Universidade de Aveiro
Ano 2023

JORGE MIGUEL SOARES DE OLIVEIRA

GATEWAY AGNOSTICA BLE PARA COAP

Universidade de Aveiro
Ano 2023

JORGE MIGUEL SOARES DE OLIVEIRA

GATEWAY AGNOSTICA BLE PARA COAP

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Informática Aplicada, realizada sob a orientação científica do Doutor Pedro Alexandre de Sousa Gonçalves, Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro, e do Doutor João Gonçalo Gomes de Paiva Dias, Professor Coordenador com Agregação da Escola Superior de Tecnologia e Gestão de Águeda da Universidade de Aveiro

Dedico este trabalho àqueles que, com sua orientação discreta e apoio valioso, foram fundamentais para o seu desenvolvimento, e cuja contribuição permanece conhecida apenas por nós.

o júri

presidente

Prof. Doutor Joaquim José de Castro Ferreira
professor Coordenador C/ Agregação da Universidade de Aveiro

vogais

Prof. Doutor Filipe Manuel Simões Caldeira
professor Adjunto do Instituto Politécnico de Viseu - Escola Superior Tecnologica e Gestão

Prof. Doutor Pedro Alexandre de Sousa Gonçalves
professor Professor Adjunto da Universidade de Aveiro

palavras-chave BLE, COAP, gateway.

resumo

A tecnologia Bluetooth tem vindo a ganhar um considerável papel na interligação de dispositivos no mundo da Internet das coisas, ironicamente quando se achava que era uma tecnologia condenada ao esquecimento. A última versão definida na norma tem vindo a propor novos modos de transmissão e poupança de energia, duplicação de alcance de transmissão, modos de funcionamento com suporte de *mesh*, e novas tecnologias de localização dos nós. Este tipo de desenvolvimento não é obviamente independente da massificação da utilização de rádios Bluetooth em dispositivos móveis como portáteis e telefones, mas também na maioria dos gadgets que utilizamos, desde pulseiras de exercício e relógios a auscultadores e até a televisões. Como sempre, a massificação oferece ganhos de escala e baixa o custo da tecnologia.

O modo *Low Energy* incorporado no Bluetooth 4.0, além do modo de transmissão muito mais eficiente em termos energéticos, e por isso adequado à utilização em redes de sensores, trouxe também um novo protocolo de troca de informação entre os *peripherals* e os *centrals* de BLE, que normaliza a informação disponibilizada em cada sensor, bem como a estrutura de dados disponível. O trabalho do *Bluetooth Special Interest Group* incluiu a normalização dos identificadores dos serviços e das características disponibilizados, o que facilita um acesso agnóstico, independentemente de se aceder a um sensor cardíaco ou a um podómetro. Ainda assim, as implementações de BLE disponíveis são completamente dependentes do perfil GATT do sensor, o que impede a reutilização com outro tipo de sensores.

O *Constrained Application Protocol* (CoAP) é um protocolo de transferência de informação muito utilizado na Internet das Coisas, que foi desenvolvido com especial foco na eficiência por ter sido destinado a dispositivos constrangidos em termos de memória e processamento, mas, dadas as tecnologias em que foi baseado (i.e. HTTP, REST), tem tudo para ser agnóstico em termos de modelo de dados, e por isso utilizável com qualquer tipo de sensor.

O trabalho proposto consiste no desenvolvimento de uma solução agnóstica de integração de BLE com o mundo da Internet, leve e que possa ser utilizada em redes de sensores genéricos.

keywords BLE, COAP, gateway.

abstract

Bluetooth technology has been gaining a considerable role in the interconnection of devices in the world of the Internet of Things, ironically when it was thought that it was a technology doomed to oblivion. The latest version defined in the standard has been proposing new transmission and energy saving modes, doubling of transmission range, operating modes with mesh support, and new node location technologies. This type of development is obviously not independent of the widespread use of Bluetooth radios in mobile devices such as laptops and phones, but also in most of the gadgets we use, from exercise bracelets and watches to headphones and even televisions. As always, massification offers gains in scale and lowers the cost of technology.

The Low Energy mode incorporated in Bluetooth 4.0, in addition to the much more efficient transmission mode in terms of energy, and therefore suitable for use in sensor networks, also brought a new protocol for exchanging information between peripherals and BLE centrals, which normalizes the information available in each sensor, as well as the available data structure. The work of the Bluetooth Special Interest Group included standardizing the identifiers of the services and features available, which facilitates agnostic access, regardless of whether you are accessing a heart sensor or a pedometer. Even so, the available BLE implementations are completely dependent on the GATT profile of the sensor, which prevents reuse with other types of sensors.

The Constrained Application Protocol (CoAP) is an information transfer protocol widely used in the Internet of Things, which was developed with a special focus on efficiency because it was intended for constrained devices in terms of memory and processing, but, given the technologies in which was based (i.e. HTTP, REST), has what it takes to be data model agnostic, and therefore usable with any type of sensor.

The proposed work consists in the development of an agnostic solution for integrating BLE with the Internet world, lightweight and that can be used in generic sensor networks.

Índice

Siglas

- API (Application Programming Interface)
- COAP (Constrained Application Protocol)
- UDP (User Datagram Protocol)
- 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks)
- GATT (Generic Attribute Profile)
- IoT (Internet of Things)
- HTTP (Hypertext Transfer Protocol)
- RAM (Random Access Memory)
- UDP (User Datagram Protocol)
- TCP (Transmission Control Protocol)
- REST (Representational State Transfer)
- XML (Extensible Markup Language)
- JSON (JavaScript Object Notation)
- CBOR (Concise Binary Object Representation)
- iOS (Apple's mobile operating system)
- ATT (Attribute)
- GATT (Generic Attribute Profile)
- IPv6 (Internet Protocol Version 6)
- 6LoWPAN (IPv6 over Low-Power Wireless Personal Area Networks)
- Web (World Wide Web)
- TCP/IP (Transmission Control Protocol/Internet Protocol)
- CBOR (Concise Binary Object Representation)
- CPU (Central Processing Unit)
- RAM (Random Access Memory)
- IRDA (Infrared Data Association)
- SIG (Special Interest Group)

- REST (Representational State Transfer)
- SOAP (Simple Object Access Protocol)
- Datagram Transport Layer Security (DTLS)
- Proof-of-Possession (PoP)
- ACE (Authentication and authorization for Constrained Environments)
- Internet Research Task Force (IRTF)
- Information-Centric Networking RG (ICNRG)
- Decentralized Internet Infrastructure RG (DINRG)
- Thing-to-Thing (T2TRG)
- Internet Engineering Task Force (IETF)

Índice de Figuras

Índice de Tabelas

1. Introdução

A realização deste projeto de desenvolvimento de uma *gateway COAP BLE* agnóstica representa um ponto de referência importante na procura por soluções inovadoras para a integração de sensores. O seu propósito é estabelecer uma conexão eficiente entre os sensores *BLE* e sistemas de monitorização baseados na nuvem, oferecendo uma ponte tecnológica que permitirá que os dados coletados sejam transmitidos, armazenados e posteriormente utilizados para análises relevantes. Os contextos para a realização deste trabalho são profundamente influenciados pelo cenário atual de crescimento na utilização de dispositivos, que comumente empregam sensores *BLE* de baixo consumo energético.

A essência deste labor consiste em abordar uma necessidade fundamental como viabilizar a integração de sensores que, embora poderosos em informações, como dados de saúde no caso de pulseiras de *fitness*, não possuem conectividade direta à *Internet*. A *gateway COAP BLE* preenche essa lacuna, atuando como um intermediário inteligente que coleta dados dos sensores e os transmite posteriormente uma base de dados, onde podem ser acedidos e processados por aplicações e sistemas de alto nível.

O grande diferencial desta *gateway* é a sua natureza agnóstica em relação ao modelo de dados. Isso significa que ela é altamente adaptável e flexível, podendo comunicar com uma variedade de sensores e dispositivos. Seja para monitorar a frequência cardíaca, medir a temperatura corporal, ou coletar dados de movimento, a *gateway* é capaz de aceitar e transmitir informações a partir de diferentes fontes, tornando-se um elemento versátil e capaz de atender a diversas aplicações.[1]

Para melhor se entender o alcance deste projeto, podemos imaginar um exemplo prático: um ginásio moderno que utiliza a *gateway COAP BLE* para integrar dados de sensores localizados nas pulseiras dos seus membros. Estes sensores podem capturar uma ampla gama de informações, desde a atividade física realizada até os sinais vitais dos utilizadores. No entanto, esses sensores frequentemente não têm conexão direta à Internet. É aí que a *gateway* entra

[1]

em ação. Ela coleta os dados dos sensores e envia-os para a base de dados, onde esses dados podem ser processados por sistemas de análise, gerando informações valiosas. Posteriormente o cliente pode aceder a esses através da *gateway* que encapsula os dados em pacotes de comunicação COAP.

Um aspecto crucial deste projeto é a consideração cuidadosa do consumo de energia. A otimização do uso de recursos, incluindo a eficiência energética, é uma prioridade. Isso ocorre porque muitos sensores operam com baterias e precisam de uma abordagem de baixo consumo de energia para garantir uma vida útil satisfatória das baterias.

Em contraste, do lado da *Internet*, o foco é a disponibilidade de largura de banda para suportar a transmissão eficiente de dados. Este projeto procura o equilíbrio entre esses dois aspectos, garantindo que o consumo de energia seja minimizado no lado dos sensores, enquanto a conectividade de alta largura de banda esteja disponível para receber e processar dados sem restrições.

A análise de desempenho detalhada apresentada neste documento confirma a eficácia da *gateway* COAP BLE em lidar com cenários desafiadores, onde diversos sensores estão constantemente a enviar dados. A estabilidade, eficiência e confiabilidade da *gateway* são fundamentais para garantir que os dados coletados sejam entregues e processados de forma precisa e oportuna.

A transformação de dados brutos de sensores em informações significativas é a chave principal. Esses dados podem ser usados para criar aplicações avançadas, como monitorização de atividade física e análises personalizadas de saúde, bem como em outros ambientes que façam uso do *BLE*.

Em resumo, a gateway *COAP BLE* representa um avanço tecnológico essencial que se adapta a um cenário cada vez mais conectado. Ela desempenha um papel vital na transição de dados de baixo nível para aplicações de alto nível, permitindo a geração de coisas valiosas para aprimorar os diversos ambientes e a experiência dos utilizadores. Este projeto lança as bases para um futuro em que os dispositivos não apenas coletam dados, mas também os utilizam para melhorar a qualidade de vida.

1.1. Motivação

A motivação para o desenvolvimento da gateway *CoAP BLE* é impulsionada por uma série de fatores essenciais, refletindo as necessidades e desafios no domínio da integração de dispositivos IoT (*Internet das Coisas*) e wearables com sistemas de monitorização na nuvem.

O aumento exponencial da utilização de sensores em várias aplicações tem sido uma tendência notável na era da *IoT*. Sensores *BLE*, em particular, são amplamente utilizados em dispositivos vestíveis, como pulseiras de *fitness* e sensores médicos. No entanto, a maioria desses sensores não têm uma ligação direta à *Internet* e operam com baixo consumo energético. A transformação destes dados de sensores em informações valiosas e a sua transferência eficiente para a nuvem tornaram-se uma necessidade premente. Nesse contexto, a gateway *CoAP BLE* desempenha um papel crucial.

Um exemplo prático seria um ginásio que faz uso de uma gateway *CoAP BLE*. As pulseiras dos membros do ginásio contêm sensores *BLE* que monitorizam várias métricas, como frequência cardíaca, níveis de oxigénio no sangue e contagem de passos. Como estes sensores, por si só, não possuem a capacidade de se conectarem diretamente à *Internet* é aqui que a *gateway COAP BLE* entra em jogo, funcionando como um intermediário. Ela coleta os dados dos sensores das pulseiras e transforma-os num formato padrão, como CBOR (*Concise Binary Object Representation*) ou *JSON* (*JavaScript Object Notation*). Esses dados, agora prontos, são transmitidos para a nuvem, onde podem ser processados, armazenados e utilizados para monitorização de *fitness*, análise de desempenho e outras aplicações de alto nível.

Um outro aspeto importante desta motivação está relacionado com o consumo de energia. Sensores *BLE* são conhecidos por serem eficientes em termos energéticos, permitindo uma longa vida útil da bateria. No entanto, a comunicação de alta largura de banda com a *Internet* pode ser intensiva em termos de energia. A *gateway CoAP BLE* serve como um mecanismo que permite que os sensores operem com eficiência em termos energéticos, ao mesmo tempo que fornece uma interface com a Internet com maior largura

de banda. Isso é vital para assegurar que os dispositivos *wearables* funcionem de forma confiável durante longos períodos.

Em resumo, o desenvolvimento da *gateway CoAP BLE* é motivado pela necessidade de uma solução versátil e eficiente para a integração de sensores *BLE* e dispositivos *IoT* com a nuvem. Essa solução é agnóstica em relação ao modelo de dados, permitindo que dados de várias fontes sejam unificados e transmitidos eficientemente.

1.2. Objetivos

O objetivo principal é o desenvolvimento de uma *gateway COAP BLE* que funcione como um componente versátil e agnóstico em termos de modelo de dados. A *gateway* foi projetada para facilitar a integração de dispositivos *IoT* e *wearables* na *cloud*, independentemente de sua origem ou formato de dados. A agnosticidade em relação ao modelo de dados significa que a *gateway* é capaz de lidar com informações provenientes de uma variedade de fontes e com diferentes estruturas de dados.

Um dos principais requisitos da *gateway* é suportar as codificações *CBOR* e *JSON*. Isso garantirá que os dados dos dispositivos conectados possam ser representados de maneira flexível, atendendo às necessidades de diferentes aplicações e sistemas. A inclusão desses dois formatos de codificação amplamente usados oferece interoperabilidade e flexibilidade na representação de dados.

Para ilustrar a aplicação prática desta *gateway*, consideremos um cenário hipotético num ginásio. A *gateway* recebe as leituras dos sensores no seu formato nativo e, graças à sua agnosticidade de modelo de dados, é capaz de interpretar essas informações em diferentes formatos. As leituras dos sensores são então codificadas em *CBOR* ou *JSON*, dependendo dos requisitos do sistema do ginásio. Isso permite que os dados sejam transmitidos para a nuvem, onde podem ser processados, armazenados e disponibilizados para os clientes e os próprios frequentadores do ginásio.

Essa aplicação do projeto exemplifica a capacidade da *gateway* de atuar como um intermediário eficiente entre dispositivos *BLE*, como as pulseiras de fitness, e a infraestrutura de *cloud*, garantindo que os dados coletados sejam acessíveis e utilizáveis. Além disso, a *gateway* oferece a flexibilidade de se adaptar a diferentes formatos de dados, tornando-a uma solução versátil e pronta para atender a uma variedade de necessidades em cenários *IoT* e de *wearables*.

1.3. Estrutura

O presente documento segue uma organização lógica, dividido em várias secções que abordam diferentes aspetos relacionados com o IoT e com a implementação de uma *gateway CoAP*/*BLE*.

Na primeira parte, encontra-se a secção de introdução, que fornece uma visão geral do conteúdo do documento. Essa introdução é subdividida em quatro partes: motivação, objetivos e estrutura, descrevendo sucintamente o que será abordado, a razão para a pesquisa, os objetivos específicos e a organização do documento.

A segunda parte deste documento dedicara-se à análise dos trabalhos relacionados, conduzida por meio de uma pesquisa detalhada no campo de estudos em questão. Esta seção fornece uma visão abrangente das pesquisas e projetos anteriores que estão diretamente relacionados com este trabalho, permitindo assim uma contextualização mais sólida das contribuições únicas e inovações que implementação de *gateway CoAP/BLE* traz para o cenário atual.

Na sequência, o documento aborda a implementação de uma *gateway CoAP*/*BLE*. Esta secção divide-se em duas partes: arquitetura e protótipo, descrevendo a estrutura da implementação e os detalhes da criação do protótipo. Isso inclui o *scan* de dispositivos, o armazenamento de dados, a criação de servidor e cliente, a criação de bases de dados locais e na nuvem, além das vantagens do uso de *CBOR* (*Concise Binary Object Representation*).

A quarta secção concentra-se em testes e análise de resultados, incluindo a criação de cenários de testes, análise do tráfego *BLE* e da interface *CoAP*.

A quinta secção é dedicada à discussão, onde os resultados são analisados em profundidade e discutidos à luz dos objetivos do documento.

A sexta e última secção apresenta as conclusões do documento, resumindo os principais resultados e suas implicações.

2. Tecnologias usadas para o desenvolvimento de soluções *IoT*

2.1. Introdução

As redes de sensores sem fios com características de baixo consumo de energia estão a tornar-se cada vez mais importantes. *Bluetooth Low Energy* (*BLE*) é uma das mais recentes tecnologias de comunicação sem fios de curto alcance que foi unida com a versão 2010 da norma Bluetooth e integrada com a especificação principal do Bluetooth versão 4.0 (Lin et al., 2018).

O objetivo geral foi obter tecnologia de baixo custo, com baixo consumo energético e com alto modo de hibernação, que, através da transmissão, consegue poupar energia e largura de banda. Contudo, ao integrar enormes quantidades de dispositivos, enfrenta-se o problema de interoperabilidade, que está salvaguardado graças ao *IPV6*, desde que os dispositivos o suportem.

A *Internet Engineering Task Force (IETF)* normalizou a transmissão de pacotes *IPV6* sobre *Bluetooth* de baixa energia utilizando *IPV6* sobre *6LoWPAN* (Lin et al., 2018), que é referida como *IPV6* sobre *BLE*. Isto permite que os nós alcancem *IPV6* de ponta a ponta, mesmo entre tecnologias diferentes ao nível da camada de ligação. Isto promove claramente a interoperabilidade com diferentes dispositivos e redes de sensores. O *6LoWPAN* foi formulado para fornecer encaminhamento *IPV6* para redes de sensores sem fios de baixo consumo de energia e com recursos limitados. Dispositivos com recursos limitados, em inglês, são denominados "constrained devices"(Vasseur & Dunkels, 2010). Estes são dispositivos eletrónicos que apresentam restrições significativas em termos de processamento, memória e energia. São frequentemente utilizados em aplicações de Internet das Coisas (IoT) e em redes de sensores, onde a eficiência energética e a otimização de recursos são fundamentais. O *6LoWPAN é* uma camada de adaptação entre a camada de ligação e a de rede do modelo *OSI*. Como os pacotes *IP* são maiores que os pacotes *BLE,* o *6LoWPAN* fornece compressão dos cabeçalhos e fragmentação dos pacotes (Lin et al., 2018).

Os dispositivos em ambientes de *IoT* são também limitados em termos de recursos, menores larguras de banda e energia. Por isso a *IETF* desenvolveu um protocolo *CoAP,* que é um protocolo de transferência e comunicação *M2M* leve, aplicado em redes com dispositivos constrangidos. O *CoAP* utiliza *UDP* combinado com o confiável mecanismo de mensagens *CoAP*, proporcionando baixa sobrecarga e transmissão leve e confiável.

O *MQTT* (Lin et al., 2018) é outro protoloco de mensagens que corre em cima do *TCP*/*IP* e suporta o modelo de publicação e assinatura de mensagens. Por conseguinte, a *IBM* propôs o *MQTT* para redes de sensores (*MQTT-SN*), que é executado sobre o UDP (Lin et al., 2018), que é direcionado para o *IoT*.

Na configuração do projeto do artigo Lin et al., 2018 o objetivo é criar um ambiente doméstico inteligente e leve, contruindo uma rede de sensores *IPV6* sobre *BLE* com nós de sensores a correr o protocolo *CoAP* e *MQTT-SN*. Para facilitar a recolha de dados foram construídos dois *Gateways*: um serve como servidor *HTTP* e um cliente *COAP*. O outro servidor serve como servidor *HTTP* e subscritor *MQTT*. Os dados recolhidos são depois armazenados em bases de dados na *cloud* (Lin et al., 2018).

2.2. Tecnologias de Comunicação

Na maioria das situações, a necessidade de suportar o Protocolo de *Internet* (*IP*) não é uma preocupação imediata para os dispositivos *IoT* que operam em ambientes locais, como as pulseiras de monitorização num ginásio. Tecnologias estabelecidas, como o *Bluetooth* e padrões específicos, como o *ISO*/*IEEE 11073*, são perfeitamente adequadas para permitir a comunicação direta entre esses dispositivos e uma *gateway* local. Neste contexto, a prioridade reside na recolha de dados e na comunicação eficaz entre os dispositivos e a *gateway*. A necessidade de integrar os dados numa rede *IP* mais alargada, como a *Internet*, apenas se torna relevante quando se pretende partilhar informações com servidores em *cloud* ou outros dispositivos fora do ambiente local.

2.2.1. Bluetooth

O *Bluetooth* é uma tecnologia de comunicação sem fio amplamente utilizada para conectar dispositivos eletrónicos, como *smartphones*, colunas, fones e periféricos de computador. Existem duas versões principais do Bluetooth, o clássico e o *BLE*. Existe uma diferença entre o *Bluetooth* clássico e o *BLE*, sendo que é possível usar um dos dois ou ambos em simultâneo. O *BLE* tem alcance maior e gasta menos energia, contudo tem uma menor largura de banda (Matthias Nefzger, 2021).

Os dispositivos *BLE* possuem vários estados diferentes. Os estados são: ***standby***, ***scanner***, ***initiator***, ***master***, ***slave*** e ***advertiser***. No estado de standby não é transmitido ou recebido qualquer dado nem o dispositivo está conectado a nenhum outro, ao passo que o *scanner* procura ativamente por anunciantes, e pode estar sob a forma de *active* ou *passive*. Por outra perspectiva o *initiator* tenta ativamente iniciar uma conexão com outro dispositivo ao enviar um pacote de solicitação de conexão para o anunciante. Um anunciante é um dispositivo BLE que envia periodicamente informações de anúncio para que outros dispositivos possam detetá-lo e interagir com ele. O *master* refere-se ao estado de conexão a outro dispositivo

como mestre e o *slave* ao estado de conexão a outro dispositivo como “escravo”. O "mestre" controla a conexão e o "escravo" segue suas instruções, permitindo a comunicação entre dispositivos. Por último, *advertiser* consiste num estado *BLE* em que são enviados *broadcasts* periódicos de anúncios, sendo este constituído por vários subestados.

Os subestados de advertiser são: ***connectable undirected***, ***connectable***, ***non-connectable undirected***, ***discoverable undirected***. Com o *connectable undirected* qualquer dispositivo scanner pode iniciar uma conexão com o dispositivo em causa, enquanto o *connectable* directed apenas um dispositivo específico pode iniciar. Por outro lado, no estado "non-connectable undirected," o dispositivo *BLE* não permite descoberta nem conexões, enquanto no estado "discoverable undirected," ele permite descoberta, mas não conexões diretas.

Matthias Nefzger, 2021 descreveu a configuração de uma aplicação para *android* e um sensor com foco na comunicação *Bluetooth* entre os dois dispositivos. Optou por usar o *CoAP* como protocolo de comunicação.

O *Bluetooth* de Baixa Energia, ou *BLE* (Kang et al., 2016), representou uma verdadeira revolução na tecnologia sem fios desde a sua integração no padrão *Bluetooth,* em 2010. Desenvolvido pela *Ericsson* e subsequentemente supervisionado pelo *Bluetooth Special Interest Group* (*SIG*), (Kang et al., 2016), o *BLE* não só reduziu significativamente o consumo de energia em comparação com o *Bluetooth* clássico, como também trouxe maior eficiência à comunicação sem fios. Esta tecnologia substituiu gradualmente sistemas mais antigos, como a *Infrared Data Association* (*IrDA*), (Kang et al., 2016), como também se tornou na escolha de eleição para dispositivos *wearables*, sensores e uma vasta gama de dispositivos *IoT*. A alocação de canais publicitários em bandas de frequência que não interferem com os canais *Wi-Fi* de 2,4 GHz, promoveu uma convivência harmoniosa entre várias tecnologias sem fios num ambiente partilhado, consolidando ainda mais a posição do *BLE* como uma solução versátil e eficiente para comunicações de curto alcance.

Além disso, reduz o número de canais, alocando os canais publicitários (*advertising channels*) em bandas de frequência que não se sobrepõem com canais *Wi-Fi* de 2,4GHz, e com pacotes de tamanho flexível.

O *BLE* opera na faixa de espectro da banda *ISM* de 2,400 ~ 2,4835 GHz e possui 40 canais de 2 MHz, enquanto o *Bluetooth* clássico usa 79 canais de 1 MHz. A taxa de transmissão do *BLE* é de cerca de 1 *Mbit*/s e usa transmissão por salto de frequência para reduzir a interferência.

Os canais de publicidade do *BLE* são os 37, 38, 39 e os dispositivos ouvem regularmente esses canais para estabelecer conexões possíveis. Todos os canais restantes são para transmitir dados. O *Bluetooth* 4.0 tem dois protocolos principais: *Attribute Protocol* (*ATT*) e o *Generic Attribute Profile* (*GATT*). O *GATT* é construído sobre o *ATT*, que usa dados do *GATT* para definir a maneira como dois dispositivos *BLE* enviam e recebem mensagens. Existem duas funções no *GATT*: servidor e cliente. O servidor *GATT* armazena os dados transportados pelo *ATT* e aceita solicitações, comandos e confirmações *ATT* do cliente *GATT*. Ao receber solicitações dos clientes *GATT*, o servidor *GATT* responde enviando mensagens de resposta. Quando configurado, o servidor *GATT* também pode enviar indicações/notificações assíncronas para o cliente *GATT* quando eventos especificados ocorrerem no servidor *GATT* (Lin et al., 2018) .

2.2.2. ATT e GATT

Para que os dispositivos habilitados para *Bluetooth LE* transmitam dados entre si, eles formam um canal de comunicação da responsabilidade do *Generic Access Profile* (*GAP)*. Ele informa que um dos dispositivos deve assumir o papel de central e o outro deve assumir o papel de periférico. O central normalmente é um dispositivo poderoso como um smartphone, enquanto o periférico costuma ser algo que requer menos energia, como um sensor.

O primeiro passo é quando o dispositivo que deseja assumir a função de periférico estiver pronto para se conectar e entra no estado de anúncio e começar a anunciar sua presença usando seu rádio *LE* para transmitir pacotes de anúncio nos três canais de anúncio primários: 37, 38 e 39, conforme se verifica na Figura 1.

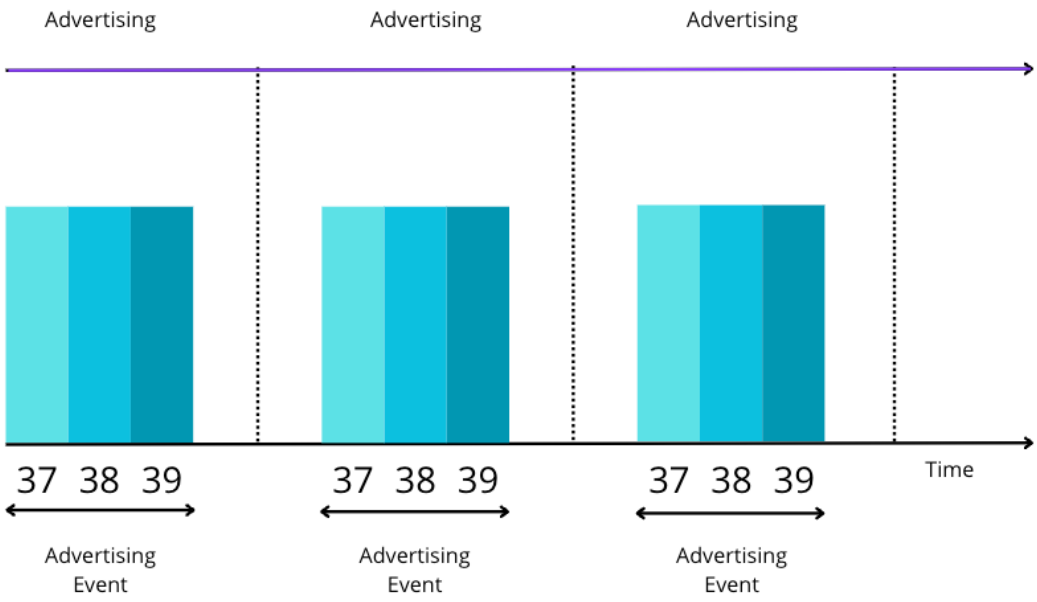

Figura 1 - Comunicação entre dispositivos Bluetooth (Adptado de Lin et al., 2018)

O segundo passo é quando o dispositivo que deseja assumir a função de central deve transitar para um estado de *scanning* e ouvir nestes três canais para um pacote de *advertising*. Quando o intervalo de *scanning* coincide com um evento *advertising* o dispositivo central descobre o dispositivo periférico como na Figura 1.

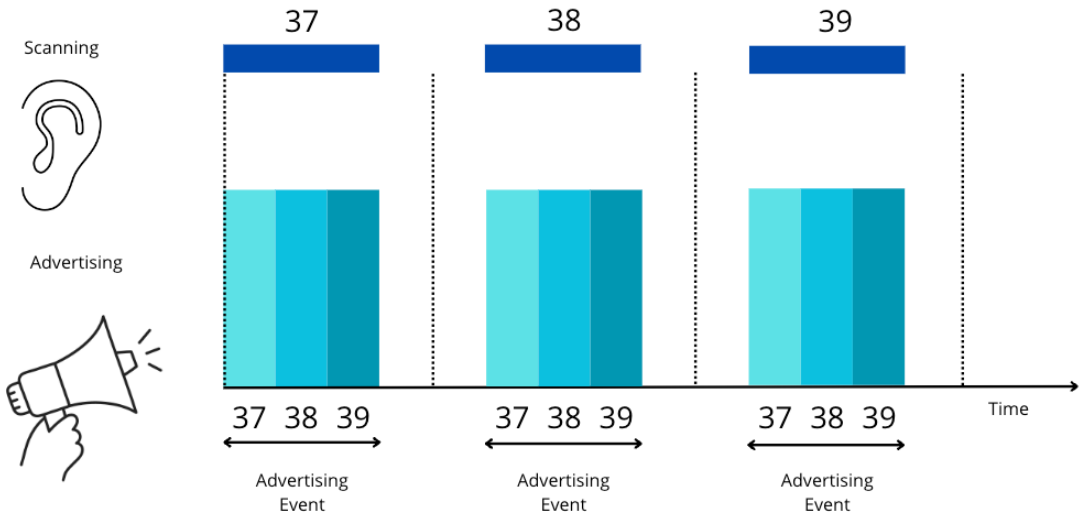

Figura 2 - Comunicação entre dispositivos Bluetooth 2 (Adptado de Lin et al., 2018)

O terceiro passo é quando um pacote de anúncio é detetado. Como tal, o próximo passo é a central iniciar uma conexão com o periférico enviando um pacote de solicitação de conexão. O dispositivo que suporta a função *GAP* central é responsável por gerir a conexão e tem a palavra final nos parâmetros de conexão. O dispositivo central decide sobre esses parâmetros (intervalo, latência, *channel map*) e agrupa-os num pacote de solicitação de conexão que envia ao periférico. Após tudo isto, a conexão está criada.

Posteriormente, o dispositivo central enviará um pacote de dados para o dispositivo periférico após um intervalo de tempo, conhecido como intervalo de conexão. Se o dispositivo periférico receber o pacote de dados e responder enviando um de volta para o dispositivo central, a conexão será considerada estabelecida como é possível verificar na Figura 3 (Lin et al., 2018).

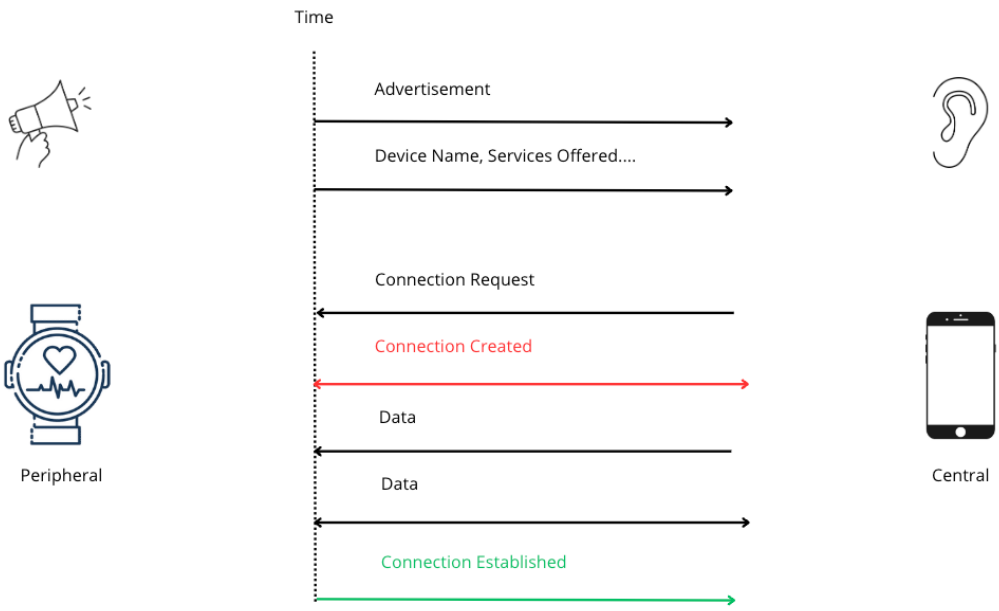

Figura 3 - Comunicação entre dispositivos BLE 3 (Adptado de Lin et al., 2018)

O quarto passo relaciona-se como é que os dispositivos comunicam depois de estabelecerem uma conexão. É aqui que o *ATT* e o *GATT* desempenham o seu papel (Lin et al., 2018).

O protocolo *ATT* define funções e cada papel vem com responsabilidades específicas. Os papéis definidos pela *ATT* são baseados numa arquitetura cliente-servidor. Assim, ambos os dispositivos numa conexão podem atuar como servidor ou cliente, dependendo do contexto dos dados que estão a ser transferidos. É responsabilidade do protocolo *ATT* fornecer meios para que o dispositivo servidor armazene dados num formato que possa ser lido e gravado pelo cliente, bem como fornecer mecanismos para que o cliente aceda, grave e leia esses dados (Lin et al., 2018).

O protocolo *ATT* define uma estrutura de dados chamada de atributo como um método padrão para organizar efetivamente os dados armazenados, acedidos e atualizados no servidor.

A estrutura de dados do atributo tem quatro campos (Lin et al., 2018), como é possível verificar na Figura 4:

- **Um identificador de atributo** – Este é um identificador exclusivo de 16 *bits* não assinado que é usado pelo cliente para referenciar um atributo no servidor. Ele torna o atributo endereçável e não muda durante uma única conexão.
- **Um tipo de atributo (UUID)** – Este é um identificador *UUID* globalmente exclusivo de 2 ou 16 *bytes* que define o tipo e o significado dos dados específicos armazenados no campo de valor do atributo. Existem dois tipos de *UUIDs*: *UUID* de serviço e *UUID* de característica.
- **Um valor de atributo** – Estes são os dados reais que estão a ser armazenados. Por exemplo, o valor de leitura do sensor de frequência cardíaca ou leitura de temperatura de um sensor de temperatura.
- **Um campo de permissões de atributo** – especifica os vários métodos que podem ser usados para aceder ao valor do atributo, bem como o nível de segurança necessário para aceder ao valor do atributo. Por exemplo, um atributo pode não exigir permissões para leitura, mas pode exigir autenticação para modificá-lo (gravá-lo).

Figura 4 - Campos ATT (Adptado de Lin et al., 2018)

O protocolo ATT também define métodos pelos quais os atributos podem ser lidos ou escritos. Os métodos dependem se é o cliente ou o servidor que inicia o procedimento de acesso ao atributo. No caso em que o cliente inicia o acesso ao atributo, são definidas duas operações: a operação de leitura e a operação de gravação (Lin et al., 2018).

A operação *Read* é usada pelo cliente para ler o valor de um atributo do servidor. O servidor responde com o valor do atributo. A operação *Write* é utilizada pelo cliente para escrever o valor de um atributo no servidor. O servidor responde com um status indicando se a gravação foi bem-sucedida ou não (Lin et al., 2018).

Por outro lado, no caso em que o servidor inicia o acesso, são definidas duas operações: operações de notificação e operações de indicação (Lin et al., 2018).

A operação notificação é utilizada pelo servidor para enviar um valor atualizado de um atributo ao cliente toda vez que ele mudar. O cliente não responde a esta operação.

A operação de indicação é semelhante à operação de notificação, mas o cliente deve enviar uma resposta com um *status* de reconhecimento indicando se recebeu ou não o valor corretamente (Lin et al., 2018). É possível observar em mais detalhe na Figura 5.

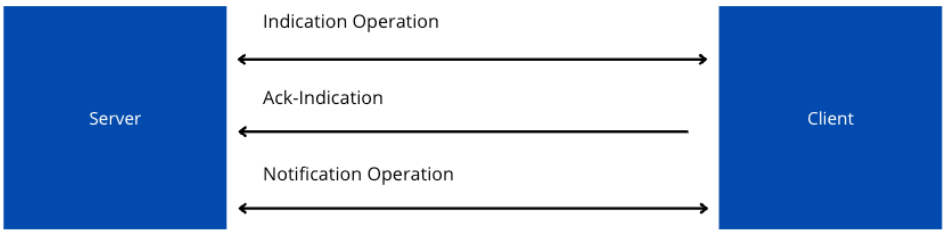

Figura 5 - Protocolo ATT (Adptado de Lin et al., 2018)

No que toca às permissões de acesso, estas determinam se um cliente pode ler ou gravar (ou ambos) um valor de um atributo (Lin et al., 2018). Cada atributo pode ser atribuído a um dos seguintes níveis de acesso:

- **None** – um cliente não pode ler nem gravar o atributo.
- **Readable** – um cliente pode ler o atributo.
- **Writable** – um cliente pode gravar o atributo.
- **Readable and writable** – o cliente pode ler e gravar o atributo.

Relativamente aos requisitos de segurança, os tipos de atributos e identificadores são informações públicas, mas o campo de valor e as permissões não são (Lin et al., 2018). O servidor pode exigir o seguinte:

- Autenticação para ler ou escrever;
- Autorização para ler ou escrever;
- Criptografia e emparelhamento para ler ou escrever.

Conforme declarado anteriormente, o protocolo *ATT* armazena os dados como atributos numa tabela, de maneira linear. A camada *GATT* é responsável por definir uma estrutura

hierárquica dos dados que demonstrem a relação entre os dados armazenados num servidor *ATT* (Lin et al., 2018).

A camada de protocolo *GATT* define uma estrutura na qual os recursos (dados) na base de dados de um servidor podem ser organizados para mostrar um relacionamento hierárquico (Lin et al., 2018).

A camada *GATT* define uma estrutura semelhante a uma árvore de 4 níveis (Lin et al., 2018) e é possível observar em mais detalhe na Figura 6:

- O nó raiz é chamado de perfil (nível 0);
- Os "filhos" do perfil são chamados de serviços (nível 1);
- Os "filhos" dos serviços são nomeados como características (nível 2);
- E as características podem ser definidas por um único valor e 0-n descritores (nível 3).

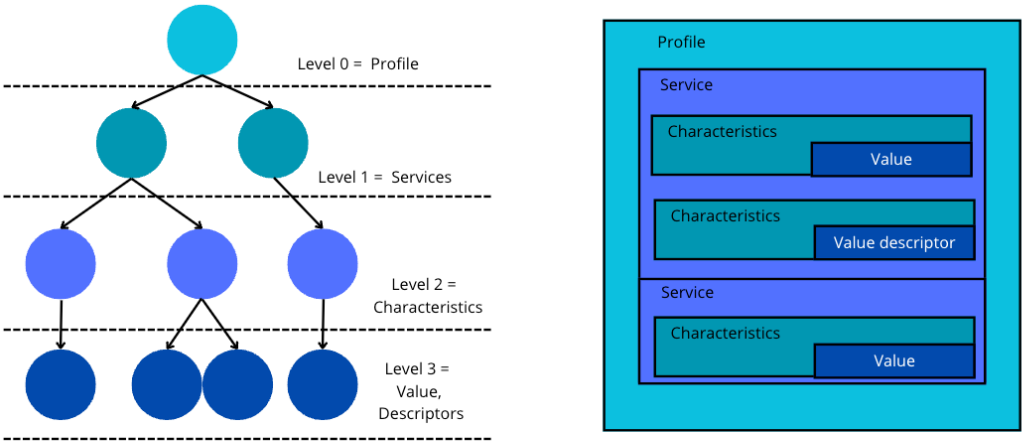

Figura 6 - GATT (Adptado de Lin et al., 2018)

Uma característica é uma unidade básica de armazenamento na qual os dados da aplicação são escritos e lidos. Cada característica é definida pelo valor e pelo descritor (opcional). O valor da característica são os dados reais armazenados pela aplicação. Um descritor é uma informação opcional que informa mais sobre o valor de uma característica (Lin et al., 2018).

Os serviços são usados principalmente para fins de organização, em vez de armazenamento.

Eles são análogos a pastas ou gavetas. No topo da hierarquia estão os perfis, que contêm serviços para um caso de uso específico suportado pelo servidor (Lin et al., 2018) como é possível ver na Figura 7.

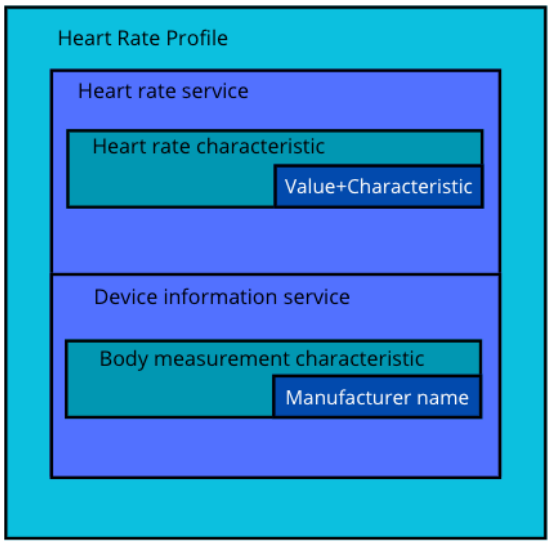

Figura 7 - Característica (Adptado de Lin et al., 2018)

O perfil *GATT* baseia-se na funcionalidade do protocolo *ATT*. A camada *GATT* fica acima da camada *ATT* (*Attribute Protocol*) na pilha de protocolos *BLE*. Ele reutiliza muitos dos conceitos e funcionalidades da camada *ATT* (Lin et al., 2018) onde é possível ver na Figura 8.

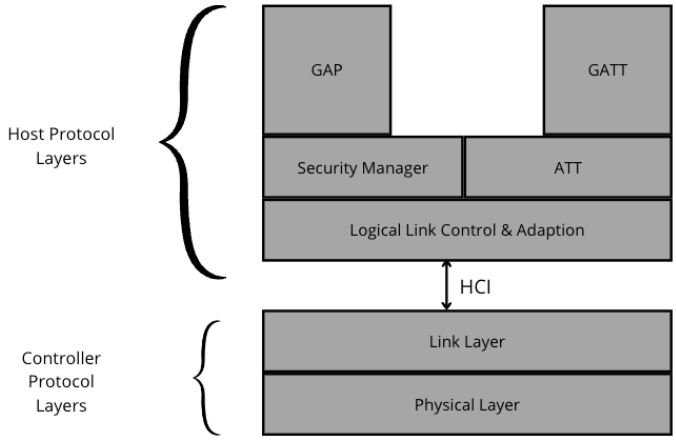

Figura 8 - Perfil GATT (Adptado de Lin et al., 2018)

A camada *GATT* usa a arquitetura cliente-servidor *ATT*. Em geral, a comunicação entre o cliente *GATT* e o servidor *GATT* começa com o cliente *GATT* realizando um serviço e descoberta de características para saber: quais são os serviços oferecidos pelo servidor *GATT* e quais as características estão associadas a cada serviço (Lin et al., 2018) onde é possível ver na Figura 9.

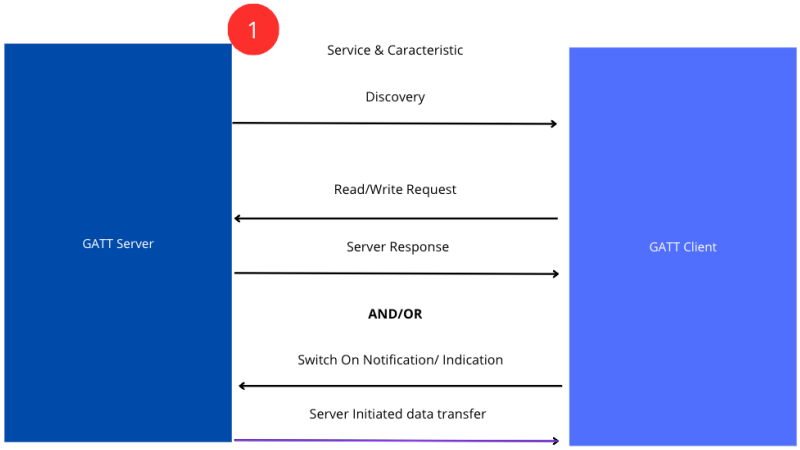

Figura 9 - Camada GATT (Adptado de Lin et al., 2018)

O *GATT* usa a estrutura de dados *attribute* para armazenar serviços e características que é possível ver na Figura 10.

Figura 10 - Estrutura de dados GATT (Adptado de Lin et al., 2018)

A camada *GATT* usa solicitações e respostas *ATT* para:

- Ler e gravar dados no servidor *GATT;*
- Descobrir todos os principais serviços oferecidos por um servidor *GATT;*
- Assinar ou cancelar a assinatura de notificações/indicações do servidor *GATT*.

O protocolo *ATT* é responsável por gerir o armazenamento de dados entre dispositivos. Ele fornece um meio para o dispositivo servidor armazenar dados num formato que o cliente possa ler e gravar, bem como fornecer mecanismos para o cliente aceder, gravar e ler esses dados (métodos de acesso e permissões) (Lin et al., 2018).

A camada *GATT* define uma estrutura hierárquica de dados que ajudam a demonstrar as conexões entre vários tipos de dados armazenados no servidor (perfil *GATT*) (Lin et al., 2018).

2.2.3. Camada de rede

Com um espaço de endereços vasto, o *IPv6* é, sem dúvida, o modo mais adequado para o *IoT*. No entanto, os tamanhos dos pacotes *IPv6* são muito grandes, por isso, os pacotes *IPv6* precisam ser fragmentados e compactados.

A utilização do *IPv6* é notável pelas suas características de endereçamento de 128 *bits*, resultando num cabeçalho combinado de origem e destino que totaliza 256 *bits*, o que pode suscitar desafios em termos de gestão e eficiência de alocação de endereços em redes de próxima geração, considerando os limites inerentes a tal extensão, sobrando apenas 2 ou 3 *bytes* para informação. Perante isto, foram feitas implementações para *IoT*, rede 802.15.4, (Lin et al., 2018):

- Compressão de cabeçalho - realizada mediante a aplicação de um mecanismo semelhante a um hash, empregando códigos de endereço predefinidos.
- Fragmentação de pacotes – os pacotes são fragmentados e no fim volta-se a montar no destino. Isto pode originar problemas de tráfego na rede. Além disso, se um pacote se perder, terá de ser enviado tudo novamente.
- Endereçamento especial de *mesh* – isto permite enviar informação pelos diversos vizinhos.

Foi então criado o *6LoWPAN* que consiste numa rede sem fios de baixa potência que permite usar *IPv6* nas redes *IEEE* 802.15.4. O 802.15.4 é um protocolo direcionado especialmente para a comunicação de sistemas com baixo poder computacional e com limitações energéticas (Lin et al., 2018).

Existem duas funções numa sub-rede *BLE*: *6LoWPAN Border Router* (*6LBR*) e *6LoWPAN Node* (*6LN*). O *6LBR* é localizado na borda de uma rede BLE e desempenha o papel de uma *gateway* entre a rede *BLE* e a *internetInternet*. Dentro da rede *BLE*, o *6LBR* é responsável por distribuir

o prefixo *IPv6* para os *6LNs*, que podem configurar automaticamente o seu endereço *IPv6* a partir do prefixo *IPv6* (Lin et al., 2018) como se verifica na Figura 11.

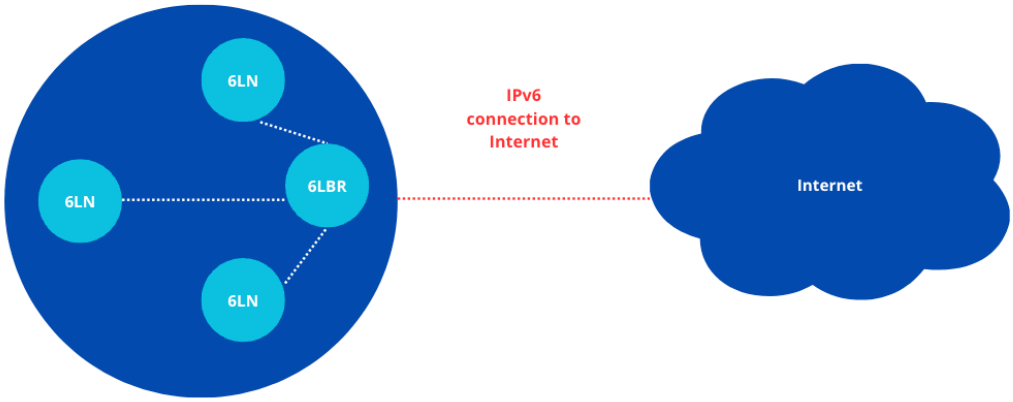

Figura 11 - 6LoWPAN (Adptado de Lin et al., 2018)

2.2.4. CoAP

O *CoAP* (Lin et al., 2018) é um protocolo de rede projetado para dispositivos e redes com recursos limitados, como dispositivos *IoT*. Ele fornece uma maneira eficiente de comunicação entre esses dispositivos, semelhante ao protocolo de comunicação usado para a transferência de dados na *Web*, mas otimizado para ambientes restritos em termos de largura de banda e energia. O *CoAP* possui um conjunto de subnormas. A norma do protocolo está especificada no *RFC 7252* (RFC 7252 - The Constrained Application Protocol (CoAP). Diversas extensões foram propostas, especialmente o *RFC 7641* (2015) que trata da Observação de Recursos no *CoAP*, o RFC 7959 (2016) que aborda as transferências em blocos no *CoAP*, o RFC 8323 (2018) que versa sobre o *CoAP* sobre *TCP*, *TLS* e *WebSockets*, e o RFC 8974 (2021) que lida com *Tokens Estendidos* e *Clientes Sem Estado* no *CoAP*. Comparado com o *HTTP*, o *CoAP* é executado sobre o *UDP* e tem um tamanho de cabeçalho menor e, por isso, é muito mais leve. Permite fazer manipulação simples de recursos e facilita a integração com a *Web*. Os recursos de baixa carga do *CoAP* tornam-no adequado para uso em dispositivos de rede limitados devido à grande

redução de sinalização. Acima do *UDP*, o *CoAP* define a camada de mensagens e a camada de *Request*/*Response.* (Lin et al., 2018).

O conceito básico de *CoAP* é muito semelhante ao conhecido e amplamente utilizado *REST* sobre o modelo *HTTP*. Um cliente *CoAP* pode aceder a recursos num servidor *CoAP* usando métodos como *GET*, *POST*, *PUT* ou *DELETE*. Como tal, o *CoAP* pode ser facilmente mapeado para *HTTP* e vice-versa. Desde a introdução do *CoAP*, foram publicadas muitas implementações do mesmo, para várias linguagens.

A grande vantagem do *CoAP* é a redução substancial de sinalização. Como complemento o *Content-Type* da mensagem pode utilizar *Etensible Markup Language* (*XML)*, *JavaScript Object Notation* (*JSON)* e *Concise Binary Object Representation (CBOR)*, sendo o último uma possível melhor opção no que toca a envio e carga. O *Content-Type* é um cabeçalho que especifica o formato dos dados no corpo de uma mensagem, sendo crucial para a interpretação adequada desses dados em protocolos como o *CoAP*.

O *CoAP* foi concebido para ser executado em dispositivos com muito pouco poder de computação, podendo ser implementado em sensores com apenas *10KiB* de *RAM* (Matthias Nefzger, 2021). A maioria das implementações do *CoAP* usa o *UDP* como um protocolo de transporte, pois adiciona apenas uma pequena sobrecarga em comparação, por exemplo, com o *TCP*. O *CoAP* foi projetado para ser uma boa combinação para casos de uso de *IoT*, pois tem uma sobrecarga muito menor em comparação com outros protocolos, como o *HTTP* (Matthias Nefzger, 2021). No entanto, o *CoAP* não foi projetado com *Bluetooth Low Energy* (*BLE*) em mente. Para implementar a configuração da aplicação para *Android*, Lin et al., 2018 teve de encontrar soluções para problemas como um tamanho limitado de pacote de *BLE* e limitações ao trabalhar com fluxos de dados (Matthias Nefzger, 2021). O *CoAP* é um protocolo especializado de transferência *web* para uso com nódulos restritos e redes restritas no *IoT*. O protocolo foi projetado para aplicações máquina a máquina - *M2M* (Kang et al., 2016). Existe uma diferença entre o uso de *IoT* e o *M2M*. O *IoT* tem obrigatoriamente uma *cloud* no meio e é constituído pelo sensor, *cloud* e a máquina (um carro, por exemplo). Por este motivo, necessita de rede *IP*. Por outro lado, o *M2M* é constituído pelo Sensor e Máquina, o que significa que não usa rede *IP*. O *M2M* permite que os dispositivos em rede troquem

informações e executem ações sem a *cloud* e de forma autónoma. Trata-se da comunicação feita entre uma máquina e outra (entre sensor e máquina).

Para entender as dificuldades de adaptar o *CoAP* ao *Bluetooth* é preciso entender alguns dos fundamentos do *BLE*. No entanto, é relevante notar que, embora o *BLE* possa parecer uma escolha lógica para transportar o *CoAP* sobre *Bluetooth*, não está amplamente disponível em dispositivos comuns para o utilizador final. O *Bluetooth Low Energy (BLE)* é frequentemente considerado a escolha natural para transportar o protocolo *CoAP* devido à sua notável eficiência energética, o que o torna ideal para dispositivos com recursos limitados, como sensores e dispositivos *IoT*. Além disso, o perfil de utilização do *BLE* e a sua capacidade de operar em curtas distâncias alinham-se bem com os requisitos comuns de comunicação em dispositivos *IoT*.

Na camada de mensagens, são trocados quatro tipos de mensagens, por *UDP*, entre os terminais: *Confirmable* (*CON*), *Non-Confirmable* (*NON*), *Acknowledgement* (*ACK*) e *Reset* (*RST*). A *RFC 7252*, intitulada "*The Constrained Application Protocol (CoAP)*" é a especificação fundamental para o *CoAP*, um protocolo projetado para dispositivos com recursos limitados, como sensores e dispositivos *IoT*. Esta *RFC* descreve como o *CoAP* opera sobre o protocolo *UDP* (*User Datagram Protocol*) e define as mensagens, operações e recursos do *CoAP*. Ela fornece orientações sobre como realizar comunicações eficientes em redes de dispositivos com recursos limitados, incluindo suporte para métodos *RESTful*, como *GET*, *PUT*, *POST* e *DELETE*, para permitir a interação com recursos no *IoT* (Lin et al., 2018).

Quando um *endpoint* recebe uma mensagem *Confirmable*, deve reconhecer a mensagem *CON* com uma mensagem *ACK*, o que garante uma transmissão confiável em *CoAP*. Nalguns casos as mensagens podem ser transmitidas de forma menos confiável, marcando-as como *Non-Confirmable*. Na camada *Request*/*Response*, o *CoAP* usa arquitetura cliente/servidor e os recursos disponíveis do servidor são identificados por *URIs* (Lin et al., 2018). Os clientes *CoAP* acedem aos recursos do servidor usando métodos como *GET*, *PUT*, *POST* e *DELETE.*

Certamente, o *CoAP* pode ser transportado com *JSON* ou *CBOR*, fornecendo flexibilidade na representação dos dados para a comunicação em dispositivos com recursos limitados. A escolha entre *JSON* e *CBOR* depende das necessidades específicas da aplicação e dos recursos disponíveis nos dispositivos. O *JSON* é mais legível e fácil de depurar, mas pode ser menos eficiente em termos de tamanho de dados. Por outro lado, o *CBOR* é mais compacto e eficiente em termos de uso de largura de banda, o que é especialmente importante em dispositivos com recursos limitados.

2.2.5. 6LowPan

O *6LoWPAN* é uma rede sem fios de baixa potência que permite usar *IPv6* em redes 802.15.4. O Protocolo 802.15. 4 é direcionado especialmente para a comunicação de sistemas com baixo poder computacional e com limitações energéticas. Não foi propriamente concebido para trabalhar no ramo do *IoT*. Perante isto foram feitas implementações para *IoT* (rede 802.15.4). Neste protocolo é feito uso de três principais pontos: compressão de cabeçalho, fragmentação de pacotes e endereçamento especial de *mesh*. A compressão de cabeçalho no protocolo *CoAP* é realizada através do uso de códigos de endereços predefinidos, que são representados por meio de um *hash* para reduzir o tamanho dos cabeçalhos das mensagens. A fragmentação de pacotes é o processo de dividir mensagens em fragmentos menores para facilitar a transmissão em redes que impõem limites de tamanho máximo para os pacotes. No entanto pode implicar problema de tráfego na rede e, caso um pacote se perca, terá de ser enviado o pacote novamente. O endereçamento especial de *mesh* permite que dispositivos desempenhem funções específicas, como encaminhar dados entre vizinhos, facilitando a comunicação eficiente na rede. A grande vantagem é o uso na indústria, automação, *smart home* e como desvantagem não permite o envio de informação demasiado extensa visto que a bateria é o mais importante neste setor (Matthias Nefzger, 2021).

Existem, portanto, dois desafios grandes na aplicação do *CoAP* sobre *BLE*. O primeiro desafio é o tamanho do pacote, isto porque um pacote *UDP* pode ter um tamanho de até 65.535

bytes. Por outro lado, o tamanho máximo de um pacote *ATT* é de 512 bytes. Os pacotes *UDP* podem ser muito maiores que os pacotes *GATT*. Se fosse necessário enviar ou receber uma mensagem *CoAP* com uma carga útil superior a 512 *bytes*, seria necessário dividir manualmente a mensagem e cuidar da transmissão de várias partes. Para mapear as solicitações do *CoAP* para as características do *GATT* existe uma solução que contorna essas questões. Em vez de se implementar uma lógica complexa do lado da aplicação, usa-se um tipo especial de serviço *GATT* chamado serviço *UART*. O segundo desafio consiste em delimitar pacotes numa *stream*. Ao usar a interface serial tipo fluxo do serviço *UART*, exita-se muito trabalho na lógica do lado da aplicação. No entanto, com esta decisão, surge um novo desafio: que é em como o recetor do fluxo pode reconstruir as mensagens *CoAP* contidas. Afinal, são enviados pequenos pedaços de mensagens num fluxo de *bytes*, portanto, uma camada abaixo da abstração real do *CoAP*. Aplica-se o *Serial Line Internet Protocol* (ou *SLIP*) antes de gravar os *bytes* resultantes no fluxo *UART*. O algoritmo é extremamente simples: anexar um *byte END* especial (*0xC0*) a cada mensagem permite que o recetor separe os *bytes* de entrada procurando a ocorrência desse *byte* especial. Um *byte END* especial delimita mensagens únicas conforme na Figura 12 (Matthias Nefzger, 2021).

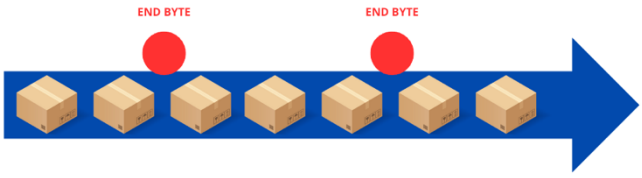

Figura 12 - Delimitar pacotes numa stream (Adptado de Matthias Nefzger, 2021)

Conforme a Figura 13, a implementação da solução do artigo (Matthias Nefzger, 2021) começou por ser executada da seguinte forma. Em primeiro lugar, foi decidido usar uma implementação de serviço *UART* para enviar e receber *bytes* entre os dois dispositivos. Isso contorna a questão de como mapear os terminais *CoAP* para características específicas do *BLE*

e permite enviar mensagens *CoAP* de tamanho arbitrário. Para delimitar mensagens *CoAP* únicas no fluxo *UART*, foi aplicado o algoritmo *SLIP* que anexa um *byte END* especial a cada mensagem. Da perspetiva da aplicação android, cada mensagem passa pelas mesmas etapas: primeiro, uma mensagem de solicitação *CoAP* é construída com os cabeçalhos, opções, carga útil e *token* necessários. O *token* é salvo em conjunto com a *RequestType* para saber como analisar a carga útil das mensagens *CoAP* recebidas (Matthias Nefzger, 2021).

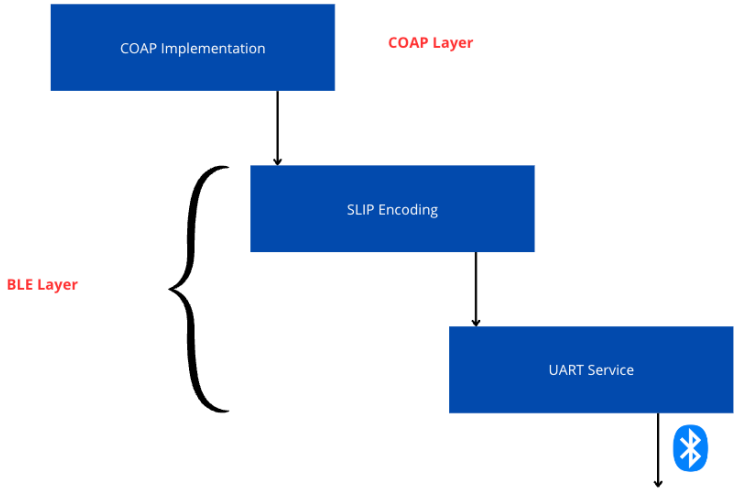

Figura 13 - CoAP sobre Bluetooth Low Energy, separado em camada CoAP e BLE (Adptado de Matthias Nefzger, 2021)

O objecto de mensagem *CoAP* é então convertido num *byte array*, que é codificado pelo algoritmo *SLIP*. O *byte array* resultante é escrito na característica *UART Rx* do dispositivo remoto. O *android* encarrega-se de dividir a mensagem em pedaços mais pequenos. Ao receber novos dados, pequenos pedaços de *bytes* chegam à aplicação no subscritor da característica *Tx* do dispositivo remoto (em *onReceive()*). Os novos dados são anexados a um *buffer* interno. Com cada nova atualização, o algoritmo *SLIP* procura a ocorrência do *byte END* nesta memória intermédia e divide o fluxo em mensagens completas. Como último passo, são limpas as mensagens completas do *buffer* e mantem-se apenas os *bytes* restantes, ainda incompletos (Matthias Nefzger, 2021).

Cada mensagem completa é analisada como uma mensagem *CoAP*. A aplicação verifica então o *token CoAP* recebido para determinar se a mensagem recebida é uma resposta esperada. Se esta verificação for bem sucedida, o *payload* é processado pelos componentes comerciais da aplicação em *handleExpectedMessage()*. Desde esta implementação inicial do *CoAP* sobre *BLE*, em Março de 2021, a solução provou ser fácil de manter e alargar. Embora os programadores precisem inicialmente de aprender a trabalhar com o *CoAP*, isso é rapidamente conseguido devido à semelhança com o *REST* sobre *HTTP*. Confiar num protocolo padrão estabelecido como o *CoAP* tem muitas vantagens em relação a desenvolver o seu próprio protocolo de comunicação *BLE* (Matthias Nefzger, 2021).

O conceito de *IoT* está firmemente associado ao *IPv6*, que permite atribuir endereços exclusivos (geralmente *IPv6*) a cada um desses dispositivos, facilitando assim sua comunicação e conectividade na Internet. No entanto, no mundo real, nem todos os dispositivos usam *IPv6*. Por exemplo, *BLE* usa o *MAC* que é um endereço para identificar dispositivos. Para unificar a comunicação entre dispositivos não *IP* e baseados em *IP*, é necessária uma camada de *software* identificador.

2.2.6. Protocolos de envio de mensagens

O MQTT é um protocolo de mensagens leve e baseado em publicação/assinatura para uso sobre *TCP/IP*. Com o modelo de publicação/assinatura, os editores e assinantes não precisam saber da existência um do outro. Em vez disso, todos os dados publicados pelo publicador serão coletados no intermediário de mensagem e, então, é responsabilidade do intermediário de mensagem enviar os dados aos assinantes (Lin et al., 2018).

Em 2013 (Lin et al., 2018), a *IBM* propôs o *MQTT-SN* (Stanford-Clark & Truong, 2013), onde o termo *SN* significa *Sensor Networks*. O *MQTT-SN* foi projetado para ser usado em ambientes de comunicação sem fio com baixa largura de banda, altas falhas e mensagem curtas. Também recomendado para dispositivos de baixo custo e recursos limitados. Embora o *MQTT-SN* tenha sido originalmente desenvolvido para ser executado sobre o *ZigBee*, a especificação refere

que ele é independente dos serviços de rede. Portanto, em dispositivos com recursos limitados, o *MQTT-SN* pode ser executado sobre *UDP* em vez de *TCP* (Lin et al., 2018).

A figura mostra o modelo de publicação e assinatura de *MQTT* e *MQTT-SN*. O broker *MQTT* é o principal responsável por receber todas as mensagens, filtrá-las, decidir quem tem acesso aos dados e transmitir mensagens aos assinantes legais.

Os assinantes desempenham um papel passivo no sistema. Mais especificamente, os assinantes não precisam de consultar se há novos dados ou não; eles simplesmente esperam que novos dados sejam enviados para eles. Dessa forma, tanto os editores quanto os assinantes podem ter um ciclo de trabalho baixo, o que significa que o consumo de energia pode ser bastante reduzido (Lin et al., 2018). É possível verificar o indicado na Figura 14

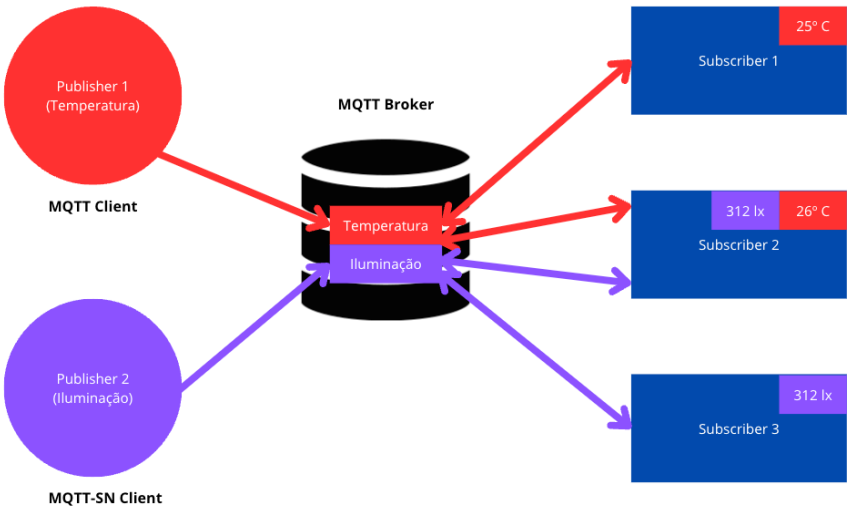

Figura 14 - MQTT Broker (Adptado de Lin et al., 2018)

O *MQTT-SN* usa o ID do tópico em vez do nome do tópico como acontece no *MQTT*. Além disso o *MQTT-SN* pode ser usado em mais protocolos adequados para redes de sensores como *ZigBee*, *Z-Wave* e assim por diante visto que não requer pilha *TCP/IP*. Já o *MQTT* depende de uma camada de transporte que entregue uma sequência ordenada e sem perda de pacotes,

como *TCP* por exemplo, para o seu funcionamento (Lin et al., 2018). É possível verificar o indicado, em forma de tabela de comparação, na Figura 15.

	MQTT	**MQTT-SN**
Comunicação	TCP-IP	UDP, não IP
Rede	Ethernet, WIFI, 3G	Zigbee, Bluetooth, RF, etc
Tamanho mínimo de mensagem	2 bytes – PING	1 byte
Tamanho máximo de mensagem	<= 24 MB	< 128 bytes
Alimentação por bateria		SIM
Clientes em modo Sleep		SIM
Connectionless mode (QoS -1)		SIM

Figura 15 - MQTT vs MQTT-SN (Lin et al., 2018)

Por outro lado, o AMQP (Advanced Message Queuing Protocol), segundo Uy & Nam, 2019) segue um modelo de fila de mensagens. As mensagens são publicadas em filas e são consumidas por aplicações que se ligam a essas filas. Cada mensagem é entregue a um único consumidor, o que é importante em cenários empresariais onde a fiabilidade é fundamental, como na integração de sistemas e processamento de mensagens complexas.

Enquanto, como visto anteriormente, o *MQTT* utiliza um modelo de publicação/subscrição, em que os dispositivos enviam mensagens para tópicos e os dispositivos interessados inscrevem-se nesses tópicos para receber as mensagens. Todas as mensagens são entregues a todos os subscritores no tópico. Este modelo é adequado para cenários de *IoT* e comunicações em tempo real, onde a escalabilidade e a baixa latência são essenciais.

No que diz respeito aos protocolos de rede, o *MQTT* é projetado para funcionar sobre *TCP/IP*, tornando-o adequado para redes *IP*, como a *Internet*. Por outro lado, o *AMQP* é mais flexível em relação às camadas de transporte e suporta diferentes protocolos, incluindo *TCP/IP*.

Em termos de flexibilidade e recursos, o *MQTT* é mais simples e direto, sendo adequado para dispositivos com recursos limitados e para aplicações que requerem baixa latência em tempo real. O *AMQP* é mais rico em recursos e oferece funcionalidades avançadas, como gestão sofisticada de filas, confirmação de entrega, tópicos avançados e segurança robusta.

A arquitetura é composta por 3 tipos de componentes:

- *Clients*;
- *Gateways*;
- *Forwarders*.

Os clientes *MQTT* são conectados ao *broker MQTT* via *gateways MQTT* usando o protocolo *MQTT-SN*. Há também casos onde o cliente acede ao *broker* através de *forwarders*, nesse caso o *forwarder* apenas encapsula os pacotes *MQTT-SN* e envia para o *Gateway MQTT-SN* (Lin et al., 2018), como se verifica na Figura 16.

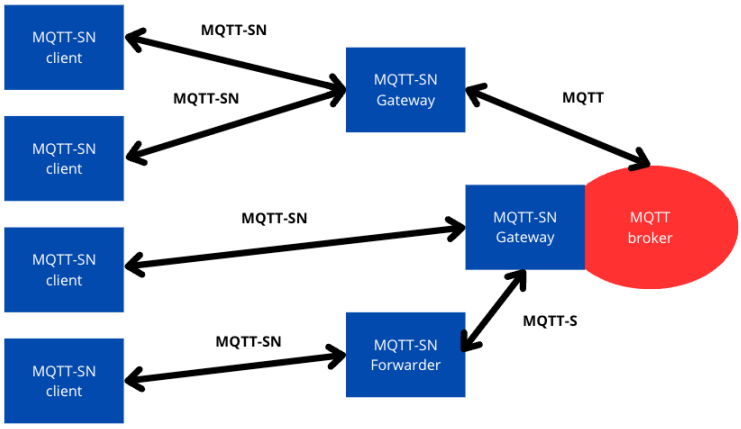

Figura 16 - Arquitetura (Adptado de Lin et al., 2018)

2.3. Bases de dados não relacionais

As bases de dados não relacionais têm vindo a ganhar popularidade devido à necessidade de lidar com grandes volumes de dados não estruturados. Uma das opções mais utilizadas é o *MongoDB*, um sistema não relacional. O *MongoDB* utiliza o modelo de documentos, onde os dados são organizados em documentos *JSON*. Esta estrutura flexível permite uma fácil representação e manipulação de dados complexos, especialmente relevante no *IoT*.

Uma das principais vantagens do *MongoDB* é a sua escalabilidade horizontal. Ao distribuir os dados em vários servidores, é possível aumentar o desempenho do sistema de forma linear à medida que se adicionam mais servidores. Esta escalabilidade é crucial para lidar com o crescimento contínuo dos dados gerados pela *IoT*.

O *MongoDB* também suporta consultas *ad hoc* e oferece indexação flexível, o que permite recuperar os dados de forma rápida e eficiente. Esta característica é especialmente útil em ambientes *IoT*, onde os dados podem ter estruturas e semânticas variadas. Além disso, o *MongoDB* tem mecanismos de tolerância a falhas e recuperação automática. Em caso de falha de um servidor, o sistema é capaz de se recuperar automaticamente, garantindo a disponibilidade contínua dos dados. Por fim, o *MongoDB* oferece recursos avançados, como suporte a transações distribuídas, replicação assíncrona e particionamento automático. Estes recursos adicionais contribuem para a robustez e escalabilidade do sistema, tornando-o uma escolha adequada para gerir dados em ambientes complexos de *IoT*.

Por tudo isso, o *MongoDB* é uma opção vantajosa para bases de dados não relacionais, devido à sua escalabilidade, flexibilidade, recuperação automática e recursos avançados. (Jatana et al., 2012)

Em *MongoDB*, um *cluster* é um conjunto de um ou mais servidores que armazenam dados. O *cluster* é responsável por garantir a alta disponibilidade, escalabilidade e confiabilidade do banco de dados *MongoDB*. Existem três tipos de *clusters*:

- *Replica Set*: um grupo de instâncias de servidor que contêm o mesmo conjunto de dados. O conjunto de dados é replicado em todas as instâncias do replica *set*, de forma que, em caso de falha de uma instância, outra possa assumir a liderança.
- *Sharded Cluster*: um conjunto de replica *sets* onde os dados são particionados e distribuídos em diferentes *shards*, de forma que o acesso aos dados seja balanceado entre diferentes instâncias do *cluster*.
- *Config Server Cluster*: um conjunto de instâncias de servidor que armazenam os metadados do *sharded cluster*, como as configurações de balanceamento de carga e de particionamento de dados.

2.4. Trabalhos relacionados

Nesta secção, serão apresentadas a arquitetura de rede e a arquitetura da aplicação de diversos trabalhos relacionados. Isso permitirá uma compreensão mais aprofundada das abordagens utilizadas em estudos anteriores, estabelecendo uma base sólida para o desenvolvimento do projeto.

No sistema experimental desenvolvido por Lin et al. (2018) a arquitetura consiste em duas sub-redes *BLE* conforme a Figura 17.

As placas *6LNs* são conectadas aos sensores ambientais (sensores de temperatura, sensores de humidade e sensores de luz ambiente) (Lin et al., 2018). A razão pela qual foi escolhido o *Raspberry Pi 3* e *Nordic nRF51-DK,* como nós *BLE,* deve-se ao suporte integrado de *Bluetooth* 4.0 e o módulo *6LoWPAN*. *Raspbian*, o sistema operativo oficial baseado em *Linux* para *Raspberry Pi*, contém vários módulos úteis que ajudam a construir as funcionalidades desejadas (Lin et al., 2018). As redes *BLE* são conectadas à *internet* por meio de um *router*. Também foi configurado um servidor *web* e um servidor de base de dados. Além disso, foi implementada uma base de dados em *cloud* como *backup* dos dados ambientais, que também podem ser acedidos pelos clientes (Lin et al., 2018).

Os prefixos de rede 2001:288:d003:a000::/64 e 2001:288:d003:a010::/64 foram os escolhidos para serem usados dentro das duas sub-redes *BLE*, respetivamente.

Em cada sub-rede *BLE*, o *6LBR* transmite o prefixo para os *hosts* da rede para que eles possam gerar seus próprios endereços usando a configuração automática de endereço sem estado especificada na *RFC 4862*. Quanto às interfaces *Ethernet* nos *6LBRs* que servem como lado *WAN*, elas estão na sub-rede *IPv6* com o prefixo de rede 2001:288:d003:1127::/64. Cada uma enfrentando duas sub-redes *IPv6* diferentes. Os *6LBRs* desempenham o papel de uma *gateway* entre a sub-rede *BLE* e a *internet* (Lin et al., 2018).

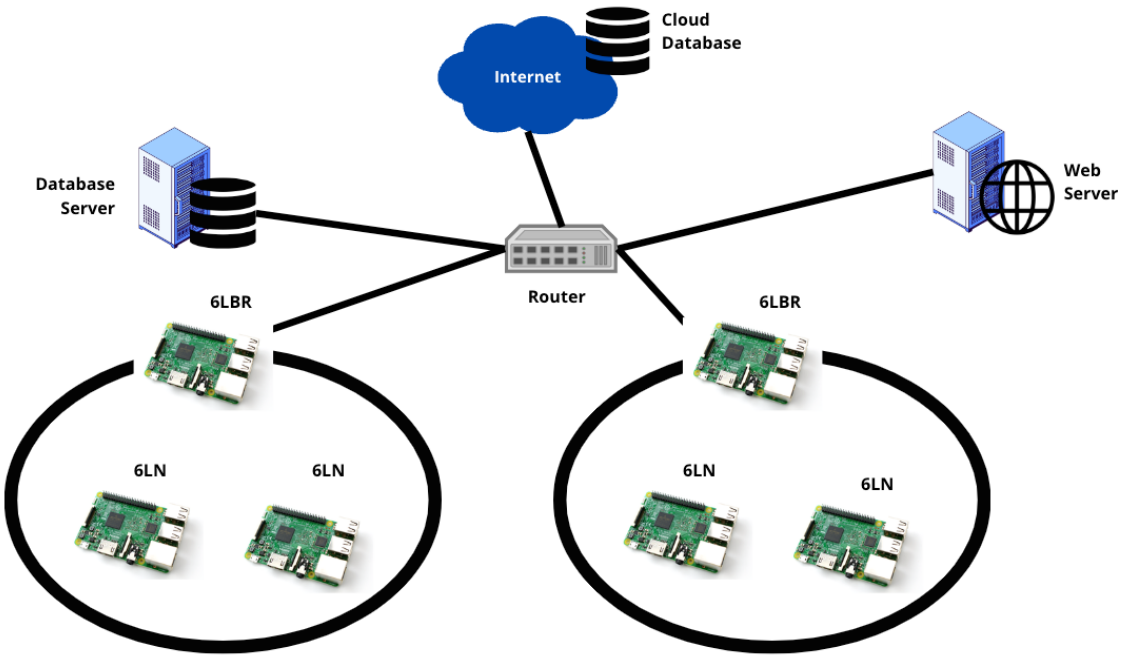

Figura 17 - Arquitetura artigo (Adptado de Lin et al., 2018)

Neste contexto, é descrito o procedimento pelo qual os *6LNs* se conectam ao *6LBR* e finalizam as suas configurações de *IP* (Lin et al., 2018). Esse processo é essencialmente composto por três etapas:

Etapa 1: Estabelecer a conexão da camada de ligação.

Etapa 2: Estabelecer o canal *L2CAP*.

Etapa 3: Configurar o endereço *IPv6*.

Quando os dispositivos *BLE* concluírem a configuração *IP* estarão prontos para executar aplicações baseadas em *IP* (*HTTP*, *SSH*, *FTP*, ...). No entanto, considerando que os nós *IoT* são limitados por recursos, foram escolhidos o *MQTT-SN* e *CoAP* como os protocolos da camada de aplicação (Lin et al., 2018), como é possível visualizar na Figura 18.

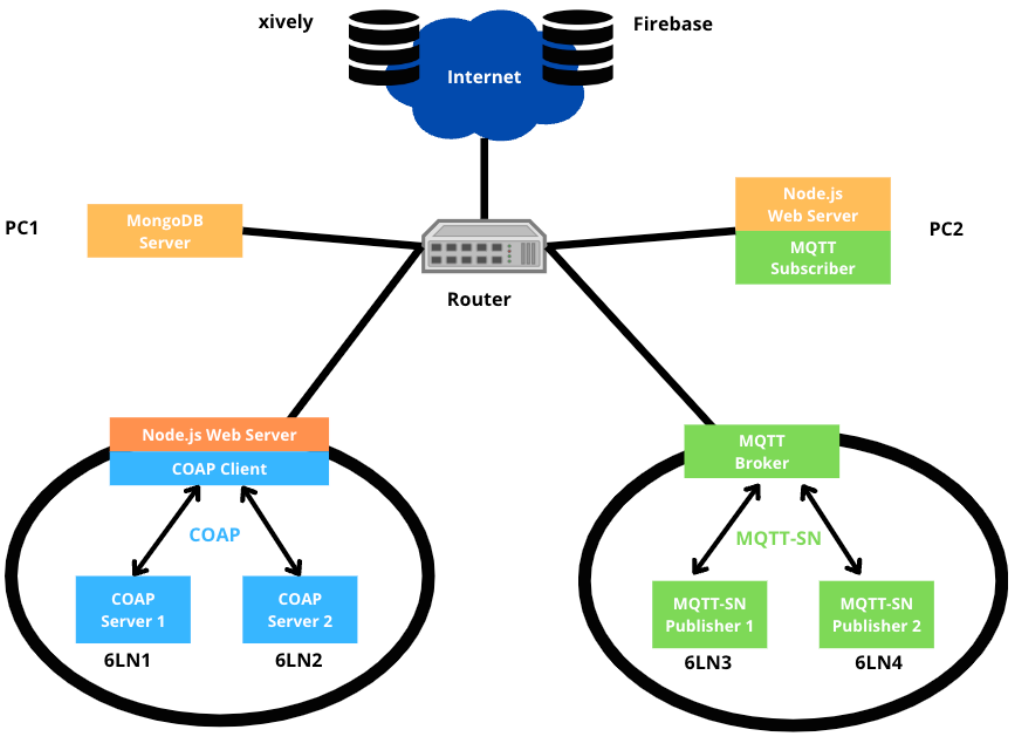

Figura 18 - Arquitetura da aplicação (Adptado de Lin et al., 2018)

As duas sub-redes *BLE* executam diferentes protocolos. Na sub-rede *BLE* 1, os dois *6LNs* desempenham o papel de servidores *CoAP*, enquanto o *6LBR1* desempenha o papel de cliente *CoAP* e também de servidor *web*. Especificamente, tornou-se *6LBR1* um dispositivo de *gateway* que aceita *HTTP* de utilizadores remotos com navegadores *Web* comuns.

Se um utilizador remoto clicar num botão para acender uma luz na sub-rede *BLE* 1, o *6LBR1* emitirá um pedido de *CoAP* para a *6LN* que se liga à luz. Depois de ligar a luz, a *6LN* responde com uma resposta *CoAP* que indica que o último estado da luz é ligado. Quando a resposta do *CoAP* é recebida em *6LBR1*, o estado mais recente é atualizado no navegador da Web através de uma resposta *HTTP* (Lin et al., 2018).

Na sub-rede *BLE* 2, os dois *6LNs* desempenham o papel de editores, enquanto o *6LBR* desempenha o papel de corretor. As trocas de mensagens entre os publicadores e o *broker* são baseadas no protocolo *MQTT-SN*. O *PC2* desempenha o papel de assinante e também de servidor da *web*, que se comunica com o corretor por meio do protocolo *MQTT*.

Especificamente, o servidor da *web* utiliza uma biblioteca cliente *MQTT* para assinar os dados do sensor. Os dados do sensor em tempo real são exibidos no *frontend* e, em seguida, enviados para a base de dados na *Cloud Firebase* (Lin et al., 2018).

No artigo "*Performance Evaluation of a Smart CoAP Gateway for Remote Home Safety Services*", o autor (Kim et al., 2015) propõe uma *gateway* inteligente baseada no protocolo *CoAP* para fornecer serviços de segurança em casa que permitem a monitorização remota de intrusões, incêndios e da qualidade do ar interior. Esta gateway *CoAP* inteligente controla um nó sensor de segurança residencial equipado com um sensor de movimento infravermelho piroelétrico, um sensor de incêndio, um sensor de temperatura e humidade e um sensor de CO2 infravermelho não dispersivo (*NDIR*), recolhendo dados dos sensores. Além disso, é capaz de converter dados de sensores físicos em informações compreensíveis e atuar como um *router* de fronteira para possibilitar uma ligação contínua entre uma rede pessoal sem fio de baixa potência (*6LoWPAN*) e a Internet (*IPv6*). Os resultados dos testes laboratoriais demonstram que o *gateway CoAP* inteligente é viável e fornece uma taxa de transferência de dados de cerca de 97,20%.

O artigo aborda a importância de garantir a segurança em ambientes residenciais, especialmente para grupos vulneráveis, como idosos, donas de casa e bebés, que passam a maior parte do tempo em ambientes fechados. Também menciona os riscos associados à qualidade do ar interior devido a materiais de construção que emitem contaminantes. Além disso, discute a necessidade de soluções técnicas para abordar esses problemas de segurança em casa, destacando o potencial das tecnologias *M2M* (*Machine-to-Machine*) e *IoT* (Internet das Coisas), com foco no protocolo *CoAP*.

O artigo apresenta uma visão geral da arquitetura e dos componentes do sistema, incluindo o nó sensor de segurança residencial e a *gateway CoAP*. O nó sensor é projetado para acomodar vários sensores, como deteção de movimento, deteção de incêndio, medição de temperatura e humidade e medição de CO2. A *gateway CoAP* oferece conectividade para dispositivos inteligentes e nós (sensores) por meio de várias tecnologias, como *Wi-Fi*, *Bluetooth* e *Zigbee*.

O protocolo *CoAP* é discutido em detalhes, destacando as suas semelhanças com o *HTTP*, mas otimizado para dispositivos com recursos limitados e redes restritas. Ele também suporta a arquitetura *REST* (*Representational State Transfer*) para aplicações *M2M* e *IoT*.

O artigo conclui enfatizando a capacidade da *gateway CoAP* inteligente em fornecer serviços de segurança residencial controláveis e monitorizáveis remotamente. O autor destaca a capacidade de converter dados de sensores físicos em informações compreensíveis e a ligação contínua entre redes de baixa potência e a *Internet*. Além disso, menciona resultados de testes laboratoriais que demonstram um bom desempenho da *gateway*.

Em resumo, o artigo apresenta uma solução prática e inovadora para serviços de segurança residencial, utilizando o protocolo *CoAP*, e destaca a importância da segurança residencial, especialmente em ambientes fechados.

No artigo "*Performance Analysis of an LTE Gateway for the IoT*" o autor Costantino et al., 2012) explora a utilização do *LTE* como uma opção para ligar *gateways IoT* à internet. O estudo baseia-se no uso do *CoAP* e recorre a simulações no *software* de código aberto *ns3* para avaliar o desempenho.

Nas primeiras partes, o artigo introduz o contexto do *IoT* e a importância das *gateways* na ligação de dispositivos *IoT* à *internet*. O *LTE* é considerado como uma solução viável devido às suas vantagens em termos de eficiência espectral, largura de banda e cobertura.

O artigo destaca que, embora o *LTE* tenha sido projetado para comunicações móveis, o seu uso para *IoT* tem desafios, como a eficiência na comunicação *M2M*.

O modelo de sistema é descrito, incluindo o uso do *CoAP* como protocolo de camada de sessão. O *CoAP* é introduzido como um protocolo leve de cliente/servidor que opera sobre o *UDP*.

As simulações realizadas no *ns3* mostram que, em condições de carga leve, o tráfego *M2M* é pequeno em comparação com o tráfego total. No entanto, em situações de carga mais alta, o desempenho do *LTE* é afetado, resultando em atrasos mais elevados e perda de pacotes. Essa perda de pacotes ocorre devido à interferência intra-*gateway*.

O tamanho dos pacotes *CoAP* também é avaliado nas simulações. Conclui-se que, para níveis baixos de probabilidade de falha, tamanhos maiores de pacotes *CoAP* afetam significativamente o desempenho, o que pode ser crítico para aplicações sensíveis à perda.

A conclusão do artigo destaca a importância de mecanismos de isolamento para proteger as comunicações *M2M* de interrupções causadas por interferência.

Em resumo, o artigo fornece informações sobre o desempenho do *LTE* como tecnologia para *gateways IoT*, destacando desafios e a necessidade de soluções para garantir a confiabilidade das aplicações *IoT*.

No artigo "*Performance Evaluation of MQTT and CoAP via a Common Middleware*" o autor Thangavel et al., 2014 aborda a importância das redes de sensores sem fio (*RSSFs*) na monitorização de parâmetros ambientais em ambientes urbanos e hostis. Nas *RSSFs*, os dispositivos são frequentemente limitados em termos de recursos, o que torna essencial a utilização de protocolos de aplicação eficientes em termos de largura de banda e energia para a transmissão de dados.

O artigo apresenta dois protocolos essenciais para dispositivos com recursos limitados em *RSSFs*: o *MQTT* (Message Queue Telemetry Transport) e o *CoAP* (Constrained Application Protocol). Ambos os protocolos baseiam-se na arquitetura "publicar-subescrever," permitindo a comunicação entre os nós (sensores) e as *gateways*. O *MQTT* utiliza uma arquitetura baseada em tópicos, enquanto o *CoAP* é fundamentado na arquitetura *Representational State Transfer (REST)* e suporta tanto o modelo de pedido-resposta como a arquitetura de "publicar-subescrever."

Para avaliar o desempenho destes protocolos, o autor desenvolveu um *middleware* comum que oferece suporte tanto ao *MQTT* como ao *CoAP*, proporcionando uma interface de

programação comum. Este *middleware* foi projetado para ser extensível, permitindo a incorporação de protocolos adicionais no futuro. Uma das principais vantagens deste *middleware* é a capacidade de selecionar o protocolo mais adequado com base nas condições da rede, o que contribui para otimizar o desempenho da mesma.

As experiências realizadas no artigo examinaram o desempenho do *MQTT* e do *CoAP* em termos de atraso e consumo de largura de banda. Os resultados indicaram que o *MQTT* tende a apresentar menor atraso do que o *CoAP* em taxas de perda de pacotes mais baixas. No entanto, à medida que a taxa de perda de pacotes aumenta, o *CoAP* supera o *MQTT* em termos de atraso devido aos menores *overheads* associados ao *CoAP*. Além disso, quando o tamanho da mensagem é pequeno e a taxa de perda de pacotes é igual ou inferior a 25%, o *CoAP* gera menos tráfego adicional do que o *MQTT* para garantir a confiabilidade da mensagem.

Em resumo, o artigo oferece uma visão abrangente sobre o desempenho do *MQTT* e do *CoAP* em *RSSFs*, destacando a importância de um *middleware* comum para escolher o protocolo adequado com base nas condições da rede. Esta abordagem pode ser valiosa para melhorar o desempenho das redes de sensores sem fios em diferentes cenários e aplicações.

No artigo "*The CoAP-based M2M Gateway for Distribution Automation System Using DNP3.0 in Smart Grid Environment*" o autor Shin et al., 2016 discute a integração de tecnologias de comunicação no contexto de uma rede elétrica inteligente (*Smart Grid*). O foco está na utilização do Protocolo de Rede Distribuída 3.0 (*DNP3.0*) e do *CoAP* para melhorar a comunicação *M2M* e a interoperabilidade.

O artigo começa por introduzir o conceito de rede inteligente, destacando a necessidade de tecnologias avançadas para melhorar a fiabilidade e eficiência da rede elétrica. O *DNP3.0* é descrito como o protocolo de comunicação mais importante para sistemas de energia.

É discutido o surgimento da comunicação *M2M*, também conhecida como comunicação de *D2D*. Está relacionada com o *IoT* e é essencial para ligar sistemas de energia à Internet.

O artigo destaca as limitações do *Simple Object Access Protocol* (*SOAP*) para comunicação *M2M* devido às mensagens pesadas e estruturas complexas. Apresenta o *CoAP* como uma

alternativa com base nos princípios de *Representational State Transfer* (*REST*). O *CoAP* é leve, adequado para dispositivos e redes limitados.

O foco principal do artigo é a proposta de uma *Gateway M2M* baseada em *CoAP* para um sistema de automação de distribuição usando *DNP3.0* num ambiente de rede inteligente. Esta *gateway* atua como uma ponte entre o *CoAP* e o *DNP3.0*, permitindo uma comunicação eficiente.

O artigo explica como os dados do *DNP3.0* são mapeados para o *CoAP* e vice-versa. Este processo de mapeamento garante que as informações do *DNP3.0* podem ser facilmente acedidas e controladas através do *CoAP*.

O autor conduziu uma avaliação de desempenho do *CoAP* e do *SOAP* usando o *OPNET Modeler 17.1*. Os resultados mostram que o *CoAP* supera o *SOAP*, com menores atrasos na rede e maior taxa de transferência.

O artigo conclui enfatizando que o *CoAP* oferece tamanhos de dados menores e transmissão mais rápida de pacotes em comparação com o *SOAP*. É mais adequado para comunicação *M2M*, especialmente em cenários em que as capacidades dos dispositivos são limitadas.

Em resumo, o artigo concentra-se na melhoria da comunicação dentro de um ambiente de rede inteligente através da introdução da *Gateway M2M* baseada *em* CoAP, que utiliza os dados do *DNP3.0*. Destaca os benefícios do *CoAP* em termos de eficiência e desempenho, especialmente ao lidar com dispositivos de capacidade limitada.

Segundo o autor Iglesias-Urkia et al., 2019, a comunicação eficaz desempenha um papel vital na evolução das redes elétricas inteligentes (*Smart Grids*), uma vez que a modernização dos sistemas de energia requer uma troca de dados ágil e confiável. Este autor explora a importância da comunicação *M2M* no contexto das *Smart Grids* e destaca o *CoAP* como uma alternativa viável e leve para melhorar a interoperabilidade e eficiência da comunicação.

O artigo começa com uma introdução abrangente, destacando a necessidade de tecnologias avançadas para melhorar a fiabilidade, estabilidade e eficiência das redes elétricas. A *Smart*

Grid é apresentada como uma resposta a essa necessidade, procurando otimizar a eficiência energética através da partilha bidirecional de informações de geração e consumo de energia.

O *DNP3.0* é identificado como um padrão crítico nos sistemas de supervisão e aquisição de dados (*SCADA*) para energia. Este protocolo é aplicado em várias áreas, incluindo Subestações, Recursos Energéticos Distribuídos (*DERs*) e Sistemas de Automação de Distribuição (*DASs*) em sistemas de energia. A importância do *DNP3.0* reside na sua capacidade de recolher e armazenar informações de estado, bem como sinais analógicos e digitais de dispositivos remotos.

A necessidade de comunicação *M2M* em *Smart Grids* é acentuada, uma vez que os sistemas de energia abrangem diversas áreas geográficas. A arquitetura funcional *M2M* recomendada pelo Instituto Europeu de Normalização das Telecomunicações (*ETSI*) é explorada como uma base sólida para a integração de serviços *web* que possibilitam a comunicação entre dispositivos.

No entanto, o documento reconhece que a comunicação M2M é desafiada pelo uso do *SOAP*, que apresenta limitações significativas, incluindo mensagens pesadas e estruturas complexas devido ao uso de *XML*. Além disso, o *SOAP* é baseado no *TCP*, o que pode não ser a melhor opção para ambientes *M2M* com recursos limitados.

Aqui é onde o *CoAP* entra como uma alternativa viável. O *CoAP* é descrito como uma solução leve que permite a transferência eficiente de dados em ambientes com recursos limitados. É baseado no modelo *REST*, o *CoAP* e utiliza *UDP* para troca de mensagens entre dispositivos. Esta abordagem é considerada mais adequada para cenários de comunicação *M2M*, especialmente em redes restritas.

O artigo destaca a principal contribuição da pesquisa, que é a proposta de um método de mapeamento *CoAP* para sistemas de automação de distribuição baseados em *DNP3.0*. Este mapeamento é essencial para garantir a interoperabilidade entre os dois protocolos, permitindo a comunicação eficaz entre dispositivos em *Smart Grids*.

Os resultados destacam a eficiência do *CoAP*, com mensagens significativamente menores e melhor desempenho, especialmente em cenários de elevada carga. A diferença entre o

tamanho das mensagens *CoAP* (27 bytes) e *SOAP* (366 bytes) demonstra a vantagem da abordagem mais leve do *CoAP*.

Concluindo, este artigo oferece uma revisão abrangente da importância da comunicação *M2M* em *Smart Grids*, destacando o papel fundamental do *CoAP* como um protocolo mais adequado para ambientes com recursos limitados.

O artigo "*IETF Protocol Suite for the Internet of Things: Overview and Recent Advancements*", escrito por Morabito & Jimenez, 2020 proporciona uma visão abrangente dos esforços do *Internet Engineering Task Force (IETF)* no domínio *IoT*, destacando atividades de normalização e investigação, bem como iniciativas de apoio relacionadas com o *IoT*. O artigo cobre várias áreas técnicas, incluindo conectividade, encaminhamento, aplicações e segurança. Discute igualmente atividades experimentais, casos de utilização, infraestruturas e o envolvimento de outras organizações relacionadas com o *IETF*. O artigo apresenta uma análise detalhada do progresso alcançado nos últimos dois anos nestas áreas e destaca os grupos de trabalho relevantes, grupos de investigação e outras comissões e organizações que contribuem para a normalização do *IoT* no âmbito do *IETF*.

Conectividade: neste tópico, o artigo aborda como os dispositivos *IoT* se ligam à rede. Discute vários grupos de trabalho, como *6LoWPAN*, *6TiSCH*, *6lo*, *LPWAN* e *ROLL*, envolvidos em soluções de conectividade e encaminhamento para o *IoT*. O artigo também destaca os esforços para trazer o *IPv6* para dispositivos e redes com recursos limitados, garantindo uma comunicação eficiente.

Encaminhamento: o encaminhamento é fundamental para garantir que os dados sejam encaminhados corretamente entre dispositivos no *IoT*. O artigo detalha os esforços relacionados com o encaminhamento na *IoT*, mencionando os grupos de trabalho e comissões que trabalham em soluções de encaminhamento específicas, como os já mencionados, bem como outros.

Aplicações: na área das aplicações *IoT*, o artigo menciona o *CoRE*, que se concentra na definição de mecanismos para aplicações orientadas a recursos. Além disso, introduz o grupo de investigação *Thing-to-Thing* (*T2TRG*) e o *CBOR*, que se concentram em aplicações específicas do *IoT*.

Segurança: a segurança no *IoT* é crucial. O artigo detalha o *DICE*, *ACE*, *SUIT*, *TEEP*, *COSE* e *RATS*, que trabalham no desenvolvimento de mecanismos de segurança e autenticação para dispositivos *IoT*. Também destaca a importância das atualizações seguras de *firmware* e dos ambientes de execução confiáveis para dispositivos *IoT*.

Casos de Utilização e Infraestruturas: nesta parte, o artigo detalha casos de utilização e infraestruturas relacionados com o *IoT*. Alguns exemplos incluem *IPWAVE*, *HOMENET* e *LWIG*, que abordam casos de utilização, desafios de rede e infraestrutura específicos do *IoT*. Também menciona o *Internet Research Task Force (IRTF)*, como o *Information-Centric Networking RG (ICNRG)* e o *Decentralized Internet Infrastructure RG (DINRG)*, que exploram tópicos relacionados com o *IoT*.

Organizações de Apoio: O artigo apresenta organizações que colaboram com o *IETF* na promoção da normalização, coordenação e orientação técnica do *IoT*. Isso inclui o *IoT Directorate*, *IAB* e *IANA*, que desempenham papéis essenciais na governança e orientação do *IoT* no contexto do *IETF*.

Estes tópicos abrangem os principais aspetos do *IoT* no âmbito do *IETF*, desde a conectividade e segurança até casos de utilização específicos e as organizações que apoiam esses esforços de normalização. Este artigo serve como um recurso valioso para compreender as diversas atividades e avanços no seio do *IETF* relacionados com o *IoT*, tornando-se uma referência útil para investigadores, programadores e partes interessadas no campo do *IoT*.

O artigo *IEEE Xplore Full-Text* aborda a integração do protocolo *CoAP* em casas inteligentes e como este pode substituir protocolos proprietários. O problema central é a integração de dispositivos existentes com o *CoAP*, que segue uma arquitetura semelhante à da *Web*.

O artigo destaca a importância do *CoAP* como um protocolo para dispositivos conectados à *Internet* por um padrão único em casas inteligentes. É discutida a configuração típica de redes domésticas e o uso de protocolos proprietários, como o *FS20*, para controlar dispositivos.

São detalhados aspectos técnicos, como a escolha de *hardware* e a integração do *CoAP* com o *FS20*. Isso inclui o mapeamento de comandos e a descoberta de dispositivos. O texto descreve a biblioteca *CoAP* e outros componentes de *software* usados na implementação.

É também explorado como os endereços do *FS20* são traduzidos para *URIs CoAP*, bem como como os comandos do *FS20* são refletidos como operações *CoAP*.

A descoberta de dispositivos é abordada numa secção separada, considerando que dispositivos podem aparecer e desaparecer dinamicamente na rede.

O artigo também se compara a trabalhos relacionados na área de automação residencial, incluindo protocolos como *KNX* e *JenNet-IP*.

As conclusões resumem os principais pontos do artigo, incluindo o sucesso na implementação de uma *gateway CoAP* para dispositivos *FS20*. A importância da segurança na comunicação também é destacada, com menção ao uso de *SSH* para garantir a segurança.

O artigo sugere possíveis direções futuras para melhorar o sistema, como a implementação de segurança de ponta a ponta à medida que mais dispositivos com suporte nativo para *CoAP* se tornam disponíveis. Em resumo, o artigo fornece uma visão abrangente da integração do *CoAP* em casas inteligentes.

O autor Beltran & Skarmeta, 2017 aborda a importância da *delegated authorization*

no contexto do protocolo *CoAP* no *IoT*, realçando a necessidade de um melhor controlo de acesso para combater as recentes violações de segurança. Apresenta a abordagem do *ACE* (*Authentication and authorization for Constrained Environments*) como uma solução

promissora para servidores e dispositivos *IoT* com recursos limitados. Explora a relação entre o *OAuth* e o *IoT*, destacando as diferenças na aplicação do *OAuth* na *web* convencional e no *IoT*.

O *ACE* é descrito detalhadamente, com ênfase no seu uso de *tokens Proof-of-Possession* (*PoP*) para reforçar a segurança. O artigo aborda a confiança estabelecida entre clientes, servidores de autorização e servidores de recursos por meio de um processo de registo. Os procedimentos de pedido de autorização, resposta, estabelecimento de segurança, pedido e resposta de recursos são explicados ao pormenor.

A conclusão enfatiza o potencial do *ACE* para melhorar a segurança, escalabilidade e interoperabilidade no IoT, embora saliente a necessidade de investigações futuras sobre o seu desempenho, especialmente em dispositivos *IoT* com recursos limitados.

No artigo "*REST-ful CoAP Message Authentication*", o autor Nguyen & Iacono, 2016, aborda a segurança das mensagens no contexto do *CoAP*, comum no *IoT*. O *CoAP* utiliza o protocolo *UDP* para transporte, sendo que a segurança padrão é fornecida pelo *Datagram Transport Layer Security* (*DTLS*), um protocolo de segurança orientado para datagramas.

No entanto, o artigo destaca que o *DTLS* não oferece uma proteção adequada em cenários de *IoT*, onde as mensagens frequentemente passam por vários componentes intermediários. Isso torna necessário adotar medidas de proteção ao nível das mensagens para complementar a segurança do transporte.

O artigo propõe um esquema de autenticação de mensagens *CoAP* baseado em *REST* com o intuito de estabelecer uma camada de segurança orientada para mensagens no *CoAP*. Isso é particularmente importante devido à arquitetura baseada em *REST* e às limitações de recursos em redes e dispositivos *IoT*.

O esquema de autenticação proposto inclui a geração e verificação de assinaturas das mensagens *CoAP*, abrangendo não só os dados da carga útil, mas também informações cruciais, como o método, o *URI* e as opções do *CoAP*. Isso contribui para garantir a integridade e autenticidade das mensagens *CoAP* como um todo.

O artigo de Naik, 2017 aborda a importância da seleção adequada de protocolos de mensagens em sistemas *IoT*. Compara e analisa quatro protocolos amplamente utilizados: *MQTT, CoAP, AMQP* e *HTTP*, com o objetivo de ajudar os leitores a compreender as características e limitações de cada protocolo, de modo a poderem fazer a escolha mais apropriada para as necessidades dos seus sistemas *IoT*. É possível verificar as comparações na Tabela 1 - Análise Comparativa de Protocolos de Mensagens para Sistemas IoT (Naik, 2017).

O MQTT (Message Queuing Telemetry Transport Protocol) é um protocolo *M2M* introduzido em 1999, projetado para comunicações leves em redes com recursos limitados. Utiliza o *TCP* como protocolo de transporte e oferece três níveis de Qualidade de Serviço (*QoS*) para garantir a entrega confiável de mensagens. O *MQTT* é mais adequado para redes de dispositivos pequenos que precisam de monitorização ou controlo a partir de um servidor na *Internet*.

O *CoAP (Constrained Application Protocol)* é um protocolo leve de *M2M* desenvolvido pelo grupo de trabalho *CoRE* do *IETF*. Suporta arquiteturas de pedido/resposta e de recursos/observação e utiliza o *UDP* como protocolo de transporte. O *CoAP* é ideal para dispositivos com recursos limitados e é especialmente adequado para a integração com a *Web* e recursos de observação.

O *AMQP (Advanced Message Queuing Protocol)* é um protocolo empresarial de mensagens introduzido em 2003, projetado para confiabilidade, segurança e interoperabilidade. Suporta arquiteturas de pedido/resposta e de publicação/assinatura com base em tópicos. O *AMQP* oferece diversas funcionalidades relacionadas a mensagens, como filas confiáveis, roteamento flexível e transações. Utiliza o *TCP* como protocolo de transporte e é amplamente utilizado em projetos de grande escala, como monitorização oceanográfica e computação em nuvem da *NASA*.

O *HTTP (Hyper Text Transport Protocol)* é o protocolo predominante da *Web*, amplamente utilizado em sistemas *IoT* para comunicações pela *Internet*. Suporta a arquitetura *RESTful* de pedido/resposta e utiliza o *TCP* como protocolo de transporte. O *HTTP* oferece recursos avançados de comunicação *web*, mas não define explicitamente a Qualidade de Serviço (*QoS*), requerendo suporte adicional para esse fim. No entanto, é menos eficiente em termos de

tamanho de mensagem, sobrecarga, consumo de energia e largura de banda em comparação com os outros protocolos.

O artigo sublinha que a escolha do protocolo de mensagens adequado para sistemas *IoT* depende das necessidades específicas de cada aplicação. Cada protocolo tem as suas próprias vantagens e desvantagens, conforme se verifica na Tabela 1 - Análise Comparativa de Protocolos de Mensagens para Sistemas IoT, sendo mais adequado para diferentes tipos de dispositivos e cenários de utilização. A seleção do protocolo ideal deve considerar fatores como consumo de recursos, segurança, *QoS*, interoperabilidade e padronização. Além disso, a evolução contínua do cenário *IoT* pode influenciar as preferências por protocolos no futuro. A seleção do protocolo deve ser baseada em considerações práticas e nas características específicas do sistema *IoT* em questão.

Tabela 1 - Análise Comparativa de Protocolos de Mensagens para Sistemas IoT (Naik, 2017)

Criteria	MQTT	CoAP	AMQP	HTTP
1. Year	1999	2010	2003	1997
2. Architecture	Client/Broker	Client/Server or Client/Broker	Client/Broker or Client/Server	Client/Server
3. Abstraction	Publish/Subscribe	Request/Response or Publish/Subscribe	Publish/Subscribe or Request/Response	Request/Response
4. Header Size	2 Byte	4 Byte	8 Byte	Undefined
5. Message Size	Small and Undefined (up to 256 MB maximum size)	Small and Undefined (normally small to fit in single IP datagram)	Negotiable and Undefined	Large and Undefined (depends on the web server or the programming technology)
6. Semantics/ Methods	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close	Get, Post, Put, Delete	Consume, Deliver, Publish, Get, Select, Ack, Delete, Nack, Recover, Reject, Open, Close	Get, Post, Head, Put, Patch, Options, Connect, Delete
7. Cache and Proxy Support	Partial	Yes	Yes	Yes
8. Quality of Service (QoS)/ Reliability	QoS 0 - At most once (Fire-and-Forget), QoS 1 - At least once, QoS 2 - Exactly once	Confirmable Message (similar to At most once) or Non-confirmable Message (similar to At least once)	Settle Format (similar to At most once) or Unsettle Format (similar to At least once)	Limited (via Transport Protocol - TCP)
9. Standards	OASIS, Eclipse Foundations	IETF, Eclipse Foundation	OASIS, ISO/IEC	IETF and W3C
10. Transport Protocol	TCP (MQTT-SN can use UDP)	UDP, SCTP	TCP, SCTP	TCP
11. Security	TLS/SSL	DTLS, IPSec	TLS/SSL, IPSec, SASL	TLS/SSL
12. Default Port	1883/ 8883 (TLS/SSL)	5683 (UDP Port)/ 5684 (DLTS)	5671 (TLS/SSL), 5672	80/ 443 (TLS/SSL)
13. Encoding Format	Binary	Binary	Binary	Text
14. Licensing Model	Open Source	Open Source	Open Source	Free
15. Organisational Support	IBM, Facebook, Eurotech, Cisco, Red Hat, Software AG, Tibco, ITSO, M2Mi, Amazon Web Services (AWS), InduSoft, Fiorano	Large Web Community Support, Cisco, Contiki, Erika, IoTivity	Microsoft , JP Morgan, Bank of America, Barclays, Goldman Sachs, Credit Suisse	Global Web Protocol Standard

No artigo "*A Comparative analysis of MQTT and IoT application protocols*" o autor Al Enany et al., 2021 realça várias diferenças entre o protocolo *MQTT* e outros protocolos de aplicação *IoT*, como *COAP* e *AMQP*. Estas diferenças são observadas em diferentes cenários de utilização, evidenciando as vantagens e desvantagens de cada protocolo.

Uma das principais diferenças abordadas é a latência e o consumo de largura de banda. O *MQTT* é elogiado pela sua confiabilidade, mas, devido aos processos de *handshaking* e à troca de *acknowledgments*, pode apresentar maior latência em comparação com o protocolo *COAP*. Em situações de baixo tráfego e mensagens de pequeno tamanho, o *COAP* destaca-se devido à menor latência e consumo de largura de banda.

Outra diferença significativa é a confiabilidade e o tamanho das mensagens. O *MQTT* é a escolha preferida para aplicações que requerem alta confiabilidade, oferecendo três níveis de Qualidade de Serviço (*QoS*). No entanto, isso pode resultar em um maior consumo de largura de banda. Em comparação com o *AMQP*, o *MQTT* demonstra eficácia na entrega ordenada de mensagens e melhor desempenho em situações de alta confiabilidade, embora possa sofrer perdas em cenários de alto atraso.

O consumo de energia também é um fator importante a ser considerado. Em cenários de dispositivos médicos e sistemas de saúde, onde a economia de energia é essencial, o protocolo *COAP* é preferível. Isso deve-se ao seu modelo de solicitação/resposta baseado em *UDP*, que consome menos energia em comparação ao *MQTT*, que depende do *TCP* para garantir a confiabilidade.

Além disso, as características dos protocolos de transporte desempenham um papel crucial. O MQTT opera sobre o protocolo TCP, tornando-o altamente confiável, mas com maior latência devido aos processos de estabelecimento de conexão e handshaking. O AMQP também utiliza um modelo de publicação/assinatura, mantendo a entrega ordenada de mensagens. No entanto, o *MQTT* destaca-se em termos de carga útil máxima e é mais eficaz em situações de baixo atraso.

Em resumo, o artigo salienta a importância da escolha do protocolo de aplicação *IoT* mais adequado com base nas necessidades específicas de cada cenário. O *MQTT* é valorizado pela sua confiabilidade em situações de alto tráfego e alto volume de mensagens, mas pode

apresentar latência e consumo de largura de banda superiores em comparação com o *COAP*. O *AMQP* oferece entrega ordenada de mensagens e é eficaz em redes instáveis. A análise comparativa fornece informações essenciais para profissionais que procuram a implementação de protocolos de aplicação *IoT*, destacando a necessidade de selecionar o protocolo adequado.

2.4.1. Bibliotecas para desenvolvimento de soluções Bluetooth

Em 2020, Andy Lee propôs a criação de um dispositivo periférico *BLE* que usa a biblioteca *BlueZ* num *Raspberry Pi 3+* (Andy Lee, 2020).

O autor decidiu criar um periférico *BLE* com um servidor *GATT* no *Raspberry Pi*. O *BlueZ* é mencionado como a pilha *Bluetooth* para *Linux*, que lida tanto com *Bluetooth BR*/*EDR* quanto com *BLE*. Para comunicar com o *BlueZ*, é utilizado o *D-Bus*, um método de comunicação entre processos (IPC) que permite que diferentes processos num sistema *Linux* comuniquem entre si.

O artigo explica os conceitos de objetos e interfaces no *D-Bus*. Os objetos são coisas concretas nas quais se pode atuar e podem implementar uma ou mais interfaces. As interfaces são compostas por métodos, sinais e propriedades.

O documento descreve o uso da ferramenta *bluetoothctl* para se conectar com o *BlueZ* e o *D-Bus.* Essa ferramenta permite controlar o adaptador *Bluetooth*, configurar dados de publicidade e adicionar serviços e características personalizadas (Andy Lee, 2020).

De seguida, é abordada a criação de uma aplicação *Python* (*Welcome to Python.org*, sem data) que utiliza as mesmas *APIs* do *D-Bus* usadas pelo *bluetoothctl*. O exemplo de aplicação apresentado é voltado para um problema real: controlar uma máquina de café expresso por meio de um periférico *BLE*. O código da aplicação em *Python* é explicado passo a passo no artigo.

No final do artigo, o autor menciona que o código completo do projeto está disponível para consulta e que o mesmo foi testado num *Raspberry Pi 3+* com *BlueZ* 5.53.

3. Implementação de uma Gateway COAP/BLE

3.1. Arquitetura

Para a implementação bem-sucedida deste projeto, foi necessária a conceção de uma arquitetura específica. A Figura 19 destaca essa arquitetura, ilustrando a relação entre o *COAP Client* e a *Gateway* e como trabalham harmoniosamente para proporcionar uma experiência eficiente ao utilizador e armazenar informações valiosas numa base de dados.

A criação dessa arquitetura personalizada foi um passo essencial, pois permitiu que o *COAP Client* e a *Gateway* se integrassem de maneira coesa e eficaz, de acordo com as exigências específicas dos projetos em questão. A arquitetura foi desenhada sob medida para atender aos requisitos únicos dessas aplicações, garantindo a interação perfeita com dispositivos *BLE*.

A escolha da arquitetura orientada a documentos, como o *MongoDB*, é uma decisão relacionada com a implementação do protótipo, uma vez que se alinha com a estrutura de dados e o modelo de aplicação necessários para o projeto.

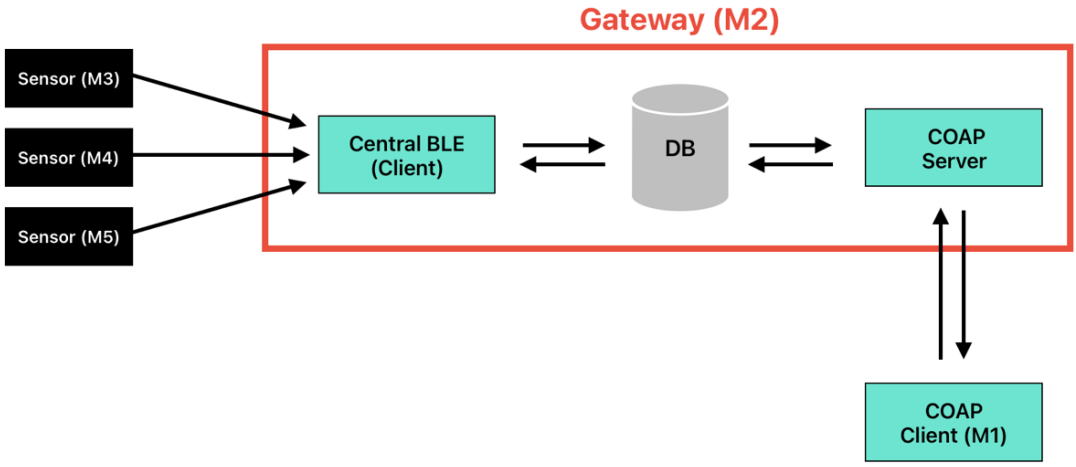

Figura 19 - Arquitetura desenvolvida para a Gateway

Conforme a Figura 19, o *COAP Client* e a *Gateway* estão relacionados no que toca à funcionalidade de interação com dispositivos *Bluetooth LE* e ao armazenamento das informações obtidas numa base de dados *MongoDB*.

A arquitetura do sistema desenvolvido envolve a interação entre duas máquinas, *M1* e *M2*, ambas a executar sistemas *Linux*. Estas máquinas desempenham papéis distintos na operação do sistema, sendo *M1* o *COAP Client* e *M2* a *Gateway BLE*. O sistema é projetado para permitir a comunicação entre essas máquinas por meio de mensagens *COAP*, enquanto também aproveita uma base de dados *MongoDB* para armazenar informações relevantes.

Na *M1*, o *COAP Client* foi implementado para interagir com o *COAP Server* localizado na *M2*. O *COAP Client* é responsável por enviar solicitações ao *COAP Server* para realizar várias operações, como obter informações de serviços, adicionar ou excluir serviços, contar serviços e calcular o espaço em disco disponível. A comunicação entre o *COAP Client* e o *COAP Server* é baseada em mensagens *COAP*.

Na *M2*, o *COAP* Server foi implementado e atua como o intermediário entre o *COAP Client* na *M1* e a base de dados *MongoDB*. A biblioteca *Python pymongo* é utilizada para estabelecer uma conexão direta com a base de dados *MongoDB* e realizar operações de consulta, inserção e exclusão de dados na coleção *MongoDB*. Isso garante que o *COAP Server* possa aceder e manipular dados na base de dados conforme necessário.

Os recursos definidos no *COAP Server*, como *ServicesResource*, *ServiceByIdResource*, *ServiceCountResource*, *DiskSpaceResource* e *ScanResource*, fornecem *endpoints* para os clientes *COAP* enviarem solicitações. Esses *endpoints* mapeiam para operações específicas no servidor, como *GET*, *POST* e *DELETE*, permitindo que o *COAP Client* aceda e manipule os dados armazenados na base de dados. O *payload* das mensagens *COAP* é formatado em *JSON/CBOR* antes de ser enviado e recebido, facilitando a transmissão de dados entre o *COAP Client* e o *COAP Server*.

Além disso, a *M2* também desempenha o papel de *Central BLE (Client).* A *Central BLE (Client)* é responsável por ler dados dos sensores distribuídos, que podem ser máquinas independentes com capacidades de computação variadas. Esses sensores partilham informações *BLE* com o *Central BLE (Client).* O *Central BLE (Client)* é, por sua vez, responsável por enviar as informações para a base de dados *MongoDB*. Isso cria uma ponte entre os dispositivos *BLE* e o sistema central, permitindo a monitorização dos dados capturados.

Em resumo, a arquitetura do sistema envolve duas máquinas *Linux (M1 e M2)* que se comunicam por meio de mensagens *COAP*. A *M1* atua como o *COAP Client*, enquanto a *M2* desempenha o papel de *Gateway BLE* e fornecendo acesso à base de dados *MongoDB*. Essa arquitetura permite a interação eficiente entre dispositivos *BLE*, o armazenamento e a manipulação de dados na base de dados *MongoDB* e a comunicação com o *COAP Client* por meio do *COAP Server*.

3.2. Protótipo

A Figura 20 descreve o processo de *scan* de dispositivos *Bluetooth Low Energy (BLE)* que foram implementados no projeto. O diagrama destina-se a tornar mais explícito a sequência de ações. Importante realçar que o COAP Server, o Central BLE Client e a base de dados constituem a designada *Gateway.*

Neste diagrama de sequência, descreve-se o processo de uma verificação de dispositivos BLE com as interações entre o COAP Client (M1), o COAP Server, o Central BLE Client e a Base de Dados. O objetivo é tornar mais explícitas as etapas envolvidas. O processo tem início quando se pretende efetuar uma verificação dos dispositivos BLE na rede. Inicialmente, é enviado um pedido de verificação via COAP para o COAP Server por parte do COAP Client (M1). O COAP Server, ao receber o pedido, identifica a necessidade de realizar uma verificação BLE e encaminha o pedido para o Central BLE Client. O Central BLE Client inicia a verificação BLE de todos os dispositivos nas imediações e recebe dados dos dispositivos BLE ao seu alcance. Os dados coletados são armazenados pelo Central BLE Client na Base de Dados.

Posteriormente, para listar todos os nós BLE armazenados na Base de Dados é enviado um novo pedido via COAP para listar os nós armazenados. O COAP Server, ao receber o pedido de listagem, reconhece a necessidade de consultar a Base de Dados e envia uma solicitação à Base de Dados. Ela processa a solicitação, localiza os dados dos nós BLE armazenados e responde ao COAP Server com os dados solicitados.

Por fim, o COAP Server recebe a resposta da Base de Dados e envia essa resposta ao COAP Client (M1), fornecendo a lista de nós BLE armazenados. Este diagrama de sequência detalha a sequência de ações envolvendo os vários componentes, permitindo uma compreensão mais clara do processo de verificação.

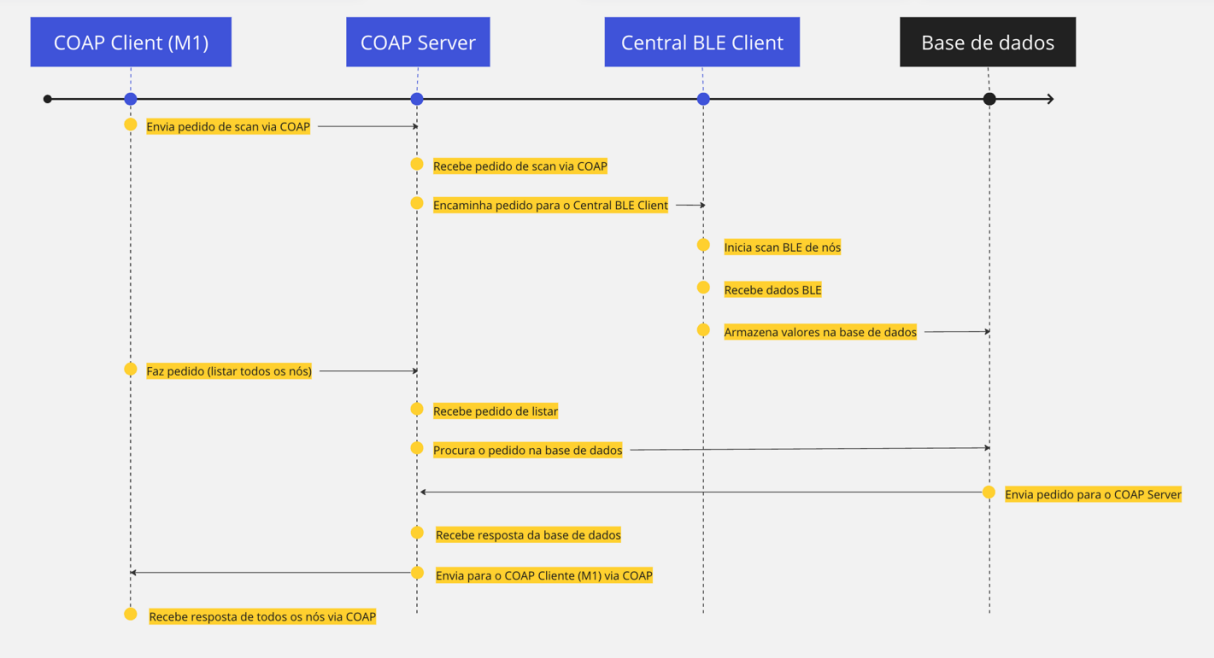

Figura 20 - Diagrama de sequência

3.2.1. Scan dos dispositivos

Primeiramente, foi desenvolvido um programa (oliveirasmj/scanBle) que utiliza a biblioteca *Bleak* (*bleak — bleak 0.21.1 documentation*, sem data) em *Python* para realizar *scan* e obter informações sobre dispositivos *BLE*, incluindo os respetivos serviços e características. A biblioteca *Bleak* é uma ferramenta agnóstica para *Bluetooth LE* que suporta diversas plataformas, tais como *Windows*, *Linux*, *macOS*, *Android* e *iOS*.

O código começa por definir uma classe de cores para imprimir as informações no terminal com cores diferentes para facilitar a visualização. Em seguida, há uma função chamada *device_details_to_dict* que é usada para converter os detalhes brutos de um dispositivo *BLE* num dicionário *Python*. A função trata possíveis erros causados por dados ausentes no dispositivo e fornece um valor *None* para esses dados.

A função principal, *main*, começa a imprimir um cabeçalho na saída do terminal e, em seguida, chama a função *BleakScanner.discover()* para fazer scan de dispositivos *BLE* que estejam

próximos. De seguida, os detalhes brutos dos dispositivos são exibidos na saída do terminal, juntamente com o número total de dispositivos encontrados.

O código então percorre cada dispositivo e usa a função *device_details_to_dict* para extrair os detalhes do dispositivo num dicionário *Python*. Esses detalhes são mostrados no terminal para cada dispositivo encontrado.

A próxima seção do código é responsável por obter uma lista de serviços e características disponíveis em cada dispositivo encontrado. Para cada dispositivo encontrado, o código tenta conectar-se ao dispositivo usando *BleakClient* e *BleakScanner.find_device_by_address()*. De seguida, o código imprime os serviços e características do dispositivo na saída do terminal.

Se ocorrer algum erro durante a execução do código, as informações relevantes serão impressas na saída do terminal para que o utilizador possa diagnosticar o problema.

Em resumo, este código é um exemplo de como usar a biblioteca *Bleak* para fazer scan de dispositivos *BLE* próximos e obter informações sobre seus serviços e características.

3.2.2. Guardar os valores na base de dados

Em particular, foi incorporada a capacidade de recolher e registar tanto o endereço como o nome de todos os dispositivos *BLE* identificados numa coleção no *MongoDB*. Além disso, o código está agora apto a estabelecer ligações com os dispositivos BLE encontrados, de modo a obter uma lista completa dos serviços e funcionalidades disponíveis, que são depois guardados numa coleção específica no *MongoDB*.

Como é feita a inserção na base de dados *MongoDB*:

- No início do código desenvolvido para este projeto para o *Server* (oliveirasmj/scanBle), foi feita uma conexão com a base de dados *MongoDB* através do *PyMongo*, utilizando a variável *cluster*.
- Foi criada uma nova base de dados chamada *dbscan*, e a variável *db* foi definida como essa base de dados.
- Quando um dispositivo é encontrado e os serviços são listados, o seu *address* e *name* são inseridos na base de dados. É criada uma nova coleção chamada *services* (se ainda não existir), e os dados são inseridos na coleção usando a variável *collection* e a função *insert_one()*. O *id* gerado pelo *MongoDB* para esse registo é guardado na variável *_id*.
- Para cada serviço encontrado, as informações (*description*, *uuid*, *subDescription*, *handle* e *properties*) são guardadas num dicionário chamado *serviceDetails*, que é então adicionado a uma lista de *services* associados ao dispositivo. A função *update_many()* é usada para atualizar o registo na coleção *services* com o novo serviço adicionado.

3.2.3. Criação do servidor *COAP*

Foi criado um servidor *CoAP* (*Constrained Application Protocol*) que fornece acesso a informações de serviços e espaço em disco por meio de recursos *CoAP*. O servidor utiliza a biblioteca *aiocoap* para implementar a lógica *CoAP* e a biblioteca *pymongo* para conectar e interagir com uma base de dados *MongoDB*.

O código define quatro classes que herdam da classe *resource*.*Resource* da biblioteca *aiocoap*:

- ***ServicesResource***: lida com as solicitações *GET* e *POST* para obter e criar serviços na coleção de serviços *MongoDB*.
- ***ServiceByIdResource***: lida com as solicitações *GET* e *DELETE* para obter e excluir um serviço específico da coleção de serviços *MongoDB*, identificado pelo seu *ID*.

- ***ServiceCountResource***: lida com a solicitação *GET* para obter o número total de serviços na coleção de serviços *MongoDB*.
- ***DiskSpaceResource***: lida com a solicitação *GET* para obter informações sobre o espaço em disco.

Além disso, há uma função principal main() que adiciona instâncias de *ServiceByIdResource* para cada serviço na coleção de serviços *MongoDB* e inicia o servidor *CoAP*.

Ao iniciar o servidor, o programa cria uma instância da classe *aiocoap*.*Context* e inicia-a com o servidor *CoAP*. O servidor escuta as solicitações *CoAP* recebidas e encaminha para as instâncias apropriadas de recursos para processamento. O servidor continua a executar até que seja interrompido por um sinal externo (como o sinal *SIGINT* gerado por pressionar *Ctrl-C* no terminal).

3.2.4. Criação do cliente *COAP*

Este código é uma implementação de um cliente *CoAP* que interage com um servidor *CoAP* criado e que expõe serviços *RESTful*. O cliente oferece várias funcionalidades, incluindo:

- Obter todos os serviços disponíveis a partir do servidor *CoAP*
- Obter um serviço específico pelo seu *ID* a partir do servidor *CoAP*
- Adicionar um novo serviço no servidor *CoAP*
- Excluir um serviço pelo seu *ID* a partir do servidor *CoAP*
- Contar quantos serviços estão disponíveis no servidor *CoAP*
- Calcular o espaço em disco disponível no servidor *CoAP*
- Realizar uma varredura *BLE* para dispositivos próximos

O cliente *COAP* interage com o servidor *CoAP* usando a biblioteca aiocoap. O *aiocoap* é uma biblioteca cliente e servidor de implementação do protocolo *CoAP* (*Constrained Application Protocol*), que é um protocolo de aplicação web projetado para dispositivos limitados por recursos, como sensores e atuadores, que exigem baixo consumo de energia e capacidade de rede limitada. O *aiocoap* implementa as especificações do protocolo *CoAP* definidas na *RFC* 7252 e fornece uma *API* assíncrona fácil de usar para *Python*. O *aiocoap* permite que os utilizadores criem aplicações de *IoT* (Internet das Coisas) e que comuniquem de forma eficiente e confiável com dispositivos limitados por recursos, bem como com outros servidores *CoAP* compatíveis. Ele é frequentemente usado em projetos que envolvem comunicação entre dispositivos *IoT* e servidores *web*.

Cada funcionalidade é implementada numa função separada e é chamada com base na escolha do utilizador. A função principal é *main()* que apresenta um menu para o utilizador escolher a opção desejada. O código também usa a biblioteca scan para executar o scan *BLE*.

O código está interligado com o *server.py* na medida em que ambos usam a mesma tecnologia de protocolo *CoAP* para se comunicarem.

3.2.5. Criação da base de dados localmente

Para a criação da base de dados localmente em *MongoDB*, foi necessário seguir os seguintes passos:

1. Instalação do *MongoDB* na máquina local. As instruções para instalação podem ser encontradas no site oficial do *MongoDB*.
2. Abrir um terminal e iniciar o servidor *MongoDB* local executando o comando *sudo service mongod start*. O servidor será executado em segundo plano e ficará em execução até ser explicitamente interrompido. É possível verificar o seu estado através do comando *sudo service mongod status*

3. Abrir outro terminal ou *prompt* de comando e iniciar o *shell* do *MongoDB* executando o comando *mongo*. O *shell* irá conectar-se ao servidor *MongoDB* local e permitir que sejam executados comandos na base de dados.
4. No shell do *MongoDB*, foi criada uma nova base de dados executando o comando *use dbscan*. Se a base de dados não existir, ela será criada automaticamente.
5. Na base de dados criada, é possível criar coleções (equivalentes a tabelas em base de dados relacionais) e inserir documentos (equivalentes a linhas em tabelas) usando o *MongoDB Shell* ou um *driver* de cliente para *MongoDB*.

Este é o processo básico para criar uma base de dados local em *MongoDB*. No código que foi mostrado anteriormente, a conexão com a base de dados é estabelecida usando um objeto *MongoClient* fornecido pela biblioteca *PyMongo*. O objeto *MongoClient* pode ser inicializado com a *URL* de conexão e outros parâmetros de configuração necessários para se conectar à instância do *MongoDB*, que pode estar local ou remota. Em seguida, é possível selecionar uma base de dados e criar coleções e documentos usando as funções fornecidas pela biblioteca *PyMongo*.

Também é possível usar o *Compass* que é uma ferramenta gráfica para o *MongoDB* que permite gerir bases de dados *MongoDB* de forma fácil e intuitiva, incluindo a criação de coleções. Basta conectar-se ao servidor *MongoDB* em que é desejável criar a coleção e seguir as instruções do *Compass*.

3.2.6. Criação da base de dados na cloud

Também é possível usar o *MongoDB Atlas* (*account.mongodb.com*) para criar uma base de dados e coleções. É uma opção para quem prefere uma solução de base de dados em *cloud* em vez de instalar e configurar um servidor local. No entanto, é importante lembrar que essa

opção pode ter custos associados, dependendo do plano escolhido. Para isso foi necessário criar um *cluster*.

Foi também criada uma base de dados em *cloud* no seguinte endereço:

https://cloud.mongodb.com/v2/63ea8abb2697e25938b245fe#/clusters/detail/Cluster0

Apesar disso, para a aplicação utilizada a base de dados local é a que está em funcionamento ainda que a base de dados em *cloud* esteja ativada e disponível para usar a qualquer momento.

3.2.7. Criação de novo servidor *COAP* e cliente *COAP* com CBOR

No *server_cbor.py* é importado no início do arquivo: *import cbor2*.

A biblioteca é usada nas funções que retornam as respostas aos pedidos. As *strings* retornadas pelo servidor são serializadas em formato *CBOR* usando a função *cbor2.dumps(),* antes de serem enviadas para o cliente.

Por exemplo, na função *render_get()* de *ServicesResource*, a lista de serviços recuperada do banco de dados é convertida em uma lista de *strings* e depois serializada em formato *CBOR*:

```
services = mongo_collection.find()
response = [str(s) for s in services]
response_payload = cbor2.dumps(response)
return aiocoap.Message(payload=response_payload)
```

O mesmo padrão é seguido nas outras funções que retornam respostas: *render_post()*, *render_get()* e *render_delete()* das classes *ServicesResource* e *ServiceByIdResource*, bem como *render_get()* de *ServiceCountResource* e *DiskSpaceResource*.

Quanto ao *cliente_cbor.py*, o formato *CBOR* é usado como o formato de serialização de dados para o servidor *CoAP*. Quando o cliente faz uma solicitação *GET* ou *POST*, a resposta do servidor é uma carga útil codificada em *CBOR*, que é então decodificada usando a biblioteca *cbor2*.

O código usa a função *cbor2.loads()* para decodificar dados *CBOR*. O *loads()* é usado para carregar objetos *Python* a partir de um fluxo de *bytes* de dados *CBOR*. Por exemplo, o código *payload = cbor2.loads(response.payload)* decodifica a carga útil *CBOR* da resposta do servidor num objeto Python.

O código também usa a função *cbor2.dumps()* para codificar dados em formato *CBOR*. A função dumps() é usada para serializar objetos *Python* num fluxo de *bytes CBOR*. Por exemplo, é possível codificar um dicionário *Python* em *CBOR* usando *cbor2.dumps(payload)*.

3.2.8. Vantagens CBOR

A utilização do *CBOR* (*Concise Binary Object Representation*) como formato de serialização de dados tem algumas vantagens em relação a outros formatos como *JSON*, por exemplo:

- Eficiência em tamanho: o *CBOR* é um formato binário compacto que ocupa menos espaço que o *JSON*, o que pode ser importante quando se trabalha com grandes quantidades de dados ou em redes de comunicação com baixa largura de banda.

- Facilidade de *parsing*: como o *CBOR* é um formato binário, é mais fácil e rápido para o computador analisar e interpretar o conteúdo, reduzindo o tempo de processamento e melhorando o desempenho em comparação com o *JSON*.

- Portabilidade: o *CBOR* é um formato padronizado, com bibliotecas disponíveis para várias linguagens de programação e sistemas operacionais, o que facilita a portabilidade e interoperabilidade entre diferentes plataformas.

- Segurança: como o *CBOR* é um formato binário, pode ser mais difícil ser interpretado por humanos ou ser explorado por ataques de injeção de código malicioso, o que pode melhorar a segurança do sistema.

No caso específico do código apresentado, a utilização do *CBOR* pode melhorar o desempenho e eficiência da comunicação entre o cliente e o servidor, além de reduzir a carga de processamento em ambos os lados da conexão.

A RFC 8949, intitulada "Concise Binary Object Representation (CBOR)" define o formato e a semântica do CBOR, que é uma alternativa compacta e eficiente ao *JSON* para representar dados estruturados em formato binário.

Esta especificação descreve como os dados são codificados e interpretados no formato CBOR, incluindo as regras para representar valores de diferentes tipos, como inteiros, *strings*, *arrays* e objetos. A RFC 8949 desempenha um papel fundamental na padronização da representação binária de dados e é frequentemente usada em protocolos de rede e sistemas onde a eficiência do tamanho dos dados é crucial.

3.2.9. Exibição de troca de mensagens com os dispositivos

Para que fosse possível ouvir a comunicação de troca de mensagens com o dispositivo *BLE* foi adicionado o seguinte código abaixo. O código deste scan de *BLE* (*Bluetooth Low Energy*) tem como objetivo habilitar as notificações para cada característica (ou atributo) encontradas num serviço *BLE*:

```
async def notification_handler(sender: int, data: bytearray): print(f"Notification received - Characteristic handle: {sender}, Data: {data}")

(...)

for c in service.characteristics:
        # Ativar notificação/indicação para cada característica
        try:
            await client.start_notify(c.handle, notification_handler)
            print(f"Notification/Indication enabled for Characteristic {c.uuid}")
        except Exception as e:
            print(f"Error enabling Notification/Indication for Characteristic {c.uuid}: {e}")
```

Este código tem como objetivo ouvir as mensagens trocadas com dispositivos *BLE*. Para isso, é definida uma função chamada "notification_handler" que lida com as mensagens recebidas dos dispositivos *BLE*. Esta função recebe dois parâmetros: o identificador da característica que enviou a mensagem e os dados contidos na mensagem.

Em seguida, percorrem-se todas as características de um serviço *BLE*. Cada característica representa uma função ou dado específico que um dispositivo *BLE* pode fornecer.

Para cada característica encontrada, tenta-se ativar a escuta para receber notificações ou indicações. Se essa ativação for bem-sucedida, será exibida uma mensagem indicando que a notificação foi ativada para essa característica.

O código continua a verificar as características à procura de notificações e, quando uma notificação é recebida, a função "notification_handler" é chamada, mostrando o identificador da característica e os dados recebidos na mensagem.

Em resumo, este código é usado para monitorizar e receber informações de dispositivos *BLE*, como sensores ou outros dispositivos, permitindo que se saiba quando há atualizações ou mudanças nesses dispositivos. Isto é útil para recolher dados de dispositivos remotos que utilizam a tecnologia *BLE*.

4. Testes e análise de resultados

Neste capítulo, o objetivo principal consistiu na avaliação da escalabilidade da aplicação desenvolvida para efetuar a deteção de dispositivos *BLE* e na exploração do impacto do uso do *CBOR* no *COAP Server*. Foram realizados diversos testes com vista a atingir estes objetivos.

Numa primeira etapa, procedeu-se à configuração de múltiplos dispositivos *BLE* com dados de publicidade. Pretendeu-se avaliar em que medida a aplicação era capaz de lidar com este cenário e se se revelava escalável. Para a concretização desta configuração, recorreu-se ao *BlueZ* e à ferramenta *bluetoothctl*, através dos quais foram definidas informações como o nome do dispositivo, os dados do fabricante e os *UUIDs* de serviço nos pacotes de publicidade.

Num segundo momento, foi efetuada uma análise do impacto decorrente do uso do *CBOR* no *COAP Server*. O *CBOR* representa um formato conciso de representação binária de dados estruturados, com potencial para otimizar o tamanho das mensagens e reduzir a quantidade de dados transmitidos. Investigou-se se a adoção do *CBOR* no *COAP Server* poderia contribuir para a diminuição do tempo necessário para a realização das operações, quando comparado com uma situação em que o *CBOR* não é utilizado.

Paralelamente, foram conduzidos testes exaustivos de modo a avaliar a escalabilidade da aplicação *BLE*. Estes testes contemplaram a análise de fatores como o tempo de resposta da aplicação, o consumo de recursos e o desempenho global. Foram recolhidos dados e métricas pertinentes, visando a avaliação da capacidade da aplicação em lidar com um ambiente onde vários dispositivos *BLE* realizam anúncios simultaneamente. Ao longo destes testes, foram registados os tempos de resposta e os recursos consumidos, proporcionando uma avaliação abrangente dos requisitos de desempenho e eficiência. Além disso, foram monitorizados os comportamentos da aplicação à medida que o número de dispositivos *BLE* aumentava,

simulando cenários de elevada procura, com o intuito de determinar os limites de escalabilidade.

Por fim, optou-se por remover a interface gráfica do *Central BLE (Client)* e posteriormente do *COAP Client* com o objetivo de reduzir o consumo de recursos. Essa medida foi adotada para assegurar que o cliente pudesse alocar uma quantidade significativamente maior de CPU e RAM para suas tarefas essenciais, como processamento de dados, serviços de rede e armazenamento, sem comprometer o desempenho global. Com essa ação, procurou-se otimizar significativamente o uso dos recursos clientes, tornando-os mais eficientes e capazes de lidar com cargas de trabalho intensivas de maneira mais eficaz.

Para tal, foram recolhidos dados relativos ao consumo de recursos tanto quando a interface gráfica estava ativa como quando estava inativa, permitindo uma comparação direta dos recursos utilizados em ambas as situações. Esta análise revelou-se fundamental para a compreensão da forma como a interface gráfica influência o desempenho global da aplicação, tendo em conta os recursos do sistema.

4.1. Criação de um periphral BLE

No âmbito do desenvolvimento e configuração de dispositivos *Bluetooth*, é fundamental estabelecer uma comunicação eficaz entre dispositivos periféricos e dispositivos centrais. Para alcançar este objetivo, foi necessário configurar o ambiente e as informações de publicidade do dispositivo periférico, de forma a permitir que os dispositivos centrais o descubram e interajam adequadamente com ele.

Um dos componentes para esta configuração é o *BlueZ*, que atua como uma interface entre o hardware *Bluetooth* e o sistema operativo. Portanto, o primeiro passo consistiu em verificar se o *BlueZ* estava instalado no sistema, uma vez que é um requisito essencial para prosseguir com a configuração.

Isto é feito através da utilização do comando "sudo dbus-monitor --system "destination='org.bluez'" "sender='org.bluez'"", que permite monitorizar as chamadas efetuadas entre o *daemon BlueZ* e o barramento do sistema. Após verificar a presença do *BlueZ*, a próxima etapa envolve o uso da ferramenta "bluetoothctl", que permite configurar o dispositivo periférico e as suas informações de publicidade. Esta ferramenta é iniciada numa nova janela de terminal, digitando simplesmente "bluetoothctl".

Com a ferramenta "bluetoothctl" em funcionamento, o adaptador *Bluetooth* é ativado no terceiro passo, tornando-o pronto para comunicações Bluetooth.

O quarto passo abrange a configuração dos dados de publicidade, que são essenciais para que os dispositivos centrais possam identificar e interagir com o dispositivo periférico. Para isso, utiliza-se o comando "menu advertise" no *prompt* "bluetoothctl" para aceder ao menu de configuração de publicidade.

A configuração inclui a definição de dados do fabricante no quinto passo, especificando um ID de fabricante e dados específicos do fabricante.

No sexto passo, define-se o nome local do dispositivo, que será visível para dispositivos centrais durante o processo de descoberta.

O sétimo passo é opcional e permite a configuração de outros dados de publicidade, como serviços ou dados *UUID*, para fornecer informações adicionais no anúncio de publicidade. Com todas as informações de publicidade configuradas, no oitavo passo inicia-se o processo de anúncio do dispositivo, tornando-o visível para dispositivos centrais. Isso é feito com os comandos "back" para sair do menu *advertise* e "advertise on" para ativar a publicidade.

Por fim, no nono passo, verifica-se o anúncio do dispositivo utilizando uma aplicação de teste *BLE* num dispositivo central compatível. Isso permite confirmar que as informações configuradas, como o nome do dispositivo e os dados do fabricante, estão a ser anunciadas corretamente.

Esta configuração é fundamental para estabelecer comunicações *Bluetooth* de baixa energia em várias aplicações e pode ser replicada em diversos dispositivos utilizando o *BlueZ* e a ferramenta *bluetoothctl*. Além disso, esta abordagem oferece flexibilidade para explorar recursos adicionais do *BlueZ* e adicionar mais serviços conforme necessário.

4.2. Adicionar serviços não genéricos a cada dispositivo

Com o intuito de estabelecer uma comunicação eficiente entre dispositivos periféricos Bluetooth e dispositivos centrais, foi desenvolvido um script personalizado. Este script é responsável por configurar um *Bluetooth Periphral* com um perfil *GATT* (*Generic Attribute Profile*) flexível e robusto, fundamental para operações em dispositivos de Internet das Coisas (*IoT*) e outras aplicações baseadas em Bluetooth de baixa energia (*BLE*).

Inicialmente, o *Bluetooth* foi ativado e configurado no dispositivo periférico através da utilização da ferramenta "bluetoothctl". Isto incluiu tornar o dispositivo visível e pareável, bem como ativar um agente Bluetooth, que foi definido como agente padrão para simplificar o emparelhamento com outros dispositivos.

Seguiu-se o registo de um serviço *GATT*, no qual um *UUID* (Universally Unique Identifier) foi fornecido para representar o serviço oferecido pelo dispositivo periférico. Este serviço foi configurado como serviço principal, atuando como ponto central para a interação entre dispositivos periféricos e centrais.

De forma a tornar o perfil *GATT* mais completo e funcional, foram registadas características relacionadas ao serviço *GATT*. Duas características foram configuradas: uma característica somente leitura com o identificador "0x1234" e outra característica com suporte a leitura e escrita, identificada como "0x4567". Isto permitiu que o *periphral* disponibilizasse informações que poderiam ser lidas pelos dispositivos centrais e, em certos casos, até mesmo modificadas.

Após o registo bem-sucedido do serviço e das características, o script notificou o sistema *BlueZ* sobre estas adições, assegurando que o dispositivo periférico fosse identificado. Uma vez configurado, o dispositivo periférico estava pronto para receber conexões de dispositivos centrais.

Quando um dispositivo central se conectava, tinha acesso aos serviços recentemente adicionados e podia ler e escrever nas características do serviço, de forma semelhante ao que faria com qualquer outra característica *BLE*.

As operações de leitura e gravação podiam ser monitorizadas através do terminal do "dbus-monitor", permitindo o acompanhamento das mensagens transmitidas entre os dispositivos.

Estas etapas de configuração desempenham um papel fundamental na habilitação de comunicações Bluetooth de baixa energia em diversas aplicações, incluindo *IoT*, dispositivos vestíveis e automação residencial.

4.3. Análise do tráfego BLE

Nesta secção, será explorada uma análise detalhada do tráfego *Bluetooth Low Energy* (*BLE*) utilizando um sistema Linux. O objetivo principal é examinar como os dispositivos *BLE* interagem durante o processo de varredura (*scan*) e conexão. Para realizar essa análise, segue-se um procedimento cuidadosamente planeado e para o qual foram utilizadas várias ferramentas disponíveis em sistemas *Linux*.

Além disso, é importante destacar que a análise do tráfego *BLE* foi realizada na *Worten* em Aveiro, com a devida autorização da gestora da loja, garantindo a conformidade com todas as políticas e regulamentos aplicáveis.

4.3.1. Preparação do Ambiente

Antes de iniciar a análise do tráfego *BLE*, o ambiente foi configurado para coletar informações precisas e detalhadas. Primeiramente, o *Central BLE (Client)* foi iniciado em *runlevel* 3, de forma a ficar sem ambiente gráfico, e livre das interrupções dos diversos agentes existentes em *runlevel* 5, com o comando *init 3*. De seguida, a captura de pacotes *BLE* foi iniciada através do comando *tcpdump -i bluetooth0 > test.cap*. Posteriormente, o programa de *scan BLE* desenvolvido ao longo do projeto foi executado. Além disso, foi criado um *script* para

monitorizar o uso do *CPU* e da memória *RAM* ao longo do teste. O teste foi desenvolvido em três etapas diferentes.

Este código foi desenvolvido com o propósito de oferecer um meio eficiente de monitorizar e analisar os recursos de um sistema durante todas as fases de teste. É particularmente útil quando se trabalha com dispositivos *BLE*, pois permite a recolha contínua de dados sobre a utilização da *CPU* e da *RAM* ao longo do tempo. Esta informação é crucial para compreender como o sistema se comporta durante a execução de tarefas intensivas em recursos, como a comunicação com dispositivos *BLE*.

O código utiliza a biblioteca *psutil* para aceder a informações detalhadas sobre o desempenho do sistema. A cada 0,2 segundos, regista a utilização da *CPU* e da *RAM* e armazena esses dados num ficheiro *CSV* denominado 'uso_recursos.csv'. Isto possibilita a criação de um histórico completo das métricas de desempenho ao longo do tempo, o que é essencial para a análise de qualquer degradação de desempenho, perdas de recursos ou outros problemas que possam surgir durante os testes *BLE*.

Ao executar este código, os dados são registados de forma contínua até que o utilizador interrompa a execução pressionando *Ctrl+C*. Nesse momento, os dados recolhidos são guardados no ficheiro 'uso_recursos.csv' para posterior análise. Este método de monitorização é fundamental para detetar e resolver problemas de desempenho em sistemas que interagem com dispositivos *BLE*, assegurando assim a fiabilidade e a estabilidade dessas aplicações em ambientes do mundo real. O código é mostrado na Figura 21.

```python
import psutil
import csv
import time

def monitorizar_recursos():
    with open('uso_recursos.csv', mode='w', newline='') as csv_file:
        fieldnames = ['Tempo (s)', 'Uso do CPU (%)', 'Uso de RAM (%)']
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        writer.writeheader()

        segundos = 0
        while True:
            uso_cpu = psutil.cpu_percent(interval=0.2)
            uso_ram = psutil.virtual_memory().percent
            print(f'Tempo: {segundos:.1f}s | Uso do CPU: {uso_cpu:.2f}% | Uso de RAM: {uso_ram:.2f}%')

            writer.writerow({'Tempo (s)': segundos, 'Uso do CPU (%)': uso_cpu, 'Uso de RAM (%)': uso_ram})

            segundos += 0.2

if __name__ == "__main__":
    try:
        monitorizar_recursos()
    except KeyboardInterrupt:
        print("\nMonitorização terminada. Os dados foram salvos no arquivo uso_recursos.csv.")
```

Figura 21 - Código de monitorização

O utilizador interage com o sistema através de uma interface de linha de comandos no *COAP Client*, onde envia pedidos *CoAP* (*Constrained Application Protocol*)) para o *COAP Server*. O *COAP Client* comunica com o *CoAP Server* usando a biblioteca *aiocoap* e a *API CoAP* disponibilizada pelo *COAP Server*. Para fazer a captura de tráfego de pacotes, foi acrescentado um ponto de captura de pacotes entre a *Central BLE (Client)* e os sensores, conforme a Figura 22, permitindo monitorizar as mensagens *BLE* trocadas.

Isto possibilita que o cliente realize várias operações, como obter uma lista de todos os serviços disponíveis, obter informações sobre um serviço específico com base no seu ID, adicionar um novo serviço, remover um serviço existente, contar a quantidade de serviços disponíveis e calcular o espaço em disco utilizado.

A captura de tráfego foi realizada com o auxílio do *Wireshark*, uma ferramenta de análise de protocolos de rede. Esta captura permitiu monitorizar as mensagens *CoAP* trocadas entre a *Central BLE (Client)* e os sensores. A captura de tráfego foi feita especificamente entre a *Central BLE (Client)* e os sensores, permitindo observar a comunicação exata entre esses dois componentes do sistema.

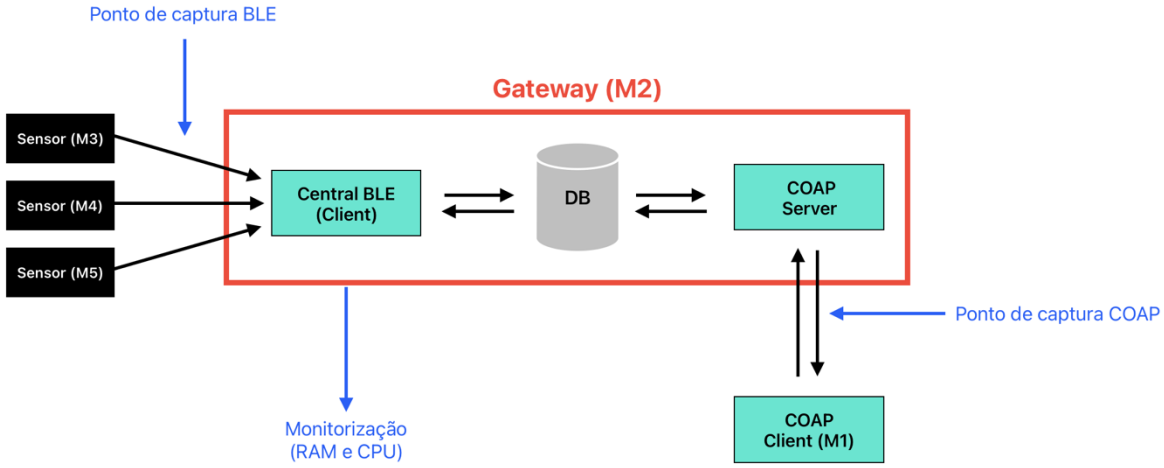

Figura 22 - Ponto de captura de tráfego

Na primeira etapa do teste, que consistiu na monitorização e análise dos pacotes *BLE* durante a fase de *scan*, obtiveram-se os resultados na Figura 23.

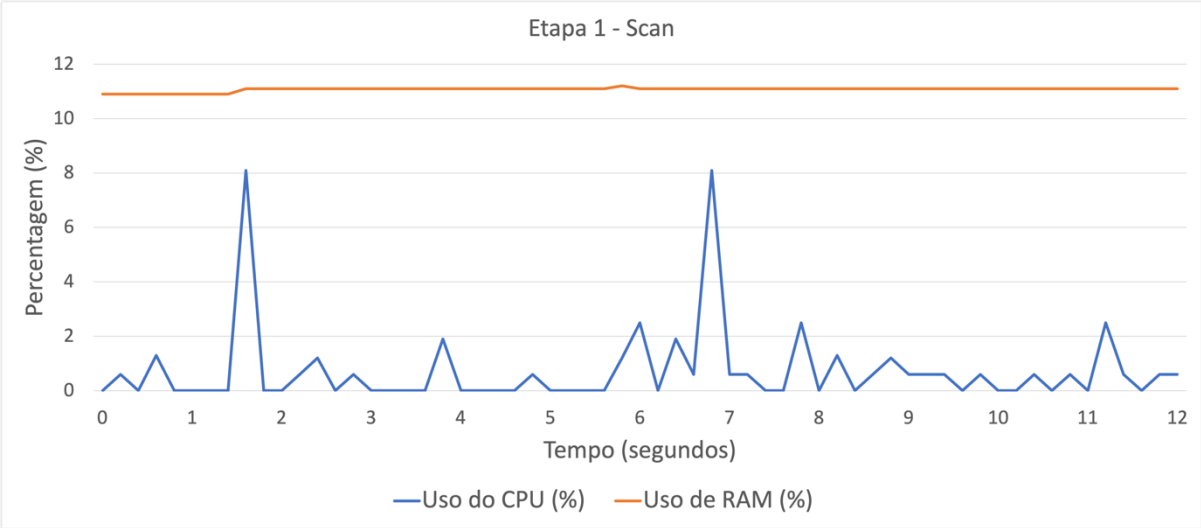

Figura 23 - Etapa 1 – Scan

Durante esta fase, observa-se que o uso do *CPU* do *Central BLE (Client)* variou ao longo do tempo, com picos de atividade em alguns momentos.

Esses resultados indicam que a fase de scan do *BLE* teve um impacto limitado nos recursos do sistema durante a maior parte do tempo. No entanto, notam-se picos de atividade, especialmente em relação ao uso do *CPU*, que podem requerer uma análise mais aprofundada para identificar possíveis causas.

Essa análise é valiosa para entender como o sistema se comporta durante a varredura *BLE* e pode ser usada como base para as próximas etapas do teste, onde a análise será expandida para incluir a fase de conexão e uma análise geral do tráfego *BLE*.

Nos dados apresentados, é possível identificar dois momentos de aumento no uso do *CPU*, evidenciando picos notáveis durante o período de monitorização. Esses picos, caracterizados por valores superiores de "Uso do *CPU* (%)" em relação à média, são indicativos de momentos em que o sistema experimentou uma demanda adicional de processamento.

Na primeira subida, observa-se um aumento repentino no uso do *CPU*, atingindo um pico de 8,1%. Embora o valor possa parecer relativamente alto, quando comparado ao contexto geral dos dados, ele representa um valor insignificante na carga de trabalho do processador. Isso sugere que, durante esse intervalo de tempo, o sistema exigiu mais recursos de CPU para executar determinadas tarefas ou processos.

O segundo pico, que também atinge 8,1% de uso do *CPU*, ocorre entre os 6º e 7º segundos. Novamente, esse valor representa um aumento insignificante, tendo em conta a quantidade de memória disponível em relação ao uso médio do *CPU*. Esse aumento sugere que, durante esse período, o sistema experimentou uma sobrecarga de recursos do *CPU*, possivelmente devido a tarefas mais intensas de processamento ou processos em execução.

Mesmo durante os picos, o uso do *CPU* permanece abaixo de 10%, o que é uma carga relativamente baixa para a maioria dos sistemas modernos. Para este teste foi utilizado um processador *Ryzen* 5 3500U e 8 *GB* de *RAM* para a execução do teste. É certo que o desempenho do computador utilizado vai estar diretamente ligado com a quantidade de recursos disponíveis na máquina. Geralmente, preocupações com desempenho surgem quando o *CPU* se aproxima ou atinge sua capacidade máxima, logo conclui-se o que o processador e memória RAM utilizados foram suficientemente bons.

Os picos de 8,1% de uso do *CPU* são de curta duração, ocorrendo em momentos específicos durante a monitorização. Isso sugere que podem ser eventos transitórios e não representam uma carga constante ou de longo prazo no processador.

A variação no uso do *CPU* é uma ocorrência normal em sistemas computacionais. O sistema operativo e as aplicações em execução podem ocasionalmente exigir mais recursos do *CPU* para tarefas momentâneas, como atualizações de sistema, verificação de segurança ou operações de background.

Se o *CPU* do sistema estiver a operar abaixo de sua capacidade máxima durante a maior parte do tempo, é provável que haja recursos suficientes para lidar com esses picos sem afetar negativamente o desempenho global.

Portanto, com base na análise dos dados e nas razões acima, pode-se concluir que os picos de 8,1% de uso da *CPU* não são motivo de grande preocupação em relação ao desempenho do sistema. Eles não representam uma sobrecarga significativa para o *CPU* e são consistentes com as flutuações normais de uso de recursos em sistemas operativos.

Na segunda etapa do teste, que consistiu na monitorização e análise dos pacotes *BLE* durante a fase de conexão, obtiveram-se os resultados na Figura 24.

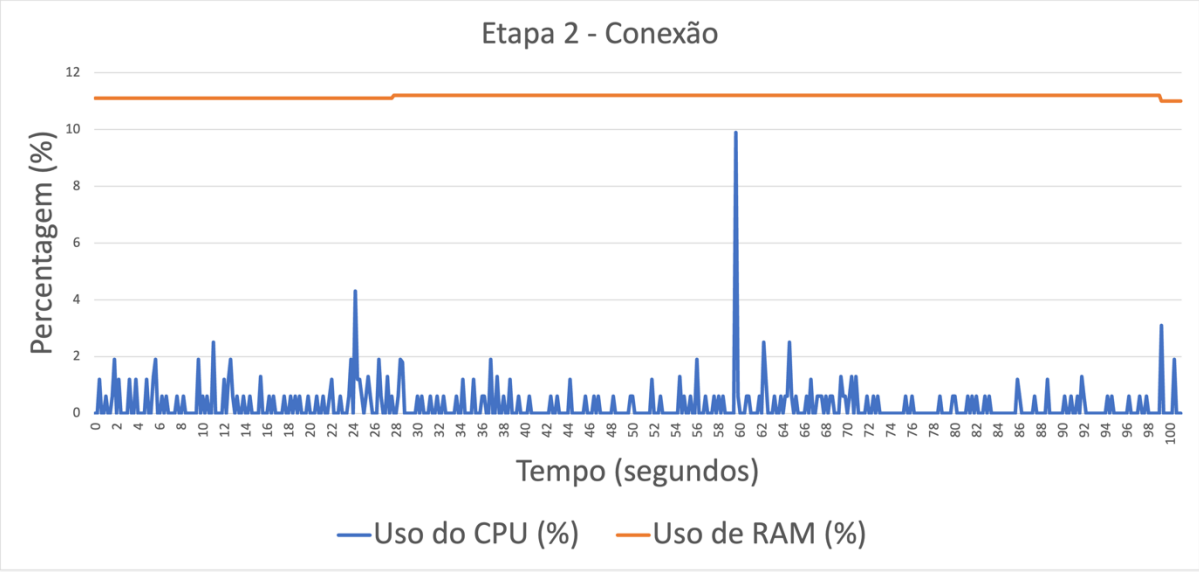

Figura 24 - Etapa 2 - Conexão

Nesta fase, observa-se que o uso do *CPU* e o uso de *RAM* mantiveram-se em níveis semelhantes aos observados na fase anterior (*scan*), com algumas flutuações. No entanto, não há picos significativos de atividade que indiquem um impacto substancial na utilização de recursos durante a fase de conexão *BLE*.

Isso sugere que a fase de conexão *BLE* não sobrecarregou significativamente o sistema em termos de uso de *CPU* e *RAM*. Os resultados parecem ser consistentes com um comportamento normal da conexão *BLE*.

Essa análise é importante para compreender como o sistema responde durante diferentes fases da comunicação *BLE* e pode servir como base para a próxima etapa do teste, que envolverá uma análise geral do tráfego *BLE*.

Na última etapa, a análise foi expandida para incluir a monitorização do tráfego *BLE* durante todo o processo, desde a varredura inicial até a conexão com os dispositivos encontrados, obtiveram-se os resultados na Figura 25.

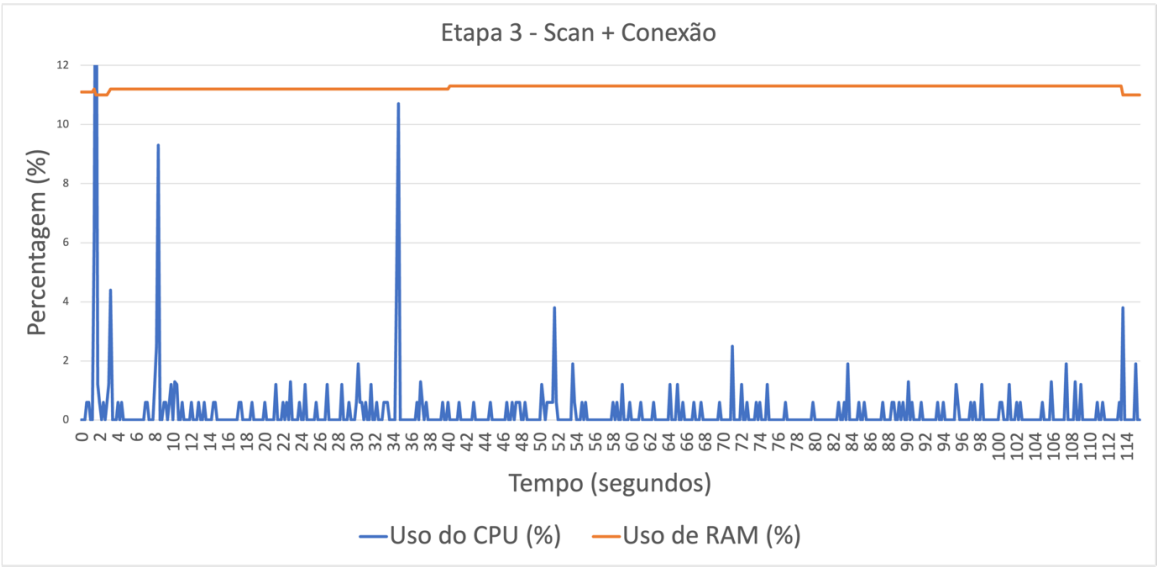

Figura 25 - Etapa 3 - Scan + Conexão

Com base na análise dos dados da fase 3, que consiste em todo os processos num conjunto da fase de scan juntamente com a fase de conexão aos nós *BLE*, observa-se que o uso do *CPU* e o uso de *RAM* mantiveram-se em níveis semelhantes aos observados nas fases anteriores,

com algumas flutuações. Não foram identificados picos significativos de atividade que indiquem um impacto substancial na utilização de recursos durante esta fase.

Isso sugere que a fase 3 não sobrecarregou significativamente o sistema em termos de uso de *CPU* e *RAM*. Os resultados são consistentes com um comportamento normal e estável do sistema durante esta fase.

Essa análise é importante para compreender como o sistema responde ao longo do tempo e durante diferentes fases de operação. Ela fornece informações valiosas sobre o desempenho do sistema e pode ajudar a identificar possíveis problemas de recursos ou anomalias que precisam ser investigados.

No geral, com base nos dados disponíveis, não parece haver preocupações significativas em relação ao uso de recursos durante a fase 3 do sistema.

4.4. Análise da interface COAP

Nesta secção, será abordada a análise detalhada do tráfego capturado durante um teste significativo. O objetivo deste teste foi avaliar o tráfego de dados entre o *COAP Server* e o *COAP Client* desenvolvidos para o projeto através de comunicação *COAP*, com um foco específico em duas etapas distintas: a análise do tráfego *COAP* sem o uso de *CBOR* e a análise do tráfego *COAP* com *CBOR.* O ponto de captura do tráfego foi no local indicado da Figura 22, igualmente feita na experiência anterior de análise de tráfego *BLE*. O local físiico da experiência efetuada corresponde à área designada na *Worten* em Aveiro para a análise de tráfego *COAP*, como previamente autorizado pela gestora da loja na experiência anterior. A captura de tráfego foi realizada com o auxílio do *Wireshark*, uma ferramenta de análise de protocolos de rede. Esta captura permitiu monitorizar as mensagens *CoAP* trocadas entre o *COAP Client* e o *COAP Server*. A captura de tráfego foi feita especificamente entre o *COAP Server* e o *COAP Client*, permitindo observar a comunicação exata entre esses dois componentes do sistema.

4.4.1. Preparação do ambiente

Antes de entrar na análise dos dados capturados, é importante compreender a configuração do teste realizada. O teste foi conduzido da seguinte forma:

O *COAP Server* foi configurado sem nenhum ambiente gráfico, utilizando o comando *init3*. Isso garantiu que o sistema estivesse em *run level 3*, num estado mínimo de recursos, disponibilizando recursos para o teste.

Iniciou-se a captura de pacotes *BLE* utilizando o comando *tcpdump -i wlp2s0 > test.cap*. Este comando direcionou a saída da captura para um arquivo chamado *test.cap*, para posterior análise.

De seguida, foram executados os programas *server.py* e *client.py* desenvolvidos ao longo do projeto.

Durante o teste, também foi também executado um *script* de monitorização de recursos para acompanhar o uso do *CPU* e da memória *RAM* ao longo do teste. O teste foi dividido em duas etapas distintas para avaliar o tráfego *COAP* com e sem o uso de *CBOR*.

Na primeira etapa do teste, o foco estava na análise do tráfego *COAP* utilizando representação no formato de codificação original do *CoAP* (*COAP - JSON*). Durante esta fase, os pacotes de dados foram capturados e posteriormente analisados para identificar os detalhes das comunicações entre o *COAP Server* e o *COAP Client*. Obtiveram-se os resultados na Figura 26.

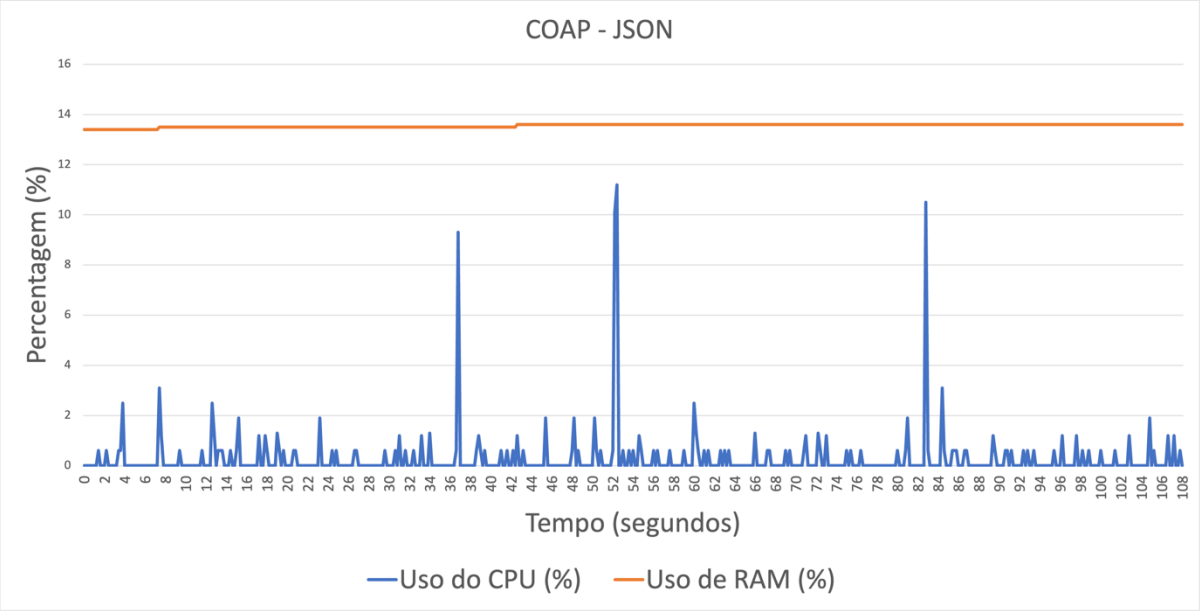

Figura 26 - COAP - JSON

A segunda etapa do teste concentrou-se na análise do tráfego *COAP* com o uso de *CBOR*. Nesta fase, os pacotes de dados capturados foram novamente examinados, com um foco especial na presença e no uso do formato *CBOR* nas comunicações entre *COAP Server* e o *COAP Client*. Obtiveram-se os seguintes resultados:

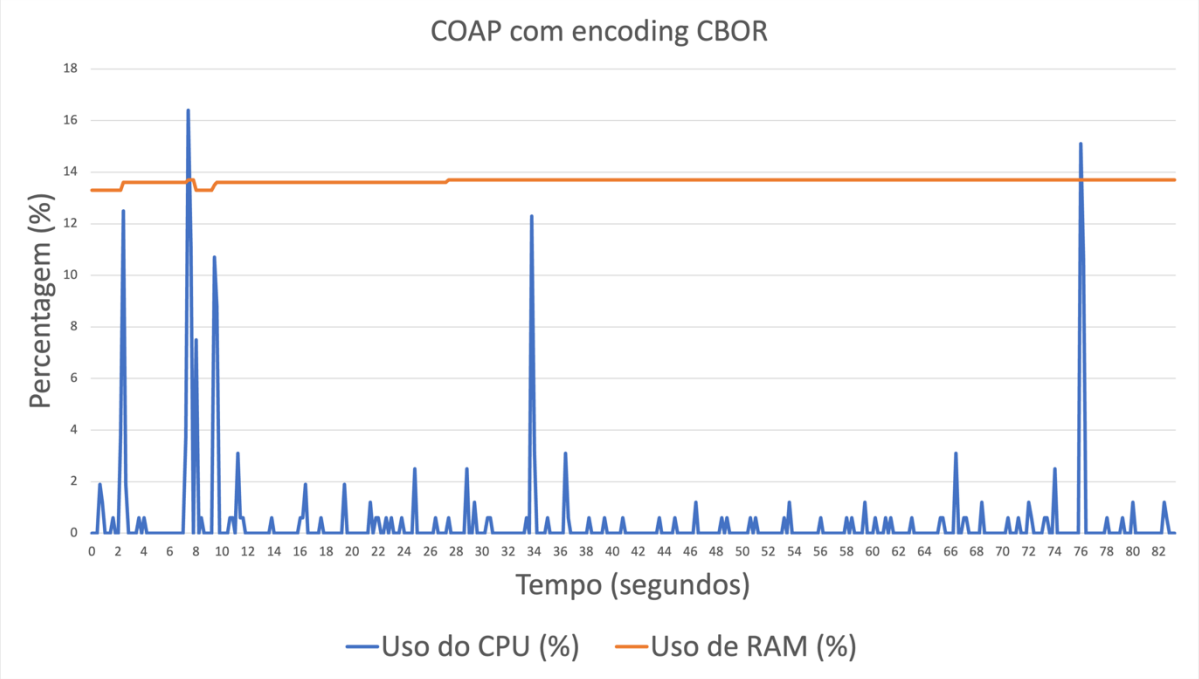

Figura 27 - COAP c/ CBOR

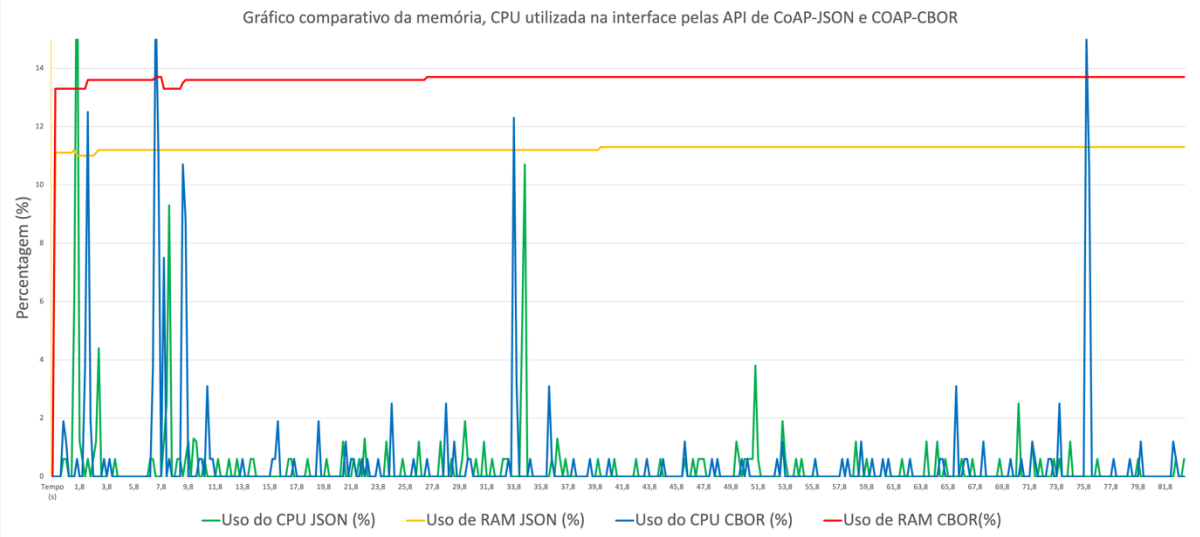

Figura 28 - Gráfico comparativo da memória, CPU utilizada na interface pelas API de CoAP-JSON e COAP-CBOR

Os resultados da Figura 27 e Figura 28 indicam que o tráfego *COAP* com codificação *CBOR* gera um aumento ligeiro na carga de trabalho da *CPU* em comparação com a utilização do formato

de codificação original do *CoAP*. Essa diferença pode ser atribuída, em grande parte, à necessidade de processar mensagens *COAP* no formato de texto, o que envolve uma análise mais detalhada e, por conseguinte, uma maior utilização da *CPU*. Os picos de uso da *CPU* foram mais frequentes quando comparados à configuração formato de codificação original do *CoAP*.

Além disso, o teste também demonstrou que a utilização de RAM é maior na configuração *CBOR*. Isso ocorre porque as mensagens *COAP* em formato *CBOR* não compactado ocupam mais espaço na memória. Além disso, a alocação e liberação de memória ocorrem com mais frequência, o que pode levar a uma maior utilização dos recursos de memória do sistema em comparação com a configuração formato de codificação original do *CoAP*.

4.5. Análise da diferença do uso da interface gráfica

Este relatório tem como objetivo analisar os dados coletados durante um teste de tráfego *COAP* com e sem uma interface gráfica do *COAP Client*, conforme é possível observar na Figura 29. O teste foi conduzido com a finalidade de avaliar a utilização de recursos, como o *CPU* e a *RAM*, nos dois cenários. Os dados coletados foram medidos em intervalos de 0,2 segundos ao longo de um período de 78 segundos.

Os dados estão divididos em duas categorias: sem interface gráfica (*UI*) e com interface gráfica (*UI*). Vão ser analisadas as métricas de uso do *CPU* e *RAM* em ambos os cenários.

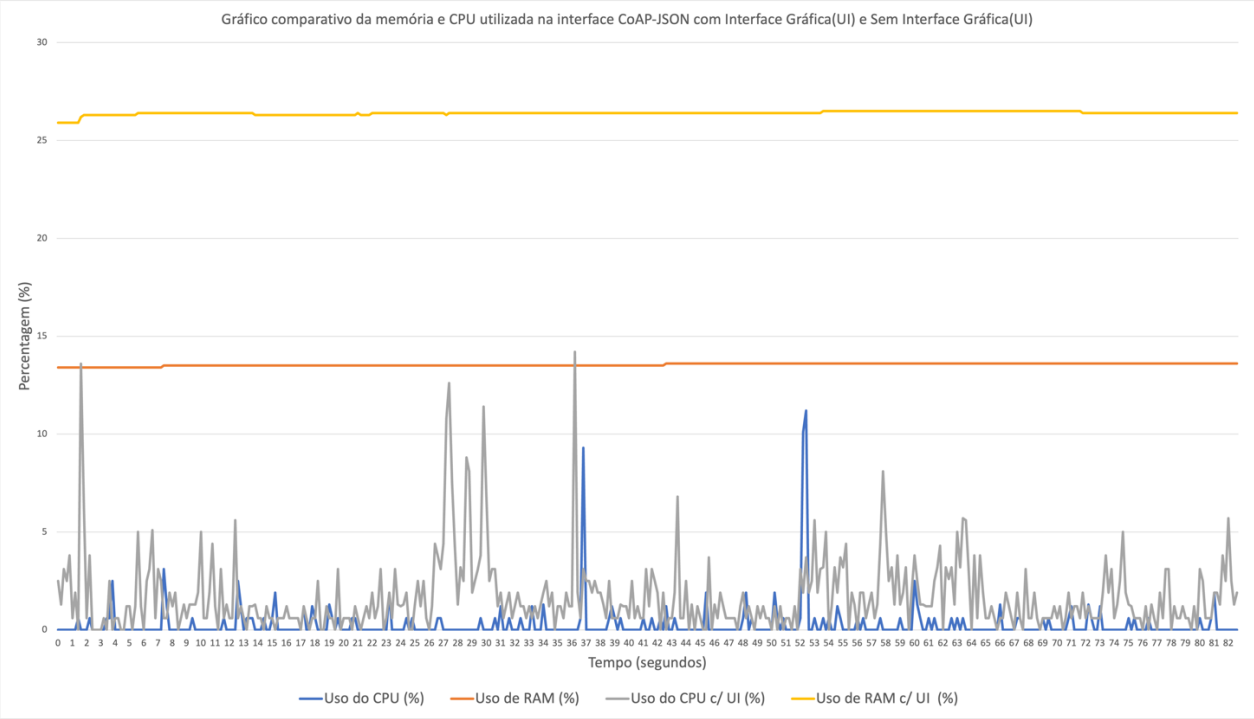

Figura 29 - Com interface gráfica

Durante o teste de tráfego *COAP*, foram observadas diferenças significativas no uso da *CPU* e da *RAM* nos dois cenários distintos. No cenário sem interface gráfica, o uso do *CPU* permaneceu quase negligenciável, mantendo-se próximo a 0% ao longo de todo o teste. Além disso, o uso da *RAM* manteve-se relativamente baixo, em torno de 13,4%. No entanto, quando a interface gráfica estava ativa, o uso do *CPU* variou de 0% a 11,2%, com um pico de 14,2% num ponto específico do teste. O uso da RAM, no mesmo cenário, foi consistentemente mais elevado, mantendo-se em torno de 26,4%. Esses resultados destacam a influência significativa que a presença de uma interface gráfica exerce sobre a utilização de recursos do sistema, sendo fundamental considerar as necessidades específicas do projeto ao decidir pela sua implementação.

É evidente a partir dos dados que a presença de uma interface gráfica no teste de tráfego *COAP* tem um impacto significativo no uso de recursos do sistema. O uso de *CPU* e a utilização da *RAM* foram consideravelmente mais altos quando a *UI* estava ativa.

Esses resultados destacam a importância de otimizar o uso de recursos em sistemas que utilizam o protocolo *COAP*, especialmente se a eficiência dos recursos for uma prioridade.

4.6. Comparação de vantagens do COAP com e sem CBOR

Na contínua procura pela melhoria da eficiência das comunicações na *Internet* das Coisas (*IoT*), a análise de pacotes entre o *COAP-CBOR* e o *COAP-JSON* revelou resultados surpreendentes. Era esperado que o *CBOR*, com sua capacidade de compressão de dados, superasse o formato de codificação original do *CoAP* em termos de otimização do tráfego de rede. No entanto, os resultados demonstraram resultados contrários, destacando a notável eficiência do *COAP-JSON*.

A análise dos pacotes *COAP-CBOR* revelou informações inesperadas, conforme é possível verificar na Tabela 2 - Codificação original COAP VS COAP com encoding CBOR. Numa amostra de 46 pacotes, foram utilizados um total de 26.099 *bytes* para a transmissão de dados. Era esperado que esta codificação, conhecida pela sua eficiência na redução do tráfego de rede, superasse o *COAP-JSON*. No entanto, a realidade surpreendeu, demostrando que o *CBOR*, apesar de sua capacidade de compressão, não conseguiu superar o *COAP-JSON* em termos de economia de *bytes* na transmissão de pacotes.

Tabela 2 - Codificação original COAP VS COAP com encoding CBOR

Measurements	**Codificação original do *CoAP***	**COAP com *encoding CBOR***
Packets	46 (18.6%)	46 (18.6%)
Time span, s	0.287	0.315
Average pps	160.1	146.0
Average packet size, B	567	567
Bytes	26071 (23.9%)	26099 (26.5%)
Average bytes/s	90 k	82 k
Average bits/s	725 k	662 k

Na mesma amostra de 46 pacotes, foram utilizados um total de 26.071 *bytes* para transmitir dados, conforme a Tabela 2 - Codificação original COAP VS COAP com encoding CBOR. Este resultado contrariou as expectativas, já que o *uso do formato de codificação original do CoAP,* através da norma *RFC 7252* (Shelby et al., 2014)*,* conseguiu ser mais eficiente do que o *CBOR*. Surpreendentemente, utilizou menos *bytes* de pacotes, o que ressalta a eficácia na otimização do tráfego de rede.

Em resumo, a experiência de análise de pacotes entre o *COAP-CBOR* e o *COAP-JSON* trouxe resultados inesperados. Esperava-se que o *CBOR*, com sua criação de codificação eficiente, fosse o vencedor claro em termos de economia de *bytes* na rede, mas o *COAP-JSON* surpreendeu ao utilizar menos *bytes* de pacotes em 46 amostras analisadas. Estes resultados indicam que o *COAP-JSON* já integra estratégias de otimização de tráfego, tornando-o uma escolha notável para implementações no *IoT*. Este estudo não só amplia a compreensão e domínio sobre as complexidades das tecnologias de comunicação no *IoT*, mas também fornece uma base sólida para futuras explorações e inovações nesta área em constante evolução.

5. Discussão

Neste capítulo, serão analisados e discutidos os resultados obtidos ao longo do projeto de desenvolvimento de uma gateway agnóstica *COAP/BLE*. Através da análise das informações recolhidas nos testes e experiências realizadas, é possível avaliar as implicações e conclusões que emergem deste processo.

Primeiramente, a análise dos resultados relacionados com a criação de um cenário de testes com dispositivos *BLE* revelou algumas informações sobre a escalabilidade da aplicação *BLE* desenvolvida. A escalabilidade de um programa é um fator crítico em muitas aplicações, incluindo sistemas que envolvem a comunicação com dispositivos *BLE*. Foi possível constatar que o programa funcionava de forma estável e eficaz com 13 nós *BLE*. Isso ocorre porque, com mais dispositivos ao redor, é necessário verificar e rastrear cada um deles, o que consome mais tempo.

Além disso, o número de dispositivos ao redor também afeta o tempo de conexão com cada nó individual, uma vez que a negociação e autenticação de conexões podem tornar-se mais demoradas à medida que o número de dispositivos aumenta. Como resultado, o tempo total necessário para concluir a análise do programa aumenta à medida que mais dispositivos são adicionados à rede.

Essas descobertas são essenciais para dimensionar adequadamente o sistema e compreender as suas limitações. Em cenários nos quais a escalabilidade é fundamental, é importante considerar não apenas o *software*, mas também o hardware subjacente. A alocação adequada de recursos, como *CPUs* mais poderosas e mais *RAM*, pode permitir a expansão do sistema para lidar com um maior número de dispositivos *BLE*, reduzindo assim os atrasos na varredura, na conexão e na análise global.

A análise do impacto do uso do *CBOR* no *COAP Server* demonstrou as vantagens do formato de codificação original do CoAP na redução do tamanho das mensagens e na quantidade de dados transmitidos. Os testes realizados comparando a utilização de *CBOR* com situações em que o *CBOR* não foi utilizado permitiram constatar que o uso do formato de codificação original do CoAP diminuiu de forma maior o número de *bytes* de pacotes necessários para a

realização das operações, o que é crucial para aplicações de *IoT*, onde a eficiência na comunicação é essencial.

A análise do tráfego *BLE* proporcionou resultados relativamente ao comportamento dos dispositivos *BLE* durante o processo de scan e conexão. Através da monitorização do uso do *CPU* e da memória *RAM*, identificou-se que, durante a fase de scan, ocorreram picos de atividade não relevantes, indicando momentos em que o sistema exigiu ligeiramente mais recursos do *CPU* para tarefas específicas.

Na fase de conexão não foram observados picos significativos de atividade que indicassem um impacto substancial na utilização de recursos. Essa constatação sugere que a fase de conexão *BLE* não sobrecarregou significativamente o sistema em termos de uso de *CPU* e *RAM*.

Por último, a análise do tráfego *COAP* com e sem o uso de *CBOR* permitiu comparar o desempenho e o consumo de recursos. Os resultados indicaram que o tráfego *COAP* com *CBOR* aumentou ligeiramente a carga de trabalho do *CPU* em comparação com a utilização do formato de codificação original do *CoAP*. Isso ocorreu provavelmente devido à necessidade de processar mensagens *COAP*, o que envolve uma análise mais detalhada e, consequentemente, um maior uso do *CPU*. Os picos de uso do *CPU* foram mais frequentes em comparação com a configuração formato de codificação original do *CoAP*. Além disso, a utilização de *RAM* foi maior na configuração com *CBOR*.

Em suma, os resultados obtidos durante os testes e análises são fundamentais para a compreensão do desempenho da *gateway COAP/BLE* desenvolvida. O trabalho visou atender às necessidades de comunicação eficiente entre dispositivos *IoT* que usam *BLE* e protocolos *COAP*, e os resultados dessas análises são passos importantes em direção a esse objetivo.

No contexto do trabalho futuro, é também importante analisar como o sistema se comportará à medida que cresce em escala. Um ponto crítico de discussão é como o tempo de *scan* será afetado quando se atingir cerca de duzentos dispositivos. É esperado que esse tempo de *scan* aumente à medida que mais dispositivos são adicionados em redor.

No entanto, é encorajador observar que, até o momento, o sistema demonstra um funcionamento aceitável, o que permite inferir que a escalabilidade não parece ser um problema insuperável. Embora seja inevitável que o tempo de *scan* aumente com a adição do

número de dispositivos, a avaliação contínua e otimizações no software e hardware podem ajudar a atenuar os impactos negativos desse crescimento. Contudo seria interessante ter conseguido utilizar mais nós *BLE* de modo a testar a escalabilidade ao limite máximo.

O trabalho futuro deve ser direcionado não apenas para avaliar o comportamento do sistema em cenários de maior escala, mas também para implementar melhorias que permitam lidar com os desafios inerentes ao crescimento.

Por fim, a segurança e privacidade dos dados dos utilizadores devem ser abordadas. É fundamental garantir que os dados sejam armazenados e transmitidos com segurança, implementando medidas rigorosas de segurança e respeitando as considerações de privacidade dos utilizadores. Seria extremamente proveitoso incorporar esta tecnologia no projeto em desenvolvimento. A utilização do *ACE* ofereceria uma camada adicional de segurança, um melhor controlo de acesso e escalabilidade, tornando o projeto mais robusto e preparado para enfrentar os desafios do *IoT*. A capacidade do *ACE* de permitir a autorização delegada em dispositivos *IoT* com recursos limitados é especialmente valiosa, uma vez que promove a segurança e a interoperabilidade. Isso garantiria que os dispositivos *IoT* pudessem autenticar-se de maneira eficaz, estabelecer conexões seguras e aceder a recursos com base em políticas de autorização predefinidas.

Além disso, o *ACE* pode ser particularmente benéfico em cenários nos quais a segurança e a privacidade dos dados são fundamentais, como em projetos de casas inteligentes ou cidades inteligentes. Portanto, a implementação do *ACE* no projeto em desenvolvimento ajudaria a criar uma base sólida para a comunicação e a gestão de dispositivos *IoT*, bem como garantir a integridade e a confidencialidade dos dados.

Uma medida útil a ser implementada no projeto seria a introdução de um esquema de autenticação de mensagens *CoAP* baseado em *REST*. Isso ajudaria a estabelecer uma camada adicional de segurança orientada para mensagens no contexto do *CoAP*. Essa medida específica envolveria a geração e verificação de assinaturas de mensagens *CoAP*.

Além disso, considerando a importância da integridade e autenticidade das mensagens em ambientes *IoT*, a implementação dessa medida ajudaria a proteger contra ameaças e garantir

que as informações transmitidas sejam seguras e confiáveis. Portanto, essa ação específica aumentaria a segurança do projeto no contexto do *CoAP* e do *IoT*.

6. Conclusões

Neste capítulo, são apresentadas as conclusões derivadas do projeto de desenvolvimento da gateway agnóstica *COAP*/*BLE*. Com base na análise dos resultados e nas observações feitas ao longo deste trabalho, é possível traçar um panorama das principais constatações e lições aprendidas.

A principal motivação deste projeto era a necessidade de criar uma *gateway COAP*/*BLE* agnóstica em termos de modelo de dados, capaz de fornecer comunicação eficiente entre dispositivos *IoT* que utilizam o padrão *BLE* e a infraestrutura da *Internet das Coisas*. A integração de dispositivos *BLE* numa infraestrutura de *IoT* é uma tarefa complexa devido às diferenças nos modelos de dados e formatos de mensagens. No entanto, a criação da gateway *COAP/BLE* demonstrou ser uma solução viável para esse desafio. A *gateway* foi projetada para funcionar como um intermediário flexível entre dispositivos *BLE* e a plataforma de *IoT*, permitindo a comunicação sem considerar a origem dos dados.

A implementação da *gateway* incluiu a capacidade de utilização de dados em formatos *CBOR* e representação no formato de codificação original do *CoAP*. A inclusão da codificação *COAP* revelou-se benéfica, uma vez que resultou em mensagens mais compactas e, consequentemente, numa eficiência de comunicação aprimorada. Isso é particularmente importante em cenários de *IoT*, onde a largura de banda e a eficiência na transmissão de dados são fundamentais. Os testes realizados comprovaram que o formato de codificação original do CoAP é uma escolha eficaz para otimizar a comunicação entre dispositivos *BLE* e a *gateway*.

Um exemplo prático de aplicação para a *gateway COAP*/*BLE* é o contexto de um ginásio. Imaginando um ginásio equipado com sensores *BLE* incorporados em pulseiras usadas por frequentadores. Esses sensores não possuem conectividade direta à Internet, mas coletam dados de atividades físicas e parâmetros de saúde. Os nós *BLE* são fundamentais na comunicação de dados em sistemas de rede sem fio. Estes dispositivos desempenham um papel crucial na coleta e transmissão de informações em várias aplicações, como *IoT*, rastreamento e monitorização de sensores. A integração destes nós com a *Central BLE (Client)*

e a base de dados é essencial para garantir uma comunicação eficaz dentro de uma arquitetura BLE.

O *Central BLE (Client)* atua como um intermediário vital, recolhendo e gerindo os dados provenientes dos nós *BLE*. É responsável por estabelecer conexões com os nós, recolher informações e interagir com a base de dados. Por sua vez, a *API CoAP* é uma parte crucial deste processo, pois define como os dados são formatados, transmitidos e processados.

A combinação da *Central BLE (Client),* da base de dados e do *COAP Server* constitui a chamada "Gateway". Esta Gateway é o ponto central de coordenação e controlo de todo o fluxo de dados provenientes dos nós *BLE*. Desempenha um papel crítico no encaminhamento de informações para a nuvem, garantindo que os dados recolhidos pelos nós sejam processados e armazenados de forma adequada.

Posteriormente, a *COAP Client* interage com o *Gateway BLE* desenvolvida que, por sua vez, disponibiliza a *API CoAP* para enviar ou obter dados para a base de dados. Esta integração entre o *COAP Client* e o *COAP Server* é um elemento-chave na jornada dos dados, permitindo que sejam disponibilizados na base de dados para análise, armazenamento, processamento e partilha. Isso desempenha um papel fundamental na capacidade de aceder e utilizar esses dados de forma significativa.

Em resumo, os nós *BLE*, o *Central BLE (Client)*, a base de dados e o *COAP Server* trabalham em conjunto para estabelecer um sistema eficiente de coleta e transmissão de dados. A *Gateway BLE*, composta pela *Central BLE (Client)*, a base de dados e o *COAP Server*, é o componente-chave que permite a integração e o fluxo contínuo de informações entre os dispositivos no terreno e a base de dados, desempenhando um papel crucial na capacitação de sistemas *IoT* e soluções de monitorização baseadas em *BLE*.

Essa aplicação demonstra o potencial da *Gateway BLE* em cenários do mundo real. Num ginásio moderno, as pulseiras de fitness equipadas com nós *BLE* incorporados são distribuídas pelos membros. Estas pulseiras registam uma variedade de informações, como a frequência cardíaca do utilizador, a distância percorrida, as calorias queimadas e até mesmo o tempo gasto em exercícios específicos.

O sistema funciona da seguinte forma: Cada membro do ginásio recebe uma pulseira de fitness equipada com um nó *BLE*. As pulseiras recolhem constantemente dados sobre a atividade física dos utilizadores.

No ginásio, existe uma unidade central equipada com uma *Gateway BLE*. Ela recolhe os dados das pulseiras dos membros à medida que estes circulam pelo ginásio, incluindo detalhes sobre o desempenho dos membros e os locais específicos onde se encontram dentro do ginásio.

A *gateway* possui também uma *API CoAP* implementada no *COAP Server* que permite a comunicação com o cliente. A *API CoAP* processa os dados recolhidos e disponibiliza-os. A combinação da *Central BLE (Client)*, da base de dados e do *COAP Server* é chamada de *Gateway BLE* do ginásio.

Neste exemplo, fica demonstrado como os nós *BLE*, as pulseiras de fitness, podem melhorar a experiência dos utilizadores em ambientes como ginásios, ao permitir a monitorização em tempo real.

Em resumo, o desenvolvimento da gateway *COAP/BLE* representa uma solução eficaz para desafios de integração de dispositivos *BLE* em ambientes de *IoT*. À medida que a Internet das Coisas continua a evoluir, a flexibilidade e a eficiência da comunicação oferecidas por essa *gateway* têm o potencial de atender às crescentes requisições por conectividade entre dispositivos *IoT* de diferentes naturezas.

O grande diferencial desta *gateway* é a sua natureza agnóstica em relação ao modelo de dados. Isso significa que ela é altamente adaptável e flexível, podendo comunicar com uma variedade de sensores e dispositivos. Seja para monitorar a frequência cardíaca, medir a temperatura corporal, ou coletar dados de movimento, a *Gateway* é capaz de aceitar e transmitir informações a partir de diferentes fontes, tornando-se um elemento versátil e capaz de atender a diversas aplicações.

Referências

Al Enany, M. O., Harb, H. M. & Attiya, G. (2021). A comparative analysis of MQTT and IoT application protocols. *ICEEM 2021 - 2nd IEEE International Conference on Electronic Engineering*. https://doi.org/10.1109/ICEEM52022.2021.9480384

Andy Lee. (2020). *Creating a BLE Peripheral with BlueZ | Punch Through*. https://punchthrough.com/creating-a-ble-peripheral-with-bluez/

Beltran, V. & Skarmeta, A. F. (2017). An overview on delegated authorization for CoAP: Authentication and authorization for Constrained Environments (ACE). *2016 IEEE 3rd World Forum on Internet of Things, WF-IoT 2016*, 706–710. https://doi.org/10.1109/WF-IOT.2016.7845482

bleak — bleak 0.21.1 documentation. (sem data). Obtido 25 de Outubro de 2023, de https://bleak.readthedocs.io/en/latest/

Costantino, L., Buonaccorsi, N., Cicconetti, C. & Mambrini, R. (2012). Performance analysis of an LTE gateway for the IoT. *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2012 - Digital Proceedings*. https://doi.org/10.1109/WOWMOM.2012.6263789

Iglesias-Urkia, M., Casado-Mansilla, D., Mayer, S., Bilbao, J. & Urbieta, A. (2019). Integrating Electrical Substations Within the IoT Using IEC 61850, CoAP, and CBOR. *IEEE Internet of Things Journal*, *6*(5), 7437–7449. https://doi.org/10.1109/JIOT.2019.2903344

Jatana, N., Puri, S., Ahuja, M., Kathuria, I. & Gosain, D. (2012). A Survey and Comparison of Relational and Non-Relational Database. *International Journal of Engineering Research & Technology*, *1*(6). https://doi.org/10.17577/IJERTV1IS6024

Kang, H.-W., Kim, C.-M. & Koh, S.-J. (2016). ISO/IEEE 11073-Based Healthcare Services over IoT Platform Using 6LoWPAN and BLE: Architecture and Experimentation. *2016 International Conference on Networking and Network Applications (NaNA)*, 313–318. https://doi.org/10.1109/NaNA.2016.26

Kim, H.-S., Seo, J.-S. & Seo, J.-W. (2015). Performance Evaluation of a Smart CoAP Gateway for Remote Home Safety Services. *KSII Trans. Internet Inf. Syst.*, *9*(8), 3079–3089. https://doi.org/10.3837/TIIS.2015.08.019

Lin, C.-Y., Liao, K.-H. & Chang, C.-H. (2018). An Experimental System for MQTT/CoAP-based IoT Applications in IPv6 over Bluetooth Low Energy. *JUCS - Journal of Universal Computer Science 24(9): 1170-1191*, *24*(9), 1170–1191. https://doi.org/10.3217/JUCS-024-09-1170

Matthias Nefzger. (2021). *Talk CoAP to me – IoT over Bluetooth Low Energy | MaibornWolff*. https://www.maibornwolff.de/en/know-how/talk-coap-me-iot-over-bluetooth-low-energy/

Morabito, R. & Jimenez, J. (2020). IETF Protocol Suite for the Internet of Things: Overview and Recent Advancements. *IEEE Communications Standards Magazine*, *4*(2), 41–49. https://doi.org/10.1109/MCOMSTD.001.1900014

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*. https://doi.org/10.1109/SYSENG.2017.8088251

Nguyen, H. V. & Iacono, L. Lo. (2016). REST-ful CoAP Message Authentication. *Proceedings - 2015 International Workshop on Secure Internet of Things, SIoT 2015*, 35–43. https://doi.org/10.1109/SIOT.2015.8

oliveirasmj/scanBle. (sem data). Obtido 25 de Outubro de 2023, de https://github.com/oliveirasmj/scanBle

RFC 7252 - The Constrained Application Protocol (CoAP). (sem data). Obtido 24 de Outubro de 2023, de https://datatracker.ietf.org/doc/html/rfc7252

Shelby, Z., Hartke, K. & Bormann, C. (2014). *The Constrained Application Protocol (CoAP)*. https://doi.org/10.17487/RFC7252

Shin, I. J., Eom, D. S. & Song, B. K. (2016). The CoAP-based M2M gateway for distribution automation system using DNP3.0 in smart grid environment. *2015 IEEE International*

Conference on Smart Grid Communications, SmartGridComm 2015, 713–718. https://doi.org/10.1109/SMARTGRIDCOMM.2015.7436385

Stanford-Clark, A. & Truong, H. L. (2013). *MQTT For Sensor Networks (MQTT-SN) Protocol Specification Version 1.2*.

Thangavel, D., Ma, X., Valera, A., Tan, H. X. & Tan, C. K. Y. (2014). Performance evaluation of MQTT and CoAP via a common middleware. *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*. https://doi.org/10.1109/ISSNIP.2014.6827678

Uy, N. Q. & Nam, V. H. (2019). A comparison of AMQP and MQTT protocols for Internet of Things. *2019 6th NAFOSTED Conference on Information and Computer Science (NICS)*, 292–297. https://doi.org/10.1109/NICS48868.2019.9023812

Vasseur, J.-P. & Dunkels, A. (2010). The 6LoWPAN Adaptation Layer. *Interconnecting Smart Objects with IP*, 231–250. https://doi.org/10.1016/B978-0-12-375165-2.00016-8

Welcome to Python.org. (sem data). Obtido 25 de Outubro de 2023, de https://www.python.org/