INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM ELETRÔNICA INDUSTRIAL

GABRIEL DA SILVA CAETANO

IMPLEMENTAÇÃO DE FIRMWARE PARA DISPOSITIVO DE INTEGRAÇÃO ENTRE CONTROLADORES DE TEMPERATURA E SISTEMA EM NUVEM UTILIZANDO CONEXÃO WI-FI E PROTOCOLO MQTT

FLORIANÓPOLIS, 2021.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM ELETRÔNICA INDUSTRIAL

GABRIEL DA SILVA CAETANO

IMPLEMENTAÇÃO DE FIRMWARE PARA DISPOSITIVO DE INTEGRAÇÃO ENTRE CONTROLADORES DE TEMPERATURA E SISTEMA EM NUVEM UTILIZANDO CONEXÃO WI-FI E PROTOCOLO MQTT

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 Tecnólogo em Eletrônica Industrial.

Orientador:
Prof. Samir Bonho, Me. Eng.

FLORIANÓPOLIS, 2021.

Ficha de identificação da obra elaborada pelo autor.

Caetano, Gabriel da Silva

Implementação de *Firmware* para dispositivo de integração entre controladores de temperatura e sistema em nuvem utilizando conexão Wi-Fi e protocolo MQTT. Gabriel da Silva Caetano; orientação de Samir Bonho – Florianópolis, SC, 2021.

72 p.

Trabalho de Conclusão de Curso (TCC) – Instituto Federal de Santa Catarina, Câmpus Florianópolis. Curso Superior de Tecnologia em Eletrônica Industrial.

IMPLEMENTAÇÃO DE FIRMWARE PARA DISPOSITIVO DE INTEGRAÇÃO ENTRE CONTROLADORES DE TEMPERATURA E SISTEMA EM NUVEM UTILIZANDO CONEXÃO WI-FI E PROTOCOLO MQTT

GABRIEL DA SILVA CAETANO

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

Florianópolis, 30 de Abril, 2021.

Banca Examinadora:

Samir Bonho, Me. Eng.

Charles Borges de Lima, Dr. Eng.

Pedro Giassi Jr.

Pedro Giassi Júnior, Dr. Eng.

Dedico este trabalho a Deus, meus pais Oscar e Célia e minha noiva Marina, que têm estado comigo, dando forças pra prosseguir nesta caminhada.

AGRADECIMENTOS

Agradeço a Deus pela vida e pela sua presença constante e infinita, pelo direcionamento e mão estendida, por trazer forças e me manter com a esperança viva sempre.

Agradeço aos meus pais, Oscar e Célia, por nunca me deixarem faltar nada e por todo amor e esforço demonstrados sempre, o que me dá força para prosseguir e agradeço também a toda minha família, por sua participação única em minha formação e construção de caráter, possibilitando a mim, ser quem hoje sou.

Agradeço a minha noiva Marina, com quem tenho partilhado sonhos e alegrias e que me faz ver a vida por uma ótica diferente e agradeço também aos meus grandes amigos Luiz Henrique P. Coelho, Matheus Schlösser e Wellington Batista Teodoro, companheiros que a graduação me trouxe e que levarei para a vida, os quais não me deixaram desistir e me trouxeram aprendizados além dos limites da instituição.

Agradeço ao meu orientador Samir Bonho, que me direcionou e tranquilizou em todo o processo de construção deste trabalho, desde a concepção até a versão final e agradeço também à professora Cláudia Regina Silveira, que me ajudou e direcionou na escrita e revisão deste trabalho.

Agradeço ao IFSC e a todo o seu corpo docente por me dar a base e estrutura necessária para todos os aprendizados e conhecimentos trazidos para este trabalho e para a vida.

“Busquem, pois, em primeiro lugar o Reino de Deus e a sua justiça, e todas essas coisas serão acrescentadas a vocês.”

Mateus 6:33.

RESUMO

A Internet das Coisas (IoT) é uma das tendências mais importantes da atualidade quando se fala em tecnologia, trazendo a ideia de um mundo totalmente conectado, acessível e controlável. Este trabalho aborda o desenvolvimento de um *firmware*, construído em linguagem de programação C, para um dispositivo comercial, de uma empresa do ramo de tecnologia da grande Florianópolis, totalmente voltado para o mundo da IoT. O desenvolvimento proposto realiza a integração, por meio de conexão sem fio Wi-Fi e protocolo de comunicação MQTT, de um controlador de temperatura com um sistema de monitoramento e controle em nuvem. O código do projeto foi embarcado em um microcontrolador ESP 32 (Espressif Systems). São apresentados, em uma interface gráfica, os dados de temperatura gerados pelos controladores, extraídos pelo dispositivo e enviados ao sistema de monitoramento em nuvem por meio dos pacotes do protocolo MQTT.

Palavras chaves: Internet das Coisas. Dispositivo. Tecnologia. Wi-Fi. MQTT. ESP 32.

ABSTRACT

The Internet of Things (IoT) is one of the most important trends today when it comes to technology, bringing the idea of a fully connected, accessible, and controllable world. This paper addresses the development of a firmware, built in C programming language, for a commercial device, from a company in the technology branch of Florianópolis, totally focused on the IoT world. The proposed development performs the integration, through Wi-Fi wireless connection and MQTT communication protocol, of a temperature controller with a cloud monitoring and control system. The project code was embedded on an ESP 32 microcontroller (Espressif Systems). The temperature data generated by the controllers, extracted by the device and sent to the cloud monitoring system through the MQTT protocol packages are presented in a graphical interface.

Keywords: Internet of Things. Device. Technology. Wi-Fi. MQTT. ESP 32.

LISTA DE FIGURAS

Figura 1 - Revisão histórica sobre o desenvolvimento do Wi-Fi.....22
Figura 2 - Exemplo de infraestrutura de uma rede Wi-Fi.....23
Figura 3 - Tipos de infraestruturas de rede 24
Figura 4 - Exemplo das camadas do protocolo TPC/IP.....29
Figura 5 - Representação do funcionamento do modelo cliente/servidor.....30
Figura 6 - Fluxo do protocolo de comunicação MQTT 33
Figura 7 - Representação do envio de um pacote MQTT com QoS 0.....35
Figura 8 - Representação do envio de um pacote MQTT com QoS 1.....36
Figura 9 - Representação do envio de um pacote MQTT com QoS 2.....37
Figura 10 - Visão geral do sistema.....40
Figura 11 - Protótipo do *hardware* desenvolvido conectado a um controlador de temperatura 41
Figura 12 - Diagrama de blocos internos do ESP 32 45
Figura 13 - Fluxograma da comunicação do dispositivo com os controladores de temperatura 46
Figura 14 - Pseudo código de demonstração da comunicação dos controladores de temperatura com o dispositivo.....47
Figura 15 - Pseudo código de demonstração do *firmware* responsável pela conexão Wi-Fi.....48
Figura 16 - Fluxograma de demonstração do *firmware* responsável pela conexão Wi-Fi 50
Figura 17 - Pseudo código de demonstração do *firmware* responsável pela comunicação MQTT 52
Figura 18 - Fluxograma de demonstração do *firmware* responsável pela comunicação MQTT 53

Figura 19 - Formato do primeiro teste, com o *client* recebendo dados de temperatura do dispositivo.....55

Figura 20 - Código do *client subscriber* para teste de comunicação com o dispositivo.....56

Figura 21 - Formato do segundo teste, com o *client* configurando e lendo dados do dispositivo.....57

Figura 22 - Código do *client subscriber* e *publisher* para teste de comunicação com o dispositivo.....58

Figura 23 - Código do *broker* local para comunicação dos *clients* com o dispositivo.....59

Figura 24 - *Logs* do *broker*.....60

Figura 25 - *Logs* dos dados enviados pelo dispositivo ao *broker* e recebidos pelo *client*.....61

Figura 26 - *Logs* do *broker*.....61

Figura 27 - *Logs* do *client*.....62

Figura 28 - *Logs* do dispositivo.....62

Figura 29 - Amostragem em tempo real da temperatura no *front end* do sistema em nuvem.....64

LISTA DE TABELAS

Tabela 1 - Apresentação de alguns dos padrões IEEE 802.11 21

Tabela 2 - Tipos de controles de pacotes do protocolo MQTT 34

Tabela 3 - Comparação entre três chips com Wi-Fi embarcado 42

LISTA DE ABREVIATURAS E SIGLAS

BSS – *Basic service set*

BSSID – *Basic Service Set ID*

DB - *Data bank*

IoT – *Internet of Things*

IP – *Internet Protocol*

LWT – *Last And Will Testament*

M2M – *Machine to Machine*

MQTT – *Message Queuing Telemetry Transport*

P&D – Pesquisa e Desenvolvimento

QoS – *Quality of Service*

RF – Rádio Frequência

SSID – *Service Set Identity*

TCP – *Transmission Control Protocol*

TLS – *Transport Layer Security*

WECA – *Wireless Ethernet Compatibility Alliance*

SUMÁRIO

1 INTRODUÇÃO 14
1.1 Definição do Problema 15
1.2 Justificativa 16
1.3 Objetivo Geral 18
1.4 Objetivos Específicos 18
2 REVISÃO DA LITERATURA 19
2.1 WI-FI 19
2.1.1 Padrão IEEE 802.11 20
2.1.2 Wi-Fi Alliance 22
2.1.3 Infraestrutura 23
2.1.4 Tipos de Redes 24
2.1.5 Recursos e Processos 26
2.1.6 Segurança 27
2.2 PROTOCOLOS DE COMUNICAÇÃO 28
2.3 TCP/IP 28
2.3.1 Camadas 28
2.3.2 Modelo Cliente/Servidor 30
2.4 MQTT 31
2.4.1 Paradigma de conexão 31
2.4.2 Fluxo da Comunicação 32
2.4.3 Qualidade de Serviço (QoS) 35
2.4.4 Outras ferramentas 37
3 METODOLOGIA 39
4 DESENVOLVIMENTO 40
4.1 Hardware 41
4.2 Comparação entre chips com Wi-Fi integrado 42
4.3 ESP 32 43
4.3.1 Especificações e Recursos 44
4.3.2 Diagrama de Blocos 45
4.4 Firmware 45

4.4.1 Comunicação com os controladores de temperatura 46
4.4.2 Wi-Fi 48
4.4.3 MQTT 51
4.4.4 Testes de Comunicação 54
5 RESULTADOS 60
6 CONCLUSÃO 65
REFERÊNCIAS BIBLIOGRÁFICAS 67

1 INTRODUÇÃO

A tecnologia tem avançado consideravelmente nos últimos anos, transformando os conceitos em relação a produtos e serviços e criando um mundo novo, totalmente conectado. Pode-se afirmar que ela segue os passos da nova Indústria 4.0 que, para Gerber e Romeo (2017) é a quarta grande mudança industrial da história desde a Revolução Industrial do século XVIII. Essa mudança é marcada principalmente pela explosão da computação, chamada de a Internet das Coisas (IoT), a qual é descrita como uma rede de dispositivos interconectados, chamados de *endpoints* (que são os componentes de uma rede IoT, indo de pequenos e simples sensores até grandes e complexos sistemas em nuvem) para fornecer dados pela Internet, sendo utilizada em uma ampla gama de setores como saúde, manufatura, automotivo, varejo e automação predial, entre outros. O valor da IoT é reconhecido em sua grande captura de dados usados para orientar decisões e melhorar a eficiência operacional (GERBER; ROMEO, 2017).

Segundo Gartner (2019), a previsão era que o mercado empresarial e automotivo da IoT aumentasse até a marca de aproximadamente 5,8 bilhões de *endpoints* em 2020; assim, haveria um aumento de 21% em relação a 2019, sendo que no fim desse mesmo ano, mais de 4,8 bilhões de *endpoints* já estavam em uso, totalizando cerca de 21.5% a mais do que em 2018. Ainda, de acordo com Gartner (2019), o setor de automação predial, impulsionado em grande parte por dispositivos de iluminação, seria o segmento com o maior crescimento em 2020 (42%), seguido pelos setores automotivo e de saúde, com previsões de crescimento de 31% e 29% até o ano de 2020, respectivamente. A área da saúde teria como principal aplicação o monitoramento de condições crônicas, enquanto no setor automotivo, os carros com conectividade IoT incorporada seriam complementados por uma variedade de dispositivos adicionais responsáveis por realizar tarefas específicas, como, por exemplo, gerenciamento de frota.

Outro setor de aplicação muito utilizado é o de monitoramento e controle a partir da medição de temperatura, estando presente e sendo importante nos mais inúmeros e variados cenários. Um dos principais e mais comuns exemplos de aplicação do controle de temperatura é a conservação e preparação de alimentos;

de acordo com Lasso *et al*. (2003, p. 7), “um dos fatores mais importantes para a degradação dos tecidos vegetais é a temperatura de armazenamento”. Outro exemplo de aplicação é o controle da incubação de ovos de galinha, cujo controle de temperatura, conforme Viola *et al*. (2019), é crucial para que o embrião se desenvolva no ovo, sendo que temperaturas muito baixas podem impedir o crescimento, e temperaturas muito elevadas podem acelerar o crescimento, ambos com possibilidade de morte embrionária. Existem ainda muitos outros exemplos de aplicações, porém um dos mais evidentes atualmente, em meio à pandemia de Covid 19, é o armazenamento de vacinas. Segundo o Ministério da Saúde (BRASIL, MS, 2013, p. 35), “a alteração da temperatura de conservação pode comprometer a potência imunogênica da vacina”.

A partir dessas constatações, este trabalho abordará o desenvolvimento de um produto com propriedade intelectual (empresa do ramo de tecnologia da grande Florianópolis) voltado para o mercado da IoT, o qual fará a integração de dispositivos de monitoramento e controle de temperatura com sistemas em nuvem. O foco da integração são dispositivos chamados de controladores de temperatura ou termostatos, utilizados nos mais diversos cenários como chocadeiras, refrigeradores, entre outros, que não possuem nenhum tipo de conexão com a Internet por padrão.

Este trabalho possui a seguinte estrutura: no primeiro capítulo será abordada a introdução, trazendo o problema de pesquisa e a justificativa para o desenvolvimento do dispositivo, além dos objetivos gerais e específicos do trabalho. O segundo capítulo apresentará o embasamento teórico sobre o qual este trabalho foi construído. O terceiro capítulo aborda a metodologia utilizada para o desenvolvimento deste trabalho e o capítulo quatro apresenta o desenvolvimento em si, este finalizado pelo capítulo cinco, que apresenta os testes. Por fim, o capítulo seis finaliza o trabalho, apresentando as considerações finais.

1.1 Definição do Problema

O mercado da IoT vem crescendo muito nos últimos anos em todo o mundo. De acordo com Gartner (2019), a previsão era que em 2020 a receita de produtos eletrônicos voltados à IoT totalizaria os 389 bilhões de dólares e, segundo Abes Software (2020, p. 5), “No segmento de IoT, o mercado, incluindo *hardware* de

conectividade, software e serviços, também cresceu, com uma taxa de 18,7% em relação a 2018”.

> [...] A IoT está permitindo às organizações reinventar o seu envolvimento com os clientes. Ela está ajudando as organizações a acelerar a velocidade de entrega de produtos e serviços, bem como a reinventar os processos industriais existentes. (ABES SOFTWARE, 2020, p. 17).

Buscando, então, reinventar e melhorar seus processos, a empresa em questão (a qual possui propriedade intelectual sobre o objeto de estudo), que possui uma série de equipamentos para monitoramento e controle de temperatura, procura se inserir no mundo da IoT. Tendo iniciado o desenvolvimento desses produtos em um momento onde a IoT não estava em tanta evidência (meados de 2010), a empresa, assim como a maior parte do mercado na época, não investiu em conexão com nuvem devido às inúmeras barreiras encontradas no início do desenvolvimento da IoT, barreiras essas que podem ser demonstradas por alguns artigos da época.

> [...] existem várias barreiras que ameaçam diminuir o desenvolvimento da IoT, incluindo a transição para IPv6, ter um conjunto comum de padrões e desenvolver fontes de energia para milhões, até mesmo bilhões, de sensores minúsculos. (EVANS, 2011, p. 2).

Tendo como base os argumentos apresentados, visto que as principais barreiras para a IoT foram vencidas e a mesma se espalhou por todo o mundo, este trabalho visa analisar a vantagem de integrar um *hardware* de monitoramento de temperatura já consolidado a outro hardware que faça conexão sem fio dos dados com algum serviço em nuvem, buscando também verificar a melhor infraestrutura e meio de conexão e o melhor protocolo de comunicação para a aplicação em questão.

1.2 Justificativa

A IoT tem possibilitado uma grande interação de dispositivos eletrônicos com quem os utiliza, sejam pessoas, outros dispositivos eletrônicos, sistemas de monitoramento e controle, entre outros. Segundo Gerber e Romeo (2017), as plataformas IoT fornecem um meio de conexão e gerenciamento de dispositivos de *hardware* e os dados que eles coletam com aplicativos móveis e da web, voltados para os usuários. A IoT também possibilita o desenvolvimento e evolução de inúmeras aplicações e, de acordo com Gartner (2019), a IoT é um conjunto de mercados e verticais individuais, que comercialmente engloba setores de saúde,

edifícios e cidades inteligentes, varejo e agricultura. Na indústria, engloba os setores de serviços públicos, transporte e manufatura, entre outros, atingindo também os mercados automotivos.

Este trabalho se justifica por apresentar uma proposta de integração entre controladores de temperatura e um sistema em nuvem (sistema completo de propriedade intelectual da empresa, a qual possui uma linha de controladores de temperatura, o sistema em nuvem e agora o dispositivo de integração entre ambos), possibilitando que os usuários tenham acesso em tempo real às medidas e atuações dos seus dispositivos, de qualquer lugar, com plataformas que possuem conexão à *Internet*, como celulares e computadores, permitindo a configuração remota, geração de relatórios, melhorias em processos, controle de funcionamento e verificações de falhas, entre outras inúmeras possibilidades.

Essa integração com um sistema em nuvem aumenta o poder de concorrência em relação ao mercado da IoT, visto que a maioria dos dispositivos do mesmo segmento já tem buscado a integração com o mundo em nuvem. Esse avanço dos sistemas em nuvem, dentro das organizações, tem se deparado com muitas aplicações ainda não modernizadas, sendo atualmente, apenas 27% das aplicações já habilitadas para isso. Sendo assim, as empresas já estão se preparando financeiramente para a ampliação e acompanhamento dos serviços em nuvem. A estimativa era que em 2020 os serviços gerenciados, voltados para ambientes de nuvem, totalizariam os 1,2 bilhões de dólares, representando um crescimento de quase 40% em relação ao ano anterior (Abes Software, 2020).

O meio de conexão sem fio definido para a integração dos controladores de temperatura com o sistema em nuvem foi o Wi-Fi por inúmeros fatores como: disponibilidade de infraestrutura, suporte, custos, facilidade de uso e desenvolvimento, entre outros. De acordo com Elkhodr, Shahrestani e Cheung (2016), redes locais sem fio (WLANs) são a tecnologia dominante para acesso sem fio de banda larga em ambientes internos e são usados tanto em produtos profissionais quanto de consumo. A propagação das WLANs tem aumentado muito, dando aos dispositivos equipados com conexão sem fio, um maior grau de mobilidade.

Por fim, a utilização do protocolo MQTT é uma boa escolha por se tratar de um protocolo de comunicação assíncrono, de baixo consumo e facilidade de

integração e escalabilidade. Como afirma Yuan (2017), o protocolo MQTT é altamente portável, de fácil customização e de baixo custo computacional, permitindo ser utilizado em redes de alta latência e banda limitada, podendo oferecer um grande suporte aos mais diversos cenários e aplicações, dispositivos e serviços voltados para a IoT.

1.3 Objetivo Geral

Desenvolver o *firmware* para um dispositivo eletrônico que recolherá os dados extraídos por controladores de temperatura e os enviará, utilizando conexão Wi-Fi e protocolo MQTT, para um sistema em nuvem.

1.4 Objetivos Específicos

Visando o alcance do objetivo geral definido, foram traçados os seguintes objetivos específicos:

a) visualizar o sistema como um todo para melhor compreensão e definição das partes do mesmo a serem desenvolvidas;
b) implementar a comunicação com os controladores de temperatura, garantindo o correto envio e recebimento de dados;
c) analisar os aspectos da conexão sem fio Wi-Fi, visando à implementação em firmware;
d) examinar o protocolo de comunicação MQTT, buscando a extração dos pontos pertinentes a serem implementados em firmware;
e) validar a comunicação por meio do protocolo MQTT a partir de testes;
f) integrar todas as partes desenvolvidas para a validação dos requisitos de implementação do sistema completo.

2 REVISÃO DA LITERATURA

Este capítulo abordará os principais conceitos envolvidos no desenvolvimento deste trabalho, trazendo a base de conhecimento sobre conexão sem fio Wi-Fi, protocolo de comunicação MQTT e alguns detalhes sobre chips que possuem Wi-Fi embarcado e sobre o microcontrolador definido para uso neste projeto, o ESP 32.

Inicialmente serão abordados os fundamentos sobre a conexão sem fio Wi-Fi, trazendo uma revisão histórica, apresentando o conceito, seus principais padrões, infraestruturas, processos, segurança, entre outros aspectos importantes.

Posteriormente, serão apresentados os conceitos sobre protocolos de comunicação e protocolo TCP/IP, que são a base para a apresentação do protocolo de comunicação MQTT. No item sobre o protocolo de comunicação MQTT, será apresentado seu paradigma de conexão, fluxo de comunicação, principais ferramentas e exemplos de aplicações.

Após isso, será apresentada uma rápida comparação entre alguns chips que possuem Wi-Fi embarcado, comparação essa que embasará a escolha do microcontrolador ESP 32 para a aplicação em questão. Por fim, serão apresentados alguns detalhes mais específicos e relevantes sobre o ESP 32.

2.1 WI-FI

As comunicações sem fio têm dominado os meios de conexão atualmente, estando fortemente presentes em nosso dia a dia, como por exemplo, Bluetooth e Wi-Fi.

A conectividade sem fio possibilita uma maior liberdade de movimento por parte dos usuários em relação a redes conectadas por cabos físicos, cenário no qual o movimento é extremamente reduzido. As redes sem fios também são altamente ajustáveis e de fácil e rápida implantação, onde a infraestrutura não necessita ser alterada por conta de um novo usuário, como se vê em redes com cabos, por exemplo, (GAST, 2002).

As redes sem fio não vieram substituir redes com fio, pois possuem inúmeros desafios e problemas a serem enfrentados. Dispositivos que compõem redes sem fio

acabam ficando restritos a operar em uma determinada faixa de frequência, o que acaba limitando as velocidades de comunicação. Utilizar ondas de rádio como meio de transporte pode ser também um grande obstáculo, pois as mesmas são suscetíveis a problemas de propagação, interferências, entre outros. A segurança é também uma das grandes preocupações para redes sem fios, já que os dados ficam disponíveis para qualquer receptor dentro do alcance da comunicação (GAST, 2002).

2.1.1 Padrão IEEE 802.11

O IEEE 802.11 é um conjunto de padrões, consistindo em uma série de avanços tecnológicos desenvolvidos ao longo de muitos anos, definidos para implementação da comunicação de redes locais sem fios (WLANs) nas bandas de frequência de 2.4 GHz e 5 GHz. Cada novo avanço e atualização do padrão é definido por uma emenda ao padrão “802.11”. O padrão 802.11 original permitia no máximo 2 Mbps, funcionando apenas sobre a de 2.4 GHz. O padrão 802.11b implementou atualizações que possibilitaram o aumento da taxa de transferência para 6 Mbps e o 802.11a iniciou o suporte à banda de 5 GHz (JUNIPER NETWORKS, 2020).

Gast (2002) afirma que o padrão 802.11 possui uma grande variedade de nomes, tendo sido chamado de “Ethernet sem fio 802.11”, recordando a Ethernet com fio tradicional (802.3), e posteriormente a *Wireless Ethernet Compatibility Alliance* (WECA) promoveu o nome Wi-Fi, derivado da palavra *wireless fidelity*.

A Tabela 1 apresenta alguns padrões IEEE 802.11, suas datas de lançamento, frequência de operação, entre outros dados.

Tabela 1 - Apresentação de alguns dos padrões IEEE 802.11

Standard	Released	Frequency (GHz)	Speed	Range
IEEE 802.11	1997	2.4	2 Mbps	Indoors: 20 m Outdoors: 100 m
Wi-Fi 1/IEEE 802.11a	1999	5/3.7	54 Mbps	Indoors: 35 m Outdoors: 120/5000 m
Wi-Fi 2/IEEE 802.11b	1999	2.4	11 Mbps	Indoors: 35 m Outdoors: 120 m
Wi-Fi 3/IEEE 802.11g	2003	2.4	54 Mbps	Indoors: 38 m Outdoors: 140 m
Wi-Fi 4/IEEE 802.11n	2009	2.4/5	600 Mbps	Indoors: 70 m Outdoors: 250 m
Wi-Fi 5/IEEE 802.11ac	2013	2.4/5	450 Mbps/1300 Mbps	Indoors: 35 m
IEEE 802.11ad (WiGig)	2012	60	6.7 Gbps	3.3 m
IEEE 802.11ah (HaLow)	2016	0.9	347 Mbps	1 km
Wi-Fi 6/IEEE 802.11ax	2019 est.	2.4/5 GHz	450 Mbps/10.53 Gbps	TBD

Fonte: Adaptado do artigo da Black Box Corporation sobre padrões IEEE 802.11[1].

Assim como apresenta a Tabela 1, a qual traz alguns dos padrões IEEE 802.11, com suas datas de lançamento, frequências de operação, velocidade e alcance em espaços fechados e abertos, pode-se perceber a evolução em relação aos padrões, principalmente quando se olha para as velocidades e os alcances em espaços abertos e fechados, sendo os grandes saltos desde o primeiro padrão no ano de 1999. Existem ainda muitos outros padrões dentro da faixa dos anos de 1999 até 2019 e muitos outros sendo desenvolvidos atualmente[2].

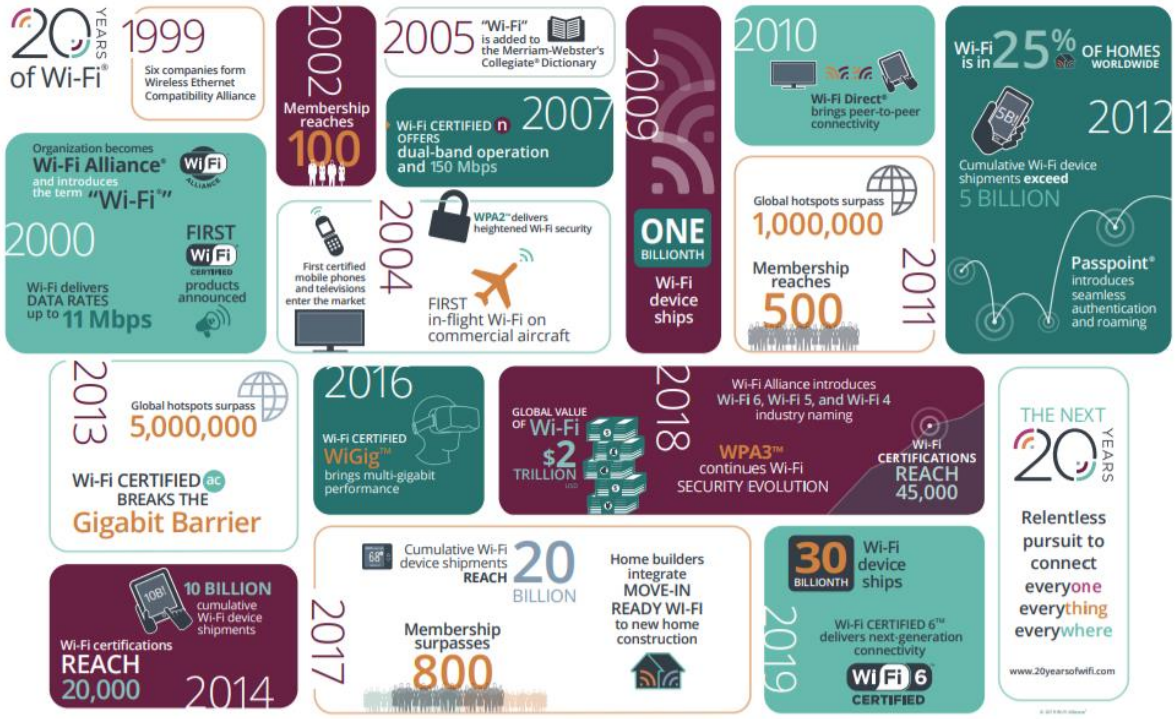
[1] Disponível em <https://www.bboxservices.com/resources/blog/bbns/2018/04/30/802.11-wireless-standards-explained>. Acesso em 21 de abril de 2021.

[2] Linha do tempo de desenvolvimento dos padrões IEEE 802.11 disponível em https://grouper.ieee.org/groups/802/11/Reports/802.11_Timelines.htm.

2.1.2 Wi-Fi Alliance

A Wi-Fi Alliance é uma rede mundial de empresas responsável por gerenciar, regulamentar, desenvolver e impulsionar a evolução global do Wi-Fi. Tem como objetivo desenvolver tecnologias inovadoras, requisitos e programas de teste para garantir que o Wi-Fi irá fornecer a quem o utiliza, interoperabilidade, segurança e confiabilidade (WI-FI ALLIANCE, 2021). A Figura 1 apresenta um histórico sobre o desenvolvimento do Wi-Fi ao longo dos anos.

Figura 1 - Revisão histórica sobre o desenvolvimento do Wi-Fi



Fonte: Adaptada do artigo da Wi-Fi Alliance sobre os 20 anos do Wi-Fi[3].

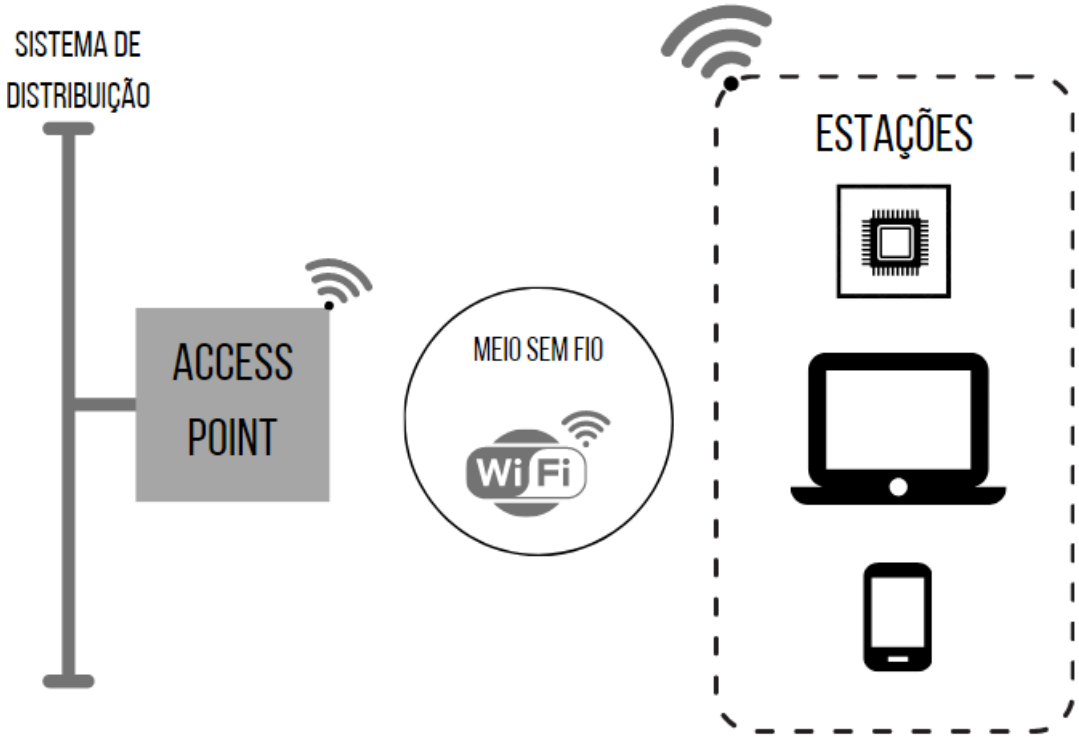
A Figura 1, elaborada pela Wi-Fi Alliance, apresenta uma revisão histórica dos 20 anos de existência do Wi-Fi, passando pela definição do nome, desenvolvimento dos padrões e segurança, até a globalização presente nos dias de hoje.

[3] Disponível em <https://www.wi-fi.org/discover-wi-fi/20-years-of-wi-fi>. Acesso em 21 de abril de 2021.

2.1.3 Infraestrutura

A rede Wi-Fi é regida por uma infraestrutura de funcionamento, formada por vários componentes. A Figura 2 apresenta um exemplo da infraestrutura de uma rede Wi-Fi.

Figura 2 - Exemplo de infraestrutura de uma rede Wi-Fi



Fonte: Adaptado de Gast (2002).

Como apresenta a Figura 2 e assim como também afirma Gast (2002), a infraestrutura de uma rede Wi-Fi é composta por quatro elementos principais: sistema de distribuição, ponto de acesso (*access point*), meio sem fio e as estações e esses elementos podem ser descritos da seguinte maneira:

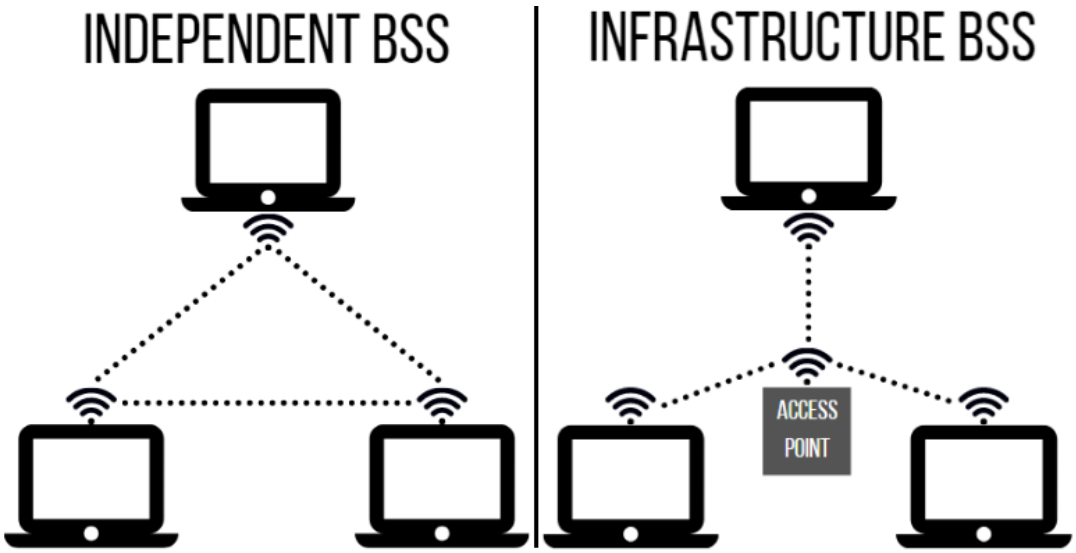
a) sistema de distribuição é o componente utilizado para encaminhar os dados ao seu destino, sendo que o padrão 802.11 não define nenhuma tecnologia específica para o sistema de distribuição;

b) ponto de acesso é componente utilizado como ponte entre os dados das estações e o sistema de distribuição;
c) meio sem fio é o meio físico no qual as ondas são transportadas, sendo neste caso o ar, com os dados transmitidos por radiofrequência;
d) estações são os dispositivos que possuem interfaces de rede sem fio e se conectam ao ponto de acesso para enviar e receber dados, podendo ser computadores, celulares, entre outros.

2.1.4 Tipos de Redes

Gast (2002) afirma que o bloco básico de construção de uma rede no padrão 802.11 é chamado de *basic service set* (BSS), que é basicamente um grupo de estações se comunicando entre si. Estas comunicações ocorrem em uma área chamada de *basic service area* (área de serviço básica), ou seja, é a área de alcance definida pelo meio de comunicação sem fio. Existem dois tipos de rede e a Figura 3 traz uma representação de ambas.

Figura 3 - Tipos de infraestruturas de rede



Fonte: Adaptado de Gast (2002).

A Figura 3 traz a representação dos dois tipos de rede, as *Indepedent* BSS à esquerda e as *Infrastructure* BSS à direita, que são descritas por Gast (2002) como sendo:

a) *independent* BSS (IBSS):
   - neste formato, as estações se comunicam diretamente uma com as outras, sendo que a menor IBSS possível é composta por duas estações. Em geral, são compostas por poucas estações e para fins específicos e de curta duração, sendo assim, muitas vezes chamadas de BSSs *ad hoc* ou redes *ad hoc*. De acordo com Ebradi (2019), o termo *ad hoc* é derivado de uma expressão de origem latina, podendo ser traduzido como “para isto” ou “para esta finalidade”.

b) *infrastructure* BSS:
   - não carregando a sigla IBSS e se diferenciando da primeira por possuir um *access point*. Os dados não trafegam diretamente de estação para estação, possuindo assim dois estágios, sendo o primeiro o envio do pacote ao *access point* e o segundo sendo o envio do dado pelo *access point* à estação de destino. Existem duas vantagens principais em relação ao primeiro tipo de rede:
     1. Por possuir um *access point*, a distância entre as estações não é um problema, visto que as estações apenas precisam estar na área de alcance do *access point*;
     2. Possibilita também o uso de economia de energia para os dispositivos que a desejam utilizar, pois o *access point* pode reconhecer um dispositivo em modo de economia e armazenar os dados enviados a ele enquanto o mesmo está com seu transceptor desligado. Quando o dispositivo ligar seu transceptor para transmitir seus dados, recebe os dados armazenados no *access point*;

No caso das *Infrastructure* BSS, as estações precisam se associar ao *access point*, o qual pode ou não liberar a associação, sendo a solicitação de associação sempre iniciada pelas estações. No padrão 802.11, associação é o processo no qual uma estação se conecta a um *access point*. A associação também é exclusiva de uma estação, ou seja, uma estação só pode se associar a um *access point*, entretanto, um *access point*, pelo padrão 802.11, não tem limitação de associações, porém a taxa de transferência das redes provavelmente limitará o número de

estações possíveis. Para serem reconhecidas pelo *access point*, as estações recebem um identificador chamado de *Basic Service Set* ID (BSSID), que no caso de uma *Infrastructure* BSS, é o endereço MAC da própria estação. Já as estações, para poderem realizar a associação junto ao *access point*, precisam conhecer o *Service Set Identity* (SSID) do *access point* no qual se associará. O SSID é uma *string* utilizada para identificar o *access point*, sendo construída de modo a ser facilmente reconhecida pelos usuários.

2.1.5 Recursos e Processos

O Wi-Fi possui uma série de recursos e processos que possibilitam seu funcionamento completo. Gast (2002) afirma que esses recursos e processos variam produto para produto, sendo definidos de acordo com o tipo de aplicação, sendo que algumas buscam o máximo de recursos possíveis para produtos complexos e outras o mínimo para produtos simples. Alguns desses recursos e processos são o *scanning* (varredura), *authentication* (autenticação), *association* (associação), *power conservation* (conservação de energia), entre outros. Gast (2002) descreve esses recursos e processos da seguinte maneira:

a) *scanning* (varredura) é o processo de identificação das redes existentes na área possível de alcance;
b) *authentication* (autenticação) é processo de identificação de uma estação junto ao *access point*;
c) *association* (associação) é o procedimento de manutenção dos registos da estação para gerenciamento dos pacotes por meio do *access point*;
d) *power conversation* (conservação de energia) é recurso de gerenciamento do estado de atuação do transceptor, desligando-o e ligando-o em certos momentos, visando diminuir o consumo de energia dos dispositivos;

2.1.6 Segurança

Juniper Networks (2020) afirma que a segurança em redes sem fio consiste em uma combinação de criptografia, autenticação e autorização, buscando a máxima proteção da rede. Existem alguns tipos de criptografia para redes sem fio, sendo elas:

a) *wired equivalent privacy* (WEP) é um modo de criptografia já obsoleto. Era composto por chaves de 64 bits e embora não tão comum, também tinha versões de 128, 152 e 256 bits (GREY, 2020). Juniper Networks (2020) afirma que o WEP não possui meios de autenticação;
b) *wi-fi protected access* (WPA) pode ser tido como uma evolução do WEP com base nas vulnerabilidades do mesmo, implementando um modo de criptografia mais forte e fornecendo também um sistema de autenticação mútua (JUNIPER NETWORKS, 2020). É o modo que iniciou a implementação do padrão 802.1i, o qual utiliza o *Temporal Key Integrity Protocol* (TKIP) que alterna a chave de criptografia a cada pacote, utiliza o *Cyclic Redundancy Checking* (CRC) e também uma chave de criptografia fixa para a autenticação dos usuários (GREY, 2020);
c) *wi-fi protected access version 2* (WPA2) é aversão certificada do padrão 802.11i, implementando novos mecanismos de criptografia, não sendo compatível com *hardwares* projetados para funcionamento em WEP (JUNIPER NETWORKS, 2020). É o padrão atual utilizado, o qual eliminou o TKIP e inseriu o CCMP (Protocolo CCM), que permite o uso de um padrão de criptografia avançada. Utiliza também uma chave de criptografia fixa para a autenticação dos usuários (GREY, 2020).

Ambos os modos de criptografia WPA e WPA2 possuem o mecanismo de autenticação PSK (pessoal), que segundo Juniper Networks (2020), fornece credenciais aos usuários para verificação de liberação ou não de permissão de acesso à rede. Essa autenticação requer uma senha, que caso seja a senha configurado no *access point*, libera o acesso à rede ao usuário.

2.2 PROTOCOLOS DE COMUNICAÇÃO

Como afirma Schragl (1983), um protocolo de comunicação é um conjunto de especificações de regras e acordos para as interações de processos cooperativos, tendo seus eventos, relações definidas.

No objeto de estudo em questão, assim como foram utilizados, serão apresentados dois protocolos de comunicação: TCP/IP e MQTT, sendo o primeiro a base para o funcionamento do segundo.

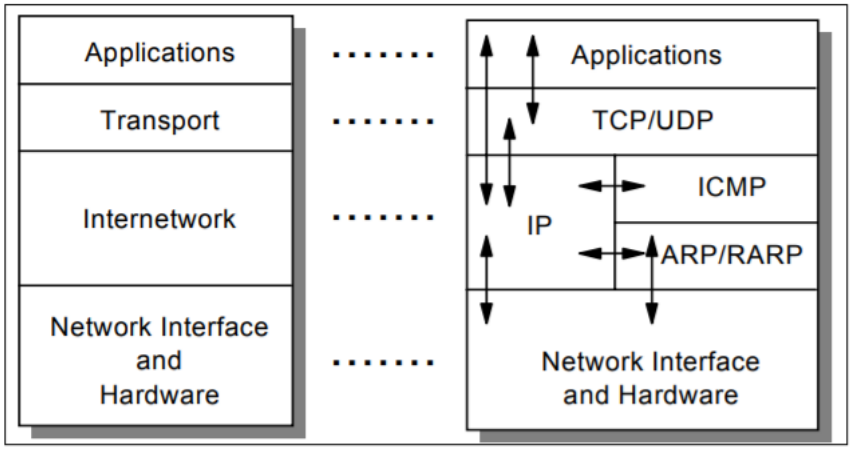
2.3 TCP/IP

De acordo com Parziale et al. (2006), o TCP/IP é um conjunto de dois protocolos, o *Transmission Control Protocol* (TCP) e o *Internet Protocol* (IP) e foi desenvolvido com o intuito de construir uma interconexão de redes que pudesse fornecer serviços de comunicação universal, que ficou conhecida como *internetwork* ou Internet.

2.3.1 Camadas

Como boa parte dos protocolos de rede, o TCP/IP é modelado em camadas, que leva ao termo pilha de protocolo, referindo-se à pilha de camadas no conjunto de protocolos. A partir da divisão da comunicação em camadas, a pilha de protocolo permite a divisão de trabalho, facilita implementações e testes de códigos e o desenvolvimento de camadas alternativas. A comunicação entre as camadas ocorre com aquelas acima e abaixo, ou seja, uma camada fornece um serviço para a camada que está diretamente acima dela e faz uso dos serviços disponibilizados pela camada que está diretamente abaixo dela. Como exemplo, a camada IP permite a transferência de dados de um local para outro sem garantir que a entrega foi realizada e os protocolos de transporte, como é o TCP, utilizam o serviço da camada IP, para gerar aplicações com entrega e fluxo de dados confiáveis (PARZIALE et al., 2006). A Figura 4 traz uma representação das camadas do protocolo.

Figura 4 - Exemplo das camadas do protocolo TPC/IP



Fonte: Tutorial técnico sobre o protocolo TCP/IP da IBM[4].

A Figura 4 traz uma representação das camadas do protocolo TCP/IP e seu funcionamento. De acordo com Parziale et al. (2006), as camadas funcionam da seguinte maneira:

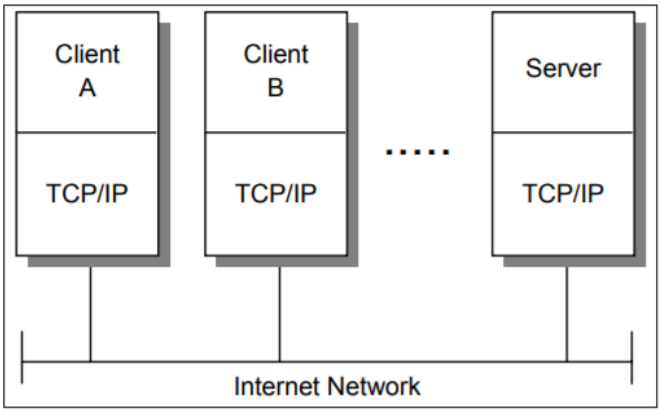
a) camada de aplicação é a camada fornecida pelo programa que utiliza o protocolo TCP/IP para realizar a comunicação;
b) camada de transporte é a camada que possibilita a transferência dos dados de ponta a ponto, entregando dados de uma aplicação para seu receptor;
c) camada de *internetwork* é a camada de atuação do protocolo IP, fornecendo uma imagem de “rede virtual”, protegendo as camadas mais acima arquitetura física abaixo dela;
d) camada de interface de rede é a camada que faz a interface para o hardware de rede real;

[4] Disponível em <https://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg243376.pdf> Acesso em: 18 de abril de 2021.

2.3.2 Modelo Cliente/Servidor

O protocolo TCP é orientado a conexão ponto a ponto, ou seja, não há conexão direta entre mestres e subordinados, porém, em geral, as aplicações utilizam o famoso modelo cliente/servidor para comunicações. O servidor pode ser definido como uma aplicação que oferece um serviço aos usuários da Internet e um cliente é um usuário desse serviço. Uma aplicação completa baseada no modelo cliente/servidor consiste em um servidor e um cliente, que podem ser executados no mesmo ou em sistemas diferentes (PARZIALE et al., 2006). A Figura 5 traz uma representação do funcionamento do modelo cliente/servidor.

Figura 5 - Representação do funcionamento do modelo cliente/servidor.



Fonte: Tutorial técnico sobre o protocolo TCP/IP da IBM[5].

Como apresenta a Figura 5, os clientes são os responsáveis por criarem as requisições para o servidor, e o servidor é quem recebe a requisição, executa o serviço e envia os resultados e as respostas esperadas, podendo lidar com várias requisições de vários clientes ao mesmo tempo. Como grande detalhe, tanto clientes como servidores utilizam o protocolo TCP/IP como transporte para suas requisições, respostas e resultados.

[5] Disponível em <https://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg243376.pdf> Acesso em: 18 de abril de 2021.

2.4 MQTT

Segundo Yuan (2017), o protocolo MQTT foi desenvolvido pela IBM no final dos anos 1990, tendo como principal motivação de aplicação a conexão de sensores em oleodutos com satélites. Como afirma OASIS (2014), o MQTT é um protocolo de transporte de mensagens, executado em TCP/IP no formato publicação/assinatura, sendo de baixo custo computacional, aberto, simples e concebido de forma a ser de fácil implementação, sendo assim considerado a forma ideal para uso em diversos cenários, como por exemplo, a comunicação *Machine to Machine* (M2M) e a IoT. Yuan (2017) ainda afirma que o MQTT é um protocolo que oferece suporte à comunicação assíncrona entre as partes, ou seja, desacopla o emissor e o receptor, tornando-se assim escalável em ambientes de rede não confiáveis.

2.4.1 Paradigma de conexão

O protocolo MQTT utiliza o padrão cliente/servidor – *publish*/*subscribe* que fornece a possibilidade da distribuição de uma mensagem a muitos receptores e desacopla as aplicações (OASIS, 2014). O padrão *publish-subscribe* possui dois tipos de participantes da comunicação: o *server* (chamado de *broker*) e os *clients* que, segundo OASIS (2014), podem ser definidos como:

a) *server/broker*:
   - um programa ou dispositivo intermediário entre os *clients* que publicam as mensagens e os *clients* que realizam as inscrições. O *broker* é responsável por:
     - gerenciar a conexão e as mensagens publicadas pelos *clients*;
     - processar as inscrições e o cancelamento de inscrições requisitadas pelos *clients*;
     - encaminhar as mensagens publicadas aos *clients* inscritos corretamente;

b) *client*:
   - um programa ou dispositivo que utiliza o MQTT como forma de comunicação e é responsável por:
     - iniciar a conexão com o *broker*;
     - publicar mensagens que podem ser de interesse de outros *clients*;
     - realizar a inscrição para receber mensagens que podem o interessar;
     - cancelar inscrições para remover a requisição de mensagens;
     - realizar a desconexão junto ao *broker*;

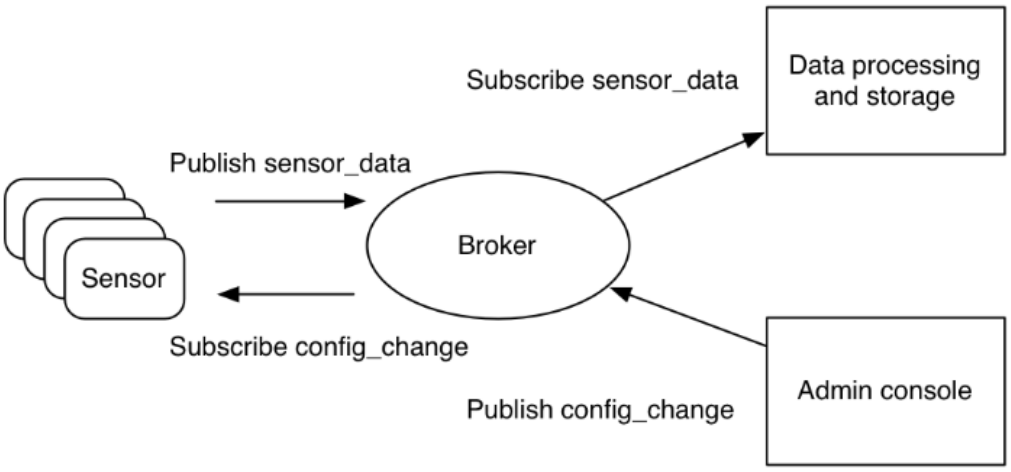
2.4.2 Fluxo da Comunicação

O fluxo da comunicação descreve exatamente o funcionamento do protocolo MQTT, do ponto de vista da aplicação e, segundo Yuan (2017), esse fluxo ocorre da seguinte maneira:

a) Os *clients* realizam a conexão junto ao *broker* e assinam qualquer “tópico” no qual poderão receber mensagens do *broker*;
b) Os *clients* publicam as mensagens em algum tópico;
c) Os *clients* assinantes do tópico em questão, recebem as mensagens do *broker*;

A Figura 6 traz uma representação do fluxo de comunicação.

Figura 6 - Fluxo do protocolo de comunicação MQTT



Fonte: Artigo da IBM sobre o protocolo MQTT[6].

A Figura 6 traz uma representação do fluxo de comunicação do protocolo MQTT, sendo um exemplo que conta com três *clients*, sendo eles alguns sensores que estão publicando e recebendo dados, por meio da inscrição em um tópico, uma interface de administração que apenas está publicando dados e um sistema de processamento e armazenamento de dados, que apenas recebe dados a partir da inscrição em um tópico. Para que esses dados trafeguem entre os *clients* é necessário que eles estejam conectados ao *broker*, que é quem gerencia as conexões, desconexões, inscrições e direcionamento correto dos pacotes publicados aos *clients* inscritos nos tópicos que os interessam.

Como afirma OASIS (2014), o protocolo MQTT funciona a partir da troca de uma série de pacotes de controle definidos pelo próprio protocolo. A Tabela 2 apresenta todos os tipos de controles de pacotes implementados pelo protocolo MQTT, que realizam o controle do fluxo da comunicação.

[6] Disponível <https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/>. Acesso em 10 de abril de 2021.

Tabela 2 - Tipos de controles de pacotes do protocolo MQTT

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Fonte: Documentação do padrão do protocolo MQTT pela OASIS[7].

Como apresenta a Tabela 2, o protocolo MQTT possui vários controles de pacotes para garantir a comunicação entre *server/broker* e os *clients*. Existem

[7] Disponível em <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf> Acesso em: 18 de abril de 2021.

controles de pacotes para conexão (CONNECT e CONNACK) e desconexão (DISCONNECT), para inscrição (SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK) e controles pacotes de publicação (PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP). De acordo com OASIS (2014), esses pacotes consistem em até três partes, seguindo sempre a ordem da tabela.

2.4.3 Qualidade de Serviço (QoS)

Segundo OASIS (2014), o protocolo MQTT é ideal para contextos onde a largura de banda da rede é limitada, sendo assim, a maioria dos sistemas que utilizam o protocolo MQTT trabalham com redes que não são 100% estáveis, ou seja, desconexões são esperadas. Partindo desta premissa, o MQTT possui algumas ferramentas para garantir que mesmo os momentos de instabilidade não sejam um problema para a comunicação em geral e, dentre essas ferramentas, a mais famosa e utilizada é o *Quality of service* (QoS). OASIS (2014) afirma que a entrega de mensagens do protocolo é feita de acordo com o QoS, ou seja, os níveis de qualidade de serviço e existem três níveis de qualidade serviço, sendo eles: QoS 0, QoS 1 e QoS 2.

O QoS 0 é o nível mais baixo, simples, rápido e também o menos custoso computacionalmente falando. A Figura 7 traz uma representação do envio de um pacote com QoS 0.

Figura 7 - Representação do envio de um pacote MQTT com QoS 0

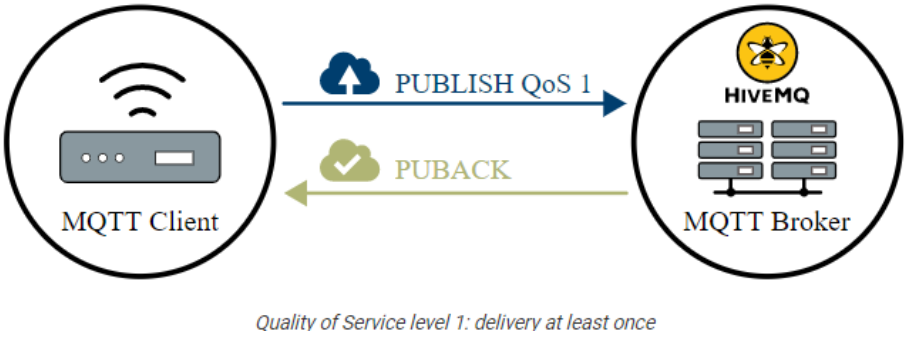
Fonte: Artigo da HIVEMQ sobre qualidade de serviço no MQTT[8].

[8] Disponível em <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/> Acesso em: 18 de abril de 2021.

Como apresenta a Figura 7, o QoS 0 não garante a entrega do dado ao *broker*, pois não recebe nenhuma resposta do *broker* relativo à entrega do dado enviado (THE HIVEMQ TEAM, 2015). Por enviar apenas um pacote, sem aguardo de resposta, é o nível mais leve do protocolo, podendo ser utilizado em cenários onde a velocidade e leveza do envio é mais importante do que a entre de 100% dos pacotes.

O QoS1 é o nível médio de qualidade, onde se garante que a mensagem foi entregue ao *broker* pelo menos uma vez, podendo ser enviada várias vezes (THE HIVEMQ TEAM, 2015). A Figura 8 traz uma representação do funcionamento do envio de um pacote com QoS 1.

Figura 8 - Representação do envio de um pacote MQTT com QoS 1



Fonte: Artigo da HIVEMQ sobre qualidade de serviço no MQTT[9].

Como demonstra a Figura 8, quando o *client* envia uma mensagem com QoS1, o *broker*, caso a receba, envia uma resposta com o pacote de controle PUBACK, indicando o recebimento da mensagem.

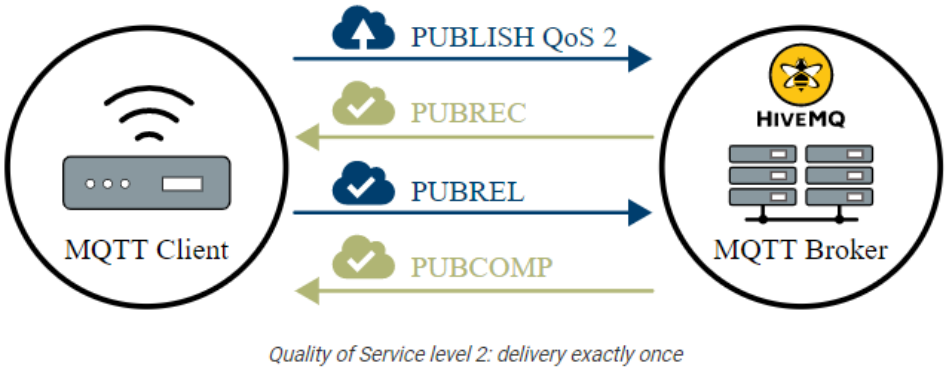
Por utilizar um envio e uma resposta, é um pouco mais custoso em relação QoS 0, porém tem vantagem em relação à garantia de qualidade na entrega dos pacotes, podendo ser utilizado para cenários um pouco mais críticos, onde há a necessidade da garantia de entrega do pacote ao menos uma vez.

Por fim, o último nível é o QoS2, sendo também mais alto e custoso nível em relação a processamento, pois o mesmo garante que a mensagem foi entregue ao

[9] Disponível em <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/> Acesso em: 18 de abril de 2021.

broker somente uma vez (THE HIVEMQ TEAM, 2015). A Figura 9 apresenta o funcionamento de um pacote MQTT com QoS2.

Figura 9 - Representação do envio de um pacote MQTT com QoS 2



Fonte: Artigo da HIVEMQ sobre qualidade de serviço no MQTT[10].

Como demonstra a Figura 9, quando o *client* envia uma mensagem com QoS2, o *broker*, caso a receba, envia uma resposta com o pacote de controle PUBREC, que indica o recebimento da mensagem. Após o envio do PUBREC, o *broker* e aguarda que o *client* responda com outro pacote de controle, o PUBREL. Quando o *client* envia o PUBREL, caso o *broker* receba a resposta, o mesmo envia o pacote final, validando a comunicação, com o último pacote de controle, o PUBCOMP. Todos esses pacotes de resposta carregam um ID da mensagem, que é validado nas pontas da comunicação.

Por carregar quatro pacotes de controle e validações em todas as pontas, o QoS 2 é o nível mais custoso em relação à processamento, porém é o nível mais seguro em relação à garantia da entrega das mensagens. É utilizado em cenários extremamente críticos, onde os pacotes precisam ser entregues apenas uma vez.

2.4.4 Outras ferramentas

O protocolo MQTT ainda conta com muitos detalhes, pacotes e ferramentas para auxiliar os *clients* na construção de um protocolo seguro e robusto. A seguir

[10] Disponível em <https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/> Acesso em: 18 de abril de 2021.

serão apresentadas duas ferramentas muito utilizadas, sendo elas o *Last Will and Testament* (LWT) e as *Retained messages*.

O *Last Will and Testament*, que pode ser traduzido como “Último desejo e testamento”, é controlado pela *Will Flag*. De acordo com OASIS (2014), se a *Will Flag* for definida como um, indica que uma mensagem deve ser armazenada no *broker* e associada à conexão *client* em questão e essa mensagem deve ser publicada quando a conexão com o *client* for encerrada, sem ter sido forçada pelo próprio *client*. Um exemplo de aplicação para essa ferramenta seria:

- Supondo uma aplicação onde há um dispositivo monitorando umidade e temperatura em uma plantação, enviando dados ao *broker* e um sistema em nuvem, com interface gráfica, que interage com o usuário, informando ao mesmo quais dispositivos estão conectados, apresentados os dados de umidade e temperatura, e quais estão desconectados. Neste caso, quando houver a desconexão do dispositivo na plantação, o *broker* irá enviar a mensagem definida aos *clients* que estiverem inscritos no tópico definido e sendo assim, estando inscrito nesse tópico, o sistema em nuvem receberá a mensagem de desconexão e poderá informar ao usuário que o dispositivo na plantação está desconectado.

As *Retained Messages* (“Mensagens retidas”), são controladas pela *Retain Flag* que, caso definido como um em um envio de pacote (PUBLISH), sinaliza que o *broker* deve armazenar a mensagem e seu QoS, para que a mesma possa ser entregue a futuros *subscribers* do tópico no qual a mensagem foi enviada (OASIS, 2014). Um exemplo de aplicação para essa ferramenta seria:

- Supondo um sistema onde há um dispositivo medindo o volume de uma caixa de água e em certo momento o mesmo envia a medida recolhida em um sábado, por exemplo, com a *Retain Flag* ativada. Então um usuário, na segunda-feira, utilizando um aplicativo em um *smartphone* para a visualização do volume da caixa, abre seu aplicativo e logo recebe a medida do sábado, podendo-a visualizar naquele momento.

3 METODOLOGIA

O projeto foi desenvolvido no setor de Pesquisa e Desenvolvimento (P&D) de uma empresa da Grande Florianópolis, em Santa Catarina, que desenvolve e fabrica, entre muitos produtos, controladores de temperatura e inversores de frequência. O nome da empresa será preservado por questões de segurança e propriedade do produto, sendo que o mesmo ainda não foi lançado oficialmente.

A parte inicial do projeto consistiu em uma revisão bibliográfica sobre as principais ferramentas a serem utilizadas no projeto, sendo estas a conexão sem fio Wi-Fi e o protocolo de comunicação MQTT.

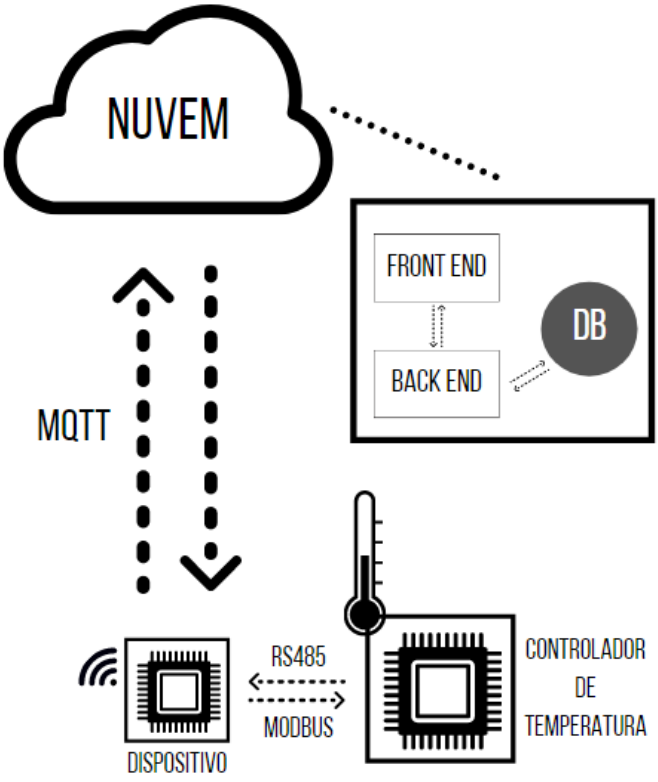
Após isto, fez-se uma revisão sobre a linha de controladores de temperatura da empresa e com isso estudou-se os protocolos de comunicação definidos internamente, tanto MQTT quanto o utilizado para comunicação com os controladores de temperatura.

A partir disto, foi desenvolvido o *firmware* do dispositivo, iniciando pela comunicação com os controladores, garantindo o correto recebimento e envio de dados. Após validar a comunicação com os controladores, implementou-se a conexão Wi-Fi, suas configurações e conexão com as redes disponíveis, e por fim, foi feita a implementação do protocolo de comunicação MQTT, iniciando com testes de envio e recebimento de dados genéricos. Com as três principais partes do *firmware* funcionando, o sistema foi integrado, sendo validado por fim na comunicação com o sistema em nuvem.

4 DESENVOLVIMENTO

O desenvolvimento deste dispositivo foi embasado pela análise geral do funcionamento completo do sistema e todos os seus componentes, e a Figura 10 apresenta uma visão geral do mesmo.

Figura 10 - Visão geral do sistema



Fonte: Autor, 2021.

Como apresentado na Figura 10, o sistema é composto por três partes principais. A primeira parte é composta pelos controladores de temperatura, realizando a coleta dos dados. A segunda parte tem o dispositivo, que por meio de conexão Wi-Fi e protocolo MQTT envia os dados coletados pelo controlador à terceira parte do sistema, que é o sistema em nuvem. O sistema em nuvem é composto por: *front end* (interface gráfica), *back end* (estrutura de apoio e ponte entre o *front end* e o banco de dados) e o banco de dados (representado na Figura 10 como DB - *data bank*), que é acessado apenas pelo *back end*.

Como já mencionado anteriormente, todo o desenvolvimento deste projeto foi baseado no sistema representado pela Figura 10, com um enfoque no *firmware* do dispositivo, implementando os tipos de conexões e comunicações necessárias entre o sensoriamento (controladores de temperatura) e a nuvem.

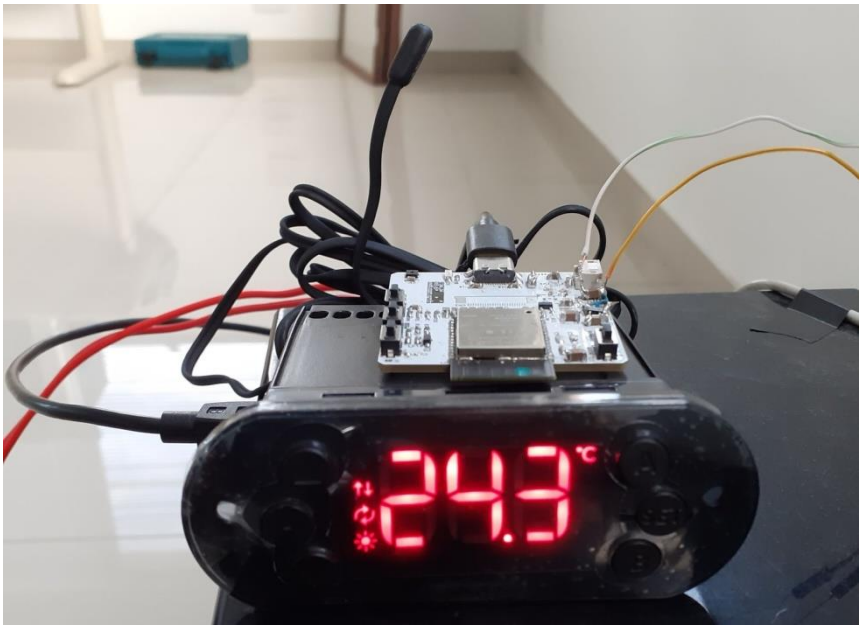
4.1 Hardware

O *hardware* para o presente projeto foi desenvolvido diretamente pela equipe de engenharia de *hardware* da empresa em questão. Dentre algumas especificações, encontrou-se a necessidade de ser um *hardware* menor, com conexão sem fio Wi-Fi. A partir disto, decidiu-se então pelo uso do microcontrolador ESP 32, que já possui internamente os componentes necessários para uso nativo da conexão Wi-Fi, contando inclusive com uma antena embarcada.

A conexão com os controladores de temperatura é feita por meio de um cabo USB C - USB Mini, onde o conector USB C fica no *hardware* desenvolvido e o USB Mini, nos controladores.

A Figura 11 apresenta o primeiro protótipo desenvolvido para o dispositivo em questão, conectado a um controlador de temperatura.

Figura 11 - Protótipo do *hardware* desenvolvido conectado a um controlador de temperatura



Fonte: Autor, 2021.

A Figura 11 apresenta o protótipo desenvolvido (placa branca) conectado a um controlador de temperatura, trazendo assim a ideia do formato final de funcionamento do sistema, quando visto da perspectiva de *hardware*. O sistema, da perspectiva de *hardware*, é composto pelo dispositivo que estará conectado e se comunicando com o controlador de temperatura, o qual estará coletando os dados. Os dados recolhidos pelo controlador serão repassados ao dispositivo e enviados ao sistema em nuvem pelo mesmo.

Detalhes mais específicos de *hardware* não serão apresentados por dois motivos: o primeiro é que não houve participação direta e efetiva do autor na concepção e construção do *hardware* e segundo que o mesmo é de total propriedade intelectual da empresa.

4.2 Comparação entre chips com Wi-Fi integrado

Existem atualmente no mercado inúmeros chips que já disponibilizam o Wi-Fi para o desenvolvimento de aplicações. A Tabela 3 traz uma comparação entre três chips de diferentes empresas, que possuem Wi-Fi integrado, buscando o levantamento de dados comparativos entre os mesmos.

Tabela 3 - Comparação entre três chips com Wi-Fi embarcado

Chip	**Taxa de dados (Mbps)**	**Padrão IEEE 802.11**	**Potência de saída (dBm)**	**Sensitividade (dBm)**	**Tipo de Antena**	**Interfaces de comunicação (I/O)**
WF111-A-V1	72.2	b/g/n	17	-97	Integrado, Chip	PIO, SDIO, SPI
ATWINC1510-MR210PB1961	72.2	b/g/n	18.5	-95	Integrado, *Trace*	I²C, SPI, UART
ESP32-WROOM-32D	150	b/g/n	20.5	-98	Integrado, *Trace*	GPIO, I²C, I²S, PWM, SDIO, SPI, UART

Fonte: Autor, 2021[11].

[11] Dados retirados do site da empresa Digikey, com acesso em 22 de abril de 2021.
Chip WF111-A-V1 - Disponível em: <https://www.digikey.com.br/product-detail/pt/silicon-labs/WF111-A-V1/1446-1031-2-ND/7593930>.

A Tabela 3 traz uma série de dados comparativos entre três chips de diferentes empresas, que possuem Wi-Fi integrado. O primeiro chip, o WF111-A-V1, é da empresa multinacional, Silicon Labs, o segundo chip, o ATWINC1510-MR210PB1961 é da empresa americana Microchip Technology e o terceiro chip, sendo o mais popular entre os três atualmente, o ESP32-WROOM-32D, é da também multinacional Espressif Systems.

Como visto a partir da Tabela 3, os três chips trabalham com os mesmos padrões IEEE 802.11, sendo eles o “b”, o “g” e o “n”. Os chips WF111-A-V1 e ATWINC1510-MR210PB1961 são muito parecidos, trabalhando com taxas de dados e potência de saída bem próxima, possuindo também poucas interfaces de comunicação. O chip ESP32-WROOM-32D possui uma taxa de dados duas vezes maior que os outros dois, possuindo também uma maior potência de saída e sensitividade. Além destas vantagens, o ESP32-WROOM-32D possui diversas interfaces de comunicação e é atualmente o mais barato entre os três chips, o que o torna, por fim, a escolha ideal para este projeto.

4.3 ESP 32

Segundo Espressif Systems (2021), o ESP 32 foi projetado para ser utilizado em aplicativos móveis, eletrônicos vestíveis e IoT e é um único chip de 2.4GHz, que conta com Wi-Fi e Bluetooth, projetado para melhor potência e desempenho em comunicação de rádio frequência (RF), buscando maior robustez para a maior variedade de aplicações.

Chip ATWINC1510-MR210PB1961 - Disponível em: <https://www.digikey.com.br/product-detail/pt/microchip-technology/ATWINC1510-MR210PB1961/ATWINC1510-MR210PB1961-ND/9657714>.

Chip ESP32-WROOM-32D - Disponível em: <https://www.digikey.com.br/product-detail/pt/espressif-systems/ESP32-WROOM-32D-4MB/1965-ESP32-WROOM-32D-4MB-DKR-ND/9381748>.

4.3.1 Especificações e Recursos

De acordo com Espressif Systems (2021), o ESP 32 possui diversas especificações e recursos, como:

a) wi-fi:
   - padrão IEEE 802.11 b/g/n, frequência de 2.4GHz e taxa de dados de até 150 Mbps;
   - quatro interfaces virtuais;
   - suporte simultâneo a modo *station* e *softAP*.

b) bluetooth:
   - bluetooths v4.2 e BLE, com controle de energia aprimorado;
   - altas velocidades (acima de 4Mbps) e multiconexões em *Classic BT* e BLE;

c) *cpu* e memória:
   - *single/dual-core* de 32-bits, 448 KB ROM e 520 KB SRAM;

d) *clocks* e *Timers*:
   - oscilador interno de 8 MHz e oscilador interno RC, ambos com calibração;
   - cristal oscilador externo de 2 MHz ~ 60 MHz (necessário 40 MHz para uso de Wi-Fi e/ou Bluetooth);
   - cristal oscilador externo de 32kHz para o RTC, com calibração;
   - dois grupos de *timers*, sendo dois de 64 bits e um *watchdog* principal em cada grupo e um RTC *timer*;

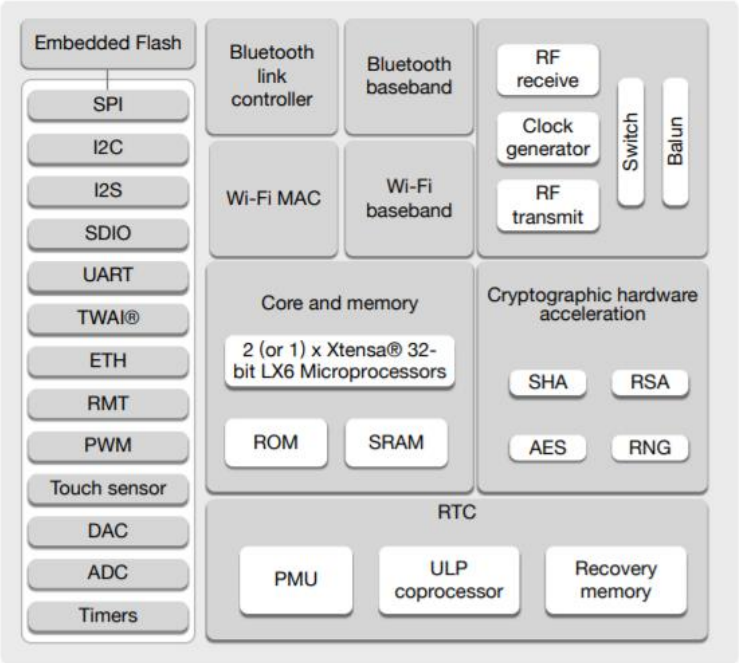
e) periféricos:
   - 34 GPIOs programáveis, contando com ADC de 12 bits com 18 canais; e dois DAC de 8 bits, contando também com 4 SPI, 2 I^2C, 3 UART;
   - interface MAC Ethernet com DMA dedicado e suporte IEEE 1588;

f) segurança:
   - *boot* seguro e criptografia de flash;

4.3.2 Diagrama de Blocos

A Figura 12 apresenta o diagrama de blocos interno do ESP 32.

Figura 12 - Diagrama de blocos internos do ESP 32



Fonte: *Datasheet* ESP 32[12].

A Figura 12 traz uma representação de um diagrama de blocos interno do ESP 32, onde se pode ver uma série de periféricos e as estruturas em blocos do Wi-Fi e Bluetooth, dos *timers*, CPUs e memórias, *hardwares* para aceleração de criptografia e circuito para comunicação RF. Espressif Systems (2021) afirma que as séries são diferentes umas das outras em termos de suporte a *flash* embarcada e número de CPUs.

4.4 Firmware

Principal área de atuação do projeto em questão, o *firmware* foi desenvolvido em linguagem de programação C e pode ser dividido em três partes principais:

[12] Disponível em <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em 22 de abril de 2021

comunicação do dispositivo com os controladores de temperatura, conexão Wi-Fi e comunicação MQTT com o sistema em nuvem.

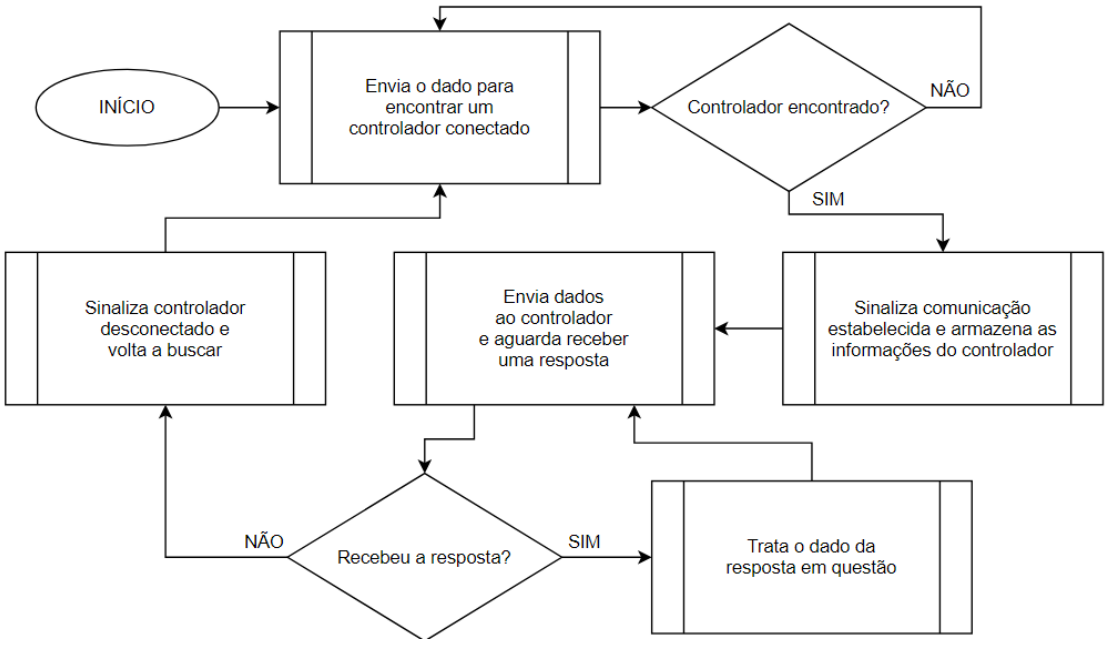
4.4.1 Comunicação com os controladores de temperatura

Os controladores de temperatura são os responsáveis pela coleta dos dados (e atuações) que serão enviados ao sistema em nuvem, podendo também ser configurados por meio do sistema. A comunicação com os controladores foi construída a partir do protocolo já implementado pelos próprios controladores, sendo basicamente uma comunicação serial (*modbus* / RS-485);

Por ser um protocolo de propriedade da empresa, será apresentado de forma sucinta, ou seja, apenas seu funcionamento, sem códigos explícitos ou descrição de como os dados são montados e trafegam entre os dispositivos.

A Figura 13 apresenta um fluxograma do algoritmo que implementa o processo de comunicação do dispositivo com os controladores.

Figura 13 - Fluxograma da comunicação do dispositivo com os controladores de temperatura



Fonte: Autor, 2021.

Como apresenta o fluxograma da Figura 13, o algoritmo inicia buscando um controlador para estabelecer a comunicação, enviando um dado e esperando o

recebimento de uma resposta (dados definidos pelo protocolo de propriedade da empresa). Ao receber a resposta do dado, o dispositivo assume que possui conexão estabelecida com este controlador e entra no processo de enviar dados e esperar receber a resposta, de acordo com o protocolo em questão, em intervalos de tempo constantes. Enquanto o dispositivo recebe dados do controlador, continua assumindo que a conexão está estabelecida e, em caso de não recebimento da resposta, o algoritmo sinaliza ao dispositivo que a conexão com o controlador foi perdida e o mesmo retorna para o passo de busca de um controlador conectado, permanecendo neste ponto até estabelecer outra conexão. A Figura 14, apresenta um pseudocódigo resumido, em linguagem C, que implementaria o fluxograma da Figura 13.

Figura 14 - Pseudo código de demonstração da comunicação dos controladores de temperatura com o dispositivo

```
void algoritmo_comunicacao_controladores(void) {
  while(1) {
    if(controlador_desconectado()) {
      bool controlador_encontrado = busca_controlador();
      if(controlador_encontrado) {
        sinaliza_controlador_conectado();
      }
    } else if(controlador_conectado()) {
      bool resposta_recebida = envia_dado_ao_controlador();
      if(resposta_recebida == false) {
        sinaliza_controlador_desconectado();
      }
    }
  }
}
```

Fonte: Autor, 2021.

O pseudocódigo apresentado na Figura 14 traz uma ideia de como se pode trazer o fluxograma da Figura 13 para dentro de um código em linguagem C, sem entrar nas implementações específicas do algoritmo e de suas funções. Seguindo a ideia do fluxograma, o código inicia verificando a conexão com um controlador e caso o encontre, entra no passo de comunicação, enviando dados e aguardando uma resposta. Assim como no fluxograma, caso não receba a resposta, retorna ao passo de busca e assim funciona indefinidamente, em um laço de repetição infinito.

4.4.2 Wi-Fi

A conexão Wi-Fi é a parte mais importante do projeto, pois é o meio físico que permite a integração do *hardware* com o sistema em nuvem.

O *firmware* para comunicação Wi-Fi foi construído a partir de uma biblioteca que implementa a conexão Wi-Fi, disponibilizada pela própria fabricante do ESP 32, a Espressif e suas principais funções podem ser vistas no arquivo “esp_wifi.h”[13]. A Figura 15 apresenta um pseudo código, em linguagem C, que mescla funções nativas da biblioteca utilizada, com funções genéricas, para apresentar um pouco de como foi implementada a parte do *firmware* que trata da conexão Wi-Fi.

Figura 15 - Pseudo código de demonstração do *firmware* responsável pela conexão Wi-Fi

```
void wifi(void) {
  // ------------------------------------------------- //
  // Inicialização
    esp_netif_init();
    esp_event_loop_create_default();
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    esp_wifi_init(&cfg);
  // ------------------------------------------------- //
  // Configurações
    wifi_config_t wifi_config = {
      .sta = {
        .ssid = "ssid_da_rede",
        .password = "senha_da_rede",
      }
    };
    esp_wifi_set_mode(WIFI_MODE_STA);
    esp_wifi_set_config(WIFI_IF_STA, &wifi_config);
    esp_wifi_start();
    esp_wifi_connect();
  // ------------------------------------------------- //
  // Loop de funcionamento
    while(1) {
      if(wifi_conectado()) {
        wifi_executa_tarefas();
      } else if(wifi_desconectado()) {
        wifi_reconecta();
      }
    }
  // ------------------------------------------------- //
}
```

Fonte: Autor, 2021.

[13] Disponível em <https://github.com/espressif/esp-idf/tree/master/components/esp_wifi/include>

Como apresenta a Figura 15, a implementação possui, basicamente, três partes principais, sendo elas: inicialização, configuração e *loop* de funcionamento.

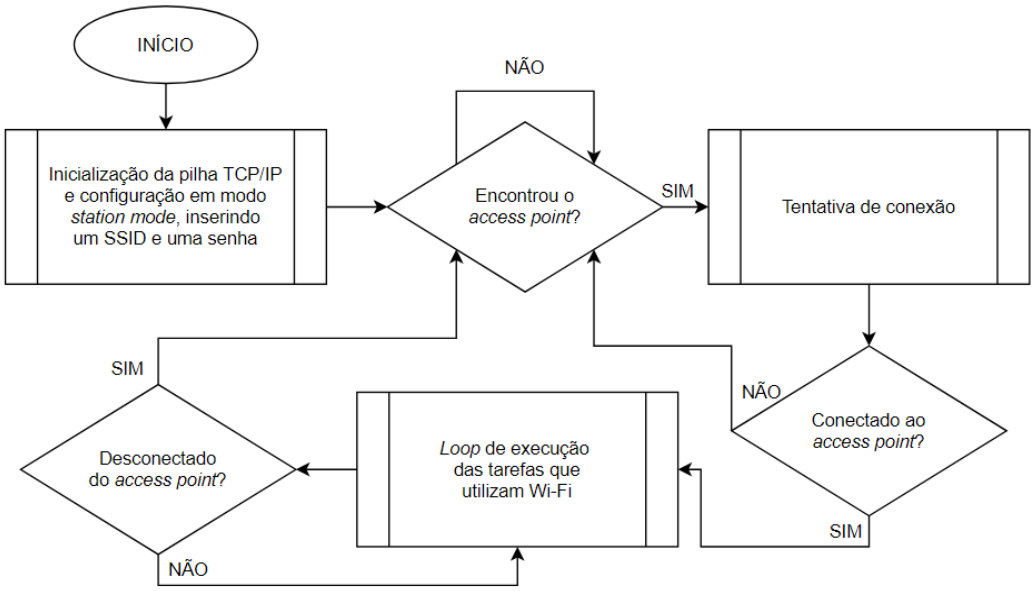
a) inicialização:
   - chamada de inicialização da pilha TCP/IP e algumas configurações gerais de funcionamento;

b) configurações:
   - inserção de um SSID e uma senha para iniciar o processo de conexão e configuração em modo de operação *station mode* na função “esp_wifi_set_mode(WIFI_MODE_STA)”, definindo que o Wi-Fi irá buscar o *access point* configurado, dentro da linha da alcance (pode ser um modem, um repetidor ou até mesmo outro ESP 32) e após encontrar a rede, fará a inserção da senha e a realizará a tentativa de conexão para estabelecer a infraestrutura da comunicação.

c) *loop* de funcionamento:
   - passo final e de infinita permanência, onde o *firmware* se encarregará de monitorar e gerenciar conexões, desconexões e reconexões, e tratar todas as tarefas pertinentes à utilização do Wi-Fi e suas aplicações, como no caso mais específico, o protocolo MQTT.

A Figura 16 apresenta um fluxograma da implementação da parte do *firmware* que implementa a conexão Wi-Fi.

Figura 16 - Fluxograma de demonstração do *firmware* responsável pela conexão Wi-Fi



Fonte: Autor, 2021.

O fluxograma apresentado na Figura 16 demonstra o funcionamento da parte do *firmware* responsável pela implementação da conexão Wi-Fi, a qual foi ilustrada a partir do pseudocódigo da Figura 15. O fluxograma em si, assim como o pseudocódigo, não se aprofunda nas partes mais específicas da implementação, pois busca dar enfoque a como o algoritmo irá executar suas tarefas.

Neste caso, o fluxograma apresenta os processos de inicialização e configuração, que são cruciais para o funcionamento correto da conexão com o *access point* no qual se deseja conectar, inicializando a pilha TCP/IP, responsável pelo tráfego dos dados e configurando o SSID e senha do *access point*, para que se possa estabelecer uma conexão com o mesmo.

Apresenta também o processo que parte do ponto da busca pelo *access point* configurado até a tentativa de conexão com o mesmo, que, caso seja estabelecida, dá início ao último e permanente processo, o *loop* de funcionamento, que é responsável por permitir que outras partes do *firmware*, que necessitam de conexão Wi-Fi, funcionem corretamente, como por exemplo o protocolo de comunicação MQTT. O *loop* também verifica se houve alguma desconexão do *access point* em questão, para que o algoritmo possa reiniciar o processo e buscar novamente a conexão.

4.4.3 MQTT

O protocolo de comunicação MQTT é o ponto que interliga toda a aplicação, visto que a ideia central do projeto é estabelecer uma comunicação entre os controladores de temperatura e o sistema de monitoramento e controle em nuvem. A partir do protocolo MQTT, construiu-se todos os passos e dados necessários para que essa comunicação fosse possível.

Utilizou-se uma biblioteca com a implementação do MQTT da própria Espressif (cujas funções podem ser visualizadas no arquivo "esp_mqtt_client.h"[14]) como base para a construção do *firmware*, que tem como objetivo final, implementar o protocolo definido pela própria empresa, possibilitando a comunicação com o sistema em nuvem.

A implementação possui duas partes principais, sendo elas a inicialização e configuração e o *loop* de funcionamento. A Figura 17 apresenta um pseudo código para ilustração de como está implementada a parte do *firmware* responsável pela comunicação e gerenciamento do protocolo MQTT. O pseudo código em questão mescla funções nativas da biblioteca utilizada com funções genéricas que representam implementações reais.

[14] Disponível em <https://github.com/espressif/esp-mqtt/tree/9fdf7b61385633075d5c3b84803f2dd0578d7869/include>

Figura 17 - Pseudo código de demonstração do *firmware* responsável pela comunicação MQTT

```
void mqtt(void) {
  // ------------------------------------------------- //
  // Inicialização e configuração
    esp_mqtt_client_handle_t mqttClient;
    esp_mqtt_client_config_t mqtt_cfg = {
      .uri = "mqtt://url_broker:1883",
      .client_id = "clientID",
      .username = "usuario",
      .password = "senha",
    }
    mqttClient = esp_mqtt_client_init(&mqtt_cfg);
  // ------------------------------------------------- //
  // Loop de funcionamento
    while(1) {
      while(!mqtt_client_conectado_ao_broker());
      char *topico_subscribe = "/dado/receber";
      mqtt_client_subscribe(topico_subscribe, QOS_0);

      while(mqtt_client_conectado_ao_broker()) {
        char *dado = mqtt_client_carrega_dado();
        char *topico_publish = "/dado/enviar";
        mqtt_client_publish(topico_publish, dado, QOS_0);
        if(mqtt_client_recebeu_dados()) {
          mqtt_client_trata_dados_recebidos();
          mqtt_client_responde();
        }
      }
    }
  // ------------------------------------------------- //
}
```

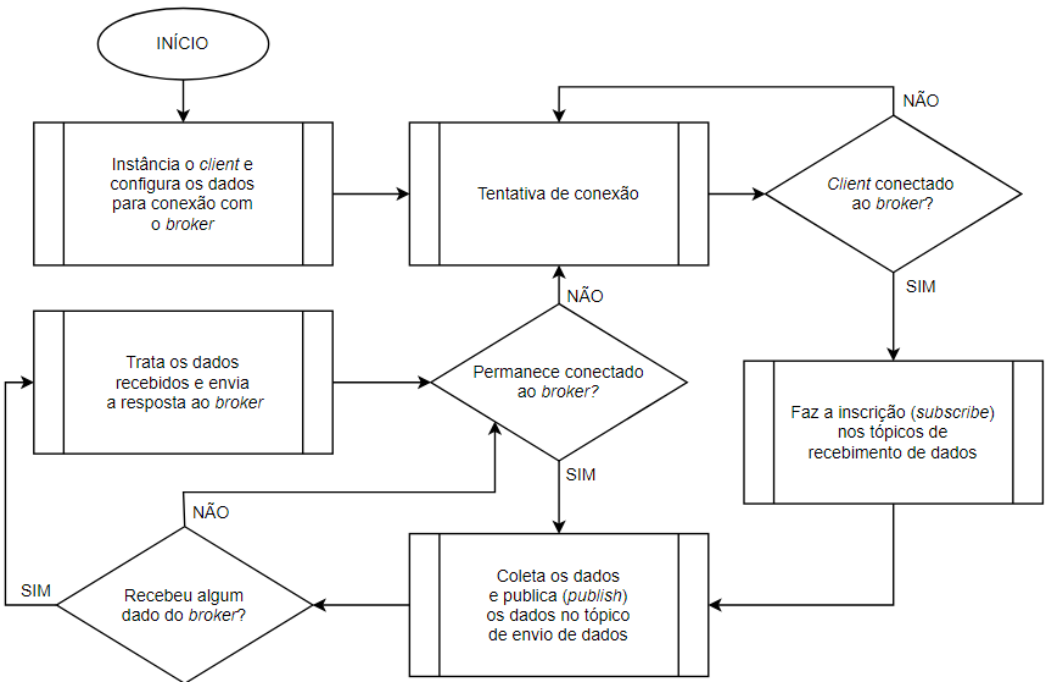
Fonte: Autor, 2021.

O pseudocódigo da Figura 17, traz uma representação bem próxima a real de como o *firmware* foi implementado. Partindo do ponto responsável pelas inicializações e configurações, há o instanciamento do *client* MQTT e suas configurações, onde há a inserção da url do *broker* no qual se conectará, inserção do ID do *client*, que será a identificação do *client* junto ao *broker* (podendo ser qualquer tipo de dado, desde números a letras ou nomes) e a configuração de um usuário e senha para o *login*, quando necessário, como no caso em questão, pois o *broker* autentica o usuário e senha do *client* a partir de dados armazenados em um banco de dados. Caso o *client* fosse implementado utilizando a comunicação criptografada por meio de um TLS, por exemplo, sobre o protocolo TCP/IP, essa autenticação com usuário e senha não seria necessariamente necessária, pois os certificados e chaves já seriam os dados utilizados para autenticação.

O *loop* de funcionamento implementa todo o fluxo de tratativas de conexão e desconexão junto ao *broker* e a execução das tarefas que necessitam de comunicação MQTT para funcionamento, como a maior finalidade do projeto em questão, que é a comunicação com o sistema em nuvem, enviando e recebendo dados relativos ao controlador de temperatura conectado ao dispositivo. No exemplo do pseudo código, o *loop* de funcionamento trata diretamente o processo de conexão com o *broker*, aguardando a conexão ser estabelecida para se inscrever no tópico de recebimento de dados, por meio da função genérica “mqtt_client_subscribe”. Após estabelecer a conexão e se inscrever no tópico de recebimento de dados, entra em um *loop*, mantido pela conexão com o *broker* que de tempos em tempos (por padrão três segundos) envia os dados coletados por meio da função genérica “mqtt_client_publish” e também verifica se recebeu algum dado no tópico no qual se inscreveu anteriormente e caso tenha recebido, trata esses dados de acordo com um protocolo específico e envia uma resposta ao *broker*.

A Figura 18 traz uma representação do fluxo de funcionamento da implementação do MQTT, por meio de um fluxograma.

Figura 18 - Fluxograma de demonstração do *firmware* responsável pela comunicação MQTT



Fonte: Autor, 2021.

A Figura 18 traz a representação do funcionamento do *firmware* em forma de um fluxograma. Assim como apresentado no pseudocódigo da Figura 17, o *firmware* possui duas partes bem definidas, onde a primeira trata do processo de inicialização e configuração, que permite a inserção dos dados e configurações necessárias para acesso à conexão junto ao *broker* e a segunda parte trata do processo do *loop* de funcionamento, com várias tarefas. O algoritmo inicia pela tentativa de conexão e, caso a conexão seja estabelecida, há a inscrição nos tópicos de recebimento de dados (*subscribe*). Após a inscrição nos tópicos de recebimento de dados, o algoritmo entra em um processo permanente de coleta e publicação/envio de dados (*publish*) e verificação de recebimento de dados. Caso receba algum dado, os mesmos são tratados de acordo com o protocolo definido e há o envio de uma resposta, também gerada e enviada com base no protocolo. Há também a verificação do status da conexão com o *broker*, que, em caso de queda, quebra o estado permanente atual e automaticamente reinicia o processo, buscando novamente uma conexão. Como não foram encontrados pontos críticos de comunicação ou dados que não poderiam ser perdidos de forma alguma, os *publishs* e os *subscribes* realizados utilizaram QoS 0, utilizando o nível mais simples e leve de qualidade de serviço.

4.4.4 Testes de Comunicação

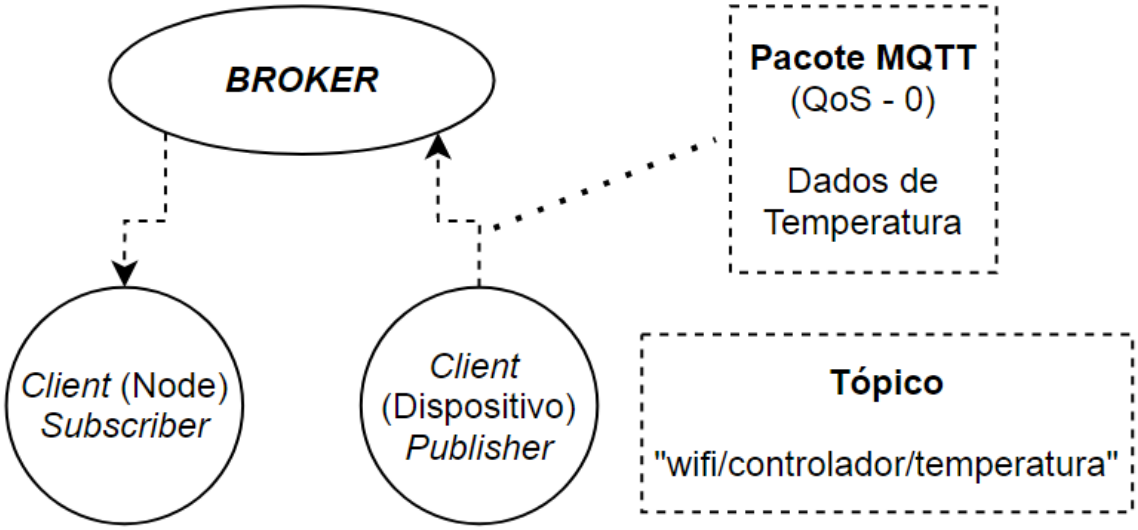
Após o desenvolvimento das três principais partes do *firmware* do dispositivo, sendo elas a comunicação com os controladores de temperatura, conexão Wi-Fi e protocolo de comunicação MQTT, o sistema foi integrado e submetido ao seu estado de funcionamento real.

Antes do processo final de integração com o sistema em nuvem, buscando validar a comunicação, foram utilizados um *broker* local e dois *clients*, sendo o primeiro *client* funcionando apenas como *subscriber*, para simular o recebimento de dados de temperatura e o segundo funcionando como *subscriber* e *publisher*, para simular o processo de envio e recebimento de dados por parte do sistema em nuvem ao dispositivo. O *broker* e os dois *clientes* foram desenvolvidos com a linguagem de programação Javascript e executados por meio do ambiente Node.js, para verificação de sucesso no envio e recebimento dos dados. O Node.js é um ambiente

de execução para o Javascript, baseado em eventos assíncronos projetado para construir aplicativos de rede escalonáveis (NODEJS.ORG, 2020).

A Figura 19 apresenta um diagrama que descreve o funcionamento do primeiro teste realizado.

Figura 19 - Formato do primeiro teste, com o *client* recebendo dados de temperatura do dispositivo



Fonte: Autor, 2021.

Como mostra a Figura 19, o primeiro *client* de testes funciona apenas como *subscriber*, com o objetivo de receber os dados de temperatura enviados pelo dispositivo, passando sempre pelo *broker*, a partir do tópico “wifi/controlador/temperatura”, tópico este no qual o dispositivo publica os dados, sendo que todos os dados trafegam com QoS 0, por não haver a necessidade de garantir a entrega de 100% dos dados enviados.

A Figura 20 apresenta o código desenvolvido para o primeiro *client* de testes.

Figura 20 - Código do *client subscriber* para teste de comunicação com o dispositivo

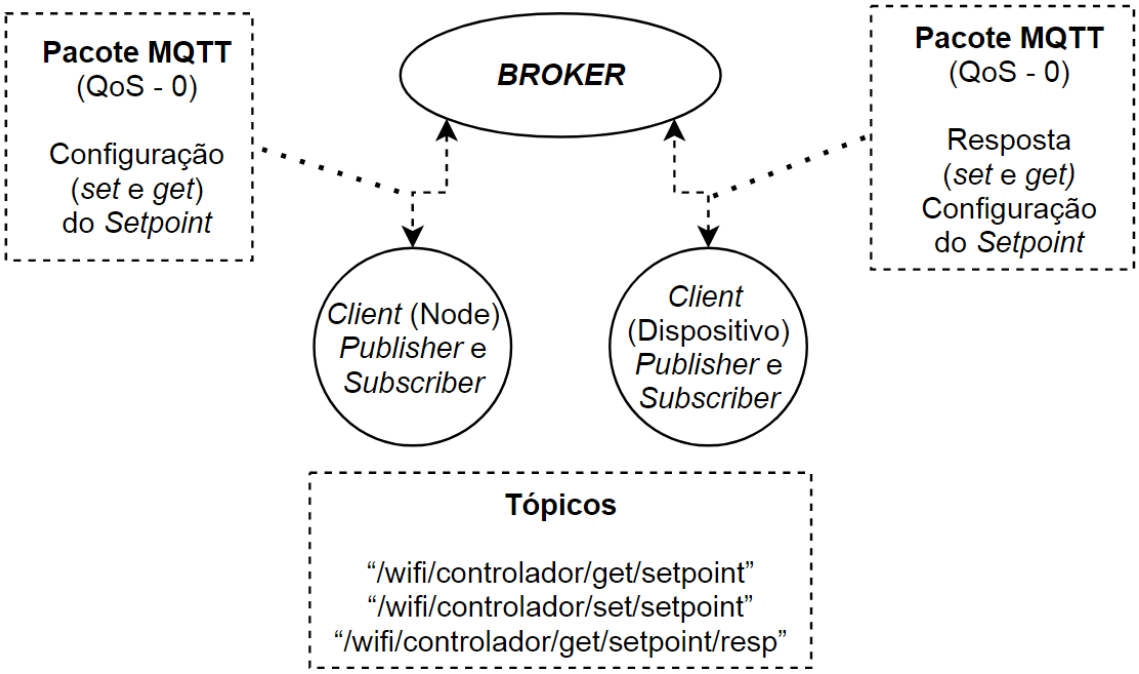
```
// Instância da biblioteca
const mqtt = require('mqtt');
// Configurações do client
const configClient = {
  clean: false,
  keepalive: 300, // segundos
  reconnectPeriod: 500, // milissegundos
  clientId: 'client_1_subscriber',
};
// Configurações de conexão com o broker local
const client = mqtt.connect('mqtt://localhost:1883', configClient);
// Evento assíncrono de conexão
client.on('connect', function () {
  console.log(`\n ## A conexão com o broker foi estabelecida! `);
  client.subscribe(`/wifi/controlador/temperatura/`, {qos: 0});
});
// Evento assíncrono de desconexão
client.on('close', function () {
  console.log(`\n ## A conexão com o broker foi fechada! `);
});
// Evento assíncrono de recebimento de mensagem
client.on('message', function (topico, mensagem) {
  console.log(`\n ## Nova mensagem! \n ## Tópico: (${topico}) \n ## Dado: ${mensagem}`);
});
```

Fonte: Autor, 2021.

O código apresentado na Figura 20 possui dois blocos, onde o primeiro bloco é responsável por instanciar a biblioteca e configurar o *client* para que o mesmo consiga estabelecer a conexão com o *broker*. O segundo bloco conta com os eventos assíncronos de conexão, desconexão e recebimento de mensagens, onde se podem implementar as tratativas de acordo com o que se deseja, como por exemplo, no evento de conexão, foi feita a inscrição no tópico de recebimento de dados “/wifi/controlador/temperatura/”.

A Figura 21 apresenta o diagrama que descreve o funcionamento do segundo teste realizado.

Figura 21 - Formato do segundo teste, com o *client* configurando e lendo dados do dispositivo



Fonte: Autor, 2021.

Como mostra a Figura 21, o segundo *client* de testes funciona como *subscriber* e *publisher*, com o objetivo de enviar um dado de configuração ao dispositivo e verificar se o mesmo foi alterado com sucesso, lendo posteriormente esse mesmo dado. O tópico de configuração utilizado foi "/wifi/controlador/set/setpoint" e o tópico de verificação de configuração foi "/wifi/controlador/get/setpoint". O *client* ainda se inscreve no tópico "/wifi/controlador/get/setpoint/resp", tópico este no qual o dispositivo envia a resposta da requisição do *client*. Todos os dados trafegam com QoS 0 por não haver a necessidade de garantir a entrega de 100% dos dados enviados.

A Figura 22 apresenta o código do segundo *client* de testes.

Figura 22 - Código do *client subscriber* e *publisher* para teste de comunicação com o dispositivo

```
// Instância da biblioteca
const mqtt = require('mqtt');
// Configurações do client
const configClient = {
  clean: false,
  keepalive: 300, // segundos
  reconnectPeriod: 500, // milissegundos
  clientId: 'client_2_subscriber_e_publisher',
};
// Configurações de conexão com o broker local
const client = mqtt.connect('mqtt://localhost:1883', configClient);
// Evento assíncrono de conexão
client.on('connect', function () {
  console.log(`\n ## A conexão com o broker foi estabelecida! `);
  client.subscribe('/wifi/controlador/get/setpoint/resp', {qos: 0});
  client.publish(`/wifi/controlador/get/setpoint`, JSON.stringify({}));
  setTimeout(() => {
    client.publish(`/wifi/controlador/set/setpoint`, JSON.stringify({setpoint: 30}));
  }, 500); // delay de 500 milissegundos
  setTimeout(() => {
    client.publish(`/wifi/controlador/get/setpoint`, JSON.stringify({}));
  }, 1000); // delay de 1 segundo
});
// Evento assíncrono de desconexão
client.on('close', function () {
  console.log(`\n ## A conexão com o broker foi fechada! `);
});
// Evento assíncrono de recebimento de mensagem
client.on('message', function (topico, mensagem) {
  console.log(`\n ## Nova mensagem! \n ## Tópico: (${topico}) \n ## Dado: ${mensagem}`);
});
```

Fonte: Autor, 2021.

Para a construção do código apresentado na Figura 22, manteve-se a base utilizada no primeiro *client*, alterando apenas as tratativas e tarefas implementadas no evento assíncrono de conexão. A função assíncrona de conexão, a qual implementa as tarefas principais do *client* em questão, inicia pela inscrição no tópico “/wifi/controlador/get/setpoint/resp”, que busca receber dados de resposta de configuração de um parâmetro do controlador de temperatura chamado *setpoint*. Posteriormente, publica um dado vazio no tópico “/wifi/controlador/get/setpoint”, buscando uma resposta do dispositivo, no tópico “/wifi/controlador/get/setpoint/resp”, sobre o valor atual do *setpoint* do controlador conectado ao mesmo. Após aguardar 500 milissegundos, publica um dado de configuração de *setpoint* no tópico “/wifi/controlador/set/setpoint”, tópico este no qual o dispositivo deve estar inscrito e

pelo qual o mesmo poderá extrair os dados de configuração do *setpoint*. Após enviar o dado de configuração, o *client* aguarda um segundo e pede novamente o valor do *setpoint* do dispositivo, no tópico “/wifi/controlador/get/setpoint”, para validar se a configuração ocorreu corretamente como o esperado.

A Figura 23 apresenta o código do *broker* utilizado para a comunicação entre os dispositivos e os *clients.*

Figura 23 - Código do *broker* local para comunicação dos *clients* com o dispositivo

```
// Instância da biblioteca
const mosca = require('mosca');
// Configurações do broker
const configBroker = { port: 1883 };
// Instância e configuração final do broker
const mqttBroker = new mosca.Server(configBroker);
// Evento assíncrono de start do broker
mqttBroker.on('ready', function() {
  console.log(`\n ## Broker rodando na porta ${configBroker.port} \n`);
});
// Evento assíncrono de conexão de um client
mqttBroker.on('clientConnected', function(client) {
  console.log(` ## Client ${client.id} conectado! \n`);
});
// Evento assíncrono de desconexão de um client
mqttBroker.on('clientDisconnected', function(client) {
  console.log(` ## Client ${client.id} desconectado! \n`);
});
```

Fonte: Autor, 2021.

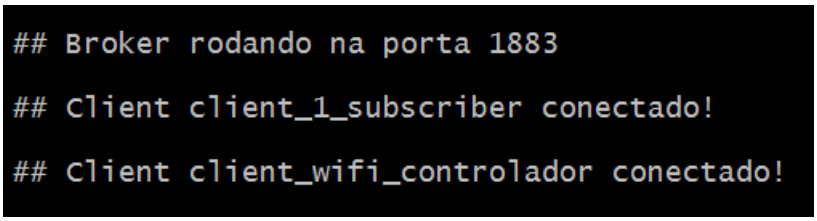
Como apresenta a Figura 23, assim como nos códigos dos *clients*, o código desenvolvido para o *broker* pode ser dividido didaticamente em dois blocos, sendo o primeiro bloco responsável pela chamada da biblioteca, configurações e instanciamento do *broker* e o segundo bloco sendo responsável pelos eventos assíncronos de *start* do próprio *broker*, conexão e desconexão dos *clients*. Para os testes, o *broker* foi executado localmente na porta 1883.

5 RESULTADOS

Após todas as etapas de desenvolvimento finalizadas, o sistema foi integrado e testado, possibilitando a verificação do funcionamento e extração dos resultados.

Com o *broker* e os dois *clients* de testes de comunicação prontos para a validação, foram executados o *broker* e o primeiro *client*, o qual funciona como *subscriber*, e assim a comunicação entre eles foi estabelecida. Então o dispositivo foi energizado e conectado a um controlador de temperatura e a uma rede Wi-Fi. Após conectar-se a rede, o dispositivo estabeleceu a conexão com o *broker* assim como mostram os *logs* presentes na Figura 24.

Figura 24 - *Logs* do *broker*

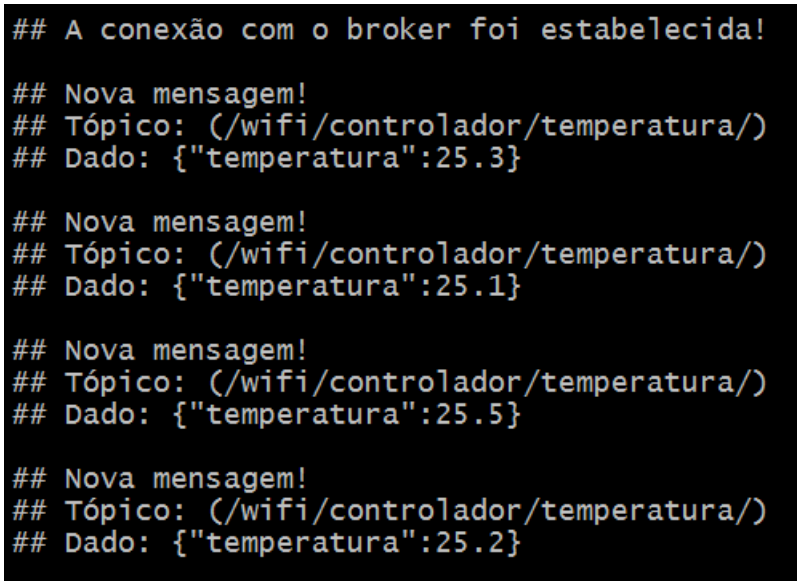


Fonte: Autor, 2021.

A Figura 24 traz os *logs* de conexão do *client*, como *id* “cliente_1_subscriber” e do dispositivo com *id* “cliente_wifi_controlador” com o *broker*, validando assim o processo de conexão entre eles.

Conectado ao *broker*, o dispositivo iniciou o envio dos dados de temperatura coletados a partir do controlador no tópico “/wifi/controlador/temperatura/”, tópico este no qual o *client* também se inscreveu e começou a receber os dados, como mostram os *logs* do *client* na Figura 25.

Figura 25 - *Logs* dos dados enviados pelo dispositivo ao *broker* e recebidos pelo *client*

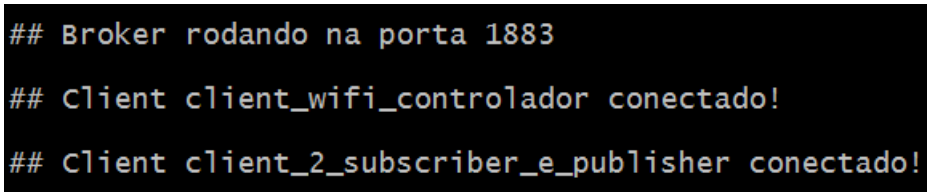


Fonte: Autor, 2021.

A Figura 25 mostra que a comunicação entre o dispositivo e o *client*, por meio do *broker*, ocorreu com sucesso, pois o *client*, assim como era esperado, recebeu os dados de temperatura enviados pelo dispositivo, no tópico “/wifi/controlador/temperatura/”.

Após a validação dos testes com o primeiro *client*, o segundo *client* de testes foi executado, visando validar o envio de dados ao dispositivo. O processo se repetiu como no primeiro *client*, onde inicialmente se executou o *broker* e o *client* e a conexão entre eles foi estabelecida. O dispositivo foi energizado, conectado a um controlador de temperatura, a uma rede Wi-Fi e finalmente ao *broker*. A Figura 26 traz os *logs* do *broker*.

Figura 26 - *Logs* do *broker*

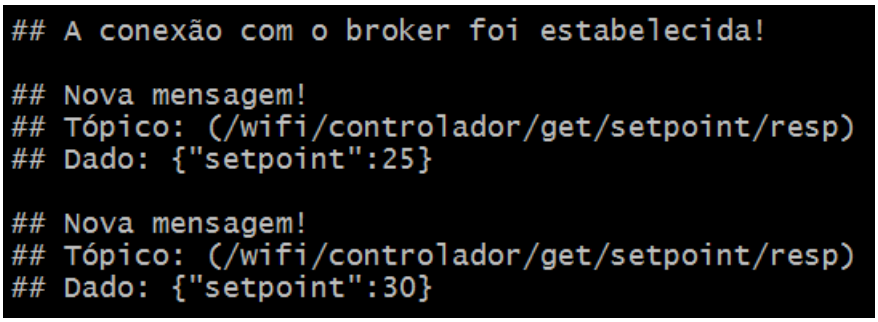


Fonte: Autor, 2021.

A Figura 26, a qual apresenta os *logs* do *broker*, confirma que a conexão entre o *cliente* com *id* “cliente_2_subscriber_e_publisher” e o dispositivo com *id* “cliente_wifi_controlador” foi estabelecida com sucesso.

Com o *client* e o dispositivo conectados ao *broker*, o *client* executou suas tarefas e conseguiu, com sucesso, configurar e receber as respostas do dispositivo, assim como mostra a Figura 27, que traz os *logs* do *client*.

Figura 27 - *Logs* do *client*



Fonte: Autor, 2021.

A Figura 27 confirma que o *client* conseguiu, com sucesso, verificar o valor do *setpoint* do dispositivo e configurá-lo corretamente. A Figura 28 finaliza a confirmação do processo de teste, apresentando os *logs* do dispositivo.

Figura 28 - *Logs* do dispositivo

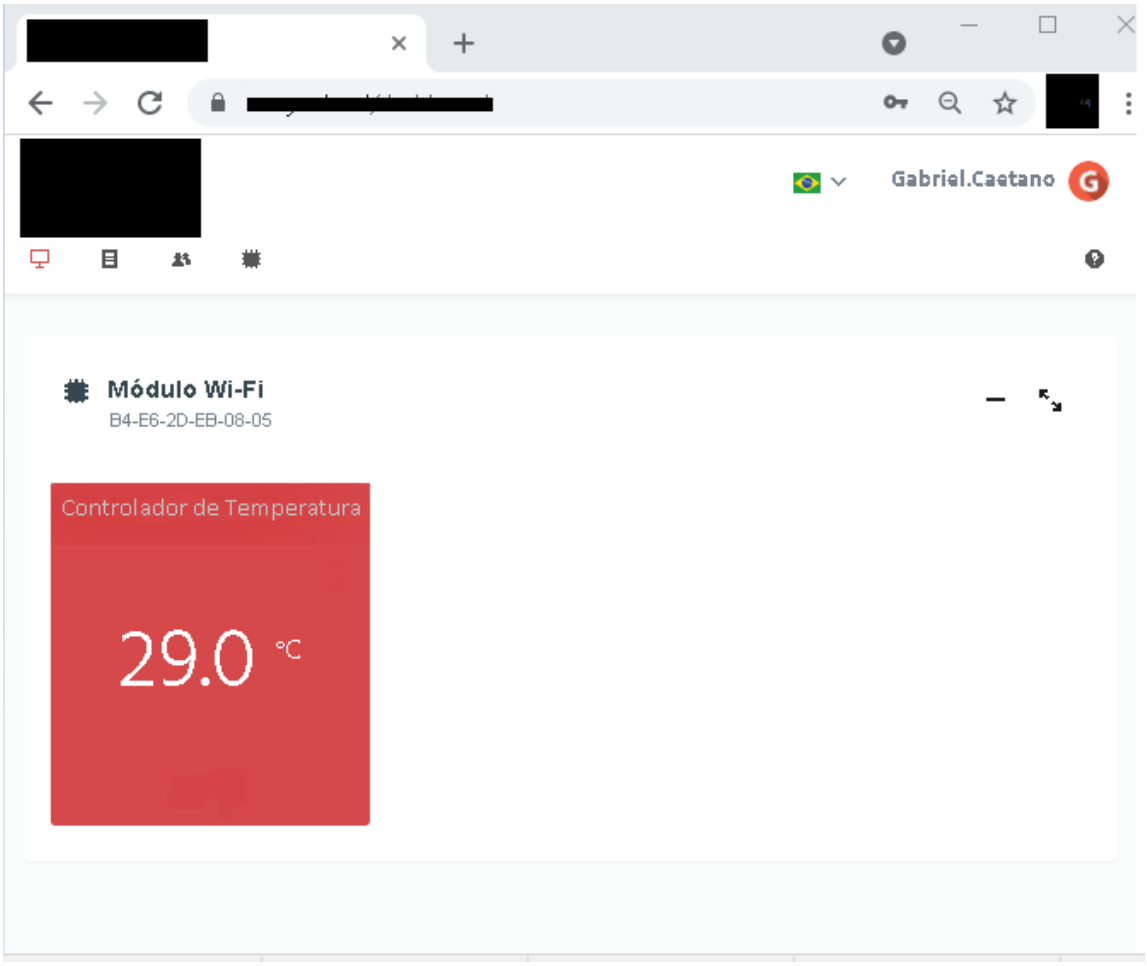
Fonte: Autor, 2021.

A Figura 28, que traz os *logs* do dispositivo, apresenta a execução completa do teste, onde o dispositivo recebe a verificação do valor do *setpoint*, pelo *client*, no tópico “wifi/controlador/get/setpoint” e responde, no tópico “wifi/controlador/get/setpoint/resp”, com o valor 25, que era o valor atual no momento da verificação. Posteriormente o dispositivo receber do *client* a configuração do *setpoint*, com o valor 30, no tópico “wifi/controlador/set/setpoint” e realiza a configuração do *setpoint* a partir de um processo interno. Após a configuração, o *client* solicita novamente o valor do *setpoint*, confirmando o novo valor de 30, como o mesmo havia configurado anteriormente, validando assim os testes de envio e recebimento de dados.

Com isso, pode-se perceber que a conexão Wi-Fi e o protocolo MQTT foram implementados corretamente, possibilitando que o dispositivo fosse conectado à rede e ao *broker*, permitindo o envio e recebimento de dados ao mesmo. Percebe-se também, como premissa do sistema, que a comunicação com o controlador de temperatura foi estabelecida de forma correta e os dados de temperatura foram extraídos com sucesso, sendo enviados, em graus Celsius, no tópico “/wifi/controlador/temperatura” ao *broker*, possibilitando o recebimento desses mesmos dados por meio do *client* de testes funcionando como *subscriber*, construído em Node.js, conectado ao mesmo *broker* e inscrito no mesmo tópico onde os dados foram enviados. Também constatou-se que o recebimento de dados por meio do dispositivo funcionou corretamente, a partir dos testes de configuração, realizados com o *client* de testes, funcionando como *publisher* e *subscriber*.

Após os testes realizados, o dispositivo foi finalmente ligado ao sistema em nuvem e a Figura 29 apresenta o objetivo final da implementação, no qual os dados de temperatura ficam disponíveis em uma interface gráfica em nuvem.

Figura 29 - Amostragem em tempo real da temperatura no *front end* do sistema em nuvem



Fonte - Autor, 2021.

Como se pode notar, a partir da Figura 29, o sistema foi integrado corretamente e seu objetivo final de funcionamento foi validado, onde o *subscriber* responsável por integrar o *front end* da aplicação com o *broker* recebeu os dados enviados e com isso permitiu que o mesmo *front end* pudesse apresentar, em tempo real, os dados de temperatura coletados pelo controlador conectado ao dispositivo.

6 CONCLUSÃO

Neste trabalho foi desenvolvido um *firmware*, embarcado em um microcontrolador ESP 32, para um dispositivo comercial de propriedade intelectual de uma empresa da Grande Florianópolis, em Santa Catarina, que fabrica e desenvolve controladores de temperatura. O código desenvolvido realiza a integração dos controladores de temperatura dessa empresa, utilizando conexão sem fio Wi-Fi e protocolo de comunicação MQTT, com um sistema em nuvem, inserindo assim o conceito de IoT a estas aplicações.

Para a construção deste *firmware*, fez-se uma visualização completa do sistema para uma melhor compreensão e definição dos componentes presentes no mesmo, permitindo assim a sua divisão em algumas partes, facilitando o desenvolvimento. O *firmware* foi desenvolvido iniciando-se pela implementação da comunicação com os controladores de temperatura, passando pela análise e implementação da conexão sem fio Wi-Fi e finalizando na implementação do protocolo MQTT, realizando assim a integração do sistema e os testes de comunicação com o sistema em nuvem.

Dados os testes e a integração final com o sistema em nuvem, pode-se afirmar que o *firmware* atendeu as especificações definidas, pois o mesmo foi capaz de extrair os dados de temperatura dos controladores, conectar-se a uma rede Wi-Fi e estabelecer a comunicação com o *broker*, enviando assim os dados extraídos dos controladores de temperatura, ao sistema em nuvem, como esperado.

Utilizar conexão sem fio Wi-Fi, por mais consolidada que seja essa tecnologia, é um grande desafio, pois os sistemas que a utilizam precisam estar preparados para os cenários mais críticos, onde nem sempre o sinal disponível será satisfatório, podendo afetar assim as outras partes do sistema que utilizam o Wi-Fi para a transmissão de seus dados, como por exemplo, neste trabalho, o protocolo MQTT, o qual realiza a comunicação com o sistema em nuvem.

Por se tratar de um sistema que se comunica utilizando o ar como meio físico, uma melhoria futura seria a implementação de criptografia TLS junto ao protocolo de comunicação MQTT, garantindo assim uma maior robustez e segurança às informações por ele transmitidas. Outro trabalho futuro seria a possibilidade de

desenvolvimento e/ou de integração com algum aplicativo *mobile* para realizar as configurações do dispositivo, como as redes Wi-Fi, por exemplo.

Do exposto, pode-se concluir que a implementação do *firmware* foi realizada e os objetivos definidos foram alcançados. Estar atualizado aos novos conceitos e tecnologias vigentes é de suma importância, principalmente para evolução do conhecimento e aumento de competitividade dentro do mercado. Tecnologias tão consolidadas como conexão sem fio Wi-Fi e protocolo de comunicação MQTT ainda estão em constante evolução, ou seja, estudá-las e aplicá-las possibilita a ampliação do portfólio e gera experiências importantes para o futuro onde a IoT estará cada vez mais presente.

REFERÊNCIAS BIBLIOGRÁFICAS

ABES SOFTWARE (Brasil). **Mercado Brasileiro de Software: panorama e tendências, 2020**. 2020. Disponível em: https://abessoftware.com.br/wp-content/uploads/2020/10/ABES-EstudoMercadoBrasileirodeSoftware2020.pdf. Acesso em: 15 de abril 2021.

ABOUT Node.js. 2020. Disponível em: https://nodejs.org/en/about/#about-node-js. Acesso em: 03 de abril de 2021.

BRASIL. MINISTÉRIO DA SAÚDE. **MANUAL DE REDE DE FRIO**. 2013. Disponível em: http://bvsms.saude.gov.br/bvs/publicacoes/manual_rede_frio4ed.pdf. Acesso em: 11 de abril de 2021.

EBRADI. **Ad Hoc, Adoqui ou Ade Roqui?** 2019. Disponível em: https://www.ebradi.com.br/coluna-ebradi/adhoc-adoqui-aderoqui/. Acesso em: 21 de abril de 2021.

ELKHODR, Mahmoud; SHAHRESTANI, Seyed; CHEUNG, Hon. EMERGING **WIRELESS TECHNOLOGIES IN THE INTERNET OF THINGS: A COMPARATIVE STUDY**. 2016. Disponível em: https://arxiv.org/ftp/arxiv/papers/1611/1611.00861.pdf. Acesso em: 17 de abril de 2021.

ESPRESSIF SYSTEMS. **ESP32 Series**: datasheet. Datasheet. 2021. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acesso em: 22 de abril 2021.

EVANS, Dave. Cisco. **A Internet das Coisas**: como a próxima evolução da internet está mudando tudo. Como a próxima evolução da Internet está mudando tudo. 2011. Disponível em: https://www.cisco.com/c/dam/global/pt_br/assets/executives/pdf/internet_of_things_iot_ibsg_0411final.pdf. Acesso em: 15 de abril de 2021.

GARTNER. **Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020**. 2019. Disponível em: https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io. Acesso em: 10 de abril de 2021.

GAST, Matthew. **802.11® Wireless Networks: The Definitive Guide**. Gravenstein Highway North Sebastopol, Canadá: O'Reilly, 2002. 464 p. Disponível em: https://paginas.fe.up.pt/~ee05005/tese/arquivos/ieee80211.pdf. Acesso em: 19 de abril de 2021.

GERBER, Anna; ROMEO, Jim. **Key concepts and skills for getting started in IoT**. 2017. Disponível em: https://developer.ibm.com/articles/iot-key-concepts-skills-get-started-iot/. Acesso em: 10 de abril de 2021.

GERBER, Anna; ROMEO, Jim. **Streamlining the development of your IoT applications by using an IoT platform**. 2017. Disponível em: https://developer.ibm.com/articles/iot-lp101-why-use-iot-platform/. Acesso em: 17 de abril de 2021.

GREY, Joe. **Wifi security issues and solutions**. 2020. Disponível em: https://cybersecurity.att.com/blogs/security-essentials/security-issues-of-wifi-how-it-works. Acesso em: 21 de abril de 2021.

JUNIPER NETWORKS. **Network Director User Guide**. 3.2 Sunnyvale, California, USA: Juniper Networks, Inc., 2020. 1462 p. Disponível em: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director3.2/information-products/pathway-pages/network-director-pwp.pdf. Acesso em: 21 de abril de 2021.

OASIS. **MQTT Version 3.1.1**. 2014. Disponível em: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. Acesso em: 06 de abril de 2021.

PARZIALE, Lydia et al. **TCP/IP Tutorial and Technical Overview**. 2006. Disponível em: https://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg243376.pdf. Acesso em: 18 de abril de 2021.

LASSO, Paulo Renato Orlandi et al. **Sistema de Controle de Temperatura para Câmara de Armazenamento de Alimentos Frescos em Meio Líquido**. 2003. Disponível em: https://ainfo.cnptia.embrapa.br/digital/bitstream/CNPDIA/8730/1/BPD04_2003.pdf. Acesso em: 11 de abril de 2021.

SCHRAGL, Rudolf. **Fundamental Aspects for the Definition of Protocols**. Berlin, Heidelberg: Springer, Berlin, Heidelberg, 1983. 354 p. Disponível em: https://doi.org/10.1007/978-3-642-68829-4_16. Acesso em: 18 de abril de 2021.

THE HIVEMQ TEAM. **Quality of Service 0,1 & 2 - MQTT Essentials: Part 6**. 2015. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/. Acesso em: 18 de abril de 2021.

VIOLA, Teresa Herr et al. **Considerações técnicas sobre a incubação de ovos de galinhas**. 2019. Disponível em: https://www.infoteca.cnptia.embrapa.br/infoteca/bitstream/doc/1117323/1/Doc261AINFO0412201922.pdf. Acesso em: 11 de abril de 2021.

WI-FI ALLIANCE. **Who-we-are**. 2021. Disponível em: https://www.wi-fi.org/who-we-are. Acesso em: 21 de abril de 2021.

YUAN, Michael. **Getting to know MQTT**: why mqtt is one of the best network protocols for the internet of things. Why MQTT is one of the best network protocols for the Internet of Things. 2017. Disponível em: https://developer.ibm.com/technologies/iot/articles/iot-mqtt-why-good-for-iot/. Acesso em: 17 de abril de 2021.