POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Matter over Thread for Smart Homes

Licenciatura em Engenharia Informática

Tiago Ramos Santos

Pedro Moreira Lopes

Leiria, julho de 2024

POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Matter over Thread for Smart Homes

Licenciatura em Engenharia Informática

Tiago Ramos Santos

Pedro Moreira Lopes

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor Doutor João Ramos, do Professor Doutor David Safadinho, do Professor Doutor António Pereira, do Professor Doutor Daniel Alexander Lopes Fuentes e do Professor Doutor Luís Alexandre Lopes Frazão.

Leiria, julho de 2024

Agradecimentos

O desenvolvimento deste projeto provém do trabalho árduo e consistente por parte de nós, enquanto grupo. Esta consistência, empenho e dedicação derivam de fatores externos, especialmente dos nossos amigos e familiares, aos quais expressamos o nosso profundo agradecimento.

Gostaríamos de manifestar a nossa sincera gratidão aos professores orientadores João Ramos, David Safadinho e António Pereira por toda a ajuda fornecida ao longo do projeto, por toda a dedicação, e em especial há motivação que nos deram semana após semana mesmo nos momentos mais difíceis e desafiadores.

Por fim, queremos agradecer ao Instituto Politécnico de Leiria e à Escola Superior de Tecnologia e Gestão por nos terem fornecido todos os recursos e suporte necessário para a realização deste projeto.

Resumo

O presente projeto aborda os desafios enfrentados na universalização da comunicação entre dispositivos *IoT* (Internet das Coisas) para casas inteligentes provenientes de diferentes fabricantes e que operam em ecossistemas distintos, e o elevado consumo de energia associado ao uso do protocolo *Wi-Fi* nesses ambientes.

Com o surgimento dos softwares de automação doméstica, estes problemas escalaram ainda mais, e protocolos como o *Zigbee*, o *BLE* e o *Z-Wave*, começaram a ser utilizados para integrar e controlar dispositivos *IoT*. No entanto, estes protocolos não solucionaram a dificuldade de interoperabilidade entre dispositivos de diferentes fabricantes, e embora alguns tenham um consumo de energia mais reduzido que o *Wi-Fi*, apresentam outros problemas como pontos únicos de falha na rede, baixa taxa de transmissão de dados, e pouco alcance.

Diante destes desafios e da falta de soluções eficazes recentemente foram desenvolvidos dois protocolos designados por *Matter* e por *Thread*. O protocolo *Matter* visa universalizar a comunicação entre dispositivos *IoT* de diferentes fabricantes e melhorar a interoperabilidade entre os diferentes ecossistemas, enquanto o protocolo *Thread*, é um protocolo de comunicação que foi criado com foco para ambientes de casas inteligentes, destacando-se pela sua eficiência energética e por reduzir pontos únicos de falha (não há perca de comunicação com os dispositivos na rede).

Inicialmente, realizámos uma pesquisa detalhada sobre ambos os protocolos para adquirirmos conhecimento e criarmos a documentação necessária para o desenvolvimento da nossa solução. Essa pesquisa permitiu o desenvolvimento de vários cenários práticos. O primeiro cenário prático consistiu na integração e controlo de um LED, e posteriormente, um dispositivo composto por duas tomadas e dois interruptores. Para ambas as soluções, foram desenvolvidos primeiramente protótipos com *Matter over Wi-Fi* e posteriormente com *Matter over Thread*.

Para a implementação do cenário proposto do dispositivo com duas tomadas e dois interruptores foram utilizados os seguintes componentes: um *ESP32-C6*, duas tomadas, dois interruptores, dois relés, um *mini pc*, um adaptador *Sonoff USB ZBDongle-E* e um router *Mikrotik*. O adaptador *Sonoff USB* foi responsável por toda a rede *Thread*, o router *Mikrotik*

foi responsável pela rede *Wi-Fi* e o *mini pc* hospedou a plataforma *Home Assistant* para possibilitar a integração e o controlo da solução. Por fim, o *ESP32-C6* foi o controlador do protótipo que interliga diretamente com as tomadas e os interruptores.

Durante todos os testes realizados e os resultados obtidos, constatou-se que, apesar dos muitos desafios enfrentados devido à recente introdução dos protocolos, conseguimos superá-los e alcançar um protótipo funcional com os vários protocolos. Identificámos e apresentámos soluções para todos os problemas encontrados no desenvolvimento dos cenários tanto em *Matter over Wi-Fi* como em *Matter over Thread*, tornando este processo mais simples e intuitivo. Demos um passo significativo na evolução das *Smart Homes* e no desenvolvimento de novas soluções baseadas nestes protocolos.

Palavras-chave: Eficiência, Interoperabilidade, *IoT*, *Matter*, *Thread*, *Wi-Fi*

Abstract

This project addresses the challenges faced in universalizing communication between IoT (Internet of Things) devices for smart homes from different manufacturers and from distinct ecosystems, as well as the high energy consumption associated with the use of the Wi-Fi protocol in these environments.

With the appearance of home automation software, these problems have escalated, and protocols such as Zigbee, BLE, and Z-Wave begun to be used to integrate and control IoT devices. However, these protocols have not solved the interoperability issues between devices from different manufacturers, and although some have lower energy consumption than Wi-Fi, they present other problems.

Given these challenges and the lack of effective solutions, two protocols have been developed recently: Matter and Thread. The Matter protocol aims to universalize communication between IoT devices from different manufacturers and improve interoperability between different ecosystems, while the Thread protocol is a communication protocol specifically created for smart home environments, distinguished by its energy efficiency and lack of single points of failure (there is no loss of communication with devices in the network).

Initially, we conducted a detailed study on both protocols to acquire knowledge and create the necessary documentation for the development of our scenario. This research allowed the development of various practical scenarios. First, we integrated and controlled an LED, and secondly, a device consisting of two sockets and two switches (implementing the solutions first in Matter over Wi-Fi and then in Matter over Thread).

For the implementation of the proposed scenario of the device with two sockets and two switches, the following components were used: an ESP32-C6, two sockets, two switches, two relays, a mini PC, a Sonoff USB ZBDongle-E adapter, and a Mikrotik router. The Sonoff USB adapter was responsible for the entire Thread network, the Mikrotik router managed the Wi-Fi network, and the mini PC hosted the Home Assistant platform to enable the integration and control of the solution. Finally, the ESP32-C6 acted as the controller of the prototype, directly interfacing with the sockets and switches.

Throughout the tests conducted and the results obtained, we found that despite facing many challenges due to the recent introduction of the protocols, we successfully overcame them and achieved a functional prototype with the various protocols. We identified and presented solutions to all the problems encountered in developing the scenarios in Matter over Wi-Fi and Matter over Thread, making this process simpler and more intuitive. We took a significant step forward in the evolution of Smart Homes and in developing new solutions based on these protocols.

Keywords: Efficiency, Interoperability, IoT, Matter, Thread, Wi-Fi

Índice

Lista de Figuras

Lista de tabelas

Lista de siglas e acrónimos

ACK	Acknowledge
AES-128	Advanced Encryption Standard 128
App	Application
BIOS	Basic Input/Output System
BLE	Bluetooth Low Energy
CO2	Carbon Dioxide
DHCP	Dynamic Host Configuration Protocol
DHCPv6	Dynamic Host Configuration Protocol version 6
DNS	Domain Name System~
DNS-SD	Domain Name Service – Service Discovery
EFI	Extensible Firmware Interface
GND	Ground
GHz	Gigahertz
GUA	Global Unicast Address
gpio	general purpose input/output
HVAC	Heating, Ventilating and Air Conditioning
IEEE	Institute of Electrical and Electronics Engineers
IN	Input
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
iOS	iPhone Operating System
IoT	Internet of Things
IUT-T	International Telecommunication Union Telecommunication Standardization Sector
Kbps	Kilobits per second
MAC	Media Access Control
Mbps	Megabits per second
MHz	Megahertz
mDNS	Multicast Domain Name System
mW	Megawatt
Multi-PAN	Multi-Personal Area Network

PHY	Physical
Pro	Professional
QRCode	Quick Response code
REED	Router-Eligible End Device
RGB	Red, Green, Blue
RISC-V	Reduced Instruction Set Computer - Five
ROM	Read-Only Memory
SED	Sleepy End Device
SIG	Special Interest Group
SRAM	Static Random-Access Memory
SYN	Synchronize
SSED	Synchronized Sleepy End Device
TCP	Transmission Control Protocol
UI	User Interface
ULA	Unique Local Address
URL	Uniform Resource Locator
USB	Universal Serial Bus
VCC	Voltage Common Collector
VSCode	Visual Studio Code
WPA2	Wireless Protected Access 2
WSL	Windows Subsystem for Linux

1. Introdução

O crescimento da Internet das Coisas impulsionou a evolução das casas inteligentes, oferecendo uma maior otimização e ajuda na rotina diária das pessoas.

Inicialmente os dispositivos eram localmente controláveis, mas gradualmente, tem havido uma grande evolução neste tópico, sendo agora possível controlá-los remotamente com vários protocolos de comunicação e através de softwares de automação doméstica como o *Home Assistant*, o *Google Home*, o *HomeKit*, e entre outras plataformas.

Primeiramente, para controlar os dispositivos *IoT* remotamente começou por utilizar-se o protocolo *RFID*, e com a evolução das casas inteligentes, o *Wi-Fi* começou a ser utilizado e ainda é o protocolo mais utilizado nos dias de hoje [1]. No entanto, com a evolução dos protocolos de comunicação e a sua compatibilidade com os dispositivos *IoT*, começou também por se utilizar protocolos como o *ZigBee*, *Z-Wave* e *BLE*.

Essa evolução resultou no surgimento de diversos dispositivos *IoT* de diferentes fabricantes, acarretando dificuldades na integração desses dispositivos devido à grande diversidade de ecossistemas existentes, como a *Philips*, a *Google*, a *Apple*, entre outros, havendo uma elevada dependência de entidades e serviços externos.

Além da flexibilidade limitada na integração dos dispositivos, o protocolo de comunicação mais utilizado para os controlar é o *Wi-Fi*, que contém pontos únicos de falha (no caso de o ponto central de comunicação falhar, por norma um *router* em redes tradicionais utilizadas em casa, toda a rede falha, resultando na perca de comunicação com os dispositivos *IoT*) e um elevado consumo de energia.

Para mitigar estes problemas de compatibilidade entre dispositivos e na comunicação dos mesmos, foram recentemente desenvolvidos dois protocolos, designados por *Matter*, e por *Thread*. O *Matter* é um protocolo a nível aplicacional que é suportado por uma grande aliança de empresas, como a *Amazon*, a *Google*, a *Apple*, entre outras empresas de enorme dimensão, para ajudar a eliminar a dependência e a dificuldade de integração de equipamentos de fabricantes diferentes, universalizando este conceito. O *Thread* é um protocolo de comunicação que foi criado com o objetivo de melhorar a comunicação entre

dispositivos *IoT* em casas inteligentes, devido a não conter pontos únicos de falha, e por ser muito mais eficiente em termos de consumo de energia comparativamente ao *Wi-Fi* [2].

Dado que estes protocolos são muito recentes, ainda em desenvolvimento, e com muitas falhas por resolver, com este projeto pretendemos detalhar passo a passo todo o processo de desenvolvimento de um dispositivo compatível com *Matter over Thread*, que consiste em duas tomadas e dois interruptores. Pretendemos explorar quais os possíveis problemas que podem existir, e apresentar soluções para estes problemas. Adicionalmente, pretendemos especificar as vantagens de utilizar *Matter* para haver mais flexibilidade entre fabricantes em ambientes de casas inteligentes, e as vantagens de utilizar *Thread* para obter uma rede que não tem pontos únicos de falha, e que tem um consumo energético muito mais reduzido.

Objetivamos fazer uma pesquisa detalhada acerca deste tópico e recolher o máximo de informação possível, para que todos os temas fiquem bem esclarecidos a todos os leitores, utilizadores e futuros implementadores. Pretendemos utilizar esta pesquisa para desenvolver primeiramente um cenário em *Matter over Wi-Fi* por ser uma solução mais testada, e para perceber de forma mais aprofundada como funciona o protocolo *Matter*. Convertendo depois este cenário para *Matter over Thread*, especificando da forma mais detalhada e simplista possível o funcionamento do *Thread* e a integração entre os dois protocolos.

Como resultado pretendemos obter um estado de arte com informação detalhada, didática e informativa, obter propostas de hardware e software que possam ser utilizados em cenários em *Matter over Thread*, implementar um protótipo com duas tomadas e dois interruptores em *Matter over Thread*, e documentar todos os problemas e soluções relacionadas com o desenvolvimento de *Matter over Thread*.

1.1.Estrutura do documento

O documento está dividido nos seguintes capítulos:

- **Capítulo 2 – Estado de Arte**: Neste capítulo abordamos os softwares de automação doméstica, os protocolos *Matter* e *Thread*, os *Thread Border Routers* e os microcontroladores *ESP32*.
- **Capítulo 3 – Solução Proposta**: Este capítulo apresenta a arquitetura da solução, a explicação da mesma e todos os requisitos da implementação.

- **Capítulo 4 – Protótipo Desenvolvido**: Dividimos este capítulo por quatro subcapítulos por termos desenvolvido quatro cenários de forma incremental que se complementam. Demonstramos o trabalho resolvido, o esquema do protótipo utilizado, e o fluxograma para cada um dos quatro cenários.
- **Capítulo 5 – Testes e Resultados Obtidos**: Neste capítulo são demonstrados os testes efetuados e os resultados obtidos relativo aos quatro cenários desenvolvidos durante a implementação do projeto.
- **Capítulo 6 - Conclusão**: Corresponde às conclusões obtidas ao longo da realização de todo o projeto.
- **Capítulo 7 – Referências Bibliográfica**: Neste capítulo encontram-se as referências bibliográficas usadas para a realização do projeto.
- **Capítulo 8 – Anexos**: Neste capítulo encontram-se os anexos relativos à pesquisa e ao desenvolvimento do projeto.

2. Estado de arte

Neste capítulo abordaremos toda a informação pesquisada que consideramos relevante e importante para o desenvolvimento do nosso projeto. Começaremos por descrever o software de automação doméstica mais adequado para utilizar em soluções de casas inteligentes. Analisaremos os protocolos de comunicação sem fio utilizados para integrar e controlar dispositivos *IoT*, para através das suas vantagens e desvantagens descobrirmos o mais indicado para soluções em casas inteligentes.

Descreveremos como funcionam os protocolos *Thread* e *Matter*, e quais os dispositivos que suportam estes protocolos. Faremos uma explicação da utilização de *Matter over Wi-Fi* e *Matter over Thread* para demonstrar as suas diferenças em termos funcionais. Por fim, analisaremos os *Thread Border Routers* (*routers* utilizados especificamente para redes *Thread*), e os diferentes microcontroladores *ESP32* por serem dispositivos versáteis e ideais para ambientes de teste.

2.1. Softwares de automação doméstica

No início, os dispositivos *IoT* eram localmente controláveis, mas com a evolução das casas inteligentes, surgiram os softwares de automação doméstica para controlar os dispositivos de forma remota. Nesta secção iremos abordar as características dos softwares de automação doméstica mais utilizados, analisando as suas vantagens e desvantagens. Concluiremos, com uma análise comparativa entre eles para identificar a melhor opção.

2.1.1. Home Assistant

O *Home Assistant* é uma plataforma *Open Source* que permite integrar dispositivos de fabricantes diferentes e controlá-los de forma centralizada. Pode ser instalado em *Raspberry Pis*, computadores (máquina virtual), ou *docker containers* [3].

Vantagens:

- Compatível com vários dispositivos
- Pode ser controlado localmente
- Oferece várias extensões

Desvantagens:

- Complexidade de configuração e manutenção

2.1.2. Google Home

O *Google Home* é uma aplicação para casas inteligentes que inclui recursos integrados para fornecer funcionalidades do *Google Assistant*. Permite conectar e controlar dispositivos domésticos inteligentes, desde que sejam compatíveis com o *Google Assistant*. Um dos aspetos mais importantes do *Google Home* é que tem suporte de voz, permitindo o controlo dos dispositivos integrados na aplicação através de comandos de voz reconhecidos pela aplicação [4].

Vantagens:

- Económico
- Suporte de comandos de voz
- Contém jogos na aplicação para entretenimento
- Pode ser personalizado
- Não é necessário fazer subscrição

Desvantagens:

- Tem compatibilidades limitadas
- Não consegue receber nem enviar sinais elétricos
- Não suporta instruções de voz remotamente

2.1.3. OpenHab

Assim como o *Home Assistant*, o *OpenHAB* é uma solução de código aberto que foi criada em 2010 para integrar vários tipos de dispositivos inteligentes e controlá-los através da interface gráfica. Pode ser instalado em *Raspberry Pis*, computadores (máquina virtual), ou *docker containers* [5].

2.1.4. Home App

A *Home App* é uma aplicação desenvolvida pela Apple para dispositivos iOS. Serve como um controlador centralizado de dispositivos que são compatíveis com a estrutura

HomeKit da Apple, e fornece automação de dispositivos conforme a necessidade do utilizador, tal como acesso remoto [6].

Vantagens:

- Integração fácil com todos os dispositivos da Apple
- A aplicação garante segurança
- Automação
- Acesso remoto

Desvantagens:

- Compatibilidade limitada com outros dispositivos que não são da Apple
- Preço elevado
- Complexidade de configuração
- Por vezes há problemas de conectividade

2.1.5. Análise comparativa dos softwares de automação doméstica

Desenvolvemos a Tabela 1 - Tabela comparativa de softwares de automação doméstica para demonstrar uma análise comparativa dos softwares de automação doméstica.

Tabela 1 - Tabela comparativa de softwares de automação doméstica

Recurso	*Home Assistant*	*Google Home*	*OpenHab*	*Home App*
Tipo de software	*Open-source*	Privados, baseado em núvem	*Open-source*	Privados, baseado em núvem
Plataformas	Windows, Linux, *MacOS*, *Docker*	iOS, Android	Windows, Linux, *MacOS*, *Docker*	iOS
Compatibilidade	Grande variedade de dispositivos e protocolos	Dispositivos compatíveis com *Google Assistant*	Grande variedade de dispositivos e protocolos	Dispositivos compatíveis com *HomeKit*
Interface	Interface gráfica, aplicações	Aplicação móvel (iOS e Android)	Interface gráfica, aplicações	Aplicação móvel (iOS)

	móveis (iOS e Android)		móveis (iOS e Android)	
Integração	Integração com serviços de terceiros e plataformas	Integração limitada a dispositivos compatíveis	Integração com serviços de terceiros e plataformas	Integração limitada a dispositivos compatíveis
Automatização	Suporte avançado para automatização e scripts	Suporte básico para rotinas e automações	Suporte avançado para automatização e scripts	Suporte básico para rotinas e automações
Preço	Gratuito	Varia (alguns recursos gratuitos, outros pagos)	Gratuito	Gratuito

Tipos de Software: O *Home Assistant* e o *OpenHAB* são soluções de código aberto, que permitem aos utilizadores maior controlo e personalização das suas redes domésticas. O *Google Home* e o *Home App* são privados e baseados em nuvem, oferecendo uma experiência mais integrada e fácil de utilizar, mas muito mais limitada ao nível de software.

Plataformas: O *Home Assistant* e o *OpenHAB* funcionam em várias plataformas, incluindo Windows, Linux, *MacOS*, *Docker* e *Raspberry Pi*, oferecendo flexibilidade na instalação. O *Google Home* está disponível em dispositivos móveis iOS e Android, e o *Home App*, é o menos versátil e flexível sendo exclusivo para dispositivos iOS,

Compatibilidade: O *Home Assistant* e o *OpenHAB* são compatíveis com uma grande variedade de dispositivos e são muito versáteis. O *Google Home* é compatível apenas com dispositivos que funcionam com o *Google Assistant*, e o *Home App* apenas com dispositivos que suportam o *Homekit*, reduzindo as capacidades de serem integrados em ambientes versáteis.

Interface: O *Home Assistant* e o *OpenHAB* oferecem interfaces web e aplicações para iOS e Android, permitindo controlo e monotorização flexíveis. O *Google Home* fornece aplicações tanto para iOS como para Android, e o *Home App* oferece aplicações exclusivas para iOS.

Integração: O *Home Assistant* e o *OpenHAB* possuem integração avançada a serviços de terceiros e de diversas plataformas, oferecendo grande personalização. O *Google Home* e o *Home App* integram-se com dispositivos compatíveis da Google e da Apple, mas são limitados em termos de integração com outros serviços ao contrário.

Automatização: O *Home Assistant* e o *OpenHAB* suportam automatizações avançadas e scripts, proporcionando um alto nível de personalização e controlo. O *Google Home* e o *Home App* oferecem suporte básico para rotinas e automações, suficientes para a maioria dos utilizadores, mas limitados em comparação com o *Home Assistant* e o *OpenHAB*.

Preço: O *Home Assistant*, *OpenHab* e o *Home App* são gratuitos. O *Google Home* oferece alguns recursos gratuitos, mas outros podem ser pagos, dependendo dos serviços que o utilizador pretende utilizar.

Através da nossa pesquisa e análise de todos os softwares de automação doméstica, concluímos que o *Home Assistant* é o software de automação doméstica mais indicado por ser gratuito, versátil numa grande variedade de ambientes domésticos, e por ser possível instalá-lo em diferentes plataformas e dispositivos. Para além disto, o aspeto mais importante, é que o *Home Assistant* tem muitas extensões úteis para cenários de casas inteligentes, facilitando o desenvolvimento de soluções com muita diversidade.

2.2.Protocolos de comunicação sem fio

Com o surgimento dos softwares de automação doméstica, os protocolos de comunicação sem fio têm vindo a evoluir cada vez mais em ambientes de casas inteligentes.

Neste subcapítulo exploraremos os protocolos de comunicação sem fio utilizados na integração e controlo de dispositivos *IoT* em casas inteligentes. Analisaremos todas as vantagens e desvantagens de todos os protocolos, para chegarmos a uma conclusão sobre qual é o protocolo de comunicação mais adequado nestes ambientes.

2.2.1. Wi-Fi

Wi-Fi é uma tecnologia que permite criar uma rede para que os dispositivos com compatibilidade com este protocolo consigam comunicar entre si sem fios. A comunicação utiliza um ponto central para conectar os dispositivos, é transmitida através de ondas de rádio e funciona em diferentes bandas de frequência como 2.4GHz, 5GHz e 6GHz (difere a velocidade, alcance, interferência e compatibilidade). O *Wi-Fi* permite acesso à internet,

logo, desde que o ponto central tenha acesso à internet, todos os dispositivos conectados ao ponto central também têm [2].

O *Wi-Fi* suporta 255 dispositivos por rede, e utiliza a topologia de rede em estrela ou em malha. Por norma em redes tradicionais utilizadas em ambientes domésticas, a rede apenas tem um ponto central de acesso, logo nestes ambientes a topologia em malha tem a mesma eficácia que a topologia em estrela, contendo por norma pontos únicos falha. Em termos de segurança o *Wi-Fi* tem vindo a evoluir, e por norma utiliza criptografia *WPA2*, mas os sistemas mais recentes já utilizam criptografia *WPA3* que é considerado muito seguro. O *Wi-Fi* tem uma taxa de transmissão de dados muito elevada, tornando-o ideal para transmitir dados em tempo real, e tem um elevado consumo de energia relativamente aos outros protocolos sem fio.

Esta análise e todos os pontos mencionados estão representados na Tabela 2 - Tabela de análise do *Wi-Fi*.

Tabela 2 - Tabela de análise do *Wi-Fi*

Parâmetros	*Wi-Fi*
Consumo de energia	500-2000*w*
Segurança	Utiliza por norma criptografia *WPA2*.
Confiabilidade	Tem pontos únicos de falha.
Escalabilidade	Suporta até 255 dispositivos por rede.
Topologia	Estrela e malha.
Aplicação principal	Projetado para acesso à Internet.
Taxa de transmissão de dados	Até 54 *Mbps*.

2.2.2. Thread

O *Thread* é um protocolo que foi desenvolvido com o intuito de ser direcionado para ser utilizado na integração e controlo de dispositivos *IoT* em ambientes de casas inteligentes. Destaca-se pelo seu baixo consumo energético, por utilizar o algoritmo de criptografia AES-128 com mecanismo contra repetições, logo é muito seguro, e especialmente por utilizar a topologia de rede em malha, que elimina os pontos únicos de falha.

Cada rede *Thread* suporta até 32 routers e 511 dispositivos, permitindo que este protocolo não seja só utilizado em ambientes domésticos, mas também em ambientes de maior escala. Este protocolo, tem uma taxa de transmissão de dados relativamente baixa, mas considerado dentro da média quando o comparamos a outros protocolos utilizados nestas soluções.

Esta análise e todos os pontos mencionados estão representados na Tabela 3 - Tabela de análise do *Thread* [2].

Tabela 3 - Tabela de análise do *Thread*

Parâmetros	*Wi-Fi*
Consumo de energia	100*w*
Segurança	Suporta mecanismos de criptografia e autenticação. Utiliza criptografia AES-128 e mecanismos contra repetições.
Confiabilidade	Não tem pontos únicos de falha.
Escalabilidade	Suporta até 32 routers e 511 dispositivos por router
Topologia	Malha
Aplicação principal	Projetado para dispositivos *IoT* de casas inteligentes
Taxa de transmissão de dados	Até 250 *kbps*.

2.2.3. Z-Wave

Z-Wave é um protocolo de comunicação sem fios, com o intuito de garantir compatibilidade com dispositivos de casas inteligentes. Opera numa frequência baixa de rádio, o que reduz significativamente o risco de interferência com outros dispositivos eletricos. Adota uma estrutura de rede em malha, na qual qualquer dispositivo conectado à rede pode atuar como um repetidor aumentando o alcance da rede, tornando-a mais eficaz e eliminando pontos únicos de falha. Tem uma taxa de dados de 100 *kbps*, e em termos de segurança utiliza o algoritmo *AES-128* [7].

2.2.4. Zigbee

Zigbee é um protocolo designado para dispositivos *IoT*. Utiliza a topologia de rede em malha, logo não tem pontos únicos de falha, tem um baixo consumo de energia, que é ideal para dispositivos *IoT* em casas inteligentes, e tem um alcance idêntico aos outros protocolos. O *Zigbee* opera na banda de frequência de 2400 MHz, que oferece 250 Kbps de taxa de transmissão de dados.

O *Zigbee* tem três tipos de dispositivos na sua rede designados por *Coordinators*, *routers* e *End Devices*. Os *Coordinators* configuram a rede e tratam do processo de entrada dos *routers* e dos *End Devices* na rede. Os *routers* fazem o roteamento entre os dispositivos na rede. Por fim, os *End Devices*, normalmente estão em modo de sono, e detetam quando os routers comunicam com eles (por norma quando o utilizar os pretende controlar).

A diferença do *Zigbee* para outros protocolos é que não é agnóstico à camada aplicacional, logo é um protocolo menos versátil e adaptável. Esta diferença está representada na Figura 1 - Pilha protocolar do *Zigbee* e do *Thread*, onde podemos observar a diferenciação da pilha protocolar do *Zigbee* e do *Thread*.

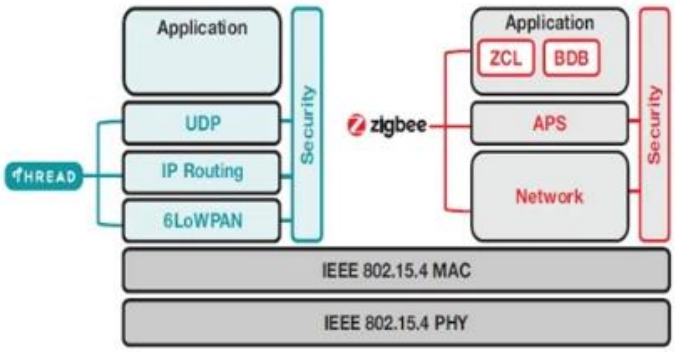

Figura 1 - Pilha protocolar do *Zigbee* e do *Thread*

2.2.5. Bluetooth Low Energy

O *Bluetooth Low Energy* é um protocolo que derivou do *Bluetooth*, e que a grande diferença entre ambos, é que o *BLE* ter um consumo de energia muito mais reduzido que o *Bluetooth* tradicional. O baixo consumo de energia deriva de os dispositivos conectados por *BLE* estarem sempre em modo de sono, e apenas se ligam por poucos milissegundos quando recebem uma transferência de dados [8].Opera na frequência de 2.4 *GHz*, oferece uma taxa de transmissão de dados de 125 *Kbps* até *2Mbps*, e tem um alcance interior de cerca de 10 metros [9].

A topologia de rede do BLE, tanto pode ser ponto a ponto ou em malha. Na comunicação ponto a ponto, há uma comunicação direta entre um dispositivo controlador com um ou mais dispositivos finais, mas o dispositivo final apenas pode comunicar com o controlador. Logo, na topologia ponto a ponto, por exemplo, um telemóvel comunica com uma coluna e uma lâmpada, mas a coluna e a lâmpada só podem comunicar com o telemóvel. Na topologia em malha, todos os dispositivos têm comunicação entre si e não dependem de um ponto central como alguns protocolos, ou seja, quando é preciso entregar uma mensagem, esse dispositivo faz broadcast da mensagem para os restantes dispositivos da rede.

2.2.6. Análise comparativa dos protocolos de comunicação sem fio

Na Tabela 4 - Análise comparativa dos protocolos de comunicação sem fio [10] demonstramos mais detalhadamente as diferenças dos protocolos de comunicação sem fio mais utilizados em ambientes de casas inteligentes.

Tabela 4 - Análise comparativa dos protocolos de comunicação sem fio [10]

Recurso	*Wi-Fi*	*Z-Wave*	*Zigbee*	*Thread*	*BLE*
Ano do primeiro lançamento	1997	2003	2003	2015	2010
Padrão	*IEEE 802.11*	*ITU-T G-9959*	*IEEE 802.15.4*	*IEEE 802.15.4*	*Bluetooth SIG*
Banda de frequência	2.4 *GHz*	900 *MHz*	2.4 *GHz*	2.4 *GHz*	2.4 *GHz*
Alcance nominal	100m	30m-100m	10-100m	10-100m	10m

Taxa máxima de transmissão de dados	54 *Mbps*	40-100 *kbps*	250 *kbps*	250 *kbps*	125 *Kbps* a 2 *Mbps*
Topologia	Estrela e malha	Malha	Malha	Malha	Ponto a ponto ou malha
Uso de energia	Alto (0.5-2 *W*)	Baixo (10 *mW*)	Médio (100 *mW*)	Médio (100 *mW*)	Baixo/Médio (10-500*mW*)
Aliança	*Wi-Fi Alliance*	*Z-Wave Alliance*	*ZigBee*	*Thread Group*	*Bluetooth*

Ano do primeiro lançamento: O *Wi-Fi* foi o primeiro protocolo a ser lançado em 1997. Após o *Wi-Fi*, surgiu o *Z-Wave* e o *Zigbee* em 2003. O *BLE* surgiu uns anos mais tarde, em *2010*, mas foi apenas uma evolução do *Bluetooth*, que já existia e era muito utilizado. O protocolo mais recente é o *Thread*, que foi lançado em 2015, e atualmente ainda continua em estado de testes.

Padrão: Os padrões especificam padrões definidos para redes sem fio. O *Wi-Fi* usa *IEEE 802.11* que é a especificação para redes locais sem fio (*WLAN – Wireless Local Area Network*). O *Z-Wave* segue o padrão *ITU-T G-9959* que contém as especificações das camadas *MAC* e *PHY* de protocolos de curto alcance. O *ZigBee* e o *Thread* usam *IEEE 802.15.4* que define os padrões para redes de área pessoal sem fio (*WPAN – Wireless Personal Area Networks*). O *BLE* é gerido pelo *Bluetooth SIG*, que são as especificações definidas pelo *Bluetooth*.

Banda de frequência: A banda de frequência na qual cada protocolo opera é crucial para o seu funcionamento e interferência. *Wi-Fi*, *ZigBee*, *Thread* e *BLE* operam na banda de 2.4 *GHz*, enquanto o *Z-Wave* opera na banda de 900 *MHz*, o que pode resultar em menos interferência e mais facilidade em penetrar obstáculos comparando com os outros protocolos de comunicação.

Alcance Nominal: O *Wi-Fi* predomina neste fator devido a ter o maior alcance de 100 metros. O *Z-Wave*, o *Zigbee* e o *Thread* são bastante idênticos neste aspeto pelo facto que

em redes de pequena dimensão não têm muito alcance, mas em redes de maior dimensão conseguem também atingir alcances de 100 metros. O *BLE* por sua vez, tem um alcance de cerca de 10 metros.

Taxa máxima de transmissão de dados: *Wi-Fi* oferece uma alta transmissão de dados, até 54 *Mbps*, ideal para aplicações que requerem alta largura de banda, como por exemplo, transmissões de vídeo em tempo real ou câmaras de vigilância. *Z-Wave* oferece taxas de 40 a 100 *Kbps*, que por norma é pouco para a transmissão de dados de equipamentos *IoT* utilizados em casas inteligentes. *Zigbee* e *Thread* chegam a 250 Kbps, que por norma é suficiente para os dispositivos inteligentes utilizados em redes domésticas, enquanto o *BLE* oferece uma taxa de transmissão mais elevada até 2 Mbps.

Topologia: O *Wi-Fi* utiliza uma topologia em estrela ou em malha. Na topologia em estrela, todos os dispositivos se conectam a um ponto central, que é um possível problema no caso do ponto central falhar. Na rede em malha, o *Wi-Fi* também é muito dependente do ponto central (em casas inteligentes que por norma apenas têm um router), logo, a topologia em malha é mais indicada, mas pouco confiável. *Z-Wave*, *ZigBee* e *Thread* usam topologias em malha, permitindo que os dispositivos possam assumir diferentes papéis na rede no caso de haver dispositivos a falhar, aumentando a confiabilidade na rede por esta topologia eliminar possíveis pontos únicos de falha. O BLE também pode utilizar dois tipos de topologias, ponto a ponto e malha, mas a mais indicada é a topologia em malha por ser mais resiliente.

Uso de energia: *Wi-Fi* é o protocolo que consome mais energia (0.5-2 *W*), adequado para dispositivos com acesso constante a energia elétrica. O *Z-Wave* é muito eficiente com um consumo de cerca de 10 *mW*. *ZigBee* e *Thread* têm um consumo também relativamente baixo de 100 *mW*, adequado para dispositivos a bateria, e ideal para cenários em casas inteligentes. O *BLE* varia entre 10 *mW* e 500 *mW*, ou seja, é muito eficiente em termos energéticos por suportar o modo sono, que oferece uma boa combinação de baixo consumo e funcionalidade.

Em resumo, o *Thread* em geral prevalece a todos os protocolos havendo sempre algumas diferenças mais benéficas. Devido a suportar a topologia de rede em malha, não ter pontos únicos de falha, ter baixo consumo energético, prevalecer em relação aos outros protocolos em termos de segurança, ter um alcance grande tanto para redes de pequenas como de grande dimensão, e a única desvantagem de maior significância ser o protocolo ter uma baixa

transmissão de dados que pode ser problemático para o uso de câmaras de vigilância, consideramos que este protocolo seja o mais ideal para soluções de casas inteligentes.

2.3. Thread

Nesta secção, será abordado o protocolo de comunicação *Thread* (o protocolo que considerámos mais ideal para ambientes de casas inteligentes na análise feita anteriormente).

Começaremos por uma introdução ao protocolo, detalharemos quais são as suas especificações, iremos referir as características da rede em malha, e as vantagens que o protocolo *Thread* tem por utilizar esta topologia. De seguida, entramos mais em detalhe, e explicaremos a pilha protocolar do *Thread* e a sua topologia. Por fim, listaremos os tipos de dispositivos que suportam *Thread*, e abordaremos o funcionamento e as diferenças do protocolo em ambientes domésticos e industriais.

2.3.1. Introdução ao *Thread*

Thread é um protocolo implementado em *IPv6*, agnóstico da camada aplicacional, como demonstrado na Figura 2 - *Thread* no modelo OSI, e que foi desenvolvido com o foco de ser utilizado na integração e controlo de dispositivos *IoT* em casas inteligentes [11].

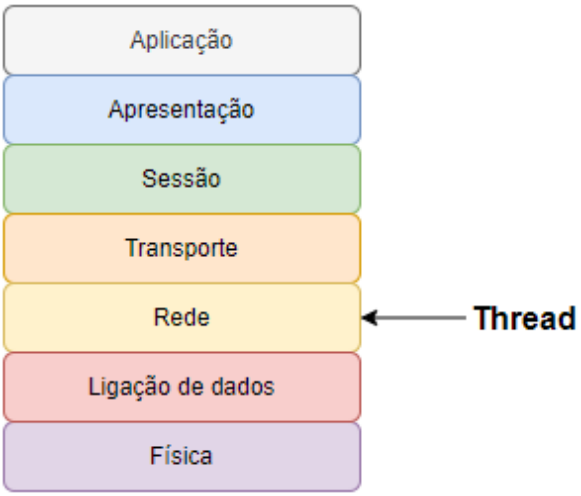

Figura 2 - *Thread* no modelo OSI

Utiliza a topologia em malha, tornando possível a eliminação de pontos únicos de falha, pelo facto que, quando um equipamento falha, outro equipamento assume o papel do dispositivo que falhou sem haver falhas na rede (o mesmo acontece quando se adiciona dispositivos à rede).

Suporta comunicações de baixa latência, que pode ser problemático, pois pode ter problemas de compatibilidade com dispositivos que necessitem de uma largura de banda elevada, como o caso das câmaras de vigilância. É muito eficiente em termos energéticos em relação a outros protocolos existentes, e é seguro devido a utilizar o algoritmo *AES-128* de encriptação com proteção contra repetições.

Este protocolo é bastante versátil, por ser possível integrá-lo não apenas ambientes domésticos, mas também em ambientes industriais. É possível ter 32 routers e 511 dispositivos por router, logo suporta tanto redes de pequena dimensão, mas também tem versatilidade para suportar redes de grande dimensão.

2.3.2. Especificação do *Thread*

Na Tabela 5 - Tabela que demonstra a especificação do *Thread* demonstramos todas as especificações do *Thread*. Destaca-se pelo seu baixo consumo energético, por utilizar topologia em malha que elimina os pontos únicos de falha na rede, e por ter uma grande escalabilidade, sendo assim possível utilizá-lo em redes de pequena e grande dimensão.

É um protocolo com um mecanismo de segurança bastante robusto, com uma taxa de transmissão de dados e largura de banda relativamente baixa, e que tem um alcance médio relativo aos outros protocolos.

Por fim, podemos também perceber que é um protocolo com muita flexibilidade por ser agnóstico da camada aplicacional, permitindo assim que seja compatível com uma grande variedade de aplicações de controlo de dispositivos *IoT*.

Tabela 5 - Tabela que demonstra a especificação do *Thread*

Especificações	*Descrição*
Consumo energético	O *Thread* tem um baixo consumo de energia.
Suporta redes de pequena e grande dimensão	O *Thread* suporta 32 routers e 511 dispositivos por router, logo tem uma grande flexibilidade e escalabilidade para trabalhar em redes de pequena e grande dimensão.
Pontos únicos de falha	A topologia em malha, é a topologia utilizada pelo *Thread*, logo não existem pontos únicos de falha na rede.
Alcance	O alcance de ponto para ponto é de 100 metros, mas pelo facto de utilizar rede em malha, rapidamente há mais escalabilidade e alcance por parte dos dispositivos.

Agnóstico à camada aplicacional	O *Thread* é agnóstico da camada aplicacional, logo é um protocolo muito versátil. Pode ser integrado em qualquer aplicação que suporte *IPv6* e que tenha uma baixa largura de banda.
Seguro	O *Thread* utiliza como encriptação o algoritmo *AES-128* com proteção contra repetições.
Taxa de transmissão de dados	A transmissão de taxa de dados do *Thread* é de 250 *Kbps*

2.3.3. Redes em malha

Numa topologia em malha os dispositivos estão todos interconectados entre si de forma descentralizada para obter redundância na rede toda (quando um dispositivo falha a tabela de roteamento é alterado de forma instantânea e automática para que não haja falhas), tal como demonstrado na Figura 3 - Autoconfiguração de redes em malha.

Isto elimina os pontos únicos de falha, por não depender de um ponto de acesso central, tornando a rede muito mais confiável e diminuindo o tempo de falha na rede [12].

A eliminação dos pontos únicos de falha aplica-se em dois casos, no caso de haver um dispositivo na rede a perder conexão (no caso de ser um dispositivo crítico, é automaticamente atribuído o mesmo papel a outro dispositivo da rede), e no caso de ser adicionado outro dispositivo à rede (a rede autoconfigura-se sem haver falhas).

O *Thread* como utiliza esta topologia de rede é muito confiável, pois a probabilidade de haver falhas de comunicação na rede é muito reduzida, permitindo controlar os dispositivos *IoT* de forma constante.

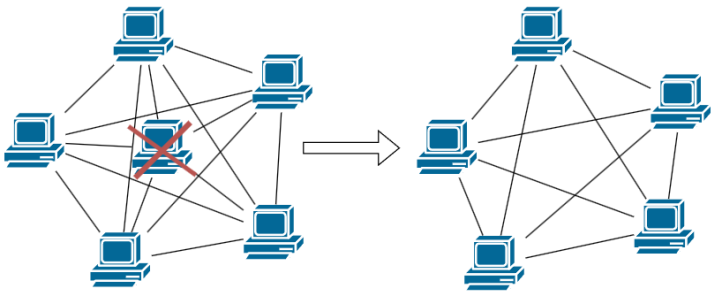

Figura 3 - Autoconfiguração de redes em malha

2.3.4. Pilha protocolar do *Thread*

Thread é agnóstico à camada aplicacional, por isso, apenas é implementado sobre a camada de rede e de transporte, e como utiliza o padrão *IEEE 802.15.4* as camadas *MAC* e *PHY* também são incluídas na sua pilha protocolar como podemos observar na Figura 4.

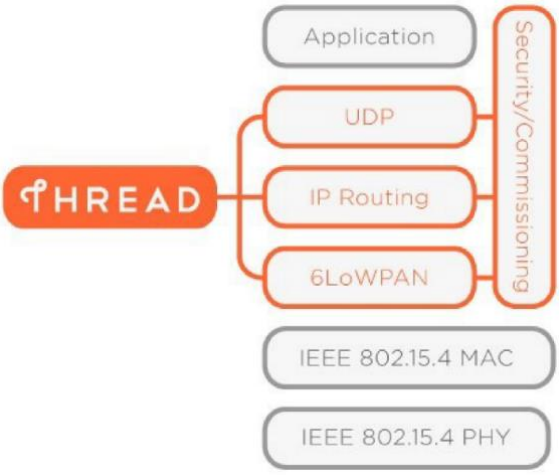

Figura 4 - Pilha protocolar do *Thread*

Camada de rede

A rede *Thread* utiliza endereçamento *IPv6* em todos os seus dispositivos sendo que estes são atribuídos automaticamente, e conforme a necessidade dos dispositivos podem ser atribuidos mais que um *IPv6*.

O *Thread* utiliza mensagens *MLE* (*Message Link Establishment*) para descobrir todos os dispositivos da rede, de forma a criar a tabela de roteamento com base nas ligações com menos custo, e também para criar a rede em topologia de malha. As mensagens MLE são trocadas entre os dispositivos periodicamente para manter sempre a tabela de roteamento e a malha atualizada no caso de haver dispositivos a entrarem ou a saírem da rede para evitar a existência de pontos únicos de falha.

Na camada de rede o *Thread* utiliza o protocolo *6LoWPAN* para adaptar o tamanho dos pacotes transmitidos para um tamanho suportado pelo *Thread* [11].

- IPv6

O *Thread* suporta endereçamento *IPv6* (*Internet Protocol version 6*) que foi criado devido à falta de endereços *IPv4*. Os endereços *IPv6* têm 128 bits, são representados por oito grupos de quatro dígitos hexadecimais e separados por dois pontos como observado na

Figura 5. O *Thread* configura por norma a rede automaticamente, logo suporta *DHCPv6* que é utilizado para atribuir endereços de um conjunto de *IPs* pré-definidos para os dispositivos da rede *Thread* [13].

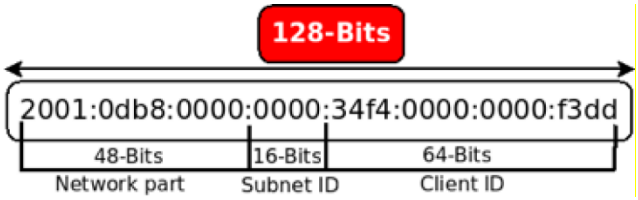

Figura 5 - Constituição dos endereços *IPv6*

O Thread utiliza maioritariamente dois tipos de endereços *IPv6* designados por *ULA* (*Unique Local Address*) e *GUA* (*Global Unicast Address*). Os *ULA* são utilizados dentro da rede *Thread* local, enquanto os *GUA* são utilizados para comunicar entre redes de *Thread* diferentes [14].

- 6LoWPAN

O *6LoWPAN* é um protocolo em *IPv6* que trabalha sobre *Low Power WPANs* (*Wireless Personal Area Networks*), que especifica o padrão das comunicações *IPv6* por *IEEE 802.15.4* (padrão utilizado pelo *Thread*). O *6LoWPAN* é utilizado na rede *Thread* para adaptar (reduzir) os pacotes para um tamanho suportado pela comunicação *Thread* (os pacotes por norma têm 1280 bytes e *Thread* apenas suporta 127 bytes) [15].

Camada de Transporte

Na camada de transporte é estabelecida e gerida a malha (topologia) através do protocolo UDP, e é feita a comunicação com a camada aplicacional através de TCP [11].

- Protocolo *UDP*

O *UDP* (*User Datagram Protocol*) é um protocolo de comunicação utilizado pelo protocolo *IP* (*Internet Protocol*) normalmente para transmissões de dados sensíveis ao tempo.

Acelera as comunicações por não estabelecer propriamente uma comunicação entre o cliente e o servidor (o cliente faz um pedido e o servidor responde, como podemos observar na Figura 6) permitindo que os dados sejam transmitidos de forma mais rápida, mas não

garantido que os dados sejam entregues devido a não haver controlo da taxa de transmissão dos dados [16].

Nas redes *Thread* é utilizado para estabelecer e gerir a topologia de rede em malha.

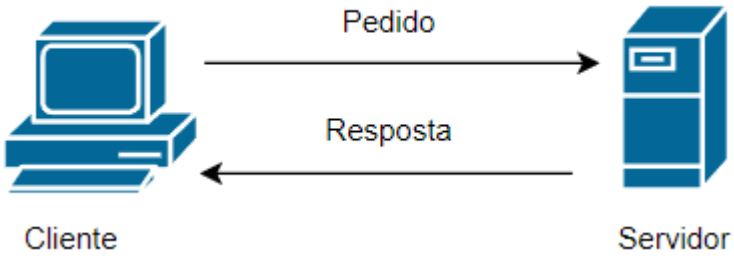

Figura 6 - Protocolo *UDP*

- Protocolo *TCP*

O *TCP* (*Transmission Control Protocol*) é um protocolo de comunicação que permite a troca de mensagens e envio de pacotes pela internet, mas que ao contrário do *UDP*, assegura a entrega dos mesmos.

Antes de transmitir dados, o protocolo *TCP* estabelece comunicação entre o cliente e o servidor para assegurar que os dados apenas são enviados após a comunicação ser estabelecida e que a mesma fica ativa até ao fim do envio de todos os dados.

Para estabelecer a comunicação entre o cliente e o servidor é utilizado um processo designado por *three-way handshake* onde o cliente inicia o processo enviando um segmento *SYN* (*synchronize*), o servidor responde com um segmento que contém o *SYN* e o *ACK* (*acknowledge*), e por fim o cliente envia o último segmento com o *ACK* para estabelecer a comunicação, como explicado na Figura 7 [17].

Nas redes *Thread* é utilizado para assegurar a comunicação com a camada aplicacional, que por norma são os softwares de automação doméstica.

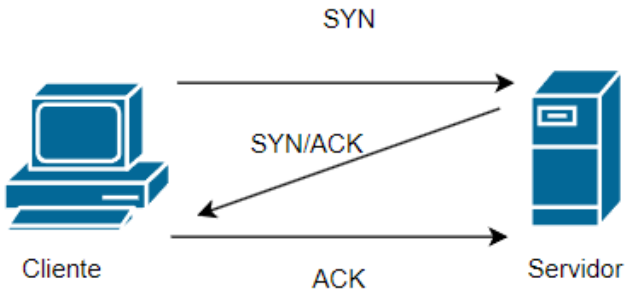

Figura 7 - Protocolo *TCP*

2.3.5. Topologia da rede *Thread*

Nesta secação abordaremos os equipamentos necessários para ser possível criar, estabelecer e gerir uma rede com o protocolo *Thread*, tal como também especificaremos as suas funções na rede. Como o encaminhamento de pacotes é feito na rede *Thread*, e detalharemos o processo de adicionar dispositivos novos à rede [18].

Na Figura 8 podemos observar de forma simplificada como funciona a rede *Thread* e os papeis dos diferentes dispositivos na rede.

Equipamentos

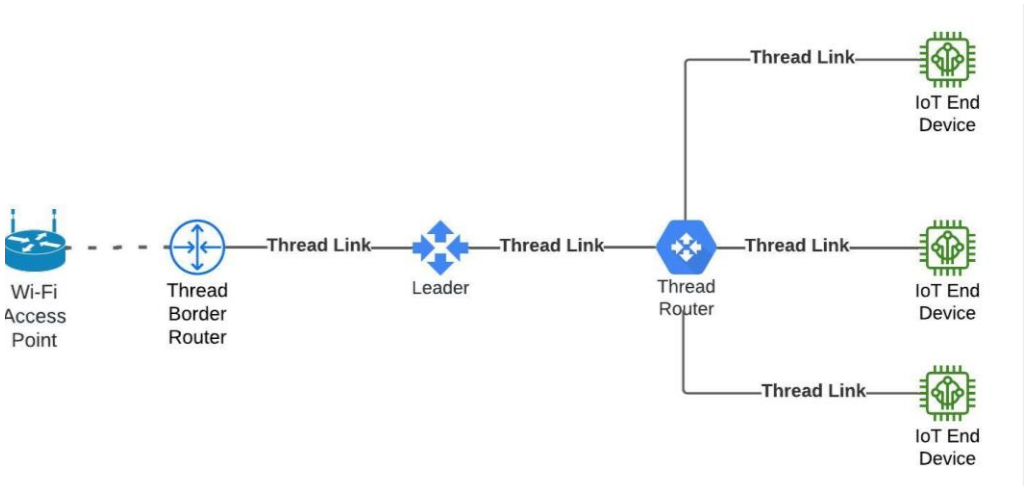

Figura 8 - Cenário simples de uma rede *Thread*

Funções dos dispositivos na rede *Thread*:

- *Thread Border Router*: Este dispositivo faz a conexão entre a rede *Thread* e outras redes externas (*Wi-fi* ou *Ethernet*).
- *Leader*: Um *Leader* é responsável por fazer a gestão de todos os routers na rede *Thread*. Quando há dispositivos a falhar é responsável por ativar outro dispositivo para a função do dispositivo que falhou. Qualquer *router* pode ser eleito como *Leader*, caso haja uma falha no *Leader* atual.
- *Thread Router*: O *Thread Router* é responsável pelo roteamento e pelo encaminhamento de pacotes para os dispositivos da rede.
- *IoT End Device*: Dispositivos que suportam *Thread* e comunicam com o *Thread Router*. Os *End Devices* não podem encaminhar pacotes para outros dispositivos.

Para além destes equipamentos, existem outros equipamentos que não são utilizados com tanta frequência em redes de pequena dimensão, mas que são úteis em termos de redundância em redes de maior escala.

Estes equipamentos são os seguintes [18]:

- *REED* (*Router-Eligible End Devices*): Os *REEDS* são dispositivos que atuam como *End Devices*, mas que conseguem receber mensagens de roteamento e que têm a possibilidade de se tornarem *Thread Routers* se a rede necessitar. Esta configuração é feita automaticamente pela rede, e apenas o *Leader* tem a capacidade de passar um *End Device* para um *Thread Router*.
- *SED* (*Sleepy End Device*): Os *SEDs* estão maioritariamente do tempo em estado de sono para diminuir o consumo de energia, de modo a terem mais durabilidade e a serem mais eficientes. Os *SEDs* apenas saem do modo de sono temporariamente para comunicarem com o *Thread Router*, que por sua vez é o único dispositivo na rede com quem conseguem comunicar.
- *SSED* (*Synchronized Sleepy End Device*): Têm as mesmas características que os *SEDs*, mas em vez de se ligarem temporariamente, apenas se ligam em intervalos definidos pelo utilizador para reduzir ainda mais o consumo de energia.

Encaminhamento e processo de adicionar novos dispositivos na rede

- Encaminhamento

O encaminhamento nas redes *Thread* é feito utilizando a informação das tabelas de encaminhamento que cada dispositivo tem acerca de todos os *routers* na rede e o endereço

do próximo salto (com menos custo). As tabelas são formadas e sempre atualizadas através das mensagens *MLE*, que são pedidas e enviadas pelos dispositivos da rede periodicamente [18].

- Adicionar um novo dispositivo à rede *Thread*

Para adicionar um novo dispositivo à rede *Thread*, o dispositivo tem de passar por três fases, e após suceder nas três é adicionar à rede como *End Device*. Este processo é importante no caso de haver um erro ao adicionar dispositivos à rede *Thread*, por poder ajudar a fazer *troubleshooting* e a descobrir em que fase está a acontecer o erro [18].

As fases são as seguintes:

1. *Discovery*
2. *Commissioning*
3. *Attaching*

Processo de *Discovery*: No processo de *discovery* (feito apenas a primeira vez que o dispositivo entra na rede), o dispositivo *Thread* anuncia mensagens *MLE* para avisar que se está a tentar juntar a uma rede *Thread*. Se existir uma rede *Thread*, o *Thread Router* dessa rede comunica com o dispositivo e indica algumas credenciais acerca da rede.

Processo de *Commissioning*: O processo de *commissioning* é constituído pela autenticação do novo dispositivo na rede, e pelo fornecimento das credenciais da rede ao dispositivo que se está a juntar. Após este passo ser bem-sucedido o novo dispositivo é anexado à rede.

Processo de *Attaching*: Neste processo, novamente, o dispositivo anuncia mensagens *MLE*, para os *routers* ou *REEDs* (podem se tornar routers para anexar o dispositivo no caso de os *routers* não receberem os anúncios *MLE*) na rede saberem do dispositivo, e que já o podem anexar à rede. O dispositivo é sempre anexado como *End Device*, mas pode ter funções diferentes após ser anexado conforme a necessidade da rede.

2.3.6. Tipos de dispositivos que suportam *Thread*

Thread é um protocolo recente, motivo pelo qual ainda não existem muitos equipamentos que suportem este protocolo, contudo a tendência de expansão dos equipamentos é elevada e as empresas estão cada vez mais a desenvolver os seus produtos com suporte para o protocolo *Thread* [19].

Os tipos de dispositivos que suportam *Thread*, mais comuns, são os seguintes:

- Fechaduras inteligentes
- Routers *Wi-Fi*
- Colunas inteligentes
- Lâmpadas inteligentes (temperatura de cor, regulável)
- Sensores de inteligentes (temperatura, contacto, humidade, CO2 e luminosidade)
- Estores inteligentes
- Termostatos
- Bombas de calor
- Alarmes de fumo e de monóxido de carbono
- Tomadas inteligentes
- Motores para cortinas
- Interruptores inteligentes
- *ESP32-C6* e *ESP32-H2*

2.3.7. *Thread* em ambientes domésticos

Na utilização de *Thread* em ambientes domésticos, os utilizadores necessitam de um telemóvel com um software de automação doméstico instalado e com acesso ao *Wi-Fi*, um *Wi-Fi Access Point*, um *Thread Border Router*, e os restantes dispositivos com suporte *Thread* que pretendem utilizar na sua casa inteligente. A Figura 9 representa um exemplo da utilização de *Thread* em ambientes domésticos [18].

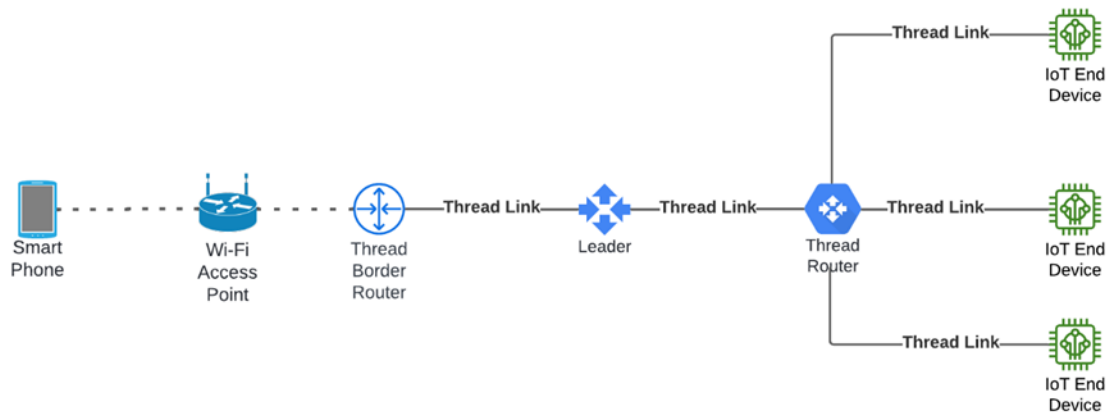

Figura 9 - Rede *Thread* num ambiente doméstico

O *Wi-Fi Access Point* fornece conexão *Wi-Fi* ao telemóvel, e conexão por *Wi-Fi* ou *Ethernet* ao *Thread Border Router*. O telemóvel tem de ter acesso ao *Wi-Fi Access Point* para conseguir controlar os dispositivos na rede *Thread* através do software de automação doméstica. O *Thread Border Router* comunica por Wi-Fi ou Ethernet com o *Wi-Fi Access Point*, e comunica por *Thread* com os dispositivos da rede. Os dispositivos estão conectados ao *Thread Border Router* por *Thread* e são controlados pelo telemóvel.

2.3.8. *Thread* em ambientes industriais

O modelo industrial de *Thread* utiliza como base o modelo doméstico, mas passa a ser uma rede industrial quando se utiliza mais que uma rede *Thread* no mesmo ambiente, como demonstrado na Figura 10 - Rede *Thread* num ambiente industrial. As várias redes *Thread* são designadas por *Thread Domain Level*, são interligadas entre si através de *Backbone Border Routers*, e são sincronizadas através de uma ligação exterior designada por *Backbone Link* [20].

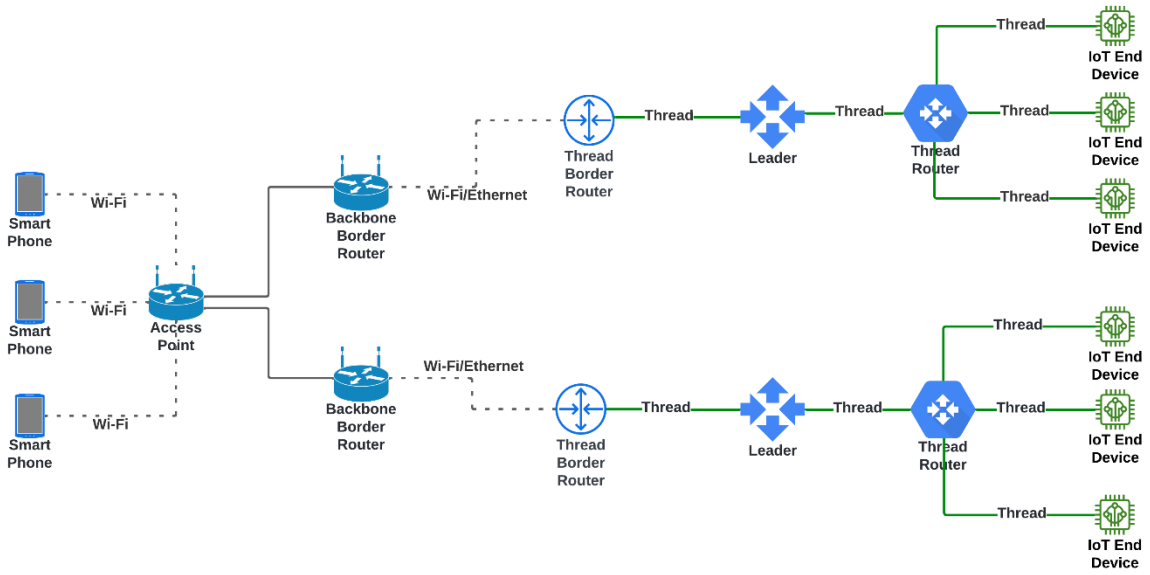

Figura 10 - Rede *Thread* num ambiente industrial

Como podemos observar existem várias redes *Thread*, mas todas utilizam o mesmo conceito interno das redes em ambientes domésticos. Todas as redes *Thread* estão conectadas por *Wi-Fi* ou *Ethernet* ao *Wi-Fi Access Point*, e os dispositivos na rede são controlados através de telemóveis que comunicam por *Wi-Fi* com os *Wi-Fi Access Points*.

A diferença dos ambientes domésticos para os ambientes industriais é a ligação e o sincronismo entre as diversas redes *Thread*. Esse conceito é demonstrado na Figura 11 - Conceitos da rede *Thread* e explicado mais detalhadamente após a figura.

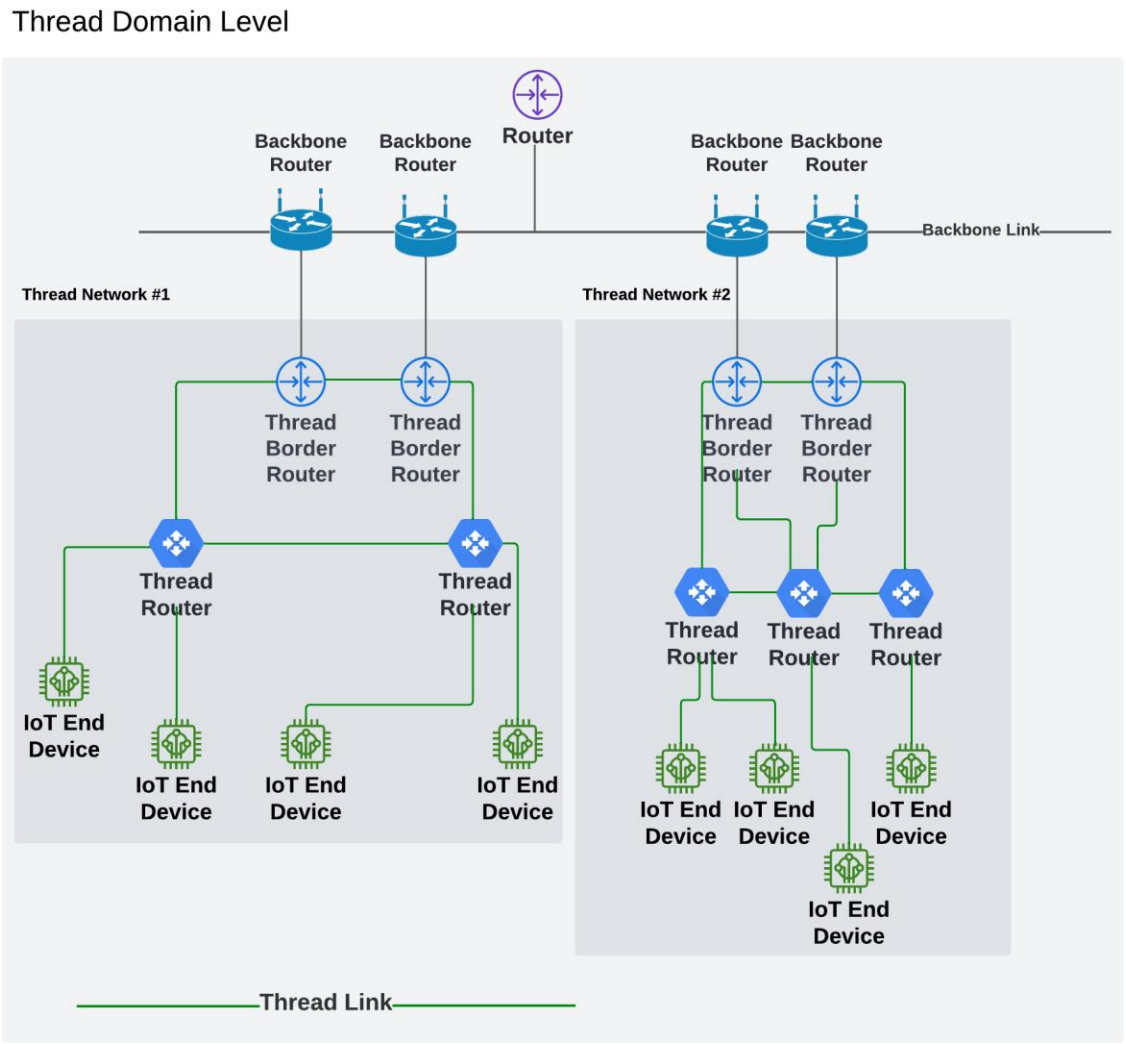

Figura 11 - Conceitos da rede *Thread* industrial

O *Thread Domain Level* representa num todo, todas a redes *Thread* utilizadas no ambiente industrial, e tem a capacidade de se autoconfigurar. Logo, se forem adicionadas mais redes *Thread*, o domínio adapta-se sem configurações adicionais.

O *Backbone Link* é a ligação criada não *IPv6* para haver sincronismo entre todos os *Backbone Border Routers*. Os *Backbone Border Routers* são routers utilizados apenas em ambientes industriais que facilitam o sincronismo de todas as redes Thread utilizadas.

2.4.Matter

Com a evolução dos softwares de automação doméstica e os protocolos de comunicação sem fios, começaram a surgir uma vasta diversidade de equipamentos *IoT*. Isto resultou num problema de interoperabilidade entre os equipamentos que utilizam diferentes ecossistemas. Para solucionar este problema, foi criado o protocolo *Matter*, suportado por uma aliança de muitas empresas mundiais, que visa universalizar a comunicação entre dispositivos de diferentes fabricantes.

Nesta secção, abordaremos o protocolo Matter, discutiremos todas as suas especificações, a estrutura do protocolo, e os dispositivos que o suportam de forma geral. Em seguida, exploraremos as diferentes versões do *Matter*, os dispositivos lançados que suportam *Matter* em cada uma das versões, e as funcionalidades suportadas por cada dispositivo.

2.4.1. Introdução ao *Matter*

Matter lançou a sua primeira versão (*Matter 1.0*) em 2022 com o intuito de universalizar um standard único entre a comunicação de dispositivos de casas inteligentes de diferentes fabricantes para melhorar a interoperabilidade entre eles. Desta forma é possível utilizar softwares de automação doméstica para controlar dispositivos de diferentes ecossistemas sem problemas de compatibilidade por todos partilharem os mesmos comandos (definidos no protocolo *Matter*).

O protocolo *Matter* foi implementado sobre a camada aplicacional e corre sobre protocolos da camada de rede, como *Thread*, *Wi-Fi* ou *Ethernet*. O protocolo *Bluetooth* é utilizado, mas apenas na parte de adicionar novos dispositivos que suportam *Matter* à rede. O *Zigbee* devido a não ser agnóstico à camada aplicacional necessita de um *Matter Bridge* para ser compatível com *Matter*. Esta explicação está exemplificada na Figura 12 - Protocolo *Matter* [21].

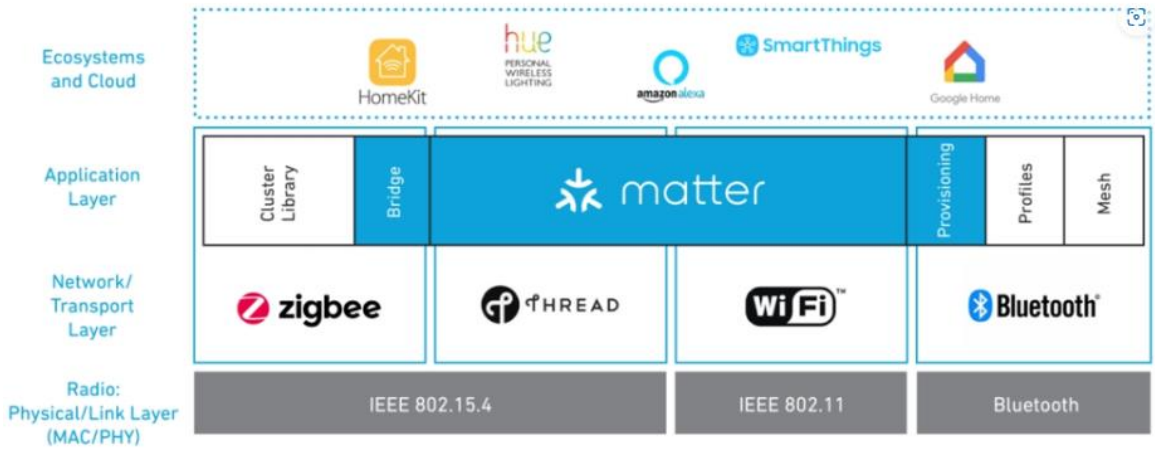

Figura 12 - Protocolo *Matter*

Existe *Matter over Thread* e *Matter over Wi-Fi*, mas a grande diferença entre estas duas opções encontra-se no protocolo de rede e não no *Matter* em si, logo é mais indicado correr *Matter over Thread* devido ao baixo consumo energético, por ser mais seguro e por não ter pontos únicos de falha.

2.4.2. Papéis na especificação *Matter*

Na especificação *Matter* existem vários papéis que podem ser assumidos, como M*atter Device*, *Matter Controller*, *Matter Fabric*, *Matter Commissioner*, *Matter Administrator* e *Matter Bridge*.

A baixo detalhamos as diferenças e especificamos a importância destes papeis [22]:

- *Matter Device*: Os *Matter Devices* são os dispositivos que suporta *Matter*. Então, podem ser controlados por um *Matter Controller*, e pertencem a uma *Matter Fabric*.
- *Matter Controller*: Os *Matter Controllers* controlam os *Matter Devices*, e pertencem a *Matter Fabric*.
- *Matter Fabric*: A *Matter Fabric* é uma rede virtual que corre sobre *Wi-Fi*, *Ethernet* ou *Thread*, e é controlada por um *Matter Administrator*. Os *Matter Devices* e os *Matter Controllers* podem estar conectados a uma ou mais *Matter Fabrics* para comunicarem entre si.
- *Matter Administrator*: Os *Matter Administrators* são responsáveis por controlar e gerir uma ou mais *Matter Fabrics*.

- *Matter Commissioner*: Para adicionar novos dispositivos *Matter* ao *Matter Fabric* é necessário verificar a compatibilidade do dispositivo com *Matter*. O *Matter comissioner* é responsável por executar este processo, caso o dispositivo seja compatível fornece as credencias da rede para adicionar o dispositivo à *Matter Fabric*.
- *Matter Bridge*: O *Matter* apenas pode correr sobre *Thread*, *Wi-Fi* e *Ethernet*, mas existem outros protocolos que podem ser utilizados em ambientes de casas inteligentes, logo, o *Matter Bridge* é utilizado para permitir que ambientes que utilizem outros protocolos de comunicação sem fio se consigam conectar a uma *Matter Fabric*, como por exemplo quando é utilizado *Zigbee*.

2.4.3. Estrutura do *Matter*

O *Matter* tem uma estrutura bastante complexa, e tal como explicado na Figura 13 - Estrutura do Matter [23],é composta por *nodes*, *endpoints*, *clusters*, *attributes*, *commands* e *events* [23].

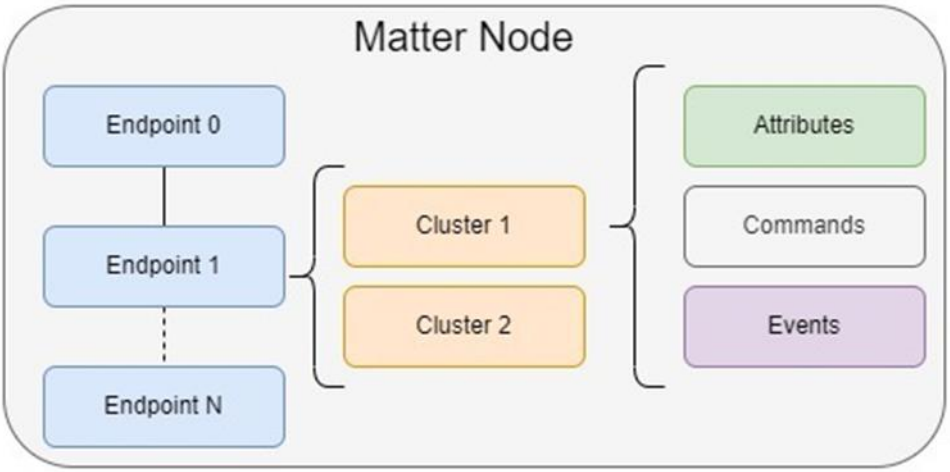

Figura 13 - Estrutura do Matter [23]

- *Nodes*: Um node é um dispositivo físico ou um grupo de dispositivos físicos que suportam *Matter*. Os dispositivos são associados ao node conforme a necessidade do utilizador. Por exemplo, pode haver um node com uma tomada e uma lâmpada.
- *Endpoints*: Um *Endpoint* separa os nodes por dispositivos, ou seja, seguindo o exemplo anterior, a tomada é um *Endpoint*, e a lâmpada é outro *Endpoint*.
- *Clusters*: Um *Cluster* é o grupo de funcionalidades associadas a cada *Endpoint*.
- *Attributes*: Os atributos são variáveis específicas atribuídas a cada cluster.

- *Commands*: Cada cluster tem comandos atribuídos por predefinição e são utilizados para realizar ações no *endpoint*.
- *Events*: Enquanto os atributos representam as transações no presente, os eventos guardam as transações executadas anteriormente.
- Clientes e Servidores: Os clientes e servidores estão atribuídos a clusters. A diferença entre eles é que o cliente inicia a comunicação e o servidor executa as operações (*commands*, *attributes* e *events*). É possível observar como a comunicação entre o cliente e servidor ocorre na Figura 14 - Comunicação entre o cliente e o servidor por Matter [24].

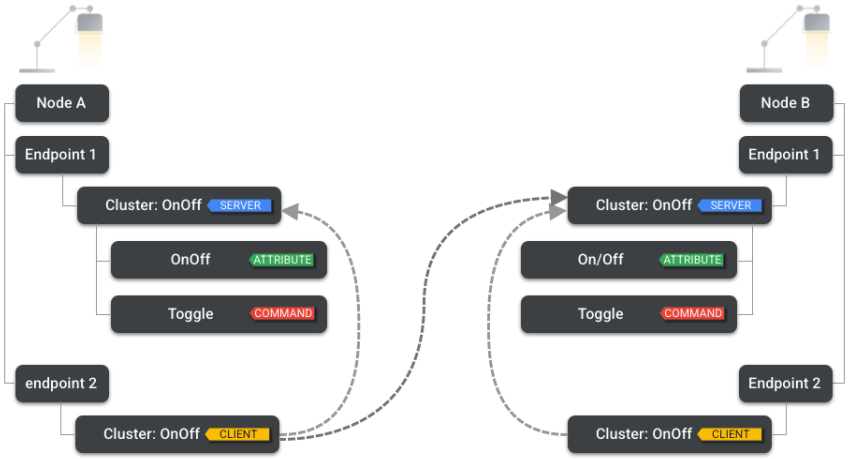

Figura 14 - Comunicação entre o cliente e o servidor por Matter [24]

2.4.4. Processo de *Discovery*

O processo de *discovery* dos dispositivos integráveis ocorre antes do processo de *commissioning*. Os dispositivos podem-se anunciar através de *Bluetooth Low Energy* (*BLE*) ou de *Domain Name Service – Service Discovery* (*DNS-SD*). Se for a primeira vez que o dispositivo estiver a ser adicionado a uma *Matter Fabric* é utilizado sempre o protocolo *BLE*. Caso já tenha sido adicionado anteriormente utiliza *DNS-SD* [25].

2.4.5. Processo de *Comissioning*

O processo de *commissioning* em *Matter* passa por vários passos até o dispositivo receber as credenciais e ser adicionado à *Matter Fabric*. A Figura 15 - Processo de commissioning [26] especifica todos os passos em ordem cronológica que ocorrem durante o processo de *commissioning* [26].

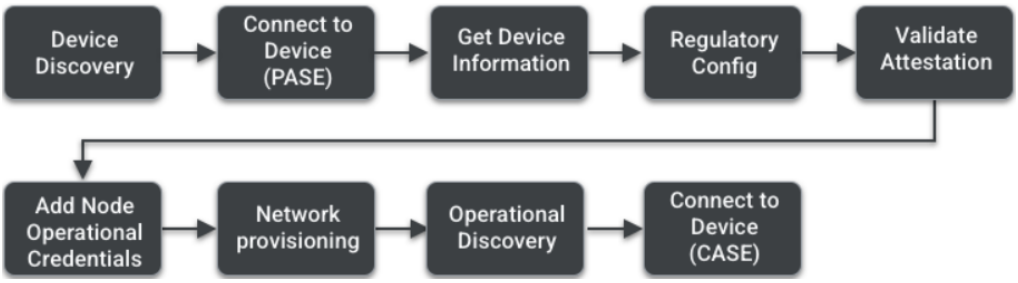

Figura 15 - Processo de commissioning [26]

2.4.6. Dispositivos suportados por *Matter*

O protocolo *Matter* tem vindo a evoluir incrementalmente, sendo a sua primeira versão o *Matter 1.0*, e a versão mais recente o *Matter 1.3*. Desde a sua existência o protocolo cada vez suporta mais dispositivos, e até à data mais recente os dispositivos suportados pelo protocolo são os seguintes [27]:

- Lâmpada
- Lâmpada regulável
- Lâmpada com temperatura de cor
- Lâmpada de cor estendida
- Tomadas inteligentes
- Tomada
- Interruptor
- Interruptor de lâmpadas
- Interruptor de luminosidade
- Sensor de contacto
- Sensor de luminosidade
- Sensor de ocupação
- Sensor de temperatura
- Sensor de pressão
- Sensor de fluxo
- Sensor de humidade
- Sensor de fumo
- Segurança
- Fechadura de portas

- Estores
- HVAC
- Unidade de aquecimento/arrefecimento
- Termostato
- Ventoinha
- Purificador de ar
- Sensor de qualidade do ar
- Dispositivo de vídeo
- Coluna
- Aplicação de conteúdo
- Controlo remoto de vídeo
- Aspirador robot
- Frigorifico
- Ar condicionado
- Máquina de lavar a roupa
- Máquina de lavar a loiça
- Micro-ondas
- Forno
- Fogão
- Exaustor
- Máquina de secar
- Controladores de água e energia
- Controlador de energia
- Controlador de carregador de carros elétricos
- Detetor de desperdício de água
- Detetor de congelamento
- Sensores de chuva
- Controlador de válvula de água

2.4.7. Matter 1.0

O *Matter 1.0* foi a primeira versão de *Matter* que foi realizada para o mercado pela *Connectivity Standards Alliance* (anteriormente conhecida como *Zigbee Alliance*). O intuito principal da primeira versão foi para os fabricantes testarem o protocolo ao máximo, para

descobrirem todas as suas vantagens e utilidades, mas também todos os possíveis erros que o protocolo pudesse ter [28].

Lâmpadas

Nesta versão, existem diversos tipos de lâmpadas, e para cada tipo de lâmpada existem diversas funcionalidades. É possível ligar e desligar a lâmpada remotamente, controlar a cor, a intensidade da luminosidade e a saturação. Se o consumidor pretender também é possível controlar a lâmpada através de um interruptor regulável [29].

Interruptores

É possível ligar e desligar interruptores remotamente, e utilizar interruptores reguláveis para controlar a intensidade da luminosidade e a cor de lâmpadas reguláveis [29].

Sensores de luminosidade

O sensor de luminosidade permite medir e reportar a intensidade da luz em tempo real [29].

Sensores de temperatura

O sensor de temperatura permite medir e reportar a temperatura em tempo real [29].

Sensores de pressão

O sensor de pressão permite medir e reportar a pressão de um fluido em tempo real [29].

Sensores de fluxo

O sensor de fluxo permite medir e reportar a taxa de fluxo em tempo real [29].

Tomadas inteligentes

Na versão do *Matter 1.0* é possível ligar e desligar a tomada inteligente remotamente, quanto tempo é que deve de estar ligada/desligada e agendar um tempo para se ligar/desligar [29].

Fechaduras inteligentes

Esta versão oferece várias funcionalidades para fechaduras inteligentes. Indica se a fechadura está trancada ou não, se a porta está aberta ou fechada, e também conta o número de vezes que a porta foi aberta e fechada, entre outras funcionalidades [29].

Controladores de aplicações de transmissão

Os controladores de aplicações de transmissão permitem controlar a inicialização de aplicações, a saída de dispositivos de reprodução de áudio, o canal a ser utilizado num determinado dispositivo de transmissão, a inicialização de conteúdo em dispositivos de reprodução de vídeo, e a reprodução de transmissão em dispositivos de transmissão.

Os clusters para controlar as aplicações de transmissão são os seguintes "*Application Launcher Cluster*", "*Audio Output Cluster*", "*Channel Cluster*", "*Content Launcher Cluster*" and "*Media Playback Cluster*". Os clusters são especificados na Tabela 6 - Tabela de especificação das funcionalidades dos Clusters utilizados pelos controladores de aplicações de transmissão [29].

Tabela 6 - Tabela de especificação das funcionalidades dos Clusters utilizados pelos controladores de aplicações de transmissão

Cluster	Funcionalidade
Application Launch CLuster	Fornece uma interface para o utilizador inicializar e terminar aplicações em dispositivos de reprodução de vídeos, como por exemplo televisões.
Audio Output Cluster	Fornece uma interface para controlar a saída de dispositivos de reprodução de áudio.
Channel Cluster	Fornece uma interface para controlar o canal atual a ser utilizado num dispositivo.
Content Launcher Cluster	Fornece uma interface para inicializar conteúdo em dispositivos de reprodução de vídeo.
Media Playback Cluster	Fornece uma interface para controlar reprodução de transmissão em dispositivos de transmissão.

Estores inteligentes

É fornecido uma interface para controlar e ajustar os estores horizontalmente ou verticalmente de forma automática [29].

Controladores de portões de garagem

Os controladores de portões de garagens são utilizados para abrir e fechar o portão conforme a necessidade do utilizador [29].

Termostatos

Os controladores de termostatos são utilizados para poder definir se pretende aquecer ou arrefecer a sua casa [29].

Controladores de HVAC

Os controladores *HVAC* possuem vários *clusters*, em que cada um tem os seus atributos. Entre eles estão o "*Pump Configuration and Control*", o "*Thermostat*" (mesmas funcionalidades que os termostatos), o "*Fan Control*" e o "*Thermostat User Interface Configuration*". Os clusters são especificados na Tabela 7 - Tabela de especificação das funcionalidades dos clusters utilizados pelos controladores *HVAC* [29].

Tabela 7 - Tabela de especificação das funcionalidades dos clusters utilizados pelos controladores *HVAC*

Cluster	Funcionalidade
Pump Configuration and Control	Fornece uma interface para configurar e controlar os dispositivos de bomba do ar condicionado, tal como também fornece relatórios automáticos do estado dos dispositivos.
Fan Control	Especifica a interface para controlar a velocidade das ventoinhas.
Thermostat User Interface Configuration	Fornece uma interface para permitir que o utilizador consiga configurar a interface de utilizador para o termostato ou para um dispositivo de controlo do termostato.

2.4.8. Matter 1.1

A versão *Matter 1.1* não foi uma atualização de grande dimensão, nem introduziu suporte *Matter* para novos dispositivos. O foco principal foi em facilitar o processo de certificação dos dispositivos por partes das empresas que pretendem que os seus dispositivos suportem *Matter*, e com base no feedback da versão 1.0, na versão 1.1 foram resolvidos alguns erros encontrados na experiência do utilizador final, como dispositivos com bateria que ficavam frequentemente *offline* (essa questão foi melhorada) [30].

2.4.9. Matter 1.2

O *Matter 1.2* ficou marcado como a segunda ronda de atualizações após a saída do *Matter 1.0* em 2022. O *Matter 1.2* trouxe compatibilidade do protocolo *Matter* com novos equipamentos, expandindo o mercado de dispositivos com suporte *Matter*. Trouxe algumas melhorias e correções relativas a erros e problemas encontrados nas versões anteriores melhorando assim a interoperabilidade e a interação com o consumidor final [31].

Frigoríficos e congeladores

É possível controlar remotamente a temperatura e fazer a monitorização de frigoríficos e congeladores [32].

Ar condicionado

Apesar de o *Matter* suportar termostatos e *HVAC* na versão 1.0, na versão 1.2 é possível controlar a temperatura e a velocidade das ventoinhas dos ares condicionados.

Os controladores dos ares condicionados possuem vários clusters, e cada um tem os seus próprios atributos. Os clusters são "*Pump Configuration and Control Cluster*", "*Thermostat Cluster*" (encontra-se igual ao Matter 1.0) e "*Fan Control Cluster*". Os clusters são especificados na Tabela 7 - Tabela de especificação das funcionalidades dos clusters utilizados pelos controladores *HVAC*[32].

Máquinas de lavar a loiça

É possível controlar remotamente, receber notificações da máquina de levar a loiça, e ativar a emissão de alarmes no caso de ser detetado algum erro na máquina [32].

Máquinas de lavar a roupa

É possível controlar remotamente, receber notificações da máquina de levar a roupa, e ativar a emissão de alarmes no caso de ser detetado algum erro na máquina [32].

Aspiradores robot

É possível ter controlo remoto sobre o aspirador, definir os modos de uso e receber notificações no caso de algum erro.

Os aspiradores têm os atributos divididos por clusters. O cluster "*Run Mode*" (modo de funcionamento), "*Clean Mode*" (modo de limpeza) e "*Operational State*" (modo de operação) [32].

Alarmes de fumo e de monóxido de carbono

Suportam notificações e sinalização de alarme por áudio e visão. Suportam alertas sobre o estado da bateria, e também suportam sensores de concentração [32].

Sensores de qualidade de ar

Os sensores que suporta capturam e emitem reportes sobre *PM1, PM2.5, PM10, CO2, NO2, VOC, CO, Ozone, Radon, and Formaldehyde* [32].

Purificadores de ar

Utiliza um sensor para fornecer informação acerca da qualidade do ar, suporta obrigatoriamente ventoinhas, e opcionalmente termostatos [32].

Ventoinhas

No *Matter* 1.2 as ventoinhas são consideradas um dispositivo único que suporta a troca da direção do ar, oscilação das ventoinhas, e a velocidade do ar [32].

2.4.10. Matter 1.3

O *Matter 1.3* é a última versão emitida até à data de hoje. Esta versão do *Matter* permite o suporte do protocolo *Matter* para mais dispositivos, sendo o foco principal em dispositivos para a cozinha e para a lavandaria. Os problemas ambientais são algo muito importante no dia a dia, e devido a isto, também houve uma atenção a este tópico, e é agora possível tornar as casas inteligentes mais eficientes através de haver suporte de gestão de energia e de água.

Adicionalmente, também houve melhorias na experiência do utilizador (executar comandos de forma síncrona e atribuir comandos a vários dispositivos), na fase de *troubleshooting* quando o consumidor está a obter erros, e na experiência do programador indesejáveis (o processo de adicionar dispositivos à rede foi melhorado e os dispositivos podem fazer anúncios de mensagens *MLE* por mais tempo para facilita o processo de *commissioning*) [33].

Micro-ondas

É possível controlar o tempo de funcionamento do micro-ondas, a potência de aquecimento/descongelamento, e receber notificações acerca do seu estado [34].

Fornos

Cada compartimento de operação do forno pode ser controlado individualmente, e é possível definir a temperatura do forno remotamente. Para além disto, é possível ativar notificações sobre o estado de aquecimento do forno [34].

Fogões

É possível controlar fogões de forma remota e receber notificações em tempo real sobre a temperatura do fogão [34].

Exaustores

O *Matter 1.3* permite o controlo remoto da luminosidade e das ventoinhas de exaustores [34].

Máquinas de secar a roupa

No *Matter 1.3*, existem diversas funcionalidades que são possíveis de utilizar para controlar remotamente máquinas de secar a roupa. As funcionalidades são as seguintes: indicar o modo de secagem, a temperatura da secagem, ligar e desligar a máquina, e receber notificações customizadas [34].

Controladores para carregadores de carros elétricos

Estes controladores tornam possível haver uma forma fácil e eficaz para controlar o carregamento de carros elétricos. Permitem controlo remoto para ligar/desligar o carregador, ajustar a taxa de carregamento, e especificar a quantidade de bateria que tem de carregar para o carro percorrer X quilómetros.

Devido a haver tantas funcionalidades, existem vários clusters com vários atributos. Os clusters são os seguintes "*Device Energy Management Cluster*", "*Energy EVSE Cluster*" e "*Energy Preference Cluster*", e são especificados na Tabela 8 - Tabela de especificação das funcionalidades dos *Clusters* utilizados pelos controladores para carregadores de carros elétricos [34].

Tabela 8 - Tabela de especificação das funcionalidades dos *Clusters* utilizados pelos controladores para carregadores de carros elétricos

Cluster	Funcionalidade
Device Energy Management Cluster	Permite controlar o consumo de energia de um dispositivo.
Energy EVSE Cluster	Electric Vehicle Supply Equipment (EVSE) é um equipamento utilizado para carregar carros elétricos e/ou híbridos. Este Cluster fornece uma interface para controlar as funcionalidades do EVSE.
Energy Preference Cluster	Fornece uma interface para especificar as preferências do consumo de energia dos dispositivos.

Dispositivos com suporte para controlo de água

Existe um vasto número de dispositivos para controlo de água que suportam *Matter*. Os dispositivos são úteis para monitorizar, controlar e proteger contra a chuva e contra o congelamento.

Os dispositivos de controlo de água são os detetores de perca de água e de congelamento, sensores de chuva, e válvulas de água que podem ser controladas remotamente [34].

2.5. Matter over Wi-Fi e *Matter over Thread*

2.5.1. Matter over Wi-Fi

Matter over Wi-Fi é uma tecnologia que permite a comunicação entre dispositivos *IoT* em ambientes domésticos utilizando o protocolo *Matter*, através da rede *Wi-Fi* doméstica. Na Figura 16 - Funcionamento de *Matter over Wi-Fi*é demonstrado de forma simplificada como a comunicação é feita.

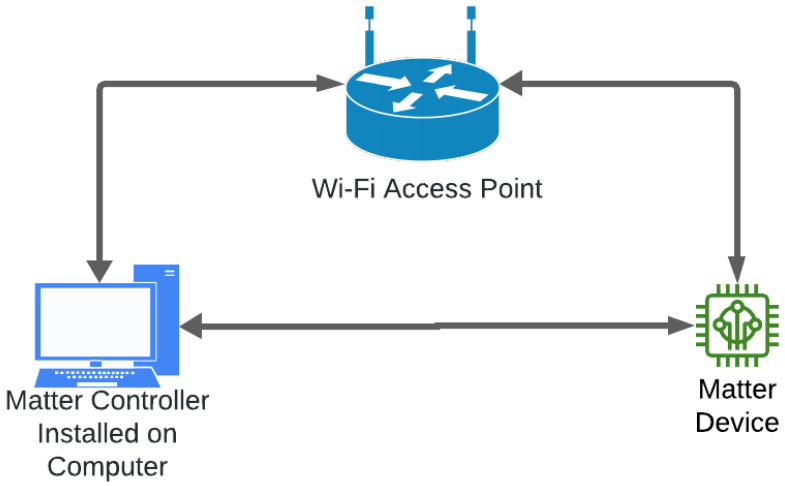

Figura 16 - Funcionamento de *Matter over Wi-Fi*

A função dos componentes é a seguinte:

- *Wi-Fi Access Point*: Os dispositivos são conectados à rede *Wi-Fi* doméstica existente.
- *Matter Device*: Dispositivo que suporta *Matter*, que pode ser integrado num software de automação doméstica e controlado através de um telemóvel pelo *Matter Controller*.
- *Matter Controller*: O *Matter Controller* neste exemplo está inserido no computador e é utilizado para controlar o *Matter Device* por *Matter over Wi-Fi*.

2.5.2. Matter over Thread

Matter over Thread é uma tecnologia que permite a comunicação entre dispositivos *IoT* em ambientes domésticos utilizando o protocolo *Matter*, através da rede *Thread*. Na Figura 17 - Funcionamento de *Matter over Thread* é demonstrado de forma simplificada como a comunicação é feita.

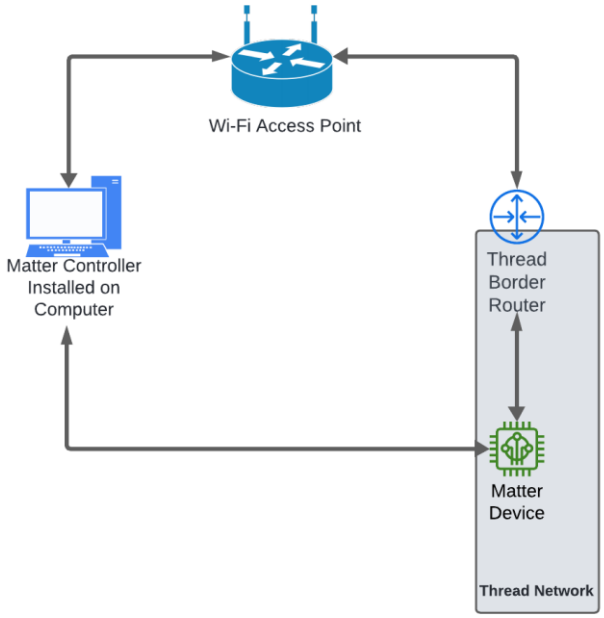

Figura 17 - Funcionamento de *Matter over Thread*

A função dos componentes é a seguinte:

- *Thread Border Router*: Permite a comunicação entre a rede *Thread* e a rede doméstica por *Wi-Fi* ou *Ethernet*.
- *Wi-Fi Access Point*: O *Thread Border Router* comunica com o *Wi-Fi Access Point* para comunicações que não sejam com a rede *Thread* (por exemplo um telemóvel com o software de automação instalado).
- *Matter Device*: Dispositivo que é integrado no software de automação e que é controlado pelo *Matter Controller* através do telemóvel por *Matter over Thread*.
- *Matter Controller*: O *Matter Controller* neste exemplo está inserido no telemóvel/computador e é utilizado para controlar o *Matter Device* por *Matter over Thread*.

2.6.Hardware compatível com *Thread*

2.6.1. Thread Border Routers

Nesta secção abordaremos os *Thread Border Routers* que são mais indicados para utilizar em ambientes de casas inteligentes. Analisaremos as suas vantagens e desvantagens, de forma a chegarmos a uma conclusão sobre o melhor *Thread Border Router* a utilizar nestas soluções.

Espressif Thread Border Router

Vantagens:

- Suporta *Wi-Fi* e o *IEEE 802.15.4*
- Interoperabilidade com dispositivos *Thread*
- Baixo consumo de energia
- Escalabilidade para maiores redes *Thread*

Desvantagens:

- Configuração complexa
- Dependência com ecossistemas e dispositivos *Espressif*
- Poucas atualizações recentemente

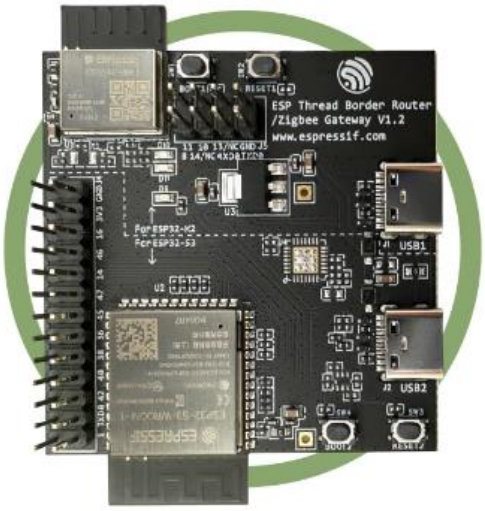

Figura 18 - Espressif Thread Border Router [35]

Nest Wi-Fi Pro

Vantagens:

- Oferece três bandas de conexão *Wi-Fi*
- Suporta *Thread*
- Integração fácil com produtos da Google
- Fácil de integrar com o Google *Home App*

Desvantagens:

- Custo elevado
- Dependência do ecossistema da Google
- Disponibilidade reduzida em certas zonas

Figura 19 - Nest Wi-Fi Pro [36]

GL-S200

Vantagens:

- Suporta *Thread*, *BLE e Wi-Fi 2.4GHz e 5GHz*
- Suporta *VPN* (*Virtual Private Network*)
- Funcionalidades de segurança
- Suporta o *firmware OpenWrt*

Desvantagens:

- Tem poder de processamento limitado
- Tem limitações no alcance
- Poucos portos de Ethernet
- O *OpenWrt* permite customização, mas é bastante complexo

Figura 20 - Router GL-S200 [37]

Sonoff Zigbee 3.0 USB Dongle

Vantagens:

- Suporta *Zigbee* e *Thread*
- Custo aceitável
- Instalação fácil

Desvantagens:

- Alcance limitado
- Afeta a bateria da máquina ao qual está ligado
- Cada adaptador *Sonoff* apenas suporta uma rede *Thread*/*Zigbee*

Figura 21 - Sonoff Zigbee 3.0 USB Dongle [38]

Análise Comparativa dos *Thread Border Routers*

Após a realização da pesquisa detalhada sobre os diferentes *Thread Border Routers* disponíveis no mercado, chegamos à conclusão que o mais indicado a utilizar é o adaptador *Sonoff Zigbee 3.0 USB Dongle*. Concluímos isto por ter um processo de instalação fácil, por ter suporte *Thread*, e por ser possível integrá-la no *Home Assistant*, que é o software de automação doméstica que considerámos mais ideal para soluções de casas inteligentes.

2.6.2. ESP32

Os *ESP32* são equipamentos que oferecem muita versatilidade em ambientes de produção para testagem, por permitirem integrar facilmente diversos tipos de dispositivos. Logo, decidimos pesquisar detalhadamente diferentes tipos de *ESP32* para descobrir o mais adequado em ambientes de casas inteligentes que utilizem *Matter over Thread*.

Desenvolvemos uma Tabela 9 - Tabela comparativa dos microcontroladores *ESP32* com as características dos *ESP32* [39] ,*ESP32-C3* [40], *ESP32-H2* [41] e *ESP32-C6* [42] , e detalhamos as vantagens, desvantagens e diferenças entre todas as placas.

Tabela 9 - Tabela comparativa dos microcontroladores *ESP32*

Recurso	*ESP32*	*ESP32-C6*	*ESP32-H2*	*ESP32-C3*
CPU	*Single/Dual-Core 32-bit LX6* até *240 MHz*	*Single-Core 32-bit RISC-V* até *160 MHz*	*Single-Core 32-bit RISC-V* até *96 MHz*	*Single-Core 32-bit RISC-V* até *160 MHz*
SRAM	520 *KB*	400 *KB*	320 *KB*	400 *KB*
ROM	448 *KB*	384 *KB*	128 *KB*	384 *KB*
Wi-Fi	*802.11 b/g/n (2.4 GHz)*	*802.11ax*	Não suporta	*802.11 b/g/n (2.4 GHz)*
Bluetooth	*802.11 b/g/n (2.4 GHz)*	*802.11ax* (*Wi-Fi 6*)	Não suporta	*802.11 b/g/n (2.4 GHz)*
gpios programáveis	Bluetooth v4.2	Bluetooth 5 (LE)	Bluetooth 5 (LE)	Bluetooth 5 (LE)
Suporte com o padrão *IEEE 802.15.4*	Não	Sim	Sim	Não

CPU: O *ESP32* possui um processador *Single/Dual-Core 32-bit LX6* que pode operar até 240 *MHz*, oferecendo um desempenho bom para aplicações que necessitam de um processamento intensivo. O *ESP32-C6* é equipado com um núcleo *Single-Core 32-bit RISC-V* que suporta até 160 *MHz*, e tem como base equilibrar o desempenho e a eficiência energética. O *ESP32-H2* também usa um núcleo *Single-Core 32-bit RISC-V*, mas com uma frequência mais baixa de até *96 MHz*, adequado para aplicações menos exigentes em termos de processamento. O *ESP32-C3* é idêntico ao *ESP32-C6*, possui um núcleo *Single-Core 32-bit RISC-V* que opera até 160 *MHz*.

SRAM e ROM: O *ESP32* tem a maior quantidade de *SRAM* com 520 *KB* e 448 *KB* de *ROM*, proporcionando amplo espaço para a execução de programas e armazenamento de dados. O *ESP32-C3* e o *ESP32-C6*, ambos têm 400 *KB* de *SRAM* e 384 *KB* de *ROM*,

suficientes por norma em ambientes de casas inteligentes. O *ESP32-H2* oferece 320 *KB* de *SRAM* e 128 *KB* de *ROM*, adequados para dispositivos que exigem menos memória.

Wi-Fi e Bluetooth: O *ESP32* suporta *Wi-Fi 802.11 b/g/n* (*2.4 GHz*) e Bluetooth v4.2, adequado para aplicações que requerem conectividade versátil. O *ESP32-C6* suporta o padrão mais recente *Wi-Fi 802.11ax* (*Wi-Fi 6*) e Bluetooth 5 (LE). O *ESP32-H2* não possui conectividade *Wi-Fi*, mas suporta Bluetooth 5 (LE), sendo menos versátil. O *ESP32-C3* é idêntico ao *ESP32-C6* em termos de Wi-Fi (802.11 b/g/n) e Bluetooth 5 (LE), mas não suporta o padrão *IEEE 802.15.4*.

gpios programáveis: O *ESP32* oferece 10 *gpios* programáveis, o que pode ser limitado para projetos com muitos dispositivos. O *ESP32-C6* oferece uma opção de 22 ou 30 *gpios*, e o *ESP32-C3* 22 *gpios*, logo, proporcionam mais flexibilidade para conectar diversos dispositivos. O *ESP32-H2* tem 19 *gpios*, que é um número normalmente suficiente para cenários de uma dimensão média.

Suporte com o padrão *IEEE 802.15.4*: O *ESP32* e o *ESP32-C3* não suportam o padrão *IEEE 802.15.4*, logo não suportam *Thread*. O *ESP32-C6* e o *ESP32-H2*, ambos suportam o padrão *IEEE 802.15.4*, tornando-os ideais para cenários que utilizam, o protocolo *Thread*.

Análise Comparativa dos *ESP32*

Para cenários que utilizem *Thread*, o *ESP32-C6* é a placa mais indicada devido a suportar o protocolo *IEEE 802.15.4* (padrão utilizado pelo protocolo *Thread*). Embora o *ESP32-H2* também suporte o mesmo padrão, o *ESP32-C6* tem mais poder de processamento e também suporta *Wi-Fi* (mais versatilidade de integração em diferentes cenários).

2.7.Síntese

As casas inteligentes têm estado em constante evolução, resultando no surgimento de novos protocolos de comunicação sem fio, tanto a nível de rede, como a nível aplicacional, com o objetivo de facilitar a integração e o controlo de dispositivos *IoT* nestes ambientes.

Primeiramente surgiram os softwares de automação doméstica para permitir o controlo de dispositivos *IoT* remotamente, que por sua vez, impulsionou a evolução de protocolos de comunicação sem fio, sendo o mais recente e desenvolvido para esta vertente, o *Thread*. Esta evolução estimulou o fabrico de novos equipamentos por parte de diferentes fabricantes e ecossistemas, resultando num problema de interoperabilidade entre todos estes

equipamentos. Devido a este problema, surgiu o protocolo a nível aplicacional designado por *Matter* para universalizar a comunicação entre dispositivos de diferentes fabricantes.

Através da nossa pesquisa, chegámos a várias conclusões:

- O *Home Assistant* é o melhor software de automação doméstica para utilizar em ambientes de casas inteligentes.
- O *Thread* é o protocolo de comunicação sem fio ideal para utilizar nestes ambientes.
- O adaptador *sonoff zigbee 3.0 USB dongle* é o *Thread Border Router* mais indicado para utilizar nestas soluções.
- O *ESP32-C6* é o microcontrolador mais recomendado para ser utilizado em soluções que utilizem *Matter over Thread*.

No próximo capítulo com base na pesquisa efetuada será proposto uma solução em *Matter over Thread* que possa ser utilizado em casas inteligentes.

3. Solução Proposta

A solução que pretendemos desenvolver é um cenário em *Matter over Thread*. No cenário é suposto conseguirmos controlar dispositivos que pertencem a uma rede ou a mais redes *Thread* através de um telemóvel ou computador (por utilizador) com acesso a um servidor do *Home Assistant* (decidimos que é o software de automação doméstica mais indicado através da pesquisa do estado de arte). A intenção é conseguir controlar os dispositivos integrados no *Home Assistant*, tanto fisicamente como através da interface gráfica do *Home Assistant*, e que ao mesmo tempo haja sincronismo entre o estado físico e o estado virtual (quando ligamos um interruptor fisicamente por exemplo, o botão do interruptor associado a esse interruptor no Home *Assistant* também liga e vice-versa).

Este cenário visa demonstrar da forma mais simples possível como desenvolver cenários em *Matter over Thread* para casas inteligentes. *Matter* e *Thread* ainda são muito recentes e ainda estão em fase de desenvolvimento, mas cada vez são mais procurados, então há essa necessidade.

3.1. Arquitetura da solução proposta

Para o desenvolvimento do projeto, tal como representado na Figura 22 - Arquitetura da solução proposta, pretendemos utilizar dispositivos que suportem *Thread*, para que possam ser integrados na rede.

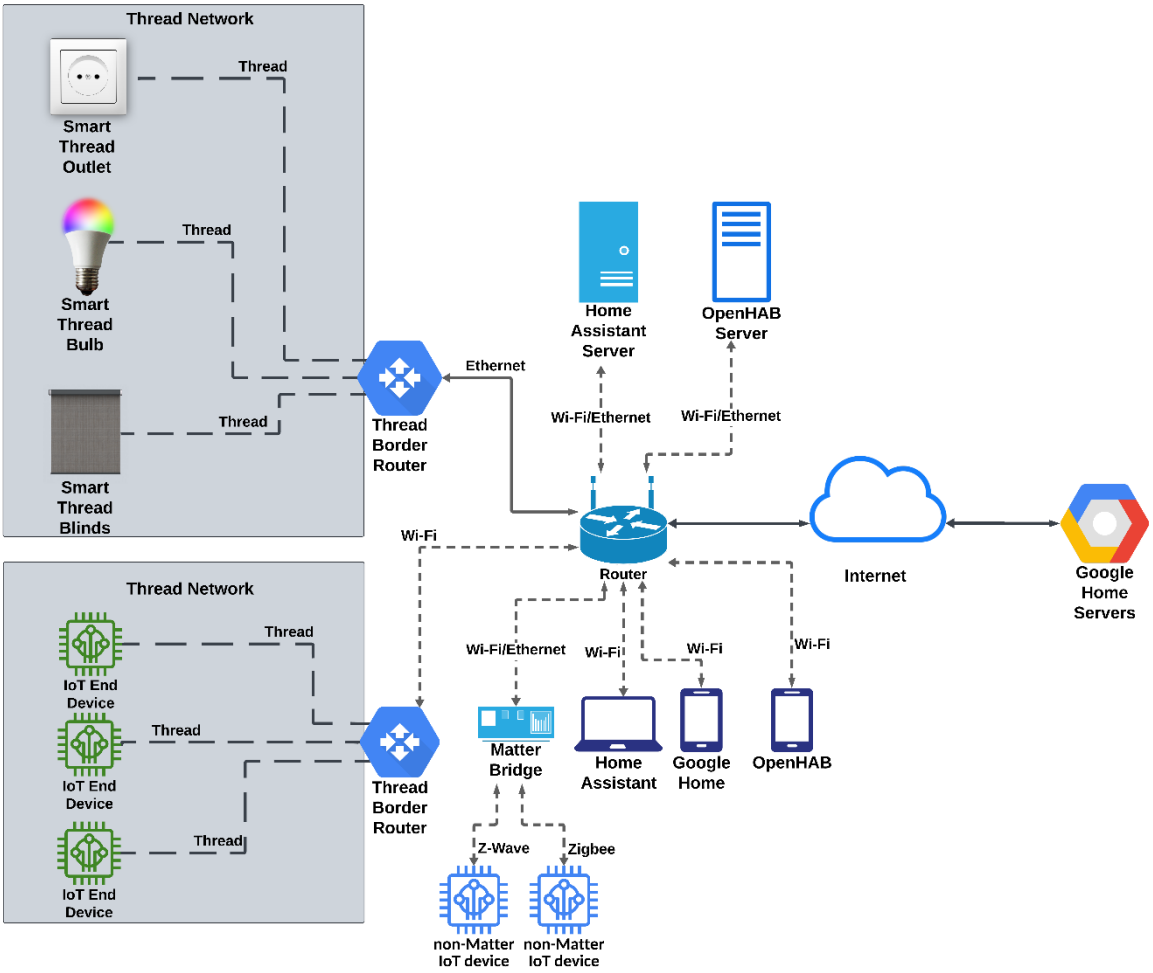

Figura 22 - Arquitetura da solução proposta

Para fazer a integração e o controlo dos dispositivos queremos utilizar o software de automação *Home Assistant* que irá estar instalado num servidor e conectado ao *Wi-Fi Access Point* por *Wi-Fi* ou *Ethernet*. Conforme a quantidade de redes *Thread* que iremos ter, necessitamos de ter o mesmo número de *Thread Border Routers*, para que haja um *Thread Border Router* para encaminhar o tráfego de cada rede. Dependendo da dimensão do cenário também iremos de ter um ou mais *Wi-Fi Access Points* (por norma um é suficiente) para fornecer rede Wi-Fi ou Ethernet ao servidor *Home Assistant*, aos clientes conectados ao ponto central de acesso para rede e ao/s *Thread Border Routers*.

Esta solução baseia-se em conseguir integrar e controlar dispositivos por *Matter over Thread*, e o objetivo é que após este cenário estar funcional, que facilmente dê para integrar quaisquer dispositivos pretendido que suporte *Matter* e *Thread* ao *Home Assistant* de forma a haver diversidade de opções.

3.2.Requisitos da solução

É suposto utilizarmos *Matter over Thread* para controlar duas tomadas e dois interruptores através do *Home Assistant*. Os requisitos da solução são os seguintes:

- Selecionar dispositivos que suportem *Matter* e *Thread* que possam ser integrados por *Matter over Thread*.
- Selecionar o *Thread Border Router* mais indicado para soluções de casas inteligentes.
- O *Home Assistant* e *Thread Border Router* terem conexão *Wi-Fi* ou *Ethernet* ao *Wi-Fi Access Point*
- Criar e configurar o *Home Assistant* num servidor.
- Criar e gerir uma rede ou mais redes *Thread*.
- Integrar dispositivos no *Home Assistant* por *Matter over Thread*.
- Controlar os dispositivos fisicamente e através da interface gráfica do *Home Assistant*.
- Sincronismo entre o estado físico do dispositivo e o estado do mesmo no *Home Assistant*.

3.3.Síntese

Neste capítulo utilizámos a pesquisa efetuada para propor uma solução em *Matter over Thread* que possa ser utilizada em casas inteligentes.

A solução proposta é composta por *Wi-Fi Access Points* para fornecer acesso *Wi-Fi* ou *Ethernet*, *Thread Border Routers* para estabelecer conexão *Thread*, softwares de automação doméstica para controlar dispositivos que suportem *Thread*, e dispositivos *Thread* que serão integrados e controlados pelos softwares de automação doméstica.

No próximo capítulo, com base em toda a pesquisa e a solução proposta iremos desenvolver um cenário que ambiciona integrar e controlar dispositivos em casas inteligentes por *Matter over Thread*.

4. Protótipo desenvolvido

Neste capítulo abordaremos os quatro cenários que desenvolvemos de forma incremental até obtermos o cenário final pretendido em *Matter over Thread*, do controlo de duas tomadas e dois interruptores através do *Home Assistant*.

4.1. Cenário de um LED em *Matter over Wi-Fi*

Para nos familiarizarmos com o *Matter*, desenvolvemos um cenário simples para integrar e controlar (ligar e desligar) um LED em *Matter over Wi-Fi* pelo *Home Assistant*. Decidimos começar por *Matter over Wi-Fi* por ser uma solução mais testada que *Thread*, e por não termos de criar e estabelecer uma rede *Thread* (desta forma focámo-nos maioritariamente no funcionamento do protocolo *Matter*).

4.1.1. Trabalho desenvolvido

No desenvolvimento deste projeto, começamos por instalar o *esp-idf*, que é a estrutura oficial de desenvolvimento da *Espressif* para os *ESP32* de forma a conseguirmos testar se o *ESP32-C6* estava funcional, e para conseguirmos instalar a biblioteca "*esp-matter*" que pertence à estrutura do *esp-idf*. Devido ao *esp-matter* não ser suportado pelo sistema operativo Windows (sistema operativo instalado nas nossas máquinas nativas), mas ser suportado em Linux, tivemos de instalar no *Visual Studio Code* a extensão *WSL* (*Windows Subsystem for Linux*).

Com o *WSL* instalado, já conseguimos instalar a biblioteca "*esp-matter*", que tem alguns exemplos desenvolvidos suportados pelo *ESP32-C6*. Analisámos os exemplos, e decidimos utilizar o exemplo "*blink*", que faz com que o LED embutido no *ESP32-C6* comece a piscar após o *upload* do código ser bem-sucedido (utilizámos este exemplo apenas para testar se o LED e o *ESP32-C6* estavam funcionais).

Quando tentámos fazer o *upload* do código, descobrimos que não conseguíamos selecionar a porta *usb-c* ao qual o *ESP32-C6* estava conectado, logo tivemos de partilhar a porta *usb-c* através da linha de comandos do *powershell* para conseguirmos fazer *upload* do exemplo no *ESP32-C6*. Partilhar a porta foi suficiente para conseguirmos falhar o *ESP32-C6*, e após o *upload* do código ter terminado o LED começou a piscar como pretendido

(demonstramos o LED ligado com o exemplo “blink” na Figura 23 - *ESP32-C6* com LED ligado).

Figura 23 - *ESP32-C6* com LED ligado

Voltámos a analisar os exemplos para ver qual era o exemplo mais indicado para ligar e desligar o LED. O exemplo “*light*” foi o exemplo que encontrámos mais indicado, fizemos *upload* desse exemplo, e após termos feito o *upload* para o *ESP32-C6*, o LED ligou (por omissão do código que foi feito *upload* no dispositivo), indicando que o estava tudo correto com o código.

Com estas tarefas concluídas, instalámos e configurámos o *Home Assistant* no *VirtualBox*. Na configuração, alterámos o adaptador de rede para “*Bridge*” de modo a haver conectividade entre o *Home Assistant* e o resto da rede, selecionámos a opção “*Allow All*” em relação ao “*Promiscuous Mode*” na placa de rede, e ativámos a opção “*Enable EFI*” (se não for ativado o *Home Assistant* não funciona).

Com o *Home Assistant* configurado, iniciámos o servidor, acedemos ao *Home Assistant* através do *URL* “*http://<IP-Servidor>:8123*” e definimos as credenciais de acesso. De seguida instalámos a extensão “*Matter Server*” (deixámos as configurações de omissão), de modo a conseguirmos fazer a integração de dispositivos que suportam *Matter*. Para a instalação da extensão, acedemos no *Home Assistant* ao separador de “*Settings*”, depois “*Add-ons*”, clicámos no botão “*ADD-ONS STORE*”, procurámos pelo *Matter Server* na barra de pesquisa e clicámos na opção “*INSTALL*”.

Por fim, instalámos a aplicação do *Home Assistant* no telemóvel (é recomendado ter a versão do sistema operativo no telemóvel atualizada), fizemos a conexão ao servidor que estava a correr na máquina virtual, iniciámos sessão na aplicação utilizando as credenciais previamente definidas, acedemos à extensão *Matter Server* e descobrimos que necessitávamos de um *QRCode* para integrar o dispositivo através da extensão.

Para criarmos o *QRCode*, utilizámos a linha de comandos fornecida pelo *esp-matter*, e através do comando "*matter onboardingcodes ble*" (comando incluído na biblioteca do *esp-matter*), criámos o *QRCode*. Fizemos scan do *QRCode* com a camara do telemóvel através da extensão "*Matter Server*", e a integração foi feita com sucesso. Para fazer scan do *QRCode* através do telemóvel, tem de se aceder ao *Home Assistant* no telemóvel, abrir o separador "*Settings*", clicar em "*Devices & Services*", "*ADD INTEGRATION*", escolher a opção "*Add Matter device*", clicar em "*No. Its's New*" (por estar a adicionar um dispositivo que não está adicionado a outro software de automação doméstica) e fazer scan do *QRCode* (se tudo correr bem o dispositivo irá aparecer no painel "*Overview*").

Após a integração ser bem-sucedida, como podemos visualizar na Figura 24 - LED integrado no *Home Assistant* referenciado por *TEST_PRODUCT*, conseguimos ligar e desligar o LED do *ESP32-C6* através do botão na interface gráfica do *Home Assistant*, tanto no telemóvel, como no computador (máquina virtual).

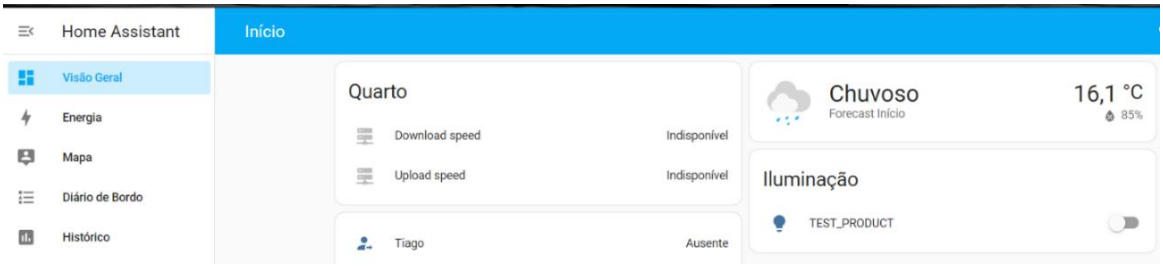

Figura 24 - LED integrado no *Home Assistant* referenciado por *TEST_PRODUCT*

4.1.2. Esquema do protótipo

No esquema do protótipo utilizamos a Figura 25 - Esquema do protótipo de integração e controlo de LED em *Matter over Wi-Fi* para demonstrar os componentes utilizados no cenário de integração e controlo de um LED em *Matter over Wi-Fi*.

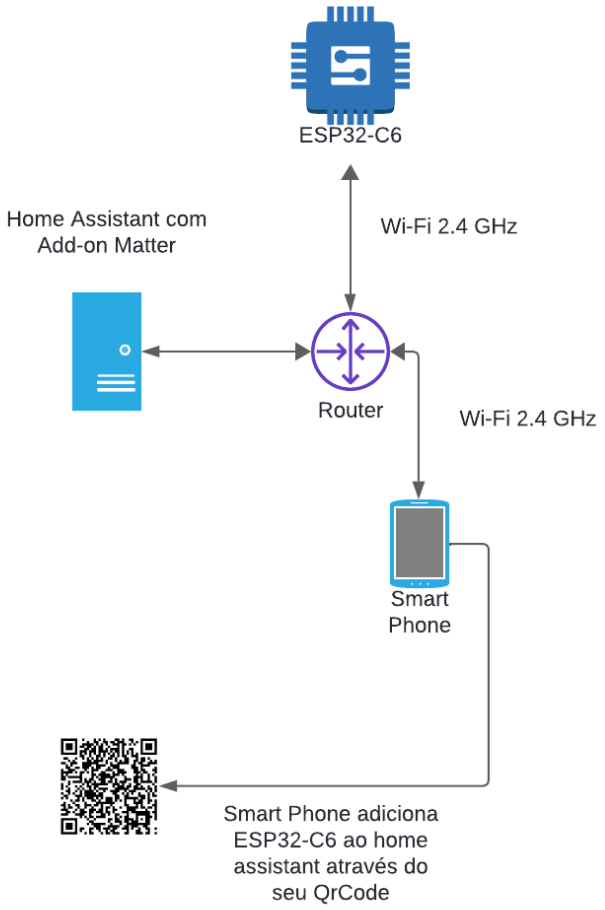

Figura 25 - Esquema do protótipo de integração e controlo de LED em *Matter over Wi-Fi*

- *ESP32-C6* - Dispositivo com o exemplo "*light*" da biblioteca do "*esp-matter*" que contém o LED, ao qual foi associado o *QRCode* para permitir a integração e o controlo do dispositivo no *Home Assistant* por *Matter over Wi-Fi*.
- *Home Assistant* - Servidor do *Home Assistant* que tem instalado a extensão "*Matter Server*" para permitir fazer a integração do *ESP32-C6*, e controlá-lo após a integração ser bem-sucedida.
- *Smart Phone* - Conectado ao servidor do *Home Assistant* por *Wi-Fi*, utilizado para através da extensão "*Matter Server*", integrar o *ESP32-C6* ao *Home Assistant* fazendo scan com a camara do telemóvel ao *QRCode*, e para controlar o LED após a integração feita.
- *QRCode* – Criado através da consola do "*esp-matter*" de forma a estar associado ao *ESP32-C6* para ser utilizado no processo de integração.
- *Router* – Dispositivo que permite a conexão e comunicação *Wi-Fi* com o S*mart Phone* e com o *Home Assistant*.

4.1.3. Fluxograma do controlo do LED por *Matter over Wi-Fi*

O fluxograma encontrado na Figura 26 - Fluxograma de controlo de um LED por *Matter over Wi-Fi*, exemplifica o comportamento do LED quando clicamos no botão da interface gráfica do *Home Assistant* conforme o estado do atributo *OnOff* (atributo do protocolo *Matter*).

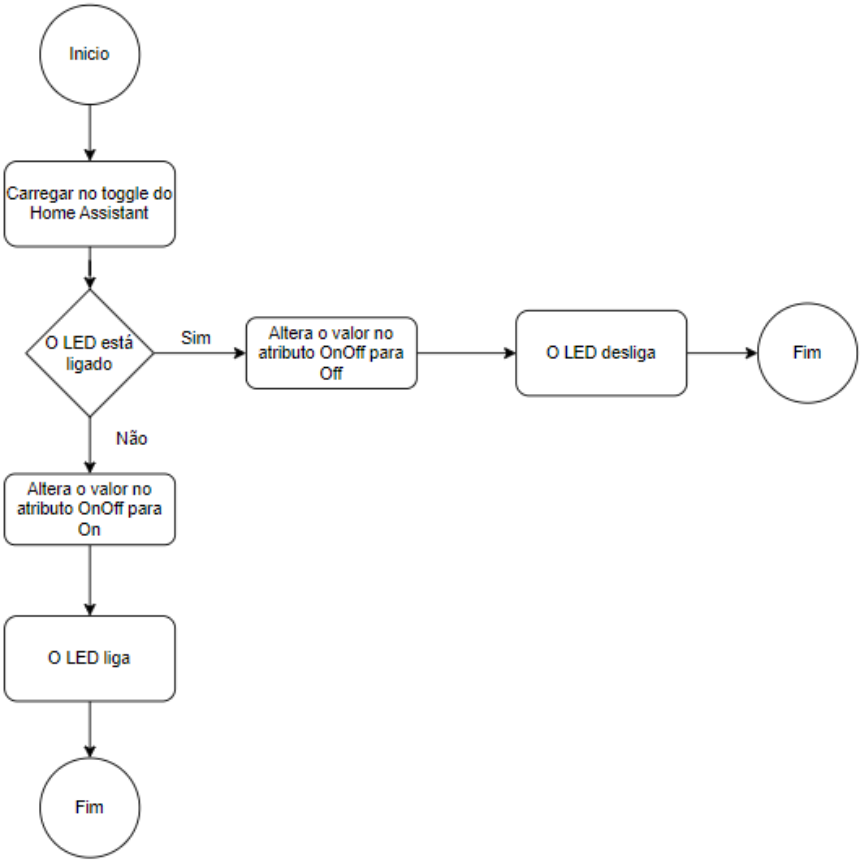

Figura 26 - Fluxograma de controlo de um LED por *Matter over Wi-Fi*

4.2. Cenário de um LED em *Matter over Thread*

Após termos implementado com sucesso o cenário em *Matter over Wi-Fi*, seguimos com a implementação do mesmo cenário (integração e controlo de LED através do *Home Assistant*), mas por *Matter over Thread*. Este cenário foi bastante mais complexo por ter mais fatores envolvidos, especialmente a criação e controlo da rede *Thread*.

4.2.1. Trabalho desenvolvido

O primeiro passo para fazer esta transição, foi alterar os parâmetros exemplo "*light*" através do "*Menuconfig*" (clicar na barra de pesquisa do *Visual Studio Code* e escrever "*>Menuconfig*"), para desativarmos a utilização do *Wi-Fi* e do *mDNS*, e ativar o *OpenThread*

como está apresentado na Figura 27 - Configurações do *Menuconfig* para o *ESP32-C6* utilizar *Thread*, para quando fizermos *upload* do código, o *ESP32-C6* utilizar *Thread* em vez de *Wi-Fi* (as alterações feitas no *Menuconfig* alteram o ficheiro *sdkconfig*). Após fazer *upload* do exemplo com as configurações alteradas para o ESP32-C6, este ficou pronto para ser utilizado como um dispositivo *Thread*.

Figura 27 - Configurações do *Menuconfig* para o *ESP32-C6* utilizar *Thread*

Como o adaptador *sonoff USB* foi produzido com o intuito de apenas ser utilizado para redes *ZigBee*, foi necessário alterar o *firmware* que este possuía. Assim, fizemos *upload* do *firmware* "*Multi-PAN*" (opção que suporta *Thread* e *Zigbee*) no adaptador *sonoff USB* para que suportasse o protocolo *Thread* (inicialmente o adaptador *Sonoff USB* foi desenvolvido para apenas suportar *Zigbee* e daí necessitarmos de fazer upload de novo *firmware*). Para fazermos *upload* do *firmware* acedemos ao *website*[1] (apenas suporta *Google Chrome* ou *Edge*), acedemos à secção "*ZBDongle-E*" demonstrado na Figura 28 - Secção da *ZBDongle-E* para fazer *upload* do *firmware Multi-PAN*, clicámos em "*Connect*", selecionámos a opção "*SONOFF Zigbee 3.0 USB Dongle Plus V2*", clicámos novamente em "*Connect*", "*CHANGE FIRMWARE*", selecionámos a opção "*Multi-PAN*", e clicámos em "*INSTALL*" para instalar o *firmware*, que demorou cerca de 5 minutos a instalar.

[1] https://darkxst.github.io/silabs-firmware-builder/

Figura 28 - Secção da *ZBDongle-E* para fazer *upload* do *firmware Multi-PAN*

Com o adaptador *sonoff USB* a suportar *Thread*, seguimos com a instalação e configuração da extensão necessária no *Home Assistant* designada por "*Silicon Labs Multiprotocol*", que foi utilizada para criar e gerir a rede *Thread* (para este cenário também necessitámos da extensão *Matter Server*, mas já estava instalada e configurada devido ao uso da mesma no cenário anterior). Para a instalação seguimos os mesmos passos da extensão *Matter Server*, e em termos de configuração, tivemos de adicionar o dispositivo que iriamos utilizar como *Thread Border Router* (*sonoff USB dongle*) como apresentado na Figura 29 - Dispositivo utilizado como *Thread Border Router*.

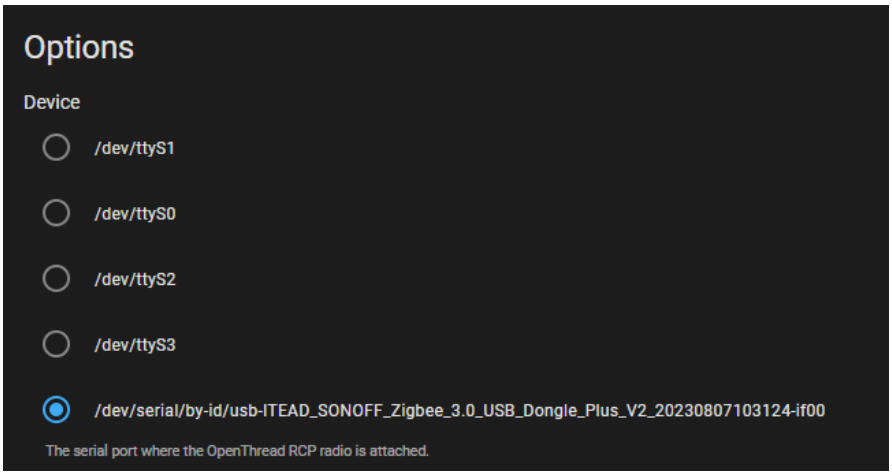

Figura 29 - Dispositivo utilizado como *Thread Border Router*

O problema foi que ao tentarmos adicionar o adaptador *sonoff USB dongle* como *Thread Border Router*, a mesma não aparecia nas opções, então pesquisámos, e descobrimos que não estava a ser detetado no *VirtualBox*. No *VirtualBox*, acedemos às configurações *USB* da máquina virtual (*Home Assistant*) e adicionámos o adaptador *sonoff USB dongle* como demonstrado na Figura 30 – Adicionar adaptador *sonoff USB dongle* ao *VirtualBox*.

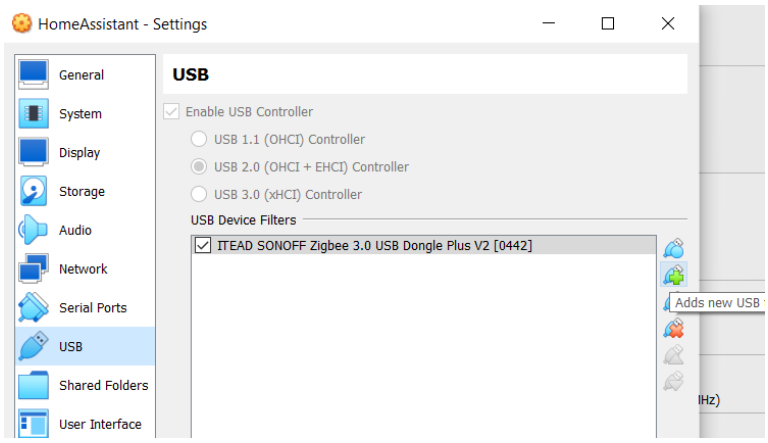

Figura 30 – Adicionar adaptador *sonoff USB dongle* ao *VirtualBox*

Reiniciámos a máquina virtual, acedemos novamente à extensão, e já tínhamos a opção para adicionar o adaptador *sonoff* como *Thread Border Router*. Após as extensões configuradas, iniciámo-las, e o *Silicon Labs Multiprotocol* automaticamente criou a rede *Thread* com a informação que podemos observar na Figura 31 - Informação relativa à rede *Thread*.

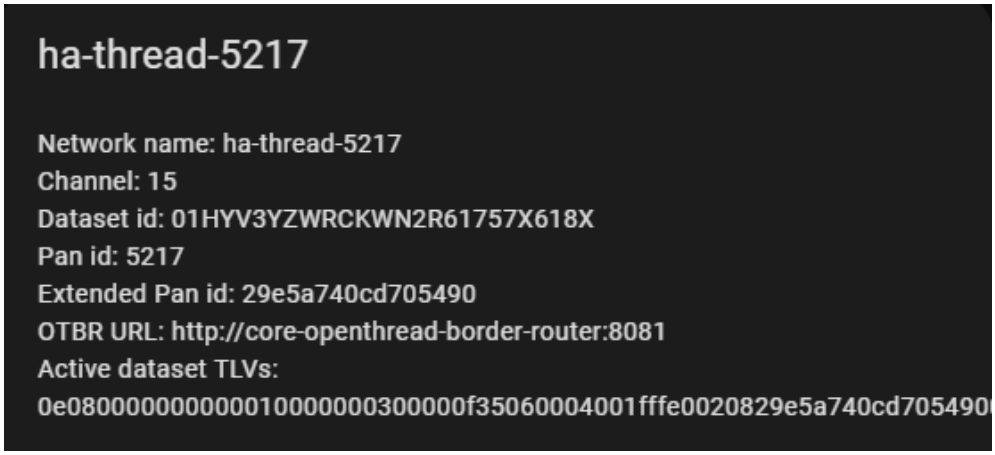

Figura 31 - Informação relativa à rede *Thread*

Com a rede *Thread* criada e o *ESP32-C6* com o respetivo código, tentámos integrar o dispositivo no *Home Assistant* através de *Matter over Thread* inúmeras vezes, mas obtínhamos continuamente o erro "*Thread Border Router Required*" em dispositivos iOS como é apresentado na Figura 32 - Erro apresentado em dispositivos iOS ao integrar LED por *Matter over Thread* e "*Can't connect to thread network*" em dispositivos Android como é apresentado na Figura 33 - Erro apresentado em dispositivos Android ao integrar LED por *Matter over Thread*.

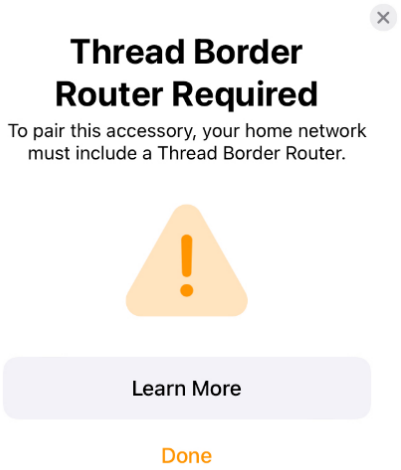

Figura 32 - Erro apresentado em dispositivos iOS ao integrar LED por *Matter over Thread*

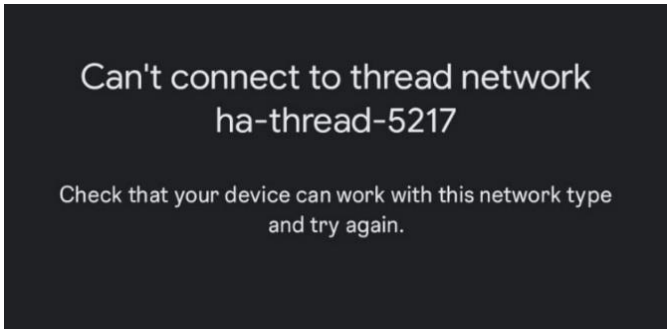

Figura 33 - Erro apresentado em dispositivos Android ao integrar LED por *Matter over Thread*

Decidimos dar um passo atrás e voltar a tentar integrar o LED por *Matter over Wi-Fi* (voltámos a alterar o *Menuconfig*), que nos surpreendeu por não conseguirmos integrar o LED mesmo seguindo os passos que tínhamos utilizado no cenário do LED em *Matter over Wi-Fi* (obtínhamos o erro "*Unable to Add Accessory*" em IOS e "*Can't find device*" em Android) como podemos observar através da Figura 34 - Erro apresentado em dispositivos iOS ao integrar LED por *Matter over Wi-Fi* e da Figura 35 - Erro apresentado em dispositivos Android ao integrar LED por *Matter over Wi-Fi*.

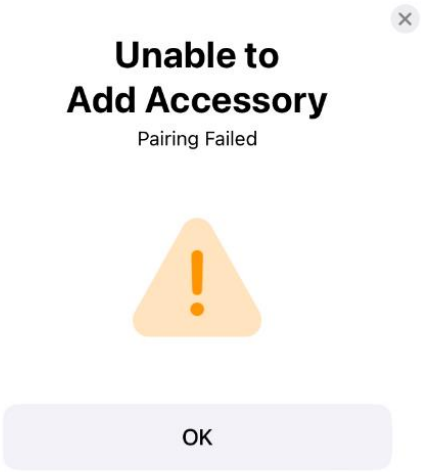

Figura 34 - Erro apresentado em dispositivos iOS ao integrar LED por *Matter over Wi-Fi*

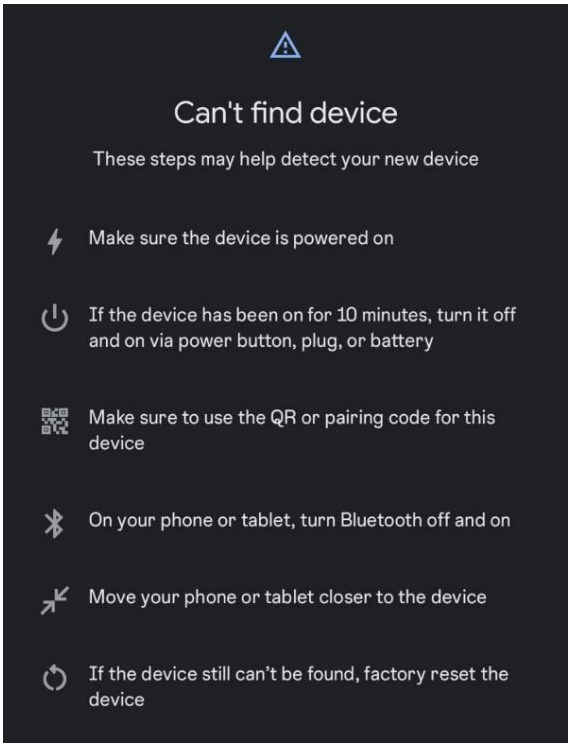

Figura 35 - Erro apresentado em dispositivos Android ao integrar LED por *Matter over Wi-Fi*

Após várias tentativas e muita pesquisa, descobrimos na documentação do *esp-idf* o comando "*matter device factoryreset*", que reinicia por completo *ESP32-C6*. Fizemos *upload* de novo do código para o dispositivo, na linha de comandos do *esp-matter* inserimos o comando, tentámos integrar de novo o dispositivo e a integração foi bem-sucedida. Voltámos a tentar integrar sem o comando, e com o comando, para termos a certeza que era este o problema, e tirámos a conclusão que antes de tentar integrar o dispositivo, tínhamos de utilizar sempre o comando "*matter device factoryreset*".

Com este problema resolvido, alterámos de novo as configurações do *Menuconfig*, fizemos *upload* do código para o ESP32-C6, utilizámos o comando para reiniciar o *ESP32-C6*, voltámos a tentar integrar o LED (a rede *Thread* já estava criada), mas voltámos a obter o mesmo erro ("*Thread Border Router Required*" em iOS e "*Can't connect to thread network*" em Android).

Analisámos todas as opções possíveis, e não conseguíamos descobrir nenhuma solução em concreto, então tentámos primeiramente alterar as diferentes opções de *mDNS* presentes no *Menuconfig* (não vimos alteração nenhuma), e secundariamente fizemos *upload* do *firmware* para o adaptador *sonoff USB* com a opção *OpenThread* em vez de *Multi-PAN*. A extensão *Silicon Labs Multiprotocol* não suporta a nova opção de *firmware* instalada no adaptador *sonoff USB*, então desinstalámos essa extensão, e instalámos e configurámos a extensão *Open Thread Border Router* (por termos visto que causava menos problemas).

Em termos de configuração da extensão *Open Thread Border Router*, selecionámos o adaptador *sonoff USB* (processo igual ao *Silicon Labs Multiprotocol*) como *Thread Border Router* e abrimos dois portos (8080 e 8081) como demonstrado na Figura 36 - Portos abertos para vermos a topologia da rede *Thread* para conseguirmos ver a topologia da rede *Thread*.

Figura 36 - Portos abertos para vermos a topologia da rede *Thread*

Estas alterações não resolveram o nosso problema, e quando tentávamos integrar o LED, o *Thread Border Router* nem aparecia na topologia de rede demonstrada no *http://<IP-Servidor>:8080* no separador "*Topology*". Voltámos a pesquisar sobre possíveis problemas com o nosso cenário, e descobrimos um grupo de *Discord* vocacionado apenas para *Matter* e *Thread*, onde referenciaram que para integrar dispositivos através de *Matter over Thread* tínhamos de importar as credencia da rede *Thread* para o telemóvel antes do processo de integração iniciar (métodos diferentes para *iOS* e *Android*).

O método para importar as credenciais em dispositivos *Android* é bastante intuitivo. Na aplicação do *Home Assistant* seguimos o caminho "*Settings*", clicando de seguida em

"*Companion app",* fomos até ao fim da página e clicámos em "Solução de Problemas", onde por último carregámos em "A sincronizar credenciais *Thread*" e selecionámos o servidor do qual queríamos importar as credenciais.

Para importar as credencias em dispositivos *iOS* tivemos de utilizar um script por ainda não haver uma opção no *Home Assistant* para importar diretamente as credenciais *Thread*. Começámos por instalar a extensão "*Studio Code Server*", através dos seguintes passos: acedemos à página "*Settings*", escolhemos a opção "*Add-ons*", clicamos em "*ADD-ONS STORE*", procurámos por "*Studio Code Server*" na barra de pesquisa, selecionámos a extensão, e clicámos em "*INSTALL*". Para a extensão aparecer na barra lateral, acedemos à extensão e selecionámos a opção "*Show in sidebar*". Acedemos à extensão através da barra lateral, criámos uma pasta designada por "*www*", e um ficheiro designado por "*call-external-message-bus.js*". Ao ficheiro adicionámos o código fornecido no *github*[2], guardámos o ficheiro, e reiniciámos a máquina virtual. Com o código inserido no *Home Assistant*, procedemos em criar um painel novo, então acedemos à página "*Settings*", escolhemos a opção "*Dashboards*", clicámos em "*ADD DASHBOARD*", selecionámos a opção "*Default Dashboard*", no título escrevemos "*Thread*", e clicámos em "*CREATE*". Para associarmos o código ao painel, tivemos de criar um recurso externo onde o ficheiro do código é chamado, e para isto, na página "*Dashboards*" clicámos nos três pontos no canto superior direito, clicámos em "*Resources*", clicámos no botão "*ADD RESOURCE*", no *URL* inserimos "*/local/call-external-message-bus.js*" para chamar o ficheiro com o código, e clicámos em "*CREATE*". Com o recurso externo criado, tivemos de associar o recurso ao painel "*Thread*", então, primeiramente criámos o cartão da seguinte forma: acedemos ao painel "*Thread*" (clicámos em *Thread* na barra lateral), clicámos no *icon* de editar no canto superior direito, clicámos em "*ADD CARD*", selecionámos o primeiro cartão e clicámos em "*SAVE*". Secundariamente, para chamarmos o recurso externo clicámos em "*EDIT*", adicionámos o código demonstrado na Figura 37 - Código do cartão no *Home Assistant* para importar credenciais *Thread*, e clicámos em "*SAVE*". Com estes passos todos concluídos, já foi possível importarmos as credenciais *Thread* no telemóvel iOS, acedendo ao painel *Thread* e clicando no botão "*Call Store In Platform Keychain*".

[2] https://gist.github.com/bmight/49315bbb004871d66445b1588cd1d82e#file-call-external-message-bus-js

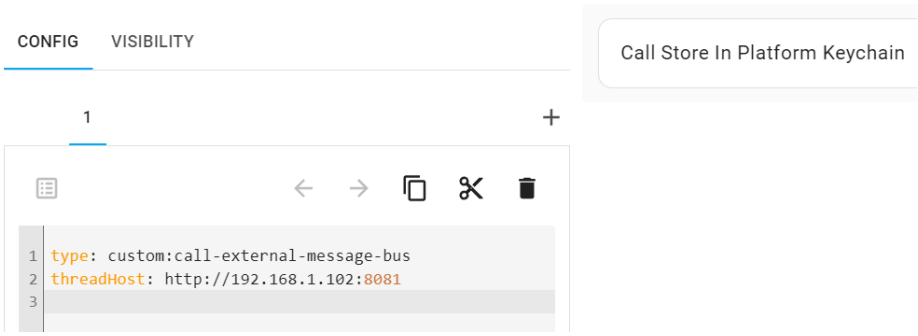

Figura 37 - Código do cartão no *Home Assistant* para importar credenciais *Thread*

O *Home Assistant* para Android já tem uma opção para integrar as credenciais de forma direta acedendo às seguintes páginas, nesta ordem: "*Settings*", "*Companion App*", "*Troubleshooting*", selecionar a opção "*Sync Thread Credentials*", e selecionar o servidor *Home Assistant* que está a ser utilizado, no caso de ter mais que um no telemóvel. Após selecionar a opção, as credenciais são importadas.

Com as credenciais de rede importadas no telemóvel conseguimos ultrapassar o erro "*Thread Border Router Required*" e "*Can´t connect to thread network*", e na topologia de rede já aparecia o adaptador *sonoff USB* como *Thread Router*, mas não aparecia o LED como *End Device*, e após cerca de dois minutos a tentar fazer a integração apareceu o erro "*Unable to Add Accessory*" e" *Can't find device*" (mesmo erro que aparecia quando não reiniciávamos o ESP32-C6, mas desta vez o erro não derivava desse fator). Tentámos continuamente integrar o LED desta forma, mas continuámos a obter o mesmo erro.

Havia a possibilidade de o erro ser derivado da rede *Thread*, então criámos outra rede *Thread* e descobrimos outro erro. Ao tentarmos integrar o LED no *Home Assistant*, voltámos a obter o erro "*Thread Border Router Required*" e "*Can´t connect to thread network*".

Para sabermos se este erro derivava do telemóvel ou da nova rede *Thread*, experimentámos integrar com outro telemóvel que ainda só tinha a nova rede *Thread* associada, e apenas aparecia o erro "*Unable to Add Accessory*" e "*Can't find device*". (ultrupassou o erro "*Thread Border Router Required*" e "*Can´t connect to thread network*"). Concluímos que só pode haver uma rede *Thread* associada ao telemóvel, ou então iriamos obter o erro "*Thread Border Router Required*". A nova rede *Thread* não nos resolveu o erro anterior, mas descobrimos outro possível erro que é possível ter ao integrar dispositivos por *Matter over Thread*.

Ainda sem soluções e como este erro não era nada explicito, ligámos o *wireshark* e capturámos os pacotes todos durante o processo de integração, mas não encontrámos nenhuma anomalia. Reinstalámos o Home Assistant, seguindo todos os passos feitos anteriormente, mas também não resolveu o problema, então decidimos analisar os *logs* das extensões do *Home Assistant*, e reparámos num *log* a indicar um erro na extensão *Matter Server* relacionado com o *mDNS* que não aparecia anteriormente (quando aparecia o erro "*Thead Border Router Required*" e "*Can´t connect to thread network"*). O erro é demonstrado na Figura 38 - Erro relacionado com o *mDNS* nos *logs* do *Matter Server*.

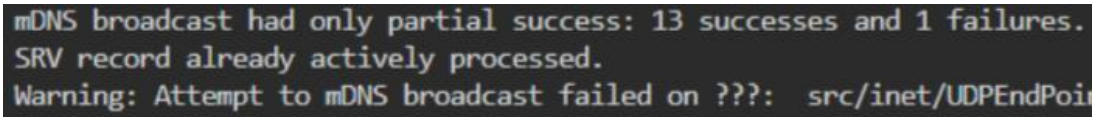

Figura 38 - Erro relacionado com o *mDNS* nos *logs* do *Matter Server*

Associámos que este erro poderia estar relacionado com dois problemas, ou o tráfego *mDNS* não estar a ser transmitido corretamente na rede, ou não estar a ser transmitido para o *Home Assistant* por estar instalado numa máquina virtual. Configurámos um *Mikrotik* router de raiz para termos a certeza de que não havia tráfego a ser bloqueado, e instalámos o *Home Assistant* num *mini pc* para o software não estar instalado em ambientes virtualizados.

Para configurar o *Mikrotik* começámos por definir um *pool ipv4* e *ipv6* como apresentado na Figura 39 - *Pool* de endereços *IPv4* e na Figura 40 - *Pool* de endereços *IPv6* de modo a criar um *DHCP Server* para ambos os protocolos.

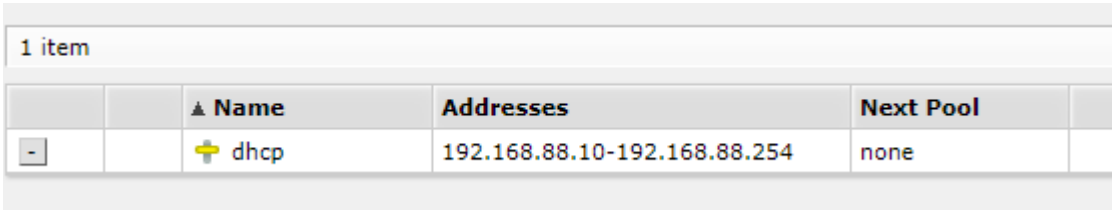

Figura 39 - *Pool* de endereços *IPv4*

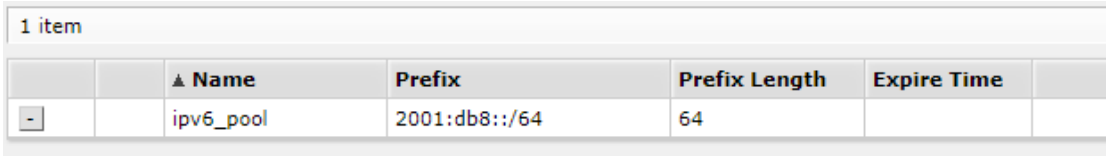

Figura 40 - *Pool* de endereços *IPv6*

Após estas *pools* estarem criadas, foram utilizadas para o *DHCP Server* de modo a os dispositivos conectados conseguirem obter um endereço *Ipv4* e *Ipv6* automaticamente. A

configuração dos mesmos encontra-se representada na Figura 40 - Configuração do *DHCP* Server para *Ipv4* e Figura 41 - Configuração do *DHCP* Server para *Ipv6*.

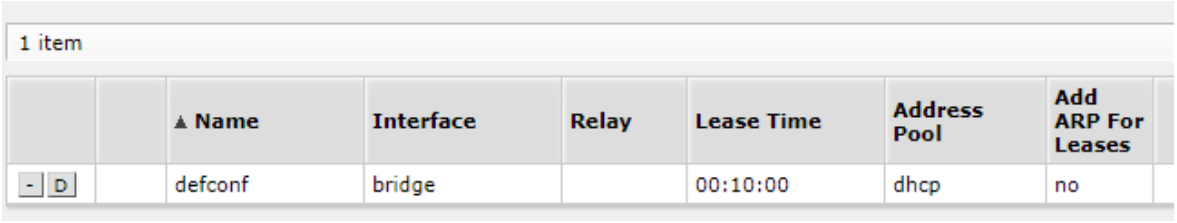

Figura 40 - Configuração do *DHCP* Server para *Ipv4*

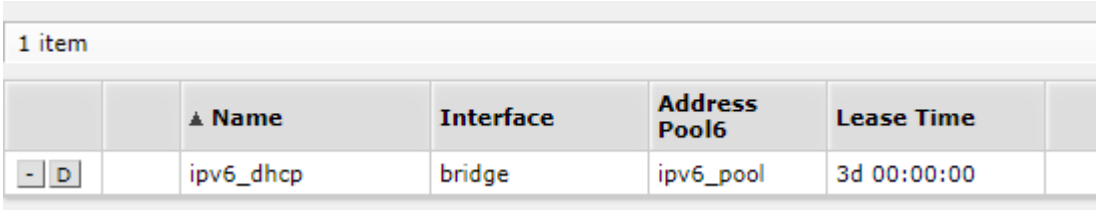

Figura 41 - Configuração do *DHCP* Server para *Ipv6*

Finalmente, foram adicionados servidores *DNS*, utilizando os *IPs* dos servidores da Google e do próprio router como retrata a Figura 42 - Servidores *DNS*.

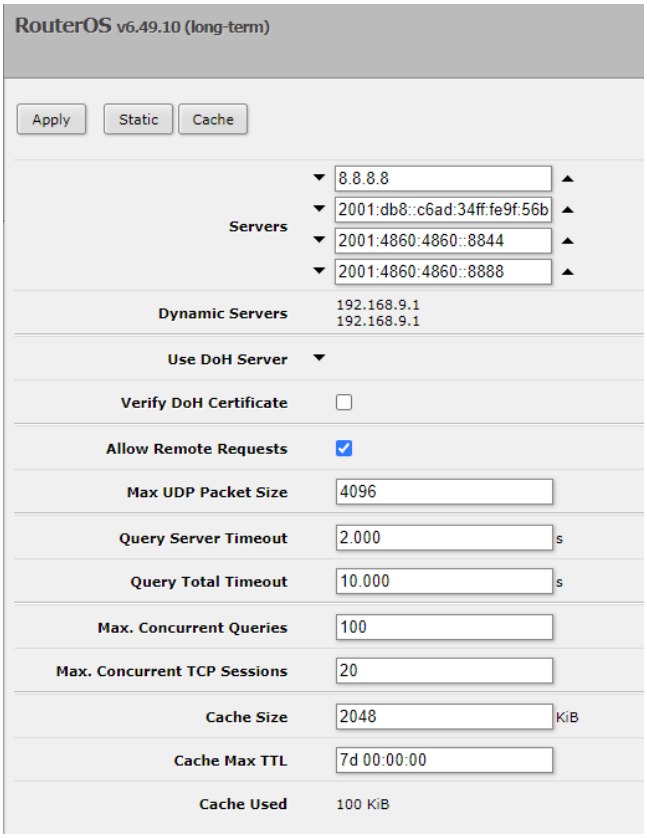

Figura 42 - Servidores *DNS*

Como por vezes havia falha de comunicação ao efetuar *pings* através do endereço *Ipv6*, adicionámos rotas estáticas como está representado na Figura 43 - Rotas *IPv6* para garantir a comunicação.

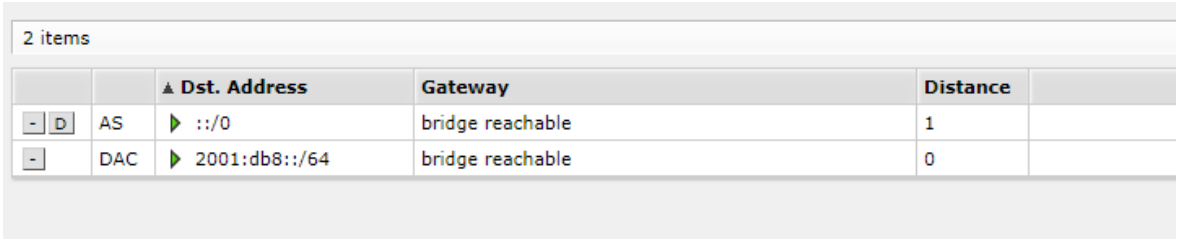

Figura 43 - Rotas *IPv6*

Após o *Mikrotik router* estar configurado e nos podermos conectar à rede, podemos observar através da Figura 44 - Teste à configuração do *Mikrotik router*, que já conseguimos obter *Ipv4* e *Ipv6*. Assim, prosseguimos para a instalação e configuração do *Home Assistant* no *mini PC*.

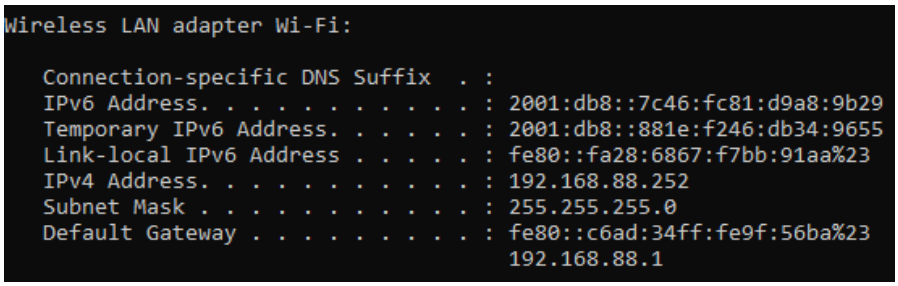

Figura 44 - Teste à configuração do *Mikrotik router*

A instalação do *Home Assistant* numa máquina física pode ser feita de duas maneiras. A primeira maneira, e a que foi utilizada é através do Sistema Operativo *Ubuntu*, sendo necessário uma *pen USB* que contenha este Sistema Operativo. No segundo método, a imagem do *Home Assistant* é escrita diretamente no *boot medium* através do programa "*balenaEtcher*". O segundo método não foi utilizado pois era necessário remover o disco do *mini pc* e utilizar um adaptador para ligar via *USB* ao computador que iria realizar a escrita do *Home Assistant* no respetivo disco.

Utilizando o primeiro método, inserimos a *pen USB* na máquina onde pretendemos instalar o *Home Assistant*, e fizemos *boot* através da mesma (pode ser necessário selecionar a *pen USB* como o *boot device* na *BIOS*). Após selecionar a opção "*Try Ubuntu*", dirigimo-

nos ao website[3] para fazer download da imagem do *Home Assistant* e abrimos a aplicação *Disks* já presente no *Ubuntu*.

De seguida selecionámos a opção "*Restore Disk Image*", como está representado na Figura 45 - Selecionar a opção que nos permite escrever a imagem do *Home Assistant* no nosso disco e escolhemos a imagem do *Home Assistant* ao qual tínhamos feito download previamente.

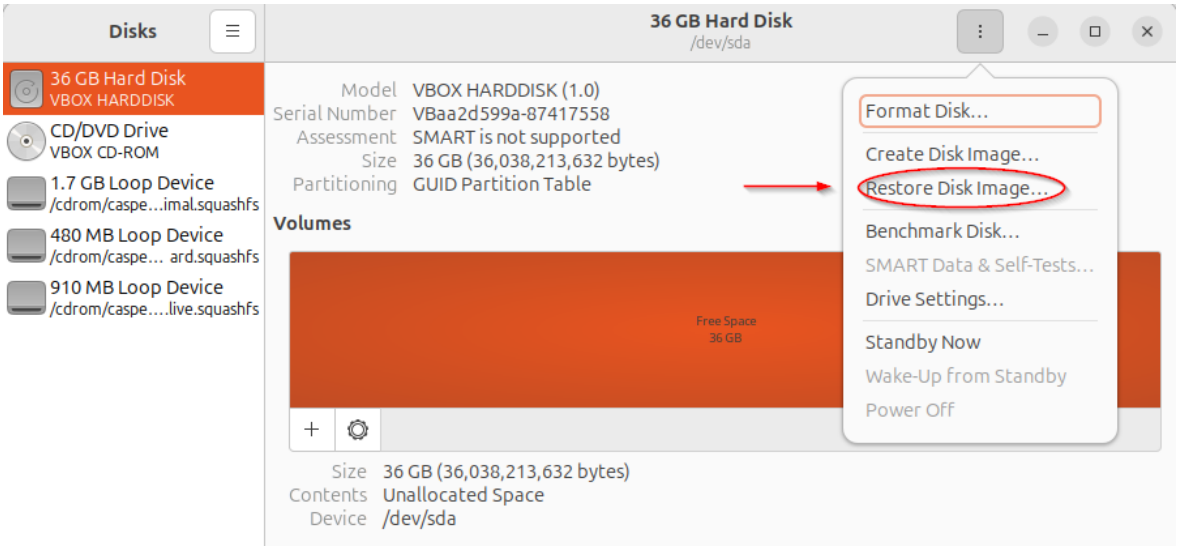

Figura 45 - Selecionar a opção que nos permite escrever a imagem do *Home Assistant* no nosso disco

Após confirmarmos que o ficheiro correto foi selecionado pelo menu apresentado na Figura 46 - Menu de confirmação após selecionar a respetiva imagem, começámos o processo de *restore*. É essencial fazer um backup ao disco antes de realizar este processo, pois toda a informação será apagada quando o *restore* terminar.

[3] Download realizado no website: https://github.com/home-assistant/operating-system/releases/

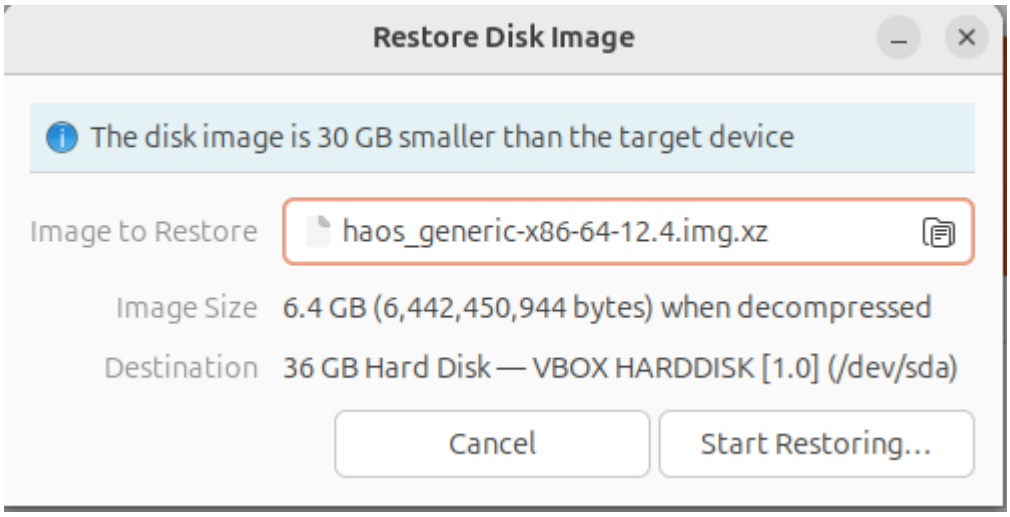

Figura 46 - Menu de confirmação após selecionar a respetiva imagem

Finalmente, conseguimos ver o progresso deste procedimento no menu do programa *Disks*, representado na Figura 47 - *Restore* da imagem do disco. Quando este processo terminou, desligámos a máquina e retiramos a *pen USB*. Ao ligarmos a máquina, o *Home Assistant* ligou, e seguimos o mesmo procedimento de instalação e configuração utilizado previamente na máquina virtual.

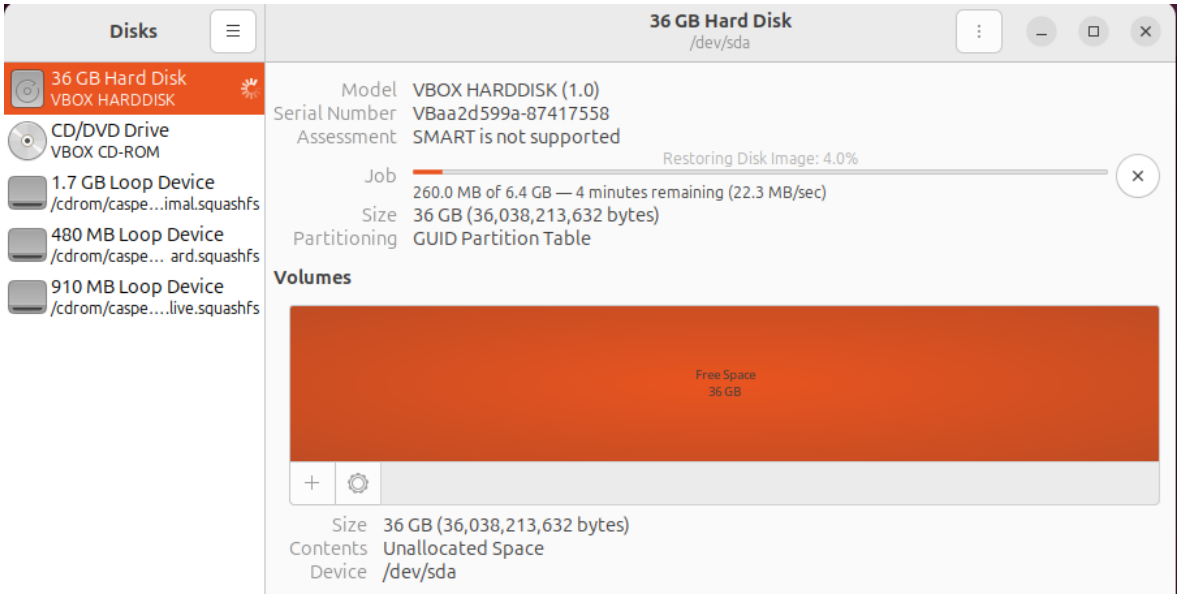

Figura 47 - *Restore* da imagem do disco

Com o *Mikrotik router* a atuar como *Wi-Fi Access Point*, e o *Home Assistant* instalado num ambiente não virtualizado, tentámos fazer a integração e conseguimos com sucesso. Na página da topologia de rede já aparecia o daptador *sonoff USB* como *Leader* e o LED como *Child* (*End Device*), como demonstrado na Figura 48 - Topologia da rede *Thread* como *Leader* e *End Device*.

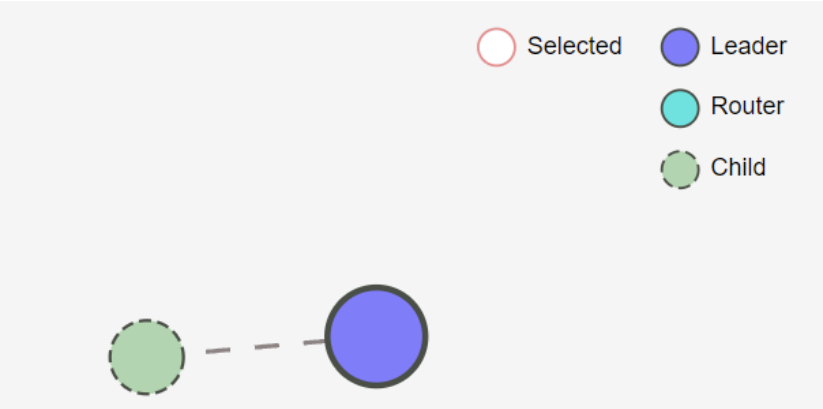

Figura 48 - Topologia da rede *Thread* como *Leader* e *End Device*

Após a integração feita com sucesso, tal como no cenário de *Matter over Wi-Fi*, também conseguimos ligar e desligar o LED do *ESP32-C6*, mas neste cenário por *Matter over Thread*.

4.2.2. Esquema do protótipo

No esquema do protótipo utilizamos a Figura 49 - Esquema do protótipo de integração de um LED em *Matter over Thread* para demonstrar os componentes utilizados no cenário de integração e controlo de um LED em *Matter over Thread*.

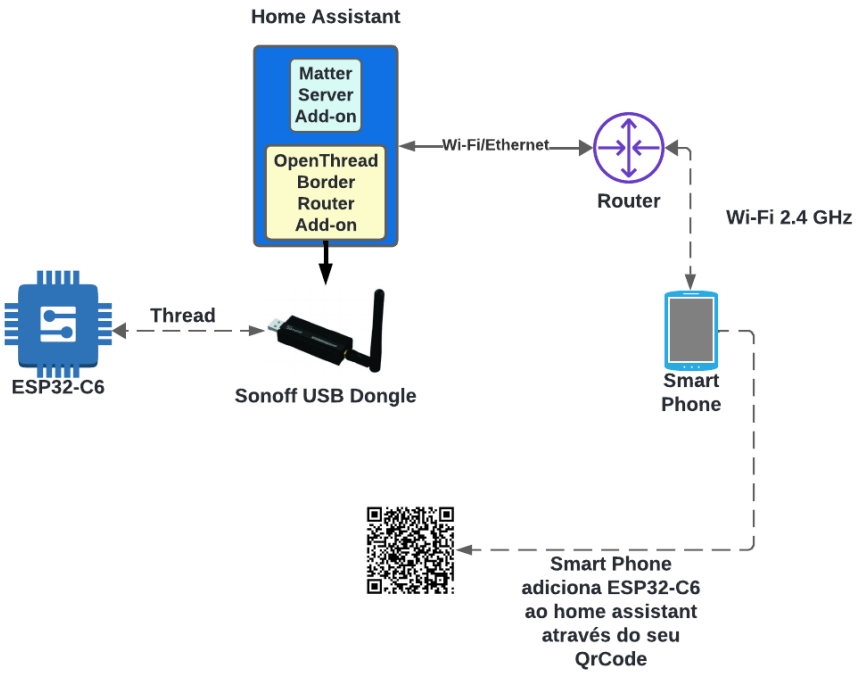

Figura 49 - Esquema do protótipo de integração de um LED em *Matter over Thread*

- *ESP32-C6* – Dispositivo com o exemplo "*light*" da biblioteca do "*esp-matter*" para o qual foram alteradas as configurações de modo a utilizar *Thread* e associado o *QRCode* para permitir a integração e o controlo do dispositivo no *Home Assistant* por Matter over Thread.
- Home Assistant – Servidor do *Home Assistant* que tem instalado as extensões "*Matter Server*" e "*Open Thread Border Router*" para permitir fazer a integração e o controlo do *ESP32-C6*.
- *Smart Phone* – Conectado ao servidor do *Home Assistant* utilizado para através da extensão "*Matter Server*" integrar o *ESP32-C6* ao *Home Assistant* fazendo scan com a camara do telemóvel ao *QRCode* e para controlar o led após a integração feita.
- *QRCode* – Criado através da consola do "*esp-matter*" de forma a estar associado ao *ESP32-C6* para ser utilizado no processo de integração.
- *Sonoff USB Dongle* – Dispositivo ligado ao *Home Assistant* e que atua como *Thread Border Router*.
- *Router* – Dispositivo que permite a conexão e comunicação *Wi-Fi* com o *Smart Phone*, com o *Home Assistant*, e com o adaptador *sonoff USB Dongle*.

4.2.3. Fluxograma do controlo do LED por *Matter over Thread*

O fluxograma encontrado na Figura 50 - Fluxograma do controlo de um LED em *Matter over Thread* exemplifica o comportamento do LED quando clicamos no botão da interface gráfica do *Home Assistant* conforme o estado do atributo *OnOff* (atributo do protocolo *Matter*).

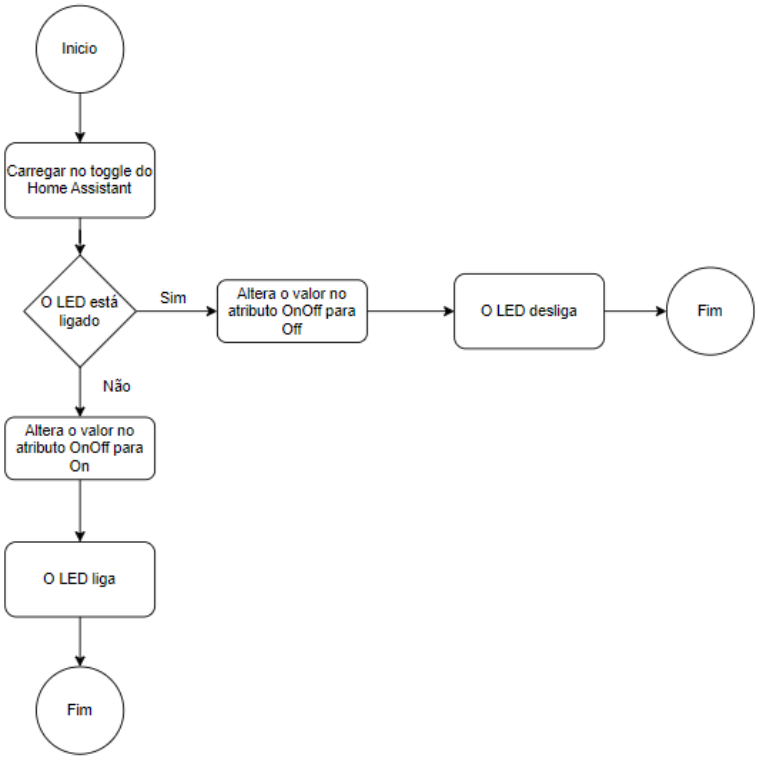

Figura 50 - Fluxograma do controlo de um LED em *Matter over Thread*

4.2.4. Discussão

Descobrimos muitos fatores a manter em consideração quando se está a desenvolver cenários em *Matter over Thread*.

Primeiramente, um tópico mais geral, temos de reiniciar sempre o *ESP32-C6* antes de tentar integrar o dispositivo num software de automação doméstica com o comando "*matter device factoryreset*".

Secundariamente, é mais indicado utilizar o *firmware OpenThread* no adaptador *sonoff USB* e a extensão *Open Thread Border Router* no *Home Assistannt* para criar e gerir a rede *Thread* por ter menos problemas.

Terciariamente, temos de importar sempre as credenciais da rede *Thread* para o telemóvel antes do processo de integração ser inicializado (processo diferente em iOS e Android).

Por fim, convém que o *Wi-Fi Access Point* permita que todo o tráfego na rede possa ser transmitido sem qualquer regras a bloqueá-lo (o mais comum é utilizar *vlans* e ter o *Home Assistant* e o *Wi-Fi Access Point* em *vlans* diferentes), e que o *Home Assistant* não esteja em

ambientes virtualizados devido ao *mDNS* não conseguir ser transmitido corretamente para ambientes virtuais.

4.3. Cenário de duas tomadas e dois interruptores em *Matter over Wi-Fi*

De seguida a termos conseguido integrado o LED por *Matter over Thread*, decidimos adicionar ao cenário duas tomadas, dois interruptores e dois relés (os relés são utilizados para controlar a energia passada para as tomadas). Pela experiência que tivemos nos outros dois cenários, concluímos que a integração e o controlo por *Matter over Thread* é mais complexo, então optámos por nos focar no desenvolvimento do código para podermos integrar e controlar as duas tomadas e os dois interruptores por *Matter over Wi-Fi*.

4.3.1. Trabalho desenvolvido

Primeiramente, começámos por fazer as conexões físicas necessárias para os interruptores, as tomadas, os relés, o *ESP32-C6*, o carregador (utlizado para estar sempre a fornecer energia ao *ESP32-C6*), e a fonte de energia do carregador.

Para os dois interruptores, fizemos uma ligação ao *GND*, atribuímos o primeiro ao *gpio pin 9* e o segundo ao *gpio pin 18* do *ESP32-C6*. Para os relés, o *VCC* ficou ligado ao *5V*, o *GND* ao *GND*, e ligámos um relé (*IN1*) ao *gpio pin 19* e o outro (*IN2*) ao *gpio pin 20*. Para as tomadas, ligámos o neutro e o *GND* à fonte de energia, e o *VCC* ao relé.

Com as ligações todas feitas, analisámos de novo todos os exemplos da biblioteca *esp-matter*, mas concluímos que o exemplo "*light*" seria de novo o melhor exemplo para utilizarmos, por ter código comentado que permitia controlar o LED do *ESP32-C6* através de um botão (bastava definir o *gpio pin* do botão, e era possível ligar e desligar o LED). No ficheiro *sdkconfig* atribuímos o variável relativa ao botão ao *gpio pin 9* (*gpio pin* ao qual o primeiro interruptor está conectado, para ser possível que esse interruptor controle o LED) como podemos observar na Figura 51 - Código para definir o *gpio pin* do primeiro interruptor, e deixámos o *gpio pin* de omissão para o LED por omissão.

```
# Button 1
#
CONFIG_BSP_BUTTON_1_TYPE_GPIO=y
# CONFIG_BSP_BUTTON_1_TYPE_ADC is not set
CONFIG_BSP_BUTTON_1_GPIO=9
CONFIG_BSP_BUTTON_1_LEVEL=0
# end of Button 1
```

Figura 51 - Código para definir o *gpio pin* do primeiro interruptor

Fizemos *upload* deste código para o *ESP32-C6*, e após o *upload* ter sido bem-sucedido, ao clicarmos no primeiro interruptor, o LED ligava e desligava.

O próximo objetivo foi, em vez de apenas conseguirmos ligar e desligar o LED com apenas o primeiro interruptor, utilizarmos os dois interruptores. Adicionámos outro botão novamente no ficheiro *sdkconfig*, ao qual atribuímos o *gpio pin 18* (*gpio pin* conectado ao segundo interruptor) como mostra a Figura 52 - Código para definir o *gpio pin* do segundo interruptor, e alterámos a função "*app_driver_button_init*" para incluir a inicialização do segundo botão.

```
# Button 2
#
CONFIG_BSP_BUTTON_2_TYPE_GPIO=y
# CONFIG_BSP_BUTTON_2_TYPE_ADC is not set
CONFIG_BSP_BUTTON_2_GPIO=18
CONFIG_BSP_BUTTON_2_LEVEL=0
# end of Button 2
# end of Buttons
```

Figura 52 - Código para definir o *gpio pin* do segundo interruptor

Após o *upload* do código para o *ESP32-C6*, testámos clicar nos interruptores, e independentemente do interruptor que clicássemos o LED ligava/desligava conforme o seu estado (estado do atributo *OnOff*).

Com os dois interruptores funcionais, o próximo passo foi conseguir controlar um dos relés em vez do led. Alterámos o *gpio pin* de omissão do LED para o *gpio pin 19* (*gpio pin* relativo ao primeiro relé), como demonstrado na Figura 53 - Código para definir o *gpio pin* do primeiro relé, e apenas através desta alteração conseguimos ligar e desligar o relé com ambos os interruptores.

```
CONFIG_BSP_LED_TYPE_RGB=y
CONFIG_ENV_MAX_LEDS=1
CONFIG_BSP_LEDS_NUM=1
CONFIG_BSP_LED_RGB_GPIO=19
CONFIG_BSP_ESP_IDF_VERSION="$ESP_IDF_VERSION"
CONFIG_BSP_LED_RGB_BACKEND_RMT=y
```

Figura 53 - Código para definir o *gpio pin* do primeiro relé

Seguindo a mesma lógica dos interruptores, queríamos conseguir controlar os dois relés através dos dois interruptores. Ao tentarmos adicionar outro *gpio pin* para o segundo relé, descobrimos que o primeiro relé estava associado a um tipo de LED designado por *Adressable RGB LED*, e neste exemplo, apenas suportava um *gpio pin*, então tivemos de alterar o tipo de LEDs no *Menuconfig* para *GPIO LEDs* (suporta mais que um), para podermos adicionar dois *gpio pins* (um com o *gpio pin 19* e o outro com o *gpio pin 20*), como demonstrado na Figura 54 - Código para definir os *gpio pins* dos relés).

```
# LED 1
#
CONFIG_BSP_LED_1_GPIO=19
CONFIG_BSP_LED_1_LEVEL=1
# end of LED 1

#
# LED 2
#
CONFIG_BSP_LED_2_GPIO=20
CONFIG_BSP_LED_2_LEVEL=1
```

Figura 54 - Código para definir os *gpio pins* dos relés

Após esta alteração, fizemos *upload* do código, e os relés não ligaram. Fizemos uma pesquisa sobre a inicialização dos *GPIO LEDS* e percebemos que têm uma forma diferente de serem inicializados relativamente aos *Addressable RGB LED*. Logo, alterámos a função "*app_driver_light_init*" (este é o nome de omissão do exemplo, mas neste caso inicializa os relés) para permitir inicializar ambos os relés corretamente em modo desligado. O código para inicializar os relés está demonstrado na Figura 55 - Código para inicializar os relés.

```
app_driver_handle_t app_driver_light_init()
{
#if CONFIG_BSP_LEDS_NUM > 0
    /* Initialize led */
    //return (app_driver_handle_t)leds[0];
    esp_rom_gpio_pad_select_gpio((gpio_num_t)19);
    esp_rom_gpio_pad_select_gpio((gpio_num_t)20);

    /* Set GPIO direction as output */
    gpio_set_direction((gpio_num_t)19, GPIO_MODE_OUTPUT);
    gpio_set_direction((gpio_num_t)20, GPIO_MODE_OUTPUT);

    /* Turn off relays initially */
    gpio_set_level((gpio_num_t)19, 0);
    gpio_set_level((gpio_num_t)20, 0);
    return (app_driver_handle_t)ESP_OK;
#else
    return NULL;
#endif
}
```

Figura 55 - Código para inicializar os relés

Após a alteração desta função, alterámos a função "*app_driver_light_set_power*", mais especificamente, adicionámos o *gpio_set_level((gpio_num_t),gpio pin, 1)*. Tivemos de alterar esta função pelo facto do comando de ligar e desligar os *GPIO LEDS* diferir do comando de ligar e desligar os *Adressable RGB LEDs*. Com a alteração desta função, independentemente do interruptor que clicássemos, ambos os relés já ligavam.

Com ambos os relés e ambos os interruptores ligados, o objetivo era agora conseguir associar o primeiro interruptor ao primeiro relé, e o segundo interruptor ao segundo relé, de forma que ambos os interruptores não alterassem o estado de ambos os relés.

Começámos por criar uma variável global designada por "*relay_to_control*" (para guardar o número 1 ou 2 conforme o interruptor clicado) inicializada com o valor 0, e que quando se clicava num dos interruptores, o valor da variável passa-se a ser 1 ou 2 conforme o interruptor clicado. Desta forma, já era possível detetar o interruptor clicado, e assim também já era possível alterar apenas a relé associado ao interruptor.

De seguida, criámos quatro *endpoints* no ficheiro "*app_main.cpp*" para os quatro equipamentos (dois interruptores e dois relés) de modo a ser possível alterar os valores no *Home Assistant* como demonstrado na Figura 56 - Criação de *endpoints* para os interruptores e para os relés.

```
uint16_t relay1_endpoint = 2;
uint16_t relay2_endpoint = 3;
uint16_t button1_endpoint = 0;
uint16_t button2_endpoint = 1;
```

Figura 56 - Criação de *endpoints* para os interruptores e para os relés

No código fornecido pelo exemplo "*light*" existia uma função para atualizar o valor do botão. Adaptámos essa função para duas designadas por "*app_driver_button_relay1_cb*" e "*app_driver_button_relay2_cb*", para detetar qual o interruptor clicado, atribuindo o valor 1 ou 2 à variável global conforme o interruptor e atualizar o estado no *Home Assistant*. Na Figura 57 - Código que deteta o interruptor clicado e altera o valor da variável *relay_to_control* apenas demonstramos a função "*app_driver_button_relay1_cb*", mas a função "*app_driver_button_relay2_cb*" é muito idêntica, apenas muda o "*relay_to_control=1*" e o respetivo *endpoint* ao qual se pretende alterar o estado.

```
static void app_driver_button_relay1_cb(void *arg, void *data)
{
    ESP_LOGI(TAG, "Button for Relay 1 pressed");

    uint16_t endpoint_id = relay1_endpoint;
    uint32_t cluster_id = OnOff::Id;
    uint32_t attribute_id = OnOff::Attributes::OnOff::Id;

    node_t *node = node::get();
    endpoint_t *endpoint = endpoint::get(node, endpoint_id);
    cluster_t *cluster = cluster::get(endpoint, cluster_id);
    attribute_t *attribute = attribute::get(cluster, attribute_id);

    esp_matter_attr_val_t val = esp_matter_invalid(NULL);
    attribute::get_val(attribute, &val);
    val.val.b = !val.val.b;
    relay_to_control = 1;
    attribute::update(endpoint_id, cluster_id, attribute_id, &val);
}
```

Figura 57 - Código que deteta o interruptor clicado e altera o valor da variável *relay_to_control*

Estas funções são chamadas quando os interruptores são inicializados, então alterámos a função "*app_driver_button_init*" e atribuímos a cada interruptor a função respetiva.

Para os interruptores e os relés atualizarem o seu estado de modo a ficarem iguais quando o estado de um dos dispositivos é alterado a partir do *Home Assistant*, foi necessário

implementar uma função representada na Figura 58 - Função utilizada para alterar o estado do relé e sincronizar o estado no *Home Assistant*. Esta função altera o estado do relé quando o estado do respetivo interruptor é alterado, e vice-versa.

```
if (cluster_id == OnOff::Id && attribute_id == OnOff::Attributes::OnOff::Id) {
    if (endpoint_id == relay1_endpoint) {
        relay_to_control = 1;
        app_driver_button_button1_cb(NULL,NULL);
        err = app_driver_relay_set_power(handle, val);
    } else if (endpoint_id == button1_endpoint) {
        relay_to_control = 1;
        app_driver_button_relay1_cb(NULL,NULL);
        err = app_driver_relay_set_power(handle, val);
    }
    else if (endpoint_id == relay2_endpoint) {
        relay_to_control = 2;
        app_driver_button_button2_cb(NULL,NULL);
        err = app_driver_relay_set_power(handle, val);
    }
    else if( endpoint_id == button2_endpoint) {
        relay_to_control = 2;
       app_driver_button_relay2_cb(NULL,NULL);
       err = app_driver_relay_set_power(handle, val);
    }


}
```

Figura 58 - Função utilizada para alterar o estado do relé e sincronizar o estado no *Home Assistant*

Para os relés estarem sincronizados quando os interruptores fossem ligados fisicamente, também alterámos a função de inicialização dos botões, como mostra a Figura 59 - Função para inicializar o segundo botão, atualizando o estado no *Home Assistant*. A inicialização do primeiro botão é semelhante, sendo que a única alteração é no nome das funções.

```
app_driver_handle_t app_driver_button2_init()
{

    button_handle_t btns[BSP_BUTTON_NUM];
    ESP_ERROR_CHECK(bsp_iot_button_create(btns, NULL, BSP_BUTTON_NUM));
    ESP_ERROR_CHECK(iot_button_register_cb(btns[1], BUTTON_PRESS_DOWN, app_driver_button_button2_cb, NULL));
     ESP_ERROR_CHECK(iot_button_register_cb(btns[1], BUTTON_PRESS_DOWN, app_driver_button_relay2_cb, NULL));
    return (app_driver_handle_t)btns[0];

}
```

Figura 59 - Função para inicializar o segundo botão

Por fim, fizemos uma condição na função "*app_driver_light_set_power*", para alterar o estado do relé (para ligar/desligar o relé) conforme o valor da variável "*relay_to_control*" (para apenas alterar o valor do relé associado ao interruptor clicado). A estrutura lógica da função está demonstrada na Figura 60 - Estrutura condicional para ligar/desligar o relé conforme o interruptor clicado.

```
static esp_err_t app_driver_light_set_power(led_indicator_handle_t handle, esp_matter_attr_val_t *val)
{
#if CONFIG_BSP_LEDS_NUM > 0
    esp_err_t err = ESP_OK;

    if(relay_to_control == 1) {
        gpio_set_level((gpio_num_t)19, val->val.b ? 1 : 0); // Control Relay 1
    } else if(relay_to_control == 2) {
        gpio_set_level((gpio_num_t)20, val->val.b ? 1 : 0); // Control Relay 2
    }
```

Figura 60 - Estrutura condicional para ligar/desligar o relé conforme o interruptor clicado

Fisicamente, o cenário já estava a funcionar como suposto, ou seja, cada interruptor associado a cada relé, e o estado dos relés a alterar conforme o interruptor clicado.

O objetivo seguinte foi conseguir obter o mesmo comportamento por parte dos interruptores e dos relés, mas de forma virtual através do *Home Assistant*. Para isto, decidimos integrar no *Home Assistant* o dispositivo composto pelos dois interruptores e pelas duas tomadas para testar se o cenário funcionava como pretendíamos.

O processo de integração foi bem-sucedido à primeira tentativa, mas como tínhamos o cenário todo atribuído a um node, apenas um relé estava a ligar assim que a integração era feita. Ao clicar nos botões do *Home Assistant* o estado dos relés não alterava, e fisicamente, independentemente do interruptor clicado apenas o estado do relé que era inicializado alterava.

Para conseguirmos que o cenário agisse através do controlo do *Home Assistant* da mesma forma que estava a agir quando o estávamos a controlar fisicamente, tivemos de fazer novamente alterações ao código.

Criámos duas funções designadas por "*app_driver_relay1_init*" e "*app_driver_relay2_init*" para inicializar os reles (têm de ser inicializados desligados). Apenas demonstramos a função "*app_driver_relay1_init*" na Figura 61 - Código de

inicialização do relé, mas a diferença entre ambas as funções é que numa está definido o *gpio pin 19*, e na outra está definido o *gpio pin 20*.

```
app_driver_handle_t app_driver_relay1_init()
{
#if CONFIG_BSP_LEDS_NUM > 0

    esp_rom_gpio_pad_select_gpio((gpio_num_t)19);

    gpio_set_direction((gpio_num_t)19, GPIO_MODE_OUTPUT);

    /* Turn off relays initially */
    gpio_set_level((gpio_num_t)19, 1);
    return (app_driver_handle_t)ESP_OK;
#else
    return NULL;
#endif
}
```

Figura 61 - Código de inicialização do relé

Para os interruptores e os relés serem inicializados quando são integrados no *Home Assistant*, tivemos de chamar as funções criadas anteriormente no ficheiro "*app_main.cpp*". Para as podermos chamar, primeiramente, tivemos de as referenciar no ficheiro "*app_priv.h*" como demonstrado na Figura 62 - Referenciar as funções de inicialização dos relés e dos interruptores no ficheiro "*app_priv.h*", e apenas após este passo completo, as podemos chamar, senão não eram reconhecidas no ficheiro "*app_main.cpp*".

```
app_driver_handle_t app_driver_relay1_init();
app_driver_handle_t app_driver_relay2_init();

app_driver_handle_t app_driver_button1_init();
app_driver_handle_t app_driver_button2_init();
```

Figura 62 - Referenciar as funções de inicialização dos relés e dos interruptores no ficheiro "*app_priv.h*"

Para conseguirmos controlar os relés de forma separada (cada interruptor a controlar cada relé) criamos dois nodes (um node por cada par de relés e interruptores) como demonstrado na Figura 63 - Código de criação de dois nodes para os relés e associámos os quatro *endpoints* criados anteriormente aos dois *nodes* de modo ao primeiro interruptor ficar no mesmo *node* que a primeira tomada e o segundo interruptor ficar no mesmo *node* que a segunda tomada.

```
// node handle can be used to add/modify other endpoints.
node_t *node1 = node::create(&node_config1, relay1_attribute_update_cb, app_identification_cb);
ABORT_APP_ON_FAILURE(node1 != nullptr, ESP_LOGE(TAG, "Failed to create Matter node"));

node_t *node2 = node::create(&node_config2, relay2_attribute_update_cb, app_identification_cb);
ABORT_APP_ON_FAILURE(node2 != nullptr, ESP_LOGE(TAG, "Failed to create Matter node"));
```

Figura 63 - Código de criação de dois nodes para os relés

Após termos feito estas alterações no código, integrámos os relés e os interruptores no *Home Assistant* e ficou a funcionar como pretendido. Conseguíamos ligar e desligar o relé da direita que por sua vez ligava e desligava a tomada da direita através do interruptor físico da direita ou o botão da interface gráfica associado ao interruptor da direita como mostra a Figura 64 - Resultado obtido após ligar o relé 2 ou interruptor 2, para o interruptor e relé da esquerda, o comportamento é semelhante como mostra a Figura 65 - Resultado após ligar o relé 1 ou interruptor 1.

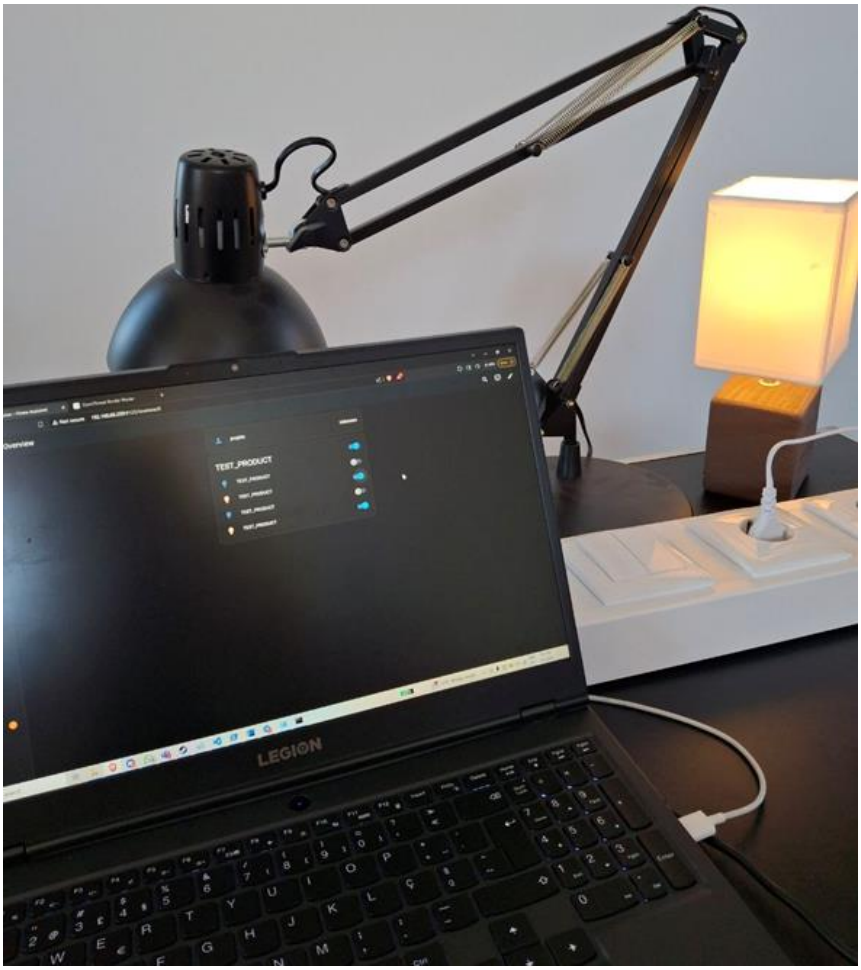

Figura 64 - Resultado obtido após ligar o relé 2 ou interruptor 2

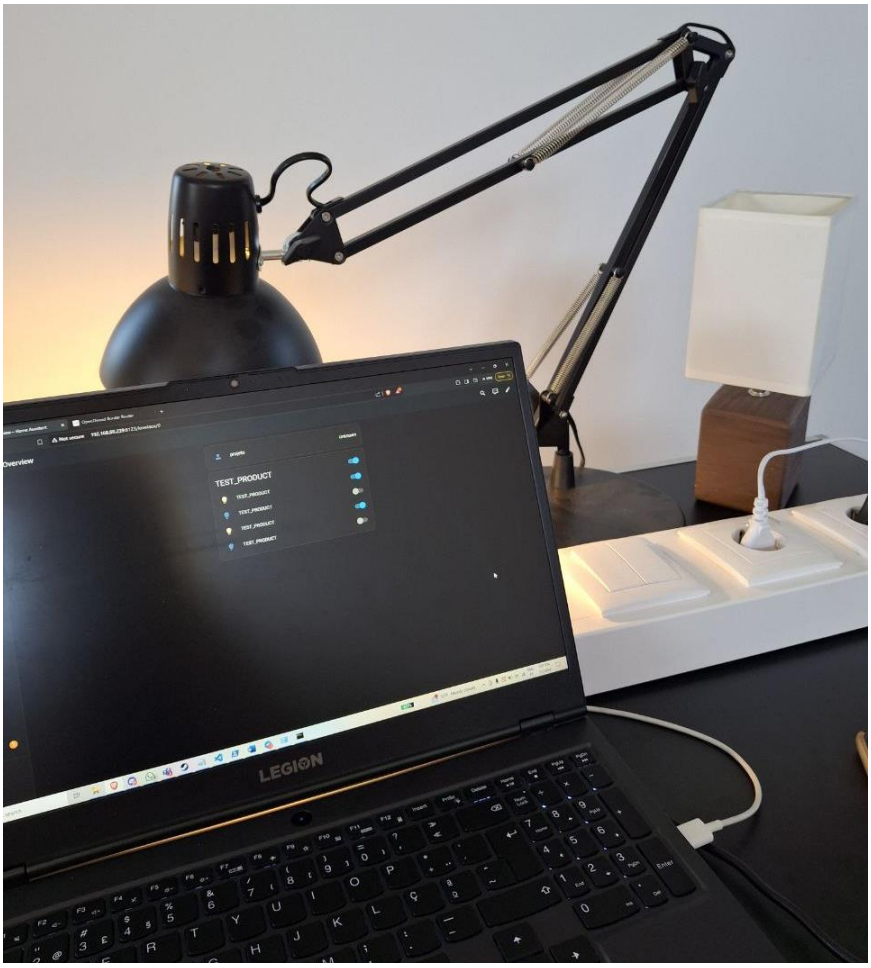

Figura 65 - Resultado após ligar o relé 1 ou interruptor 1

Fizemos um esquema representado na Figura 66 - Estrutura do *Matter* no cenário de duas tomadas e dois interruptores em *Matter over Wi-Fi* para demonstrar como os *nodes*, os *endpoints* e os *clusters* ficaram associados.

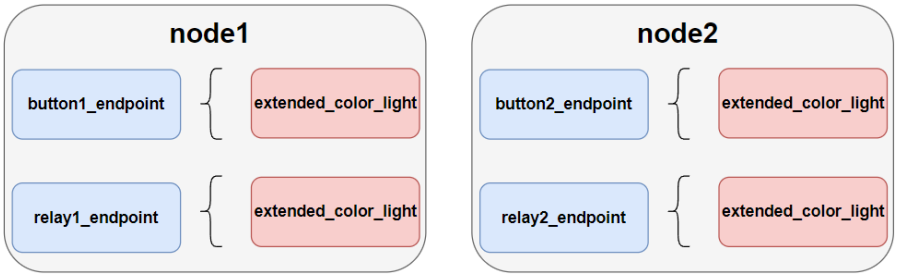

Figura 66 - Estrutura do *Matter* no cenário de duas tomadas e dois interruptores em *Matter over Wi-Fi*

O *node1* e o *node2* são relativos à forma como os interruptores e as tomadas estão associados. Os *endpoints* referem-se aos interruptores e às tomadas, e como podemos observar, o primeiro interruptor (*button1_endpoint*) e a primeira tomada (*relay1_endpoint*)

estão associados por estarem no mesmo *node* (*node1*), e o mesmo se aplica para os *endpoints* associados ao *node2*. O *extended_color_light* refere-se ao *cluster*, que já era o *cluster* utilizado por omissão no exemplo "*light*", que contém o atributo *OnOff* que foi o atributo necessário para o desenvolvimento do cenário.

4.3.2. Esquema do protótipo

Na Figura 67 - Representação das ligações entre os componentes do cenário está representado as ligações das tomadas, dos relés, dos interruptores, do *ESP32-C6*, do carregador, e da fonte de energia, para explicarmos mais ao detalhe e de forma visual como as ligações foram feitas. As outras componentes utilizadas para integrar e controlar as duas tomadas e os dois interruptores através de *Matter over Wi-Fi*, foram o *Wi-Fi Access Point*, o telemóvel, o *Home Assistant* com a extensão *Matter Server*, o *QRCode* e o *ESP32-C6*, tal como no primeiro cenário da integração e controlo do LED em *Matter over Wi-Fi*.

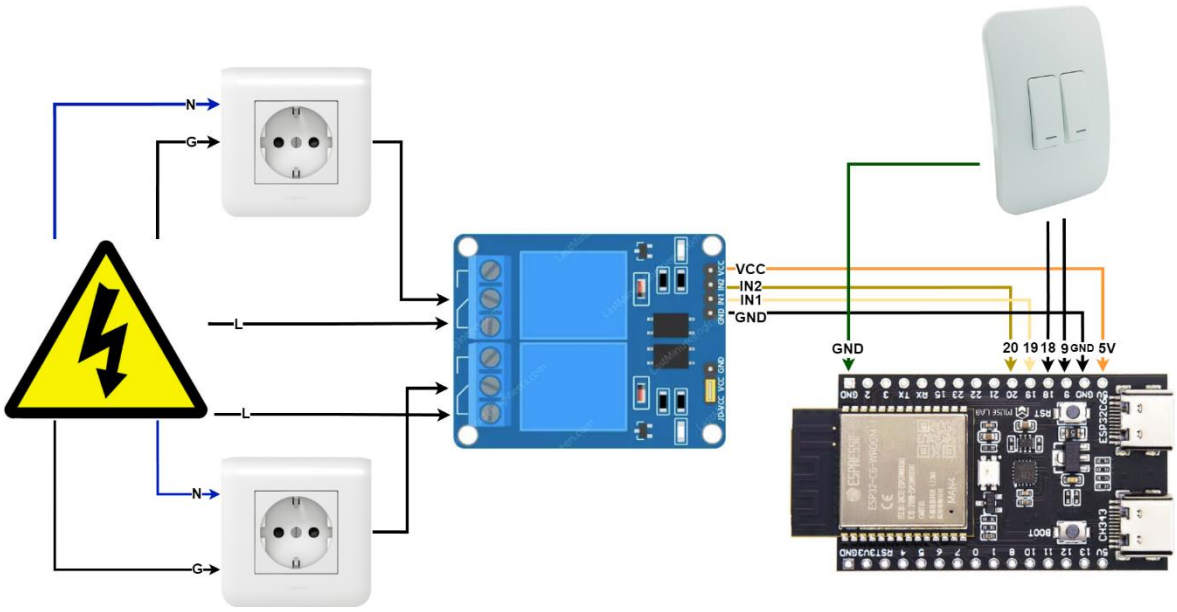

Figura 67 - Representação das ligações entre os componentes do cenário

Na representamos a arquitetura da solução e todos os equipamentos que utilizámos neste cenário.

4.3.3. Fluxogramas do cenário do controlo de duas tomadas e dois interruptores por *Matter over Wi-Fi*

O fluxograma encontrado na Figura 68 - Fluxograma do controlo da primeira tomada em *Matter over Wi-Fi* exemplifica o comportamento da primeira tomada (o mesmo comportamento reflete-se para a segunda tomada, mas os dispositivos que ligam e desligam

são o segundo interruptor, a segunda tomada e o segundo relé) através do *Home Assistant* (não é controlável fisicamente).

O fluxograma encontrado na Figura 69 - Fluxograma do controlo do primeiro interruptor em *Matter over Wi-F* exemplifica o comportamento do primeiro interruptor (o mesmo comportamento reflete-se para o segundo interruptor, mas os dispositivos que ligam e desligam são o segundo interruptor, a segunda tomada e o segundo relé) tanto fisicamente como através do *Home Assistant*.

Os estados de todos os equipamentos alteram conforme o estado do atributo *OnOff* do protocolo *Matter*.

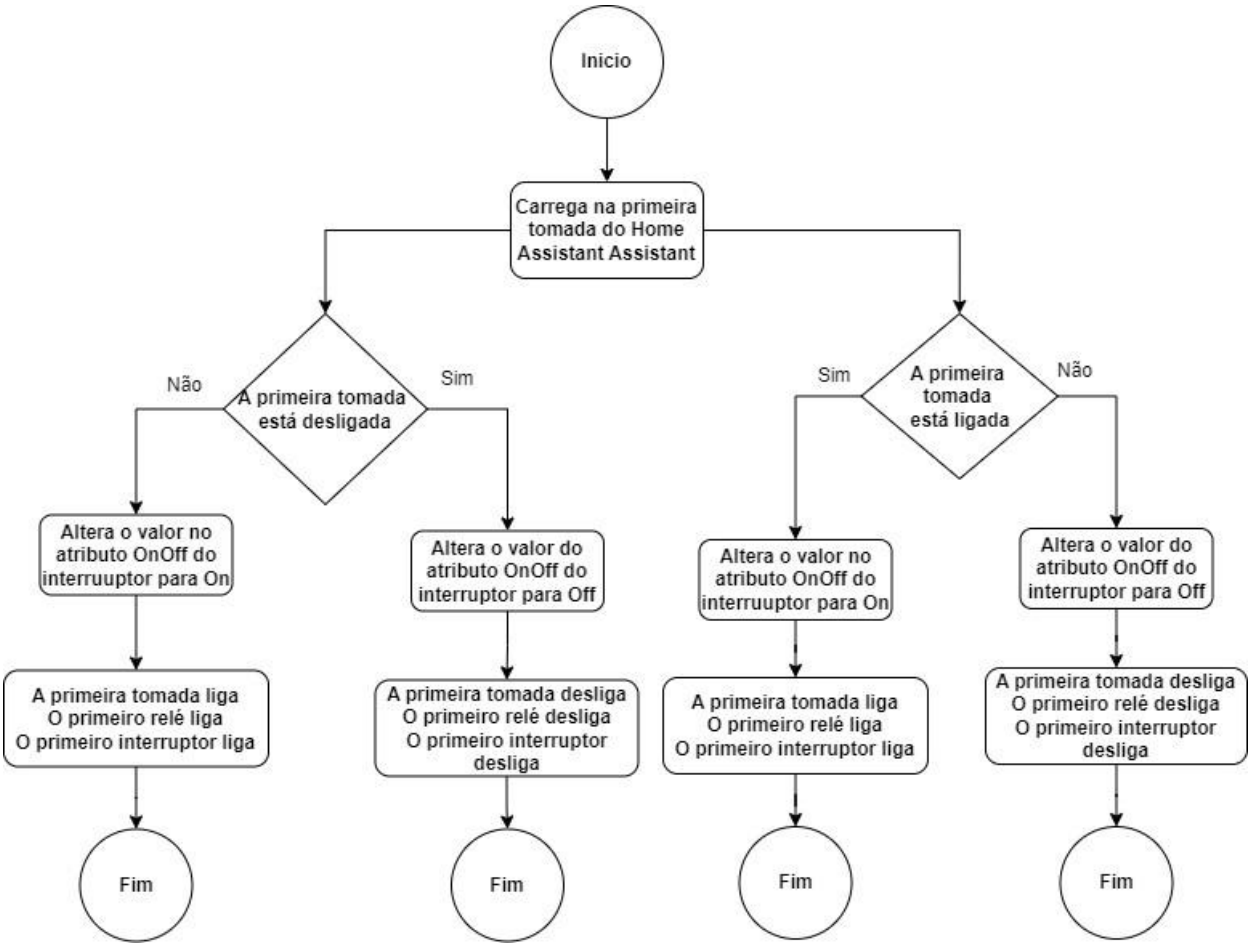

Figura 68 - Fluxograma do controlo da primeira tomada em *Matter over Wi-Fi*

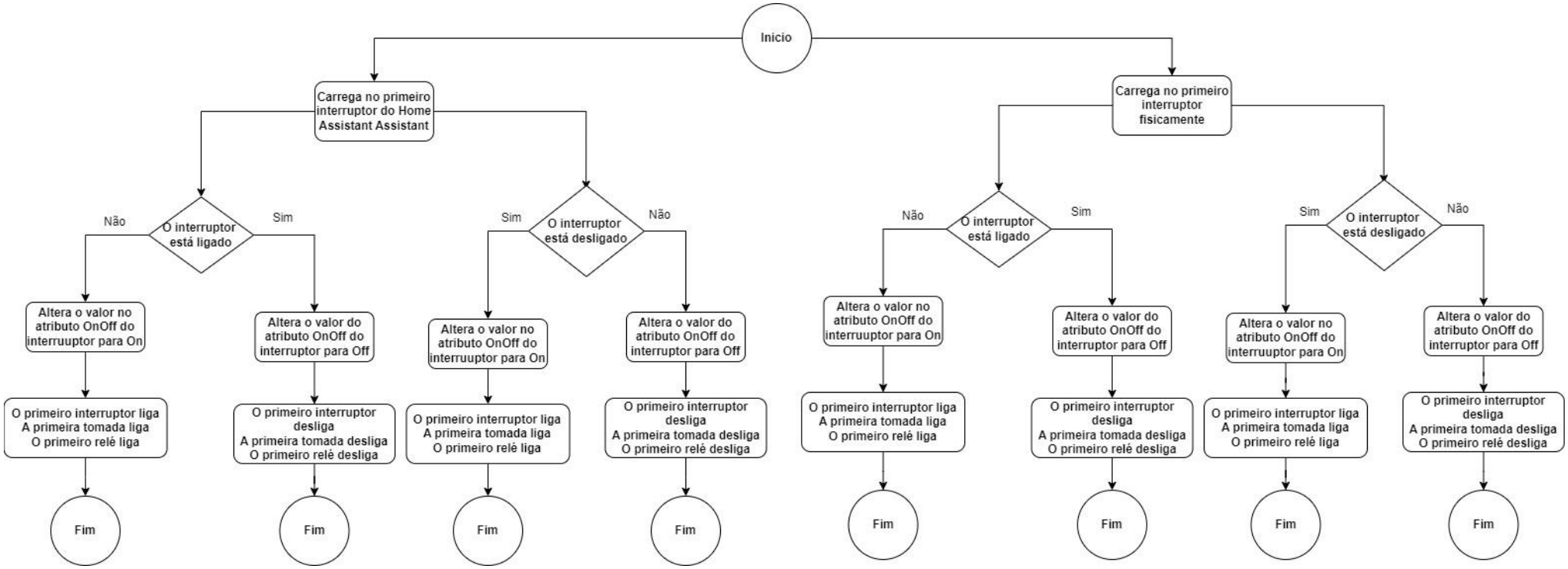

Figura 69 - Fluxograma do controlo do primeiro interruptor em *Matter over Wi-F*

4.4. Cenário de duas tomadas e dois interruptores em *Matter over Thread*

Este cenário é o cenário final, o comportamento do controlo (ligar/desligar) das tomadas e dos interruptores é igual ao cenário anterior, o que altera é a integração e o controlo dos dispositivos ser em *Matter over Thread*, em vez de *Matter over Wi-Fi*.

4.4.1. Trabalho desenvolvido

Para o desenvolvimento deste cenário alterámos o *Menuconfig* para utilizar *Thread* em vez de *Wi-Fi*, e utilizámos a rede *Thread* criada no cenário de integração e controlo de LED por *Matter over Thread*. Assim que alterámos o *Menuconfig*, fizemos *upload* do código desenvolvido no cenário anterior para o *ESP32-C6* e integrámos as duas tomadas e os dois interruptores no *Home Assistant* por *Matter over Thread* sem qualquer erro.

Pela integração e o controlo do dispositivo por Thread ter sido tão bem-sucedido, e por termos de estar sempre a inserir o comando "*matter device factoryreset*" para reiniciar o *ESP32-C6* cada vez que queríamos integrá-lo no *Home Assistant*, decidimos fazer uma melhoria no cenário, e adicionámos um botão para quando se clica no botão, o *ESP32-C6* reiniciar. Em termos de conexões, ligamos o botão ao *gpio pin 21*, e implementámos uma função demonstrada na Figura 70 - Função para reiniciar o *ESP32-C6* que é chamada cada vez que o botão é carregado.

```
static void button_factory_reset_pressed_cb(void *arg, void *data)
{
    ESP_LOGI(TAG, "Starting factory reset");
    chip::DeviceLayer::ConfigurationMgr().InitiateFactoryReset();
}
```

Figura 70 - Função para reiniciar o *ESP32-C6*

Fizemos *upload* de novo do código, antes de iniciarmos o processo de integração, carregámos no botão, e o reinício do *ESP32-C6* foi feito com sucesso. Após reiniciarmos o *ESP32-C6*, integrámos as tomadas e os interruptores no *Home Assistant*, e controlámos os mesmos tanto fisicamente, como através da interface gráfica do *Home Assistant* da maneira pretendida (cada interruptor a controlar cada tomada).

Para tornar o nosso cenário mais prático, decidimos desassociar o primeiro interruptor da primeira tomada de forma a trabalharem de forma independentes. Decidimos fazer isto

para tornar o cenário mais versátil, e para permitir a criação de automações independentes para a tomada ou para o interruptor no *Home Assistant*. Novamente, tivemos de alterar o código para obter este comportamento por parte do controlo do cenário.

Desassociámos o interruptor da esquerda do *node1*, criámos outro *node* designado por *node3* da mesma forma que criámos os nodes anteriormente. Associámos o *endpoint* relativo ao interruptor da esquerda ao *node3* (a alteração deste código está demonstrada na Figura 71 - Alteração do *endpoint* do interruptor da direita).

```
// Add button 1 endpoint
endpoint_t *test_product = extended_color_light::create(node3, &button1_config, ENDPOINT_FLAG_NONE, button1_handle);
ABORT_APP_ON_FAILURE(test_product != nullptr, ESP_LOGE(TAG, "Failed to create button 1 endpoint"));

button1_endpoint = endpoint::get_id(test_product);
ESP_LOGI(TAG, "Light created with endpoint_id %d", button1_endpoint);
```

Figura 71 - Alteração do *endpoint* do interruptor da direita

Assim, terminámos com o *node1* associado à tomada da esquerda, o *node2* associado ao interruptor e à tomada da direita (a funcionarem como anteriormente), e o node3 associado ao interruptor da esquerda.

Por termos alterado a forma que os nodes, os *endpoints* e os clusters estavam associados, alterámos a estrutura do *Matter* representada no cenário anterior. A estrutura está representada na Figura 72 - Estrutura do *Matter* no cenário de duas tomadas e dois interruptores em *Matter over Thread*.

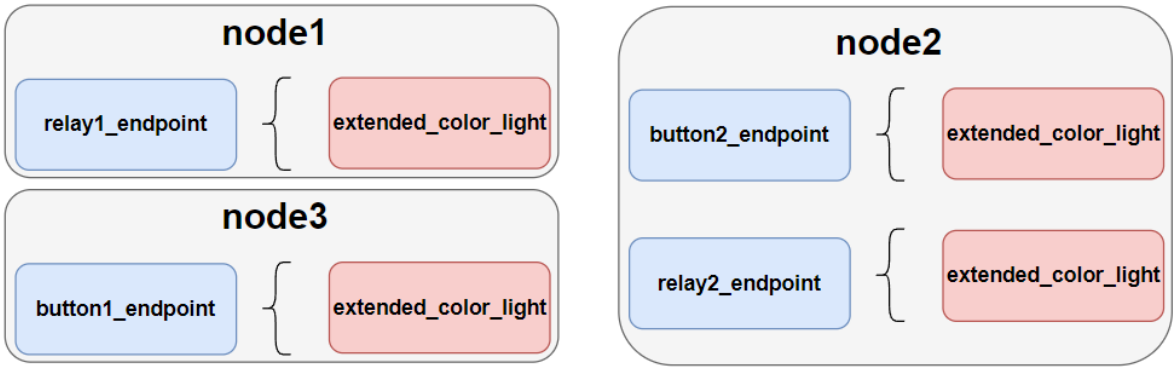

Figura 72 - Estrutura do *Matter* no cenário de duas tomadas e dois interruptores em *Matter over Thread*

A diferença desta estrutura para a anterior é que criámos o *node3* ao qual associámos o interruptor da esquerda (*button1_endpoint*), o *node1* ficou associado à tomada da esquerda (*relay1_endpoint*), e o *node2* manteve-se como estava anteriormente.

Por fim, para o estado do segundo relé ficar sincronizado com o segundo interruptor, mas o primeiro interruptor ficar separado do primeiro relé, alterámos as funções do ficheiro "*app_driver.cpp*", nomeadamente a função "*app_driver_button_button1_cb*" retirando a linha em que associava o relé a controlar com o primeiro botão para o relé não se ligar quando o botão for premido fisicamente, e a função "*app_driver_attribute_update*" de modo a quando o atributo do primeiro botão for alterado no *Home Assistant*, o primeiro relé não ligue. Esta função pode ser observadas Figura 73 - Função utilizada para atualizar o estado dos botões e relés.

```
esp_err_t app_driver_attribute_update(app_driver_handle_t driver_handle, uint16_t endpoint_id, uint32_t cluster_id,
                                      uint32_t attribute_id, esp_matter_attr_val_t *val)
{
    esp_err_t err = ESP_OK;
    led_indicator_handle_t handle = (led_indicator_handle_t)driver_handle;


    // Check if the endpoint matches any relay or button endpoint
    if (endpoint_id == relay1_endpoint || endpoint_id == relay2_endpoint ||
        endpoint_id == button1_endpoint || endpoint_id == button2_endpoint) {

        if (cluster_id == OnOff::Id && attribute_id == OnOff::Attributes::OnOff::Id) {
            if (endpoint_id == relay1_endpoint) {
                relay_to_control = 1;
                err = app_driver_relay_set_power(handle, val);
            } else if (endpoint_id == relay2_endpoint) {
                relay_to_control = 2;
                app_driver_button_button2_cb_update(NULL,NULL);
                err = app_driver_relay_set_power(handle, val);
            }
            else if( endpoint_id == button2_endpoint) {
                relay_to_control = 2;
               app_driver_button_relay2_cb_update(NULL,NULL);
               err = app_driver_relay_set_power(handle, val);
            }


        }
    }

    return err;
}
```

Figura 73 - Função utilizada para atualizar o estado dos botões e relés

Esta alteração foi feita para o utilizador poder utilizar o interruptor e a tomada da esquerda para diversos fins, conforme a sua necessidade, em vez de ser apenas para se ligarem e desligarem mutuamente.

No nosso caso, decidimos criar quatro automações no *Home Assistant*. A primeira para quando o interruptor for ligado/desligado a tomada também liga/desliga, e a segunda para fazer o inverso, ou seja, quando a tomada for ligada/desligada o interruptor também é liga/desliga. Este comportamento, é o mesmo comportamento utilizado no cenário anterior,

mas agora, a qualquer momento é possível alterar as automações e criar outras conforme a necessidade do utilizador.

Para criarmos as automações no *Home Assistant* acedemos à página de "*Settings*", clicámos em "*Automations & Scenes*", clicámos no botão "*CREATE AUTOMATION*", e clicámos em "*Create Automation*". Definimos que quando o interruptor é ligado/desligado, através de clicarmos em "*ADD TRIGGER*", "*DEVICE*" e selecionar o interruptor da esquerda, que a tomada é ligada/desligada, através de "*ADD ACTION*", "*DEVICE*" e selecionar a tomada da esquerda. Para o interruptor da esquerda se ligar/desligar quando ligamos/desligamos a tomada da esquerda fizemos o inverso do exemplo anterior, ou seja, para o gatilho selecionámos a tomada e para a ação selecionámos o interruptor. A Figura 74 - Exemplo de Automação no *Home Assistant* representa a automação desenvolvida para desligar a tomada quando o interruptor é desligado (como explicado anteriormente, para as outras automações fizemos o inverso).

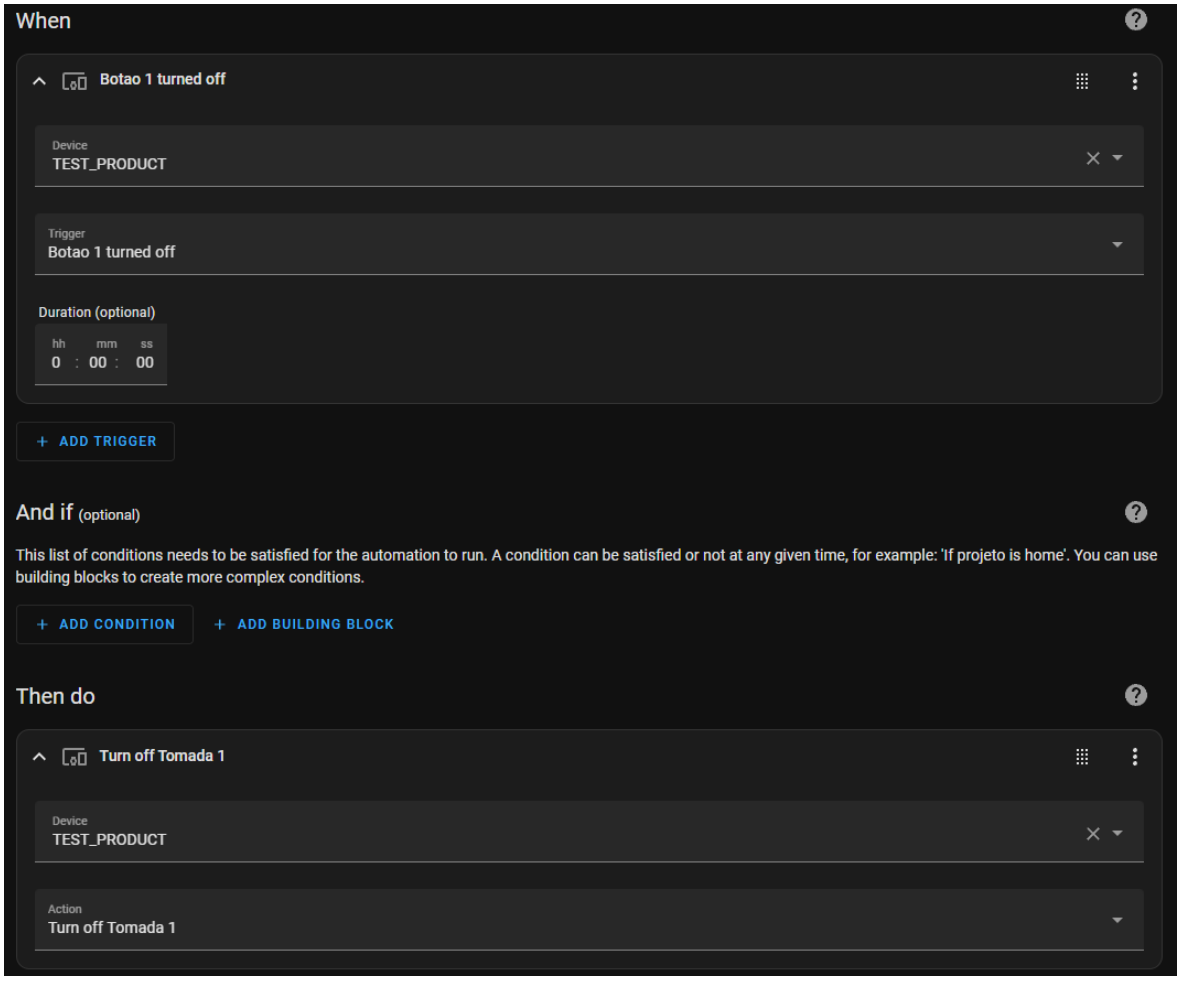

Figura 74 - Exemplo de Automação no *Home Assistant*

As automatizações ficaram criadas como demonstrado na Figura 75 - Automações desenvolvidas.

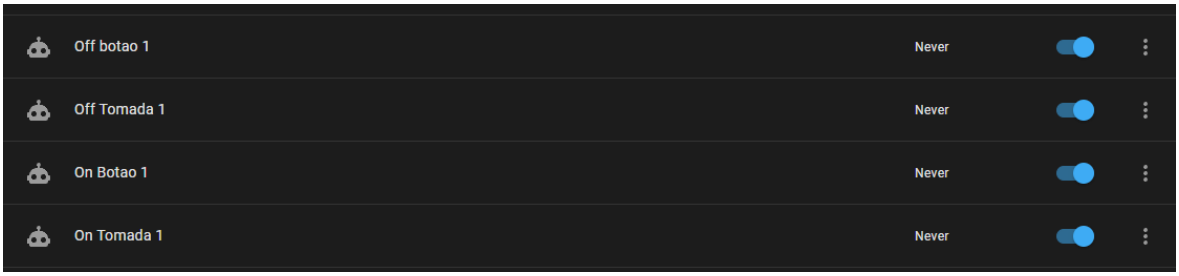

Figura 75 - Automações desenvolvidas

Com as automações feitas, testámos as mesmas, e ao ligar o interruptor da esquerda, o primeiro relé também liga, como se pode observar na Figura 76 - Resultado obtido ao ligarmos o Interruptor 1 o relé 1 após termos criado as automações no *Home Assistant*.

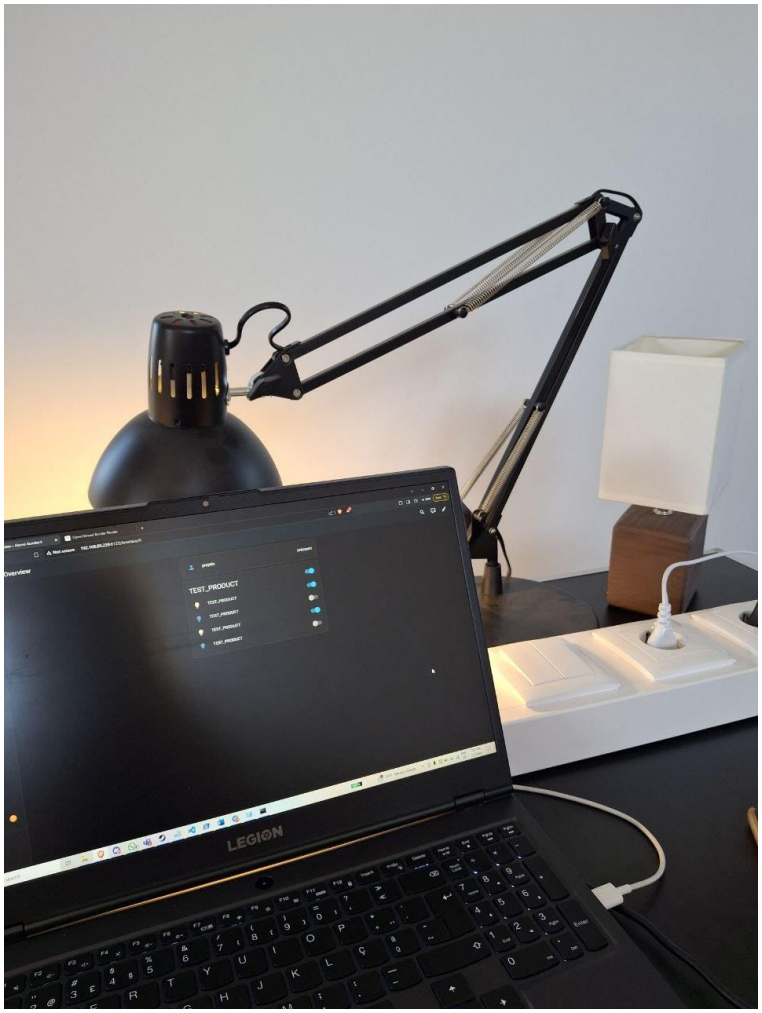

Figura 76 - Resultado obtido ao ligarmos o Interruptor 1 o relé 1 após termos criado as automações no *Home Assistant*

4.4.2. Esquema do protótipo

O esquema do protótipo deste cenário é muito idêntico ao do cenário anterior (apenas adicionámos o botão). Na Figura 77 - Representação das ligações entre os componentes do cenário com o botão demonstramos ao detalhe todas as ligações feitas para este cenário.

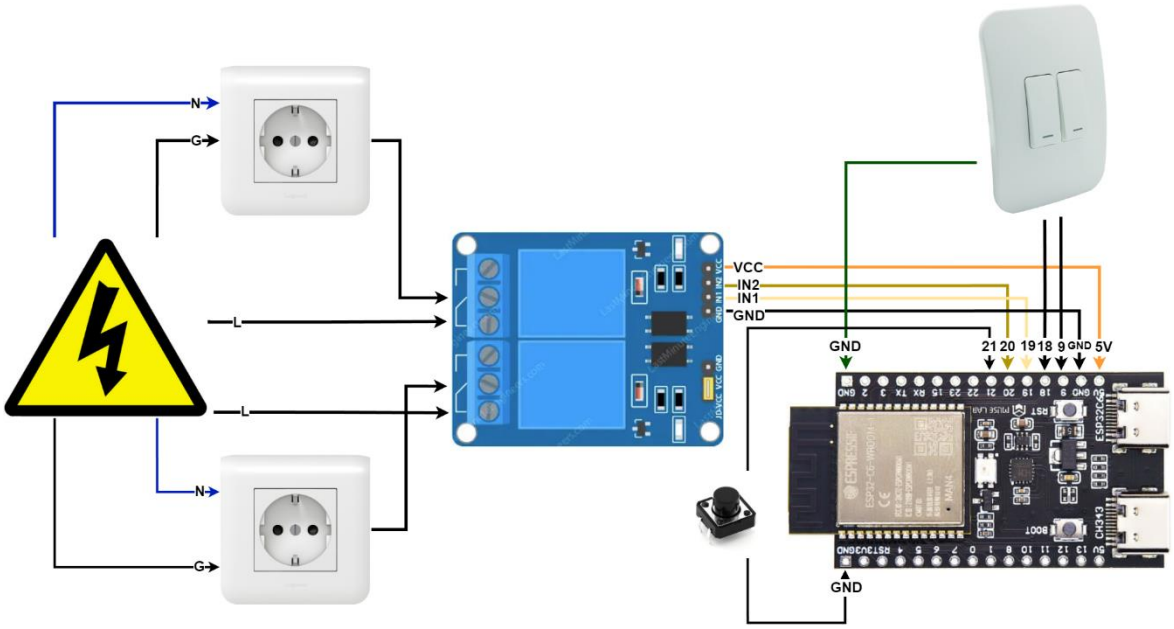

Figura 77 - Representação das ligações entre os componentes do cenário com o botão

Na Figura 78 - Arquitetura da solução de dois interruptores e duas tomadas em *Matter over Thread* representamos a arquitetura da solução com todos os componentes utilizados.

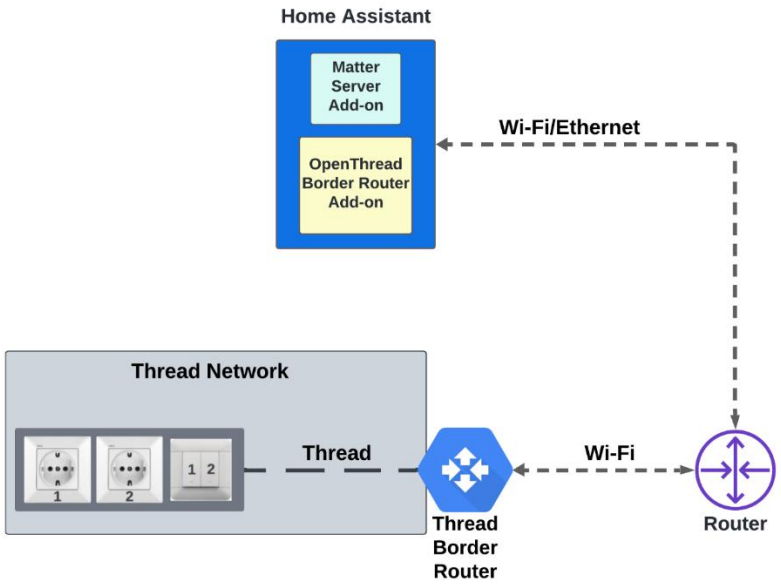

Figura 78 - Arquitetura da solução de dois interruptores e duas tomadas em *Matter over Thread*

4.4.3. Fluxogramas do cenário controlo de duas tomadas e dois interruptores por *Matter over Thread*

O Figura 79 - Fluxograma do controlo do primeiro interruptor sem automações em *Matter over Thread* exemplifica o comportamento do primeiro tanto fisicamente como através do *Home Assistant* (o comportamento da primeira tomada é igual ao do primeiro interruptor, mas o que liga e desliga é a primeira tomada) no caso das automações estarem desligadas.

No caso das automações estarem ligadas, o comportamento dos interruptores e das tomadas é igual ao cenário de duas tomadas e dois interruptores em *Matter over Wi-Fi*.

Os estados de todos os equipamentos alteram conforme o estado do atributo *OnOff* do protocolo *Matter*.

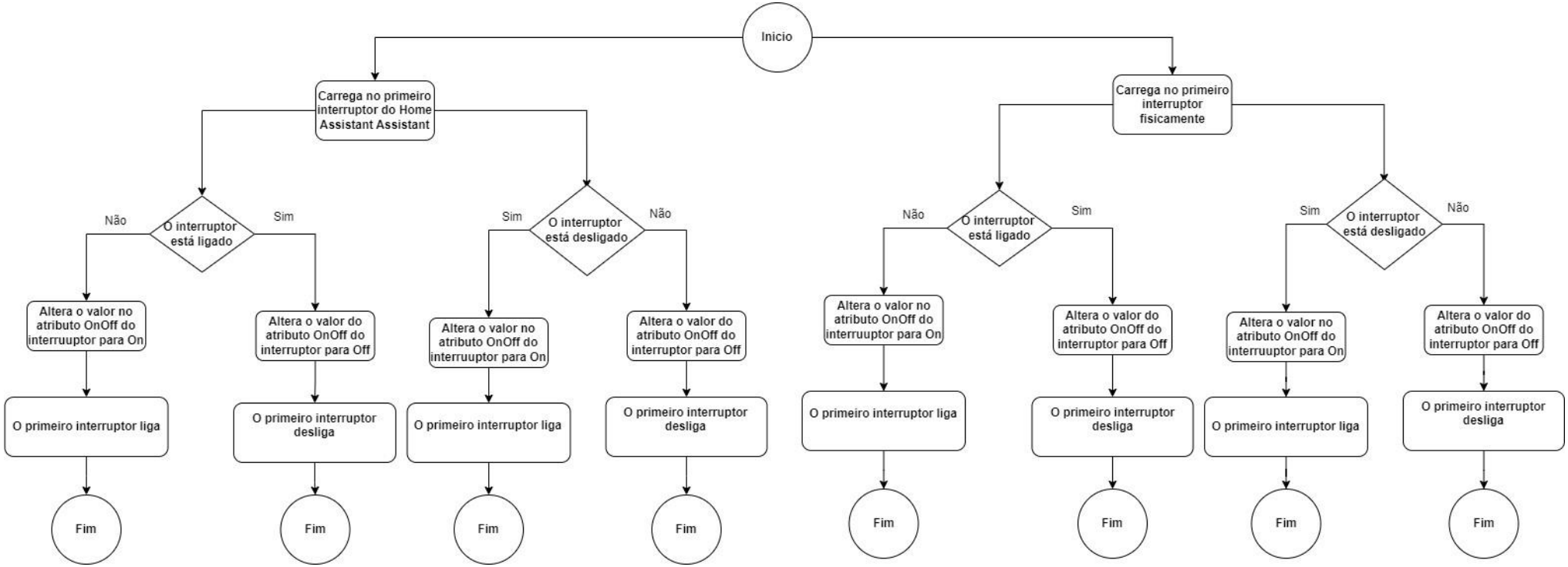

Figura 79 - Fluxograma do controlo do primeiro interruptor sem automações em *Matter over Thread*

4.5.Síntese

Neste capítulo utilizámos a solução proposta como guia e desenvolvemos quatro cenários de forma incremental até obtermos uma solução final em *Matter over Thread* que pode ser utilizada ambientes de casas inteligentes.

Começamos por desenvolver um cenário de integração e controlo de um LED em *Matter over Wi-Fi* para nos integrarmos com o protocolo *Matter* e com o *Home Assistant*. De seguida, replicámos o mesmo cenário, mas em *Matter over Thread* para percebermos o funcionamento do estabelecimento da conexão das redes *Thread*.

Posteriormente, para evoluirmos este cenário para um ambiente mais prático, adicionámos um dispositivo composto por duas tomadas, dois dispositivos e dois relés. Basicamente, o objetivo foi associar os dois interruptores às duas tomadas (os relés fornecem energia às tomadas quando estão ligados) para conseguimos controlar as tomadas através dos interruptores, e após isto integrá-los e controlá-los através do *Home Assistant* por *Matter over Wi-Fi*.

Por fim, decidimos desassociar uma tomada e um interruptor para que pudessem funcionar de forma independente, e adicionámos duas automações a esses componentes do dispositivo. Com esse ajuste, integrámos e controlámos o dispositivo através do *Home Assistant* por *Matter over Thread*.

No próximo capítulo, com base nos cenários desenvolvidos iremos realizar testes e discutir todos os resultados obtidos.

5. Testes realizados e resultados obtidos

Neste capítulo abordamos todos os testes realizados e os resultados obtidos ao longo do desenvolvimento dos quatro cenários, tendo resultados positivos, mas também resultados negativos ao qual descobrimos a solução. Estas soluções podem ajudar futuramente quem pretenda desenvolver soluções em *Matter over Wi-Fi*, mas especialmente em *Matter over Thread* nas suas casas inteligentes.

5.1. Upload do exemplo *blink* no *ESP32-C6*

Para testar o funcionamento do *ESP32-C6* efetuamos o *upload* do exemplo "*blink*" da extensão *esp-idf*. Após o *upload*, o LED embutido no *ESP32-C6* começou a piscar, indicando que o *upload* do código foi efetuado com sucesso, e que podíamos continuar para o próximo passo do projeto.

5.2. Integração e controlo de LED no *Home Assistant* através de *Matter over Wi-Fi*

Os testes neste cenário baseiam-se no *upload* do código para o *ESP32-C6*, no funcionamento do Home Assistant, na integração do LED do *ESP32-C6 no Home Assistant*, e em controlar o LED através do *Home Assistant* por *Matter over Wi-Fi*.

Iniciámos por fazer *upload* do exemplo "*light*" da biblioteca "*esp-matter*" no *ESP32-C6*, e após o *upload* ser feito com sucesso o LED do *ESP32-C6* ligou como suposto (definido no código do exemplo).

Para fazer a integração do dispositivo após o *upload* do código ser realizado, necessitávamos de aceder ao *Home Assistant* pelo telemóvel. Então após instalarmos o *Home Assistant* na máquina virtual e definirmos as credenciais de acesso, testámos através da aplicação do telemóvel se era possível aceder ao servidor com as credenciais previamente definidas, e conseguimos com sucesso (aparecem três servidores na Figura 80 - Conexão ao *Home Assistant* pelo telemóvel devido a tentativas anteriores bem-sucedidas).

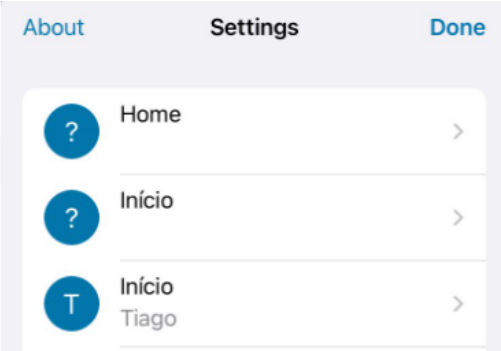

Figura 80 - Conexão ao *Home Assistant* pelo telemóvel

Com o acesso pelo telemóvel ao *Home Assistant*, já foi possível utilizar a extensão *Matter Server* para fazer a integração do LED. Logo, acedemos à extensão, e com a câmara do telemóvel fizemos scan ao *QRCode* criado previamente na consola do *esp-matter*. Conseguimos integrar o LED do *ESP32-C6* com o nome *TEST_PRODUCT* ao *Home Assistant*, como podemos observar na Figura 81 - Integração do ESP32-C6 no *Home Assistant*.

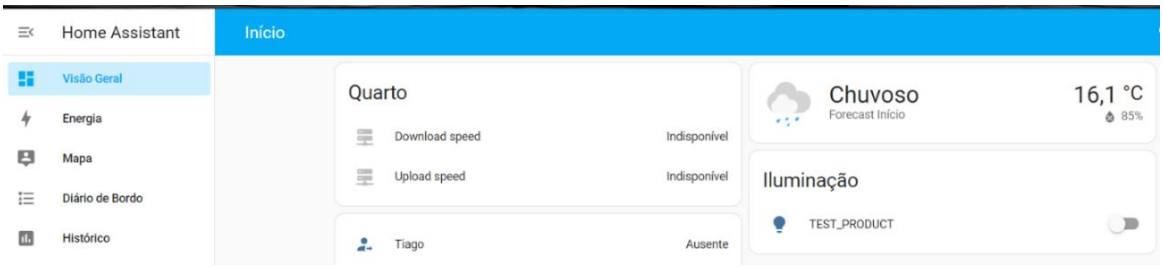

Figura 81 - Integração do ESP32-C6 no *Home Assistant*

Após o LED integrado no *Home Assistant*, testámos se era possível controlá-lo, ligando e desligando o LED através dos botões fornecidos pela interface gráfica do *Home Assistant*, tanto através do telemóvel como da máquina virtual, e conseguimos. Nas Figura 82 - Ligar o LED através do *Home Assistant* e Figura 83 - Desligar o LED através do *Home Assistant* está demonstrado que quando clicamos no botão (conforme o seu estado) o LED liga e desliga por *Matter over Wi-Fi*.

Figura 82 - Ligar o LED através do *Home Assistant*

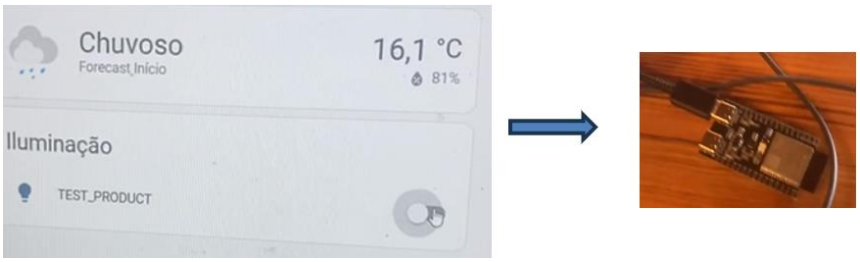

Figura 83 - Desligar o LED através do *Home Assistant*

5.3. Integração e controlo de led no *Home Assistant* através de *Matter over Thread*

Com o *Matter over Wi-Fi* funcional, o próximo teste foi integrar e controlar o LED no *Home Assistant* por *Matter over Thread*. Ao longo deste processo, tivemos bastantes obstáculos, então os testes deste subtópico baseiam-se nestes problemas e nas soluções que utilizámos para os resolver.

Utilizámos o exemplo "*light*", alterámos as opções do *Menuconfig* para o *ESP32-C6* utilizar *Thread* em vez de *Wi-Fi*, e fizemos *upload* do exemplo "*light*" para o *ESP32-C6* (com suporte para *Thread* em vez de *Wi-Fi*) com sucesso (o LED ligou após o *upload* terminar).

Para a nossa solução utilizámos o adaptador *sonoff zigbee 3.0 USB dongle* como *Thread Brder Router*, então para suportar *Thread* tentámos fazer upload do *firmware Multi-PAN* para o equipamento, e após o *upload* ser feito, obtivemos uma mensagem com a informação "*Installation Sucess*".

Com o adaptador *sonoff USB* a suportar *Thread*, já podia atuar como *Thread Border Router*, logo o próximo teste foi associar o adaptador *sonoff USB* à extensão (*Silicon Labs Multiprotocol*), mas ao tentarmos adicionar o adaptador *sonoff USB* à extensão, encontrámos o nosso primeiro desafio. Descobrimos, que devido a estarmos a utilizar uma máquina virtual, o adaptador *sonoff USB* não era detetado pelo *Home Assistant*, então tivemos de aceder às opções *USB* da máquina virtual, adicionar a porta *USB* ao qual o adaptador *sonoff USB* estava conectado, e após isto já conseguimos associar o adaptador *sonoff USB* à extensão.

O próximo teste foi tentar criar a rede *Thread*, que foi bem-sucedido logo após ligarmos a extensão (*Silicon Labs Multiprotocol*), como podemos observar na Figura 31 - Informação relativa à rede *Thread*.

Com a rede *Thread* criada, com o *upload* do código feito no *ESP32-C6*, e o *QRCode* criado, tentámos fazer a integração do LED no *Home Assistant* através da extensão *Matter Server*, mas obtivemos um erro com a mensagem "*Thread Border Router Required*" em iOS e "*Can't connect to thread network*" em Android. Não conseguíamos descobrir o problema, então voltámos a testar o cenário com *Matter over Wi-Fi*, e também já não conseguíamos fazer a integração como conseguíamos anteriormente.

Descobrimos que tínhamos de reiniciar o *ESP32-C6* sempre que o queríamos integrar, após o dispositivo já ter sido integrado anteriormente. Como em *Matter over Wi-Fi*, ao reiniciarmos o *ESP32-C6* conseguimos fazer a integração com sucesso, voltámos a tentar por *Matter over Thread*, mas continuámos a obter o mesmo erro.

Tentámos alterar as configurações do *Menuconfig* do *esp-idf* para descobrirmos se este erro poderia estar relacionado com o *mDNS*, mas não fez diferença. Logo, fizemos *upload* ao adaptador *sonoff USB* com um *firmware* diferente designado por *OpenThread* (suporta apenas *Thread*, enquanto o outro *firmware* suporta *Thread* e *Zigbee*), removemos a extensão *Silicon Labs Multiprotocol*, e instalámos a extensão *Open Thread Border Router* por ser mais indicado na documentação sobre *Thread*. Novamente a rede *Thread* foi criada automaticamente sem problemas, mas quando testámos fazer a integração do LED obtivemos o mesmo erro ("*Thread Border Router Required*" /" Can't connect to thread network").

Descobrimos que existe uma opção no *Home Assistant* (que difere entre iOS e Android) para importar as credenciais de rede *Thread* para o telemóvel. Após utilizarmos esta opção, ultrapassámos o erro "*Thread Border Router Required*" /" Can't *connect to thread network*", mas obtivemos outro erro: "*Unable to Add Accessory*" /" *Can't find device*". Uma das diferenças que notámos por termos ultrapassado o erro, foi que o adaptador *sonoff USB* já aparecia na topologia de rede como *Router*.

Para tentarmos resolver o erro "*Unable to Add Accessorry*" / "*Can't find device*", analisámos os logs do *Matter Server* e descobrimos erros relacionados com o *mDNS*.

Para tentar resolver, configurámos um *Mikrotik Router* de raiz para que não houvesse problemas com o tráfego do *mDNS*, e instalámos o *Home Assistant* num *mini PC* pelo facto de haver a possibilidade do tráfego *mDNS* estar a ser bloqueado pela máquina virtual mesmo com o adaptador de rede em modo Bridge. Estas duas alterações, permitiram que fosse possível integrar o LED no *Home Assistant* por *Matter over Thread*. Na topologia da rede *Thread*, já aparecia o adaptador *sonoff USB* como Leader e o ESP32-C6 como *Child* (como demonstrado na Figura 84 - ESP32-C6 na topologia de rede *Thread*).

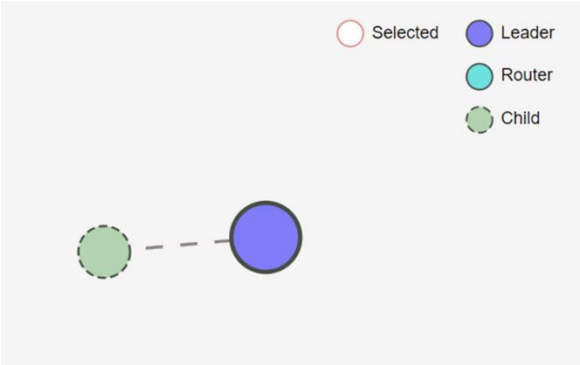

Figura 84 - ESP32-C6 na topologia de rede *Thread*

Com o LED integrado, tentámos ligá-lo e desligá-lo tanto através do telemóvel como do *mini PC*, e conseguimos sem qualquer problema.

5.4. Integração e controlo de duas tomadas e dois interruptores no *Home Assistant* através de *Matter over Wi-Fi*

Para este cenário, os testes não são tão baseados na integração das duas tomadas e dos dois interruptores, mas são mais baseados primeiramente no controlo das tomadas e dos interruptores fisicamente, e secundariamente no controlo dos mesmos através do *Home Assistant* após terem sido integrados (os testes de integração são iguais aos testes feitos nos cenários anteriores).

Começamos por testar se ao clicar no primeiro interruptor o LED do ESP32-C6 ligava e desligava conforme o estado do atributo *OnOff*. Este teste foi feito com sucesso, então adicionámos o segundo interruptor ao cenário, para ligarmos e desligarmos o LED independentemente do interruptor clicado.

Com os interruptores funcionais, o próximo teste foi primeiramente controlar o primeiro relé (que atribuía energia à primeira e ligava-a) independentemente do interruptor clicado, e secundariamente, com o primeiro relé a funcionar, adicionámos o segundo relé (que atribuía

energia à segunda tomada e ligava-a), de modo que ambos os relés ligassem/desligassem conforme o seu estado quando se clicava num dos interruptores.

O teste final antes de integrarmos os dispositivos no *Home Assistant*, foi associar o primeiro interruptor ao primeiro relé, e o segundo interruptor ao segundo relé, para os relés serem apenas controlados pelo interruptor associado, que por sua vez apenas ligava a tomada associada ao relé a ser controlado.

Estes testes foram todos feito com êxito, então integrámos as tomadas e os interruptores ao *Home Assistant* sem qualquer erro, mas após a integração ser feita, apenas um relé foi inicializado, os interruptores físicos apenas controlavam o relé inicializado, e clicarmos nos botões da interface gráfica do *Home Assistant* não alterava o estado nem dos interruptores nem dos relés.

Descobrimos que este problema derivava de termos os dispositivos todos associados a um *node* e a um *endpoint*, logo, criámos dois *nodes*, e quatro *endpoints*. Aos *endpoints* associámos cada interruptor e cada tomada, e aos nodes, associámos os *endpoints* com o primeiro interruptor e a primeira tomada a um dos nodes, e associámos os endpoints com o segundo interruptor e a segunda tomada ao outro node.

Com estas alterações ao código, o controlo das tomadas e dos interruptores começou a ocorrer de forma igual tanto fisicamente, como através dos botões da interface gráfica do *Home Assistant*.

Na Tabela 10 - Tabela de testes realizados e resultados obtidos para o cenário de duas tomadas e dois interruptores em *Matter over Wi-Fi* podemos observar todos os testes que fizemos e os resultados obtidos em relação aos interruptores, aos relés e às tomadas (as tomadas têm sempre o mesmo comportamento que os relés por estarem associados). Utilizámos o "- "quando o estado desse dispositivo não é alterado, *On* quando o dispositivo liga e o *Off* quando o dispositivo desliga (o atributo a ser alterado é o *OnOff* e apenas suporta os valores *On* e *Off* daí termos utilizado esses valores).

Tabela 10 - Tabela de testes realizados e resultados obtidos para o cenário de duas tomadas e dois interruptores em *Matter over Wi-Fi*

Teste	1º Interruptor	2º Interruptor	1º Tomada	2º Tomada	1º Relé	2º Relé	1º Interruptor *Home Assistant*	2º Interruptor *Home Assistant*	1º Tomada *Home Assistant*	2º Tomada *Home Assistant*
Ligar 1º interruptor fisicamente	*On*	-	*On*	-	*On*	-	*On*	-	*On*	-
Ligar 2º interruptor fisicamente	-	*On*	-	*On*	-	*On*	-	*On*	-	*On*
Desligar 1º interruptor fisicamente	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-
Desligar 2º interruptor fisicamente	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*
Ligar 1º interruptor no *Home Assistant*	*On*	-	*On*	-	*On*	-	*On*	-	*On*	-
Ligar 2º interruptor no *Home Assistant*	-	*On*	-	*On*	-	*On*	-	*On*	-	*On*
Desligar 1º interruptor no *Home Assistant*	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-
Desligar 1º interruptor no *Home Assistant*	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*
Ligar 1º tomada no Home Assistant	*On*	-	*On*	-	*On*	-	*On*	-	*On*	-
Ligar 2º tomada no	-	*On*	-	*On*	-	*On*	-	*On*	-	*On*

Home Assistant										
Desligar 1º tomada no Home Assistant	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-
Desligar 2º tomada no Home Assistant	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*

5.5. Integração e controlo de duas tomadas e dois interruptores no *Home Assistant* através de *Matter over Thread*

Os testes que fizemos neste cenário foram muito idênticos aos testes e aos resultados obtidos no cenário anterior. As diferenças é que a integração e o controlo foram feitos por *Matter over Thread*, tivemos de testar um botão que adicionámos para reiniciar o *ESP32-C6* antes de fazer integração dos dispositivos no *Home Assistant*, e desassociámos o primeiro interruptor da primeira tomada para trabalharem de forma independente.

Para testarmos o botão, antes de iniciarmos o processo de integração das tomadas e dos interruptores, clicámos no botão, vimos na consola do esp-matter que o comando "matter device factoryreset" foi executado sem problemas, tal como definido na função. Para termos a certeza que o EPS32-C6 reiniciou, tentámos integrar as tomadas e os interruptores no Home Assistant, e o processo foi bem-sucedido, indicando que o *ESP32-C6* reiniciou.

Com a integração feita com sucesso, testámos o controlo das tomadas e dos interruptores e o comportamento dos mesmo foi como o desejado (igual ao cenário anterior). Os testes de controlo estão explicados na Tabela 10 - Tabela de testes realizados e resultados obtidos para o cenário de duas tomadas e dois interruptores em *Matter over Wi-Fi*.

Com estes testes bem-sucedidos, decidimos tornar o primeiro interruptor e a primeira tomada independentes para tornar o cenário mais versátil. Após alterarmos o código, integrámos de novo as tomadas e os interruptores, e testámos o funcionamento do cenário. Os resultados obtidos são representados na Tabela 11 - Tabela de testes realizados e resultados obtidos para o cenário de duas tomadas e dois interruptores em *Matter over Thread*

.

Tabela 11 - Tabela de testes realizados e resultados obtidos para o cenário de duas tomadas e dois interruptores em *Matter over Thread*

Teste	1º Interruptor	2º Interruptor	1º Tomada	2º Tomada	1º Relé	2º Relé	1º Interruptor *Home Assistant*	2º Interruptor *Home Assistant*	1º Tomada *Home Assistant*	2º Tomada *Home Assistant*
Ligar 1º interruptor fisicamente	*On*	-	-	-	-	-	*On*	-	-	-
Ligar 2º interruptor fisicamente	-	*On*	-	*On*	-	*On*	-	*On*	-	*On*
Desligar 1º interruptor fisicamente	*Off*	-	-	-	-	-	*Off*	-	-	-
Desligar 2º interruptor fisicamente	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*
Ligar 1º interruptor no *Home Assistant*	*On*	-	-	-	-	-	*On*	-	-	-
Ligar 2º interruptor no *Home Assistant*	-	*On*	-	*On*	-	*On*	-	*On*	-	*On*
Desligar 1º interruptor no *Home Assistant*	*Off*	-	-	-	-	-	*Off*	-	-	-
Desligar 2º interruptor no *Home Assistant*	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*
Ligar 1º tomada no Home Assistant	-	-	-	*On*	-	*On*	-	-	-	*On*
Ligar 2º tomada no	-	*On*	-	*On*	-	*On*	-	*On*	-	*On*

Home Assistant										
Desligar 1º tomada no Home Assistant	-	-	-	*Off*	-	*Off*	-	-	*Off*	-
Desligar 2º tomada no Home Assistant	-	*Off*	-	*Off*	-	*Off*	-	*Off*	-	*Off*

Por termos o primeiro interruptor e a primeira tomada independentes, decidimos adicionar duas automações, uma para o interruptor e outra para a tomada. A primeira automação faz com que a primeira tomada ligue/desligue quando se clica no primeiro interruptor (tanto no botão físico como na interface gráfica), e a segunda automação faz o inverso, ou seja, quando se clica na primeira tomada (apenas interface gráfica) o primeiro interruptor ligasse/desligasse, conforme o seu estado.

5.6. Testes Funcionais

Esta secção, consiste nos testes funcionais, ou seja, quais são os problemas e quais são as falhas de comunicação que podem ocorrer no caso dos dispositivos utilizados no cenário de duas tomadas e dois interruptores em *Matter over Thread* falharem. Mais especificamente, os testes são focados no que acontece no caso de removermos o *Mikrotik Router*, no caso de removermos o adaptador *Sonoff USB Dongle*, no caso de desligarmos o *Home Assistant* e no caso de desligarmos o *ESP32-C6*.

Na Tabela 12 - Tabela dos testes funcionais e os resultados obtidos para o cenário de duas tomadas e dois interruptores em *Matter over Thread* demonstramos todos os testes funcionais feitos e todos os resultados obtidos para os testes executados, que são muito úteis para saber o que acontece na perca de conexão das componentes fulcrais do cenário.

Tabela 12 - Tabela dos testes funcionais e os resultados obtidos para o cenário de duas tomadas e dois interruptores em *Matter over Thread*

Recurso	*ESP32-C6*	*Mikrotik Router*	*Adaptador Sonoff USB Dongle*	*Home Assistant*	1º interruptor e 1º tomada	2º interruptor e 2º tomada
Desligar o *ESP32-C6*	Desliga	Perde conexão com o *ESP32-C6*, mas continua a ter conexão com o *Home Assistant* e o adaptador *Sonoff USB Dongle*	Perde conexão com o *ESP32-C6*	Perde conexão com o *ESP32-C6*	Perde conexão	Perde conexão
Desligar o *Mikrotik Router*	O *ESP32-C6* fica funcional como anteriormente	Desliga	Perde conexão com o *Mikrotik Router*, mas a rede *Thread* fica funcional (apenas não consegue comunicar para fora da rede)	Os estados dos dispositivos integrados no *Home Assistant* deixam de funcionar	O seu estado no *Home Assistant* deixa de atualizar e não é possível controlar fisicamente ou através do *Home Assistant*	Continuam a funcionar fisicamente (por estarem associadas por código), mas o seu estado no *Home Assistant* não atualiza
Desligar o adaptador *Sonoff USB Dongle*	O *ESP32-C6* fica funcional como anteriormente	Perde conexão com o adaptador *Sonoff USB Dongle*, mas a rede *Wi-Fi*	Desliga	Os estados dos dispositivos integrados no *Home Assistant* deixam de	O seu estado no *Home Assistant* deixa de atualizar e não é possível controlar fisicamente ou	Continuam a funcionar fisicamente (por estarem associadas por código), mas o seu estado no

		fica funcional		funcionar e perde acesso ao adaptador *Sonoff USB Dongle* na extensão *OTBR*	através do *Home Assistant*	*Home Assistant* não atualiza
Desligar o *Home Assistant*	O *ESP32-C6* fica funcional como anteriormente	Perde conexão com o *Home Assistant* e o *Thread Border Router*	Perde conexão	Desliga	O seu estado no *Home Assistant* deixa de atualizar e não é possível controlar fisicamente ou através do *Home Assistant*	Ligam e desligam fisicamente de forma conjunta (por estarem associadas por código)

5.7.Síntese

Neste capítulo utilizámos os quatros cenários desenvolvidos no capítulo anterior para realizar testes e discutir os resultados obtidos.

Concluímos que desenvolver cenários em *Matter over Wi-Fi* e *Matter over Thread* é muito desafiante, mas que com toda a pesquisa realizada e testes feitos ao longo do desenvolvimento que é possível utilizar estes protocolos em ambientes de casas inteligentes.

Em relação ao *Matter over Thread* é necessário ter em atenção os seguintes tópicos: reiniciar sempre o dispositivo que irá ser integrado antes do processo de integração ser inicializado, utilizar Wi-Fi/Ethernet que não tenha regras de bloqueio de tráfego, utilizar o *Home Assistant* em ambientes não virtualizados para não haver o *mDNS* ser transmitido facilmente, ter em atenção se existe mais do que uma rede *Thread* no telemóvel, importar sempre as credenciais *Thread* para o telemóvel, e por fim utilizar a extensão *OTBR* em vez do *Silicon Labs Multiprotocol*.

No próximo capítulo, iremos descrever as conclusões obtidas ao longo do desenvolvimento de todo o do projeto.

6. Conclusão

A pesquisa e o desenvolvimento deste projeto permitiram-nos entender que ambos os protocolos, *Matter* e *Thread*, ainda são muito recentes, há escassez de documentação e os exemplos práticos disponíveis ainda são muito vagos. No entanto, concluímos que já é possível desenvolver cenários para casas inteligentes em *Matter over Thread*.

Desenvolvemos o projeto de forma incremental, e para facilitar a implementação, não utilizámos apenas *Matter over Thread*, mas também utilizámos *Matter over Wi-Fi*. Constatáámos que o desenvolvimento de *Matter over Wi-Fi* é mais simples, pois não requer o passo adicional de criar e gerir uma rede *Thread*. , eContudo, a abordagem de *Matter over Wi-Fi*, não é uma solução ideal, pois não resolve o problema do elevado consumo energético associado ao protocolo *Wi-Fi* tal como mencionado anteriormente.

Em relação à abordagem *Matter over Thread*, concluímos que há vários fatores críticos para a integração e controlo de dispositivos *IoT* em casas inteligentes. Descobrimos e apresentámos soluções de como ultrapassar essas dificuldades, tal como utilizar um *Wi-Fi Access Point* que não tenha regras de bloqueio de tráfego (especialmente *vlans*), utilizar o *Home Assistant* em abientes não virtualizado, importar sempre as credenciais da rede *Thread* para o telemóvel, reiniciar sempre os dispositivos antes de os integrar, e eliminar as redes *Thread* do telemóvel no caso de existir mais que uma.

Identificámos um problema critico quando há mais que uma rede *Thread* no telemóvel, e a solução não é estável pois apenas foi possível remover as redes em alguns dispositivos Android e não foi possível em dispositivos iOS, pelo que concluímos que ainda é uma funcionalidade que está em desenvolvimento. A única solução que descobrimos para contornar este problema em iOS foi utilizar apenas uma rede *Thread* associada ao telemóvel, para não haver interferência entre as várias redes, e em Android foi limpar os serviços do *Google Play*. Quando este procedimento não funcionou em Android, a solução foi também só ter uma rede *Thread*.

A realização deste projeto proporcionou-nos um entendimento aprofundado sobre os protocolos *Matter* e *Thread*, que têm um grande potencial de utilização em ambientes de casas inteligentes, devido às suas características únicas. Chegámos à conclusão que o melhor protocolo para utilizar em soluções de casas inteligentes é o *Thread* por ter um baixo

consumo energético, não ter pontos únicos de falha, por utilizar um protocolo de encriptação seguro, por ter o alcance suficiente para ambientes de casas inteligentes, e por ter uma taxa de transmissão de dados consideravelmente boa (apenas pode ser problemático se for integrado com câmaras de vigilância).

Pesquisámos e listámos todos os dispositivos compatíveis com *Matter* e *Thread*, o *Thread Border Router* mais ideal para utilizar, e todo o software necessário e indicado nestes ambientes. Conseguimos agora explicar e exemplificar como é possível utilizar *Matter over Wi-Fi*, e especialmente *Matter over Thread* em casas inteligentes, promovendo a interoperabilidade e a eficiência energética devido ao baixo consumo do protocolo *Thread*.

Apresentámos um protótipo funcional, com todos os passos de desenvolvimento e demonstrámos todas as soluções para os desafios enfrentados durante o desenvolvimento da solução. Por fim, solucionámos uma arquitetura que futuros utilizados possam utilizar, com equipamentos alternativos, de baixo custo, programáveis e adaptáveis consoante as necessidades dos utilizadores.

Futuramente, o mais importante seria descobrir uma solução para eliminar as redes *Thread* do telemóvel, pois foi o fator mais crítico que encontrámos e que pode causar muito transtorno no desenvolvimento de *Matter over Thread* em casas inteligentes.

Bibliografia

[1]“Internet of Things | Definition, History, Examples, & Privacy Concerns | Britannica.” Accessed: Jul. 04, 2024. [Online]. Available: https://www.britannica.com/science/Internet-of-Things

[2]“Thread vs WiFi | Difference between Thread and WiFi.” Accessed: Mar. 25, 2024. [Online]. Available: https://www.rfwireless-world.com/Terminology/Thread-vs-WiFi.html

[3]“What is Home Assistant, how does it work, and what do you need to get started?” Accessed: Jun. 15, 2024. [Online]. Available: https://www.pocket-lint.com/what-is-home-assistant-how-does-it-work/

[4]“Home Assistant vs Google Home.” Accessed: Mar. 20, 2024. [Online]. Available: https://www.dusuniot.com/blog/home-assistant-vs-google-home-what-are-the-pros-and-cons/

[5]“OpenHAB vs Home Assistant.” Accessed: Mar. 20, 2024. [Online]. Available: https://www.dusuniot.com/blog/openhab-vs-home-assistant-which-one-is-right-for-you/

[6]“Apple HomeKit and Home app: What are they and how do they work?” Accessed: Jun. 15, 2024. [Online]. Available: https://www.pocket-lint.com/apple-homekit-and-home-app-what-are-they-and-how-do-they-work/

[7]“Z-Wave vs Thread: What is the Difference？ – SyncHabitat.com.” Accessed: Mar. 04, 2024. [Online]. Available: https://synchabitat.com/z-wave-vs-thread-what-is-the-difference/

[8]“Bluetooth Vs. Bluetooth Low Energy: What’s The Difference? | Blog.” Accessed: Jun. 09, 2024. [Online]. Available: https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy

[9]“Topology Options | Bluetooth® Technology Website.” Accessed: Jun. 09, 2024. [Online]. Available: https://www.bluetooth.com/learn-about-bluetooth/topology-options/

[10] “Smart home wireless protocols explained (Zigbee, Thread, Matter etc).” Accessed: Mar. 04, 2024. [Online]. Available:

https://www.smartspaces.optus.com.au/blog/zigbee-thread-z-wave-bluetooth-and-matter-explained

[11] A. O. Blasi, “Evaluating Thread protocol in the framework of Matter”.

[12] “What is Mesh Topology? How Does Data Travel in a Mesh Network | Lenovo US.” Accessed: May 26, 2024. [Online]. Available: https://www.lenovo.com/us/en/glossary/mesh-topology/?orgRef=https%253A%252F%252Fwww.google.com%252F

[13] V. Bajrami, “What you need to know about IPv6.” Accessed: Jun. 10, 2024. [Online]. Available: https://www.redhat.com/sysadmin/what-you-need-know-about-ipv6

[14] “Part 1: Smart Home Matter and Thread Deep Dive - Derek Seaman’s Tech Blog.” Accessed: May 26, 2024. [Online]. Available: https://www.derekseaman.com/2023/10/part-1-smart-home-matter-and-thread-deep-dive.html

[15] “What is 6LoWPAN? - GeeksforGeeks.” Accessed: Mar. 09, 2024. [Online]. Available: https://www.geeksforgeeks.org/what-is-6lowpan/

[16] “What is the User Datagram Protocol (UDP)? | Cloudflare.” Accessed: May 26, 2024. [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/

[17] “What is TCP/IP in Networking? | Fortinet.” Accessed: May 26, 2024. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/tcp-ip

[18] “UG103.11: Thread Fundamentals”.

[19] “Certified Products.” Accessed: May 24, 2024. [Online]. Available: https://www.threadgroup.org/Certified-Products

[20] K. Ingram, “Thread In Commercial Backgrounder”.

[21] “A Comprehensive Guide to Smart Home Protocol: Wi-Fi, Bluetooth, Zigbee, Z-Wave, Thread, and Matter - DFRobot.” Accessed: Feb. 29, 2024. [Online]. Available: https://www.dfrobot.com/blog-13453.html

[22] “Roles - Matter.” Accessed: May 26, 2024. [Online]. Available: https://handbook.buildwithmatter.com/howitworks/roles/

[23] “Matter Data Model - latest - Silicon Labs Matter Silicon Labs.” Accessed: Jul. 07, 2024. [Online]. Available: https://docs.silabs.com/matter/latest/matter-fundamentals-data-model/

[24] “The Device Data Model - Matter.” Accessed: May 26, 2024. [Online]. Available: https://handbook.buildwithmatter.com/howitworks/datamodel/

[25] “Discovery - Matter.” Accessed: May 26, 2024. [Online]. Available: https://handbook.buildwithmatter.com/howitworks/discovery/

[26] “Commissioning - Matter.” Accessed: May 26, 2024. [Online]. Available: https://handbook.buildwithmatter.com/howitworks/commisioning/

[27] “Supported Device Types - Matter.” Accessed: May 26, 2024. [Online]. Available: https://handbook.buildwithmatter.com/howitworks/devicetypes/

[28] “Matter Arrives Bringing A More Interoperable, Simple And Secure Internet Of Things to Life - CSA-IOT.” Accessed: Jun. 13, 2024. [Online]. Available: https://csa-iot.org/newsroom/matter-arrives/

[29] “Matter 1.0 Application Clusters”.

[30] “Matter 1.1 release — Enhancements for developers and devices - CSA-IOT.” Accessed: May 28, 2024. [Online]. Available: https://csa-iot.org/newsroom/matter-1-1-release-enhancements-for-developers-and-devices/

[31] “Matter 1.2 Arrives with Nine New Device Types & - CSA-IOT.” Accessed: Mar. 16, 2024. [Online]. Available: https://csa-iot.org/newsroom/matter-1-2-arrives-with-nine-new-device-types-improvements-across-the-board/

[32] “Matter 1.2 Application Clusters”.

[33] “Matter 1.3 Specification Released - CSA-IOT.” Accessed: Jun. 13, 2024. [Online]. Available: https://csa-iot.org/newsroom/matter-1-3-specification-released/

[34] “Matter 1.3 Application Clusters”.

[35] "ESP Thread Border Router, Conselho de Desenvolvimento Zigbee Gateway, Espressif Systems, Série ESP32-H2 - AliExpress 502." Accessed: Jun. 30, 2024. [Online]. Available: https://pt.aliexpress.com/item/1005005872059584.html?src=google&src=google&albch=shopping&acnt=631-313-3945&slnk=&plac=&mtctp=&albbt=Google_7_shopping&gclsrc=aw.ds&albagn=888888&isSmbAutoCall=false&needSmbHouyi=false&src=google&albch=shopping&acnt=631-313-3945&slnk=&plac=&mtctp=&albbt=Google_7_shopping&gclsrc=aw.ds&albagn=888888&ds_e_adid=&ds_e_matchtype=&ds_e_device=c&ds_e_network=x&ds_e_product_group_id=&ds_e_product_id=pt1005005872059584&ds_e_product_merchant_id=552159742&ds_e_product_country=PT&ds_e_product_language=pt&ds_e_product_channel=online&ds_e_product_store_id=&ds_url_v=2&albcp=19821743728&albag=&isSmbAutoCall=false&needSmbHouyi=false&gad_source=1&gclid=CjwKCAjwhIS0BhBqEiwADAUhc1qLOiW0TlMBUQBWd1CPoFF5HfSkPvTDVC91X5BKTdzqy7N5wpgVeRoCobkQAvD_BwE&aff_fcid=ba9530df532544b2aa10393e44afc538-1719772987183-09983-UneMJZVf&aff_fsk=UneMJZVf&aff_platform=aaf&sk=UneMJZVf&aff_trace_key=ba9530df532544b2aa10393e44afc538-1719772987183-09983-UneMJZVf&terminal_id=3d96bfabd3d445d7b7363627e10674d7&afSmartRedirect=n

[36] "GA03030AU_google_nest_wifi_pro_home_mesh_wi_fi_6e_system_white.jpg (1000×1000)." Accessed: Jun. 30, 2024. [Online]. Available: https://s3-ap-southeast-2.amazonaws.com/wc-prod-pim/JPEG_1000x1000/GA03030AU_google_nest_wifi_pro_home_mesh_wi_fi_6e_system_white.jpg

[37] "GL-S200 - GL.iNet." Accessed: Jun. 30, 2024. [Online]. Available: https://www.gl-inet.com/products/gl-s200/

[38] "SONOFF Zigbee 3.0 USB Dongle Plus-P - SONOFF Official." Accessed: May 24, 2024. [Online]. Available: https://sonoff.tech/product/gateway-and-sensors/sonoff-zigbee-3-0-usb-dongle-plus-p/

[39] "Esp32 Introduction To Esp32 | Esp32." Accessed: Jun. 15, 2024. [Online]. Available: https://www.electronicwings.com/esp32/introduction-to-esp32

[40] "ESP32-C3 Espressif Systems | Mouser Portugal." Accessed: Jul. 07, 2024. [Online]. Available: https://pt.mouser.com/ProductDetail/Espressif-Systems/ESP32-C3?qs=iLbezkQI%252BsiMXe7tMG%252Bo%2FA%3D%3D

[41] "ESP32-H2-DevKitM-1 — esp-dev-kits documentation." Accessed: Mar. 20, 2024. [Online]. Available: https://espressif-docs.readthedocs-hosted.com/projects/esp-dev-kits/en/latest/esp32h2/esp32-h2-devkitm-1/user_guide.html

[42] "ESP32-C6-DevKitC-1 v1.2 — esp-dev-kits documentation." Accessed: Mar. 20, 2024. [Online]. Available: https://espressif-docs.readthedocs-hosted.com/projects/esp-dev-kits/en/latest/esp32c6/esp32-c6-devkitc-1/user_guide.html

Anexos

Anexo A

Tabela 13 - Tabela de testes realizados e resultados obtidos para o cenário de duas tomadas e dois interruptores em *Matter over Thread*

Atributo	Descrição	Equipamento	Versão	Cluster
OnOff	Indica se o dispositivo está ligado ou desligado.	Lâmpadas	1.0	-
		Interruptores	1.0	
		Tomadas Inteligentes	1.0	
		Controladores de portões de garagens	1.0	
		Fogões	1.3	
		Exaustores	1.3	
StartUpOnOff	Define se o dispositivo deve começar ligado ou desligado quando recebe energia.	Interruptores	1.0	-
		Tomadas Inteligentes	1.0	
OnTime	Indica o tempo restante em segundos que o dispositivo vai continuar ligado.	Interruptores	1.0	-
		Tomadas Inteligentes	1.0	
		Fogões	1.3	
		Exaustores	1.3	
OffWaitTime	Indica o tempo que está desligado em segundos até ser ligado.	Tomadas Inteligentes	1.0	-
CurrentHue	Indica valor atual da matriz de cores.	Lâmpadas	1.0	-
CurrentSaturation	Indica valor atual de saturação.	Lâmpadas	1.0	-
ColorMode	Define o modo da cor.	Lâmpadas	1.0	-
RemainingTime	Define o tempo restante em segundos até a lâmpada ser desligada.	Lâmpadas	1.0	-
DriftCompensation	Indica como a intensidade da luminosidade deve aumentar/reduzir ao longo do tempo.	Lâmpadas	1.0	-
MeasuredValue	Indica o valor medido pelo sensor, mas no caso do valor está a 0 é porque ocorreu algum erro na medição por parte do sensor.	Sensores de luminosidade	1.0	-
		Sensores de temperatura		
		Sensores de fluxo		
		Sensores de pressão		
MinMeasuredValue	Indica o valor mínimo que pode ser medido.	Sensores de luminosidade	1.0	-
		Sensores de temperatura		
		Sensores de fluxo		
		Sensores de pressão		
MaxMeasuredValue	Indica o valor máximo que pode ser medido.	Sensores de luminosidade	1.0	-
		Sensores de temperatura		
		Sensores de fluxo		

		Sensores de pressão		
LightSensorType	Indica o tipo de sensor que está a ser utilizado.	Sensores de luminosidade	1.0	-
LockState	Indica de está trancada.	Fechaduras Inteligentes	1.0	-
LockType	Indica o tipo de fechadura presente.	Fechaduras Inteligentes	1.0	-
ActuatorEnabled	Indica se a fechadura pode ou não executar processos remotos.	Fechaduras Inteligentes	1.0	-
DoorState	Indica se a porta está aberta ou fechada.	Fechaduras Inteligentes	1.0	-
DoorOpenEvents	Conta o número de vezes que a porta foi aberta.	Fechaduras Inteligentes	1.0	-
DoorClosedEvents	Conta o número de vezes que a porta foi fechada.	Fechaduras Inteligentes	1.0	-
OpenPeriod	Indica o tempo que a porta esteve aberta em minutos.	Fechaduras Inteligentes	1.0	-
CatalogList	Especifica uma lista das aplicações suportadas pelo dispositivo de reprodução de vídeo.	Controladores de aplicações de transmissão	1.0	*Application Launch Cluster*
CurrentApp	Especifica a aplicação que está a ser controlada.	Controladores de aplicações de transmissão	1.0	*Application Launch Cluster*
LaunchApp	Inicializa a aplicação.	Controladores de aplicações de transmissão	1.0	*Application Launch Cluster*
StopApp	Termina a aplicação.	Controladores de aplicações de transmissão	1.0	*Application Launch Cluster*
OutputList	Especifica uma lista de outputs suportados pelo dispositivo de reprodução de vídeo.	Controladores de aplicações de transmissão	1.0	*Audio Output Cluster*
CurrentOutput	Especifica o output que está a ser controlado.	Controladores de aplicações de transmissão	1.0	*Audio Output Cluster*
SelectAudioOutput	Define qual a saída que o utilizador pretende controlar.	Controladores de aplicações de transmissão	1.0	*Audio Output Cluster*
RenameOutput	Permite que o utilizador altere o nome da saída.	Controladores de aplicações de transmissão	1.0	*Audio Output Cluster*
ChannelList	Especifica a lista de canais suportados	Controladores de aplicações de transmissão	1.0	*Channel Cluster*
CurrentChannel	Especifica o canal que está a ser apresentado pelo dispositivo.	Controladores de aplicações de transmissão	1.0	*Channel Cluster*
ChangeChannel	Permite alterar o canal para o canal passado no argumento.	Controladores de aplicações de transmissão	1.0	*Channel Cluster*
ChangeChannelByNumber	Os canais têm números atribuídos, e isto permite ao utilizador mudar o canal para o canal com o número especificado.	Controladores de aplicações de transmissão	1.0	*Channel Cluster*

SkipChannel	Permite que se mude de canal ou para cima ou para baixo, mas saltando um x número de canais conforme definido pelo utilizador.	Controladores de aplicações de transmissão	1.0	*Channel Cluster*
AcceptHeader	Especifica a lista dos tipos de conteúdo aceites pelo dispositivo de reprodução de vídeo.	Controladores de aplicações de transmissão	1.0	*Content Launcher Cluster*
SupportedStreaming Protocols	Especifica os protocolos de transmissão suportados.	Controladores de aplicações de transmissão	1.0	*Content Launcher Cluster*
LaunchContent	Permite inicializar o conteúdo especificado pelo utilizador.	Controladores de aplicações de transmissão	1.0	*Content Launcher Cluster*
LaunchURL	Permite inicializar o conteúdo através do *URL* desde que seja compatível com o cabeçalho e o protocolo de transmissão	Controladores de aplicações de transmissão	1.0	*Content Launcher Cluster*
CurrentState	Especifica o estado de reprodução, ou seja, se está parado ou inicializado.	Controladores de aplicações de transmissão	1.0	*Media Playback Cluster*
StartTime	Especifica o tempo para a reprodução começar.	Controladores de aplicações de transmissão	1.0	*Media Playback Cluster*
PlaybackSpeed	Especifica a velocidade que o media irá ser reproduzido.	Controladores de aplicações de transmissão	1.0	*Media Playback Cluster*
Play	Inicia a reprodução.	Controladores de aplicações de transmissão	1.0	*Media Playback Cluster*
Stop	Para a reprodução.	Controladores de aplicações de transmissão	1.0	*Media Playback Cluster*
StartOver	Volta a reproduzir	Controladores de aplicações de transmissão	1.0	*Media Playback Cluster*
Type	Especifica o tipo de estore que está a ser utilizado.	Estores Inteligentes	1.0	-
PhysicalClosedLimit Lift	Especifica em centímetros o máximo que o estore pode subir.	Estores Inteligentes	1.0	-
PhysicalClosedLimit Tilt	Especifica em centímetros o máximo que o estore pode descer.	Estores Inteligentes	1.0	-
CurrentPositionLift	Indica em centímetros quanto o estore está para cima.	Estores Inteligentes	1.0	-
CurrentPositionTilt	Indica em centímetros quanto o estore está para baixo.	Estores Inteligentes	1.0	-
NumberOfActuations Lift	Número de vezes que subiram os estores desde a instalação.	Estores Inteligentes	1.0	-
CurrentPositionLiftP ercentage	Número de vezes que desceram os estores desde a instalação	Estores Inteligentes	1.0	-
OperationalStatus	Estado operacional do estore naquele determinado momento.	Estores Inteligentes	1.0	-

Mode	Modo de funcionamento.	Estores inteligentes	1.0	-
		Frigoríficos e congeladores	1.0	-
		Máquinas de lavar a loiça	1.2	-
		Máquinas de lavar a roupa	1.2	-
		Aspiradores robot	1.2	*Run Mode Cluster*
		Aspiradores robot	1.2	*Clean Mode Cluster*
SafetyStatus	Sensores de segurança que são ativados quando necessário	Estores inteligentes	1.0	-
TemperatureDifference	valor = (temperatura in °C) x 100, logo -4°C ⇒ -400	Termostatos	1.0	-
UnsignedTemperature	valor = (temperatura in °C) x 10, logo -4°C ⇒ -40	Termostatos	1.0	-
ACErrorCodeBitmap	Valor que altera entre 0 e 4, onde cada um destes valores está atribuído a um erro para especificar se o termostato tem algum problema.	Termostatos	1.0	-
AlarmCodtmap	Valor que altera entre 0 e 2 para ativar o alarme do termostato num caso de um possível erro.	Termostatos	1.0	-
HVACSystemTypeBitmap	Especifica se o ar condicionado está em modo de aquecimento ou arrefecimento e qual o combustível utilizado pelo sistema *HVAC*.	Termostatos	1.0	-
RemoteSensingBitmap	Especifica se o ar condicionado ajusta a temperatura conforme a temperatura local ou a temperatura exterior	Termostatos	1.0	-
ACLouverPositionEnum	Especifica o quão aberto está o ar condicionado.	Termostatos	1.0	-
ThermostatControlSequence	Define a sequência em que o ar condicionado deve de trabalhar.	Termostatos	1.0	-
ThermostatSystemMode	Especifica o modo em que trabalha, sendo o mais importante, se trabalha manualmente ou automaticamente.	Termostatos	1.0	-
PumpStatusBitmap	Especifica o estado do dispositivo de bombeamento do ar condicionado.	Controladores de *HVAC*		
OperationModeEnum	Especifica o modo que está a ser utilizado, sendo possível ser o modo normal, mínimo, máximo e local (valor definido por omissão).	Controladores de *HVAC*		*Pump Configuration and Control Cluster*
		Ar condicionado		

ControlModeEnum	Define a velocidade de bombeamento conforme um valor definido, pressão, temperatura ou fluxo.	Controladores de *HVAC*	1.0	*Pump Configuration and Control Cluster*
		Ar condicionado	1.2	
MaxPressure	Especifica o valor máximo de pressão para a bomba.	Controladores de *HVAC*		*Pump Configuration and Control Cluster*
MaxSpeed	Especifica o valor máximo da velocidade da bomba.	Controladores de *HVAC*		*Pump Configuration and Control Cluster*
MaxFlow	Especifica o valor máximo de fluxo da bomba.	Controladores de *HVAC*		*Pump Configuration and Control Cluster*
RockBitmap	Especifica se a oscilação das ventoinhas é da esquerda para a direita ou da direita para a esquerda, e a oscilação circular.	Controladores de *HVAC*	1.0	*Fan Control Cluster*
		Ar condicionado	1.0	*Fan Control Cluster*
		Ventoinhas	1.2	-
WindBitMap	Permite definir se a ventoinha é suposta trabalhar em modo natural ou em modo de dormir, sendo 0 o modo de dormir e 1 o modo natural.	Controladores de *HVAC*	1.0	*Fan Control Cluster*
		Ar condicionado	1.0	*Fan Control Cluster*
		Ventoinhas	1.2	-
AirflowDirectionEnum	Especifica a direção do fluxo do ar.	Controladores de *HVAC*	1.0	*Fan Control Cluster*
		Ar condicionado	1.0	*Fan Control Cluster*
		Ventoinhas	1.2	-
FanModeEnum	Especifica o modo que está a ser utilizado pelas ventoinhas.	Controladores de *HVAC*	1.0	*Fan Control Cluster*
		Ar condicionado	1.0	*Fan Control Cluster*
		Ventoinhas	1.2	-
FanModeSequenceEnum	Especifica os modos que são possíveis utilizar pela ventoinha de forma a limitar os modos que o utilizador não pretende que as ventoinhas utilizem.	Controladores de *HVAC*	1.0	*Fan Control Cluster*
		Ar condicionado	1.2	
TemperatureDisplayModeEnum	Especifica se a temperatura é demonstrada em Celsius ou Fahrenheit.	Controladores de *HVAC*	1.0	*Thermostat User Interface Configuraton Cluster*
KeypadLockoutEnum	Especifica quais as funcionalidades é que não podem ser utilizadas no termostato.	Controladores de *HVAC*	1.0	*Thermostat User Interface*

				Configuraton Cluster
ScheduleProgrammingVisibilityEnum	Define se é possível ou não utilizar agendamento do funcionamento do termostato.	Controladores de *HVAC*	1.0	*Thermostat User Interface Configuraton Cluster*
SupportedModes	Modos suportados	Frigoríficos e congeladores	1.2	-
		Máquinas de lavar a loiça	1.2	
		Máquinas de lavar a roupa	1.2	
		Micro-ondas	1.3	
		Fornos	1.3	
CurrentMode	Modo que está a ser utilizado	Frigoríficos e congeladores	1.2	-
		Máquinas de lavar a loiça	1.2	
		Máquinas de lavar a roupa	1.2	
		Micro-ondas	1.3	
		Fornos	1.3	
StartUpMode	O modo definido para quando o dispositivo é inicializado	Frigoríficos e congeladores	1.2	-
		Máquinas de lavar a loiça	1.2	-
		Máquinas de lavar a roupa	1.2	-
		Aspiradores robot	1.2	*Run Mode Cluster*
		Micro-ondas	1.3	-
		Fornos	1.3	-
AlarmMap	Utilizado para avisar se a porta do dispositivo está aberta.	Frigoríficos e congeladores	1.2	-
		Máquinas de lavar a loiça	1.2	
ChangeToModeResponse	Códigos de estado específicos para erros que estejam a ocorrer com o aspirador.	Aspiradores robot	1.2	*Run Mode Cluster*
OperationalStateEnum	Utilizado para indicar se o aspirador está a ir para o espaço de carregamento, se está a carregar, ou se está no espaço de carregamento, mas simplesmente parado.	Aspiradores robot	1.2	*Operational State Cluster*
ErrorStateEnum	Indica erros que estejam a acontecer com o aspirador por não estar a carregar automaticamente.	Aspiradores robot	1.2	*Operational State Cluster*
AlarmStateEnum	Define o estado de alarme, ou seja, se está ativado (0) ou se está ativado (1-2).	Alarmes de fumo e de monóxido de carbono	1.2	-
SensitivityEnum	Utilizado para definir a sensibilidade do alarme.	Alarmes de fumo e de monóxido de carbono	1.2	-
AirQuality	Especificação estado da qualidade do ar. O valor 0 indica um estado desconhecido, e se o estado for conhecido, varia entre 1 a 6, sendo 1 a melhor qualidade possível.	Sensores de qualidade de ar	1.2	-
SpeedMax	Valor entre 0 e 100 para definir a velocidade da ventoinha	Ventoinhas	1.2	-

CookTime	Indica o tempo que o micro-ondas irá funcionar.	Micro-ondas	1.3	-
MaxCookTime	Indica o tempo máximo que é possível definir no "*CookTime*".	Micro-ondas	1.3	-
PowerSetting	Indica a potência que irá ser utilizada pelo micro-ondas quando está em funcionamento.	Micro-ondas	1.3	-
MinPower	Potência mínima que é possível definir.	Micro-ondas	1.3	-
MaxPower	Potência máxima que é possível definir.	Micro-ondas	1.3	-
TemperatureSetpoint	Indica a temperatura que se pretende atingir pelo fogão.	Fogões	1.3	-
MinTemperature	Indica a temperatura mínima do fogão.	Fogões	1.3	-
MaxTemperature	Indica a temperatura máxima do fogão.	Fogões	1.3	-
FanMode	Indica o modo que está a ser utilizado pela ventoinha do exaustor.	Exaustores	1.3	-
SpeedSetting	Indica a velocidade a ser utilizada pela ventoinha do exaustor.	Exaustores	1.3	-
SpeedMax	Indica a velocidade máxima que pode ser utilizada pela ventoinha.	Exaustores	1.3	-
SupportedDrynessLevels	Indica os níveis de secagem suportados pela máquina de secar.	Máquinas de secar a roupa	1.3	-
SelectedDrynessLevel	Indica o nível selecionado para ser utilizado na secagem da roupa.	Máquinas de secar a roupa	1.3	-
TemperatureLevel	Indica a temperatura selecionada na secagem da roupa.	Máquinas de secar a roupa	1.3	-
ESAType	Indica as propriedades básicas, como a energia que é consumida, gerada e guardada no dispositivo.	Controladores para carregadores de carros elétricos	1.3	*Device Energy Management Cluster*
AbsMinPower	Indica a energia mínima que um dispositivo pode consumir quando é ligado.	Controladores para carregadores de carros elétricos	1.3	*Device Energy Management Cluster*
AbsMaxPower	Indica a energia máxima que um dispositivo pode consumir quando é ligado.	Controladores para carregadores de carros elétricos	1.3	*Device Energy Management Cluster*
State	Indica se a tomada do carregador está inserida do carro ou não	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*
ChargingEnabledUntil	Indica as horas (*UCT*) que o carregador tem de parar de carregar.	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*

MinimumChargeCurrent	Indica a corrente mínima utilizada para carregar o carro.	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*
MaximumChargeCurrent	Indica a corrente máxima utilizada para carregar o carro.	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*
NextChargeStartTime	Indica as horas (*UCT*) que o carregador tem de começar a carregar.	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*
NextChargeTargetTime	Indica uma estimativa das horas (*UCT*) a que o carro tem de estar completamente carregado.	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*
NextChargeRequiredEnergy	Indica uma estimativa da quantidade de energia que tem de ser adicionada ao carro no próximo carregamento que o carro faça.	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*
ApproximateEVEfficiency	Indica uma estimativa da quantidade de quilómetros que o carro vai conseguir percorrer com a quantidade de energia carregada.	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*
StateOfCharge	Indica em percentagem a bateria do carro.	Controladores para carregadores de carros elétricos	1.3	*Energy EVSE Cluster*
EnergyBalances	Indica uma lista com as prioridades do dispositivo.	Controladores para carregadores de carros elétricos	1.3	*Energy Preference Cluster*
CurrentEnergyBalance	Indica o que o dispositivo está a priorizar.	Controladores para carregadores de carros elétricos	1.3	*Energy Preference Cluster*
LowPowerModeSensitivities	Indica uma lista das condições em que o dispositivo deve de entrar em modo de consumo baixo de energia.	Controladores para carregadores de carros elétricos	1.3	*Energy Preference Cluster*
StateValue	É um atributo booleano que indica se o mesmo está a verdadeiro ou a falso. Por exemplo no caso dos detetores de congelamento, se o atributo estiver a verdadeiro é porque há a probabilidade de haver congelamento naquela área, se estiver a falso é porque não há a probabilidade de haver congelamento.	Dispositivos para controlo de água	1.3	-
SupportedSensitivityLevels	Indica o nível de sensibilidade dos detetores e sensores.	Dispositivos para controlo de água	1.3	-
DefaultSensitivityLevel	Indica o nível de sensibilidade que deve de ser utilizado por omissão.	Dispositivos para controlo de água	1.3	-

AlarmsActive	Indica se o alarme está ativado ou desativado.	Dispositivos para controlo de água	1.3	-