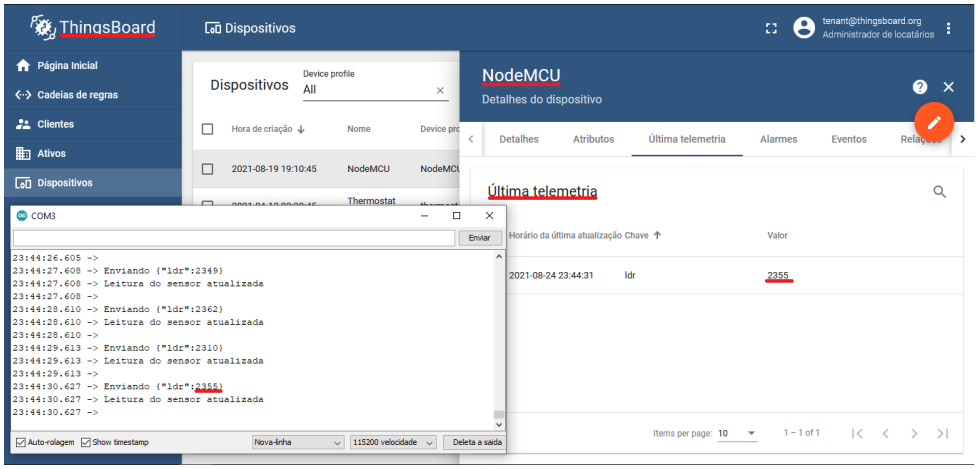


Figura D.3: Leitura do sensor de luminosidade pelo monitor serial e ThingsBoard

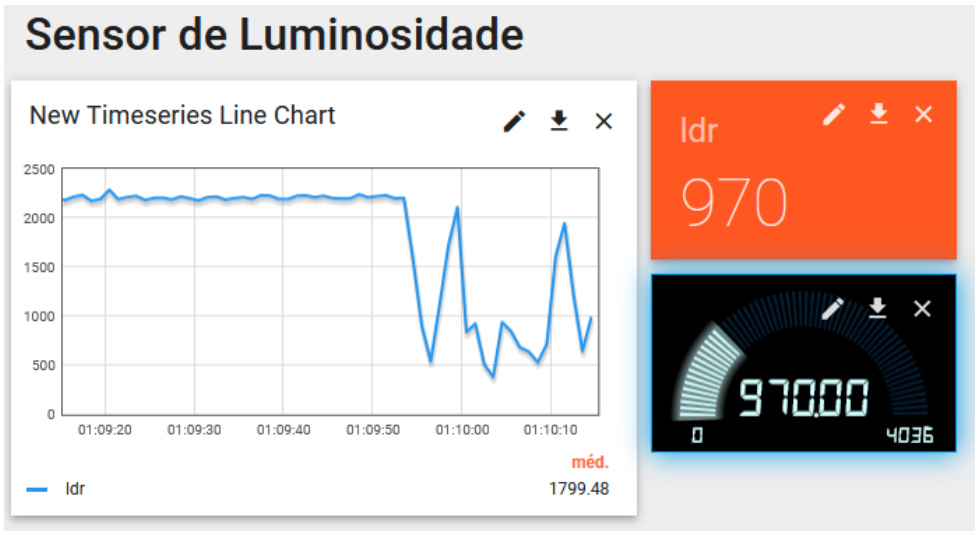


Figura D.4: Exemplo de dashboard mostrando telemetria recebida pelo ThingsBoard

para controlar a vazão. Nesta etapa, foram unidos os Algoritmos 3 e **??** e adaptados para envio de telemetria a cada minuto. A primeira versão teve o tempo de análise do Algoritmo **??**, vazão, ampliado para 60 segundos, porém essa abordagem apresentou valores errados então foi possível apenas validar a telemetria de pressão enviada com dados muito próximos aos encontrados na seção 4.3 como apresentado na Figura D.5.

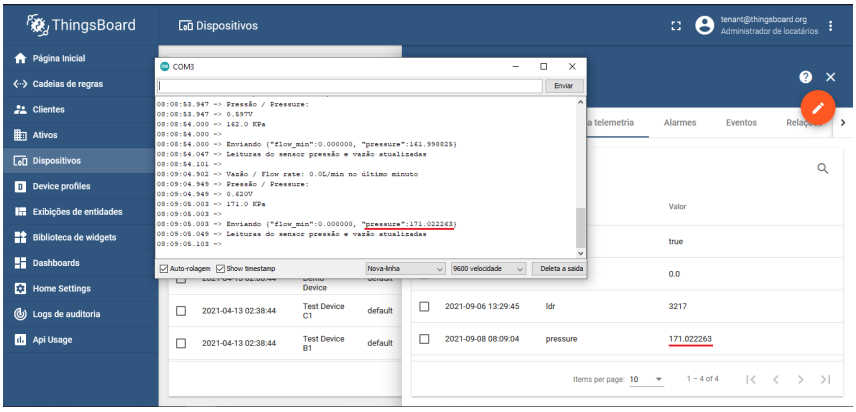


Figura D.5: Envio de telemetria de vazão e pressão

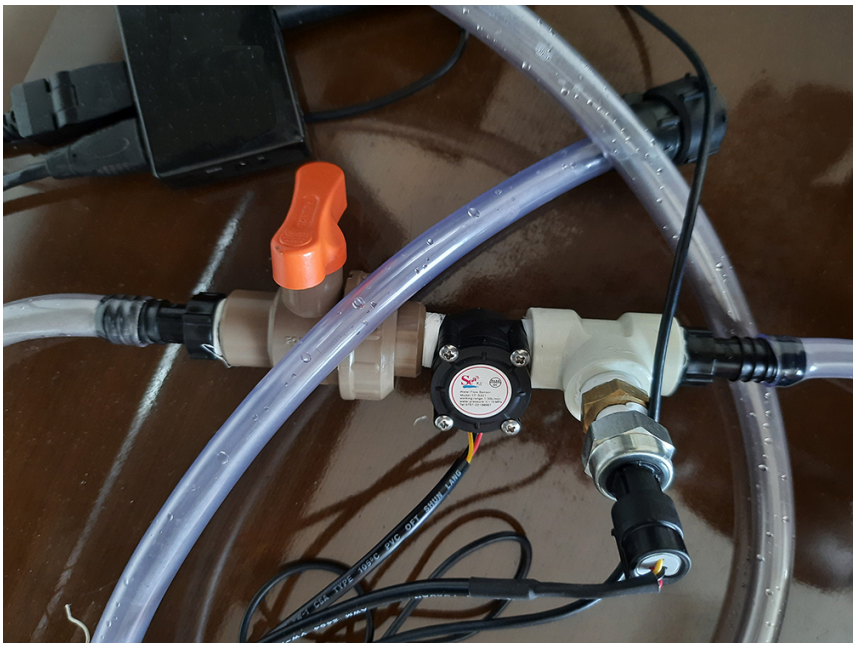


Figura D.6: Montagem hidráulica dos sensores de vazão e pressão em uma mangueira

Apesar de o sensoriamento de vazão não estar funcionando corretamente, pois o valor indicado foi de 40,39L quando devia ser algo próximo a 1,5L, foi verificado na prática um fato muito corriqueiro na casa das pessoas. Durante o enchimento de uma caixa acoplada da descarga sanitária é possível notar uma redução na vazão da água na torneira da pia do mesmo banheiro isso se dá pela redução da pressão na tubulação. Enquanto estava ocorrendo consumo de água no experimento a pressão caiu e a telemetria a seguir mostra exatamente esse fato.

15:16:44.176 − > Vazão / Flow rate: 0.0L/min no último minuto

15:16:44.223 − > Pressão / Pressure: 0.637V = 178.1 KPa

15:16:44.324 − > Leituras do sensor pressão e vazão atualizadas

15:16:44.377 − >

15:17:45.164 − > Vazão / Flow rate: 18.4L/min no último minuto

15:17:45.217 − > Pressão / Pressure: 0.358V = 66.3 KPa

15:17:45.317 − > Leituras do sensor pressão e vazão atualizadas

15:17:45.364 − >

15:18:46.152 − > Vazão / Flow rate: 21.9L/min no último minuto

15:18:46.205 − > Pressão / Pressure: 0.619V = 170.7 KPa

15:18:46.306 − > Leituras do sensor pressão e vazão atualizadas

Para solucionar o problema de leitura da vazão foram necessárias diferentes ações. Descobriu-se que ao adicionar o código de conexão ao WiFi e ao broker MQTT a porta 2 do NodeMCU, inicialmente utilizada para ler o sensor, parou de funcionar sendo substituída pela porta 34 crê-se que tal comportamento é normal, pois novos recursos do hardware foram ativados e a fim de reduzir custos e baratear portas são compartilhadas funcionalidades diferentes. Após algum tempo o sensor de vazão parou de girar e sua abertura foi necessária para limpeza do eixo que apresentava algo parecido com a gordura notada nos primeiros usos da borracha transparente de água. Por fim, o algoritmo foi alterado. Notou-se que tanto o código do fabricante quanto outros encontrados na internet levava em conta o número de interrupções (giros do sensor) dentro de um segundo e não em 60 segundos. Aproximadamente 60 leituras realizadas em um minuto tiveram seu fluxo acumulado enviado ao final sendo reiniciado para o próximo minuto. Na prática passou-se a enviar o que foi consumido de água no minuto anterior. O código até aqui leva centenas de milésimos de segundos para ser processado, porém esse tempo de processamento deverá aumentar até todas as funcionalidades estarem implementadas. Com essa perda de tempo para o processamento durante um experimento que mediu por três vezes a vazão de 500 mililitros chegou a um erro de precisão de até 10%. O Algoritmo 7 mostra a ideia do envio da telemetria de pressão e vazão até o momento atual.

Envio de Telemetria Simultânea

Continuando os experimentos de comunicação optou-se por testar o envio de telemetria de dois diferentes NodeMCU simultaneamente. Na Figura D.7 podemos verificar o valor sendo lido e transmitido de um nó com sensor de vazão e pressão e de um segundo nó com apenas sensor de vazão. Este teste foi feito “a seco” simplesmente assoprando os sensores sem aferir os valores de vazão. Na imagem podemos ver tanto o monitor serial quanto a página do ThingsBoard com a

telemetria em ambos.

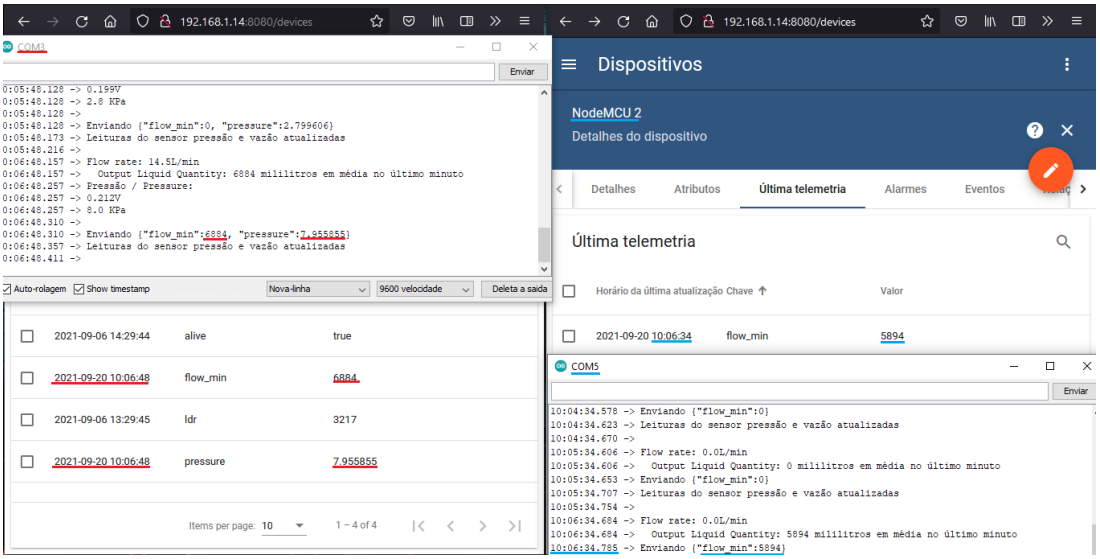


Figura D.7: Telemetria de dois NodeMCU sendo enviados simultaneamente