UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Lucas Müller Bordignon

ANÁLISE DE PROTOCOLOS DE COMUNICAÇÃO PARA A EFICIÊNCIA ENERGÉTICA EM SISTEMAS IOT

Santa Maria, RS
2023

Lucas Müller Bordignon

ANÁLISE DE PROTOCOLOS DE COMUNICAÇÃO PARA A EFICIÊNCIA ENERGÉTICA EM SISTEMAS IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**. Defesa realizada por videoconferência.

Orientador: Prof. Osmar Marchi dos Santos

Santa Maria, RS
2023

Lucas Müller Bordignon

ANÁLISE DE PROTOCOLOS DE COMUNICAÇÃO PARA A EFICIÊNCIA ENERGÉTICA EM SISTEMAS IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Bacharel em Engenharia de Computação**.

Aprovado em 20 de dezembro de 2023:

Osmar Marchi dos Santos, Dr. (UFSM)
(Presidente/Orientador)

Andrei Piccinini Legg, Dr. (UFSM)

Simone Regina Ceolin, Dra. (UFSM)

Santa Maria, RS
2023

AGRADECIMENTOS

Ao atingir este momento tão significativo em minha trajetória acadêmica, é com imensa alegria e emoção que dedico este trabalho de conclusão de curso. Foram meses de dedicação intensa, estudo aprofundado e aprendizado contínuo, e é com grande prazer que expresso meus agradecimentos a todas as pessoas que desempenharam papéis essenciais para a concretização deste sonho.

- À minha família, especialmente minha mãe, pelo amor, apoio e incentivo incondicionais.
- À minha namorada Rosiele, por ser meu porto seguro e fonte de motivação durante todos os dias.
- Ao meu orientador Osmar, pela orientação e conhecimento compartilhado.
- Por último, mas não menos importante, agradeço a todos os amigos pelos momentos de risadas, desafios e experiências compartilhadas.

RESUMO

ANÁLISE DE PROTOCOLOS DE COMUNICAÇÃO PARA A EFICIÊNCIA ENERGÉTICA EM SISTEMAS IOT

AUTOR: Lucas Müller Bordignon
Orientador: Osmar Marchi dos Santos

O número de dispositivos da Internet das Coisas tem aumentado significativamente nos últimos anos. Prevê-se que, até 2025, haverá uma conexão à internet de 22 bilhões de dispositivos IoT. Em paralelo a esse crescimento, nota-se que existe um constante avanço no quesito eficiência energética para todos esses dispositivos. Os protocolos da camada de aplicação tem um impacto importante no consumo de energia, logo é essencial estabelecer uma estrutura comum para permitir interações eficientes entre eles. O presente trabalho visa realizar um estudo comparativo entre os protocolos HTTP e MQTT, a fim de investigar qual deles possui uma eficiência energética maior ao utilizar um ESP8266, como dispositivo de teste. Os dados de consumo foram coletados por meio de um osciloscópio que registrava a tensão e a corrente do microcontrolador, realizando variações no tamanho de pacotes, no período de transmissão e na potência de envio. Após a análise dos resultados, foi possível concluir que o protocolo MQTT apresenta um consumo menor em comparação com o HTTP.

Palavras-chave: IoT. HTTP. MQTT. Consumo Energético.

ABSTRACT

ANALYSIS OF COMMUNICATION PROTOCOLS FOR ENERGY EFFICIENCY IN IOT SYSTEMS

AUTHOR: Lucas Müller Bordignon
ADVISOR: Osmar Marchi dos Santos

The number of Internet of Things devices has significantly increased in recent years. It is predicted that by 2025, there will be an internet connection for 22 billion IoT devices. Alongside this growth, there is a constant advancement in energy efficiency for all these devices. Application layer protocols have a significant impact on energy consumption, so it is essential to establish a common framework to enable efficient interactions among them. This study aims to conduct a comparative analysis between HTTP and MQTT protocols to investigate which one exhibits higher energy efficiency when using an ESP8266 as a test device. Consumption data was collected using an oscilloscope that recorded the voltage and current of the microcontroller, varying packet sizes, transmission delays, and transmission power. After analyzing the results, it was possible to conclude that the MQTT protocol has lower consumption compared to HTTP.

Keywords: IoT. HTTP. MQTT. Energy Consumption.

LISTA DE FIGURAS

Figura 1 – Ilustração do funcionamento de um sistema IoT. 14
Figura 2 – Representação da estrutura entre cliente e servidor. 15
Figura 3 – Diagrama representando a arquitetura de troca de mensagens no MQTT. . 17
Figura 4 – Fluxograma do algoritmo de testes para o ESP8266. 20
Figura 5 – Código dos testes no ESP8266 sem os protocolos implementados parte 1. 21
Figura 6 – Código dos testes no ESP8266 sem os protocolos implementados parte 2. 22
Figura 7 – Implementação do HTTP no código do cliente. 23
Figura 8 – Implementação do MQTT no código do cliente. 24
Figura 9 – Sistema de testes captando dados. 25
Figura 10 – Arquivo *csv* gerado pelo osciloscópio. 26
Figura 11 – Arquivo *csv* gerado após as manipulações com a biblioteca Pandas. 27
Figura 12 – Consumo do protocolo HTTP para 5dBm de transmissão. 28
Figura 13 – Consumo do protocolo HTTP para 8dBm de transmissão. 29
Figura 14 – Consumo do protocolo HTTP para 11dBm de transmissão. 30
Figura 15 – Análise do protocolo HTTP para pacotes de 300 em 5dBm. 31
Figura 16 – Análise do protocolo HTTP para pacotes de 300 bytes. 32
Figura 17 – Análise do protocolo HTTP para pacotes de 100 bytes. 33
Figura 18 – Análise do protocolo HTTP para pacotes de 500 bytes. 34
Figura 19 – Consumo do protocolo MQTT para 5dBm de transmissão. 35
Figura 20 – Consumo do protocolo MQTT para 8dBm de transmissão. 36
Figura 21 – Consumo do protocolo MQTT para 11dBm de transmissão. 37
Figura 22 – Análise do protocolo MQTT para pacotes de 300 bytes. 38
Figura 23 – Análise do protocolo MQTT para pacotes de 100 bytes. 39
Figura 24 – Análise do protocolo MQTT para pacotes de 500 bytes. 40

LISTA DE QUADROS

Quadro 1 – Comparação do consumo de energia para 5dBm 41
Quadro 2 – Comparação do consumo de energia para 8dBm 41
Quadro 3 – Comparação do consumo de energia para 11dBm 42

LISTA DE SIGLAS

AMQP	*Advanced Message Queuing Protocol*
CoAP	*Constrained Application Protocol*
CSV	*Comma-separated values*
dBm	decibel-miliwatt
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
IP	*Internet Protocol*
M2M	*Machine to Machine*
mAh	miliAmpère-hora
MQTT	*Message Queue Telemetry Transport*
NAT	*Network Address Translation*
SSID	*Service Set Identifier*
TCP	*Transmission Control Protocol*
USB	*Universal Serial Bus*
URL	*Uniform Resource Locator*
V	Volts
W	Watts

SUMÁRIO

1 INTRODUÇÃO 10
1.1 OBJETIVOS 11
1.1.1 Objetivo Geral 11
1.1.2 Objetivos Específicos 11
1.2 JUSTIFICATIVA 11
1.3 ESTRUTURA DO TRABALHO 12
2 FUNDAMENTAÇÃO TEÓRICA 13
2.1 INTERNET DAS COISAS 13
2.2 HTTP 14
2.3 MQTT 16
2.4 TRABALHOS RELACIONADOS 18
3 EQUIPAMENTOS E MÉTODOS 20
3.1 CENÁRIOS DE TESTES 20
3.2 IMPLEMENTAÇÃO DO CLIENTE 21
3.2.1 Código do protocolo HTTP 22
3.2.2 Código do protocolo MQTT 23
3.3 EQUIPAMENTOS 25
4 RESULTADOS 26
4.1 LIMPEZA E PRÉ-PROCESSAMENTO DOS DADOS 26
4.2 ANÁLISE DO CONSUMO DO PROTOCOLO HTTP 27
4.3 ANÁLISE DO CONSUMO DO PROTOCOLO MQTT 35
4.4 DISCUSSÃO DOS RESULTADOS 41
5 CONCLUSÃO 43
REFERÊNCIAS BIBLIOGRÁFICAS 44
APÊNDICE A – CÓDIGO DO SERVIDOR HTTP 46
APÊNDICE B – CÓDIGO DO CLIENTE HTTP NO ESP8266 47
APÊNDICE C – CÓDIGO DO SERVIDOR MQTT 50
APÊNDICE D – CÓDIGO DO CLIENTE MQTT NO ESP8266 51

1 INTRODUÇÃO

Nos últimos anos, o número de dispositivos da *Internet of Things* (IoT) conectados à Internet vem aumentando em cada projeção realizada. De acordo com Oracle Corporation (2019), as previsões apontam que o número de dispositivos IoT conectados será próximo de 22 bilhões em 2025. Em paralelo a esse crescimento, nota-se um constante avanço no quesito eficiência energética para todos esses dispositivos.

De acordo com Mouha (2021) a IoT, é algo de grande importância tecnológica, social e econômica nos dias atuais. Ela se refere a conexão de vários objetos do cotidiano à Internet, trazendo mudanças em nossas vidas e ambientes de trabalho. Isso se manifesta em produtos como casas inteligentes, dispositivos de saúde, veículos conectados e até aplicações agrícolas e industriais, que prometem melhorar a eficiência, a qualidade de vida e a economia global.

No entanto, a Internet das Coisas também apresenta desafios significativos como, por exemplo: questões de segurança, consumo de energia, entre outros obstáculos a qual eleva a complexidade tecnológica que precisa ser abordada para atingir todo o seu potencial (MOUHA, 2021). O consumo de energia é de suma importância nos dispositivos IoT, visto que um alto gasto energético pode inviabilizar a utilização de baterias ou painéis solares em aplicações particulares (OLIVEIRA, 2017).

Além disso, o custo relacionado ao consumo contínuo de energia em dispositivos IoT conectados à rede elétrica é algo que deve ser considerado. O uso de energia varia de acordo com a aplicação, sendo essencialmente baixo em cenários que dependem de baterias sem recarga viável ou com altos custos de recarga. Isso é particularmente relevante em ambientes com orçamentos restritos, como residências, e em aplicações de monitoramento ambiental que não têm acesso a uma fonte de energia permanente (OLIVEIRA, 2017).

Os protocolos de comunicação da camada de aplicação são de extrema relevância para aplicações IoT. Dado que a maioria dos dispositivos IoT possui características distintas, é essencial estabelecer uma estrutura comum para permitir interações eficientes entre eles (JAIKAR; IYER, 2018).

Neste contexto, o presente trabalho se propõe a realizar uma análise, efetuando medições e comparações do consumo de energia em um microcontrolador ESP8266 que utiliza tanto o *HyperText Transfer Protocol* (HTTP) quanto *Message Queuing Telemetry Transport* (MQTT) como protocolos de comunicação. Ademais, será explorado alguns fatores que afetam o uso de energia, como o tamanho dos pacotes, a potência do sinal de transmissão e o tempo de inatividade do envio de dados.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

Analisar o consumo energético em um microcontrolador ESP8266, utilizando os protocolos HTTP e MQTT da camada de aplicação em sistemas IoT.

1.1.2 Objetivos Específicos

Para alcançar o objetivo geral, foi necessário cumprir os objetivos específicos a seguir:

- Realizar uma revisão bibliográfica com foco na aplicação da Internet das Coisas e sobre os protocolos de comunicação HTTP e MQTT;
- Desenvolver algoritmos de cliente para o ESP8266 e servidor para ambos os protocolos;
- Montar o ambiente de testes e realizar a coleta de dados de potência;
- Analisar os dados de potência consumida no ESP8266 para ambos os protocolos;
- Demonstrar o protocolo que consome menos energia no microcontrolador.

1.2 JUSTIFICATIVA

A escolha de um protocolo de comunicação ineficiente pode trazer um impacto na eficiência energética em aplicações IoT, podendo resultar em um consumo excessivo de energia, o que pode limitar a vida útil das baterias dos dispositivos e aumentar os custos operacionais. Protocolos amplamente utilizados, como o HTTP e o MQTT, oferecem diferentes abordagens em relação ao consumo de energia, tornando essencial a análise comparativa.

A opção pelo ESP8266 para avaliar o consumo de energia destaca-se pela sua versatilidade em ambientes IoT. Ao comparar os protocolos HTTP e MQTT, comumente usados, é possível analisar de maneira eficiente o impacto dessas tecnologias no consumo energético do dispositivo.

Esse trabalho visa contribuir com dados sobre o consumo de energia associado aos protocolos HTTP e MQTT em um dispositivo ESP8266. Esses dados são valiosos para

pesquisadores e acadêmicos que buscam uma base para o desenvolvimento de sistemas IoT.

Também, os resultados deste estudo têm aplicação prática, pois podem ser diretamente utilizados por empresas e desenvolvedores para otimizar seus sistemas IoT, resultando em economia de energia, redução de custos e melhorias na sustentabilidade. Dessa forma, o estudo oferece uma contribuição significativa no âmbito de incentivar o uso mais eficiente da tecnologia.

1.3 ESTRUTURA DO TRABALHO

Para estruturar este trabalho de forma adequada, o estudo foi organizado em cinco capítulos, sendo eles:

- O Capítulo 1 teve como propósito a introdução e contextualização do tema, expondo algumas razões que motivaram a realização do estudo;
- O Capítulo 2 oferece uma fundamentação teórica concisa para esclarecer os conceitos explorados neste trabalho. Nesse contexto, são discutidos os conceitos relacionados à Internet das Coisas, que é a área de aplicação principal, além dos conceitos de protocolo HTTP e MQTT que servem como base teórica para a pesquisa. E uma análise dos estudos previamente realizados na área de pesquisa abordada. Para essa finalidade, foram escolhidos trabalhos que compartilham semelhanças, especialmente no que diz respeito aos protocolos estudados;
- O Capítulo 3 mostra os equipamentos e métodos utilizados para obter os dados de consumo de energia dos protocolos MQTT e HTTP em um ESP8266, o microcontrolador é utilizado como dispositivo de teste. As etapas incluíram configuração da ESP8266, monitoramento do consumo de energia e coleta de dados;
- O Capítulo 4 apresenta os resultados da pesquisa sobre o consumo de energia dos protocolos MQTT e HTTP, além de realizar uma análise comparativa entre eles. A comparação destaca as diferenças entre os protocolos e fornece percepções valiosas para a escolha apropriada em cenários específicos;
- O Capítulo 5 realiza uma revisão dos objetivos propostos e das contribuições do trabalho, resumindo os principais resultados obtidos. Foram abordadas as implicações práticas e as possíveis aplicações futuras da abordagem proposta.

2 FUNDAMENTAÇÃO TEÓRICA

A fundamentação teórica deste trabalho abrangeu conceitos relevantes relacionados à Internet das Coisas, aos protocolos de aplicação HTTP e MQTT e trabalhos relacionados nesse contexto. Na área da Internet das Coisas, foram abordados pré-requisitos fundamentais para a sua implementação e os elementos necessários para o seu funcionamento contínuo. No que se refere o protocolo HTTP, foram explorados conceitos de transferência de dados na internet, incluido a estrutura de mensagens e os seus métodos. Já no âmbito do MQTT, foram discutidos aspectos da troca de mensagens e os desafios enfrentados nesse contexto. Por último, foi realizado uma abordagem de estudos relacionados a esses assuntos.

Essa base teórica permite fundamentar as abordagens propostas neste trabalho, proporcionando um entendimento consistente dos princípios e métodos utilizados na construção e desenvolvimento do estudo.

2.1 INTERNET DAS COISAS

A Internet das Coisas representa uma profunda revolução no campo da Tecnologia da Informação como um todo. O conceito, que também pode ser abreviado como IoT em inglês, está ligado à busca por uma existência completamente interligada. Como indivíduos, nos tornamos parte intrínseca da própria rede de conexões, conforme referido por Mark Weiser em seu artigo Scientific American em 1991: “As tecnologias mais impactantes são aquelas que se mesclam. Elas se integram ao tecido da vida cotidiana até se tornarem indistinguíveis”(MADAKAM et al., 2015, p. 1). Essa rede de redes consiste em inúmeras conexões privadas, públicas, acadêmicas, corporativas e governamentais, que se estendem desde níveis locais até globais. Dessa forma, conexões são estabelecidas por meio de uma ampla variedade de tecnologias de rede, incluindo eletrônicas, sem fio e ópticas (MADAKAM et al., 2015).

O termo “Coisa” refere-se a qualquer dispositivo equipado com sensores capazes de coletar dados e transmiti-los pela rede sem a necessidade de intervenção manual. A tecnologia integrada nesse objeto facilita a interação com estados internos e o ambiente externo, desempenhando um papel crucial no processo de tomada de decisões (MOUHA, 2021).

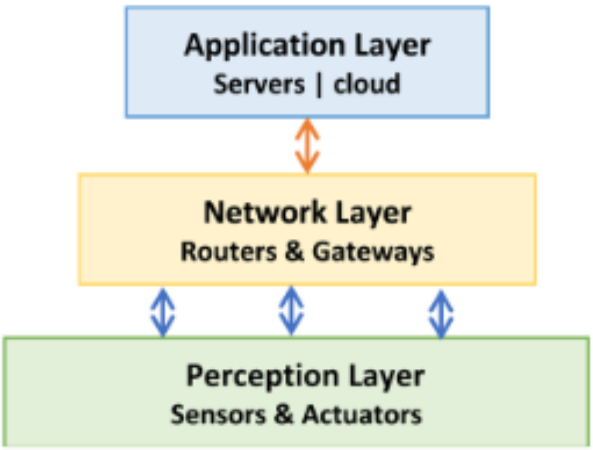
Ainda não há uma única definição para o conceito de IoT, assim, um entendimento amplo é compartilhado no meio acadêmico e inovativo. A internet, em sua origem, fundamentou-se em dados gerados por pessoas, enquanto a IoT é fundada em dados oriundos por “coisas”. Neste sentido, “Uma rede aberta e abrangente de objetos inteligen-

tes que possuem a habilidade de auto-organização, compartilhamento de dados, informações e recursos, além de reagir e responder a circunstâncias e alterações no ambiente" (MADAKAM et al., 2015, p. 2).

Para seu funcionamento contínuo, são fundamentais três elementos, que são:

- Hardware: Constituído por sensores, atuadores, câmeras, etc;
- Middleware: Composto por servidores para processamento dos dados adquiridos;
- Interface: Englobando ferramentas de visualização, que podem ser, por exemplo, aplicativos de smartphones. A Figura 1 exemplifica esse funcionamento.

Figura 1 – Ilustração do funcionamento de um sistema IoT.



Fonte: Mouha (2021).

2.2 HTTP

O *Hypertext Transfer Protocol* (HTTP), é um protocolo de comunicação de ampla utilização na camada de aplicação. Ele desempenha um papel fundamental em sistemas de informação, possibilitando a transferência de documentos em hipermídia e, assim, facilitando a distribuição e colaboração de conteúdo. A hipermídia reúne diversos tipos de mídia, incluindo imagens, vídeos, áudio e texto, em documentos interativos navegáveis, cuja leitura ocorre de maneira não linear. Um exemplo comum de documento em hipermídia é qualquer página da *web* (GOURLEY; TOTTY, 2002).

Desde sua criação em 1990, ele desempenha um papel central na comunicação da *web*. Sua primeira versão, HTTP/0.9, tinha como objetivo a simples transmissão de dados pela internet. Na evolução para a versão 1.0, o protocolo recebeu aprimoramentos

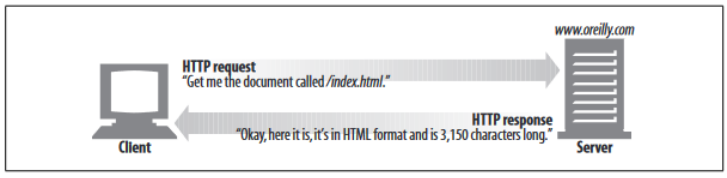
significativos, incluindo a introdução de cabeçalhos contendo informações sobre os dados transmitidos e um mecanismo de confirmação de sucesso na comunicação (GOURLEY; TOTTY, 2002). Atualmente, a versão mais recente do HTTP é a 3.0, que traz melhorias como redução de latência, compatibilidade com IPv6, NATs (*Network Address TranslationInternet Protocol*) e outras inovações.

De acordo com Gourley e Totty (2002), o HTTP possui as seguintes características:

- Simplicidade: As mensagens HTTP são projetadas para serem facilmente legíveis e compreensíveis por seres humanos, sem necessidade de equipamentos adicionais;
- Extensibilidade: Os cabeçalhos das mensagens HTTP são concebidos de forma que novas propriedades possam ser adicionadas de maneira intuitiva;
- Sem Estado (*Stateless*): O protocolo não mantém estado entre solicitações independentes, evitando compartilhamento de dados entre diferentes solicitações;
- Confiabilidade: O HTTP assegura a integridade dos dados durante a transmissão, utilizando a camada de transporte TCP/IP (*Transmission Control Protocol*). Essa integração com o TCP/IP torna o protocolo confiável e minimiza a possibilidade de perda de informações sem detecção.

O HTTP opera no modelo de requisição-resposta estabelecendo uma estrutura cliente-servidor, como pode-se observar na Figura 2, construida por Gourley e Totty (2002). Navegadores atuam como clientes ao iniciarem uma solicitação a uma URL (*Uniform Resource Locator*). Essa solicitação é direcionada a um servidor que processa a requisição e retorna uma resposta detalhando o processamento e, possivelmente, incluindo o conteúdo solicitado no corpo da mensagem.

Figura 2 – Representação da estrutura entre cliente e servidor.



Fonte: Gourley e Totty (2002).

A comunicação entre cliente e servidor HTTP segue uma estrutura precisa, composta por três partes fundamentais, que são: a linha de requisição, o cabeçalho e o corpo da mensagem (GOURLEY; TOTTY, 2002). A linha de requisição descreve o método de solicitação e a versão do protocolo, se quem realizou a solitação foi o cliente, e para o servidor tem-se a versão do protocolo junto com o código da resposta da mensagem (GOURLEY; TOTTY, 2002).

O cabeçalho, a partir da segunda linha, contém informações adicionais, como data/hora, endereço e tipo de servidor, tamanho e tipo de conteúdo da resposta, entre outros detalhes. Após o cabeçalho, uma linha em branco separa o corpo da mensagem, que é opcional e contém o conteúdo propriamente dito. Ao acessar uma URL em um navegador, é nesse corpo que o servidor envia o arquivo HTML para renderização (GOURLEY; TOTTY, 2002). No protocolo HTTP existem nove métodos que realizam as interações entre cliente e servidor, sendo eles:

- GET: Usado para fazer a busca de informações, podendo ou não conter conteúdo no corpo da resposta;
- HEAD: Similar ao GET, mas sem conteúdo no corpo da resposta;
- POST: Envia informações ao servidor, frequentemente causando alterações de estado;
- PUT: Também envia informações ao servidor, frequentemente para atualização;
- DELETE: Remove recursos específicos;
- TRACE: É empregado para rastrear o percurso que uma solicitação percorre até alcançar o servidor. São fornecidas ao cliente informações sobre os proxys e as máquinas pelos quais sua solicitação passa antes de chegar ao destino;
- PATCH: Utilizado para efetuar modificações parciais em um recurso específico;
- CONNECT: Estabelece um túnel de comunicação entre cliente e servidor, geralmente exigindo autenticação do cliente;
- OPTIONS: Utilizado para obter as opções de solicitação permitidas para um recurso específico no servidor.

2.3 MQTT

O MQTT (*Message Queuing Telemetry Transport*) é um protocolo M2M (*Machine to Machine*) desenvolvido pela IBM, concebido com base em princípios que o tornam especialmente adequado para aplicações IoT. Seus princípios fundamentais incluem otimizar a eficácia da largura de banda e recursos do dispositivo, assegurar confiabilidade e garantir a entrega bem-sucedida de mensagens. Esse protocolo é notavelmente leve e emprega uma estrutura de publicação e assinatura para comunicação (STANFORD-CLARK; TRUONG, 2013).

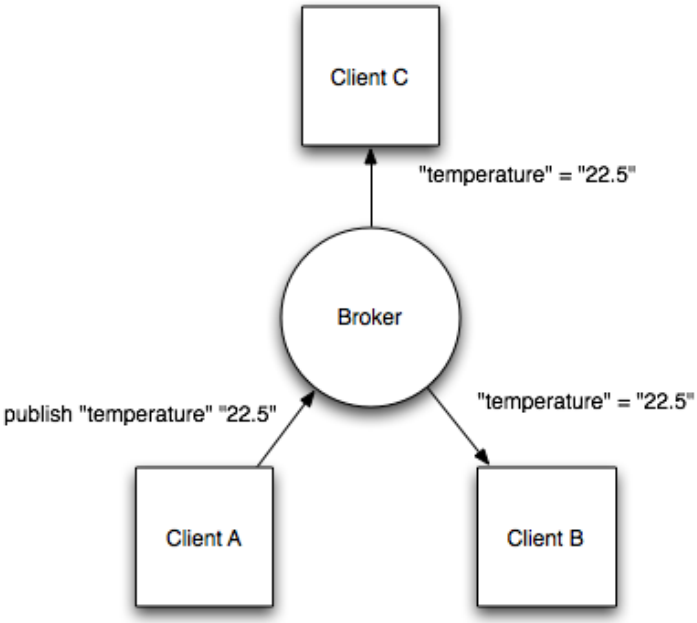
Uma característica destacada do MQTT é sua utilização do protocolo TCP/IP, assegurando a transmissão sem perda de dados. A arquitetura de publicação e assinatura viabiliza a distribuição de mensagens para múltiplos destinatários, isso é particularmente benéfico em ambientes com conectividade frágil ou quando limitar dados nas mensagens para otimização dos dados transferidos (STANFORD-CLARK; TRUONG, 2013).

De acordo com Tang et al. (2013), O MQTT oferece três níveis de serviço de QoS (*Quality of Service*). No nível 0, não há garantia de entrega; no nível 1, as mensagens são entregues pelo menos uma vez; e no nível 2, as mensagens são entregues exatamente uma vez, garantindo a entrega sem duplicatas, embora com maior sobrecarga. Esses níveis permitem adaptar a confiabilidade da entrega de mensagens conforme necessário.

A assincronicidade é suportada, separando transmissão e recepção no tempo e espaço, o que melhora a escalabilidade e a adaptação a redes não confiáveis. O MQTT distingue dois componentes: *Broker* e clientes. O *Broker* gere mensagens entre os clientes e direciona-as aos destinatários adequados. Já os clientes podem ser sensores IoT ou aplicativos de processamento de dados (MANANDHAR, 2017).

A comunicação de mensagens é realizada por meio de tópicos, no qual os clientes se conectam ao Broker, inscrevem-se em tópicos e publicam mensagens nestes. O *Broker* por sua vez gerencia essas mensagens reenviando aos inscritos nos tópicos. Assim, a Figura 3, retirada de Eclipse Foundation (2014) mostra esse processo.

Figura 3 – Diagrama representando a arquitetura de troca de mensagens no MQTT.



Fonte: Eclipse Foundation (2014).

2.4 TRABALHOS RELACIONADOS

Nesta seção, exploramos estudos relacionados que investigam protocolos na camada de aplicação e sua influência no consumo de energia em dispositivos IoT.

O artigo de Álvarez-Rosado et al. (2022) avaliou o consumo de energia usando um microcontrolador NodeMCU-32 e o protocolo MQTT. No estudo, foram propostas três fontes de alimentação diferentes: uma bateria recarregável NiMH (Níquel Metal Hidreto) de 450 mAh (milliampere-hora), que fornece uma voltagem de 9 V; um *power bank* USB de 5000 mAh, que gera uma tensão de 5 V; e, por último, uma bateria de Li-Po (lítio-polímero) de 500 mAh, que disponibiliza uma carga de 3,7 V. Para medir a corrente gerada pela NodeMCU-32, foi utilizado um osciloscópio USB (*Universal Serial Bus*).

O algoritmo de teste usou o serviço “no máximo um” e o Broker Mosquitto. Dessa forma, o algoritmo acordava, estabelecia a conexão Wi-Fi, definia a direção e se conectava ao servidor MQTT, em seguida publicava uma mensagem com o caractere “A” no tópico “test” e entrava em modo de hibernação, permanecendo nesse loop. Após vários testes, foi detectado que o menor consumo ocorreu com a bateria Li-Po de 500 mAh.

O trabalho de Stefanec e Kusek (2021) considerou os seguintes protocolos: AMQP, CoAP (*Constrained Application Protocol*), HTTP, HTTP2 e MQTT. Durante o estudo, foi proposta a ideia de enviar 1000 pacotes com tamanhos de 10 ou 1000 bytes, usando uma Raspberry Pi 3 B como cliente e um laptop com o sistema operacional Debian 9 como servidor. O cliente foi alimentado por um *power bank Anker PowerCore* 26800 para manter uma voltagem estável de 5V, enquanto no servidor, um Multímetro Uni-T UT71D foi usado como voltímetro para medir a tensão na fonte de alimentação e obter medidas automáticas de amostragem de tensão a cada 500ms.

Para iniciar uma medição, o comando “date” foi utilizado, e o mesmo comando foi usado no final da medição para determinar a duração da mesma. O cálculo do consumo de energia foi realizado multiplicando a diferença de energia entre o tempo final e o tempo inicial. Após os testes, ficou evidente que o protocolo que consome menos energia é o CoAP, seguido de MQTT, HTTP2, HTTP e AMQP.

Naik (2017) conduziu uma análise abrangente dos protocolos AMQP, CoAP, HTTP e MQTT no contexto dos sistemas IoT. Inicialmente, ele introduziu cada um desses protocolos e explicou seu funcionamento em sistemas IoT. Durante essa análise, diversos aspectos foram considerados, incluindo o tamanho da mensagem *versus* sobrecarga da mensagem, consumo de energia contra demanda de recursos e largura de banda em relação a latência.

Os resultados das três análises foram consistentes, com o protocolo CoAP apresentando vantagens significativas. O CoAP demonstrou ter um tamanho de mensagem menor, um menor consumo de energia e necessidade de recursos reduzida, além de uma menor largura de banda e latência, quando comparado com os outros protocolos. Na sequência,

foram classificados MQTT, AMQP e HTTP, respectivamente.

No estudo conduzido por Wukkadada et al. (2018), foi realizada uma comparação entre os protocolos HTTP e MQTT no contexto de sistemas IoT. Os resultados revelaram diferenças significativas nos tamanhos de cabeçalho e pacote, sendo o MQTT o protocolo com o menor tamanho em ambos aspectos.

Ao realizar medidas em uma conexão de internet 3G, foi observado que o MQTT apresentou uma taxa de transferência 90 vezes mais rápida do que o HTTP. A principal conclusão do estudo foi que, ao enviar pacotes com ambos os protocolos em intervalos de 60, 120, 240 e 480 segundos, ambos consumiram menos energia durante os intervalos mais longos em que permaneceram “acordados” para envio de pacotes.

Uma avaliação experimental do consumo de energia dos dispositivos móveis durante o compartilhamento de localização é realizada no artigo Vergara, Prihodko e Nadjm-Tehrani (2013). Os testes utilizaram os protocolos HTTP e MQTT, desse modo os resultados mostraram que o MQTT consome menos energia e gera menos dados do que o HTTP. No entanto, como o HTTP atualiza a localização dos usuários em intervalos de tempo específicos, consequentemente consome menos energia quando o número de clientes aumenta. Além disso, foram encontrados espaços vazios entre as atualizações de localização que podem ser aproveitados para reduzir ainda mais o uso de energia. Os autores enfatizam que o MQTT é amplamente utilizado em aplicações de telemetria e em dispositivos com recursos limitados, sugerindo que pode ser uma alternativa viável para o compartilhamento de localização em dispositivos móveis.

Diferente dos estudos realizados por Álvarez-Rosado et al. (2022) e Stefanec e Kusek (2021) que alimentaram seus dispositivos por um *power bank*, o presente trabalho alimentou o microcontrolador ESP8266 com uma fonte de bancada. E diferentes tamanhos de pacotes de dados, período envio de pacotes e potência de transmissão também foram analisados no presente estudo.

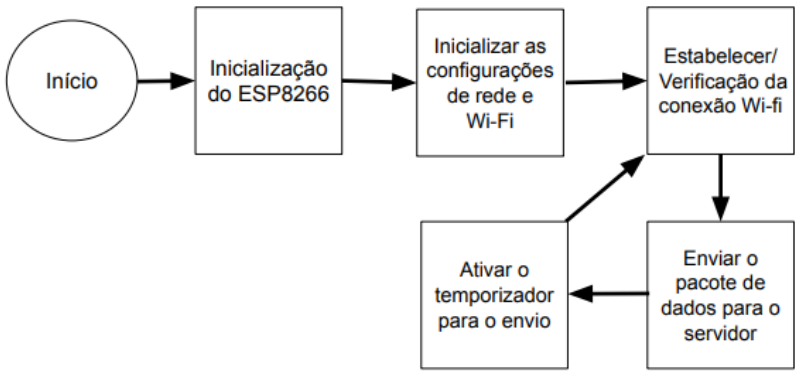
3 EQUIPAMENTOS E MÉTODOS

O propósito deste capítulo é fornercer uma descrição detalhada dos equipamentos, ferramentas e métodos utilizados na análise dos protocolos apresentados. Isso inclui a descrição dos procedimentos empregados na elaboração e condução do experimento que serve como base para este trabalho, bem como na captura, tratamento e análise dos dados coletados durante o experimento.

3.1 CENÁRIOS DE TESTES

Os testes foram feitos a partir dos cenários definidos, nos quais, eram feitas variações no período de envio de pacotes e na potência do sinal de transmissão, utilizando o ESP8266 como cliente da aplicação. O servidor da aplicação foi um notebook que estava conectado na mesma rede local. A Figura 4, apresentada em forma de fluxograma, representa os algoritmos de cliente da aplicação desenvolvidos para a condução dos testes no ESP8266. Cabe ressaltar que o fluxograma é aplicável para ambos os protocolos.

Figura 4 – Fluxograma do algoritmo de testes para o ESP8266.



Fonte: Autor.

No decorrer dos testes, foram feitas mudanças nas configurações do experimento, sendo elas:

- Variações na potência de transmissão, que abrangeram os valores de 5, 8 e 11 dBm;
- Atraso na transmissão de pacotes em 1, 3 e 5 segundos, com a relação entre o tempo de transmissão e o tamanho dos pacotes sendo de 1 segundo equivalendo a 100 bytes, 3 segundos a 300 bytes e 5 segundos a 500 bytes.

3.2 IMPLEMENTAÇÃO DO CLIENTE

Para o desenvolvimento dos *scripts* de cliente da aplicação foi utilizada a Arduino IDE[1] (*Integrated Development Environment*) na versão 2.2.1. A conexão com a rede Wi-Fi foi implementada com a biblioteca ESP8266WiFi[2]. Essa biblioteca tornou possível definir os parâmetros necessários para autenticação e conectividade à rede, incluindo SSID (*Service Set Identifier*) e senha da rede Wi-Fi. Além do uso para conexão na rede, ela tem outras funcionalidades, sendo a de definir a potência de transmissão do ESP8266. As Figuras 5 e 6 demonstram o código desenvolvido para o microcontrolador, sem os protocolos MQTT e HTTP implementados.

Figura 5 – Código dos testes no ESP8266 sem os protocolos implementados parte 1.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <map>

const char* SSID_REDE = "SEU SSID AQUI";
const char* SENHA_REDE = "SUA SENHA AQUI";
const char* serverName = "http://SEU IP AQUI:5000/post";
WiFiClient client;
unsigned long startTime;
unsigned long currentTime;
byte dataToSend[100];
unsigned int dataSize = 100;
unsigned long delayValue = 5000; // Delay do tempo de transmissão

std::map<unsigned long, unsigned int> delayToSize = {
    {3000, 300}, // 3 segundos correspondem a 300 bytes
    {5000, 500}  // 5 segundos correspondem a 500 bytes

};
void init_wifi(void);
void conecta_wifi(void);
void verifica_conexao_wifi(void);

void init_wifi(void)
{
    Serial.println("------WI-FI -----");
    Serial.println("Conectando-se a rede: ");
    Serial.println(SSID_REDE);
    Serial.println("\nAguarde...");

    conecta_wifi();
}

void conecta_wifi(void)
{
    if (WiFi.status() == WL_CONNECTED)
    {
        return;
    }

    WiFi.begin(SSID_REDE, SENHA_REDE);

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(100);
    }

    Serial.println("Conectado com sucesso a rede wi-fi \n");
    Serial.println(SSID_REDE);
}
```

Fonte: Autor

[1] <https://www.arduino.cc/en/software>
[2] <https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html>

Figura 6 – Código dos testes no ESP8266 sem os protocolos implementados parte 2.

```
52 void verifica_conexao_wifi(void)
53 {
54     conecta_wifi();
55 }
56
57 void setup(){
58   Serial.begin(9600);
59   init_wifi(); // Inicia a conexão WI-FI
60   WiFi.setOutputPower(5);
61     if (delayToSize.find(delayValue) != delayToSize.end()) {
62     dataSize = delayToSize[delayValue];
63     } else {
64     // Caso o valor de atraso não esteja mapeado, use um tamanho padrão
65         dataSize = 100;
66     }
67 }
68 void loop(){
69   startTime = millis();
70   verifica_conexao_wifi();
71   currentTime = millis();
72   delay(delayValue - (currentTime - startTime));
73 }
```

Fonte: Autor

O código tem as seguintes funcionalidades:

- Constantes como SSIDREDE (nome da rede Wi-Fi), SENHAREDE (senha da rede Wi-Fi) e delayValue (período entre transmissões) são definidas;
- Um *map*, uma funcionalidade própria da linguagem C++, chamado delayToSize que mapeia valores do período de transmissão para tamanhos de dados correspondentes;
- Funções de inicialização do Wi-Fi são fornecidas: init-wifi(), conecta-wifi(), e verifica-conexaowifi(). Elas cuidam da conexão e da verificação de conectividade Wi-Fi;
- No método setup(), a comunicação serial é inicializada, e a potência de transmissão Wi-Fi é definida. O tamanho dos dados é ajustado com base no período de transmissão;
- O método loop() é o código principal do programa. Ele verifica e estabelece a conexão Wi-Fi, e controla o período entre as transmissões.

3.2.1 Código do protocolo HTTP

Para o uso do protocolo HTTP a biblioteca ESP8266HTTPClient[3] foi utilizada, ela é uma biblioteca de código aberto que fornece uma interface entre o protocolo HTTP e o ESP8266. Ela pode ser usada para enviar e receber dados de um servidor HTTP. Um *script* de um servidor HTTP foi desenvolvimento para validação dos dados enviados pelo microcontrolador, utilizando a linguagem Python e a biblioteca Flask[4]. A Figura 7 mostra

[3] <https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266HTTPClient/src/ESP8266HTTPClient.h>
[4] <https://flask-ptbr.readthedocs.io/en/latest/>

o que foi acrescentado no código das Figuras 5 e 6, para viabilizar a conexão entre o ESP8266 e o servidor da aplicação.

Figura 7 – Implementação do HTTP no código do cliente.

```
1  #include <ESP8266WiFi.h>
2  #include <ESP8266HTTPClient.h>
3  .
4  .
5  .
6  .
7  WiFiClient client;
8  HTTPClient http;
9
10 void loop(){
11   startTime = millis();
12   verifica_conexao_wifi();
13
14   http.begin(client, serverName);
15   http.addHeader("Content-Type", "application/json");
16   int httpResponseCode = http.POST(dataToSend, dataSize);
17   http.end();
18   currentTime = millis();
19
20   delay(delayValue - (currentTime - startTime));
21 }
```

Fonte: Autor

Na linha 2 da Figura 6, é incluída a biblioteca ESP8266HTTPClient. Já na linha 8, é criada uma variável do tipo HTTPClient para poder utilizar todas as funções que a biblioteca disponibiliza. No intervalo da linha 14 até a linha 17, é iniciada a conexão HTTP, adicionado o cabeçalho do pacote, realizado o envio do vetor de bytes e, por fim, a conexão é encerrada.

3.2.2 Código do protocolo MQTT

Já para o protocolo MQTT foi adotada a biblioteca PubSubClient[5], é uma biblioteca de código aberto que possibilita a conexão entre o microcontrolador e o servidor MQTT. O servidor foi desenvolvido em Python, utilizando a biblioteca Paho[6] MQTT, e o Broker aplicado para os testes foi da Foundation (2023), pois ele é de uso gratuito. A Figura 8 mostra o que foi adicionado no código das Figuras 5 e 6, para realizar a conexão entre o ESP8266 e o servidor da aplicação.

[5] <https://www.arduino.cc/reference/en/libraries/pubsubclient/>
[6] <https://pypi.org/project/paho-mqtt/>

Figura 8 – Implementação do MQTT no código do cliente.

```
1  #include <ESP8266WiFi.h>
2  #include <PubSubClient.h>
3  #define MQTT_MAX_PACKET_SIZE 500
4  #define MQTT_MAX_TRANSFER_SIZE 500 // Defina o tamanho máximo desejado
5  const char* mqttServer = "mqtt.eclipseprojects.io";
6  const int mqttPort = 1883;
7
8  WiFiClient client;
9  PubSubClient client(client);
10
11 void setup() {
12   // Inicialização da conexão serial
13   Serial.begin(9600);
14   startTime = millis();// Recebe o tempo inicial em que o ESP8266 foi ligado
15   // Configuração do broker MQTT
16   client.setBufferSize(MQTT_MAX_PACKET_SIZE);
17   client.setServer(mqttServer, mqttPort);
18   init_wifi(); // Inicia a conexão WI-FI
19   WiFi.setOutputPower(17);
20
21 }
22
23 void loop() {
24   startTime = millis();
25   verifica_conexao_wifi();
26   if (!client.connected()) {
27     reconnect();
28   }
29
30
31   client.publish("topico/lucas/tcc",dataToSend, dataSize);
32
33   delay(delayValue - (currentTime - startTime)); // Intervalo de envio das mensagens
34 }
35
36 void reconnect() {
37   // Loop até que esteja conectado ao broker MQTT
38   while (!client.connected()) {
39     Serial.println("Conectando ao broker MQTT...");
40     if (client.connect("ESP8266Client")) {
41       Serial.println("Conectado ao broker MQTT");
42     } else {
43       Serial.print("Falha na conexão ao broker, código de erro: ");
44       Serial.print(client.state());
45       Serial.println(" Tentando novamente...");
46       delay(100);
47     }
48   }
49 }
```

Fonte: Autor

No intervalo das linhas 2 a 6, é inserida a biblioteca PubSubCliente, definindo o tamanho máximo dos pacotes e a capacidade de transferência. Além disso, declaram-se as variáveis que servem como o broker e a porta da conexão. Na linha 9, o cliente é criado para permitir a utilização das funções da biblioteca. Entre as linhas 16 e 17, ocorre a definição do tamanho do buffer, bem como a configuração do broker e da porta de conexão.

No intervalo entre as linhas 26 e 31, uma condição verifica se o cliente está conectado ao broker. Caso ele não esteja, a rotina reconnect, definida na linha 36, é chamada. Por outro lado, se o cliente estiver conectado, ele publica a mensagem no tópico “topico/lucas/tcc”. A função reconnect se faz necessária, pois, caso o cliente não esteja conectado, ele não consegue enviar o pacote de dados, o que pode resultar no travamento do programa dentro do método loop.

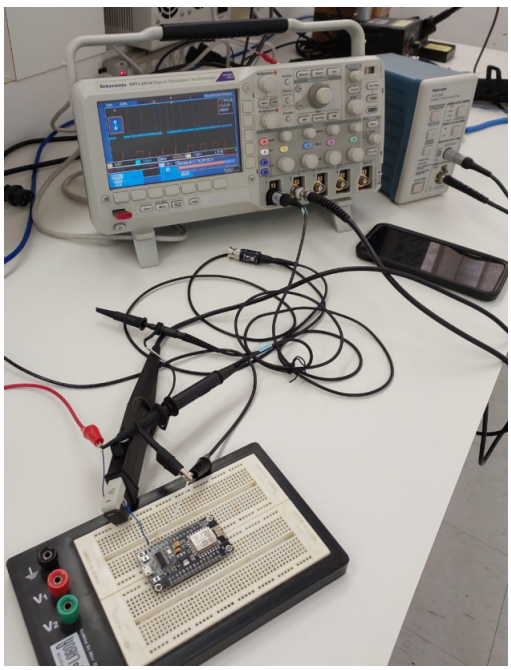
3.3 EQUIPAMENTOS

Para a alimentação do ESP8266, foi utilizado uma fonte de bancada que forneceu uma tensão estável de 5V para alimentar o dispositivo. A tensão constante era necessária para garantir que as medições de corrente e voltagem fossem confiáveis e repetíveis. Com o propósito de medir a corrente consumida pelo ESP8266, foi utilizada a sonda de corrente Tektronix TCPA300. Essa sonda é projetada para medições de corrente de alta precisão e permite a captura de formas de onda de corrente com alta fidelidade. Para os testes, a sonda foi conectada em série com a fonte de alimentação do microcontrolador para monitorar o consumo de corrente em tempo real.

O osciloscópio Tektronix DP02014 foi usado para a aquisição e análise dos dados. Este osciloscópio de 4 canais oferece alta largura de banda e taxa de amostragem, permitindo a captura precisa dos valores de tensão e corrente do microcontrolador. Os dados adquiridos pelo osciloscópio foram gerados a partir da multiplicação do canal 1 pelo canal 2, onde o primeiro registrava a tensão e o segundo a corrente do dispositivo, gerando dados de potência. Os valores de potência foram salvos em um arquivo *csv* para uma análise posterior.

O notebook utilizado como servidor de aplicação para receber os dados enviados foi um Acer Aspire 5, com 8GB de memória RAM e processador Intel Core i5, rodando o sistema operacional Ubuntu na versão 22.04. A Figura 9 mostra o sistema de testes em funcionamento, coletando dados de tensão e corrente do ESP8266.

Figura 9 – Sistema de testes captando dados.



Fonte: Autor.

4 RESULTADOS

Este capítulo apresenta os resultados obtidos com os experimentos no trabalho. A partir dos dados coletados é realizado uma análise gráfica dos valores obtidos e são efetuadas comparações. As análises gráficas foram realizadas com a linguagem Python e as bibliotecas Matplotlib[1], Seaborn[2].

4.1 LIMPEZA E PRÉ-PROCESSAMENTO DOS DADOS

Primeiramente foi realizado uma verificação dos dados coletados pelo osciloscópio, para validação dos testes. A Figura 10 demonstra as primeiras linhas do arquivo *csv*.

Figura 10 – Arquivo *csv* gerado pelo osciloscópio.

1	Model	DPO2014
2	Firmware Version	1.35
3		
4	Point Format	Y
5	Horizontal Units	S
6	Horizontal Scale	2
7	Sample Interval	1.6e-05
8	Filter Frequency	100000000
9	Record Length	1.25e+06
10	Gating	0.0% to 100.0%
11	Probe Attenuation	1
12	Vertical Units	Watts
13	Vertical Offset	0
14	Vertical Scale	1
15	Label	POWER
16	TIME	MATH
17	-2.000000e+00	0.127109

Fonte: Autor.

As primeiras 16 linhas são de configurações definidas para os testes e informações do modelo do osciloscópio, para uma melhor confiabilidade dos dados, foi definido um intervalo de amostra de 16μ segundos e 1.25M pontos de gravação, a partir da linha 17

[1] <https://matplotlib.org/>
[2] <https://seaborn.pydata.org/>

são mostrados os valores obtidos. Para todos os arquivos *csv* gerados, as 16 primeiras linhas são iguais, logo para realizar a análise dos dados foi utilizado a linguagem Python e a biblioteca Pandas[3], a Figura 11 mostra o novo *csv* gerado após a remoção dessas linhas iniciais, e definindo o nome das colunas tempo e potência.

Figura 11 – Arquivo *csv* gerado após as manipulações com a biblioteca Pandas.

1	Tempo	Potencia
2	-2.0	0.127109
3	-1.999984	0.169531
4	-1.999968	0.163125
5	-1.999952	0.169531
6	-1.999936	0.169531
7	-1.99992	0.169531
8	-1.999904	0.169531
9	-1.999888	0.169531
10	-1.999872	0.127109
11	-1.999856	0.169531
12	-1.99984	0.163125

Fonte: Autor.

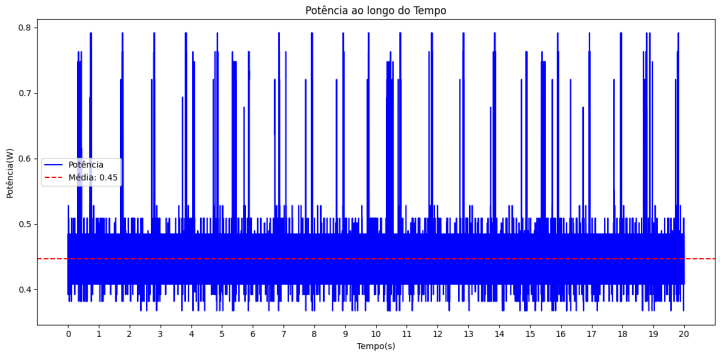
4.2 ANÁLISE DO CONSUMO DO PROTOCOLO HTTP

Nesta seção, é realizada a análise do consumo energético no ESP8266 utilizando o protocolo HTTP. Todos os gráficos gerados mostram a potência consumida em um período de 20 segundos e a potência média no mesmo intervalo de tempo.

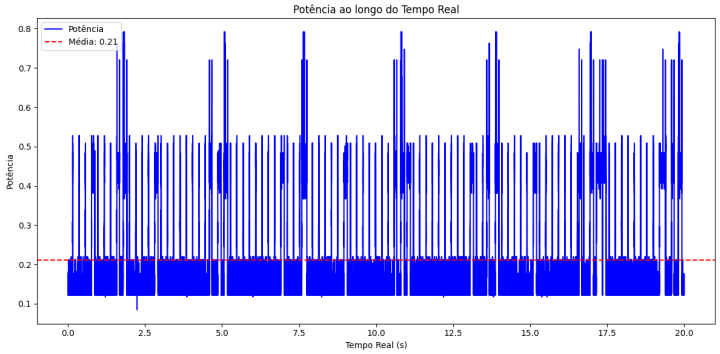
A Figura 12 apresenta os dados do consumo energético para pacotes de 100, 300 e 500 bytes, considerando uma potência de transmissão de 5dBm. Observou-se que o envio de um pacote de 100 bytes resultou em uma potência média 0,45W, enquanto os conjuntos de dados de 300 e 500 bytes apresentaram uma potência média de 0,21W e 0,20W, respectivamente.

[3] <https://pandas.pydata.org/>

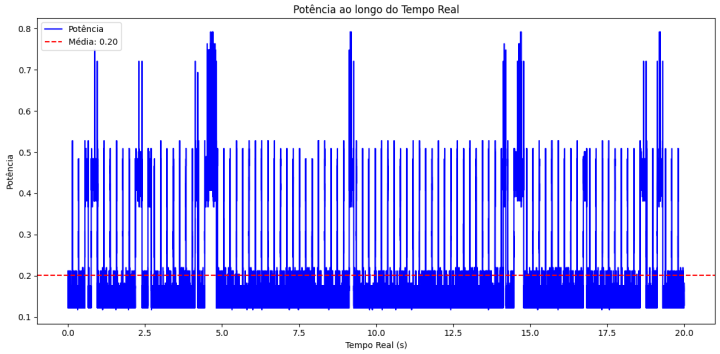
Figura 12 – Consumo do protocolo HTTP para 5dBm de transmissão.



(a) 100 bytes



(b) 300 bytes

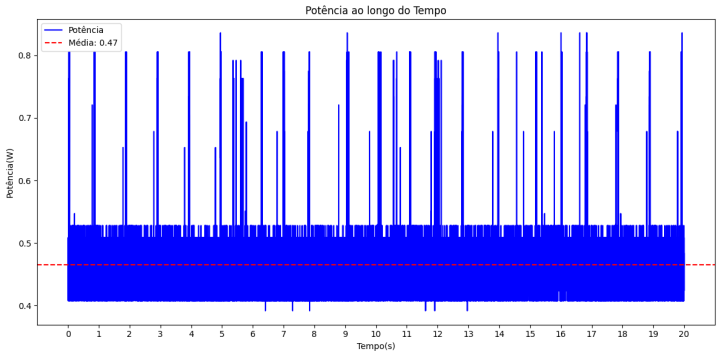


(c) 500 bytes

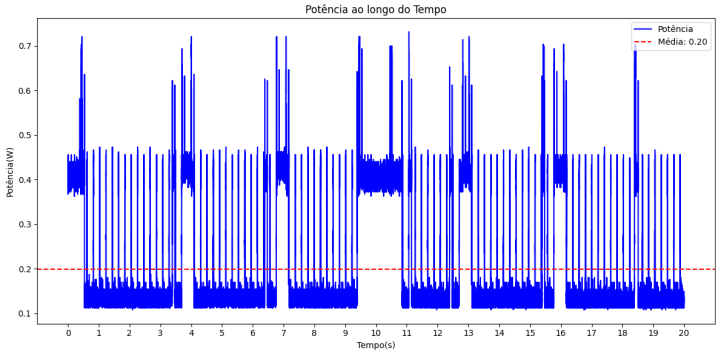
Fonte: Autor.

Os dados relativos ao consumo energético para pacotes de 100, 300 e 500 bytes, considerando uma potência de transmissão de 8dBm, são apresentados na Figura 13. Verificou-se que o envio de um pacote de 100 bytes resultou em uma potência de 0,45W, enquanto os conjuntos de dados de 300 e 500 bytes registraram potência média de 0,20W e 0,21W, respectivamente.

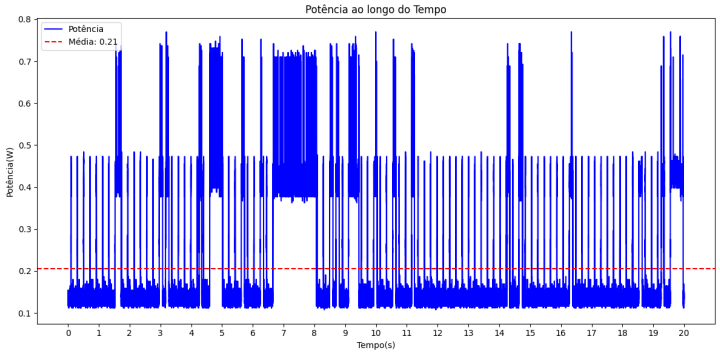
Figura 13 – Consumo do protocolo HTTP para 8dBm de transmissão.



(a) 100 bytes



(b) 300 bytes

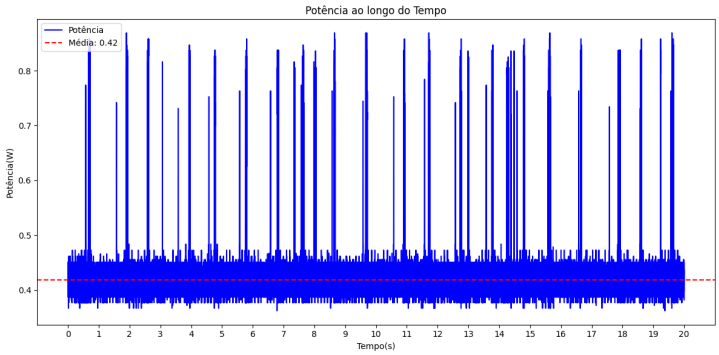


(c) 500 bytes

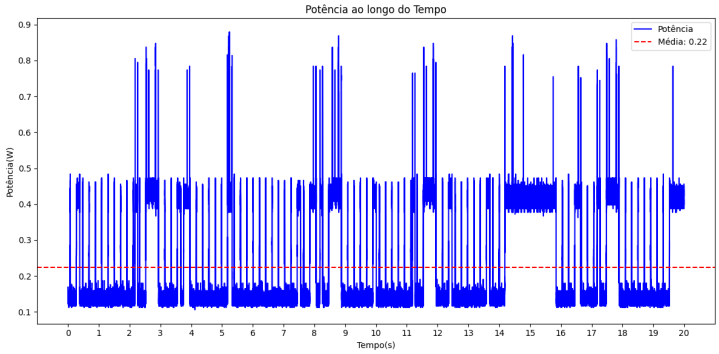
Fonte: Autor.

Os dados referentes ao consumo de energia para pacotes de 100, 300 e 500 bytes, considerando uma potência de transmissão de 11dBm, estão ilustrados na Figura 14. Observou-se que o envio de um pacote de 100 bytes resultou em uma potência média de 0,42W, enquanto os conjuntos de dados de 300 e 500 bytes apresentaram potências médias 0,22W e 0,23W, respectivamente.

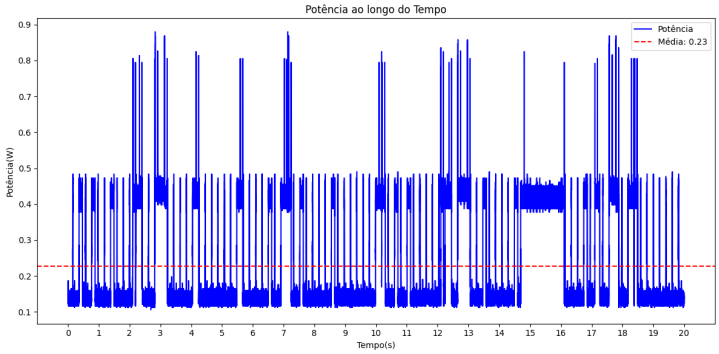
Figura 14 – Consumo do protocolo HTTP para 11dBm de transmissão.



(a) 100 bytes



(b) 300 bytes

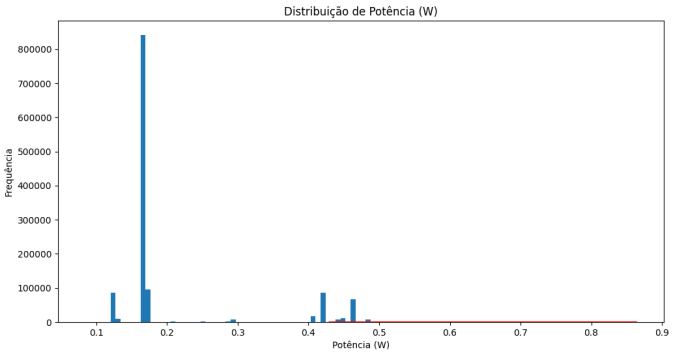


(c) 500 bytes

Fonte: Autor.

Para obter uma compreensão mais detalhada do consumo no ESP8266, variando a potência do sinal de transmissão, foi utilizado um gráfico histograma, como mostra a Figura 15. Essa abordagem foi escolhida devido à limitação dos gráficos da potência média no tempo, que não foram suficientes para verificar o impacto dessa troca de valores. Além disso, o histograma inclui uma linha vermelha representando a densidade dos valores, proporcionando uma visualização mais precisa e facilitando a identificação de padrões e variações no consumo de energia em relação às diferentes potências de transmissão.

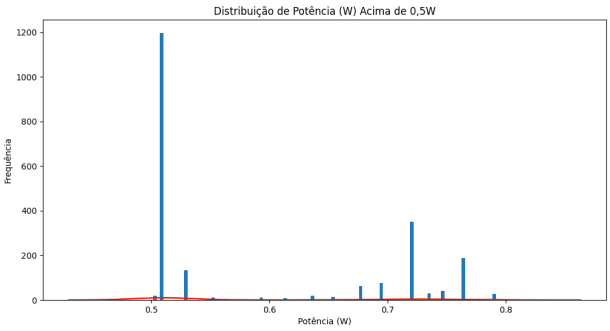
Figura 15 – Análise do protocolo HTTP para pacotes de 300 em 5dBm.



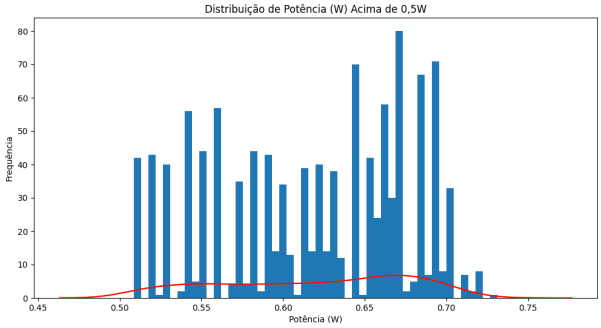
Fonte: Autor.

Porém devido a alta frequência de valores em uma determinada faixa, não foi possível detectar o impacto da variação da potência de transmissão no consumo do microcontrolador, logo para uma melhor resolução foram selecionados valores acima de 0,5 W. Essa abordagem permitiu identificar tendências no consumo de energia do microcontrolador, pois a maior clareza na visualização e interpretação dos dados só foi possível com a utilização de valores acima de 0,5 W. A Figura 16 mostra os testes de pacotes de 300 bytes com a variação da potência de transmissão.

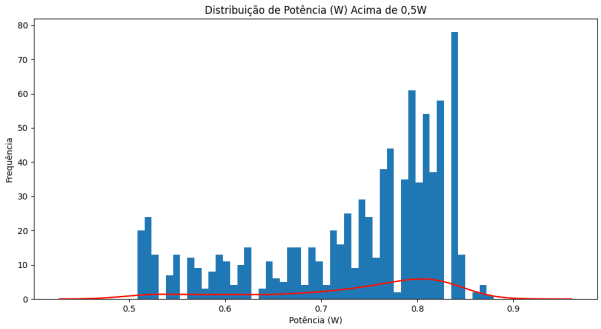
Figura 16 – Análise do protocolo HTTP para pacotes de 300 bytes.



(a) 5dBm



(b) 8dBm

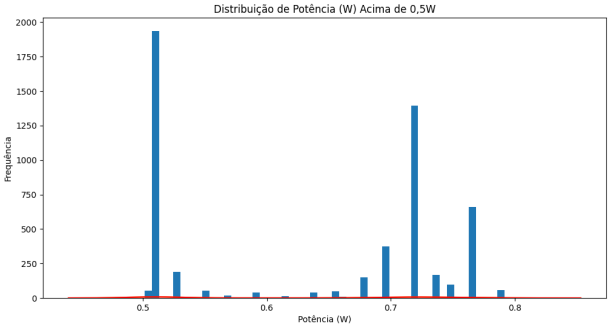


(c) 11dBm

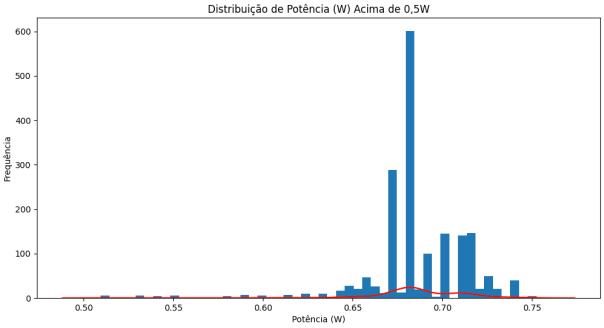
Fonte: Autor.

Observa-se que aumentando a potência do sinal de transmissão, valores mais altos de potência aparecem com maior frequência. Analisando os gráficos, verificou-se o mesmo comportamento para os pacotes de 100 e 500 bytes, as Figuras 17 e 18 demonstram isso.

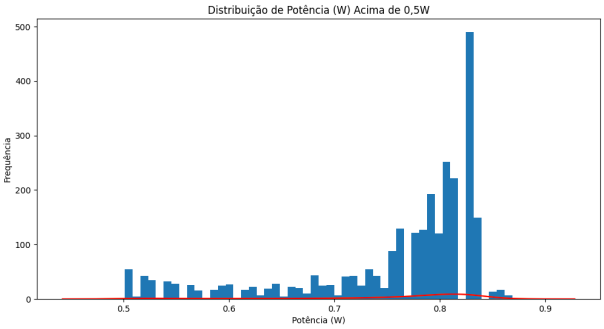
Figura 17 – Análise do protocolo HTTP para pacotes de 100 bytes.



(a) 5dBm



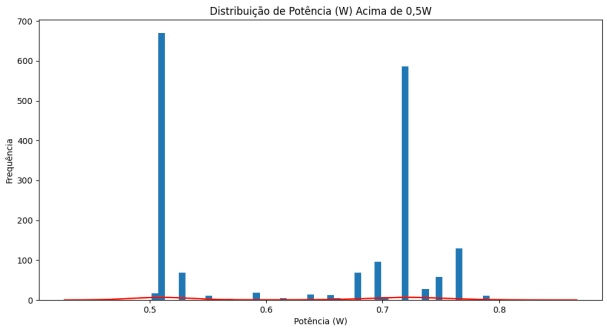
(b) 8dBm



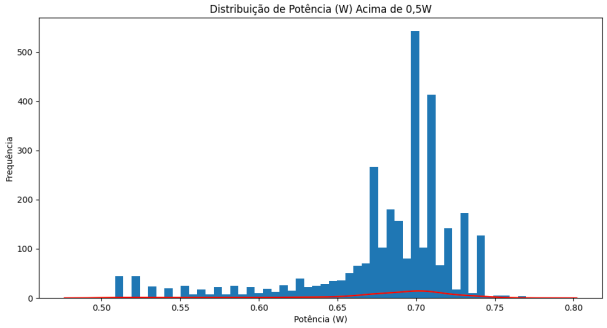
(c) 11dBm

Fonte: Autor.

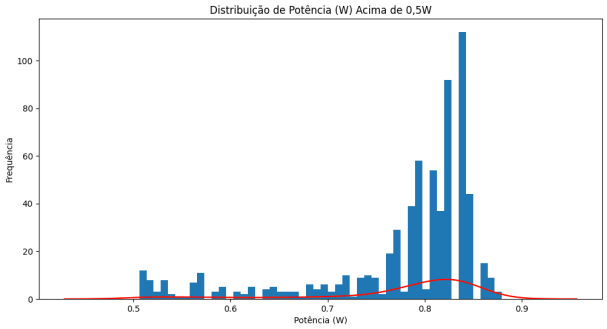
Figura 18 – Análise do protocolo HTTP para pacotes de 500 bytes.



(a) 5dBm



(b) 8dBm



(c) 11dBm

Fonte: Autor.

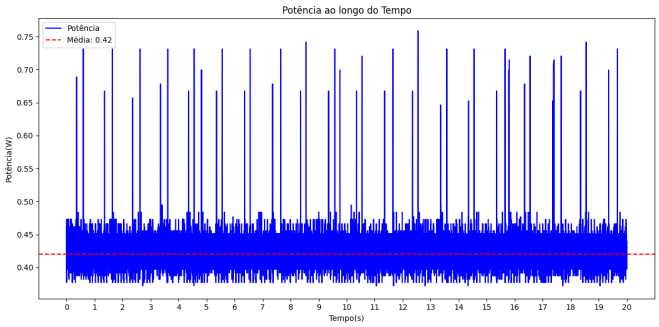
Realizando a análise da potência média para valores acima de 0,5W, foi possível observar que o aumento da potência do sinal de transmissão impactou na potência média do dispositivo. Para pacotes de 100 bytes, com envios de 5, 8 e 11dBm, a potência foi de 0,63W, 0,68W e 0,75W, respectivamente. Em relação aos pacotes de 300 bytes, a potência média foi de 0,59W, 0,61W e 0,73W para os mesmos níveis de potência (5, 8 e 11dBm), respectivamente. Quanto aos pacotes de 500 bytes, o consumo médio foi de 0,63W, 0,68W e 0,75W, respectivamente.

4.3 ANÁLISE DO CONSUMO DO PROTOCOLO MQTT

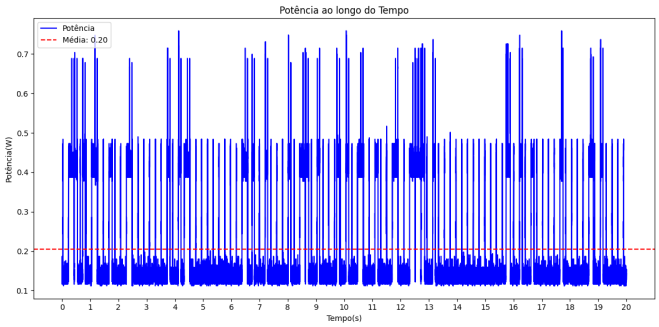
Nesta seção, é apresentada a análise do consumo energético no ESP8266, por meio do protocolo MQTT. Todos os gráficos gerados exibem a potência consumida ao longo de um período de 20 segundos, assim como a potência média durante esse intervalo.

A Figura 19 apresenta os dados da potência média para pacotes de 100, 300 e 500 bytes, considerando uma potência de transmissão de 5dBm. Observou-se que o envio de um pacote de 100 bytes resultou em uma potência média de 0,42W, enquanto os conjuntos de dados de 300 e 500 bytes apresentaram potências médias de 0,20W e 0,19W, respectivamente.

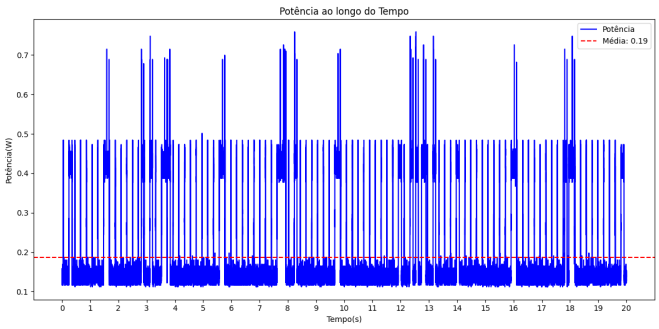
Figura 19 – Consumo do protocolo MQTT para 5dBm de transmissão.



(a) 100 bytes



(b) 300 bytes

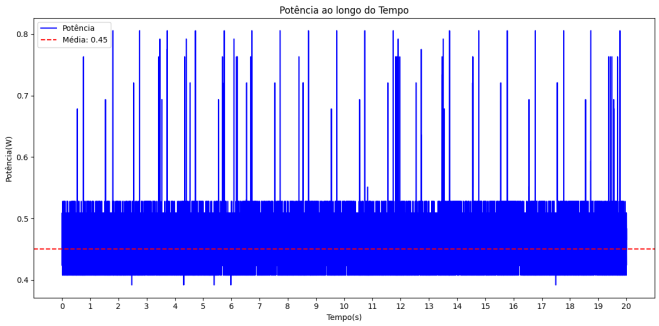


(c) 500 bytes

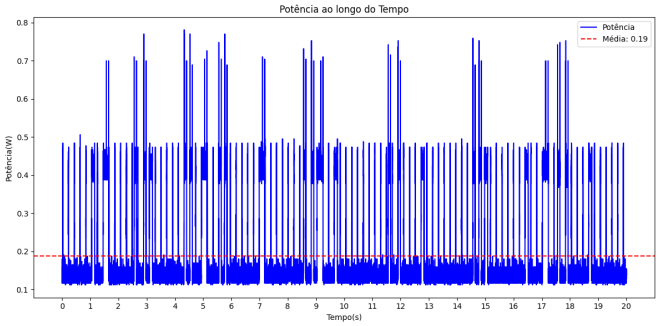
Fonte: Autor.

Os dados referentes à potência média para pacotes de 100, 300 e 500 bytes, considerando uma potência de transmissão de 8dBm, estão ilustrados na Figura 20. Observou-se que o envio de um pacote de 100 bytes resultou em uma potência média de 0,45W, enquanto os conjuntos de dados de 300 e 500 bytes apresentaram uma potência média de 0,19W.

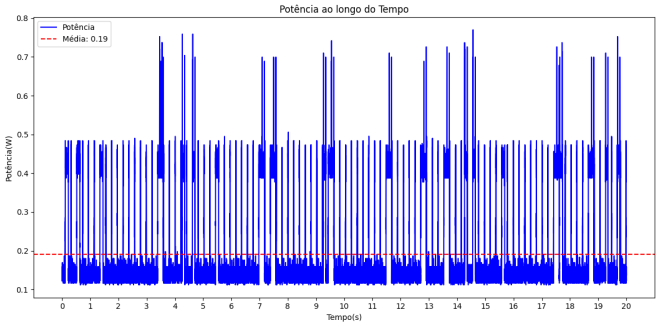
Figura 20 – Consumo do protocolo MQTT para 8dBm de transmissão.



(a) 100 bytes



(b) 300 bytes

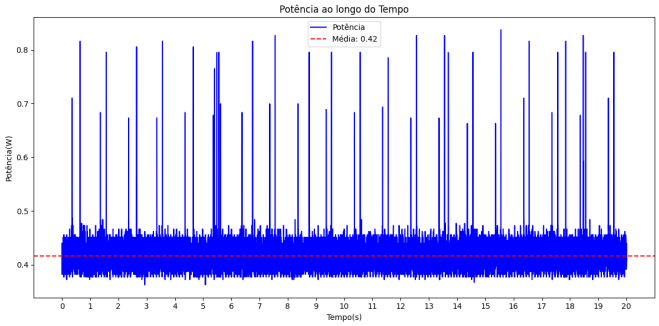


(c) 500 bytes

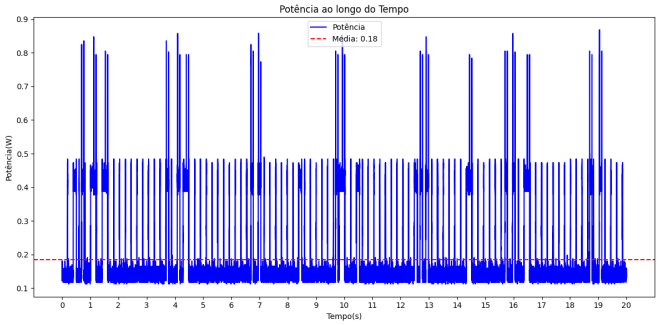
Fonte: Autor.

A Figura 21 demonstra a potência média para pacotes de 100, 300 e 500 bytes em uma potência de transmissão de 11dBm. Neste teste, o cenário de envio de um pacote de 100 bytes resultou em uma potência média de 0,42W, enquanto os conjuntos de dados de 300 e 500 bytes consumiram, em média, 0,18W.

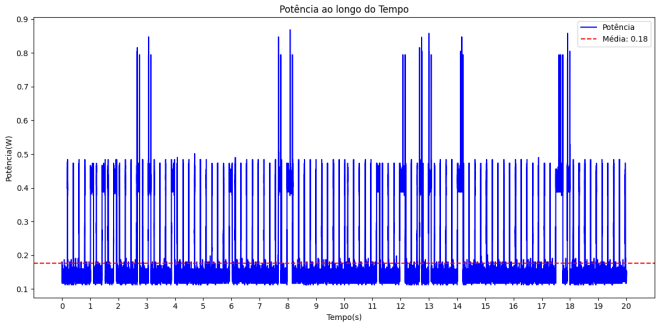
Figura 21 – Consumo do protocolo MQTT para 11dBm de transmissão.



(a) 100 bytes



(b) 300 bytes

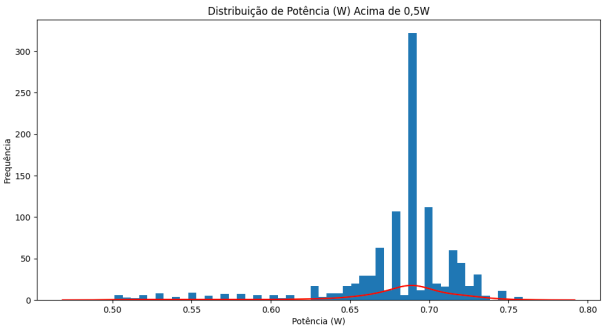


(c) 500 bytes

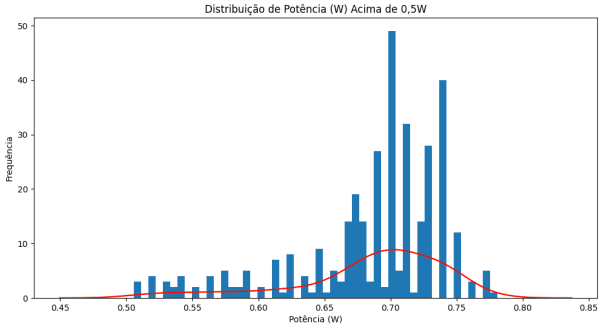
Fonte: Autor.

Para obter uma compreensão mais detalhada do consumo no ESP8266 variando a potência do sinal de transmissão com o protocolo MQTT, foi utilizada a mesma abordagem de gráfico histrograma, focando em valores acima de 0,5 W. A Figura 22 mostra os testes de pacotes de 300 bytes com a variação da potência de transmissão.

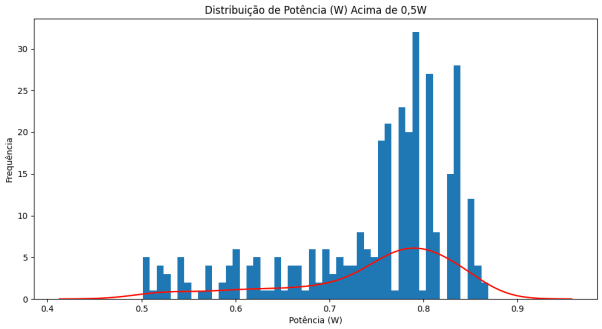
Figura 22 – Análise do protocolo MQTT para pacotes de 300 bytes.



(a) 5dBm



(b) 8dBm

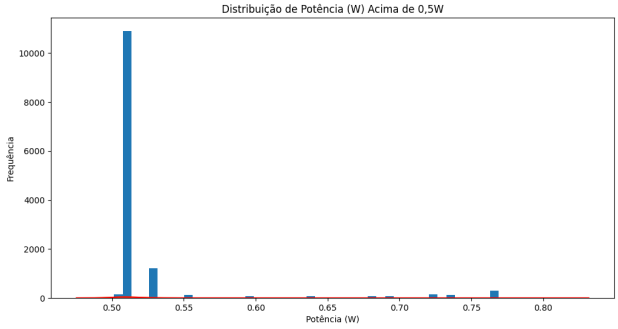


(c) 11dBm

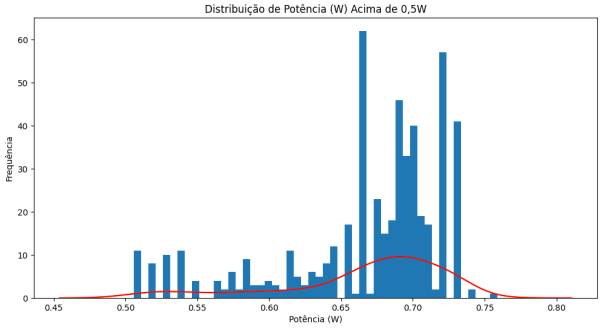
Fonte: Autor.

Verificou-se que aumentando a potência do sinal de transmissão, valores mais altos de potência aparecem com maior frequência. Analisando os gráficos, se observa o mesmo comportamento para os pacotes de 100 e 500 bytes, as Figuras 23 e 24 demonstram isso.

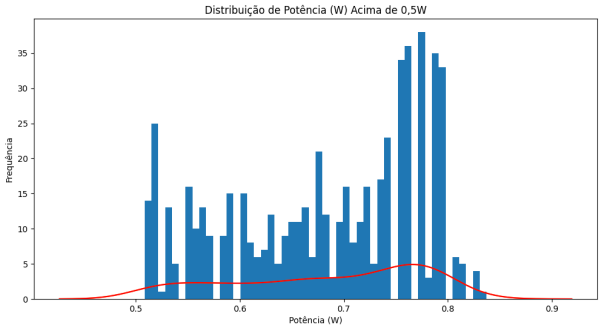
Figura 23 – Análise do protocolo MQTT para pacotes de 100 bytes.



(a) 5dBm



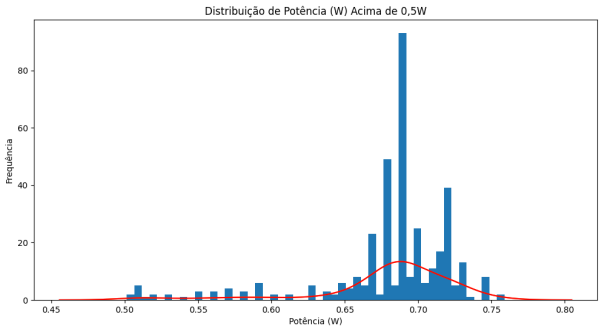
(b) 8dBm



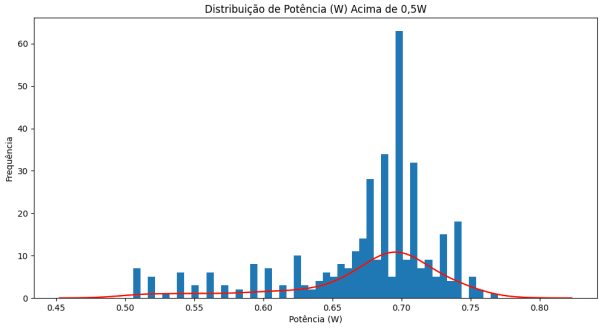
(c) 11dBm

Fonte: Autor.

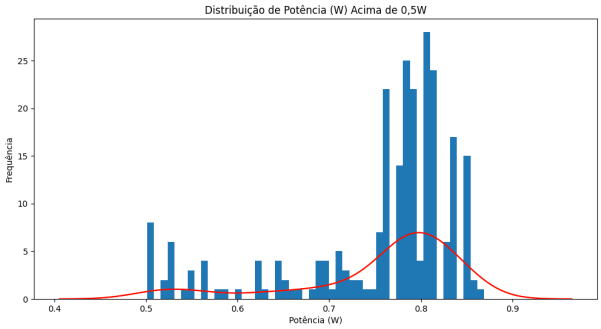
Figura 24 – Análise do protocolo MQTT para pacotes de 500 bytes.



(a) 5dBm



(b) 8dBm



(c) 11dBm

Fonte: Autor.

Realizando a análise da potência média para valores acima de 0,5W, foi possível observar que o aumento da potência do sinal de transmissão impactou na potência média do dispositivo. Para pacotes de 100 bytes, com envios de 5, 8 e 11dBm, a potência foi de 0,62W, 0,66W e 0,68W, respectivamente. Em relação aos pacotes de 300 bytes, a potência foi de 0,67W, 0,68W e 0,74W para os mesmos níveis de potência (5, 8 e 11dBm), respectivamente. Quanto aos pacotes de 500 bytes, a potência média foi de 0,65W, 0,68W e 0,75W, respectivamente.

4.4 DISCUSSÃO DOS RESULTADOS

Com base nas coletas de dados e análises gráficas realizadas, os resultados deste estudo corroboram com os resultados obtidos por Stefanec e Kusek (2021) e Wukkadada et al. (2018), em que o protocolo MQTT tem um consumo energético menor comparado ao HTTP. Para realizar a comparação do consumo de energia, multiplicam-se os valores da potência média pelo tempo de captura de dados, que era de 20 segundos, gerando valores em Joules. O Joule é a medida básica de energia, utilizado para quantificar a quantidade de trabalho realizado ou a energia transferida em diferentes processos físicos(HALLIDAY et al., 1996).

O Quadro 1 demostra a comparação do consumo de energia para 5dBm de transmissão. Para pacotes de 100 bytes, o protocolo HTTP consumiu cerca de 7,15% a mais que o MQTT e para pacotes de 300 e 500 bytes o HTTP teve um consumo cerca de 5% maior que o MQTT.

Quadro 1 – Comparação do consumo de energia para 5dBm

Protocolo	100 bytes	300 bytes	500 bytes
HTTP	9J	4,2J	4J
MQTT	8,4J	4J	3,8J

Fonte: Autor.

O Quadro 2 ilustra a análise comparativa do consumo de energia para transmissões de 8dBm. Em relação a pacotes de 100 bytes, observa-se que o protocolo HTTP apresentou um consumo aproximadamente 4,5% superior ao MQTT. Já para pacotes de 300, o consumo do HTTP para foi 5,25% maior que o do MQTT , e 500 bytes foi 10,5% maior.

Quadro 2 – Comparação do consumo de energia para 8dBm

Protocolo	100 bytes	300 bytes	500 bytes
HTTP	9,4J	4J	4,2J
MQTT	9J	3,8J	3,8J

Fonte: Autor.

O Quadro 3 apresenta a comparação do consumo de energia para transmissões de 11dBm. Neste teste, para pacotes de 100 bytes, tanto o HTTP quanto o MQTT registraram o mesmo consumo, cabe ressaltar que este foi o único teste que apresentou ambos apresentaram o mesmo valor de consumo. Em pacotes de 300 bytes, o HTTP consumiu 22% a mais que o MQTT, enquanto em pacotes de 500 bytes, o protocolo HTTP teve um consumo aproximadamente 27% maior que o MQTT.

Quadro 3 – Comparação do consumo de energia para 11dBm

Protocolo	100 bytes	300 bytes	500 bytes
HTTP	8,4J	4,4J	4,6J
MQTT	8,4J	3,6J	3,6J

Fonte: Autor.

A relação entre período de trasmissão e o tamanho do *payload* mostrou em que transmissões a cada 1 segundo, o consumo energético é maior para ambos os protocolos. Para o HTTP, observou-se que aumentando os valores da potência de transmissão para 8 e 11dBm os pacotes de 500 bytes tiveram um consumo maior que 300 bytes. Já para o protocolo MQTT o consumo se manteve o mesmo para os pacotes de 300 e 500 bytes, nessa variação de valores de potência de 8 e 11dBm.

5 CONCLUSÃO

O presente trabalho apresentou comparações dos protocolos HTTP e MQTT, em diferentes cenários, analisando o consumo energético em um ESP8266. Considerando variáveis como tamanho de pacotes, atraso de transmissão e potência do sinal de transmissão. A análise desses aspectos visa proporcionar uma compreensão aprofundada dos impactos desses protocolos na eficiência energética de dispositivos IoT.

Ao examinar a relação entre a variabilidade do tamanho de pacotes e atraso de transmissão, observou-se que o consumo de energia foi no mínimo 90% maior para transmissões de pacotes de 100 bytes em intervalos de 1 segundo, comparado aos demais intervalos e pacotes de dados. Logo a relação entre tamanho de pacote e atraso de transmissão mostrou uma eficiência energética maior para intervalos maiores com *payloads* elevados.

A potência do sinal de transmissão não teve um impacto relevante no consumo médio. Isso se deve ao fato de que o tempo de captura dos dados foi limitado a apenas 20 segundos, devido às restrições do osciloscópio. Em futuras abordagens, a utilização de um osciloscópio mais robusto resolveria esse problema.

Uma abordagem alternativa para avaliar o impacto da variação da potência de transmissão no consumo seria a adoção de uma metodologia diferente da exclusão de valores acima de 0,5W. Considerar a implementação de um método que permita uma análise mais precisa, trabalhar em diferentes faixas de potência. Isso poderia envolver a categorização dos dados em intervalos específicos, possibilitando uma compreensão mais detalhada de como o consumo varia em relação à potência de transmissão.

A partir das análises gráficas realizadas, é possível afirmar que o HTTP tem um consumo energético maior comparado ao MQTT, tornando-o menos indicado para uso em dispositivos IoT, quando o consumo de energia é um fator importante para a implementação de algum sistema IoT específico.

Este trabalho oferece uma base para futuros estudos sobre a relação de consumo de energia e protocolos de comunicação em contextos específicos, também oferece percepções práticas valiosas para a implementação eficaz de comunicação em dispositivos IoT. Nesse sentido, este estudo contribui para o avanço contínuo no campo de pesquisa e desenvolvimento tecnológico relacionado à Internet das Coisas.

REFERÊNCIAS

ÁLVAREZ-ROSADO, G. L. et al. Energy consumption of an internet of things development board. **Revista de ciencias tecnológicas**, Universidad Autónoma de Baja California, v. 5, n. 4, 2022. Disponível em: <https://recit.uabc.mx/index.php/revista/article/view/234>.

Eclipse Foundation. **Eclipse Newsletter**. 2014. Acesso em: 30 de agosto de 2023. Disponível em: <https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2-.php>.

FOUNDATION, E. **Eclipse IoT Sandboxes**. 2023. Acesso em 31 ago. 2023. Disponível em: <https://iot.eclipse.org/projects/sandboxes/>.

GOURLEY, D.; TOTTY, B. **HTTP: the definitive guide**. [S.l.]: "O'Reilly Media, Inc.", 2002.

HALLIDAY, D. et al. **Fundamentos de física**. [S.l.: s.n.], 1996.

JAIKAR, S. P.; IYER, K. R. A survey of messaging protocols for iot systems. **International Journal of Advanced in Management, Technology and Engineering Sciences**, v. 8, n. 2, p. 510–514, 2018. Disponível em: <https://www.ijamtes.org/gallery/12-.feb%20ijamtes%20254.pdf>.

MADAKAM, S. et al. Internet of things (iot): A literature review. **Journal of Computer and Communications**, Scientific Research Publishing, v. 3, n. 05, p. 164, 2015. Disponível em: <https://www.scirp.org/journal/paperinformation.aspx?paperid=56616>.

MANANDHAR, S. **MQTT based communication in IoT**. 2017. Dissertação (Mestrado), 2017. Disponível em: <https://trepo.tuni.fi/handle/123456789/25376>.

MOUHA, R. A. Internet of things (iot). **Journal of Data Analysis and Information Processing**, Scientific Research Publishing, v. 9, n. 2, p. 77–101, 2021. Disponível em: <https://-/www.scirp.org/journal/paperinforcitation.aspx?paperid=108574>.

NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: IEEE. **2017 IEEE international systems engineering symposium (ISSE)**. [S.l.], 2017. p. 1–7.

OLIVEIRA, S. de. **Internet das coisas com ESP8266, Arduino e Raspberry PI**. [S.l.]: Novatec Editora, 2017.

Oracle Corporation. **Oracle - What is IoT?** 2019. Acesso em 11 nov. 2023. Disponível em: <https://www.oracle.com/br/internet-of-things/what-is-iot/#why-is-iot-important>.

STANFORD-CLARK, A.; TRUONG, H. L. Mqtt for sensor networks (mqtt-sn) protocol specification. **International business machines (IBM) Corporation version**, v. 1, n. 2, p. 1–28, 2013. Disponível em: <https://www.oasis-open.org/committees/download.php/66091-/MQTT-SN_spec_v1.2.pdf>.

STEFANEC, T.; KUSEK, M. Comparing energy consumption of application layer protocols on iot devices. In: IEEE. **2021 16th International Conference on Telecommunications**

(ConTEL). 2021. p. 23–28. Disponível em: <https://ieeexplore.ieee.org/abstract/document-/9495993?casa_token=tQbO70SZ38wAAAAA:5TRQe_07RXoR7DM_qoFVvQgp7ceBA31b0I-gq6eaFAM30mGIP3qOXtPK6yl5w780CiRrFkVGUg>.

TANG, K. et al. Design and implementation of push notification system based on the mqtt protocol. In: ATLANTIS PRESS. **2013 International Conference on Information Science and Computer Applications (ISCA 2013)**. 2013. p. 116–119. Disponível em: <https:/-/www.atlantis-press.com/proceedings/isca-13/9566>.

VERGARA, E. J.; PRIHODKO, M.; NADJM-TEHRANI, S. Mobile location sharing: an energy consumption study. In: **Proceedings of the fourth international conference on Future energy systems**. [s.n.], 2013. p. 289–290. Disponível em: <https://dl.acm.org/doi-/10.1145/2487166.2487211>.

WUKKADADA, B. et al. Comparison with http and mqtt in internet of things (iot). In: IEEE. **2018 International Conference on Inventive Research in Computing Applications (ICIRCA)**. 2018. p. 249–253. Disponível em: <https://ieeexplore.ieee.org/document-/8597401>.

APÊNDICE A – CÓDIGO DO SERVIDOR HTTP

```python
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/post', methods=['POST'])
def receber_pacote_bytes():
    try:
        data = request.data  # Lê os dados brutos do corpo da solicitação HTTP
        if data:
            tamanho_pacote = len(data)
            # Faz algo com os dados em bytes aqui
            print(f"Pacote de bytes recebido. Tamanho: {tamanho_pacote} bytes")
            return jsonify({'message': f'Pacote de bytes recebido com sucesso. Tamanho: {tamanho_pacote} bytes'}), 200
        else:
            return jsonify({'error': 'Nenhum pacote de bytes no corpo da solicitação'}), 400
    except Exception as e:
        return jsonify({'error': str(e)}), 500
if __name__ == '__main__':
    app.run(debug=True, host='SEU IP AQUI', port=5000)
```

APÊNDICE B – CÓDIGO DO CLIENTE HTTP NO ESP8266

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <map>

const char* SSID_REDE = "SEU SSID AQUI";
const char* SENHA_REDE = "SUA SENHA AQUI";
const char* serverName = "SEU IP AQUI:5000/post";
unsigned long startTime;
unsigned long currentTime;

 byte dataToSend[100];
 unsigned int dataSize = 100;
 unsigned long delayValue = 1000; // Delay do tempo de
    transmissão

 std::map<unsigned long, unsigned int> delayToSize = {
    {1000, 100}, // 1 segundo corresponde a 100 bytes
    {3000, 300}, // 3 segundos correspondem a 300 bytes
    {5000, 500}  // 5 segundos correspondem a 500 bytes

};

WiFiClient client;
HTTPClient http;

void init_wifi(void);
void conecta_wifi(void);
void verifica_conexao_wifi(void);

void init_wifi(void)
{
    Serial.println("------WI-FI -----");
    Serial.println("Conectando-se a rede: ");
    Serial.println(SSID_REDE);
    Serial.println("\nAguarde...");

```

```
36      conecta_wifi();
37  }
38
39  void conecta_wifi(void)
40  {
41      if (WiFi.status() == WL_CONNECTED)
42      {
43          return;
44      }
45
46      WiFi.begin(SSID_REDE, SENHA_REDE);
47
48      while (WiFi.status() != WL_CONNECTED)
49      {
50          delay(100);
51      }
52
53      Serial.println("Conectado com sucesso a rede wi-fi \n");
54      Serial.println(SSID_REDE);
55  }
56
57  void verifica_conexao_wifi(void)
58  {
59      conecta_wifi();
60  }
61
62  void setup(){
63    Serial.begin(9600);
64    startTime = millis();// Recebe o tempo inicial em que o
         ESP8266 foi ligado
65    init_wifi(); // Inicia a conexão WI-FI
66    WiFi.setOutputPower(11); // Define a potência de
         transmissão
67
68      if (delayToSize.find(delayValue) != delayToSize.end()) {
69      dataSize = delayToSize[delayValue];
70      } else {
71      // Caso o valor de atraso não esteja mapeado, use um
           tamanho padrão
```

```
72          dataSize = 100;
73      }
74  }
75  void loop(){
76    startTime = millis();
77    verifica_conexao_wifi();
78    http.begin(client, serverName);
79    http.addHeader("Content-Type", "application/json");
80    int httpResponseCode = http.POST(dataToSend, dataSize);
81    http.end();
82    currentTime = millis();
83    delay(delayValue - (currentTime - startTime));
84  }
```

APÊNDICE C – CÓDIGO DO SERVIDOR MQTT

```python
import paho.mqtt.client as mqtt

# Configurações do broker MQTT
mqtt_server = "mqtt.eclipseprojects.io"
mqtt_port = 1883

# Tópico MQTT para receber mensagens
mqtt_topic = "topico/lucas/tcc"

# Função de callback chamada quando uma mensagem é recebida
def on_message(client, userdata, message):
    payload = message.payload
    length = len(payload)
    print(f"Recebido pacote de {length} bytes")


# Configura e conecta o cliente MQTT
client = mqtt.Client()
client.on_message = on_message
client.connect(mqtt_server, mqtt_port, 60)

# Inscreve-se no tópico MQTT para receber mensagens
client.subscribe(mqtt_topic)
```

APÊNDICE D – CÓDIGO DO CLIENTE MQTT NO ESP8266

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <map>
#define MQTT_MAX_PACKET_SIZE 500
#define MQTT_MAX_TRANSFER_SIZE 500 // Defina o tamanho máximo desejado

const char* SSID_REDE = "SEU SSID AQUI";
const char* SENHA_REDE = "SUA SENHA AQUI";

const char* mqttServer = "mqtt.eclipseprojects.io";
const int mqttPort = 1883;
unsigned long startTime;
unsigned long currentTime;

 byte dataToSend[100];
 unsigned int dataSize = 100;
 unsigned long delayValue = 1000; // Delay do tempo de transmissão

 std::map<unsigned long, unsigned int> delayToSize = {
    {1000, 100},// 1 segundo corresponde a 100 bytes
    {3000, 300}, // 3 segundos correspondem a 300 bytes
    {5000, 500}  // 5 segundos correspondem a 500 bytes

};
// Cria uma instância do cliente Wi-Fi
WiFiClient espClient;
PubSubClient client(espClient);
void init_wifi(void); // Função que inicializa a conexão WI-FI
void conecta_wifi(void);// Função que conecta o ESP8266 na rede WI-FI
void verifica_conexao_wifi(void); // Função que verifica se o ESP8266 está conectado na rede

```

```
32 void init_wifi(void)
33 {
34     Serial.println("------WI-FI -----");
35     Serial.println("Conectando-se a rede: ");
36     Serial.println(SSID_REDE);
37     Serial.println("\nAguarde...");
38 
39     conecta_wifi();
40 }
41 
42 void conecta_wifi(void)
43 {
44     if (WiFi.status() == WL_CONNECTED)
45     {
46         return;
47     }
48 
49     WiFi.begin(SSID_REDE, SENHA_REDE);
50 
51     while (WiFi.status() != WL_CONNECTED)
52     {
53         delay(100);
54     }
55 
56     Serial.println("Conectado com sucesso a rede wi-fi \n");
57     Serial.println(SSID_REDE);
58 }
59 
60 void verifica_conexao_wifi(void)
61 {
62     conecta_wifi();
63 }
64 
65 void setup() {
66   // Inicialização da conexão serial
67   Serial.begin(9600);
68   startTime = millis();// Recebe o tempo inicial em que o
        ESP8266 foi ligado
69 
```

```
70    // Configuração do broker MQTT
71    client.setBufferSize(MQTT_MAX_PACKET_SIZE);
72    client.setServer(mqttServer, mqttPort);
73    init_wifi(); // Inicia a conexão WI-FI
74    WiFi.setOutputPower(11);
75
76    if (delayToSize.find(delayValue) != delayToSize.end()) {
77      dataSize = delayToSize[delayValue];
78    } else {
79      // Caso o valor de atraso não esteja mapeado, use um
          tamanho padrão
80        dataSize = 100;
81    }
82
83  }
84
85  void loop() {
86
87    // Conectando-se ao broker MQTT
88    startTime = millis();
89    verifica_conexao_wifi();
90    if (!client.connected()) {
91      reconnect();
92    }
93
94    client.publish("topico/lucas/tcc",dataToSend, dataSize); //
        publica no tópico desejado
95    currentTime = millis();
96    delay(delayValue - (currentTime - startTime)); // Intervalo
        de envio das mensagens
97  }
98
99  void reconnect() {
100   // Loop até que esteja conectado ao broker MQTT
101   while (!client.connected()) {
102     Serial.println("Conectando ao broker MQTT...");
103     if (client.connect("ESP8266Client")) {
104       Serial.println("Conectado ao broker MQTT");
105     } else {
```

```
106         Serial.print("Falha na conexão ao broker, código de
                erro: ");
107         Serial.print(client.state());
108         Serial.println(" Tentando novamente em 0.1 segundos..."
                );
109         delay(100);
110       }
111     }
112 }
```

NUP: 23081.161542/2023-03 **Prioridade:** Normal

Homologação de ata de defesa de TCC e estágio de graduação

125.322 - Bancas examinadoras de TCC: indicação e atuação

COMPONENTE

Ordem	Descrição	Nome do arquivo
14	Trabalho de conclusão de curso (TCC) (125.32)	TCC_LUCAS-BORDIGNON-VERSAO-FINAL.pdf

Assinaturas

01/02/2024 12:39:48

LUCAS MULLER BORDIGNON (Aluno de Graduação - Aluno Regular)
07.09.09.01.0.0 - Curso de Engenharia de Computação - 121624