

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA
CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA**

YURI MARQUES BARBOZA

**ESTUDO E APLICAÇÃO DO ESP32H2 EM REDES IoT BASEADAS
NO PROTOCOLO ZIGBEE**

FLORIANÓPOLIS, 2025.

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA
CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELETRÔNICA**

YURI MARQUES BARBOZA

**ESTUDO E APLICAÇÃO DO ESP32H2 EM REDES IoT BASEADAS
NO PROTOCOLO ZIGBEE**

Trabalho de Conclusão de Curso
submetido ao Instituto Federal de
Educação, Ciência e Tecnologia de Santa
Catarina como parte dos requisitos para
obtenção do título de Bacharel em
Engenharia Eletrônica.

Orientador:
Prof. Renan Augusto Starke, Dr.

FLORIANÓPOLIS, 2025.

Ficha de identificação da obra elaborada pelo autor.

Barboza, Yuri Marques
ESTUDO E APLICAÇÃO DO ESP32H2 EM REDES IoT BASEADAS
NO PROTOCOLO ZIGBEE / Yuri Marques Barboza; orientação de
Renan Augusto Starke. - Florianópolis, SC, 2025.
58 p.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal
de Santa Catarina, Câmpus Florianópolis. Bacharelado
em Engenharia Eletrônica. Departamento Acadêmico
de Eletrônica.
Inclui Referências.

1. Zigbee. 2. Internet das coisas. 3. Automação
residencial. 4. ESP32-H2. 5. Engenharia Eletrônica. I.
Starke, Renan Augusto. II. Instituto Federal de Santa Catarina.
III. ESTUDO E APLICAÇÃO DO ESP32H2 EM REDES
IoT BASEADAS NO PROTOCOLO ZIGBEE.

ESTUDO E APLICAÇÃO DO ESP32-H2 EM REDES IOT BASEADAS NO PROTOCOLO ZIGBEE

YURI MARQUES BARBOZA

Este trabalho foi julgado adequado para obtenção do título de Bacharel em Engenharia Eletrônica e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 22 de Agosto, 2025.

Banca Examinadora:

Documento assinado digitalmente
 **RENAN AUGUSTO STARKE**
Data: 03/09/2025 17:31:32-0300
Verifique em <https://validar.iti.gov.br>

Prof. Renan Augusto Starke, Dr. Eng.

Documento assinado digitalmente
 **MATHEUS LEITZKE PINTO**
Data: 05/09/2025 14:47:38-0300
Verifique em <https://validar.iti.gov.br>

Prof. Matheus Leitzke Pinto, Msc. Eng.

Documento assinado digitalmente
 **DANIEL LOHMANN**
Data: 04/09/2025 23:05:48-0300
Verifique em <https://validar.iti.gov.br>

Prof. Daniel Lohmann, Msc. Eng.

AGRADECIMENTOS

Primeiramente eu quero agradecer a minha mãe que mesmo apesar de todas as dificuldades e empecilhos da vida, fez o possível e o impossível para me apoiar e me incentivar ao longo de toda a minha vida. Sua força, carinho e determinação sempre serão minhas fontes de inspiração.

Quero agradecer a minha família, que além de me apoiar sempre me fizeram sentir amado e encorajado a seguir meus sonhos, vocês foram o motivo pelo qual essa jornada foi capaz.

Também quero agradecer meus amigos que tornaram meu tempo na faculdade menos estressante e também me proporcionaram vários momentos e amizades que ficarão guardadas em um espaço especial no meu coração.

Finalmente gostaria de agradecer aos professores que marcaram a minha vida como estudante, e me ajudaram a se tornar não só um engenheiro melhor como também um ser humano melhor. Um agradecimento especial ao meu orientador, que além de professor se tornou um amigo e conselheiro me ajudando a esclarecer qualquer tipo de dúvida.

“Ser capaz de perseguir nossos próprios
sonhos é o que nos faz forte.”
– Chester Bennington

RESUMO

Este trabalho tem como objetivo explorar o potencial do microcontrolador ESP32-H2 no desenvolvimento de aplicações voltadas para a *Internet of Things* (IoT), destacando suas possíveis capacidades de integrações com sensores e protocolos de comunicação. O ESP32-H2, fabricado pela Espressif Systems oferece suporte nativo a Zigbee, sendo esse, um protocolo de comunicação escolhido neste trabalho, por ser bastante utilizado em projetos de automação residencial. A metodologia adotada neste trabalho envolveu o *setup* e a configuração do ambiente de desenvolvimento do ESP32-H2, a implementação de uma aplicação utilizando um sensor externo de umidade, HTU2X, o próprio chip de temperatura integrado no ESP32-H2 e um sensor virtual criado dentro dos códigos de desenvolvimento do *firmware* com o intuito de testar e utilizar a comunicação via Zigbee com um coordenador em uma rede doméstica. Os resultados obtidos mostraram a viabilidade de uso do ESP32-H2 em redes Zigbee, tendo uma vasta flexibilidade para adaptação em diferentes contextos de IoT, assim como, diversos protocolos já definidos em sua *database* que facilitam a conexão de mais sensores conforme a necessidade do usuário. O presente estudo tenta evidenciar que o ESP32-H2 é uma alternativa para o desenvolvimento de dispositivos conectados sem fio, assim como contribui para a disseminação do uso de microcontroladores em projetos IoT podendo servir como base para futuras pesquisas.

Palavras-chave: ESP32-H2. ZigbeeMQTT. Internet das coisas. Zigbee. Automação residencial.

ABSTRACT

This work aims to explore the potential of the ESP32-H2 microcontroller in the development of applications focused on the Internet of Things (IoT), highlighting its possible capabilities for integration with sensors and communication protocols. The ESP32-H2, manufactured by Espressif Systems, offers native support for Zigbee, which was the communication protocol chosen in this study due to its widespread use in home automation projects. The methodology adopted in this work involved setting up and configuring the development environment for ESP32-H2, implementing practical applications using an external humidity sensor (HTU2X), the internal temperature sensor integrated in the ESP32-H2, and a virtual sensor created within the development code in the firmware in order to test and utilize Zigbee communication with a coordinator in a home network. The results showed the feasibility of using the ESP32-H2 in Zigbee networks, with great flexibility for adaptation to different IoT contexts, as well as various pre-defined protocols in its database that facilitate the connection of additional sensors as needed by the user. This study seeks to demonstrate that the ESP32-H2 is an alternative for the development of wireless connected devices, and it also contributes to the dissemination of the use of microcontrollers in IoT projects, potentially serving as a foundation for future research.

Keywords: ESP32-H2. ZigbeeMQTT. Internet of Things. Zigbee. Home Automation.

LISTA DE FIGURAS

Figura 1: Diagrama de blocos do ESP32-H2 SoC.....	17
Figura 2: Comparação de tecnologias wireless.....	19
Figura 3: Arquitetura Zigbee.....	20
Figura 4: Topologia de uma rede Zigbee.....	22
Figura 5: <i>Binding</i> em um exemplo de controle de luz.....	23
Figura 6: Arquitetura do MQTT Publish/Subscribe.....	24
Figura 7: Arquitetura ZigbeeMQTT.....	27
Figura 8: Sensor HTU2X.....	28
Figura 9: Hardware utilizado no projeto.....	30
Figura 10: Diagrama de blocos do <i>firmware</i>	31
Figura 11: ZigbeeMQTT interface html.....	32
Figura 12: Lista de <i>cluster</i> do <i>firmware</i> visível na interface HTML.....	35
Figura 13: <i>Log</i> de medição gerado pelo <i>firmware</i> no terminal do VSCode.....	45
Figura 14: <i>Log</i> de envio na rede Zigbee visto pelo terminal com o Zigbee2MQTT ativo.....	46
Figura 15: Valores visíveis na interface HTML.....	47
Figura 16: Calibração dos valores medidos.....	47
Figura 17: Mapa mostrando coordenador e o ESP32-H2 na interface HTML.....	48
Figura 18: <i>Setup</i> medição de energia.....	49
Figura 19: Consumo de energia médio.....	50
Figura 20: Consumo de energia em envio de informação.....	50

LISTA DE QUADROS

Quadro 1: Função <i>main</i> inicial do projeto.....	33
Quadro 2: Criação da lista de <i>clusters</i> utilizado no <i>firmware</i>	36
Quadro 3: <i>Report</i> de dados automático para todos os <i>de temperatura</i>	37
Quadro 4: <i>Report</i> de dados automático para todos os <i>de pressão e umidade</i>	38
Quadro 5: Configuração padrão <i>para o sensor</i>	39
Quadro 6: Função <i>handler</i> para <i>cluster</i> de umidade.....	40
Quadro 7: Função <i>handler</i> para <i>cluster</i> de temperatura.....	40
Quadro 8: Função de leitura de umidade.....	41
Quadro 9: Função de leitura de temperatura.....	42
Quadro 10: Tarefa <i>i2c_HTU2X_task</i> leitura do sensor.....	43

LISTA DE ABREVIATURAS E SIGLAS

ADC	<i>Analog-to-Digital Converter</i>
API	<i>Application Programming Interface</i>
BLE	<i>Bluetooth Low Energy</i>
GPIOs	<i>General Purpose Input/Output</i>
ID	<i>Identity</i>
IDE	<i>Integrated Development Environment</i>
IDF	<i>IoT Development Framework</i>
IoT	<i>Internet of Things</i>
I2C	<i>Inter-Integrated Circuit</i>
MQTT	<i>Message Queuing Telemetry Transport</i>
PWM	<i>Pulse Width Modulation</i>
RAM	<i>Random Access Memory</i>
ROM	<i>Read Only Memory</i>
SAP	<i>Service Access Points</i>
SDK	<i>Software Development Kit</i>
SoC	<i>System on a Chip</i>
SPI	<i>Serial Peripheral Interface</i>
UART	<i>Universal Asynchronous Receiver-Transmitter</i>
ZCL	<i>Zigbee Cluster Library</i>
ZED	<i>Zigbee End Device</i>

SUMÁRIO

1	INTRODUÇÃO.....	12
1.1	Justificativa.....	13
1.2	Definição do Problema.....	13
1.3	Objetivo Geral.....	14
1.4	Objetivos Específicos.....	14
1.5	Estrutura do Trabalho.....	15
2	FUNDAMENTAÇÃO TEÓRICA.....	16
2.1	Microcontrolador ESP32-H2.....	16
2.2	Protocolo de rede Zigbee.....	18
2.2.1	<i>Clusters.....</i>	<i>22</i>
2.3	Protocolo MQTT.....	24
2.4	Gateway ZigbeeMQT.....	26
2.5	Sensor HTU2X.....	27
2.6	Resumo.....	29
3	METODOLOGIA.....	30
3.1	Métodos Aplicados.....	31
4	APRESENTAÇÃO DOS RESULTADOS.....	45
4.1	Análise e discussão dos resultados.....	45
4.2	Consumo de energia.....	49
5	CONSIDERAÇÕES FINAIS.....	52
5.1	Sugestões para trabalhos futuros.....	53
	REFERÊNCIAS.....	54

1 INTRODUÇÃO

O ESP32-H2 é um microcontrolador desenvolvido pela empresa Espressif Systems que de acordo com a Espressif (2025), tem um suporte integrado para os protocolos *Zigbee*, *Thread* e *Bluetooth Low Energy* (BLE), possuindo um microprocessador do tipo RISC-V de 32 bits. Ainda de acordo com a Espressif (2025), o “ESP32-H2 foi projetado para garantir baixo consumo de energia e segurança para os dispositivos conectados”, características que tornam esse microcontrolador ideal para dispositivos IoT usados principalmente em áreas como automatização de casas, área da saúde, robôs de serviços, automação industrial entre outros.

Com o avanço da tecnologia e a crescente demanda por soluções conectadas, o conceito de Internet das Coisas, ou do termo em inglês *Internet of Things* (IoT), tem ganhado espaço em diversos setores da sociedade, especialmente em ambientes domésticos, industriais e urbanos. De acordo com a Internet of Things Wiki (2025), IoT é basicamente, uma rede de dispositivos interconectados, podendo ser sensores, softwares e outros eletrônicos, que são capazes de coletar e trocar dados entre si de forma responsiva. Neste trabalho, o protocolo MQTT é empregado no *gateway* enquanto a rede Zigbee realiza a comunicação local entre os sensores. De acordo com a Zigbee2MQTT (2025, tradução nossa), “Zigbee2MQTT integra-se bem com a maioria das soluções de automação residencial...”.

Apesar do potencial do ESP32-H2, ainda há escassez de aplicações práticas e estudos documentados que explorem seu uso como nó de sensoriamento em redes Zigbee no contexto da IoT. Alguns exemplos básicos podem ser encontrados no repositório *open-source* do espressif idf (esp-idf), que seria o *framework* oficial fornecido pela Espressif. Essa lacuna justifica a realização deste trabalho, que busca não apenas validar o uso do ESP32-H2 em um cenário real de automação e monitoramento, mas também contribuir como um possível material de pesquisa para projetos futuros.

1.1 Justificativa

O avanço das tecnologias sem fio tem impulsionado significativamente o desenvolvimento de soluções inteligentes e conectadas. Por entanto, o ESP32-H2 surge como alternativa moderna possuindo integração nativa com protocolos de comunicação voltados a conexão domésticas e industriais, também oferecendo recursos de segurança por hardware e baixo consumo energético. No entanto, por ser tratar de um microcontrolador recente, o material de apoio, exemplos práticos e aplicações documentadas com esse modelo ainda são escassos, o que pode limitar a sua adoção por parte de desenvolvedores e pesquisadores.

Frente a essa carência de documentação técnica e estudos aplicados, este trabalho se justifica pela necessidade de explorar e validar o uso do ESP32-H2 em cenários de comunicação e sensoriamento sem fio, especialmente no contexto de IoT, onde dispositivos precisam se comunicar de forma eficiente, segura e com baixo consumo energético. A escolha do Zigbee como protocolo de comunicação baseia-se em sua ampla aceitação no mercado de automação residencial, por permitir a formação de redes em *mesh*, fácil integração com protocolos de mensagens como Message Queuing Telemetry Transport (MQTT), entre outras características, Zigbee (2025, tradução nossa).

Ao implementar e testar uma aplicação prática utilizando o ESP32-H2 em uma rede Zigbee, este trabalho contribui com conhecimento técnico e experimental que pode auxiliar desenvolvedores interessados em iniciar projetos com este microcontrolador. Dessa forma, o estudo se mostra relevante tanto do ponto de vista acadêmico quanto prático, podendo servir de base para futuras pesquisas e aplicações no campo da computação embarcada e da automação.

1.2 Definição do Problema

O ESP32-H2 é um microcontrolador da Espressif Systems que oferece suporte nativo a protocolos como Zigbee, *Thread* e *Bluetooth Low Energy* (BLE), além de recursos de segurança embarcados e baixo consumo de energia, características essenciais para aplicações em IoT. No entanto, ainda há carência de

estudos aplicados que explorem sua utilização em redes Zigbee com sensores físicos e virtuais, podendo dificultar sua adoção prática em projetos embarcados.

O problema principal deste trabalho consiste em responder a seguinte pergunta:

Como utilizar o protocolo Zigbee no ESP32-H2 e ter dados sendo transmitido pela rede Zigbee?

1.3 Objetivo Geral

Analisar e estudar o microcontrolador ESP32-H2 como plataforma de desenvolvimento de aplicações voltadas à Internet das Coisas (IoT), com ênfase na comunicação via protocolo Zigbee.

1.4 Objetivos Específicos

Para alcançar o objetivo descrito, os seguintes objetivos específicos foram destacados:

- a) Estudar as características técnicas e funcionais do microcontrolador ESP32-H2, com foco em seus recursos voltados para aplicações IoT e suporte nativo ao protocolo Zigbee;
- b) Analisar o funcionamento do protocolo Zigbee e sua aplicabilidade em redes de sensores sem fio voltadas à automação residencial;
- c) Configurar o ambiente de desenvolvimento compatível com o ESP32-H2, utilizando ferramentas e bibliotecas fornecidas pela Espressif;
- d) Desenvolver e validar uma aplicação utilizando o sensor interno de temperatura do ESP32-H2, explorando sua comunicação via Zigbee com um coordenador de rede doméstica.
- e) Integrar e testar um sensor externo de umidade e temperatura (HTU2X), avaliando sua leitura, envio dos dados na rede Zigbee.

- f) Simular um sensor virtual por meio de variáveis programadas, com o objetivo de testar diferentes tipos de *payloads* e o comportamento da pilha Zigbee em múltiplos dispositivos lógicos.
- g) Verificar o consumo energético do protótipo durante a execução das aplicações.

1.5 Estrutura do Trabalho

Este projeto é composto por cinco capítulos, onde o primeiro capítulo tem como objetivo apresentar o tema, justificar o motivo para a esta pesquisa, uma definição do problema e finalmente os objetivos do estudo.

O segundo capítulo apresenta a fundamentação teórica utilizada para a elaboração e execução deste trabalho. Diversas documentações oficiais e exemplos criados por outros usuários foram utilizados para que se tivesse um embasamento teórico e referencial para o desenvolvimento da metodologia.

O terceiro capítulo explica a metodologia utilizada neste trabalho. Nesse capítulo é apresentado a forma com que o projeto foi escalonado conforme as aplicações eram implementadas com sucesso, e a comunicação da rede Zigbee era vista funcionando.

O quarto capítulo exhibe os resultados obtidos, assim como é feita uma análise do desempenho do ESP32-H2 com a aplicação criada ao decorrer deste projeto, trazendo imagens da interface criada pelo Zigbee2MQTT e também o consumo de energia do sistema.

Finalmente, o quinto capítulo apresenta as conclusões, considerações finais e possíveis ideias para futuros projetos encontrados após análises feitas neste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

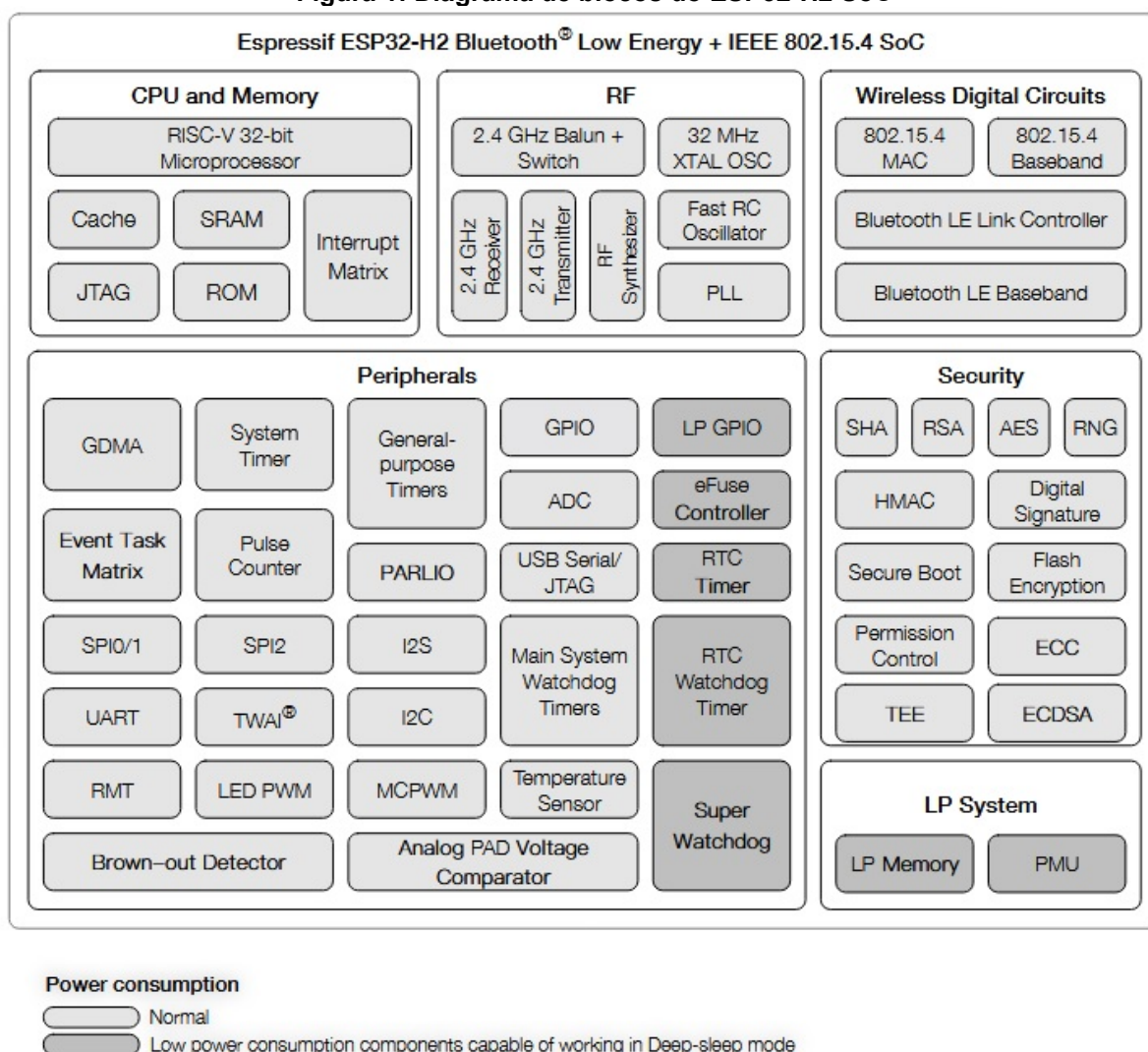
Este capítulo tem como objetivo apresentar os conceitos fundamentais necessários para a compreensão e desenvolvimento da aplicação prática proposta neste trabalho. Inicialmente, são discutidas as características técnicas do microcontrolador ESP32-H2, suas principais propriedades e relação com aplicações para IoT. Em seguida, é abordado o protocolo Zigbee, suas especificações, topologias de rede e propriedades na comunicação de dispositivos. Em sequência, é descrito sobre o protocolo MQTT e o ZigbeeMQTT, que servirá como *gateway* neste projeto e como a entidade que manterá a rede Zigbee ativa.

2.1 Microcontrolador ESP32-H2

O ESP32-H2 é um microcontrolador desenvolvido pela Espressif Systems, que de acordo com a própria Espressif (2025a), “[...] O ESP32-H2 foi projetado para garantir baixo consumo de energia e segurança para os dispositivos conectados.”. Características essenciais para aplicações IoT que necessitam garantir estabilidade e longevidade em suas aplicações que em certas situações são alimentadas por baterias ou fontes de energia limitadas.

A Espressif (2025a) define o ESP32-H2 como microcontrolador de característica, *system on a chip*(SoC), de baixo consumo de energia, com rádio de 2.4 GHz integrado. Sendo composto por um microprocessador RISC-V de 32 bits, uma *baseband* 802.15.4, um módulo de RF e outros muitos periféricos, como pode ser visto no diagrama de blocos da Figura 1.

Figura 1: Diagrama de blocos do ESP32-H2 SoC



Fonte: ESPRESSIF SYSTEMS (2025, p 2).

Como pode ser visto na Figura 1, o microcontrolador é baseado em um núcleo RISC-V de 32 bits, memória RAM (*Random Access Memory*) e ROM (*Read Only Memory*) integradas, além de múltiplas interfaces periféricas como *Inter-Integrated Circuit* (I2C), *Serial Peripheral Interface* (SPI), *Universal Asynchronous Receiver/Transmitter* (UART), conversor analógico-digital (ADC), *Pulse Width Modulation* (PWM) e *General-Purpose Input/Output* (GPIOs) configuráveis, o que viabiliza a conexão com uma ampla variedade de sensores e atuadores.

Uma funcionalidade relevante explorada neste trabalho é o sensor de temperatura interno do ESP32-H2, que de acordo com a Espressif (2025b, tradução nossa), “[...] permite o monitoramento em tempo real das variações de temperatura

dentro do chip.” Este sensor foi utilizado inicialmente durante o processo de metodologia deste projeto para testar o funcionamento da comunicação do ESP32-H2 utilizando a rede Zigbee enviando as informações de temperaturas medidas por esse sensor integrado.

O ESP32-H2 também tem como característica relevante seu preço acessível e soluções para um maior nível de segurança, fatores esses que geram interesses na mídia voltada a tecnologia, como a plataforma Medium, Embarcados e comunidades focadas em automação residencial entre outros. De acordo com a Medium (2025, tradução nossa), “O Espressif ESP32-H2 foi desenvolvido para oferecer uma solução de segurança acessível para todos e, por isso, integra uma variedade de recursos de segurança.”.

Portanto, este e outros fatos que podem ser encontrados na documentação oficial do ESP32-H2 (ESPRESSIF, 2025b) consolidam ele como uma plataforma flexível e eficiente para projetos que exigem baixo consumo de energia, múltiplos protocolos de comunicação e facilidade de integração, atributos que o tornam apropriados para aplicações IoT.

2.2 Protocolo de rede Zigbee

O protocolo Zigbee é um protocolo de comunicação sem fio baseado na especificação IEE 802.15.4, projetado especificamente para aplicações que exigem baixo consumo de energia, baixo custo e alta confiabilidade na transmissão de dados entre dispositivos.

As tecnologias Zigbee e Thread são protocolos baseados em padrões que operam principalmente na faixa de 2.4 GHz e oferecem uma infraestrutura integrada de rede em *mesh*, segurança e aplicações para dispositivos embarcados de baixo consumo de energia e baixo custo (Texas Instruments, 2018, p. 3. tradução nossa).

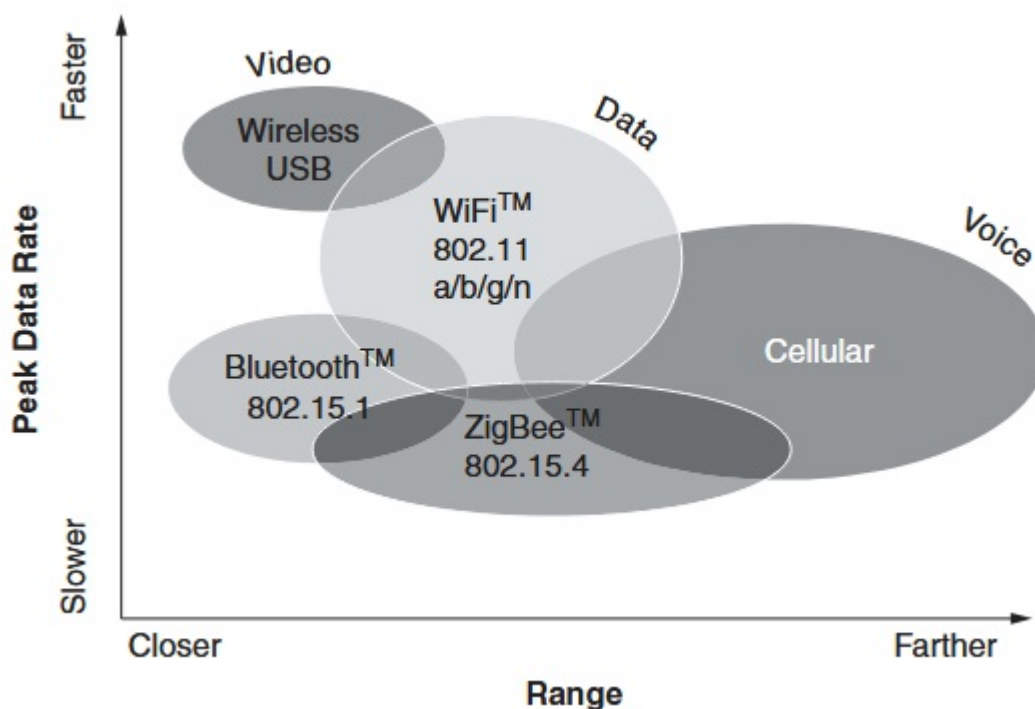
Entre as principais características do Zigbee está sua topologia flexível. De acordo com a Connectivity Standards Alliance (2025), o Zigbee foi desenvolvido com foco em oferecer variedade e flexibilidade para usuários e desenvolvedores. Isso se deve ao fato de seu desenvolvimento ter sido pensado para garantir compatibilidade retroativa e futura, permitindo que diferentes serviços e produtos

funcionem em conjunto por meio da padronização e da certificação de todas as camadas da pilha de protocolos.

No contexto deste trabalho, o protocolo Zigbee é empregado com o objetivo de viabilizar a comunicação local entre o microcontrolador ESP32-H2 e os dispositivos coordenadores responsáveis pela gestão da rede. Proporcionando suporte à construção de aplicações escaláveis, possibilitando a incorporação dinâmica de novos sensores e atuadores, sem que haja comprometimento do desempenho do sistema.

De acordo com Drew (2008), a frequência de rede utilizada pelo protocolo Zigbee preenche um espaço existente entre com relação a outros protocolos de rede, oferecendo um suporte para certas aplicações que é atendido por outros padrões *wireless*, como pode ser visto na Figura 2. Ainda de acordo com Drew (2008), Zigbee foi criado para ser um protocolo de rede *wireless* que seja fácil, simples e confiável.

Figura 2: Comparação de tecnologias wireless



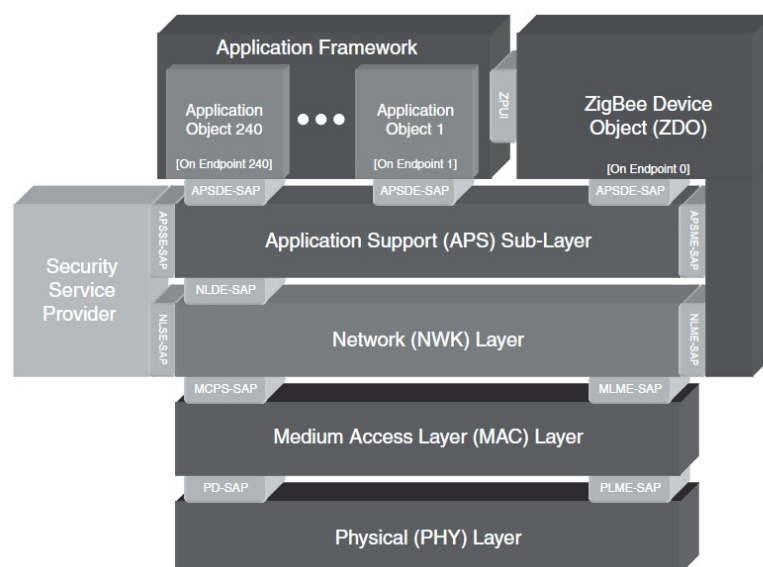
Fonte: DREW GISLASON (2008, p 10).

A tecnologia wireless vem se tornando mais importante a cada ano que se passa, de acordo com Wang (2016), nos últimos anos, houve um desenvolvimento significativo nas tecnologias de comunicação sem fio, como o Wi-Fi para redes locais sem fio (WLAN), o Bluetooth e o Zigbee para redes pessoais sem fio, entre outras, mostrando como mesmo após tanto tempo a rede Zigbee ainda é utilizada.

A interoperabilidade proporcionada pelo Zigbee com plataformas e dispositivos de diversos fabricantes, é possível graças ao uso de perfis padronizados para sensores de temperatura, umidade, atuadores, entre outros. Essa padronização facilita o desenvolvimento de soluções IoT o que permite com que afirmações como as da Connectivity Standards Alliance (2025, tradução nossa) sejam feitas, “O Zigbee é interoperável com grande ecossistema de produtos disponíveis que funcionam perfeitamente em conjunto”.

Sobre a arquitetura de pilhas do Zigbee, é explicado de forma resumida pelo autor Drew Gislason (2008), que cada camada da pilha não tem conhecimento sobre a camada acima dela, sendo a camada superior um “mestre” que comanda o seu escravo abaixo para realizar o trabalho, e conseqüentemente cada camada constrói níveis crescentes de sofisticação sobre a fundação das camadas inferiores.

Figura 3: Arquitetura Zigbee



Fonte: Drew Gislason (2008).

De acordo com Drew (2008), entre as camadas da pilha Zigbee existem os SAPs (*Service Access Points*), que funcionam como APIs, isolando o funcionamento interno de cada camada das demais, sendo que cada camada tem dois SAPs, um para dados e outro para gerenciamento. Pode-se classificar as camadas de abstração de protocolos vistas na Figura 3 como:

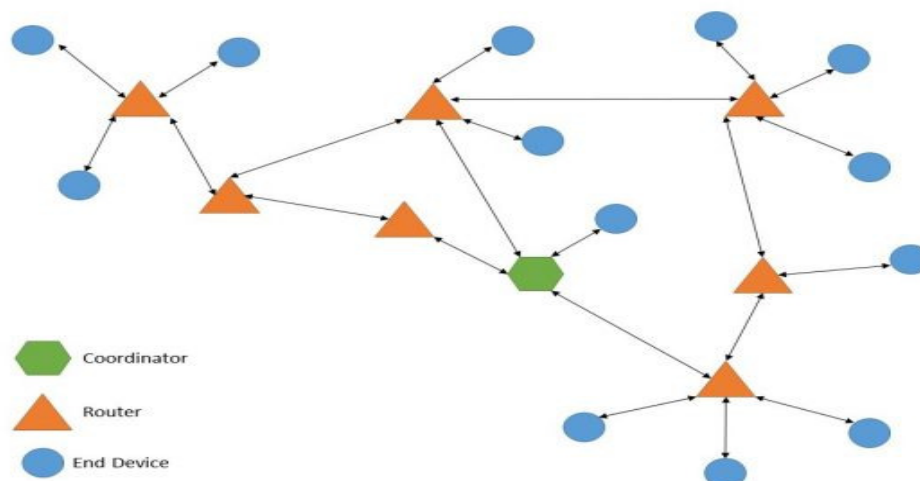
- *PHY layer*: Converte os pacotes em bits para transmissão via rádio.
- *MAC layer*: Identificação da rede, descoberta de dispositivos e confirmações ponto a ponto.
- *NWK layer*: Responsável pela rede *mesh*, incluindo roteamento de pacotes, segurança e criptografia de ponta a ponta.
- *APS sub-layer*: Entende a lógica das aplicações, como *clusters* e *endpoints*, filtrando mensagens duplicadas e mantendo uma tabela de associações (*binding*) para saber com quem o nó deve se comunicar.
- Camada ZDO: Cuida da gestão da rede, descobrindo nós e serviços, além de gerenciar o estado do dispositivo.
- O *Application Framework*: Contém a Zigbee Cluster Library (ZCL) e define os processos das aplicações, usando *endpoints* para separá-las.
- Final o *Security Services* pode ser usado por uma variedade de *layers*, ou pelos outros *layers* ZDO, APS ou NWK, sendo assim na figura ele é posicionado fora da pilha.

Para complementar o fluxo de dados da rede Zigbee, neste projeto foi adotado o protocolo MQTT no gateway, integrando a comunicação local Zigbee às plataformas de monitoramento e controle.

A Figura 4 ilustra a topologia de uma rede Zigbee típica, composta por um coordenador, múltiplos roteadores e dispositivos finais. Observa-se que o coordenador centraliza o gerenciamento da rede, enquanto os roteadores desempenham o papel de intermediários na comunicação Drew (2008), permitindo a expansão da área de cobertura e garantindo redundância nos caminhos de transmissão. Já os dispositivos finais, representados em azul, se conectam apenas a roteadores, sem desempenhar funções de encaminhamento. Essa organização

caracteriza a topologia em malha (*mesh*), a qual é uma das possíveis funções da rede Zigbee (Zigbee, 2025).

Figura 4: Topologia de uma rede Zigbee



Fonte: Flemmingss (2025).

2.2.1 Clusters

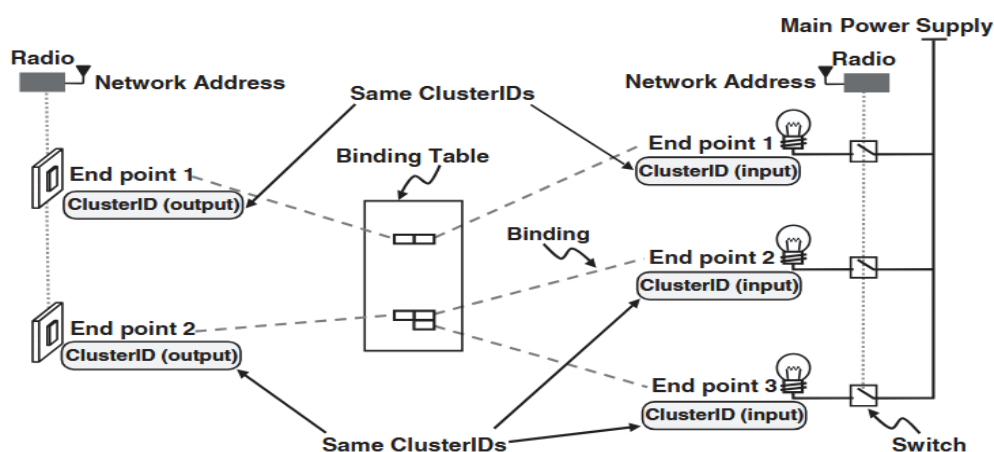
Qualquer desenvolvedor que estiver pensando em fazer algum tipo de aplicação utilizando Zigbee, tem como recomendação verificar a Zigbee Cluster Library (ZCL), essa “biblioteca” define um conjunto de funções padronizadas com base em cenários de aplicação comuns e nas características clássicas dos dispositivos, conhecidos como *clusters* (ESPRESSIF, 2025a). A Zigbee Cluster Library é um conjunto de elementos como *frame structures*, comandos de acesso a atributos e *data types* (Zigbee Alliance, 2025).

Um cluster pode ser considerado como um conjunto de atributos agrupados, onde cada *cluster* é identificado por um número exclusivo de 16 bits, chamado de *cluster identifier*, e cada atributo em um cluster também é identificado por um número exclusivo de 16 bits, chamado de *attribute identifier* (Farahani, 2008).

Esses atributos são utilizados para reservar dados ou valores de estado no *cluster* (Farahani, 2008). Um *cluster identifier* indica a qual conjunto de comandos um determinada instrução pertence (Gislason, 2008).

Por exemplo, em uma aplicação de controle de temperatura, um dispositivo que atua como sensor pode armazenar o valor da temperatura atual em um atributo. Em seguida, outro dispositivo que atua como controlador pode receber o valor desse atributo e usar essa informação para realizar algum tipo de comando (Farahani, 2008).

Figura 5: Binding em um exemplo de controle de luz



Fonte: Farahani (2008, p.116).

Outro exemplo que ilustra o uso dos *cluster identifier (clusterID)* e o processo de criação de *binding* que pode ser observado na Figura 5. O *binding* é o processo de estabelecer conexões lógicas entre aplicações relacionadas, em que dispositivos logicamente associados em uma *binding table* são chamados de *bound devices* (Farahani, 2008).

No exemplo de controle de luz mostrado na Figura 5, os dois interruptores estão na mesma rede e são diferenciados por *endpoints* distintos dentro da *binding table*. Consequentemente, cada interruptor pode ter seu próprio objeto de aplicação, acessado de forma independente por meio de seu respectivo *endpoint*. As lâmpadas também estão conectadas à mesma rede, cada uma com seu próprio *endpoint*, e ambas são controladas pelos dois interruptores. Como um *cluster* pode ser de entrada (*input cluster*) ou de saída (*output cluster*) (Zigbee Alliance, 2025), durante o processo de *binding*, dois dispositivos são considerados compatíveis se possuírem os mesmos *clusterIDs*, sendo um do tipo *input* e o outro do tipo *output*. Assim, como

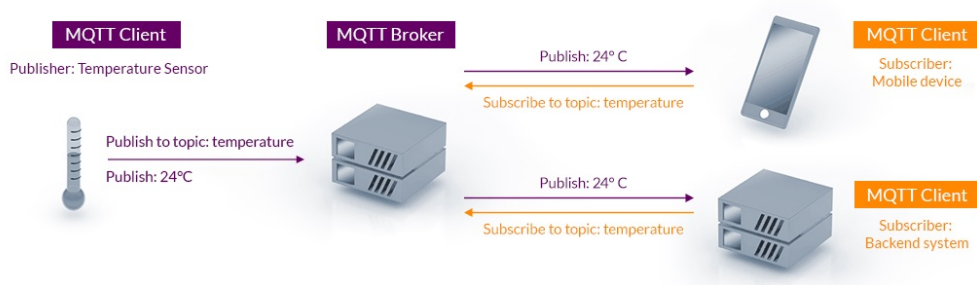
o interruptor com *endpoint1* e a lâmpada com *endpoint1* compartilham os mesmos *clusterIDs*, eles são considerados *bound devices*. O mesmo vale para o segundo interruptor e as outras duas lâmpadas (Farahani, 2008).

2.3 Protocolo MQTT

De acordo com a OASIS MQTT Technical Committee (2025, tradução nossa), “MQTT é um protocolo de mensagens padrão para a Internet das Coisas (IOT)”. A partir desta declaração, pode se entender que o MQTT é um protocolo de comunicação otimizado para a troca de mensagens em redes com recursos limitados e conexões instáveis ou de baixa largura de banda.

Assim como pode ser visto no protocolo de especificação publicado pela OASIS (2025), é especificado que o MQTT opera sobre um protocolo TCP/IP e utiliza um *broker* como intermediário central para gerenciar as mensagens publicadas por dispositivos (*publishers*) e entregá-las aos assinantes (*subscribers*). Essa arquitetura favorece uma escalabilidade e maior simplicidade na adição de novos dispositivos. De acordo com a OASIS (2025, tradução nossa) o “MQTT pode ser escalonado para milhões de dispositivos IoT”.

Figura 6: Arquitetura do MQTT Publish/Subscribe



Fonte: OASIS MQTT Technical Committee (2025).

No contexto deste trabalho, o MQTT é utilizado no *gateway* entre a rede Zigbee, formada pelo microcontrolador ESP32-H2 e os sensores conectados a ele, e as plataformas de gerenciamento de dados. Dessa forma, o ESP32-H2 envia os valores de temperatura, umidade e pressão, e o sistema de monitoramento recebem atualizações instantâneas ou periódicos dos valores medidos pelos sensores.

Essa integração entre Zigbee e MQTT permite que dispositivos como sensores de baixo consumo, operem em rede local eficiente (Zigbee) e ao mesmo tempo enviem dados para a internet ou para sistemas de controle remoto (MQTT), sem sobrecarregar a rede local com protocolos pesados.

De acordo com a OASIS (2025), alguns casos de usos atuais do MQTT são:

- Ambientes automotivos, como o HiveMQ utilizado pela BMW para ter uma conectividade confiável.
- Em logística para transportadoras.
- Manufatura onde o MQTT é usado em soluções industriais, de acordo com Bevywise (2025, tradução nossa), “Alcance eficiência e precisão incomparáveis conectando seus processos de fabricação, pessoas e máquinas com *insights* baseados em MQTT.”.
- Na indústria de energia onde o MQTT é implementado em inovações IoT para indústrias. De acordo com um caso de estudo feito pela Cirrus Link Solutions (Case Study: Energy Company Maximizing MQTT for control and Efficient Image Data transfer, pg. 3, tradução nossa), “O MQTT empacota e valida os dados da imagem utilizando muito menos largura de banda do que normalmente seria necessário para enviar uma imagem.”.
- O MQTT também é utilizado em automação de casas.
- O MQTT é utilizado em diferentes meios de transporte, desde aviação à até sistema ferroviário.

Finalmente, o uso do MQTT neste projeto consolida a proposta de criar uma solução IoT flexível, escalável e de fácil integração, viabilizando aplicações reais de automação residencial, monitoramento e controle remoto, com base no microcontrolador ESP32-H2 e a rede Zigbee.

2.4 Gateway ZigbeeMQT

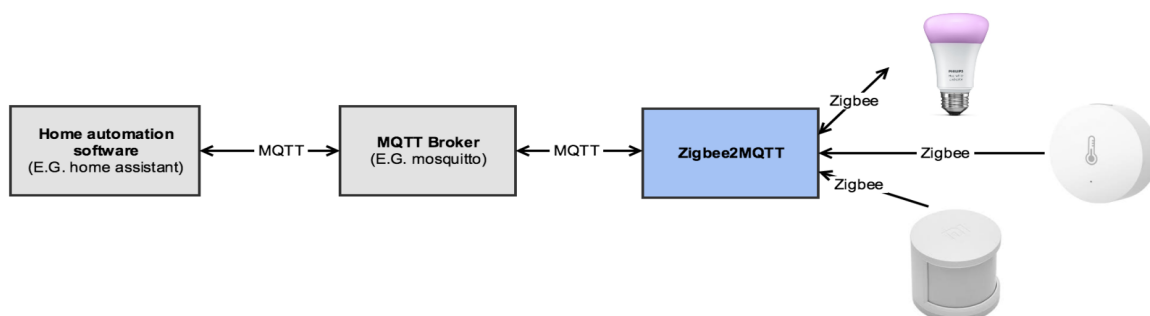
De acordo com a documentação oficial do Zigbee2MQTT (2025), o ZigbeeMQT permite que os dispositivos Zigbee sejam usados sem uma ponte ou *gateway* do fabricante, fazendo sua própria ponte de eventos via MQTT. Desta forma, é possível integrar dispositivos Zigbee com qualquer infraestrutura de automação residencial que estiver utilizando.

Para usar o ZigbeeMQTT, de acordo com o site oficial Zigbee2MQTT (2025), é preciso de um adaptador Zigbee, como o Zigbee 3.0 USB Dongle Plus fabricado pela empresa Sonoff, um *host system* como um computador e por fim um ou mais dispositivos que serão pareados com o ZigbeeMQTT.

A partir da documentação oficial do Zigbee2MQTT (2025), pode se entender que o ZigbeeMQTT é composto por três módulos, desenvolvidos em projetos *open-source* no GitHub, esses três módulos são: zigbee-herdsman, zigbee-herdsman-converts e Zigbee2MQTT. De acordo com a Zigbee2MQTT (2025) os módulos têm como característica:

- zigbee-herdsman: Conecta o adaptador Zigbee e disponibiliza uma API para os níveis superiores.
- zigbee-herdsman-converts: Lida com o mapeamento de modelos específicos de dispositivos para os clusters Zigbee que eles suportam.
- Zigbee2MQTT: Controla o zigbee-herdsman e faz o mapeamento das mensagens Zigbee para mensagens MQTT, também acompanhando o *status* do sistema.

Figura 7: Arquitetura ZigbeeMQTT



De acordo com a empresa Sonoff (2025), alguma das vantagens do Zigbee2MQTT são o seu vasto número de dispositivos suportados, permite personalizar as configurações dos dispositivos e parâmetros a rede, e por ser baseado em MQTT, é ideal para quem integra com outros sistemas baseados em MQTT. Ainda de acordo com a empresa Sonoff (2025), algumas desvantagens do Zigbee2MQTT é sua configuração complexa, exigindo um *broker* MQTT, uma curva de aprendizado alta e tendo alguns coordenadores que precisam de *firmware* personalizado para funcionar corretamente.

2.5 Sensor HTU2X

O sensor HTU2X é um sensor digital de medição de temperatura e umidade relativa do ar. De acordo com a fabricante TE connectivity (2025, tradução nossa), o sensor HTU2X detecta a umidade relativa do ar e a temperatura, oferecendo o melhor desempenho da sua categoria e um valor custo-benefício superior comparado a outros sensores disponíveis no mercado.

Uma das principais características do HTU2X é sua capacidade de operar em uma ampla faixa de umidade relativa (0–100%) e temperatura (-40 °C a 125 °C), além disso, é possível utilizar diferentes modos de operação com o sensor, incluindo medições em tempo real ou sob demanda, o que possibilita ajustar a frequência de amostragem de acordo com os requisitos do sistema TE connectivity (2025).

De acordo com a fabricante TE connectivity (2025), algumas aplicações para o sensor HTU2X são:

- Aquecedores, ventilação, ar-condicionado e refrigeradores.
- Equipamentos de construção.
- Equipamentos de controle.
- Máquinas de lavar.

Para o protótipo. O sensor HTU2X é utilizado para medir a temperatura de umidade e temperatura do ambiente, se comunicando com o ESP32-H2 através do barramento serial I2C, e finalmente tendo os valores medidos sendo transmitidos na rede Zigbee. A Figura 8 demonstra a aparência do sensor.

Figura 8: Sensor HTU2X



Fonte: TE connectivity (2025).

2.6 Resumo

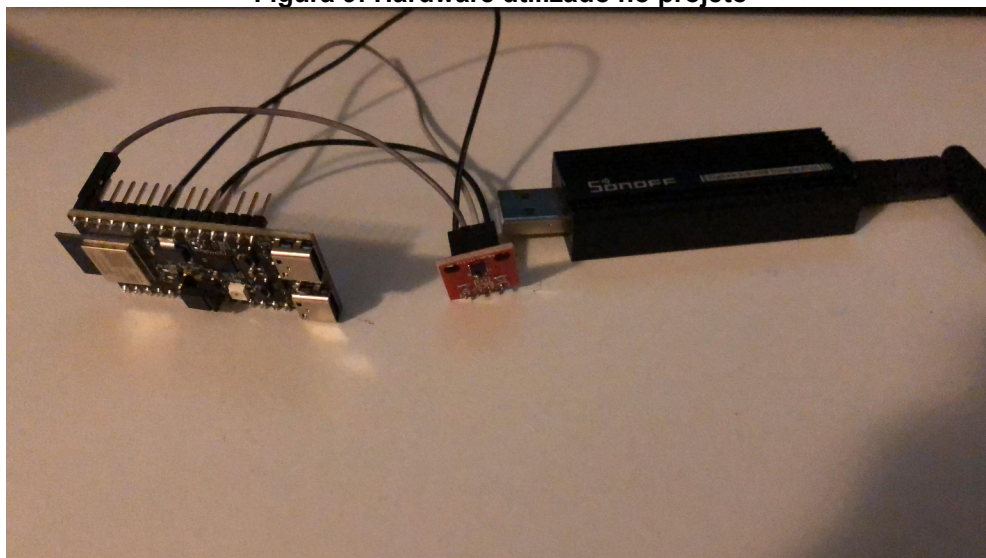
Finalmente, resumindo todos os dispositivos e tecnologias utilizadas neste trabalho.

- O ESP32-H2 foi o SoC selecionado, esse chip foi escolhido por já ter um protocolo Zigbee suportado.
- O *host* é o computador responsável por conectar o Zigbee Dongle e executar o serviço ZigbeeMQTT.
- O protocolo de mensagem MQTT sendo usado no gateway, temos então a comunicação e o pareamento dos dispositivos Zigbee com o ZigbeeMQTT.
- O sensor de temperatura e umidade utilizado para o projeto foi o sensor HTU2X, tendo suas medições transmitidas na rede Zigbee.

3 METODOLOGIA

Este trabalho visa analisar e validar de forma experimental, as capacidades do ESP32-H2 em um cenário de aplicação IoT, integrando comunicação local via Zigbee e publicação de dados utilizando o protocolo MQTT. Diante disso, foi criado um *endpoint* Zigbee utilizando o microcontrolador ESP32-H2 do DevKitM-1 (ESPRESSIF, 2025c), um Zigbee 3.0 USB Dongle Plus fabricado pela Sonoff, e um sensor de umidade e temperatura modelo HTU2X. A Figura 9 mostra o *hardware* utilizado nesse projeto.

Figura 9: Hardware utilizado no projeto



Fonte: Elaboração própria (2025).

A pesquisa adotou uma abordagem experimental e qualitativa, pois este estudo tem uma implementação de um protótipo funcional, a coleta de dados reais a partir do sensor, análise dos dados transmitidos e análise do desempenho do protótipo em relação a consumo de energia.

Neste capítulo, são detalhados os métodos utilizados para a implementação do protótipo, quais foram os passos tomados e os conhecimentos necessários para desenvolver esse projeto desde uma implementação de exemplo fornecida pelo espressif-sdk, até o momento final onde temos três *clusters* criados na rede Zigbee com valores sendo medidos pelo sensor HTU2X.

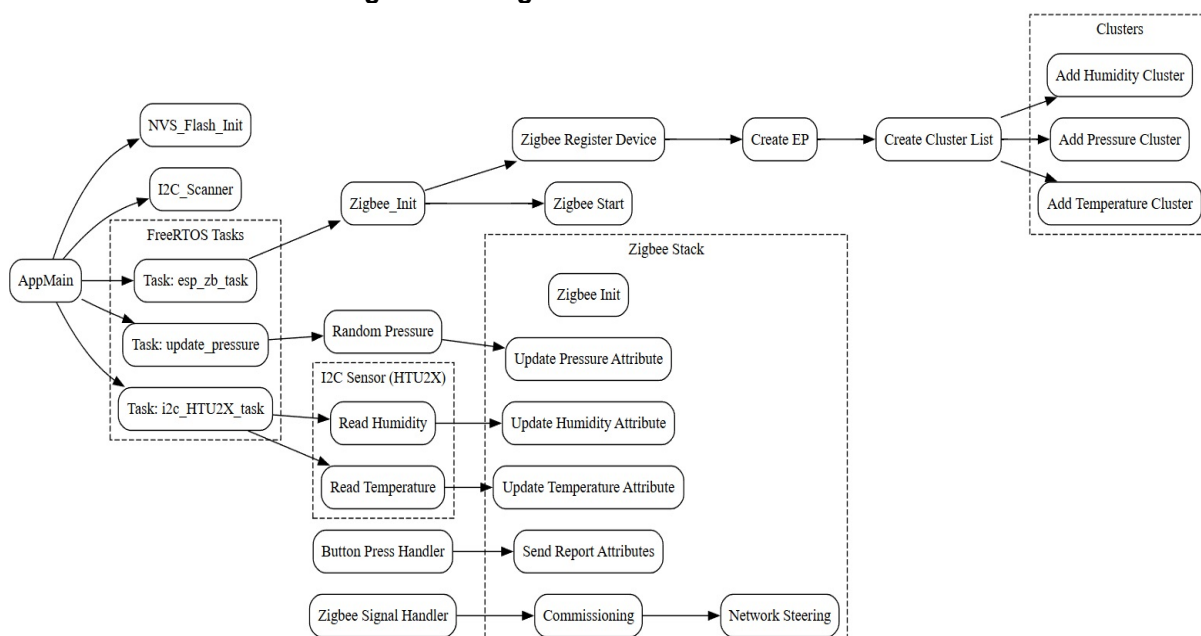
3.1 Métodos Aplicados

Em termos de hardware, temos como *chip* o ESP32-H2, um Zigbee USB Dongle, responsável por atuar como coordenador da rede Zigbee e *gateway* de integração com o *broker* MQTT, permitindo a formação da rede *mesh* local, o pareamento com o nó ESP32-H2 e a recepção dos dados provenientes dos sensores e um sensor de temperatura e umidade HTU2X.

O desenvolvimento do *firmware* foi realizado utilizando como base um dos únicos exemplos disponíveis no repositório oficial Zigbee, *esp-zigbee-sdk*, que disponibiliza uma variedade de APIs, sendo uma tentativa de simplificar o processo de desenvolvimento de produtos Zigbee, Espressif (2025a).

Para uma representação do *firmware*, foi criado o diagrama de blocos da Figura 10.

Figura 10: Diagrama de blocos do *firmware*



Fonte: Elaboração própria (2025).

A primeira etapa deste projeto teve com intuito entender e conseguir estabelecer a rede Zigbee com algum tipo de valor sendo comunicado entre o *host* e o sensor. Sendo assim o melhor caminho encontrado para isso foi utilizando o SDK Zigbee e um exemplo de aplicação disponível no repositório oficial do SDK, onde é

demonstrado como configurar um sensor de temperatura para automação residencial em um dispositivo Zigbee.

Inicialmente a medição de temperatura foi fornecida utilizando o sensor de temperatura integrado ao chip ESP32-H2. Para permitir que a medição do valor de temperatura do sensor interno do ESP32-H2 estivesse sendo transmitido na rede Zigbee, o gateway ZigbeeMQTT teve que ser inicializado, a fim de permitir a inclusão do novo sensor a rede, com a rede ativa e a partir da interface *html* do ZigbeeMQTT foi possível verificar que o ESP32-H2 estava conectado a rede, e que o valor de temperatura medido tinha sido atualizado. Após a confirmação que as medições do sensor estavam sendo transmitidas na rede Zigbee, foram então feitas mudanças para que o valor de temperatura fosse fornecida pelo sensor HTU2X, assim como o valor de umidade.

Pode ser confirmado que o código estava funcionando ao abrir a página da rede Zigbee, e verificar que o sensor estava conectado e ativo na rede como visto na Figura 11, que mostra os dispositivos encontrados na rede Zigbee, sendo o único dispositivo na lista o microcontrolador ESP32-H2 utilizado.

Figura 11: ZigbeeMQTT interface html

#	Pic	Friendly name	IEEE Address	Manufacturer	Model	LQI	Power
1		Custom Sensors	0x4831b7f1fec06e0b (0xc5AE1)	Unsupported	esp32h2	219	?

Fonte: Elaboração própria (2025).

A função *app_main* do Quadro 1 pode ser vista no *firmware* utilizado para a programação do ESP32-H2, essa função é o ponto de entrada principal da aplicação. Nela é configurado as tarefas que estão relacionadas ao Zigbee, e configura para que as três tarefas criadas com relação ao Zigbee sejam executadas de forma concorrente.

Quadro 1: Função *main* inicial do projeto

```
void app_main(void)
{
    esp_zb_platform_config_t config = {
        .radio_config = ESP_ZB_DEFAULT_RADIO_CONFIG(),
        .host_config = ESP_ZB_DEFAULT_HOST_CONFIG(),
    };
    ESP_ERROR_CHECK(nvs_flash_init());
    ESP_ERROR_CHECK(esp_zb_platform_config(&config));

    /* Start Zigbee stack task */
    xTaskCreate(esp_zb_task, "Zigbee_main", 4096, NULL, 5, NULL);
    xTaskCreate(update_pressure, "update_pressure_task", 4096, NULL, 3u, NULL);
    xTaskCreate(i2c_HTU2X_task, "i2c_HTU2X_task", 4096, NULL, 5, NULL);
}
```

Fonte: Elaboração própria (2025).

O próximo passo tomado para desenvolver o projeto, foi entender como criar um novo “*cluster*”. De acordo com a definição para *cluster* por Zigbee2MQTT (2025, tradução nossa).

“O Zigbee é mais do que apenas uma tecnologia de rede; ele define uma série de aplicações padrão que rodam na rede. Essas aplicações são definidas na especificação da Biblioteca de Clusters Zigbee. Cada “*cluster*” define um tipo de comunicação de aplicação. Cada tipo de *cluster* tem seu próprio ID inteiro e está disponível em dois tipos: cliente e servidor” (Zigbee2MQTT, 2025, tradução nossa).

Sendo assim, como os *clusters* Zigbee estão relacionados com a padronização dos sensores, foi preciso pesquisar qual seria o *cluster ID* das medições desejadas, esta informação pode ser encontrada na documentação do ESP Zigbee SDK. Para o protótipo deste trabalho foram utilizadas as informações de temperatura, pressão e umidade, sendo os *clusters ID* para essas medições

ESP_ZB_ZCL_CLUSTER_ID_TEMP_MEASUREMENT	(0x0402),
ESP_ZB_ZCL_CLUSTER_ID_PRESSURE_MEASUREMENT	(0x0403),
ESP_ZB_ZCL_CLUSTER_ID_REL_HUMIDITY_MEASUREMENT	(0x0405).

Salienta-se que embora o Zigbee SDK possua certas estruturas já preparadas para vários sensores, ainda é necessário por parte do desenvolvedor

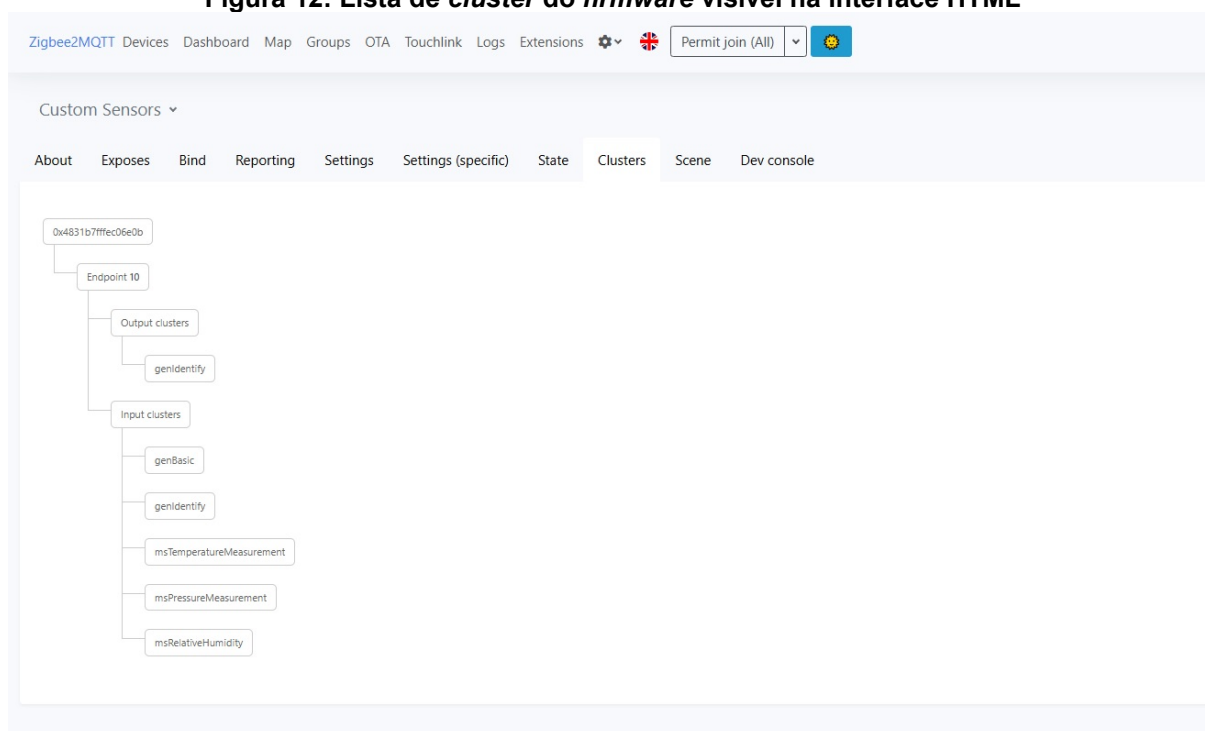
entender como a comunicação entre os sensores e o microcontrolador acontecem, devido à necessidade de implementar funções que realizem a interpretação das informações passadas pelo sensor. No caso deste projeto foi utilizado a comunicação I2C para efetuar a comunicação entre o sensor HTU2X e o ESP32-H2.

Tendo estas informações, após confirmar o funcionamento do sistema com o sensor de temperatura do ESP32-H2, foi criado um sensor de pressão virtual com valores gerados de forma aleatória a cada certo período de tempo para que fosse entendido como adicionar novos *clusters* em uma aplicação onde já existisse um outro *cluster* sendo utilizado. Finalmente foi adicionado um último *cluster* para leitura de umidade e alterado o *cluster* de temperatura para que ambos tivessem os valores sendo transmitidos pelo sensor HTU2X acoplado ao ESP32-H2.

Todo o desenvolvimento de código utilizado neste projeto foi feito utilizando a IDE VSCode com a extensão para chips da Espressif, esta extensão está disponível de forma gratuita até o momento de criação deste projeto. O sistema operacional utilizado ao longo do desenvolvimento foi o Windows.

Na Figura 12 é apresentada a continuidade da interface *HTML* anteriormente demonstrada. Nesta etapa, observa-se a aba destinada aos *clusters*, onde é exibida a lista de *clusters* implementados no *firmware* do ESP32-H2. Considerando o planejamento do sistema, no qual três variáveis de medição deveriam ser transmitidas pela rede Zigbee, foram desenvolvidos *clusters* específicos para cada tipo de medição monitorada. Dessa forma, a interface permite a visualização direta desses *clusters*, evidenciando a estrutura de dados configurada no dispositivo.

Figura 12: Lista de *cluster* do *firmware* visível na interface HTML



Fonte: Elaboração própria (2025).

Para viabilizar a criação da lista de clusters necessária ao projeto, foi desenvolvida a função apresentada no Quadro 2. O objetivo principal dessa função é instanciar e configurar a lista de *clusters* Zigbee associados às medições definidas. A função recebe como parâmetro de entrada uma estrutura de dados que armazena as configurações individuais criadas para este protótipo, permitindo a correta inicialização dos *clusters* correspondentes.

Nesta função primeiramente é inicializado uma lista vazia para adição dos *clusters* e então um *cluster* básico é definido para que em sequências os atributos personalizados sejam adicionados a ele, sendo eles o ID do modelo “esp32-h2” e o nome do fabricante. O *cluster* básico é então definido como um servidor na lista de *clusters* (servindo essas informações para quem requisitar), e então adicionado um *Identify cluster* para a aplicação até que finalmente é criado e adicionado o *cluster* de leitura para temperatura, umidade e pressão.

Quadro 2: Criação da lista de *clusters* utilizado no *firmware*

```

static esp_zb_cluster_list_t *custom_sensors_clusters_create(custom_sensor_cfg_t *yuri_sensor)
{
    ...

    // [Sensores]
    ESP_ERROR_CHECK(esp_zb_cluster_list_add_temperature_meas_cluster(cluster_list,
    esp_zb_temperature_meas_cluster_create(&(yuri_sensor->temp_meas_cfg)),
    ESP_ZB_ZCL_CLUSTER_SERVER_ROLE));
    ESP_ERROR_CHECK(esp_zb_cluster_list_add_pressure_meas_cluster(cluster_list,
    esp_zb_pressure_meas_cluster_create(&(yuri_sensor->pressure_meas_cfg)),
    ESP_ZB_ZCL_CLUSTER_SERVER_ROLE));
    ESP_ERROR_CHECK(esp_zb_cluster_list_add_humidity_meas_cluster(cluster_list,
    esp_zb_humidity_meas_cluster_create(&(yuri_sensor->humidity_meas_cfg)),
    SP_ZB_ZCL_CLUSTER_SERVER_ROLE));
    return cluster_list;
}

```

Fonte: Elaboração própria (2025).

Para os três *clusters* criados neste projeto, todos foram configurados para reportar seus dados automaticamente, através da função *esp_zb_task* vista no Quadro 3 e 4. A função *esp_zb_task()* é responsável pela inicialização e configuração dos sensores no contexto da pilha Zigbee. Inicialmente, é criada a estrutura *sensor_cfg*, que define valores padrão para os parâmetros do sensor. Em seguida, são especificados os limites mínimo e máximo para a medição de temperatura (*min_value* e *max_value*), convertidos para o formato esperado pelo protocolo Zigbee por meio da função *zb_temperature_to_s16()*.

E finalmente pode ser visto a parte onde é feito a configuração do relatório periódico de atributos Zigbee (*reporting*). Para isso, é definida a estrutura *esp_zb_zcl_reporting_info_t temp_reporting_info*, que especifica os parâmetros de envio do atributo de temperatura, sendo então feito a chamada onde é aplicado todas as configurações e habilitando o protótipo a enviar automaticamente os valores de temperatura para a rede Zigbee, o mesmo é feito para a medição de umidade e pressão.

Quadro 3: Report de dados automático para todos os de temperatura

```

static void esp_zb_task(void *pvParameters)
{
    custom_sensor_cfg_t sensor_cfg = ESP_ZB_DEFAULT_CUSTOM_SENSOR_CONFIG();
    sensor_cfg.temp_meas_cfg.min_value = zb_temperature_to_s16(ESP_TEMP_SENSOR_MIN_VALUE);
    sensor_cfg.temp_meas_cfg.max_value = zb_temperature_to_s16(ESP_TEMP_SENSOR_MAX_VALUE);
    esp_zb_ep_list_t *esp_zb_sensor_ep = custom_sensor_yuri_ep_create(HA_ESP_SENSOR_ENDPOINT,
&sensor_cfg);
    ...
    esp_zb_zcl_reporting_info_t temp_reporting_info = {
        .direction = ESP_ZB_ZCL_CMD_DIRECTION_TO_SRV,
        .ep = HA_ESP_SENSOR_ENDPOINT,
        .cluster_id = ESP_ZB_ZCL_CLUSTER_ID_TEMP_MEASUREMENT,
        .cluster_role = ESP_ZB_ZCL_CLUSTER_SERVER_ROLE,
        .dst.profile_id = ESP_ZB_AF_HA_PROFILE_ID,
        .u.send_info.min_interval = 1,
        .u.send_info.max_interval = 0,
        .u.send_info.def_min_interval = 1,
        .u.send_info.def_max_interval = 0,
        .u.send_info.delta.u16 = 100,
        .attr_id = ESP_ZB_ZCL_ATTR_TEMP_MEASUREMENT_VALUE_ID,
        .manuf_code = ESP_ZB_ZCL_ATTR_NON_MANUFACTURER_SPECIFIC,
    };
    esp_zb_zcl_update_reporting_info(&temp_reporting_info);
    ...
}

```

Fonte: Elaboração própria (2025).

Quadro 4: Report de dados automático para todos os de pressão e umidade

```

static void esp_zb_task(void *pvParameters)
{
...
    /* Pressure reporting */
    esp_zb_zcl_reporting_info_t pressure_reporting_info = {
        .direction = ESP_ZB_ZCL_CMD_DIRECTION_TO_SRV,
        .ep = HA_ESP_SENSOR_ENDPOINT,
        .cluster_id = ESP_ZB_ZCL_CLUSTER_ID_PRESSURE_MEASUREMENT,
        .cluster_role = ESP_ZB_ZCL_CLUSTER_SERVER_ROLE,
        .dst.profile_id = ESP_ZB_AF_HA_PROFILE_ID,
        .u.send_info.min_interval = 1,
        .u.send_info.max_interval = 0,
        .u.send_info.def_min_interval = 1,
        .u.send_info.def_max_interval = 0,
        .u.send_info.delta.u16 = 100,
        .attr_id = ESP_ZB_ZCL_ATTR_PRESSURE_MEASUREMENT_VALUE_ID,
        .manuf_code = ESP_ZB_ZCL_ATTR_NON_MANUFACTURER_SPECIFIC,
    };
    esp_zb_zcl_update_reporting_info(&pressure_reporting_info);
    /* Humidity reporting */
    esp_zb_zcl_reporting_info_t humidity_reporting_info = {
        .direction = ESP_ZB_ZCL_CMD_DIRECTION_TO_SRV,
        .ep = HA_ESP_SENSOR_ENDPOINT,
        .cluster_id = ESP_ZB_ZCL_CLUSTER_ID_REL_HUMIDITY_MEASUREMENT,
        .cluster_role = ESP_ZB_ZCL_CLUSTER_SERVER_ROLE,
        .dst.profile_id = ESP_ZB_AF_HA_PROFILE_ID,
        .u.send_info.min_interval = 1,
        .u.send_info.max_interval = 0,
        .u.send_info.def_min_interval = 1,
        .u.send_info.def_max_interval = 0,
        .u.send_info.delta.u16 = 100,
        .attr_id = ESP_ZB_ZCL_ATTR_REL_HUMIDITY_MEASUREMENT_VALUE_ID,
        .manuf_code = ESP_ZB_ZCL_ATTR_NON_MANUFACTURER_SPECIFIC,
    };
    esp_zb_zcl_update_reporting_info(&humidity_reporting_info);
...
    esp_zb_zcl_update_reporting_info(&humidity_reporting_info);
    esp_zb_set_primary_network_channel_set(ESP_ZB_PRIMARY_CHANNEL_MASK);
    ESP_ERROR_CHECK(esp_zb_start(false));
    esp_zb_stack_main_loop();
}

```

Fonte: Elaboração própria (2025).

A tarefa *esp_zb_task* atualiza a *stack* Zigbee definindo o dispositivo como Zigbee End Device (ZED), é então utilizado uma configuração padrão criada especialmente para este projeto.

Para todas as tarefas, é utilizado a configuração padrão para os *clusters* como visto no Quadro 5, definindo os valores mínimos e máximos medido pelo sensor. O *endpoint* e seus *clusters* são registrados como parte do dispositivo Zigbee tornando o sensor “visível” para outros dispositivos na rede Zigbee. Finalizando com a configuração opcional de *reporting* periódicos dos valores de medição de temperatura, enviando atualizações do valor medido de forma automática.

Quadro 5: Configuração padrão para o sensor

```
#define ESP_ZB_DEFAULT_CUSTOM_SENSOR_CONFIG()
{
    .basic_cfg =
    {
        .zcl_version = ESP_ZB_ZCL_BASIC_ZCL_VERSION_DEFAULT_VALUE,
        .power_source = ESP_ZB_ZCL_BASIC_POWER_SOURCE_DEFAULT_VALUE,
    },
    .identify_cfg =
    {
        .identify_time = ESP_ZB_ZCL_IDENTIFY_IDENTIFY_TIME_DEFAULT_VALUE,
    },
    .temp_meas_cfg =
    {
        .measured_value = ESP_ZB_ZCL_TEMP_MEASUREMENT_MEASURED_VALUE_DEFAULT,
        .min_value = ESP_ZB_ZCL_TEMP_MEASUREMENT_MIN_MEASURED_VALUE_DEFAULT,
        .max_value = ESP_ZB_ZCL_TEMP_MEASUREMENT_MAX_MEASURED_VALUE_DEFAULT,
    },
    .pressure_meas_cfg =
    {
        .measured_value = 122,
        .min_value = 1,
        .max_value = 1000,
    },
    .humidity_meas_cfg =
    {
        .measured_value = ESP_ZB_ZCL_REL_HUMIDITY_MEASUREMENT_MEASURED_VALUE_DEFAULT,
        .min_value = ESP_ZB_ZCL_REL_HUMIDITY_MEASUREMENT_MIN_MEASURED_VALUE_DEFAULT,
        .max_value = ESP_ZB_ZCL_REL_HUMIDITY_MEASUREMENT_MAX_MEASURED_VALUE_DEFAULT,
    },
};

typedef struct {
    esp_zb_basic_cluster_cfg_t basic_cfg;
    esp_zb_identify_cluster_cfg_t identify_cfg;
    esp_zb_temperature_meas_cluster_cfg_t temp_meas_cfg;
    esp_zb_pressure_meas_cluster_cfg_t pressure_meas_cfg;
    esp_zb_humidity_meas_cluster_cfg_t humidity_meas_cfg;
} custom_sensor_cfg_t;
```

Fonte: Elaboração própria (2025).

A função `esp_app_humidity_sensor_handler` disponível no Quadro 6, é utilizada para a criação do *handler* responsável por receber um valor de umidade lido pelo sensor e atualizá-lo no atributo correspondente do *cluster* Zigbee. A função `esp_zb_zcl_set_attribute_val` é onde o valor está sendo atualizado no *cluster* de medição de umidade relativa.

Quadro 6: Função *handler* para *cluster* de umidade

```
static void esp_app_humidity_sensor_handler(float humidity)
{
    int16_t measured_value = (int16_t)(humidity * 100);
    esp_zb_lock_acquire(portMAX_DELAY);
    esp_zb_zcl_set_attribute_val(
        HA_ESP_SENSOR_ENDPOINT,
        ESP_ZB_ZCL_CLUSTER_ID_REL_HUMIDITY_MEASUREMENT,
        ESP_ZB_ZCL_CLUSTER_SERVER_ROLE,
        ESP_ZB_ZCL_ATTR_REL_HUMIDITY_MEASUREMENT_VALUE_ID,
        &measured_value,
        false);
    esp_zb_lock_release();
}
```

Fonte: Elaboração própria (2025).

Outra função de *handler* também foi criada para o sensor de temperatura, como visto no Quadro 7.

Quadro 7: Função *handler* para *cluster* de temperatura

```
static void esp_app_temp_sensor_handler(float temperature)
{
    int16_t measured_value = zb_temperature_to_s16(temperature);
    esp_zb_lock_acquire(portMAX_DELAY);
    esp_zb_zcl_set_attribute_val(
        HA_ESP_SENSOR_ENDPOINT,
        ESP_ZB_ZCL_CLUSTER_ID_TEMP_MEASUREMENT,
        ESP_ZB_ZCL_CLUSTER_SERVER_ROLE,
        ESP_ZB_ZCL_ATTR_TEMP_MEASUREMENT_VALUE_ID,
        &measured_value,
        false);
    esp_zb_lock_release();
}
```

Fonte: Elaboração própria (2025).

Continuando, a função *read_humidity* de leitura de umidade, realiza a leitura dos dados a partir do sensor HTU2X através do barramento serial I2C. Nesta função é enviado o comando para iniciar a medição de umidade, os dados de medição são lidos em três *bytes*, processados e convertidos para umidade real em porcentagem, como visto no Quadro 8.

Quadro 8: Função de leitura de umidade

```
esp_err_t read_humidity(i2c_master_dev_handle_t *dev_handle, float *humidity)
{
    esp_err_t ret;

    // Comando: TRIGGER_HUMD_MEASURE_NOHOLD (0xF5)
    uint8_t cmd = 0xF5;
    ret = i2c_master_transmit(*dev_handle, &cmd, 1, -1);
    if (ret != ESP_OK) return ret;

    // Aguarda tempo de conversão (típico ~16ms, máx 29ms)
    vTaskDelay(pdMS_TO_TICKS(30));

    // Lê 3 bytes: MSB, LSB, CRC
    uint8_t rx_buf[3] = {0};
    ret = i2c_master_receive(*dev_handle, rx_buf, sizeof(rx_buf), -1);
    if (ret != ESP_OK) return ret;

    // Converte os dados (mascara os bits de status)
    uint16_t raw_humidity = ((uint16_t)rx_buf[0] << 8) | (rx_buf[1] & 0xFC);
    *humidity = -6.0 + 125.0 * ((float)raw_humidity / 65536.0);

    return ESP_OK;
}
```

Fonte: Elaboração própria (2025).

Aproveitando que o sensor HTU2X também tem a funcionalidade para medir temperatura, então foi necessário a criação da função *read_temperature* para a leitura do valor de temperatura como visto no Quadro 9.

Quadro 9: Função de leitura de temperatura

```

esp_err_t read_temperature(i2c_master_dev_handle_t *dev_handle, float *temp_c) {

    esp_err_t ret;

    // Envia comando: TRIGGER_TEMP_MEASURE_NOHOL
    uint8_t cmd = TRIGGER_TEMP_MEASURE_NOHOLD;
    ret = i2c_master_transmit(*dev_handle, &cmd, 1, -1);
    if (ret != ESP_OK) return ret;

    // Espera a leitura (typically ~50ms)
    vTaskDelay(pdMS_TO_TICKS(50));

    // Lê 3 bytes: MSB, LSB, CRC
    uint8_t rx_buf[3] = {0};
    ret = i2c_master_receive(*dev_handle, rx_buf, sizeof(rx_buf), -1);
    if (ret != ESP_OK) return ret;

    // Converte os dados
    uint16_t raw_temp = ((uint16_t)rx_buf[0] << 8) | (rx_buf[1] & 0xFC);
    *temp_c = -46.85 + 175.72 * ((float)raw_temp / 65536.0);

    return ESP_OK;
}

```

Fonte: Elaboração própria (2025).

Finalmente foi criada a tarefa *i2c_HTU2X_task* responsável para as medições do sensor HTU2X na rede Zigbee. Esta função configura e inicializa o barramento I2C, em consequência adiciona o sensor HTU2X como um dispositivo no barramento como escravo, e finalmente fazendo uma leitura a cada 60 segundos dos valores de temperatura e umidade obtidos pelo sensor. Esta função também é inicializada na *main* do projeto.

A tarefa *i2c_HTU2X_task* utiliza as funções de leitura dos dados de temperatura e umidade para a atualização dos dados nos *cluster* Zigbee, porém também escreve no terminal do usuário os valores obtidos na medição, essa tarefa pode ser vista por completo no Quadro 10.

Quadro 10: Tarefa *i2c_HTU2X_task* leitura do sensor

```

static void i2c_HTU2X_task(void *pvParameters)
{
    i2c_master_dev_handle_t dev_handle;
    i2c_master_bus_config_t i2c_mst_config = {
        .clk_source = I2C_CLK_SRC_DEFAULT,
        .i2c_port = I2C_PORT_NUM_0,
        .scl_io_num = I2C_MASTER_SCL_IO,
        .sda_io_num = I2C_MASTER_SDA_IO,
        .glitch_ignore_cnt = 7,
        .flags.enable_internal_pullup = true,
    };
    i2c_master_bus_handle_t bus_handle;
    ESP_ERROR_CHECK(i2c_new_master_bus(&i2c_mst_config, &bus_handle));

    i2c_device_config_t dev_cfg = {
        .dev_addr_length = I2C_ADDR_BIT_LEN_7,
        .device_address = HTU2X_ADDR,
        .scl_speed_hz = 100000,
    };
    ESP_ERROR_CHECK(i2c_master_bus_add_device(bus_handle, &dev_cfg, &dev_handle));

    float temperature = 0.0f;
    float humidity = 0.0f;
    while (1) {
        if (read_temperature(&dev_handle, &temperature) == ESP_OK) {
            ESP_LOGI("HTU2X", "Temperature: %.2f °C", temperature);
            esp_app_temp_sensor_handler(temperature);
        }
        else {
            ESP_LOGE("HTU2X", "Failed to read temperature values");
        }
        if (read_humidity(&dev_handle, &humidity) == ESP_OK) {
            ESP_LOGI("HTU2X", "Humidity: %.2f", humidity);
            esp_app_humidity_sensor_handler(humidity);
        }
        else {
            ESP_LOGE("HTU2X", "Failed to read humidity values");
        }
        vTaskDelay(10000);
    }
}

```

Fonte: Elaboração própria (2025).

É interessante comentar que a ponto de estudo, um *cluster* de pressão foi criado tendo seus valores medidos a partir de uma função que atualiza o valor de medição com um número aleatório de 0 a 1000 a cada 60 segundos. Esse *cluster* está presente no sistema final, e pode ser visto na lista de *cluster* da Figura 12.

Uma função chamada *i2c_scanner* foi criada e utilizada para verificar se o sensor HTU2X estava conectado e funcionando com o barramento I2C. Essa função permitiu confirmar que o endereço para comunicação com o sensor HTU2X estava correto. Desta forma confirmando que o sensor HTU2X estava funcionando e principalmente confirmando se o endereço encontrado para este sensor realmente condiziam com os fornecidos na referência, sendo ele 0x40.

Em todos os casos de testes durante o desenvolvimento deste projeto, não houve variação de ambiente para testes do protótipo. Em todos os momentos onde os dispositivos estavam ligados, o ESP32-H2 se manteve próximo ao coordenador da rede, o Zigbee Dongle

4 APRESENTAÇÃO DOS RESULTADOS

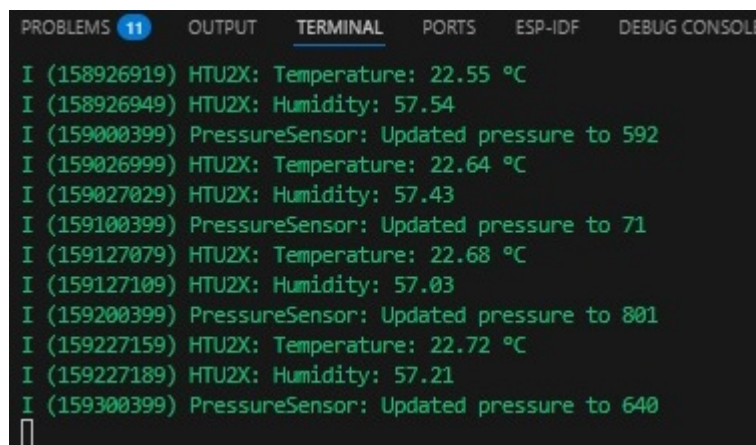
Este capítulo apresenta os resultados obtidos a partir das comunicações realizadas com sucesso entre o sensor e a rede Zigbee. A confirmação que as trocas de informações entre o sensor e o chip estavam acontecendo podem ser vistas pelos *logs* nos terminais de desenvolvimento e principalmente na interface *HTML* do Zigbee2MQTT.

Adicionalmente, será mostrado o consumo de energia do protótipo medidas a partir do uso do *hardware* Power Profiler Kit II da NORDIC semiconductor, NORDIC (2025). O consumo de energia será usado para a análise se o sistema está apto para uma aplicação que é necessário o mínimo possível de gastos de energia.

4.1 Análise e discussão dos resultados

Durante todo o avanço do projeto, foi utilizado dois terminais como interface para verificar através de *logs* do *firmware* e do *software* Zigbee2MQTT os valores das medições sendo atualizados e enviados na rede. No Quadro 10 pode-se notar as chamadas de ESP_LOG que enviam que os valores de temperatura e umidade no terminal da IDE VSCode utilizada para o desenvolvimento do *firmware*, como pode ser visto na Figura 13.

Figura 13: Log de medição gerado pelo *firmware* no terminal do VSCode

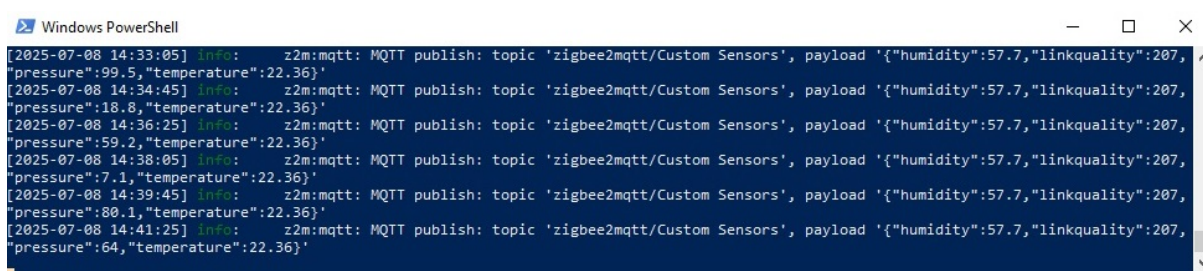


```
PROBLEMS 11 OUTPUT TERMINAL PORTS ESP-IDF DEBUG CONSOLE
I (158926919) HTU2X: Temperature: 22.55 °C
I (158926949) HTU2X: Humidity: 57.54
I (159000399) PressureSensor: Updated pressure to 592
I (159026999) HTU2X: Temperature: 22.64 °C
I (159027029) HTU2X: Humidity: 57.43
I (159100399) PressureSensor: Updated pressure to 71
I (159127079) HTU2X: Temperature: 22.68 °C
I (159127109) HTU2X: Humidity: 57.03
I (159200399) PressureSensor: Updated pressure to 801
I (159227159) HTU2X: Temperature: 22.72 °C
I (159227189) HTU2X: Humidity: 57.21
I (159300399) PressureSensor: Updated pressure to 640
```

Fonte: Elaboração própria (2025).

Enquanto isso, no terminal onde temos o *software* Zigbee2MQTT ativado, pode ser visto a mensagem quando alguma medição é transmitida na rede, podendo também ser visto outras informações do *payload* da mensagem como visto na Figura 14.

Figura 14: Log de envio na rede Zigbee visto pelo terminal com o Zigbee2MQTT ativo



```

[2025-07-08 14:33:05] info: z2m:mqtt: MQTT publish: topic 'zigbee2mqtt/Custom Sensors', payload '{"humidity":57.7,"linkquality":207,"pressure":99.5,"temperature":22.36}'
[2025-07-08 14:34:45] info: z2m:mqtt: MQTT publish: topic 'zigbee2mqtt/Custom Sensors', payload '{"humidity":57.7,"linkquality":207,"pressure":18.8,"temperature":22.36}'
[2025-07-08 14:36:25] info: z2m:mqtt: MQTT publish: topic 'zigbee2mqtt/Custom Sensors', payload '{"humidity":57.7,"linkquality":207,"pressure":59.2,"temperature":22.36}'
[2025-07-08 14:38:05] info: z2m:mqtt: MQTT publish: topic 'zigbee2mqtt/Custom Sensors', payload '{"humidity":57.7,"linkquality":207,"pressure":7.1,"temperature":22.36}'
[2025-07-08 14:39:45] info: z2m:mqtt: MQTT publish: topic 'zigbee2mqtt/Custom Sensors', payload '{"humidity":57.7,"linkquality":207,"pressure":80.1,"temperature":22.36}'
[2025-07-08 14:41:25] info: z2m:mqtt: MQTT publish: topic 'zigbee2mqtt/Custom Sensors', payload '{"humidity":57.7,"linkquality":207,"pressure":64,"temperature":22.36}'
  
```

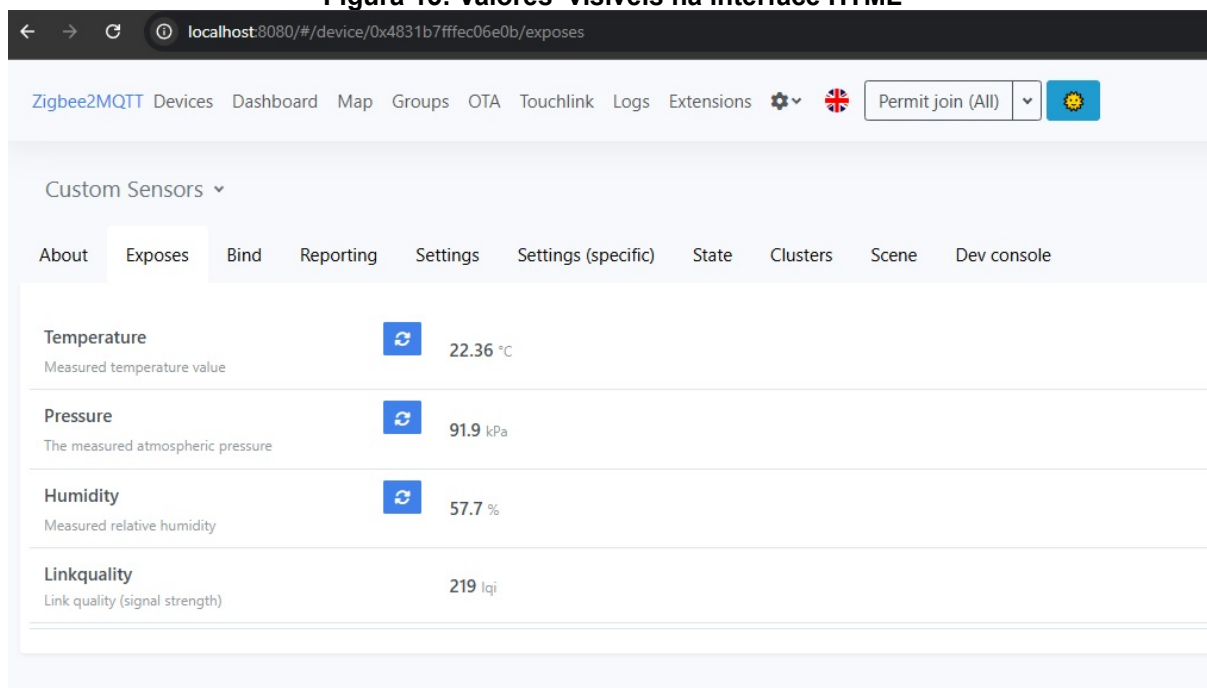
Fonte: Elaboração própria (2025).

Embora seja possível averiguar os valores medidos pelo sensor, em uma aplicação real o chip não enviaria *logs* periodicamente, todavia as informações disponíveis podem ser acompanhadas diretamente por meio da interface HTML gerada pelo Zigbee2MQTT.

A partir deste ponto, as figuras apresentadas na subsequência referem-se à interface HTML disponibilizada pelo Zigbee2MQTT. Além das informações previamente demonstradas, essa interface permite a visualização de dados adicionais, tais como o valor atual das medições, a lista de clusters implementados no *firmware*, o mapa de topologia da rede (incluindo o coordenador, o *dongle* Zigbee e o dispositivo ESP32-H2), bem como uma opção para calibração das medições.

Sendo assim na Figura 15, pode ser visto os valores medidos pelo sensor e transmitidos na rede Zigbee, sendo disponibilizados na interface HTML do Zigbee2MQTT.

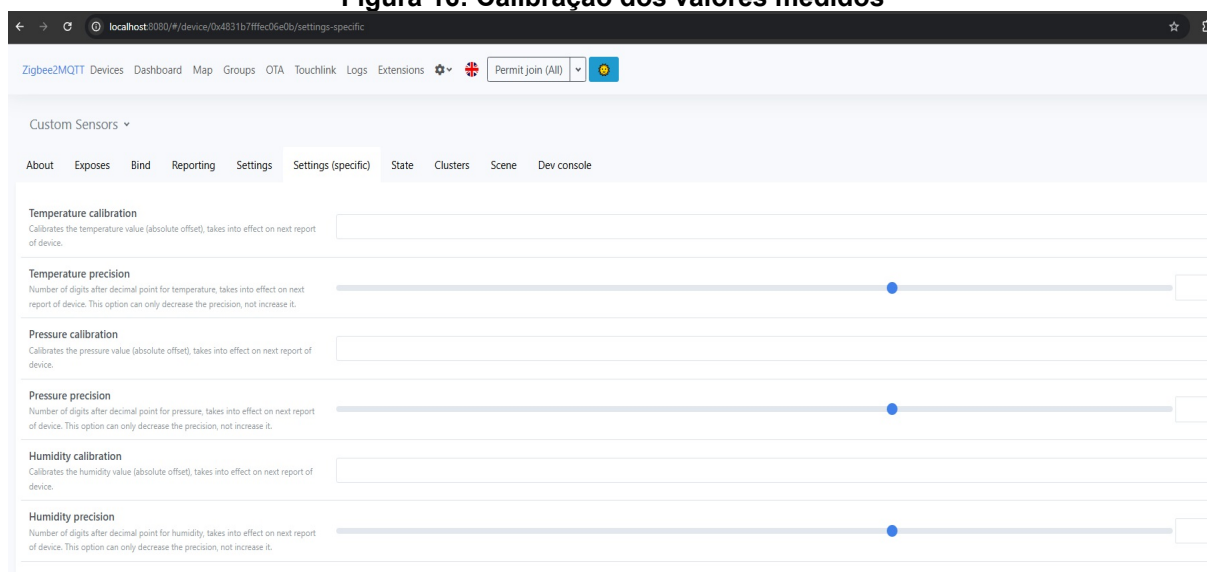
Figura 15: Valores visíveis na interface HTML



Fonte: Elaboração própria (2025).

Outra funcionalidade importante da interface HTML disponibilizada por causa do Zigbee2MQTT, é a possibilidade de interagir com os dispositivos Zigbee diretamente pela interface, podendo mandar a comunicação para atualizar os valores medidos de forma instantânea, ou até calibrar os valores medidos pelo sensor, como pode ser visto na Figura 16.

Figura 16: Calibração dos valores medidos

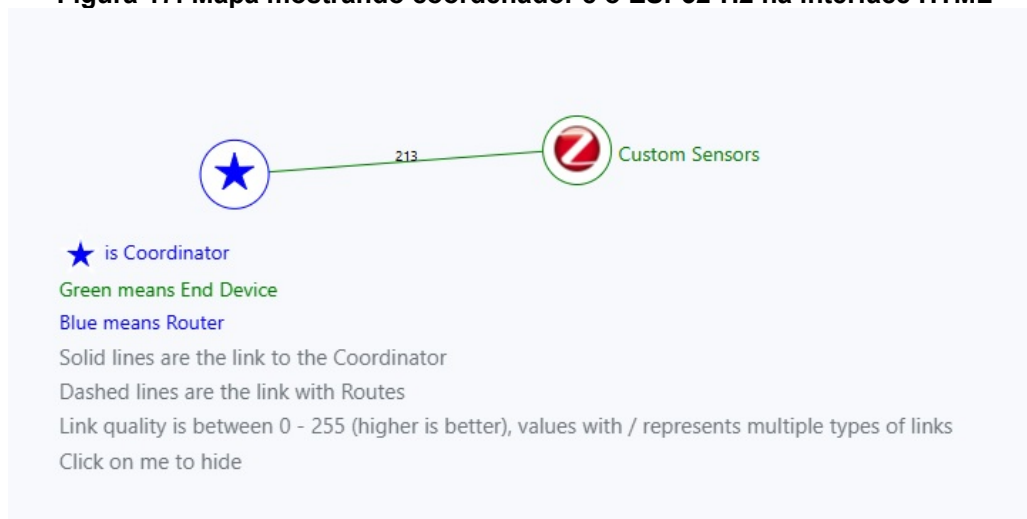


Fonte: Elaboração própria (2025).

Durante os testes, os valores de temperatura e umidade exibidos na interface HTML mostraram variações coerentes com as condições onde o sensor estava posicionado, evidenciando o funcionamento da leitura I2C no ESP32-H2, a formação estável da rede Zigbee e toda a cadeia de comunicação: sensor → rede Zigbee → coordenador → Zigbee2MQTT → interface.

Todas as possíveis funcionalidades da interface HTML não foram exploradas neste projeto considerando que o foco principal é validar o uso do ESP32-H2 em aplicações IoT utilizando a rede Zigbee, e isso já é possível de ser concluído apenas com os valores de medições sendo lidos corretamente. Porém, é notório que a interface disponibilizada pelo Zigbee2MQTT é altamente configurável, possuindo até mesmo um mapa mostrando a localização do coordenador em relação ao *end device*, como visto na Figura 17.

Figura 17: Mapa mostrando coordenador e o ESP32-H2 na interface HTML



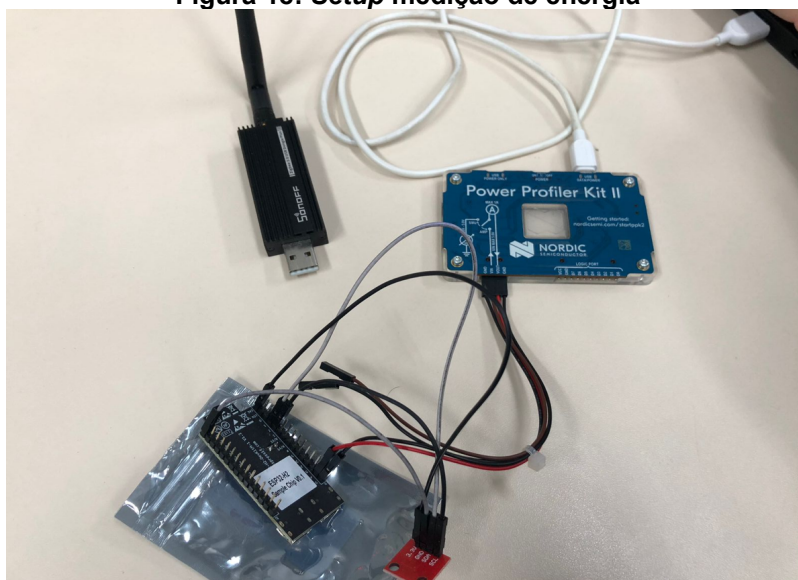
Fonte: Elaboração própria (2025).

Por fim, destaca-se que os resultados obtidos validam a proposta da primeira parte do trabalho, onde é possível utilizar o ESP32-H2 para comunicação local via Zigbee, e consequentemente seu uso em aplicações IoT. Informações mais aprofundadas do código criado para este trabalho podem ser encontrados no *link*: <https://github.com/Yuri-m-b/TCC>.

4.2 Consumo de energia

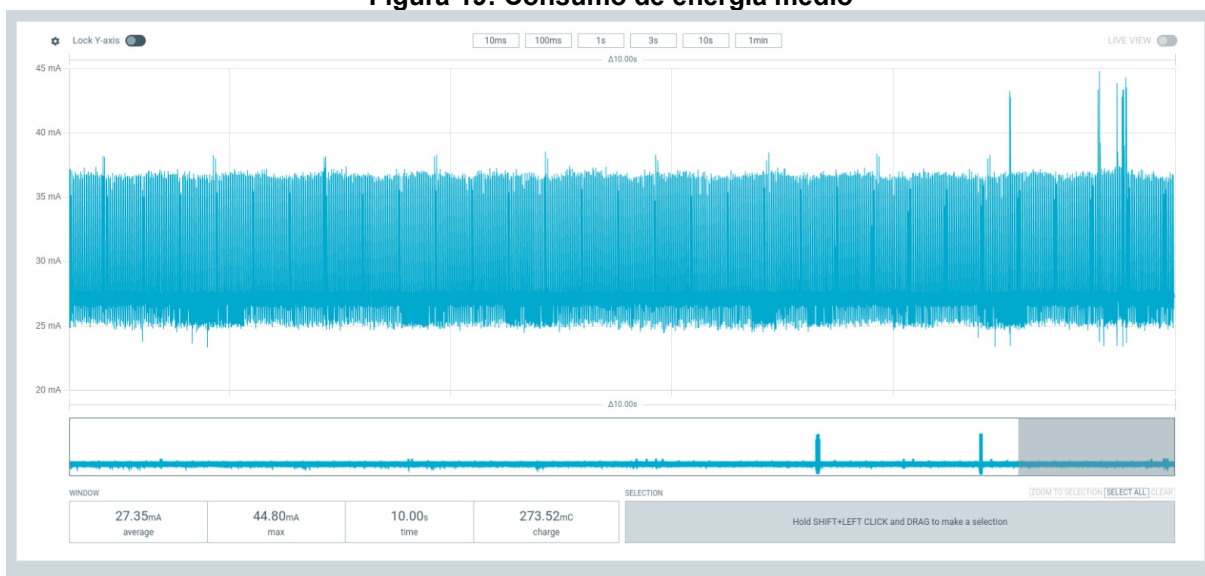
Além da validação funcional do protótipo, foi realizada a medição do consumo de energia da aplicação, utilizando o Power Profiler Kit da empresa NORDIC como ferramenta de monitoramento, visto na Figura 18. O objetivo dessa medição foi verificar o consumo médio do sistema está dentro de parâmetros aceitáveis para aplicações IoT, onde geralmente é necessário uma eficiência energética para garantir maior autonomia ao dispositivo, principalmente se a fonte de alimentação for limitada.

Figura 18: Setup medição de energia

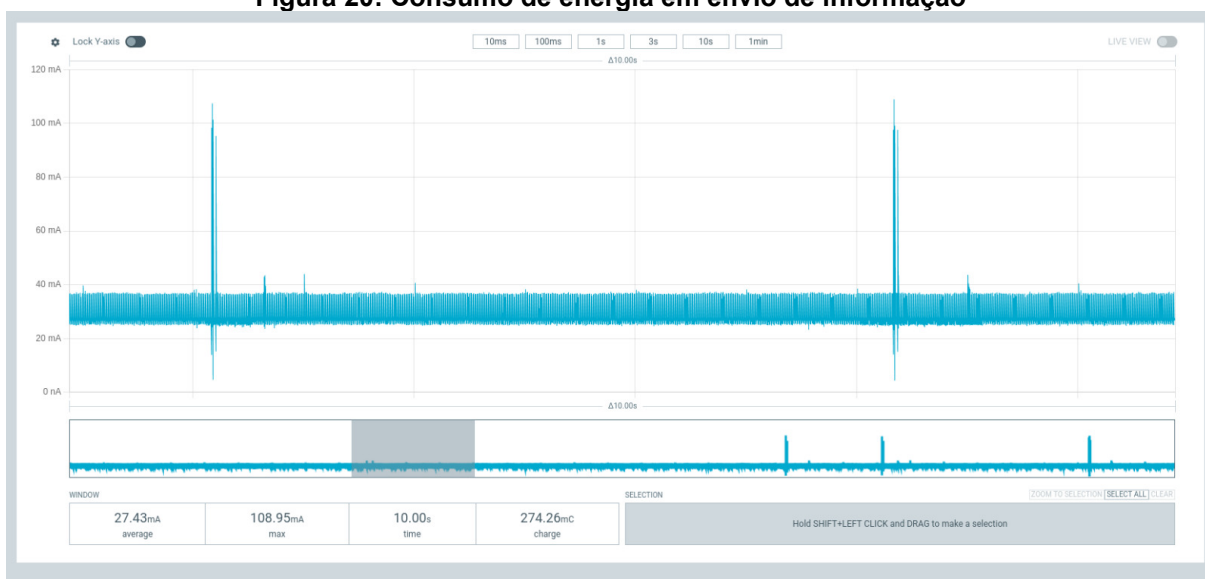


Fonte: Elaboração própria (2025).

Para a medição, o ESP32-H2 com o sensor HTU2X foram alimentados através do Power Profiler Kit, que registrou em seu *software* o consumo de corrente ao longo do tempo enquanto o sistema estava em funcionamento. Os resultados obtidos para a medição em modo *stand-by*, sem envios de medições, e ao enviar podem ser vistos na Figura 19 e Figura 20:

Figura 19: Consumo de energia médio

Fonte: Elaboração própria (2025).

Figura 20: Consumo de energia em envio de informação

Fonte: Elaboração própria (2025).

Como visto na Figura 19, o consumo médio de corrente durante o funcionamento “normal” do sistema é de 27,43 mA, enquanto que em momentos onde um envio de sinal está sendo realizado a corrente máxima vista foi de 108.95 mA. Já na Figura 20 podemos ver os picos de consumo de energia, quando um dado está sendo lido e enviado.

Considerando uma bateria que tenha uma capacidade de corrente de 2.000 mA, o protótipo criado consumiria a energia desta bateria em aproximadamente 72 horas no seu estado *stand-by*. Um dos possíveis motivos para este alto valor de consumo médio de energia, é o fato que o ESP32-H2 não foi configurado para entrar em seu modo baixo consumo enquanto espera novos comandos, consumindo então energia de forma não eficaz. O modo baixo consumo não seria integrado no projeto a tempo suficiente de serem feitos novos testes de medições e novas análises sobre o consumo energético, sendo assim uma ideia não implementada no protótipo final.

Além disso, é importante destacar que os valores de corrente medidos representam um cenário em que o dispositivo permanece constantemente ativo, sem aproveitamento dos modos de economia de energia disponíveis no ESP32-H2. Na prática, aplicações IoT raramente demandam processamento contínuo, mas sim ciclos curtos de atividade intercalados com longos períodos de inatividade. Dessa forma, a ausência de uma estratégia de gerenciamento energético adequada resulta em um desperdício significativo de carga da bateria.

É válido observar também que os picos de corrente, embora altos em comparação ao consumo médio, são transitórios e de curta duração, sendo menos críticos para a autonomia total do sistema. O impacto mais significativo é causado pelo consumo constante durante os períodos de espera, que deveria ser reduzido a níveis menores. Essa diferença mostra que o gargalo principal não está no envio de dados em si, mas na forma como o dispositivo permanece aguardando novas instruções.

Por fim, a análise sugere que a otimização do consumo energético do protótipo seja efetuada a partir de uma revisão da lógica do *firmware*, de modo a explorar os estados de baixa potência do microcontrolador. Com a configuração adequada do *deep sleep*, espera-se que a autonomia protótipo possa ser melhorada, tornando o sistema viável para aplicações IoT, onde a necessidade de uso de uma bateria e sua reposição frequente seja inviável tanto por custo quanto por praticidade.

5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo principal o estudo e a aplicação do microcontrolador ESP32-H2 em redes de Internet das Coisas (IoT) utilizando o protocolo Zigbee.

Inicialmente, foi realizado um levantamento teórico sobre o microcontrolador escolhido, os fundamentos do protocolo Zigbee, destacando suas características de comunicação em malha, baixo consumo de energia e adequação a aplicações embarcadas, explicando com detalhes o funcionamento dos seus *clusters*. Em seguida, foi analisada o protocolo de mensagem MQTT. E por fim feito a fundamentação teórica do *gateway* ZigbeeMQTT e sua relação neste projeto.

A implementação prática consistiu no desenvolvimento de uma aplicação embarcada de um sensor utilizando o ESP32-H2 como dispositivo final (ZED). Sensor de temperatura e umidade (HTU2X) foi integrado ao sistema por meio da interface I2C, permitindo a leitura periódica dos dados e sua atualização nos respectivos *clusters* Zigbee.

Apesar da correta funcionalidade da aplicação e da integração bem-sucedida com sensores e *clusters* Zigbee, observou-se que o consumo de energia final do sistema foi elevado, sobretudo para os padrões esperados em dispositivos finais de uma rede Zigbee, que normalmente exigem longa autonomia operando em modo de baixa potência. Esse resultado indica que ainda há oportunidades de otimização, tanto na frequência das medições quanto na utilização de modos de economia de energia do ESP32-H2, como sleep profundo (*deep sleep*), para adequar o sistema a cenários onde a alimentação elétrica é limitada.

Em síntese, este projeto permitiu validar o uso do ESP32-H2 em aplicações Zigbee, contribuindo para o conhecimento prático e teórico da tecnologia, e apontou caminhos relevantes para sua evolução em contextos reais de IoT com restrições energéticas.

5.1 Sugestões para trabalhos futuros

Sugere-se, para trabalhos futuros, ampliar o escopo deste projeto criando um sistema de dispositivos com maior quantidade de dispositivos utilizando a rede Zigbee, preferencialmente microcontroladores diferentes.

Além disso, é recomendada uma otimização aprofundada da aplicação, com foco especial na redução do consumo energético, visando aumentar a eficiência e a autonomia do sistema. Esse aprimoramento se torna essencial para aplicações em ambientes remotos ou de difícil acesso, onde o fornecimento de energia é limitado ou intermitente, como em sistemas alimentados por baterias ou fontes de energia renovável. Ao alcançar maior eficiência energética, o sistema poderá operar de forma contínua e confiável por períodos prolongados, o que amplia consideravelmente seu campo de aplicação prática.

REFERÊNCIAS

ALLIANCE, C. S. Zigbee Protocol. **Zigbee**. Disponível em: <https://csa-iot.org/all-solutions/zigbee/>. Acesso em: 11 jun. 2025.

ALLIANCE, Z. **Zigbee Cluster Library (ZCL)**. ZigBee Cluster Library Specification. Disponível em: <https://zigbeealliance.org/wp-content/uploads/2019/12/07-5123-06-zigbee-cluster-library-specification.pdf>. Acesso em: 11 jun. 2025.

CONNECTIVITY STANDARDS ALLIANCE. **Zigbee Protocol**. Disponível em: <https://csa-iot.org/all-solutions/zigbee/>. Acesso em: 11 jun. 2025.

BEVYWISE. **MQTT Manufacturing Solutions**. Disponível em: <https://www.bevywise.com/mqtt-usecases/manufacturing-solutions.html>. Acesso em: 25 jun. 2025.

CIRRUS LINK SOLUTION. **Case Study: Energy Company Maximizing MQTT for Control and Efficient Image Data Transfer**. Disponível em: <https://cirrus-link.com/case-study-energy/>. Acesso em: 25 jun. 2025.

CONNECTIVITY STANDARDS ALLIANCE. **Zigbee Protocol**. Disponível em: <https://csa-iot.org/all-solutions/zigbee/>. Acesso em: 12 jun. 2025.

GISLASON, Drew. **Zigbee Wireless Networking**. London: Newnes, 2008.

EMBARCADOS. **ESP32-H2-DevKitM-1: Primeira placa da Espressif sem WiFi**. Disponível em: <https://embarcados.com.br/esp32-h2-devkitm-1-primeira-placa-da-espressif-sem-wifi/>. Acesso em: 28 jun. 2025.

EMQ. SAIC Volkswagen build next-gen IoV with EMQX. Disponível em: <https://www.emqx.com/en/customers/saic-volkswagen>. Acesso em: 26 jun. 2025.

ESPRESSIF SYSTEMS. **ESP32-H2**. Disponível em:
<https://www.espressif.com/en/products/socs/esp32-h2>. Acesso em: 10 jun. 2025.

ESPRESSIF SYSTEMS. **ESP32-H2 Series Datasheet**. Disponível em:
https://www.espressif.com/sites/default/files/documentation/esp32-h2_datasheet_en.pdf. Acesso em: 10 jun. 2025.

ESPRESSIF SYSTEMS. **ESP32-H2-DevKitM-1**. Disponível em:
https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32h2/esp32-h2-devkitm-1/user_guide.html. Acesso em: 23 jul. 2025.

ESPRESSIF. **ESP-ZIGBEE-SDK**. Disponível em: https://github.com/espressif/esp-zigbee-sdk/blob/main/examples/esp_zigbee_HA_sample/HA_temperature_sensor/README.md. Acesso em: 02 jun. 2025.

FARAHANI, Shahin. **Zigbee Wireless Networks and Transceivers**. Amsterdam: Newnes, 2008.

HIVEMQ. **BMW Car-Sharing application relies on HiveMQ for reliable connectivity**. Disponível em: <https://www.hivemq.com/case-studies/bmw-mobility-services/>. Acesso em: 25 jun. 2025.

HIVEMQ. **Transportation & Logistic company cuts costs and improves asset tracking with HiveMQ**. Disponível em: <https://www.hivemq.com/case-studies/large-transportation/>. Acesso em: 25 jun. 2025.

IBM. **Telemetry use case: Home energy monitoring and control**. Disponível em: <https://www.ibm.com/docs/en/ibm-mq/9.4.x?topic=cases-telemetry-use-case-home-energy-monitoring-control>. Acesso em: 26 jun. 2025.

IOT101 INC. **Internet of Things 101**. iBooker, 2018.

IOT ECLIPSE FOUNDATION. **Deploying the Internet of Things on Germany's DB railway system**. Disponível em: <https://iot.eclipse.org/community/resources/case-studies/pdf/Eclipse%20IoT%20Success%20Story%20-%20DB.pdf>. Acesso em: 26 jun. 2025.

INTERNET OF THINGS WIKI. **Internet of things definition**. Disponível em: <https://internetofthingswiki.com/internet-of-things-definition/>. Acesso em: 04 jul. 2025.

MEDIUM. **Build affordable secure connected devices with ESP32-H2**. Disponível em: <https://medium.com/the-esp-journal/build-affordable-secure-connected-devices-with-esp32-h2-b8d542df8cb4>. Acesso em: 28 jun. 2025.

NORDIC SEMICONDUCTOR. **Power Profiler KIT II**. Disponível em: <https://www.nordicsemi.com/Products/Development-hardware/Power-Profiler-Kit-2>. Acesso em: 23 jul. 2025.

OASIS. **MQTT**. Disponível em: <https://mqtt.org/>. Acesso em: 12 jun. 2025.

OASIS. **MQTT V3.1 Protocol Specification**. Disponível em: <https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html>. Acesso em: 16 jun. 2025.

SONOFF. **ZHA vs. Zigbee2MQTT: The Best Zigbee Integration for Home Assistant?**. Disponível em: <https://sonoff.tech/blogs/news/zha-vs-zigbee2mqtt-the-best-zigbee-integration-for-home-assistant/>. Acesso em: 04 jul. 2025.

TEXAS INSTRUMENTS. **Thread vs Zigbee. 2018**. Disponível em: <http://www.ti.com/general/docs/lit/getliterature.tsp?baseLiteratureNumber=SWAY012&fileType=pdf>. Acesso em: 12 jun. 2025.

TE CONNECTIVITY. **HTU2X Humidity Sensors**. Disponível em: <https://www.ttiEurope.com/content/dam/tti-europe/manufacturers/te-connectivity/resources/c1de7b32-1394-42ed-9516-4647502d9aed.pdf>. Acesso em: 23 jul. 2025.

WANG, Chonggang; JIANG, Tao; ZHANG, Qian. **Zigbee Network Protocols and Applications**. Boca Raton: Auerbach Publications, 2016.

ZIGBEE ALLIANCE. **Cluster Library Specification**. 8. ed. 2019.

ZIGBEE ALLIANCE. **Zigbee Cluster Library (ZCL): Specification**. Disponível em: <https://zigbeealliance.org/wp-content/uploads/2019/12/07-5123-06-zigbee-cluster-library-specification.pdf>. Acesso em: 11 jun. 2025.

ZIGBEE2MQTT. **Zigbee2MQTT**. Disponível em: <https://www.zigbee2mqtt.io/>. Acesso em: 10 jun. 2025.

ZIGBEE2MQTT. **Zigbee2MQTT documentation**. Disponível em: <https://github.com/Koenkk/zigbee2mqtt>. Acesso em: 04 jul. 2025.