UNIVERSIDADE FEDERAL DO PAMPA

MAGNO COSTA MAIA

DESENVOLVIMENTO DE UM FRAMEWORK PARA AUTOMATIZAR OS PROCESSOS DE CONSTRUÇÃO E GERENCIAMENTO DE REDES MESH EM AMBIENTES IOT

Alegrete
2022

MAGNO COSTA MAIA

DESENVOLVIMENTO DE UM FRAMEWORK PARA AUTOMATIZAR OS PROCESSOS DE CONSTRUÇÃO E GERENCIAMENTO DE REDES MESH EM AMBIENTES IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Telecomunicações da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Telecomunicações.

Trabalho de Conclusão de Curso defendido e aprovado em: 09 de Agosto de 2022.

Banca examinadora:

Prof. Dr. Lucas Compassi Severo

Orientador

UNIPAMPA

Prof. Dr. Alessandro Gonçalves Girardi

UNIPAMPA

Prof. Dr. Crístian Müller

UNIPAMPA

Assinado eletronicamente por **LUCAS COMPASSI SEVERO**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 09/08/2022, às 16:58, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **ALESSANDRO GONCALVES GIRARDI**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 09/08/2022, às 18:16, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **CRISTIAN MULLER**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 09/08/2022, às 20:30, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0887465** e o código CRC **2835C85C**.

Universidade Federal do Pampa, Campus Alegrete
Av. Tiarajú, 810 – Bairro: Ibirapuitã – Alegrete – RS CEP: 97.546-550

Telefone: (55) 3422-8400

RESUMO

A Internet das Coisas (IoT) é um dos principais temas de projetos na academia e na indústria atualmente. Um grande aumento no número de dispositivos conectados à internet é esperado nos próximos anos. Nestes dispositivos, o uso de Redes *Mesh* Sem Fio (WMN, do inglês, *Wireless Mesh Networks*) é muito importante para que o roteamento do sinal seja aperfeiçoado através de numerosos nós. Entretanto, as WMN tem como desvantagem uma curva de aprendizado íngreme devido a grande quantidade de configurações necessárias. Para reduzir o esforço relacionado ao design da rede em si, este trabalho propõe um *framework* para automatizar o processo de implementação de uma Rede *Mesh* em um dispositivo IoT, provendo todas as configurações, alguns protocolos de rede e um esquema de encaminhamento de protocolo utilizando o microcontrolador da ESP32 da Espressif. Para validar o *framework* proposto, algumas aplicações simples são construídas a fim de exemplificar cenários comumente encontrados em soluções relacionados à Internet das Coisas. Estas aplicações consistiram tanto do software embarcado no dispositivo IoT, quanto da plataforma Web responsável por processar e exibir os dados recebidos da rede de sensores.

Palavras-chave: *Framework*. Internet das Coisas. Redes Mesh Sem Fio.

ABSTRACT

The Internet of Things (IoT) is one of the main research and industry subjects nowadays. A big increase in the number of internet-connected devices is expected for the next years. In these devices, the use of Wireless Mesh Networks (WMN) is very important to improve signal routing through several nodes. However, the WMN has the disadvantage of presenting a considerable steep learning curve due to all the settings needed. To reduce the design effort, this work proposes a framework to automate the process of the WMN implementation in an IoT device by providing all the settings, some internet protocols, and propagation schemes using Espressif's ESP32 processor. To validate the proposed framework, some simple applications were made to exemplify commom scenarios found in Internet of Things related solutions. These applications included both the device's embedded software and the web plataform responsable to process and display the input data.

Keywords: Framework. Internet of Things. Wireless Mesh Networks.

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A internet desde seu início tem sofrido constantes mudanças e evoluções (KHAN et al., 2012). Ela está sempre se transformando em algum novo tipo de hardware e software, o que a torna indispensável (FAROOQ et al., 2015). A Internet do futuro é a Internet das Coisas (IoT, do inglês, *Internet of Things*), que permite a conexão com qualquer dispositivo em qualquer lugar e a qualquer momento (KHAN et al., 2012). A primeira fase do IoT teve como base a comunicação Máquina-Máquina (M2M, do inglês *Machine-to-Machine*), que permitiu a revolução atual nas tecnologias de Internet, sensores inteligentes e dispositivos móveis (AL-FUQAHA et al., 2015). Desde então a concepção de redes de dispositivos inteligentes, que são capazes de se comunicar entre si, é cada vez mais constante no dia a dia.

A Internet das Coisas pode ser encontrada atualmente em diversos aspectos da vida cotidiana, começando por um simples interruptor inteligente, que já é popular em muitas residências, até aplicações em que centenas de dispositivos se intercomunicam de maneira autônoma (propostas para carros e cidades inteligentes). Diversas outras áreas e indústrias já experienciam os efeitos do IoT, sendo algumas destas exemplificadas pela Figura 1.

Um reflexo dessa popularidade e difusão do IoT pode ser percebida através do número de dispositivos conectados, que vem crescendo rapidamente, com um valor estimado de 30 dispositivos IoT por pessoa para 2030 (CHETTRI; BERA, 2019). Existem fatores que vem permitindo e alavancando o desenvolvimento do IoT, sendo alguns deles: *Near Field Communications* (NFC), *Beacons* e até mesmo o *Bluetooth* (GUPTA; GUPTA, 2016). A escalabilidade destas infraestruturas também será beneficiada pela implantação do 5G, que possui em suas resoluções recursos destinadas especificamente à Internet das Coisas (PALATTELLA et al., 2016) .

Figura 1 – Aplicações de IoT na sociedade.

Fonte: (RAIN, 2022)

A combinação de um grande número de dispositivos se comunicando entre eles através de tecnologias sem fio e um cenário plausível no qual estes dispositivos possuem uma infraestrutura mutável é muito importante para o desenvolvimento de novos sistemas de comunicação voltados para IoT.

Neste tipo de sistema cada dispositivo é chamado de nó e pode ser conectar a um ou mais nós. Assim como em cenários convencionais, os nós que constituem a rede são interconectados utilizando alguma topologia especifica, esta geralmente possuí um nó (ou nós) que fazem a interface entre a rede local e a internet em si. A mais comum é a topologia Estrela, em que todos os nós da rede conectam-se diretamente a um dispositivo central. Para o contexto de IoT, no qual os dispositivos finais se comunicam de maneira sem fio e muitas vezes são instalados em lugares de difícil acesso, a topologia Estrela possui a desvantagem de limitar a área geográfica que a rede pode abranger de acordo com o alcance do sinal do dispositivo central (LIU et al., 2017).

Por possuir uma grande quantidade de dispositivos que podem estar dispersos heterogeneamente sobre uma área, uma alternativa a topologia Estrela seria a utilização da topologia Mesh. Em Redes Mesh Sem Fio (WMN, do inglês, *Wireless Mesh Networks*), não há a necessidade de todos os nós estarem conectados em um único ponto central, de modo que a área de cobertura da rede cresce com a adição de nós mais dispersos geograficamente. O estabelecimento de conexão dinâmica entre os pares faz parte da capacidade de auto-organização da rede que, junto com a autoconfiguração dinâmica, constituem como características especificas das WMN (LIU et al., 2017). Estas características diminuem o tempo e dificuldade de implantações de redes IoT.

Sendo assim, este trabalho apresenta o desenvolvimento de um *framework* para inicialização e gerenciamento da rede Mesh, removendo a necessidade de um conhecimento específico sobre a configuração e concepção redes *Mesh* em IoT, e automatiza o funcionamento destas, removendo a necessidade de tratar os eventos que ocorrem enquanto a rede está em operação. O *framework* será desenvolvido utilizando como base o protocolo ESP-WIFI-MESH da Espressif, e permitirá que o desenvolvedor de soluções IoT possa se dedicar em exclusivo à sua aplicação, e não à infraestrutura de comunicação da rede.

1.1 Objetivos

O presente trabalho tem como objetivo principal o desenvolvimento de um *framework* para configuração e gerenciamento automático de redes *Mesh* sem fio para IoT. O *framework* propõe também fornecer a todos os nós a capacidade de comunicação ponto-a-ponto e, para disponibilizar acesso a protocolos baseados na internet, um esquema de encaminhamento de protocolo foi desenvolvido. Este esquema será utilizado então na implementação da comunicação bidirecional utilizando o protocolo MQTT foi realizada. Foi desenvolvida também uma aplicação a fim de validar o *framework* e sua usabilidade no contexto de IoT. Mensagens padronizadas utilizando JSON são utilizadas para comunicar os nós e a plataforma que foi desenvolvida para

processar e exibir os dados.

1.2 Objetivos Específicos

Como parte dos objetivos específicos este trabalho apresenta os seguintes itens:

- Desenvolver uma camada de abstração para configuração do protocolo ESP-WIFI-MESH, para remover a necessidade do usuário de conhecer os detalhes específicos do protocolo, garantindo uma configuração e inicialização consistentemente correta;
- Tornar automático o gerenciamento da rede *Mesh* a partir de sistemas supervisórios encarregados por lidar com eventos gerados internamente ou pela associação e desconexão de nós;
- Prover acesso a protocolos da pilha TCP/IP aos nós da rede *Mesh* utilizando-se de um esquema de encaminhamento de protocolo, a fim de dar acesso a protocolos (em especial o MQTT) aos nós da rede *Mesh*, que normalmente não poderiam utilizá-los;
- Testar o *framework* utilizando diversos tipos de periféricos diferentes, a fim de exemplificar, de maneira simples, diversos cenários que podem ser encontrados em soluções IoT;
- Desenvolver uma plataforma web simples capaz de processar e exibir os dados transmitidos pela rede *Mesh*.

1.3 Organização do Trabalho

Este trabalho é organizado em 5 capítulos. No Capítulo 1 é feita uma introdução ao trabalho, mostrando alguns dos objetivos que propõem-se alcançar. No Capítulo 2 os conceitos fundamentais para o entendimento do trabalho proposto são descritos. O Capítulo 3 compreende o desenvolvimento do *framework* em si, mostrando aspectos da sua implementação interna. Para validar o funcionamento e funcionalidade do *framework* no contexto IoT, o Capítulo 4 apresenta uma aplicação é desenvolvida utilizando o *framework* proposto, além de uma plataforma web capaz de processar os dados transmitidos. Por fim, o Capitulo 5 oferece uma conclusão ao trabalho, assim como a abertura para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os conceitos relacionados com a utilização de rede *Mesh* e seus requisitos, a fim de contextualizar os diversos elementos que compõem este trabalho.

2.1 Internet das Coisas e sua Infraestrutura

O termo Internet das Coisas em si tem sido construído com o passar dos anos, atualmente tendo sua definição geral relacionada com a capacidade de, a partir da utilização da internet, se ter acesso a dados gerados por um dispositivo, assim como controle sobre este (GUPTA; GUPTA, 2016). Uma descrição mais detalhada do tópico também cita certas características importantes encontradas em soluções direcionadas a IoT, sendo algumas delas a capacidade de auto-configuração, interoperabilidade entre protocolos de comunicação e integração com redes de informação, sendo a internet o maior exemplo delas (LI; XU; ZHAO, 2015).

Quanto aos componentes da Internet das Coisas, as diferentes características de um dispositivo final e requisitos de uma aplicação acabam tornando a sua infraestrutura heterogênea (RAZZAQUE et al., 2015). Consequentemente, existe a necessidade da subdivisão do ecossistema em camadas para obter-se uma abstração das diferentes partes da implementação comum das soluções projetadas (AL-FUQAHA et al., 2015) e, ao mesmo tempo, definir o escopo de cada um desses níveis. O número exato de camadas varia dentro da literatura, podendo variar entre 3 e 5 camadas (AL-FUQAHA et al., 2015)(RAZZAQUE et al., 2015), sendo adotado um modelo intermediário de quatro camadas, que mais se assemelha ao descrito em (AL-FUQAHA et al., 2015), ilustrado na Figura 2. Neste modelo a estrutura é organizada em 4 quadrantes representando o ambiente (local ou remoto) e o seu tipo (físico ou virtual), conforme detalhado abaixo:

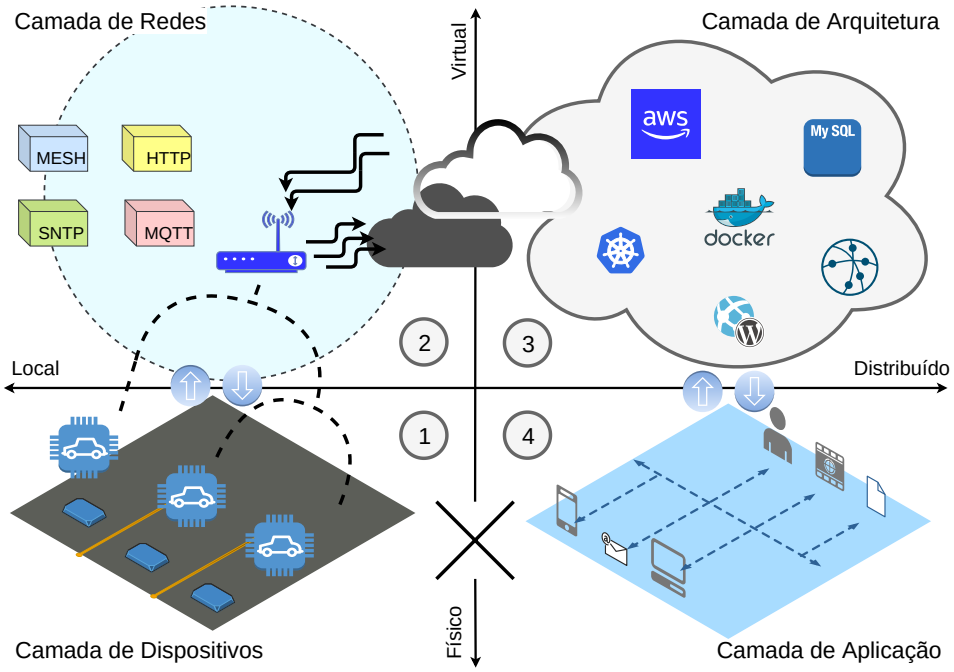
- Quadrante 1: Camada de Dispositivos (Local e Físico)
- Quadrante 2: Camada de Redes (Local e Virtual)
- Quadrante 3: Camada de Arquitetura (Remoto e Virtual)
- Quadrante 4: Camada de Aplicação (Remoto e Físico)

As seções seguintes detalham cada uma das 4 camadas representadas na Figura 2.

2.1.1 Camada de Dispositivos

A interface entre o mundo real e os dados digitais é feita na camada de dispositivos, mostrada no quadrante 1 da Figura 2. Ela é constituída usualmente de dispositivos físicos distintos (sensores, atuadores e microcontroladores) que são responsáveis por coletar e pré-processar as informações para que possam ser enviadas as camadas superiores (AL-FUQAHA

Figura 2 – Organização genérica da Internet das Coisas.



Fonte: Autoral.

et al., 2015). O tipo de sensor utilizado depende do tipo de aplicação desejada, mas entre os mais comuns, encontram-se (CHETTRI; BERA, 2019):

- Sensores de Luminosidade;
- Sensores de Temperatura;
- Sensores de Umidade;
- Sensores Ultrassônicos;
- Sensores de Proximidade;
- Giroscópios;
- Acelerômetros;
- Identificação por radiofrequência (RFID).

A comunicação entre sensores (e/ou atuadores) e o microcontrolador em si é feita através de protocolos de comunicação serial ou paralela. Podendo-se citar alguns como:

- SPI;
- I2C;
- CAN;

- Modbus.

É importante ressaltar que microcontroladores também possuem periféricos como ADC (Conversor Analógico-Digital), DAC (Conversor Digital-Analógico) que podem são utilizados como interface com sensores ou atuadores.

2.1.2 Camada de Redes

Esse nível é o responsável pela transmissão dos dados gerados pela camada anterior para a rede externa (essa podendo ser a internet em si, ou uma rede privada) responsável pelo processamento das informações. Essa camada é representada pelo segundo quadrante da Figura 2. O principal objetivo dessa camada é prover uma abstração para a camada superior, de modo que a modificação ou substituição de dispositivos ou protocolos anteriores não interfiram no funcionamento desta, desde que a convenção feita para a interface entre as duas seja mantida. Por consequência dessa interface nítida, a solução em IoT tem a capacidade de agregar dados, vindos de um ambiente heterogêneo, a uma infraestrutura de redes já existente (LI; XU; ZHAO, 2015).

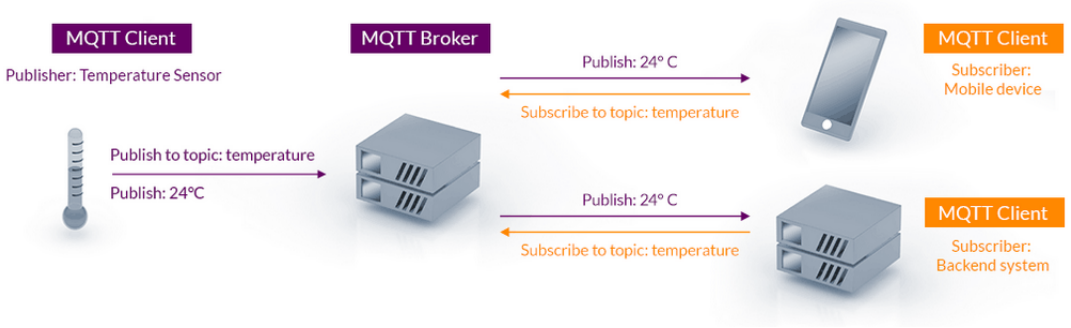
Essa camada pode ser separada entre três diferentes aspectos: tecnologia sem fio, topologia de rede e protocolos de comunicação. Existem diferentes tecnologias sem fio disponíveis, sendo que nenhuma delas se mostrou como líder, devido a falta de padrões definidos para IoT e por diferentes tipos de soluções possuírem diferentes requisitos (PALATTELLA et al., 2016). Alguns dos padrões são:

- *Bluetooth Low Energy* (BLE);
- *Wifi*;
- *Long Range* (LoRa).

Se tratando de protocolos de comunicação, um dos mais utilizados no contexto de Internet das Coisas é o MQTT, um protocolo leve, confiável e seguro que foi projetado para comunicações bidirecionais entre dispositivos remotos (MQTT.ORG, 2022). A Figura 3 exemplifica o funcionamento de uma aplicação utilizando o protocolo MQTT, sendo possível identificar três tipos de dispositivos diferentes:

- *MQTT Client (Publisher)*: Publica dados para o Tópico do *MQTT Broker* configurado;
- *MQTT Client (Subscriber)*: Se inscreve em um tópico, recebendo as mensagens que forem publicadas a este tópico;
- *MQTT Broker*: Responsável por encaminhar as mensagens publicadas a certo tópico a todos os dispositivos inscritos nele.

Figura 3 – Arquitetura MQTT



Fonte: (MQTT.ORG, 2022).

2.1.3 Camada de Arquitetura

Esta camada é representada pelo quadrante 3 da Figura 2 e é onde os serviços oferecidos na nuvem encontram-se. Nesta camada, os dados recebidos da camada de Redes são processados e encaminhados para a camada de aplicação. Esta camada é conhecida como nuvem, pois na sua maioria estão hospedadas em servidores remotos que executam funções e processamento específicos para cada aplicação.

Existem várias plataformas disponíveis nessa camada, tais como Amazon AWS, MySQL, Docker, entre outras.

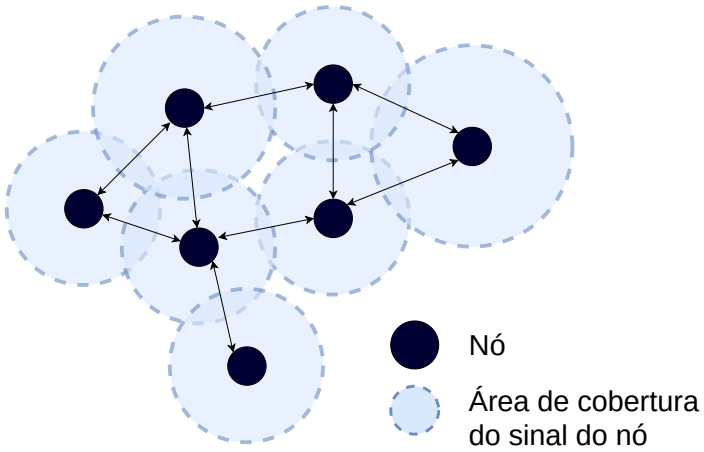
2.1.4 Camada de Aplicação

Esta camada, equivalente ao quadrante 4 mostrado na Figura 2 e é responsável por apresentar dados, receber informações e comandos de usuários ou sistemas supervisores. Como esta camada realiza a interface com os usuários, diversos meios de iterações podem ser utilizados, tais como mensagens de texto, mensagens em aplicativos de comunicação, e-mails, aplicativos específicos em smartphones ou telas, entre outros. Exemplos práticos dessa camada seriam aplicações *Mobile*, Totens Digitais e Sistemas de Controle Industrial.

2.2 Redes Mesh Sem Fio

Os cenários atuais de IoT por muitas vezes possuem em comum certas características que precisam ser atendidas, sendo uma delas, um número de dispositivos variável. As redes sem fio tradicionais nem sempre conseguem atender esses pré-requisitos. Nessas topologias tradicionais, todos os dispositivos que pertencem a rede (chamados de nós) são conectados a um ponto central (chamado de *gateway*), sendo este o responsável por fazer o intermédio entre a sua rede e a rede externa, assim como gerar o sinal sem fio no qual os demais nós irão utilizar como canal de comunicação. É possível perceber que, como o sinal é gerado pelo nó central, apenas dispositivos que estejam na área de alcance deste sinal conseguirão se associar a rede.

Figura 4 – Rede Mesh Sem Fio Genérica



Fonte: Autoral.

A única solução para que um dispositivo que esteja geograficamente fora do alcance do nó central consiga utilizar a rede é estender a cobertura deste sinal. Para isso, é necessário elevar o nível de potência do sinal transmitido e a sensibilidade dos transceptores de RF presentes nó central e no dispositivo cliente, resultando em maiores níveis de consumo de energia. Este problema persistirá sempre que outro nó com localização extrema tenha que ser adicionado.

Com a utilização de redes *Mesh* Sem Fio, o nó que deseja utilizar a rede precisa apenas estar dentro do alcance de qualquer um dos nós já associado à rede. Como mostrado na Figura 4, a área geográfica coberta pela rede é estendida a cada nó adicionado, sendo sempre limitada apenas pelos nós na extremidade. Assim como os nós podem ser dinamicamente adicionados a rede, nós que sejam desconectados da rede (intencionalmente ou não) não inutilizam a rede, adicionando então as WMN à capacidade de tolerância a falhas.

Pelas redes *Mesh* não possuírem um nó central no qual todo o tráfego seja afunilado para, os integrantes da rede possuem tabelas de roteamento (que são listas de todos os nós conectados aos arredores deste nó especifico), para que o tráfego seja direcionado corretamente e não seja necessário o desperdício de banda de transmissão utilizando a técnica *broadcast* de transmissão.

2.3 Sistemas Operacionais de Tempo-Real (RTOS)

Para o gerenciamento da rede de comunicação e demais recursos presente nos micro-controladores modernos, são utilizados sistemas operacionais embarcados. Estes Sistemas Operacionais (SO) são desenvolvidos especificamente para este tipo de aplicação embarcada, uma vez que os dispositivos que os usam possuem limitações de memória, processamento e consumo de energia.

Os SOs para sistemas embarcados podem ser divididos quanto a serem destinados ou não para computação em tempo real (*real-time computing*). Além disso, para cenários onde há

grandes limitações no projeto, pode ser implementado uma aplicação sem um sistema operacional em si. Esse tipo de programação, chamada de *Bare Metal* (FELIZ, 2022), é bastante utilizada em dispositivos de baixo-custo ou que possuam baixos recursos (memória RAM, Flash), pois, em muitas vezes, estes não possuem a capacidade de executar um sistema operacional. Quando empregada à programação *Bare Metal*, é comum encontrar um loop de controle simples (LUTKEVICH, 2022) que consiste da execução contínua de uma determinada sequência de ações, normalmente controlando apenas uma variável (temperatura, por exemplo). Este tipo de programação é bastante comum em aplicações IoT *Bare Metal*, mas também pode ser empregada em casos onde há um sistema operacional embarcado.

Uma das características que destaca um Sistema Operacional comum de um Sistema Operacional de tempo-real (RTOS) é que, no segundo, tarefas executadas neste possuem prazos de expiração, que devem ser alcançados para que o sistema funcione corretamente. Mesmo para sistemas que não sejam *time-critical*, RTOSes oferecem vantagens em seu uso, sendo algumas delas (TAN; ANH, 2009):

- Melhor Desenvolvimento de Software: RTOSes, assim como sistemas operacionais convencionais, também possuem o conceito de funções que são desempenhadas de maneira concorrente (muitas vezes chamada de Tarefas), o que permite que o desenvolvimento do software para este microcontrolador seja modularizado quanto a função desempenhada pela tarefa e o desenvolvimento destas podem ser distribuído entre diferentes desenvolvedores;

- Gerenciamento de Recursos: Gerenciamento de tarefas, memória e interrupções são alguns dos recursos que RTOSes podem gerenciar com maior facilidade, através da abstração que RTOSes proveem;

- Gerenciamento de Temporização: RTOSes proveem mecanismos para gerenciamento de tempo, como temporizadores, funções para criação de atrasos precisos (dependendo do grau de precisão necessário), criando para o desenvolvedor uma camada de abstração conveniente quanto a implementação física destes.

Sistemas operacionais de tempo-real (assim como sistemas operacionais comuns) possuem, por muitas vezes, a capacidade de ser multitarefas, sendo capaz de gerenciar diversos processos (ou tarefas) simultaneamente (LUTKEVICH, 2022). Esse gerenciamento é feito a partir de agendadores (ou, do inglês, *schedulers*) que decidem qual processo irá executar de acordo com o seu algoritmo interno de agendamento (SIEWERT, 2016).

Existem diferentes tipos de sistemas operacionais, alguns deles são baseados em Unix e software proprietário, como, por exemplo, o VxWorks. Contudo, existem também RTOSes Open Source específicos para microcontroladores como, por exemplo, o FreeRTOS que é utilizado como padrão em microcontroladores como o ESP32 e STM32 e RTOSes como o Zephyr,

que é instalado como padrão em microcontroladores da Nordic Semiconductor. Vale ressaltar que, em muitos casos, é possível substituir (ou até mesmo remover) o sistema operacional utilizado em um microcontrolador.

2.4 Requisitos de Infraestrutura

Para o projeto de redes IoT que utiliza-se da infraestrutura *Mesh*, o consumo de energia, o custo e o tamanho físico dos dispositivos devem ser levados em consideração, já que os recursos computacionais e monetários são limitados (LI; XU; ZHAO, 2015). Adicionalmente, os seguintes fatores devem ser observados:

- Características dos dispositivos: Estes devem ser de baixo custo para manter o custo individual de cada dispositivo, uma vez que o objetivo final é ter uma rede com alto número de dispositivos. Apesar do baixo custo, estes devem possuir a capacidade de processamento para suportar redes *Mesh*;
- Baixo custo e complexidade de instalação: Como redes *Mesh* são dinâmicas e muitas vezes espalhadas geograficamente, é necessário que a instalação da infraestrutura seja simples e de baixo custo;
- Capacidade de energia: É esperado que dispositivos IoT operem sem a intervenção de humanos por um longo período, o que faz a necessidade de uma longa bateria ou sistemas de captação de energia algo indispensável. Outro fator que influencia este pré-requisito é que, frequentemente, estes dispositivos são instalados em locais remotos, o que dificulta (ou, impossibilita) o acesso físico para a manutenção deste;
- Cobertura do sinal: A capacidade de expansão dinâmica do alcance do sinal sem fio da rede como um total é extremamente necessária, uma vez que dispositivos podem ser instalados temporalmente e geograficamente espalhados;
- Segurança: A segurança, tanto lógica, quanto física, deve ser garantida em cada dispositivo individualmente e na rede como um todo.

3 METODOLOGIA

A fim de encapsular a camada de rede, camada 2 do modelo de organização de um ecossistema em IoT, o *framework* proposto neste trabalho foi separado entre 3 partes distintas. Seu primeiro modulo é responsável por configurar e inicializar os parâmetros relevantes da Rede *Mesh*. O segundo módulo irá gerenciar a organização da rede, assim como lidar com mensagens internas que são trocadas entre os nós. Já o terceiro módulo será implementado para disponibilizar o acesso aos protocolos disponíveis na camada física de rede.

As próximas subseções detalham o desenvolvimento do *framework* proposto.

3.1 Escolha da Plataforma

O primeiro passo no desenvolvimento do *framework* foi escolher uma plataforma embarcada para usar como referência no desenvolvimento deste trabalho. Existem diversas alternativas disponíveis comercialmente que poderiam ser aplicadas no nicho de utilização deste trabalho. Entre elas detacam-se:

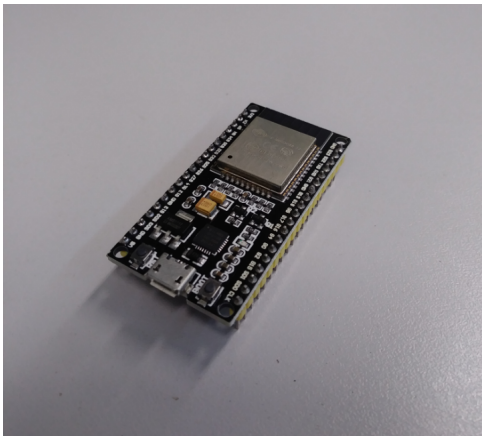
- STM32: Uma família de microcontroladores com alta variedade de diferentes dispositivos, variando de acordo com fatores como: performance, baixo consumo e conectividade sem fio (ST, 2022);
- Nordic: O portfólio da Nordic tem foco na variedade de tipos de tecnologias sem fio oferecidas, tendo como exemplo: Wi-Fi, Bluetooth Low Energy, Thread, Zigbee, entre outros (NORDIC, 2022).

Se tratando de microprocessadores, existe outra gama enorme de dispositivos, desde o raspberry pi (FOUNDATION, 2022) até o Labrador, uma placa nacionalmente desenvolvida pela Caninos Loucos (CANINOS, 2022). Estes microprocessadores normalmente vem com o sistema operacional Linux (ou com alguma variação dele), mas é possível encontrar também dispositivos que suportam as versões mais recentes do Windows, um exemplo sendo a Latte-Panda 4G/64GB (SLANT, 2022).

O candidato escolhido foi o ESP32, um sistema-em-um-chip (SoC, do inglês, *System On Chip*). Este consiste de um microcontrolador robusto e confiável que foi projetado para dispositivos móveis, eletrônicos vestíveis e aplicações IoT (ESPRESSIF, 2021a). As características que o fazem um candidato interessante para aplicações em Internet das Coisas são suas funcionalidades como um chip híbrido *Wi-Fi/Bluetooth*, processador com dois núcleos e, ainda assim, possuir um baixo consumo e uma boa relação custo-benefício. No que se refere a parte de software, a Espressif (empresa desenvolvedora do ESP32), oferece um ambiente de desenvolvimento chamado *Espressif IoT Development Framework* (esp-idf), que utiliza no ESP32 o sistema operacional FreeRTOS e diversas APIs, incluindo uma implementação proprietária de um protocolo *Mesh*, chamada ESP-WIFI-MESH.

A Figura 5 ilustra uma placa de desenvolvimento de baixo custo com o processador ESP32, base para o desenvolvimento deste projeto.

Figura 5 – Microcontrolador ESP32.



Fonte: Autoral.

3.2 Características do protocolo ESP-WIFI-MESH

Como citado anteriormente, o esp-idf possui uma implementação própria de um protocolo *Mesh*, sendo que o *framework* proposto neste trabalho foi implementado em cima deste. Os detalhes do funcionamento do ESP-WIFI-MESH são descritos nesta seção.

É necessário deixar claro que utilizar o protocolo ESP-WIFI-MESH como base para o desenvolvimento do *framework* proposto cria uma dependência ao microcontrolador, porém este aspecto do trabalho foi considerado como aceitável em favor da aceleramento que o desenvolvimento do trabalho ganharia.

Quanto ao funcionamento e organização da implementação da rede *Mesh* da Espressif, este provê características (discutidas nas seções anteriores) como autoconfiguração e tolerância a falhas. Em relação à tecnologia sem fio utilizada, como o próprio nome sugere, o ESP-WIFI-MESH utiliza-se do *WiFi* para a comunicação entre os seus nós e o *Gateway*. O *Gateway* (em singular) denota a maior diferença entre redes *Mesh* usuais e o ESP-WIFI-MESH. Nas tradicionais, a topologia de rede é a própria *Mesh*, enquanto a da Espressif é organizada em uma topologia em árvore. Essa diferença de topologia acarreta em outras características distintas, sendo o conhecimento delas necessário para utilização correta do protocolo, assim como ter entendimento do seu funcionamento interno.

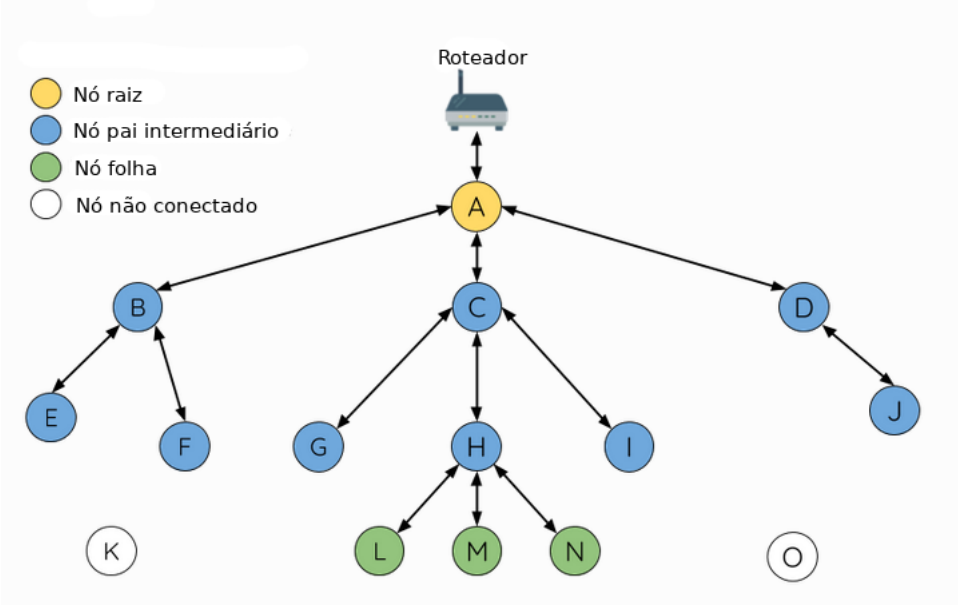
Nessa topologia, cada dispositivo conectado à rede é chamado de nó. Quando conectados entre si, estes nós podem assumir duas funções: pai ou filho. Um nó é considerado filho quando conectado à interface Ponto de Acesso (*SoftAP*) de outro dispositivo (utilizando sua interface de rede lógica no modo Estação) e chamado de nó pai quando outros dispositivos utilizam suas respectivas interfaces no modo Estação (*SoftAP*, mas também pode ser chamada

de modo Cliente) para se conectar à interface Ponto de Acesso deste nó em particular. O nó (no singular) pai de cada dispositivo pode ser identificado pelo nó imediatamente acima deste (sentido em direção ao roteador na Figura 6). Cada nó guarda em sua tabela de roteamento uma relação de todos os seus nós filhos, os diretamente conectados em sua interface Ponto de Acesso. O nó conectado ao roteador é o único que possui acesso direto à rede externa provida pelo roteador e recebe o nome especial de nó raiz. Os nós que não possuem filhos conectados a eles constituem o final lógico da rede, sendo chamados de nós folha.

A escolha de qual nó cada dispositivo se conectará (qual será o seu nó pai) é feito a partir de um algoritmo que decide qual o melhor nó de acordo com o nível do sinal (*rssi*) dos nós ao seu redor. Esta rotina ocorre toda vez que um novo nó tenta se conectar à rede e, não necessariamente, este novo dispositivo deve se conectar às camadas mais profundas da rede (mais longe do roteador). Por se tratar de um nó característico e de extrema importância para a rede *Mesh*, no primeiro momento em que os dispositivos começam a formar a rede, este nó é escolhido. Ele é selecionado através de um algoritmo de votação, no qual todos os nós transmitem entre si o seu nível de sinal em relação ao roteador e, a partir de todos estes dados, cada nó decide quem seria o melhor dispositivo para se tornar o nó raiz. Uma porcentagem miníma de aprovação na escolha dos nós deve ser atingida para que a votação seja válida (um exemplo sendo 90% dos nós escolham o dispositivo com endereço mac igual a AA:BB:CC:33:11:22).

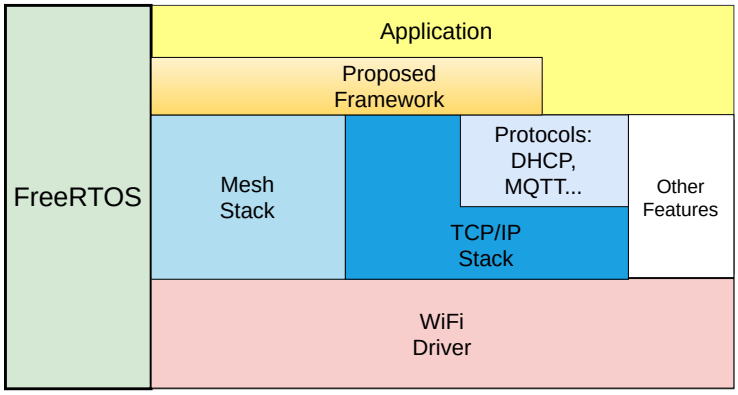
Como é possível observar através da Figura 6, todo o tráfego direcionado para a rede externa (em direção ao roteador), passará pelo nó raiz, tornando-se o gargalo da comunicação da rede *Mesh* e a rede externa. Por ser o único nó diretamente conectado ao *gateway*, o nó raiz é o único ponto da rede que possui acesso direto a protocolos de rede.

Figura 6 – Topologia da Rede *Mesh* implementada pela Espressif.



Fonte: Adaptado de (ESPRESSIF, 2021b).

Figura 7 – *Framework* proposto em relação aos demais recursos do esp-idf.



Fonte: Adaptado de (ESPRESSIF, 2021b).

3.3 Abrangência do *framework* implementado

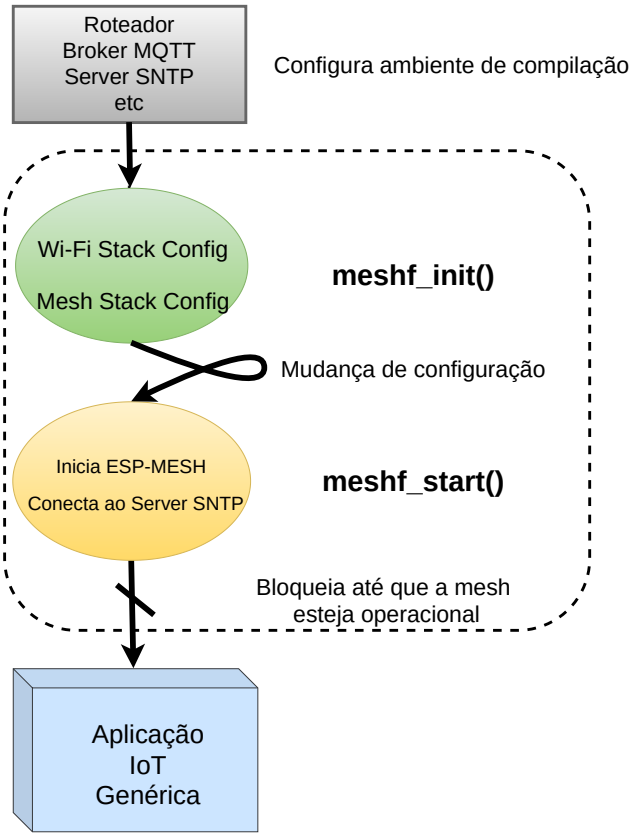
Com um entendimento base do funcionamento do protocolo, é então possível definir a abordagem quanto à implementação do *framework*, assim como o escopo do mesmo.

Analisando a Figura 7, é possível observar que a implementação *Mesh* é feita ao lado da pilha TCP/IP, ou seja, o usuário que desenvolver uma aplicação que utiliza-se diretamente do ESP-WIFI-MESH, apesar de ter completo acesso aos protocolos TCP/IP, estes são inoperantes, a menos que sejam utilizados no nó raiz. Esta característica seria uma grande desvantagem ao utilizar redes *Mesh*, uma vez que soluções em IoT muitas vezes dependem de protocolos como o MQTT, HTTP e WebSockets para enviar dados para redes externas. Pensando nisso, o *framework* proposto tem como objetivo, além da automatização da configuração e gerenciamento da rede, oferecer ferramentas para acessar os protocolos TCP/IP.

Um modelo genérico do *framework* pode ser dividido em diferentes partes com papéis distintos quanto ao ESP-WIFI-MESH e sua interação com as demais partes da aplicação (tanto do próprio ESP, quanto do usuário do *framework*). Os primeiros 3 módulos, que são mostrados na Figura 8, têm como objetivo primário garantir que a rede *Mesh* esteja operacional, antes que a aplicação do usuário possa ser executada. Isso permite que o desenvolvedor possa focar seus recursos especificamente a sua aplicação, sem a necessidade de se preocupar com a operação da rede *Mesh* em si.

O ESP-IDF provê um ambiente, baseado em Python, chamado *kconfiglib*, sendo nele o local onde configurações do projeto podem ser gravadas em tempo de compilação. Neste ambiente é feita uma seção para que parâmetros relacionados ao *framework* possam ser guardados. Uma das vantagens que utilizar este espaço gera é a remoção de informações sensíveis de dentro do código em si (que muitas vezes iriam ser gravados em texto plano). Os três primeiros parâmetros são diretamente relacionados ao ponto de acesso que o nó raiz irá se conectar. O primeiro parâmetro diz respeito ao nome da rede sem fio, que é chamado de SSID (do inglês, *service set identifier*). O segundo é a senha dessa respectiva rede e o terceiro corresponde ao ca-

Figura 8 – Fluxo inicial do código do *framework* em conjunto com a aplicação genérica do desenvolvedor.



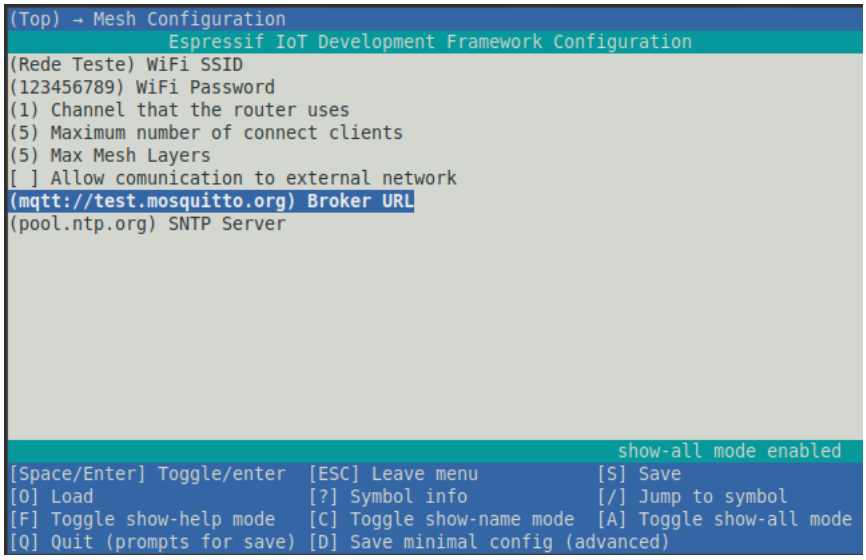
Fonte: Autoral.

nal WiFi utilizado. As três configurações seguintes são diretamente relacionadas ao *framework* em si. Como foi descrito anteriormente, cada nó possui um número de nós filhos diretamente conectados a ele e, a opção "*Maximum number of connected children*"limita o número máximo permitido. A opção seguinte limita o número de camdas que a rede (como completa) pode ter, e a opção seguinte "*Allow communication to external network*"controla o acesso a redes externas por parte da rede Mesh, útil para aplicações que possuam apenas comunicação interna entre os nós. Os dois últimos parâmetros são relacionados aos protocolos MQTT e SNTP, sendo definidos aqui devido ao esquema de encaminhamento de protocolo. Estas configurações são o endereço IP do servidor (S)NTP (Protocolo de Tempo para Redes, ou, do inglês, *Network Time Protocol*) e o endereço IP do Broker MQTT. Este menu, assim como uma configuração exemplo, são mostrados na Figura 9.

3.4 Configuração e Inicialização da Rede *Mesh*

Os dois próximos módulos mostrados na Figura 8 foram implementados em código e suas funções devem ser chamadas antes do código da aplicação para que a operação da rede *Mesh* ocorra como esperado. A primeira função, chamada meshf_init() se encarrega de inicia-

Figura 9 – Ambiente para configuração das variáveis de ambiente.



Fonte: Autoral.

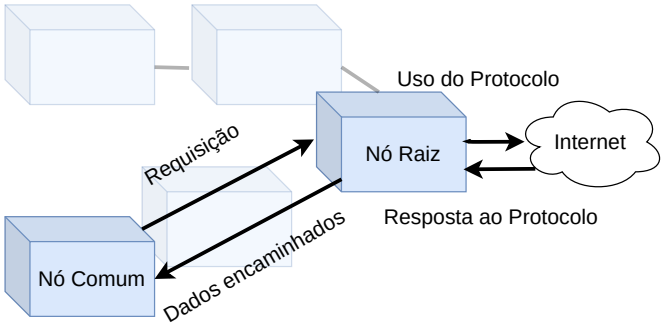
lizar todas as configurações do ESP-WIFI-MESH para valores padrões ou considerados como ideais. Através do uso das funções do próprio ESP-WIFI-MESH é possível fazer alterações nessas configurações, ou seja, apesar da escolha destas configurações fugir do escopo desta função, um usuário com conhecimento sobre o ESP-WIFI-MESH poderia alterar parâmetros conforme seu interesse.

A função de meshf_init() é simples, porém extremamente importante. Ela inicializa as configurações definidas até o momento e bloqueia o fluxo do código enquanto a rede *Mesh* não estiver totalmente operacional. Quando esta função retorna, o código do usuário (que depende da rede) pode ser executado sem nenhum receio. Estes três módulos iniciais em conjunto garantem o desacoplamento inicial entre o código do usuário e o do *framework* em si, deixando bem definidos em que momentos da execução do programa o *framework* tem controle do fluxo do código no momento inicial da organização da rede *Mesh*. O Apêndice A mostra, de um ponto de vista do desenvolvimento do código em si, como essas funções devem ser utilizadas, servindo com um exemplo base para o desenvolvimento de uma aplicação real que empregue o *framework*.

3.5 Esquema de Encaminhamento de Protocolo

Além de uma infraestrutura funcional, o *framework* precisa fornecer meios para que os nós possam se comunicar entre si e com a rede externa. Porém, o fato de que apenas o nó raiz está conectado diretamente à rede externa (ou seja, a internet), impede que os demais nós consigam usufruir de protocolos baseados na Camada de Aplicação da pilha TCP/IP. Apesar de uma comunicação interna entre nós ser facilmente alcançada nativamente com o ESP-WIFI-MESH, um mecanismo para que nós comuns tenham acesso a protocolos de rede é de extrema

Figura 10 – Esquema de encaminhamento de protocolo.



Fonte: Autoral.

importância para que uma aplicação baseada em Internet das Coisas realmente tenha acesso à internet. Para contornar este problema, um esquema de encaminhamento de protocolo é proposto, mostrado na Figura 10. Neste, o nó que deseja utilizar o protocolo TCP/IP envia uma mensagem para o nó raiz requisitando a utilização de tal. O nó raiz, ao receber esta requisição, conecta-se à internet e utiliza nativamente o protocolo, encaminhando para o nó original os dados que a internet o retornar.

Este esquema de encaminhamento de protocolo oferece uma camada de abstração ao usuário do *framework*, de modo que ele não precise explicitamente declarar que irá utilizar a funcionalidade nativamente ou utilizando este esquema proposto. Então, se tratando da implementação interna, este é o primeiro tópico a ser discutido.

Para garantir que o usuário acesse os protocolos utilizando-se das funcionalidades providas pelo *framework*, os protocolos são primeiramente chamados indiretamente a partir de uma função que embrulhe (chamada de *wrapper*) o acesso à função do esp-idf. Este primeiro passo possui um pequeno custo computacional adicional quando o nó é raiz, pois, internamente, seria apenas uma função que acessaria a funcionalidade nativa. Pelo fato que a rede *Mesh* não possui as características comuns que uma rede de computadores baseada no modelo TCP/IP comum (sendo a falta de endereçamento IP uma característica significativa), todos os protocolos que dependem destes aspectos retornariam falha caso fossem utilizados diretamente pelos nós comuns. Para lidar com isto, dois componentes distintos são desenvolvidos. O primeiro sendo, em tempo de execução, identificar se o nó em questão é raiz ou não e, caso não seja, enviar uma mensagem de requisição para que o nó raiz faça a utilização deste protocolo em nome do nó requisitante.

O segundo componente tem como função tratar essas requisições de modo autônomo, sem que seja necessária qualquer intervenção do desenvolvedor da aplicação. Para isso, uma tarefa em plano de fundo é responsável, continuamente, por gerenciar as requisições que o nó recebe (quando em execução em um nó raiz). Quando em um nó comum, tem como função lidar com a resposta vinda do nó raiz. Cada protocolo possui um comportamento característico, então, para cada protocolo, essa tarefa supervisória necessita ter um comportamento personali-

zado para este protocolo em particular.

Para melhor exemplificar as peculiaridades que a implementação deste esquema de encaminhamento pode acarretar para cada protocolo, detalhes quanto à sua implementação para o uso do protocolo MQTT são apresentados. Tendo como base os conceitos apresentados sobre o protocolo MQTT, duas funções foram implementadas: a habilidade de publicar alguma mensagem em algum tópico e a capacidade de se inscrever em algum tópico, recebendo os dados enviados a este.

Na primeira função, *meshf_mqtt_publish()*, os parâmetros relacionados ao tópico e aos dados a serem publicados são organizados em um pacote formatado em JSON e enviados para o nó raiz. Quando este pacote chega ao nó raiz, a tarefa supervisória extrai esses campos e utiliza-os para publicar nativamente. Quanto a função *meshf_mqtt_subscribe()*, a fim de manter uma interface consistente para ambos os tipos de nós, a implementação para o nó raiz também sofreu alterações.

A funcionalidade de inscrição em tópicos MQTT pode ser dividida em duas partes, a primeira se assemelhando muito ao *meshf_mqtt_publish()*, onde o nó raiz se inscreve nativamente no tópico de interesse. A segunda parte é relacionada a recepção dos dados em si. Quando o nó raiz recebe um dado vindo do tópico no qual é inscrito, esta mensagem é processada por um gerenciador de eventos da implementação nativa do MQTT no ESP32. A fim de padronizar o acesso a essa informação, foi decidido implementar uma injeção de dependência no *meshf_mqtt_subscribe()*, que recebe um ponteiro para um função que será chamada para processar os dados vindos do gerenciador de eventos. Quando a mensagem recebida tem como alvo o nó raiz, esta função é chamada localmente. Caso contrário, a mensagem é encaminhada para o nó correto (utilizando-se da tarefa supervisória) e, neste nó, essa função é então invocada. Mensagens enviadas para a rede *Mesh* utilizando o protocolo MQTT devem ser formatadas em JSON e conter obrigatoriamente as chaves ”mac”e ”data”, sendo essa primeira responsável por identificar para quem essa mensagem é direcionada.

3.6 Detalhes quanto à implementação interna do *framework*

Esta seção tem como objetivo entrar em detalhes quanto a implementação dos mecanismos do *framework* que foram apresentados nas seções anteriores.

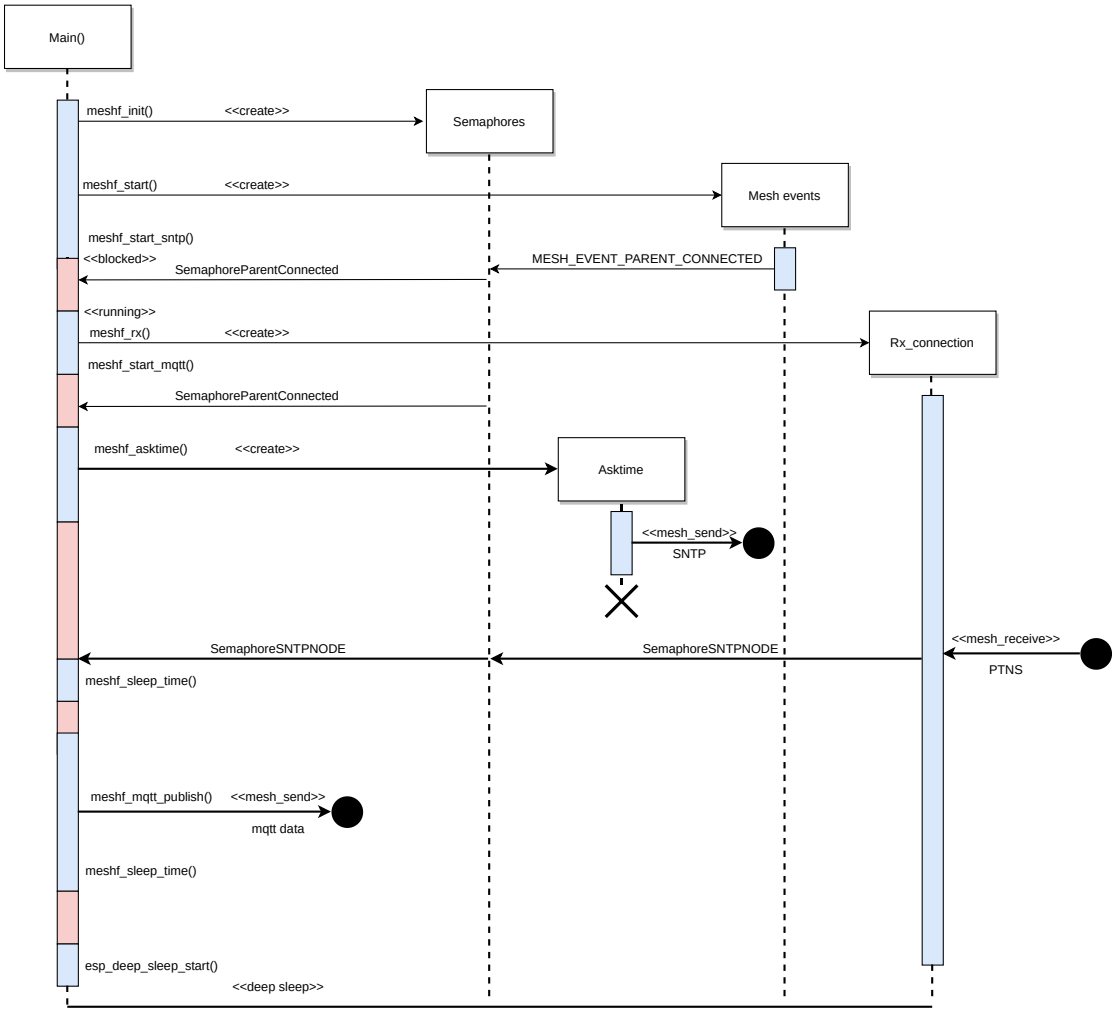
O primeiro mecanismo utilizado, citado ao fim da seção sobre a função meshf_start(), é a habilidade do *framework* bloquear o fluxo do código até que certa condição seja satisfeita. Esta capacidade de impedir o avanço do código é utilizada quando a função necessita de algum recurso externo. No design das funções que compõem este *framework*, cada uma destas possui no máximo um ponto fixo em seu fluxo de execução onde ela possa ser bloqueada (chamado de ponto de sincronização).

O processo de bloquear o fluxo do código é implementado utilizando mecanismos do sistema operacional FreeRTOS, chamados de Semáforos Binários. Estes semáforos possuem dois estados, cheio ou vazio, que podem ser alterados utilizando-se de suas funções *take*, para

retirar, e *give*, para preencher. Quando uma entidade tenta utilizar o *take* de um semáforo que já esta vazio, ela é bloqueada até que outra entidade do sistema operacional preencha esse semáforo novamente (utilizando o *give*).

Cada recurso externo ou relativo a rede *Mesh* tem sua disponibilidade representada por um semáforo, onde o *take* é realizado pelas funções do *framework* que são disponíveis para o desenvolvedor e o *give* é feito por partes internas do *framework*, ou eventos do Protocolo ESP-WIFI-MESH, ou algum protocolo baseado na pilha TCP/IP. Nas Figuras 11 e 12, a seta apontando para o eixo dos semáforos representa a operação *give*, enquanto a seta saindo do eixo, representa a operação *take*.

Figura 11 – Fluxo de execução de uma aplicação em um nó comum.



Fonte: Autoral.

Em *meshf_start()*, a função tenta retirar o semáforo *SemaphoreParentConnected*, porém, este só é preenchido quando o evento do protocolo ESP-WIFI-MESH (representado pelo eixo *Mesh events*), MESH_EVENT_PARENT_CONNECTED, ocorre. Até que este evento ocorra, o fluxo do código fica bloqueado, estado que é representado pela cor vermelha no eixo *Main()* nas Figuras 11 e 12.

Além dos semáforos e eventos, o *framework* desenvolvido também utiliza-se de tarefas para realizar as suas funções internas. Estas tarefas ocorrem simultaneamente entre elas e a tarefa principal *Main()*. O principal objetivo da utilização destas, é a habilidade de realizar tarefas descorrelacionadas de maneira simultânea, diminuindo o tempo ocioso que o(s) processador(es) teria(m) enquanto alguma outra tarefa estaria bloqueada ou simplesmente inativa.

Este conceito de tarefas executando de maneira concorrente é explorada para atingir a implementação do esquema de encaminhamento de protocolo. Como descrito nas seções anteriores, quando um nó não-raiz quer utilizar algum protocolo TCP/IP, ele envia uma requisição para o nó raiz, que por sua sua vez, utiliza o protocolo e encaminha para o nó original a resposta obtida. O desenvolvedor da aplicação que utiliza o *framework* é abstraído dessa comunicação entre os nós. Essa abstração é obtida através da utilização de uma tarefa persistente encarregada de lidar com as mensagens recebidas pelo nó.

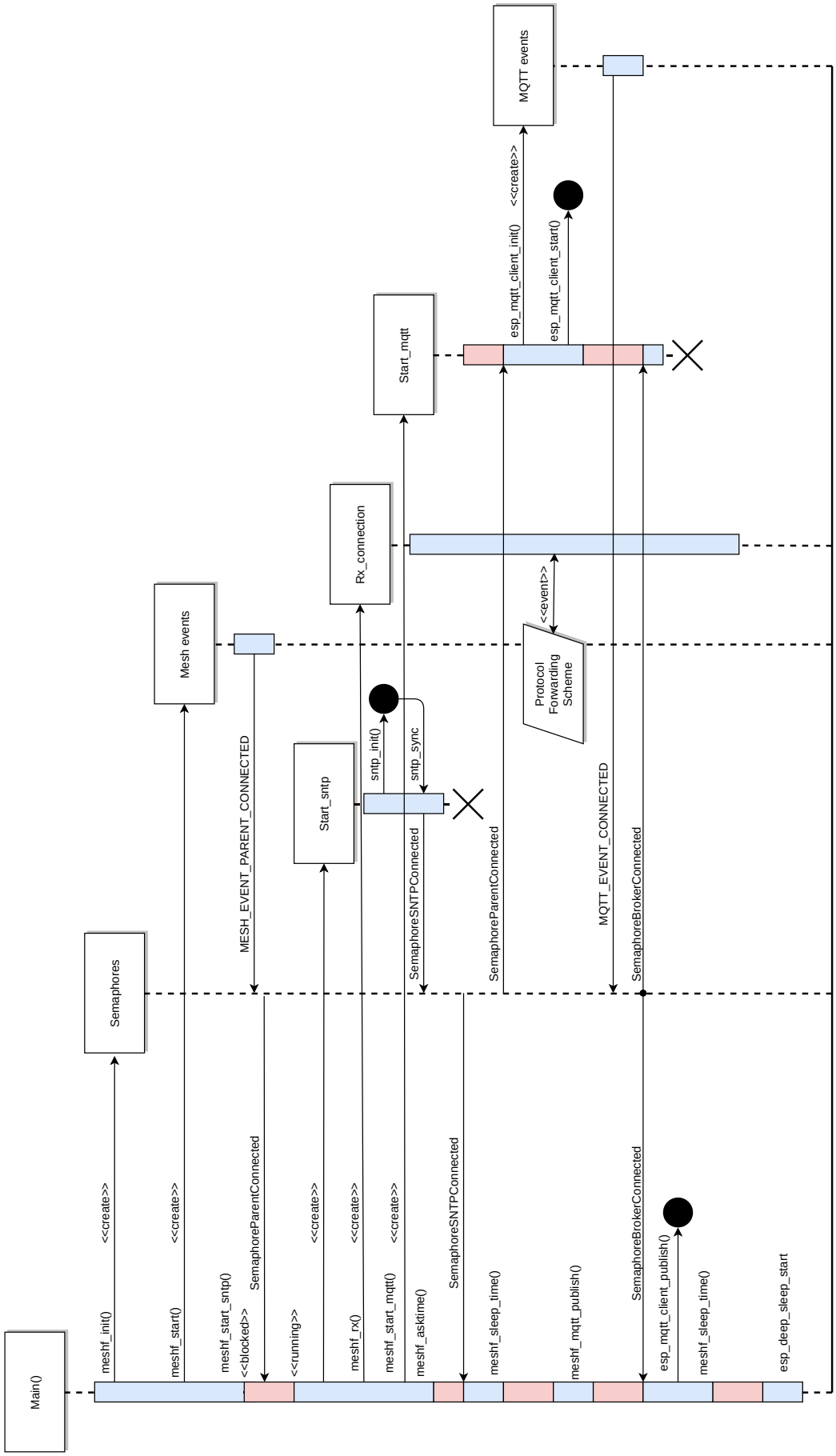
Para exemplificar a interação entre os mecanismos apresentados nessa seção, o funcionamento da função *meshf_asktime()* será apresentado em detalhes.

Meshf_asktime() tem como função atualizar o relógio de tempo real (RTC, ou, do inglês, *real-time clock*) interno do ESP32 utilizando o protocolo SNTP. Para ter acesso a esse protocolo, o *framework* implementado aplica o esquema de encaminhamento de protocolo. Ao ser chamada, a função *meshf_asktime()* cria a tarefa *Asktime* e então bloqueia o fluxo do código de *(Main())* ao tentar retirar o semáforo *SemaphoreSNTPNODE*. A tarefa recém criada, *Asktime*, envia para o nó raiz a requisição dos dados necessários para atualizar seu relógio RTC. Eventualmente, a tarefa *Rx_connection*, que é responsável por gerenciar as mensagens que chegam ao nó, recebe a resposta referente ao pedido da tarefa *Asktime*, preenchendo o semáforo *SemaphoreSNTPNODE* e, consequentemente, desbloqueando a função *meshf_asktime()* em *Main()*. Vale ressaltar que a tarefa *Asktime* não é persistente, ou seja, ao atingir seu objetivo, ela é destruída, não desperdiçando recursos do sistema.

O funcionamento geral é o mesmo para nós comuns e para o nó raiz da rede, porém o último possui um número maior de eventos para lidar, pois este utiliza os protocolos TCP/IP diretamente.

Assim como o nó é escolhido dinamicamente pelo protocolo ESP-WIFI-MESH, o *framework* desenvolvido identifica dinamicamente quem é o nó raiz, tendo o comportamento condizente com o que é esperado deste. Mais detalhes quanto a operação comum de um nó raiz podem ser encontrados na Figura 12.

Figura 12 – Fluxo de execução de uma aplicação no nó raiz.



Fonte: Autoral.

4 VALIDAÇÃO DO *FRAMEWORK* E RESULTADOS OBTIDOS

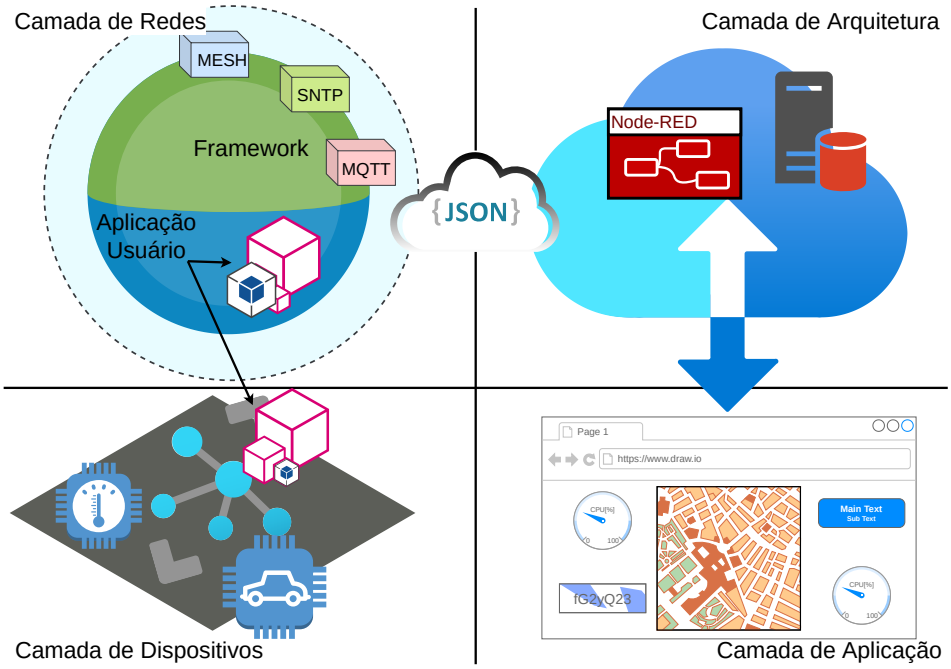
Com o desenvolvimento do *framework*, a etapa seguinte do trabalho consiste na validação do funcionamento adequado do mesmo e, para isto, foi feita a coleta de alguns parâmetros e o comportamento do *framework* em diferentes cenários. Como um dos objetivos do trabalho é gerar um *framework* que possa ser utilizado de maneira genérica, a capacidade de integração com diferentes tipos de periféricos é uma exigência.

Diferentes aplicações em IoT possuem diferentes componentes e requisitos mas, olhando de uma maneira superficial, todos possuem certas partes em comum e estas similaridades entre aplicações são utilizadas como os elementos que compõem a aplicação desenvolvida. A definição da versão genérica do sistema completo é representada na Figura 13. Para que cada parte do trabalho possa ser validado de maneira individual, certos elementos da aplicação desenvolvida tiveram sua funcionalidade falsificada e, à medida que os parâmetros do *framework* são certificados conforme o comportamento que era esperado destes, estas funcionalidades simuladas são substituídas por componentes reais, tornando a aplicação mais realista em relação a uma solução que poderia ser encontrada atualmente.

4.1 Implementação de aplicação utilizando o *Framework* e falsos sensores

Para esta primeira concepção de um sistema que emule uma aplicação real, foi considerado um cenário onde cada nó da rede *Mesh* possui um sensor conectado e este sensor faz a

Figura 13 – Solução Desenvolvida e sua organização quanto à IoT.



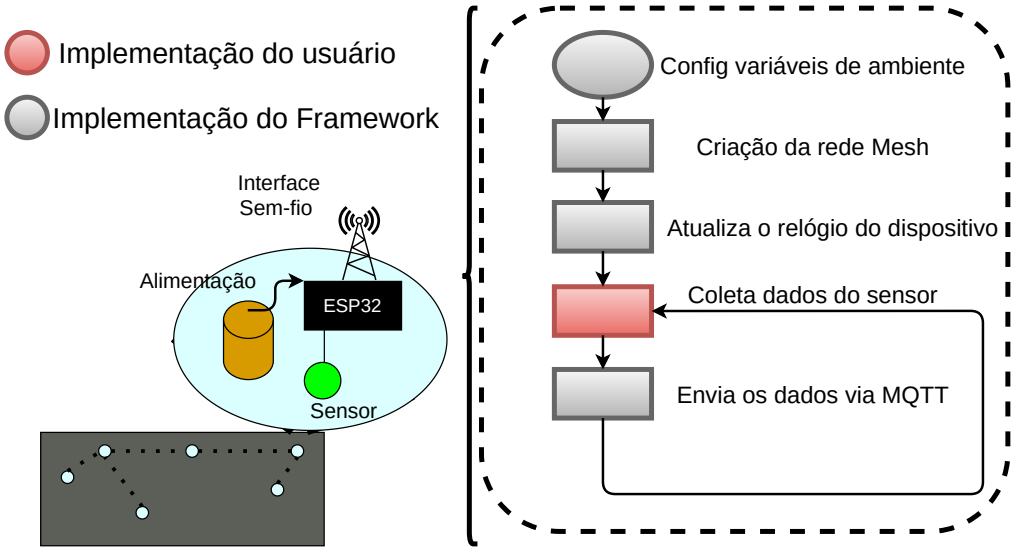
Fonte: Autoral.

coleta de um parâmetro abstrato, representando apenas a ativação ou não deste sensor. A sua funcionalidade ainda não é de interesse a este ponto no desenvolvimento do trabalho. Neste cenário proposto, os dispositivos são dispersos sob uma área, cada nó possuindo um destes sensores. Eles são conectados então à rede *Mesh*, mas apenas o nó raiz (que é escolhido de maneira automática no *framework*) é conectado a internet. Cada um destes dispositivos possui:

- O microcontrolador ESP32: O ponto central do projeto, que carrega a implementação do programa do usuário, assim como o do *framework*;
- Fonte de Alimentação: São diversos os métodos utilizados para alimentar um sistema IoT, sendo o mais comum o uso de uma bateria, mas também ser utilizados painéis fotovoltaicos ou uma alimentação externa. Neste trabalho foi considerado um sistema de alimentação externo;
- Sensor: Responsável por aferir grandes físicas. Neste trabalho, este sensor começa sendo apenas uma variável gerada internamente pelo dispositivo porém, conforme o avançar do trabalho, será substituído por diferentes periféricos como: Sensor de Temperatura e Módulo GPS.

O objetivo principal da aplicação é prover, periodicamente, as informações coletadas pela rede de sensores para um sistema de supervisão superior e, dependendo da aplicação especifica este sistema processaria os dados de acordo. Para implementar este comportamento, o fluxo do código da Figura 14 foi implementado, sendo composto pela interação de funções definidas pelo *framework* e definidas pelo usuário em si. Nesta aplicação desenvolvida, o *framework* é utilizado para configurar e organizar a rede *Mesh*, atualizar o relógio local de cada

Figura 14 – Diagrama da aplicação teste.



Fonte: Autoral.

ESP32 e transmitir dados através do protocolo MQTT. Após a transmissão, o ESP32 entraria em um estado de espera por um tempo determinado fixo e, ao termino deste intervalo, repete a rotina descrita. Os dados que serão transmitidos pelo MQTT são gerados a partir do código desenvolvido pelo usuário, sendo este responsável por ler o estado do sensor de presença e verificar suas transições.

A aplicação desenvolvida passou por diferentes estágios, sendo que em cada um deles o *software* embarcado desenvolvido se propôs a refletir o cenário real, porém o escopo desta primeira parte foi delimitada para abranger o desenvolvimento do software, sem desenvolver nenhum hardware específico para a aplicação. Com isto em mente, em cada um destes estágios, a aplicação emulou as características e peculiaridades de modo a melhor se modelar de acordo com o cenário real.

Já no primeiro estágio do processo de desenvolver o *software* proveniente do usuário e relacioná-la ao sistema proposto, alguns desafios se revelaram. O primeiro deles, relacionado à implementação da rede *Mesh*, consiste na necessidade de, sempre que for necessário utilizar um protocolo de rede baseado na pilha TCP/IP, ser necessário possuir uma conexão lógica entre o nó requisitante e o nó raiz, causando, em cenários onde nem todos os nós estejam ligados simultaneamente, ser necessário garantir que o nó raiz (no mínimo) esteja ligado quando algum dos outros nós necessite utilizar algum destes protocolos.

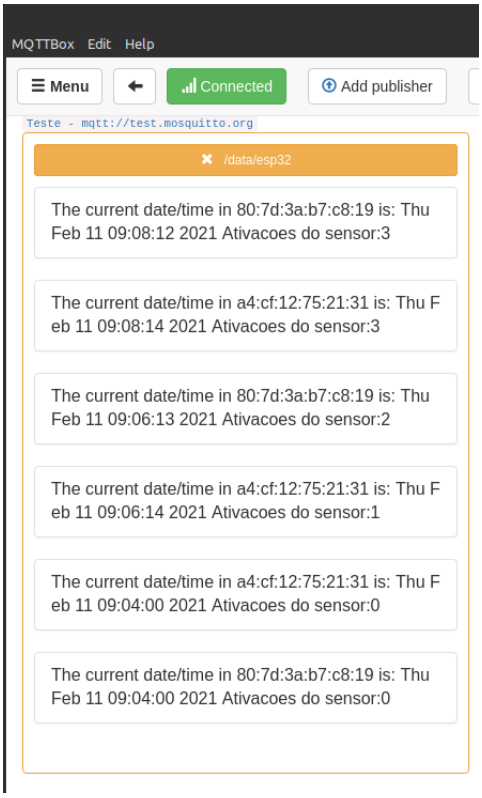
Para garantir isso, no código desenvolvido pelo usuário, utilizou-se do protocolo SNTP para atualizar e sincronizar os relógios entre todos os nós que forem participar da rede *Mesh* e definindo um intervalo fixo de tempo no qual todos os nós da rede devem iniciar a rotina de envio de dados. O primeiro teste foi feito então utilizando-se dois ESP32, com um pino GPIO (entrada e saída de propósito geral ou, do inglês, *General Purpose Input/Output*), configurado como responsável incrementar o valor fictício de ativações do sensor. Cada vez que este botão era pressionado, o dispositivo então incrementava um contador interno. A cada intervalo fixo (e sincronizado) de 2 minutos, ambos os microcontroladores transmitiam via MQTT o número de vezes no qual este botão havia sido pressionado. Utilizando-se o cliente MQTTBox, foram registradas algumas amostras dos pacotes transmitidos.

A partir destas amostras, mostradas na Figura 15, é possível perceber que a sincronia entre os ESP32 possui um erro de, no máximo, 2 segundos, o que garante que todos os nós estarão transmitindo simultaneamente. Este primeiro teste valida então a capacidade do sistema de transmitir ciclicamente alguma informação de maneira síncrona entre todos os seus nós.

4.2 Interação entre aplicação, *Framework* e Plataforma Web

O passo seguinte na validação do *framework* consistiu-se da emulação de implantação de uma infraestrutura com diversos dispositivos geograficamente dispersos, utilizando-se como exemplo o arranjo da Figura 14. Por não possuir uma fonte de alimentação embarcada ao dispositivo, os dispositivos foram montados em um ambiente interno, mas ainda assim mantendo uma certa distância entre os nós, para que o alcance do sinal entre eles e com o roteador pu-

Figura 15 – Mensagens recebidas através do cliente MQTT.



Fonte: Autoral.

desse variar, permitindo que diferentes organizações fossem possíveis quando a rede *Mesh* se formasse.

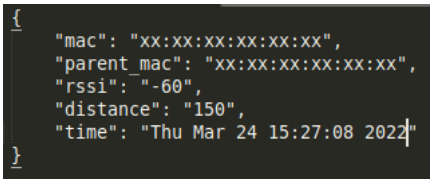
Como é mostrado na Figura 15, no primeiro estágio dos testes da aplicação, os dados eram enviados como um texto plano e interpretados apenas visualmente. Porém, em um sistema real, estes dados seriam interpretados de maneira automatizada, necessitando assim que estas mensagens possuam algum tipo de padronização para que possam ser facilmente interpretadas por um sistema computacional.

Para isto, o formato de mensagem JSON (*JavaScript Object Notation*) foi escolhido, por ser um padrão leve, livre e bastante difundido independente da plataforma. Para compor o corpo desta mensagem, foram escolhidas informações interpretadas como importantes para a identificação do dispositivo remetente e os dados coletados por este.

Neste ponto do desenvolvimento do trabalho, essas informações são: o endereço MAC do dispositivo, o endereço MAC do dispositivo pai a ele, o nível do sinal entre estes dois dispositivos (rssi), um campo com valor variando entre 0 e 300 (dado gerado aleatoriamente nesta versão, que origina-se do pseudo-sensor utilizado anteriormente) e a data do relógio de tempo real do dispositivo. A Figura 16 mostra um modelo de um pacote de dados utilizados neste teste.

Este pacote então é enviado (utilizando-se do protocolo MQTT) para uma plataforma online, responsável por processar e exibir dos dados recebidos.

Figura 16 – Modelo utilizado para a comunicação.



Fonte: Autoral.

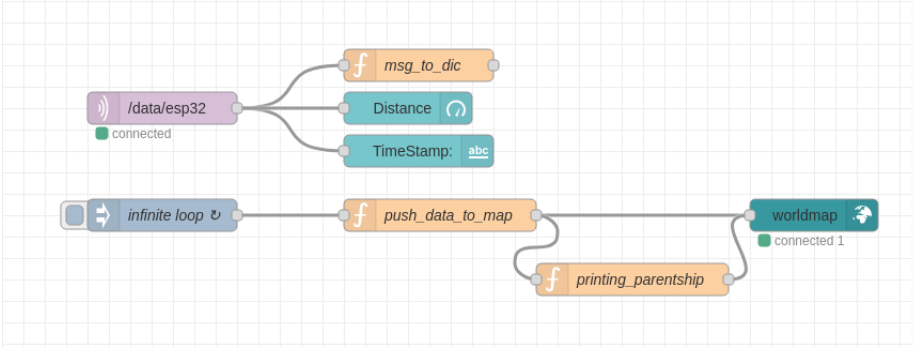
Para o desenvolvimento da plataforma, foi utilizado o Node-RED, que é uma ferramenta de programação para unir dispositivos, API's e hardware em uma aplicação, sendo bastante utilizado em aplicações relacionadas à Internet das Coisas (OPENJS, 2022). Esta ferramenta facilita o desenvolvimento de uma plataforma capaz de receber os dados vindos dos sensores, os processar e então exibi-los através de uma interface para usuários. O desenvolvimento da aplicação é feito a partir de blocos chamados de *nodes*, estes *nodes* são compostos por blocos relacionados à protocolos de rede, *parsers* (JSON, por exemplo) e, até mesmo elementos de *dashboard*. Blocos personalizados podem ser desenvolvidos utilizando a linguagem de programação *JavaScript*.

A plataforma desenvolvida possui três funcionalidades principais: Mostrar as informações características do dispositivo, computar os dados coletados e apresentá-los ao usuário e relacionar cada dispositivo a uma localização física única. Contextualizando a descrição genérica, com a aplicação de estacionamento inteligente, o valor de rssi e *timestamp* seriam mostrados sem modificações, os dados de distância são processados para então identificar a ocupação ou não de uma vaga e os dados de *parent_mac* são utilizados (em conjunto com um dado de localização que não é de origem do dispositivo IoT em si) para gerar uma informação de posição geográfica única e a relação parental entre os dispositivos que estão comunicando. Associando as objetivos gerais da ferramenta, com as funcionalidades a serem desenvolvidas e é possível associar a essa plataforma web o processamento dos dados à Camada de Arquitetura e o interface do usuário a estes dados processados à Camada de Aplicação. Essa relação é representada na Figura 13.

A Figura 17 mostra a primeira implementação da lógica de processamento dos dados vindos da rede de sensores no *Node-RED*. O primeiro bloco (em roxo) é encarregado de se inscrever no tópico no qual a rede Mesh irá publicar, recebendo assim os dados dos dispositivos. Os blocos em azul são responsáveis por mostrar os dados recebidos extraídos da mensagem em uma lista que também contém os dados recebidos anteriormente. O fluxo inferior é responsável por, periodicamente, varrer essa lista e traçar, em um mapa (também localizado na interface de usuário), os dispositivos em uma posição geográfica preestabelecida, além de desenhar uma ligação referente as relações parentais deste nó, conectando o nó e seu pai através de uma linha.

Com os dados enviados pelos dispositivos e a versão mostrada do sistema supervisório já é possível extrair uma relação de como os nós estão conectados entre eles a partir dos macs dos campos *parent_mac* e *mac* e, em conjunto com o nível do sinal *rssi*, é possível estimar se

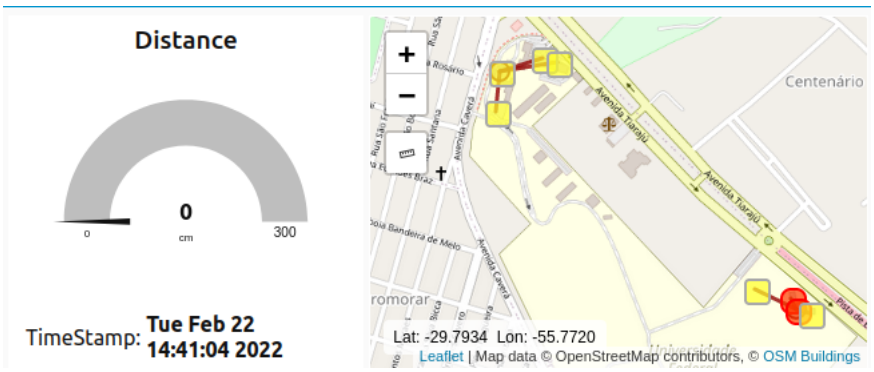
Figura 17 – Plataforma.



Fonte: Autoral.

os nós terão uma boa comunicação entre si. Um aspecto que é ausente na interpretação destes dados é a falta de um parâmetro que associe cada dispositivo à sua localização geográfica especifica, uma vez que o espalhamento dos pontos é um aspecto comum em redes IoT e ainda mais importante quando no contexto de redes *Mesh*. Para solucionar essa deficiência, cada mac foi então associado a uma localização geográfica predefinida e, neste ponto do desenvolvimento do trabalho, essa posição foi definida como estática. Cada vez que um pacote é recebido pelo servidor, as informações deste dispositivo são colocados em um mapa conforme a posição relativa a ele. A interface para o usuário ao sistema aqui descrito é mostrado na Figura 18 e tem, em seu lado direito, cada um dos dispositivos sendo representado por um ponto colorido.

Figura 18 – Primeira versão da interface do Usuário.



Fonte: Autoral.

Neste ponto do projeto, a plataforma desenvolvida é capaz de representar visualmente, em tempo real, as relações organizacionais que a rede de dispositivos possuí, permitindo testar a habilidade do *framework* de organizar dinamicamente os seus nós. Para testar isso, quatro nós foram dispostos de modo que o nível do sinal entre nós seja bem parecidos, fazendo que a rede possa se organizar de maneiras diferentes mesmo sem a mudança física da posição do nó. A Figura 19 mostra quatro organizações diferentes que puderam ser observadas. O círculo em volta do nó condiz com sua situação de nó raiz da rede e, além disso, as posições geográficas da

Figura 19 – Rede Mesh reorganizando.

Fonte: Autoral.

Figura 19 não condizem com a posição real dos sensores, mas foram escolhidas de tal maneira para uma melhor visualização das possíveis organizações.

4.3 Implementação da Camada de Dispositivos: Conclusão da solução IoT

Considerando a definição anteriormente feita em relação as camadas que uma aplicação de Internet das Coisas pode ser definida, podemos considerar que, a partir deste ponto do trabalho, as Camadas de Rede, Arquitetura e de Aplicação foram validadas. A criação e gerenciamento da rede Mesh foi comprovada a partir das Figuras 14 e 15 que demonstram a capacidade de enviar dados para a internet utilizando o *framework* e a capacidade de auto-organização e auto-gerenciamento é demonstrado na Figura 19. A camada de Arquitetura é demonstrada pelo servidor hospedado utilizando-se do Node-RED e mostrado pela Figura 17, responsável pelo

processamento dos dados recebidos. A exibição dos dados recebidos, posições geográficas e relações organizacionais (Figuras 18 e 19) validam a camada de Aplicação, que dá acesso aos dados a um usuário a partir de uma interface web. Por fim, é necessário desenvolver uma camada de Dispositivos para que seja possível validar a interface do *framework* com periféricos (como, por exemplo, sensores) e também, indiretamente, obter uma solução IoT completa.

Pelo fato da proposta do *framework* visar o desenvolvimento de uma rede Mesh que possa ser utilizada genericamente, para validar a compatibilidade dele com a camada de Dispositivos, é necessário que diferentes periféricos sejam utilizados. As características especificas de cada um deles, assim como o motivo de sua escolha, serão descritos posteriormente.

A rotina de execução da aplicação exemplo continua sendo a exemplificada pela Figura 14, porém, de acordo com o periférico adicionado, tem-se um tipo de dado gerado diferente. Durante o desenvolvimento desta aplicação, o *framework* se mostrou de fácil instalação em um projeto baseado no esp-idf, uma vez que pode ser instalado assim como qualquer outra biblioteca externa que o usuário desejasse utilizar. Por outro lado, devido a detalhes de implementação, bibliotecas (ou mesmo código desenvolvido pelo usuário) que faça acesso direto as interfaces sem fio do dispositivo, certamente haveriam problemas de funcionamento, uma vez que o *framework* assume o total controle destas interfaces, não fazendo nenhum teste para garantir que apenas ele tenha acesso sobre elas, ou mesmo iniba o acesso de outras partes do código em execução.

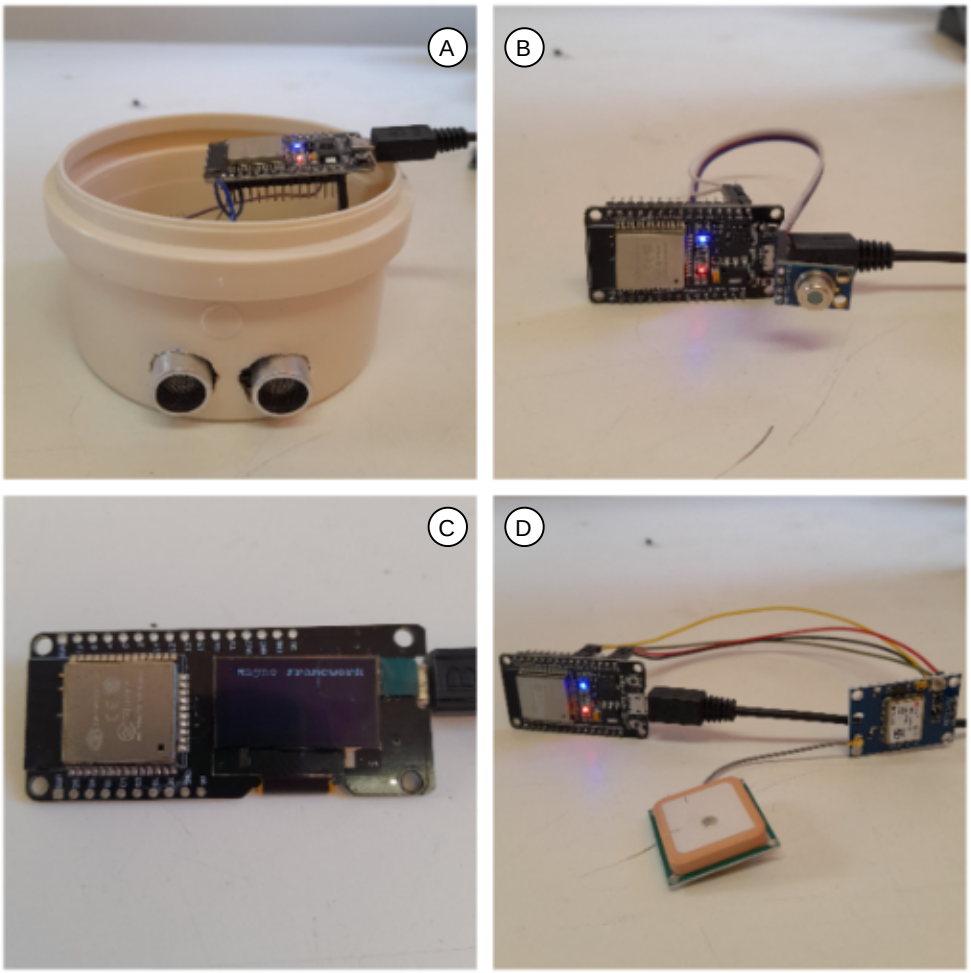
O primeiro cenário considerado foi o utilizando o sensor ultrassônico (TECHNOLOGIES, 2013). Ele é comumente utilizado para determinar a distância com um objeto e, no contexto de aplicações IoT, ele poderia ser facilmente empregado em redes de sensores em um estacionamento inteligente. Na aplicação de *software* desenvolvida, este sensor substituiu o pseudo-sensor codificado para as etapas anteriores, já que a faixa de valores coletada por eles é idêntica. O dispositivo é mostrado na Figura 20-A.

O sensor de temperatura (MELEXIS, 2019) provê tanto dados da temperatura, quanto do ambiente no qual este sensor se encontra. Diversas aplicações dependem do monitoramento da temperatura de objetos (motores, por exemplo) ou câmaras frias. No código desta aplicação, um novo campo do pacote JSON (Figura 16) foi adicionado, relacionado diretamente a temperatura ambiente medida. É mostrado na Figura 20-B.

O terceiro cenário propõe reproduzir uma característica diferente das aplicações anteriores. Em diversas soluções IoT, a habilidade de expor certo parâmetro visualmente localmente para o usuário é extremamente importante. Então, a fim de proporcionar isso, um ESP32 com o display OLED SSD1306 (SYSTECH, 2008), foi utilizado para exibir alguma informação que seja de interesse do usuário. É mostrado na Figura 20-C.

Algumas aplicações IoT tem interesse na posição geográfica no qual o dispositivo se encontra, aspecto já discutido anteriormente neste trabalho e que levou a decisão de dispor os nós em um mapa, mostrado, pela primeira vez, na Figura 18. A tecnologia GPS foi escolhida para obter os dados da posição geográfica, utilizando-se da solução da empresa u-blox (UBLOX,

Figura 20 – Sensores utilizados.

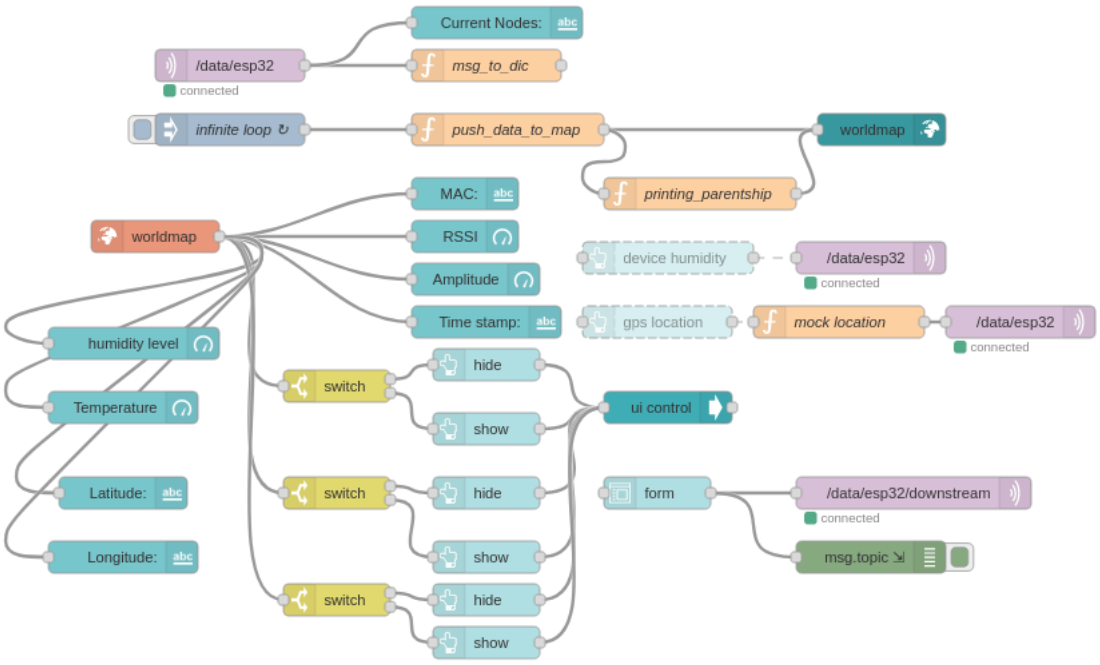


Fonte: Autoral.

2011). Anteriormente, a posição do dispositivo era escolhida e fixada previamente no servidor hospedado Node-RED porém, agora, essa informação é gerada pelo dispositivo e anexada no pacote JSON. Consequentemente, a localização no mapa é atualizada dinamicamente de acordo com este JSON. A Figura 20-D mostra o dispositivo. A versão final do aplicação desenvolvida para os dispositivos pode ser encontrada na pasta exemplos do repositório encontrado em (MAIA, 2022).

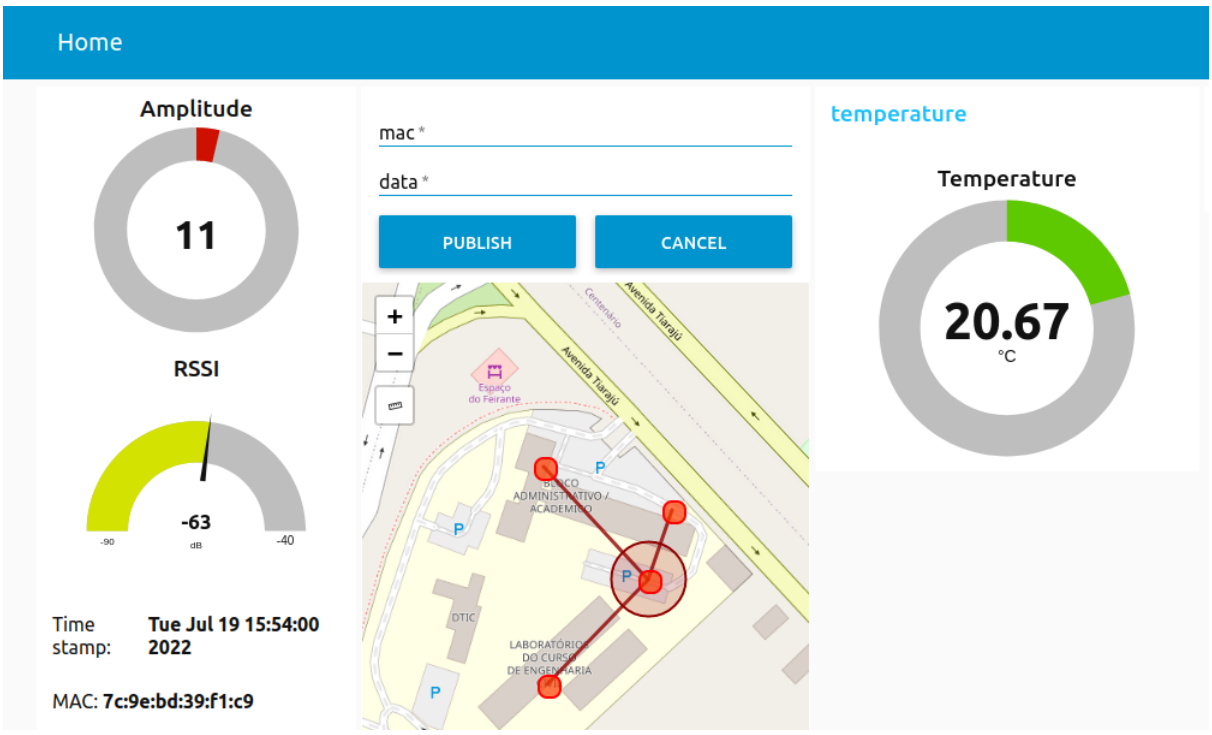
As alterações feitas na camada de dispositivos fez com que os dados enviados provenientes dos nós não sejam sempre os da Figura 16, variando de acordo com o tipo de periférico conectado a ele. Para solucionar isto, foi implementada a lógica de processamento da Figura 21, que processa condicionalmente de acordo com quais dados o JSON recebido possui, proporcionando a capacidade de uma interface que se ajusta dinamicamente conforme necessário. Como o dispositivo da Figura 20-C requer uma comunicação bidirecional, também foi adicionada essa funcionalidade. As Figuras 22 e 23 mostram a interface do usuário após as mudanças citadas, sendo que as informações mostradas conforme o nó selecionado.

Figura 21 – Back-End da versão final da plataforma.



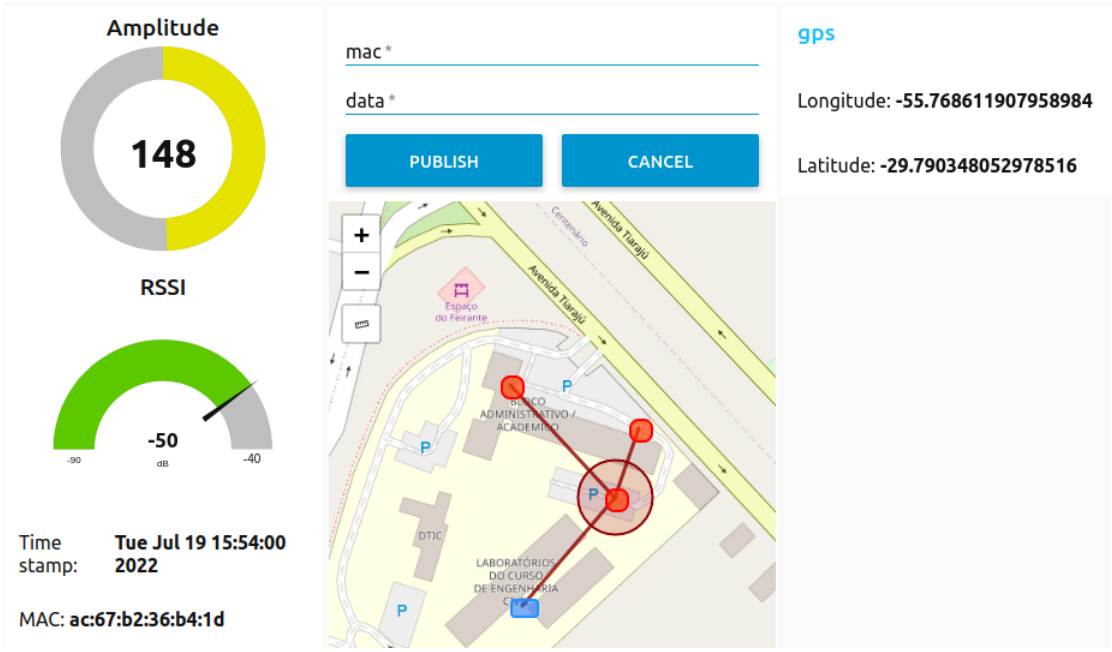
Fonte: Autoral.

Figura 22 – Plataforma com dados temperatura.



Fonte: Autoral.

Figura 23 – Plataforma com dados GPS.



Fonte: Autoral.

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho partiu da descrição de conceitos de IoT e redes *Mesh* com objetivo de instruir o leitor antes de apresentar o trabalho desenvolvido em si. O trabalho desenvolvido consistiu do desenvolvimento de um *framework* para a configuração e gerenciamento de redes *Mesh* em ambientes IoT. Além disso, o *framework* também disponibilizou a todos os nós da rede (através de um esquema de encaminhamento de protocolo) protocolos baseados na pilha TCP/IP.

A fim de validar o funcionamento do *framework*, o *software* embarcado para alguns dispositivos foi desenvolvido, sendo que estes nós são interligados pela rede *Mesh* e são conectados a periféricos que aferem grandezas físicas, sendo estes dados transmitidos via MQTT. Para processar os dados enviados, uma aplicação web foi desenvolvida com base no Node-RED. Durante toda a seção relacionada a validação do *framework*, os dispositivos foram atrelados a posições geográficas, levantando também a questão do real alcance que a rede *Mesh* pode cobrir.

Levando em consideração o conceito dos diferentes paradigmas de programação, o *framework* se mostrou intuitivo quando utilizado com a programação com um loop de controle simples, que se encaixa perfeitamente com as aplicações desenvolvidas (independente dos periféricos). Quanto a cenários multitarefas (possibilidade viável, já que o ESP32 utiliza o FreeRTOS), a implementação do *framework* pode, muitas vezes, interferir com as intenções do programador, devido a implementações internas do próprio *framework*. Porém, como o *framework* possui código aberto, usuários em nível avançados podem trabalhar com implementações dentro das funções implementadas no *framework* proposto.

Como solução proposta para trabalhos futuros, a analise de melhores métodos de sincronização, além de uma exposição, para desenvolvedores avançados, aos parâmetros de agendamento das tarefas internas ao *framework* é de grande importância. Além disso, apesar da transmissão de dados já ter sido testada e validada com sucesso, certas aplicações necessitam que a infraestrutura tolere um alto volume de mensagens. Tendo isso em mente, a análise do desempenho do *framework* e latência dos dados podem ser mensuradas em relação a carga da rede.

Ao longo do desenvolvimento deste trabalho alguns artigos foram publicados em eventos acadêmicos. Entre eles destacam-se:

- MAIA, M.; COMPASSI SEVERO, L. FRAMEWORK PARA REDES MESH COM APLICAÇÃO EM ESTACIONAMENTO INTELIGENTE. **Anais do Salão Internacional de Ensino, Pesquisa e Extensão**, v. 12, n. 2, 4 dez. 2020.;
- MAIA, M.; COMPASSI SEVERO, L. Development of a Framework to Automate the Building and Managing Processes of Wireless Mesh Networks in IoT Environments. **26° IBERCHIP Workshop**, 2021.

REFERÊNCIAS

AL-FUQAHA, A. et al. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE communications surveys & tutorials**, IEEE, v. 17, n. 4, p. 2347–2376, 2015.

CANINOS. **Labrador 64 bits**. 2022. ⟨https://caninosloucos.org/pt/labrador-64-pt/⟩.

CHETTRI, L.; BERA, R. A comprehensive survey on internet of things (iot) toward 5g wireless systems. **IEEE Internet of Things Journal**, IEEE, v. 7, n. 1, p. 16–32, 2019.

ESPRESSIF. **Espressif products**. 2021. ⟨https://www.espressif.com/en/products/socs/esp32⟩.

ESPRESSIF. **Espressif products**. 2021. ⟨https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp-wifi-mesh.html⟩.

FAROOQ, M. U. et al. A review on internet of things (iot). **International journal of computer applications**, Foundation of Computer Science, v. 113, n. 1, p. 1–7, 2015.

FELIZ, M. F. **What is Bare Metal Programming**. 2022. ⟨https://www.liquidweb.com/kb/what-is-bare-metal-programming/⟩.

FOUNDATION raspberry pi. **Computing for everybody**. 2022. ⟨https://www.raspberrypi.com/⟩.

GUPTA, R.; GUPTA, R. Abc of internet of things: Advancements, benefits, challenges, enablers and facilities of iot. In: IEEE. **2016 Symposium on Colossal Data Analysis and Networking (CDAN)**. [S.l.], 2016. p. 1–5.

KHAN, R. et al. Future internet: the internet of things architecture, possible applications and key challenges. In: IEEE. **2012 10th international conference on frontiers of information technology**. [S.l.], 2012. p. 257–260.

LI, S.; XU, L. D.; ZHAO, S. The internet of things: a survey. **Information Systems Frontiers**, Springer, v. 17, n. 2, p. 243–259, 2015.

LIU, Y. et al. Wireless mesh networks in iot networks. In: IEEE. **2017 International workshop on electromagnetics: applications and student innovation competition**. [S.l.], 2017. p. 183–185.

LUTKEVICH, B. **Embedded Operating Systems**. 2022. ⟨https://www.techtarget.com/iotagenda/definition/embedded-operating-system⟩.

MAIA, M. **Mesh IoT Framework**. 2022. ⟨https://github.com/wireless-hazard/iot-mesh-framework⟩.

MELEXIS. **MLX90614 family - Datasheet Single and Dual Zone Infra Red Thermometer in TO-39**. 2019. ⟨https://media.melexis.com/-/media/files/documents/datasheets/mlx90614-datasheet-melexis.pdf/⟩.

MQTT.ORG. **MQTT: The Standard for IoT Messaging**. 2022. ⟨https://mqtt.org/⟩.

NORDIC. **Explore our product portfolio**. 2022. ⟨https://www.nordicsemi.com/Products⟩.

OPENJS. **Node-RED. Low-code programming for event-driven applications**. 2022. ⟨https://nodered.org/⟩.

PALATTELLA, M. R. et al. Internet of things in the 5g era: Enablers, architecture, and business models. **IEEE journal on selected areas in communications**, IEEE, v. 34, n. 3, p. 510–527, 2016.

RAIN, D. **O que é IoT?** 2022. ⟨https://www.datarain.com.br/blog/o-que-e-iot/⟩.

RAZZAQUE, M. A. et al. Middleware for internet of things: a survey. **IEEE Internet of things journal**, IEEE, v. 3, n. 1, p. 70–95, 2015.

SIEWERT, J. P. S. System resources. In: **Real-Time Embedded Components and Systems with Linux and RTOS**. [S.l.]: Mercury Learning and Information, 2016. p. 33–70.

SLANT. **What is the best alternative to esp32**. 2022. ⟨https://www.slant.co/options/36414/alternatives/~esp32-alternatives⟩.

ST. **STM32 32-bit Arm Cortex MCUs**. 2022. ⟨https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html#overview⟩.

SYSTECH, S. **SSD1306 - Advance Information**. 2008. ⟨https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf⟩.

TAN, S.-L.; ANH, T. N. B. Real-time operating system (rtos) for small (16-bit) microcontroller. In: IEEE. **2009 IEEE 13th International Symposium on Consumer Electronics**. [S.l.], 2009. p. 1007–1011.

TECHNOLOGIES, C. **User's Manual: HC-SR04**. 2013. ⟨https://cdn.awsli.com.br/945/945993/arquivos/HCSR04.pdf⟩.

UBLOX. **NEO-6 - u-blox 6 GPS Modules**. 2011. ⟨https://content.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf⟩.

APÊNDICE A – CÓDIGO APLICAÇÃO SIMPLES USANDO O FRAMEWORK

O trecho seguinte demonstra como utilizar o *framework* de maneira simples:

```
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "mesh_framework.h"

static uint8_t rx_mensagem[180] = {0};

void app_main(void) {
  meshf_init(); //Init Mesh Mandatory Configurations
  //Waits up to 45 seconds until the mesh is built
  esp_err_t mesh_on = meshf_start(45000/portTICK_PERIOD_MS);
  //Init the Mesh Network itself
  if (mesh_on != ESP_OK){
    //Error handling in case the Network is not able
    //to the built
  }
  meshf_rx(rx_mensagem);
}
```

Caso a função *mesh_on* retorne o valor *ESP_OK*, o desenvolvedor pode utilizar a infraestrutura, assim como utilizar as funcionalidades dependentes da internet. Um exemplo básico é mostrado a seguir:

```
//An example on how to use publish a message using MQTT
const char *payload = "I am a pretty important message";
const char *mqtt_topic = "/iot/esp32";
meshf_start_mqtt(); //Connects to MQTT's Broker
//Publishes the payload message in the mqtt_topic
esp_err_t pub_err =
    meshf_mqtt_publish(mqtt_topic, strlen(mqtt_topic),
    payload, strlen(payload));
if (pub_err != ESP_OK){
  //Message couldn't be published!
}
```

Exemplos mais elaborados, documentação quanto ao uso das funcionalidades e o código fonte do *framework* podem ser encontrados no repositório do projeto (MAIA, 2022).