Universidade Federal de Uberlândia
Faculdade de Engenharia Elétrica
Pós-graduação em Engenharia Elétrica

Willian Douglas Caixeta Nunes

# Proposta e Desenvolvimento de um Sistema de gerência de Dispositivos de Rede Restritos baseados em LoRaWAN e LwM2M

Uberlândia
2020

Willian Douglas Caixeta Nunes

# Proposta e Desenvolvimento de um Sistema de gerência de Dispositivos de Rede Restritos baseados em LoRaWAN e LWM2M

Dissertação de mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como exigência parcial para a obtenção do título de mestre em Ciências.

Orientador: Prof. Dr. Alan Petrônio Pinheiro

WILLIAN DOUGLAS CAIXETA NUNES

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

N972 Nunes, Willian Douglas Caixeta, 1993-
2020 Proposta e desenvolvimento de um sistema de gerência de dispositivos de redes restritos baseados em LoRaWAN e LwM2M [recurso eletrônico] / Willian Douglas Caixeta Nunes. - 2020.

Orientador: Alan Petrônio Pinheiro.
Dissertação (Mestrado) - Universidade Federal de Uberlândia, Pós-graduação em Engenharia Elétrica.
Modo de acesso: Internet.
Disponível em: http://doi.org/10.14393/ufu.di.2020.254
Inclui bibliografia.
Inclui ilustrações.

1. Engenharia elétrica. I. Pinheiro, Alan Petrônio,1982-, (Orient.). II. Universidade Federal de Uberlândia. Pós-graduação em Engenharia Elétrica. III. Título.

CDU: 621.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:
Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

# PROPOSTA E DESENVOLVIMENTO DE UM SISTEMA DE GERÊNCIA DE DISPOSITIVOS DE REDE RESTRITOS BASEADOS EM LORAWAN E LWM2M

Dissertação de mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como exigência parcial para a obtenção do título de mestre em Ciências.

Banca Examinadora:

| | |
|---|---|
| Alan Petrônio Pinheiro (orientador), Dr. – UFU | Flávia Coimbra Delicato, Dra. – UFF |
| Daniel Pereira de Carvalho, Dr. – UFU | Marcelo Barros de Almeida, Dr. – UFU |

Uberlândia, 19 de fevereiro de 2020

**UNIVERSIDADE FEDERAL DE UBERLÂNDIA**
Coordenação do Programa de Pós-Graduação em Engenharia Elétrica
Av. João Naves de Ávila, 2121, Bloco 3N - Bairro Santa Mônica, Uberlândia-MG, CEP 38400-902
Telefone: (34) 3239-4707 - www.posgrad.feelt.ufu.br - copel@ufu.br

# ATA DE DEFESA - PÓS-GRADUAÇÃO

| | | | | | |
|---|---|---|---|---|---|
| Programa de Pós-Graduação em: | Engenharia Elétrica | | | | |
| Defesa de: | Dissertação de Mestrado Acadêmico, 732, PPGEELT | | | | |
| Data: | Dezenove de fevereiro de dois mil e vinte | Hora de início: | 09:00 | Hora de encerramento: | 11:55 |
| Matrícula do Discente: | 11812EEL011 | | | | |
| Nome do Discente: | Willian Douglas Caixeta Nunes | | | | |
| Título do Trabalho: | Proposta e desenvolvimento de um sistema de gerência de dispositivos de rede restritos baseados em LORAWAN e LWM2M | | | | |
| Área de concentração: | Processamento da informação | | | | |
| Linha de pesquisa: | Processamento digital de sinais | | | | |
| Projeto de Pesquisa de vinculação: | Título: Estudo e desenvolvimento piloto de novos modelos de serviços e infraestrutura de TIC voltados ao uso de antenas de telecomunicações da rede de distribuição da CEB alinhados ao cenário de SG e IoT Agência Financiadora: CEB/Aneel Início 23/11/18 Término 22/5/2021 MESTRADO No. do Projeto na agência: PD-05160-1805/2018 Professor Coordenador: Alan Petrônio Pinheiro | | | | |

Reuniu-se no Anfiteatro 1E, Campus Santa Mônica, da Universidade Federal de Uberlândia, a Banca Examinadora, designada pelo Colegiado do Programa de Pós-graduação em Engenharia Elétrica, assim composta: Professores Doutores: Marcelo Barros de Almeida - FEELT/UFU; Daniel Pereira de Carvalho - FEELT/UFU; Flávia Coimbra Delicato - UFF; Alan Petrônio Pinheiro - FEELT/UFU, orientador(a) do(a) candidato(a).

Iniciando os trabalhos o presidente da mesa, professor Alan Petrônio Pinheiro, apresentou a Comissão Examinadora e o candidato e concedeu ao Discente a palavra para a exposição do seu trabalho. A duração da apresentação do discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir o senhor(a) presidente concedeu a palavra, pela ordem sucessivamente, aos(às) examinadores(as), que passaram a arguir o(a) candidato(a). Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o(a) candidato(a):

Aprovado(a).

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre.

O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme foi assinada pela Banca Examinadora.

Documento assinado eletronicamente por **Alan Petronio Pinheiro**, **Professor(a) do Magistério Superior**, em 19/02/2020, às 15:33, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Daniel Pereira de Carvalho**, **Professor(a) do Magistério Superior**, em 19/02/2020, às 15:52, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Flavia Coimbra Delicato**, **Usuário Externo**, em 20/02/2020, às 14:03, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Marcelo Barros de Almeida**, **Professor(a) do Magistério Superior**, em 20/02/2020, às 15:36, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **1825763** e o código CRC **627A787F**.

**Referência:** Processo nº 23117.004246/2020-09 SEI nº 1825763

# Agradecimentos

Primeiramente, aos meus pais, pelo incentivo e dedicação em me auxiliar na finalização deste projeto.

Ao meu orientador, pela enorme dedicação e presença nos esclarecimentos das minhas dúvidas, além das muitas contribuições para com o projeto, o que tornou possível a realização dessa pesquisa.

Por fim, agradeço aos colegas de laboratório, pela companhia e ajuda ao longo desta pesquisa.

*Pesquisa realizada na:*

Universidade Federal de Uberlândia
Programa de pós-graduação em Engenharia Elétrica

CePEDRI - Centro de P&D em Redes Inteligentes
www.cepedri.ufu.br

*Financiamento e apoio:*

Agência Nacional de Energia Elétrica
Programa de Pesquisa e Desenvolvimento

Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

Esta pesquisa foi financiada pela Companhia Energética de Brasília (CEB), por meio do projeto de P&D ANEEL nº 05160-1805/2018 e pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES – processo 001)

# Resumo

Este trabalho propõe um sistema de gerenciamento da comunicação de dispositivos restritos. Foi desenvolvido uma série de funcionalidades que podem ser utilizadas em diferentes dispositivos que possuam recursos limitados, como tipicamente se emprega em IoT. A solução criada foi utilizada no contexto de redes *Low Power Wide Area Network* (LPWAN), caracterizadas por possuir um grande alcance, baixo consumo energético e limitações de taxa de dados. LoRa é umas das principais tecnologias desta categoria. Por isso foi empregada aqui como referência de uso. O sistema de gerenciamento proposto emprega o protocolo *Lightweight Machine to Machine* (LwM2M) como padrão de fluxo da informação. Ele define um modelo de comunicação e de dados entre clientes (dispositivos) e servidores, tanto para operações de gerência de rede quanto para o suporte ao tráfego de dados entre aplicações e dispositivos. A solução apresentada é validada em um cenário com dispositivos LoRa que utilizem o protocolo LoRaWAN. As principais funcionalidades sugeridas foram implementadas e testadas em diferentes cenários, para ilustrar a eficácia da proposta no contexto de redes restritas. Os resultados mostram que o sistema desenvolvido permite executar as principais operações de gerência seguindo o padrão LwM2M, como também provê uma série de informações que auxiliam em intervenções na rede e contribuem para o seu bom funcionamento. O sistema criado pode ser utilizado em diferentes aplicações dentro do contexto de IoT e em diferentes tecnologias de comunicação.

**Palavras-chave**: LwM2M, LoRa, LoRaWAN, Gerenciamento de dispositivos, IoT.

# Abstract

This work proposes a communication management system for restricted devices. Several features have been developed that can be used on different devices that have limited resources, as is typically used in IoT. The created solution was used in the context of Low Power Wide Area Network (LPWAN) networks, characterized by having a long range, low energy consumption and data rate limitations. LoRa is one of the main technologies in this category. That is why it was used here as a reference for use. The proposed management system uses the Lightweight Machine to Machine (LwM2M) protocol as an information flow standard. It defines a communication and data model between clients (devices) and servers, both for network management operations and for supporting data traffic between applications and devices. The presented solution is validated in a scenario with LoRa devices that use the LoRaWAN protocol. The main functionalities suggested were implemented and tested in different scenarios, to illustrate the effectiveness of the proposal in the context of restricted networks. The results show that the developed system allows the execution of the main management operations following an LwM2M standard, as well as providing a series of information that assist in interventions on the network and contribute to its correct functioning. The created system can be used in different applications within the context of IoT and in different communication technologies.

**Keywords**: LwM2M, CoAP, LoRa, LoRaWAN, Device management, IoT.

# Lista de figuras

# Lista de tabelas

# Lista de abreviaturas e siglas

| | |
|---|---|
| ABP | *Activation by Personalization* |
| ADR | *Adaptative Data Rate* |
| AMQP | *Advanced Message Queuing Protocol* |
| API | *Application Programming Interface* |
| CoAP | *Constraint Application Protocol* |
| CSMA/CA | *Carrier Sense Multiple Access With Collision Avoidance* |
| CSS | *Chirp Spread Spectrum* |
| DUT | *Device Under Test* |
| ETSI | *European Telecommunications Standards Institute* |
| GSN | *Global Sensor Network* |
| HTTP | *Hypertext Transfer Protocol* |
| JSON | *JavaScript Object Notation* |
| LNS | LoRaWAN *Network Server* |
| DDS | *Data Distribution Service* |
| IEEE | *Institute of Electrical and Electronics Engineers* |
| IETF | *Internet Engineering Task Force* |
| IoT | *Internet of Things* |
| IP | *Internet Protocol* |
| IPSO | *Internet Protocol for Smart Objects* |
| IPv6 | *Internet Protocol version 6* |
| ITU | *International Telecommunication Union* |
| I2C | *Inter-Integrated Circuit* |
| LNS | *LoRaWAN network server* |
| LPWAN | *Low Power Wide Area Networks.* |
| LTE-*M* | *Long term Evolution - Machine Type Communication* |
| LoRa | *Longe Range* |
| LwM2M | *Lightweight Machine to Machine* |
| KPI | *Key Parameters Indicator* |
| MCU | *Microcontrolador* |
| MQTT | *Message Queuing Telemetry Transport* |
| MQTT-*SN* | *Message Queuing Telemetry Transport - Sensor Network* |
| M2M | *Machine to Machine* |

| | |
|---|---|
| NB-IoT | *Narrow Band Internet of Things* |
| NGSI | *Next Generation Service Interfaces* |
| OPC-UA | *Open Platform Communications Unified Architecture* |
| OSI | *Open System Interconnection* |
| OTAA | *Over The Air Activation* |
| PCHA | *Personal Connected Health Alliance* |
| PDR | *Package Delivery Ratio* |
| QoS | *Quality of Service* |
| REST | *Representational State Transfer* |
| RPL | *Routing Protocol for Low-Power and Lossy Networks* |
| RSSI | *Receive Signal Strength Indicator* |
| SAAS | *Software as a Service* |
| SF | *Spreading Factor* |
| SNMP | *Simple Network Management Protocol* |
| SNR | *Signal Noise Ratio* |
| SPI | *Serial Peripheral Interface* |
| SoA | *Service-Oriented Architecture* |
| SAWSDL | *Semantic Annotation for Web Service Description Language* |
| OMA | *Open Mobile Alliance* |
| TCP | *Transmission Control Protocol* |
| TTN | *The Things Network* |
| UDP | *User Datagram Protocol* |
| XML | *Extensible Markup Language* |
| XMPP | *Extensible Messaging and Presence Protocol* |
| WPAN | *Wireless Personal Area Network* |
| WS | *Web Services* |
| 3GPP | *3rd Generation Partnership Project* |
| 6LoWPAN | IPv6 *over Low power Wireless Personal Area Networks* |

# Sumário

# 1 Introdução

## 1.1 Apresentação

Internet das Coisas (IoT) é um mercado importante e promissor, como ilustra a pesquisa feita em [1]. Ela prevê que em 2030 haverá cerca de 500 bilhões de dispositivos conectados à internet. Uma outra pesquisa [2] feita pela mesma empresa aponta que cerca de 74% dos projetos IoT tiveram algum tipo de problema, seja no processo de desenvolvimento ou planejamento. Isto demostra a complexidade de sistemas IoT, que é uma infraestrutura complexa que envolve diversos componentes, como microcontroladores, sensores, *middlewares*, protocolos, tecnologias de transmissão, virtualizações, processamentos da informação (*data analytics*) etc. Além do mais, a IoT emprega com frequência dispositivos com recursos limitados, que pode estar no poder de processamento do dispositivo e da memória ou no enlace de comunicação; aqui, portanto, está um dos desafios de sua implantação real.

Existem protocolos e tecnologias bem estabelecidas em IoT, como as providas pelo 3GPP (3rd *Generation Partnership Project*), Wi-Fi, Bluetooth, ZigBee, dentre outros. Porém, com a necessidade de que redes IoT alcancem distâncias maiores com um menor consumo de potência, surgiram as redes chamadas LPWAN (*Low Power Wide Area Networks*). As principais tecnologias representantes deste tipo de rede são a LoRa, a SigFox e a NB-IoT, que podem ser aplicadas em situações de medições remotas, na agricultura, na *smart grids* etc.

A tendência natural é que aumente a quantidade de dispositivos IoT em diferentes tipos de aplicação, com cada uma delas usando uma tecnologia de comunicação mais propícia. Por exemplo, dispositivos que se dedicam a automatizar casas (*Smart Homes*), em geral, tendem a utilizar o Wi-Fi, enquanto que aplicações que exigem grandes coberturas de área e baixo consumo de energia tendem a utilizar o LoRa. Comum a estas

aplicações está a necessidade, quase que imperativa, de fornecer ao provedor da rede a capacidade de gerenciamento de seus dispositivos de forma remota, que ajuntado ao aumento no número de dispositivos, faz surgir a necessidade maior de que se tornem capazes de serem gerenciados. Operações como *reset*, atualização de *firmware*, troca de parâmetros de rede, dentre outros, são importantes para que os dispositivos continuem operando durante um grande período e de forma estável, além de permitirem ao operador do sistema saber a "saúde" da infraestrutura IoT implantada.

As redes que trabalham com dispositivos IP são relativamente consolidadas quanto as funções de gerência dos dispositivos e da própria rede. Contudo estas novas tecnologias que apresentam restrições ao uso do IP (como é o caso da tecnologia LoRa) não são simples e devem seguir características específicas que visam contornar as limitações do dispositivo (ou da tecnologia de comunicação) ou prover recursos específicos para as aplicações a que os dispositivos estão associados. Mesmo sendo dispositivos restritos[1], é fundamental que possam seguir um padrão que possibilite as mesmas operações de gerência comuns a todas as classes de dispositivos (IPs e não-IPs) e que, até mesmo, facilite a interoperabilidade entre diferentes redes.

Neste sentido, a OMA (*Open Mobile Alliance*) propôs um protocolo denominado de LwM2M (*Lightweight Machine to Machine*) para operações de dispositivos limitados de arquitetura M2M (*Machine to Machine*). É um protocolo de alto nível que indica o formato das operações de gerenciamento e troca de dados entre aplicações e dispositivos que vêm ganhando grande atenção. É apontado por [3] como uma forte tendência de amplo uso para IoT nesta classe de dispositivos.

Contudo o processo de gerência não se resume ao(s) uso(s) de protocolo(s), mas também a definição de recursos a serem gerenciados, como o de ligar e de desligar o

[1] Dispositivo restrito é aquele que apresenta algum recurso de computação, memória, *throughput* ou consumo de energia com limitações. A comparação é feita geralmente com dispositivos capazes de executar protocolos IP e com recursos computacionais aptos a executar até mesmo sistemas operacionais. Este já seriam considerados, dentro do contexto IoT, dispositivo ricos em recursos.

dispositivo, mudar calibração de um sensor, alterar os parâmetros de rede, atualizar, acrescentar ou excluir os recursos gerenciados, de como serão realizadas estas operações, de criação da interface de comunicação entre dispositivo e aplicação, de estabelecimento do fluxo de dados etc., sempre verificando como estes serão estruturados (JSON, XML, etc). Todas estas questões são importantes e fazem parte de um **sistema de gerência de dispositivos**. IoT é um mercado muito fragmentado, com as mais diversas necessidades, por exemplo, requerimentos para aplicações de "cidades inteligentes" são diferentes de aplicações para "redes elétricas inteligentes" (*smart grids*). Por isso, um sistema de gerência deve procurar atender diversos domínios.

## 1.2 Objetivos da pesquisa

Considerando as questões apontadas anteriormente, a possibilidade de se desenvolver recursos de gerência de dispositivos restritos que sigam o padrão LwM2M e que podem ser usadas em aplicações de IoT, esta pesquisa teve por objetivo principal: definir um sistema funcional de gerência de dispositivos restritos, usando o LwM2M que possua funcionalidades fundamentais relacionadas às gerências de rede e dispositivos. No geral, o sistema cria condições técnicas para o desenvolvimento de uma série de funções de gerência para dispositivos restritos em um contexto de IoT. A troca de dados de operações de baixo nível do sistema de gerência é realizado por meio do padrão LwM2M, ou seja, o sistema de gerência criado nesta pesquisa se encontra em camadas superiores à do protocolo.

Para ser possível cumprir os objetivos propostos neste trabalho, uma série de recursos foram integrados, adaptados e desenvolvidos, a fim de que no final se produza uma solução que permita a gerência de dispositivos restritos no contexto de IoT. Além do sistema conceitual, foi criada uma API (*Application Programming Interface*) implementada no dispositivo restrito e no *back-end* do servidor de gerência que possibilitasse a execução das operações de gerência de dispositivos.

Esta proposta vai além da definição de um sistema de gerência, pois cria condições técnicas para que a implementação do padrão aqui adotado pudesse ser executado em dispositivos restritos, mas especificamente em uma rede LoRaWAN. Ainda que esta pesquisa estivesse, inicialmente, cerceado ao uso e implementação de dispositivos restritos do tipo LoRa, o sistema desenvolvido possibilita que ações futuras possam facilmente integrar outras tecnologias de comunicação, a usarem a mesma estrutura de gerência. Outros objetivos decorrem deste trabalho:

- Avaliar, ainda que preliminarmente, o potencial e pertinência da associação entre as tecnologias LwM2M e LoRaWAN como solução IoT.
- Levantar em campo alguns dados básicos de rede, com base nos recursos de gerência de rede desenvolvidos.
- Viabilizar recursos mais sofisticados para rede LoRa através da integração entre o servidor LoRaWAN[2] e os recursos de gerência de rede para que melhorem a disponibilidade de seu tráfego e disponibilidade de seus nós.
- Analisar a validade de se usar o LwM2M na comunicação LoRaWAN e em quais condições isto é viável.

## 1.3 Justificativa

Um dos problemas para muitas aplicações IoT é que os dispositivos são restritos em recursos simples e poderão hibernar na maior parte do tempo (como sensores alimentados com bateria), além de apresentarem alta escalabilidade. Logo, estes fatores introduzem o problema de gerenciamento de dispositivos em redes IoT, que nem sempre podem usar a pilha IP/TCP/UDP. É importante que os dispositivos de redes do tipo LPWAN possam ser gerenciados por um protocolo padronizado que consiga lidar com a

[2] O servidor LoRaWAN ou LNS (*LoRa network server*), como será visto mais à frente, tem, como uma de suas responsabilidades, controlar os rádios LoRa no uso dos canais de comunicação da forma mais automática possível. A gerência de dispositivos (que vai além do controle da camada física) não é feita por ele, mas sim por instâncias 'superiores', como a aqui proposta.

complexidade de se usar um espectro não-licenciado (por exemplo o LoRa). Como a tecnologia LPWAN é relativamente nova, a quantidade de soluções de gerência é limitada.

É importante dar ao operador formas de conhecimento de indicadores básicos de funcionamento de cada dispositivo, para permitir a ele definir quais tipos de aplicações aquele dispositivo pode atender. Não menos importante, dispor de meios para descoberta de recursos, preferencialmente *plug and play*, que facilitem comissionar o dispositivo em campo e vinculá-lo à aplicação de destino. Todas estas demandas incentivam o desenvolvimento de um sistema que utilize os protocolos padrões que possam, no futuro, tornar-se interoperável com outras aplicações.

## 1.4 Contribuição

Espera-se que o sistema de gerenciamento aqui proposto seja capaz de contribuir para o gerenciamento de dispositivos restritos e não-IPs, como o LoRa[3]. O fato de se utilizar uma padrão de comunicação (LwM2M) que ajude na interoperabilidade com outros serviços, a capacidade de ofertar recursos que possibilitem o controle de canal do dispositivo, o levantamento de índices básicos de rede e comissionamento e operações específicas devem contribuir para o uso de protocolos padronizados para o gerenciamento de dispositivos IoT. O sistema criado pode também ser estendido para outros tipos de rede (como o Wi-Fi, Bluetooth etc.).

## 1.5 Escopo, limitações e convenções

Este trabalho propõe um sistema para gerência, que utiliza o padrão LwM2M, para realizar a interface de comunicação entre servidor e dispositivos. Também

[3] Com relação a tecnologia de comunicação, ressalta-se mais uma vez que neste primeiro momento os esforços foram conduzidos com a tecnologia LoRa, uma vez que este trabalho está inserido em um contexto de um projeto de P&D que já emprega LoRa.

desenvolve um esforço de implementação de funções de baixo nível, mais próximas do *hardware* do dispositivo de IoT (*i.e.*, sendo executadas pelo microcontrolador) e na implantação da infraestrutura LoRaWAN que dá suporte ao sistema de gerência de dispositivo. Contudo apenas as partes tidas como mais importantes foram implementadas e, por consequência, testadas. Os detalhes do que foram de fato implementados estão descritos no Capítulo 4 deste trabalho.

Outros elementos que compõem o sistema de referência, como a interface gráfica do sistema no servidor e algumas de suas funções de *back-end* e banco de dados foram implementadas por outros autores vinculados a um contexto maior que é um projeto de P&D, na qual esta pesquisa também se insere. Neste sentido, em termos práticos, as implementações próprias deste trabalho estão associadas ao dispositivo, ao formato das mensagens, a estrutura de dados, a forma como o dispositivo responde aos eventos de gerência, a troca de mensagens com o servidor LoRaWAN e ao seu *link* com o *back-end* da aplicação de gerência.

Para que os propósitos básicos desta pesquisa pudessem ser alcançados, foi necessário delimitar algumas questões e estabelecer algumas convenções. A principal delas é o fato de que os testes de gerenciamento foram realizados apenas em redes LoRa. Ainda que a estrutura criada possua objetivos de ser genérica, entendeu-se, em um primeiro momento, que LoRa representaria o caso mais desafiador de gerência devido as suas limitações. A extensão a outras tecnologias seria mais fácil de ser implementada. Ao final, apresenta-se um sistema mais "transparente" (em relação a tecnologia de comunicação empregada) possível para o gerente de dispositivos. A criação do *middleware* que identifica a aplicação e qual a tecnologia de comunicação não foi implementado nesta pesquisa será feito em trabalho futuros.

## 1.6 Organização do texto

No Capítulo 2 foi feita a fundamentação teórica sobre o tema aqui pesquisado. Nele é dado enfoque ao protocolo CoAP (*Constraint Application Protocol*) e LwM2M, que são a base técnica desta proposta. São explicados em detalhes os seus elementos e como funcionam.

No Capítulo 3 foi realizada uma análise do estado da arte e são apresentados os trabalhos mais relevantes encontrados na literatura técnica a respeito do protocolo LwM2M e suas aplicações em diversos campos. Também são apresentadas algumas questões de *middlewares* IoT, uma vez que a gerência de dispositivos está geralmente vinculada ao *middleware* como sendo um de seus serviços.

No Capítulo 4, a metodologia abordada nesta pesquisa é detalhada. Nele são apresentados os componentes da arquitetura de gerenciamento proposta. É detalhada o fluxo de informação dentro da comunicação entre servidor de aplicação e dispositivos LoRaWAN.

O Capítulo 5 traz os resultados e discussões acerca dos experimentos feitos. Análises do sinal e da estabilidade de comunicação dos dispositivos são realizadas com frequência, graças a plataforma construída. Também são verificadas algumas limitações da estrutura proposta dentro do contexto aqui abordado.

Finalmente, o Capítulo 6 tece as conclusões obtidas e apresenta sugestões de trabalhos futuros.

# 2 Principais elementos de IoT

Este capítulo reúne uma série de informações pertinentes às tecnologias que deram suporte direto ou indireto à pesquisa desenvolvida. Ainda que temas secundários também foram tratados de modo a abranger, sem muitos detalhes, o grande domínio de IoT, eles ajudam o leitor a identificar, mais à frente, os métodos empregados nesta pesquisa e as discussões feitas.

## 2.1 IoT

IoT pode ser entendido como um conjunto de "coisas" diversas (sensores, máquinas, processos etc.) conectadas para gerar dados e, a partir destes, produzir informações. O objetivo é entender o contexto e ser capaz de controlá-lo [4]. IoT é a respeito de dados e não de "coisas", portanto toda infraestrutura construída em IoT deve ter como um dos seus objetivos uma forma de atribuir valor aos seus dados coletados [5].

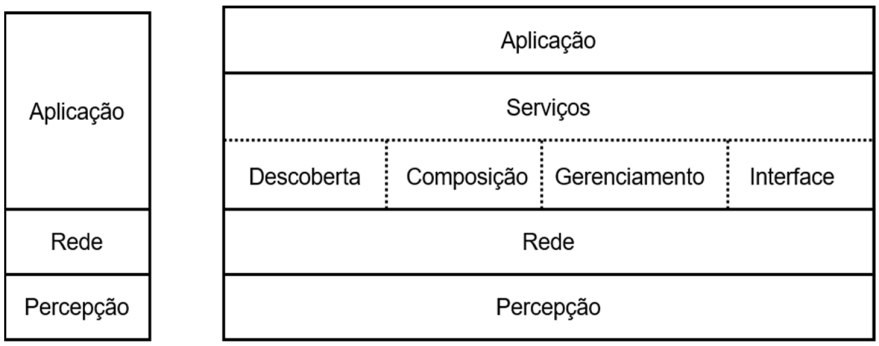
A infraestrutura IoT é complexa e grande, composta por diversos tipos de protocolos, ferramentas, sistemas de comunicação, controladores, sensores, plataformas etc. Para esclarecer este universo, este capítulo aborda alguns tópicos relacionados à infraestrutura de IoT, como: arquitetura, protocolos de aplicação, sistemas de comunicação etc. Ainda que o tema segurança seja importante, ele está além do escopo deste trabalho. De toda forma, este tema é abordado com detalhamento em [6].

## 2.2 Arquitetura IoT

As pesquisas realizadas por Al-Fuqaha *et al.* [7] e Lin *et al.* [8] descrevem alguns modelos de arquiteturas IoT. Eles destacam duas arquiteturas genéricas: (i) a de três camadas e (ii) a orientada a serviços (SoA - *Service Oriented Architecture*). A arquitetura

de 3 camadas é composta pelas camadas de aplicação, rede e percepção. A arquitetura SoA é formada pelas camadas de aplicação, rede, serviços e percepção. A Figura 1 ilustra estas camadas. Uma breve explicação sobre algumas características específicas delas é feita de forma pontual na sequência:

Figura 1 - Arquitetura de 3 camadas (a esquerda) e arquitetura orientada a serviços




Fonte: o autor

- **Camada de percepção**: são os dispositivos físicos, compostos tipicamente de MCU (microcontroladores), sensores e (ou) atuadores. É o local onde os dados são coletados e atuações no ambiente podem ser realizadas [7].
- **Camada de rede**: é responsável por se associar aos sensores (ou atuadores) da camada de percepção, de modo a poder transmitir os dados gerados por estes. O roteamento e encaminhamento de pacotes para a aplicação e dispositivos também são realizadas nesta camada. Deve ser capaz de transmitir os seus dados entre diferentes aplicações e dispositivos. Para isso, é interessante que possua suporte para uma grande quantidade de tecnologias de transmissão e protocolos [10].
- **Camada de gerenciamento de serviços ou *middleware***: é a camada que deve ser capaz de trabalhar com dispositivos que utilizem diferentes protocolos e tecnologias, de forma a buscar interoperabilidade nos protocolos e controlar os diferentes fluxos de dados aos seus destinos. Esta camada deve criar uma interface de abstração entre dispositivos físicos IoT e a aplicação para facilitar esta interoperabilidade. Isto deve permitir que as aplicações possam ser oferecidas independentemente da tecnologia de comunicação empregada. Pode ser

decomposta em 4 subcamadas: descoberta, composição, gerenciamento e interface. A subcamada de descoberta é capaz de encontrar quais são os recursos disponíveis na infraestrutura IoT. A subcamada de composição de serviços decompõe a requisição da aplicação em mais de uma tarefa para se alcançar a resposta desejada. A subcamada de gerenciamento de recursos é usada para a configuração de como os serviços podem ser executados. Por fim, a interface é usada para suportar interações dentre todos os serviços existentes [8].

- **Camada de aplicação**: é camada de mais alto nível em IoT, onde são realizadas as tarefas requisitadas pelos usuários, como mostrar gráficos, históricos, requisições de atuação no sistema, mostrar o estado atual do dispositivo etc. Geralmente esta camada deve oferecer suporte ao banco de dados, *backups*, análise dos dados coletados, dentre outras funções [8]. Ela é a consumidora dos dados produzidos nos sensores e organizados pelo *middleware*. Normalmente é aqui que o dado é transformado em informação útil ao usuário.

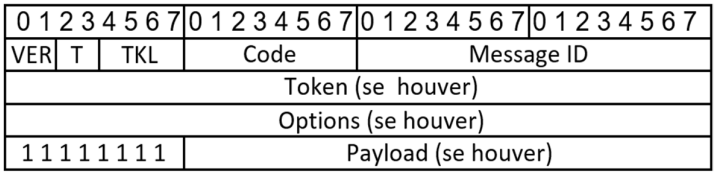
## 2.3 Protocolos de aplicação IoT

Esta subseção trata de importantes protocolos que foram usados nesta pesquisa: CoAP, LwM2M e MQTT (*Message Queuing Telemetry Transport*).

### 2.3.1 CoAP

O CoAP é um protocolo baseado na arquitetura cliente-servidor, criado pelo IETF por meio do RFC 7252 [9]. É um protocolo de transferência de dados desenvolvido para ser usado em dispositivos com recursos limitados, em redes tolerantes à perda de dados (*lossy networks*) e que não exijam uma alta taxa de transmissão de dados. O formato da mensagem CoAP é mostrado na Figura 2.

Figura 2 - Formato da mensagem CoAP




Fonte: adaptado de [9]

- **Ver**: campo de dois bits que mostra qual a versão do protocolo. Nesta pesquisa foi usada a versão 1;
- **T**: campo de dois bits para informar o tipo da mensagem. Ela pode ser confirmável (**CON**), o que quer dizer que aguarda uma resposta à mensagem enviada através do segundo tipo de mensagem, o **ACK**. Esta última é uma mensagem de resposta a uma requisição do tipo CON. Outra opção é ser uma mensagem não confirmável (**NON**). Por último, pode ser uma mensagem do tipo reset (**RST**).
- **TKL**: campo de quatro *bits*, que indica o tamanho do *token*.
- ***Code***: do tamanho de um *byte*, estabelece o método que está sendo usado pelo CoAP. Também mostra se a mensagem está vazia ou se ocorreu algum problema durante a troca de mensagens.
- ***Message* ID**: com dois *bytes* de tamanho, esta informação é usada para detectar pacotes duplicados e é usado para comparação de mensagens do tipo confirmáveis (se o servidor enviar uma mensagem do tipo CON, a resposta a esta mensagem deve possuir o mesmo *message id* e *token* previamente enviados).
- ***Token***: é um campo variável de até 16 bits utilizado para efeito de comparação entre mensagens.
- ***Options***: contém configurações que podem ser adicionados neste campo como, o endereço para onde a mensagem deve ser enviada, se existe ou não um *proxy* no sistema, qual o formato da mensagem de *payload,* entre outras funções.
- ***Payload***: é o conteúdo da mensagem CoAP. É separado do resto do protocolo por 0xFF e pode estar em diferentes formatos (o formato deve ser informado no

comando opções por meio de *Content-Format*). Alguns dos padrões suportados são: CBOR, JSON, XML, TLV etc.

O protocolo COAP é baseado na arquitetura REST (*Representational State Transfer*), que estabelece os métodos **POST**, **GET**, **PUT**, **DELETE** (mensagens confirmáveis do tipo CON) para comunicação entre aplicações.

O método POST é usado normalmente para inserir informações do cliente (dispositivo) no servidor (servidor da aplicação) ou algum comando (ligar e desligar, por exemplo) que o servidor deseja enviar para o dispositivo. A resposta a um comando POST é uma mensagem ACK, informando que a operação requisitada foi realizada com sucesso ou se ocorreu algum problema.

O método GET pode ser usado para extrair uma determinada informação do dispositivo IoT. O servidor envia uma requisição GET, o cliente (ou dispositivo *end-device* de IoT) reconhece o comando e encapsula a resposta em uma mensagem do tipo ACK, devolvendo-a ao servidor.

O método PUT é tipicamente usado para atualizar parâmetros no dispositivo IoT. O comando é enviado pelo servidor e, então, o dispositivo realiza a atualização requerida e devolve uma mensagem de ACK indicando que a operação foi realizada com sucesso.

O protocolo CoAP é baseado no tradicional HTTP. O seu objetivo não é criar uma versão compacta do HTTP, mas, sim, criar uma configuração reduzida da arquitetura REST otimizada para dispositivos M2M [9].

Uma configuração importante do CoAP é a observação de recursos. Neste modo o servidor envia uma requisição de observação para o cliente. Esta mensagem é do tipo CoAP GET com uma informação no campo de opções que é do tipo OBSERVER. Quando a requisição chega ao dispositivo, ele realiza uma operação OBSERVER em um determinado recurso (lê um valor de sensor, por exemplo, o de temperatura). O dispositivo então envia uma mensagem ACK ao servidor com o valor do recurso desejado (o valor de temperatura). A partir daí o dispositivo envia mensagens todas as vezes que

ocorrer alguma mudança no recurso observado (atualização da leitura do sensor), sem precisar de uma nova requisição do servidor. Estas mensagens são do tipo NON, acrônimo de não confirmável.

Dispositivos e servidores CoAP conseguem identificar recursos por meio da URI (*Uniform Resource Identifier*), que sempre é enviado em requisições, do tipo CON (GET, POST, DELETE e PUT). A URI é informado no campo de opções.

O CoAP é normalmente usado com o UDP e em redes 6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Network*). Porém é um protocolo de aplicação genérico e pode ser utilizado em diferentes tecnologias de comunicação.

### 2.3.2 OMA LwM2M

O LwM2M é um protocolo para gerenciamento de dispositivos, habilitação de serviços, transferências e monitoramento de dados que conseguem trabalhar em dispositivos com recursos limitados (de processamento, comunicação, energia etc.) [10]. Foi criado pela OMA (*Open Mobile Alliance*). O LwM2M utiliza o protocolo CoAP para a troca de mensagens entre cliente (dispositivo) e servidor. Ao contrário do COAP, ele não estipula o formato do pacote de dados da rede e, sim, como deve ser a sequência de procedimentos entre um servidor e um cliente para troca de diferentes tipos de mensagens. Neste sentido, ele visa organizar e orientar os procedimentos de gerência de rede, troca de informações e outras ações típicas de rede.

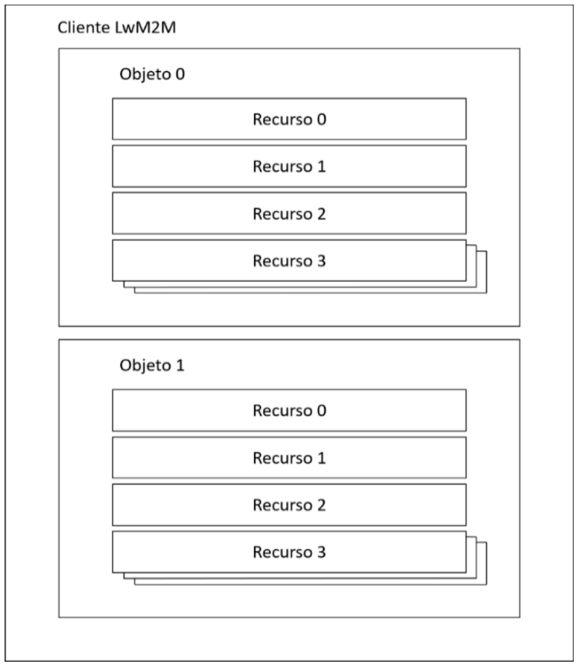
O LwM2M utiliza um modelo de recursos chamado IPSO *smart objects* (grupo de trabalho pertencente a OMA). O modelo padroniza uma série de objetos e seus recursos [11]. Recursos são organizados dentro de objetos e cada recurso possui um identificador (um número), assim como o objeto. Um sensor de temperatura, por exemplo, é um objeto, definido pelo número 3303. Seus recursos são: valor de temperatura atual; temperatura mínima e máxima medida; unidade de temperatura (Celsius, Fahrenheit, Kelvin); descrição da extensão de valores que o sensor consegue captar (máxima e mínima); e, por

fim, reinicialização dos valores máxima e mínimos medidos; que são identificados pela numeração 5700, 5601, 5602, 5603, 5604, 5701 e 5605, respectivamente. A Figura 3 ilustra o escopo desta formatação de objetos e recursos definidos pelo IPSO. No site do IPSO é disponibilizado, em formato XML, uma grande quantidade de objetos com seus respectivos recursos.

A partir dos modelos disponibilizados pelo grupo de trabalho IPSO é possível criar objetos e recursos para atender demandas específicas de cada aplicação, deste que sigam o modelo proposto.

O LwM2M possui 4 modos de operação: (i) *bootstrap*, (ii) registro (iii) gerenciamento de dispositivo e habilitação de serviços e (iv) notificação. A Figura 4 ilustra esquematicamente estas operações.

Figura 3 - Modelo de objetos e recurso desenvolvido pelo IPSO




Fonte: adaptada de [10]

Figura 4 - Operações que podem ser realizadas pelo LwM2M


Fonte: adaptada de [12]

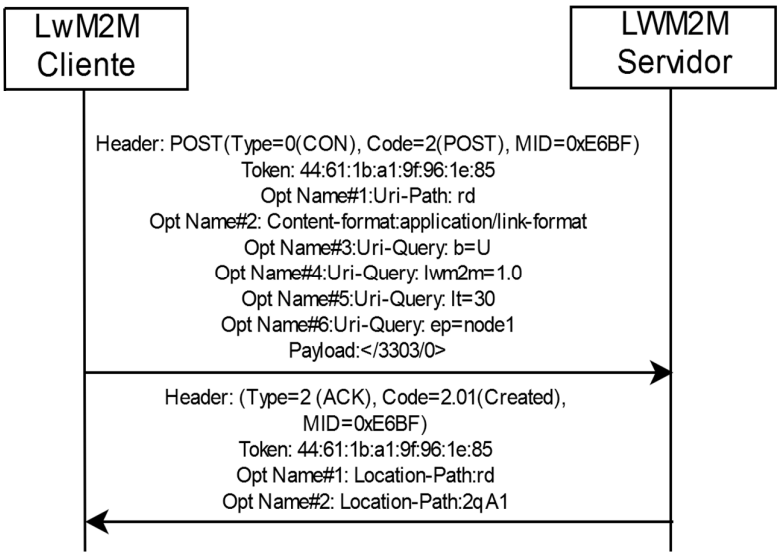
Para fins de esclarecimento, os próximos pontos descrevem resumidamente as funções de cada uma destas operações retratadas na Figura 6:

- ***Boostrap***: esta é uma etapa de segurança, onde chaves de segurança são trocadas entre cliente e servidor. Esta etapa não foi utilizada ao longo do projeto.
- **Registro**: nesta etapa, o dispositivo informa ao servidor os seus objetos (sensor de temperatura, humidade etc.) para que o servidor possa cadastrar o dispositivo e saber suas capacidades. Este procedimento é feito via mensagem CoAP POST, após o envio da mensagem o servidor LwM2M devolve uma mensagem CoAP ACK. Após esta etapa, o gerenciamento e transferência de dados pode ser realizada. A Figura 5 ilustra este procedimento. Nela é possível verificar que é enviado ao servidor o objeto 3303, o que indica que o dispositivo possui um sensor de temperatura a ser gerenciado.
- **Gerenciamento do dispositivo e habilitação de serviços**: nesta etapa do processo, o servidor gerencia dados e dispositivos. Podem ser realizadas tarefas, como: leitura, escrita, execução de operações (ligar, desligar, por exemplo), criação, remoção e atualização de recursos etc. Por exemplo, operações de leitura são feitas via CoAP GET e atualizações no dispositivo são realizadas via CoAP PUT. Operações de execução são feitas por meio de CoAP POST. A Figura 6 ilustra o formato de uma mensagem de leitura seguindo a normatização LwM2M

e o protocolo CoAP. Nesta mensagem é feita uma tentativa de leitura por parte do servidor no objeto 3303 (sensor de temperatura) e no recurso 5700, o que significa o valor atual de temperatura. Em seguida, o dispositivo envia a mensagem de resposta com o valor de temperatura atual requerido.

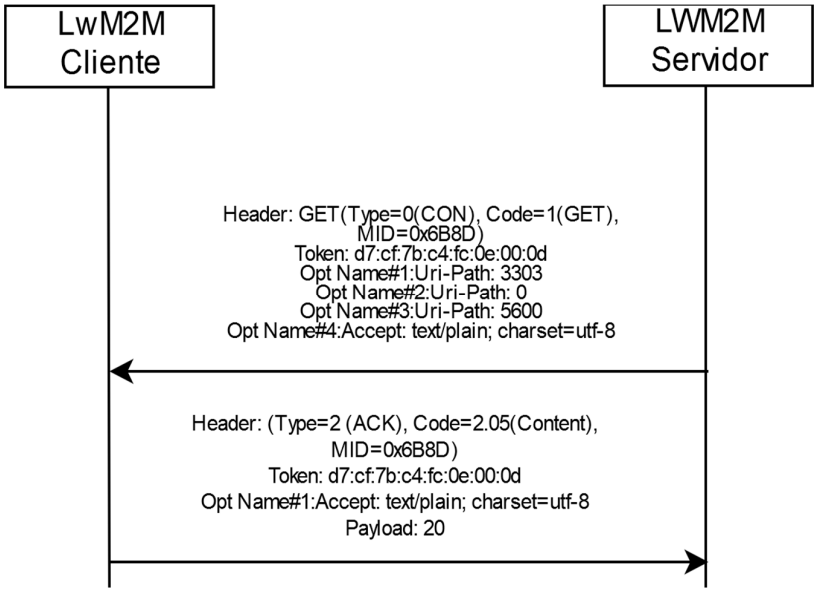
- **Notificação**: neste modo de operação, o dispositivo envia mensagens para o servidor em caso de algum valor nos seus recursos sofrerem alguma modificação. Esta operação é feita por meio do procedimento de observação (*observer*) do CoAP. Não é uma operação automática, por isso cabe ao servidor enviar o comando de observação no recurso desejado no dispositivo para que ele entre neste estado de observação. A Figura 8 ilustra o procedimento de notificação.

Figura 5 - Fluxo de mensagens CoAP para uma operação de registro do LwM2M




Fonte: o autor

Figura 6 - Fluxo de mensagens CoAP para uma operação de leitura do LwM2M




Fonte: o autor

### 2.3.3 MQTT

O MQTT utiliza uma arquitetura de comunicação, conhecida como *publish-subscribe*. É um protocolo aderente à arquitetura M2M, que deve utilizar um elemento chamado de *broker*, por onde os dispositivos e aplicações se conectam a ele. Quando um dispositivo tem alguma mensagem para ser publicada (*publish*), é enviado uma mensagem para um tópico ao *broker*. Neste caso, um tópico seria, por exemplo, "dados/temperatura/". Os servidores (ou outros elementos de rede) que desejarem ler estas mensagens devem se subscrever (*subscribe*) neste tópico no *broker*. Isto cria um sistema escalável e decentralizado, onde um dispositivo pode enviar uma mensagem para vários outros dispositivos ou aplicações. Basta que estejam conectados ao *broker* e subscritos no tópico desejado. A Figura 7 ilustra o fluxo de mensagens de uma aplicação que faz uso do MQTT.

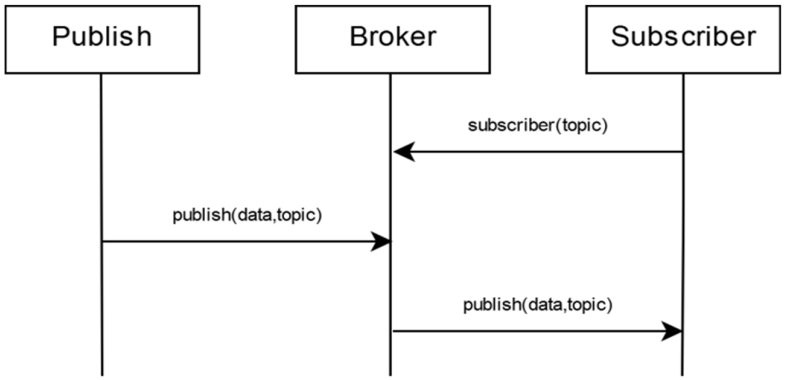
O transporte de mensagens MQTT é feito pelo TCP/IP, o que possibilita uma confiabilidade. Existe um padrão de QoS (*Quality of Service*) de três níveis para o MQTT. A primeira configuração é QoS-0, que estabelece que quando uma mensagem é

transmitida não é aguardado a confirmação se de fato a mensagem chegou no destinatário. No segundo padrão, QoS-1, a mensagem é transmitida e é aguardada a confirmação de chegada da mensagem. Neste modo é garantido a entrega da mensagem pelo menos uma vez. Na terceira configuração (QoS-2), a mensagem é enviada e é garantido que será entregue exatamente uma vez, evitando assim duplicações [13]. No entanto, dada a complexidade do TCP/IP e o suporte de rede requerido por ele, o MQTT convencional pode ser inviável para alguns dispositivos restritos. Por isto existe uma derivação conhecida como MQTT-SN (*Sensor Network*) de onde o protocolo de transporte usado é o UDP, facilitando o uso em dispositivos restritos, uma vez que o UDP é um protocolo de transporte mais simples [14].

Thangavel et al. [15] traz uma comparação entre o desempenho do CoAP e MQTT. Basicamente, chega-se à conclusão de que mensagens MQTT convencionais possuem uma latência menor se comparadas ao CoAP quando a taxa de erros do canal é pequena. Porém, quando a taxa de perda de pacotes é alta (*link* de comunicação com menor confiabilidade), o CoAP possui um desempenho melhor, pois não gera um grande volume de tráfego, uma vez que não possui nenhuma garantia de entrega de mensagens e utiliza o UDP.

Além do CoAP e MQTT, existem outros importantes protocolos de aplicação para IoT, como o DDS (*Data Distribution Service*), o AMQP (*Advanced Message Queuing Protocol*) e o XMPP (*Extensible Messaging and Presence Protocol*) [7]. Estes protocolos não estão dentro do escopo de trabalho e não foram utilizados.

Figura 7 - Fluxo de mensagens MQTT, que ilustra o processo de MQTT PUBLISH e MQTT SUBSCRIBE




Fonte: o autor

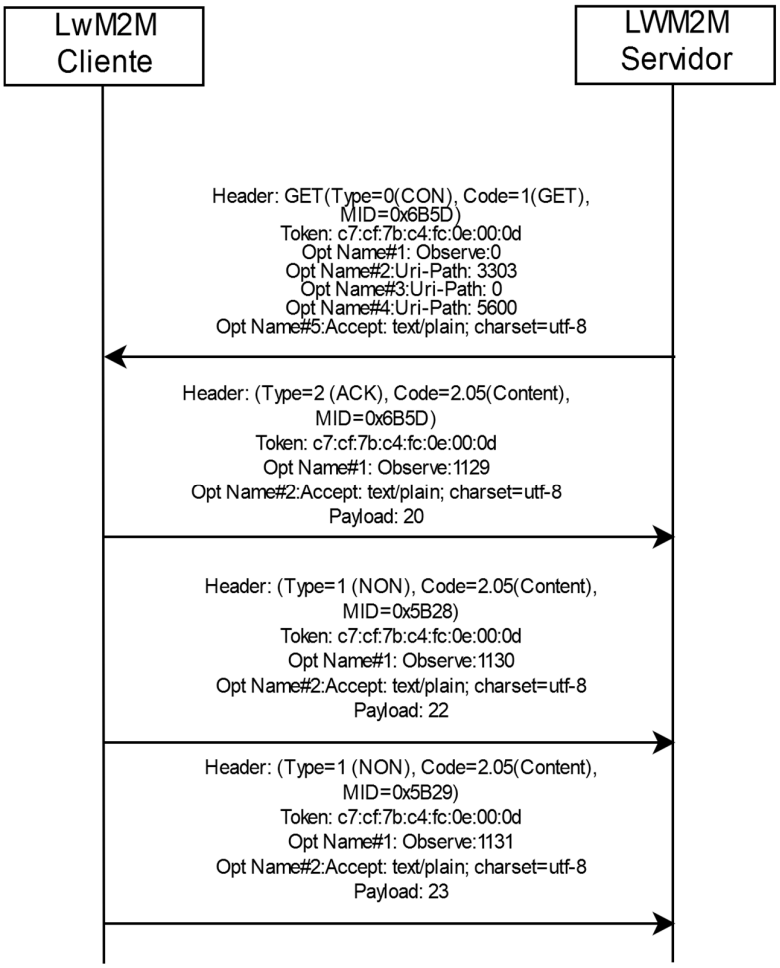
## 2.4 Protocolos de Infraestrutura em IoT

Além dos protocolos descritos anteriormente, que são mais associados à camada de aplicação (ou seja: mais de "alto nível"), existem também outros importantes protocolos de camadas mais inferiores (camada 2 especialmente, considerando o modelo OSI - *Open System Interconnection*), que viabilizam a comunicação em IoT. E, dada esta importância, alguma destas tecnologias são tratadas resumidamente na sequência.

### 2.4.1 IEEE 802.15.4

Criado pelo IEEE (*Institute of Electrical and Electronics Engineers*), estabelece um padrão aberto de camada de enlace e física (camada 1 e 2, considerando o modelo OSI), ideal para redes onde não exista a necessidade de uma alta taxa de dados de transmissão e que são tolerantes a perda de dados. Esta rede se enquadra na categoria de redes WPAN (*Wireless Personal Area Network*). Trabalha em frequências não licenciadas, como 868GHz, 915MHz e 2.4MHz; e possui taxas máximas de transmissão de 20Kbps a 250Kbps.

Figura 8 - Fluxo de mensagens CoAP para uma operação de notificação do LwM2M




Fonte: o autor

O padrão apresenta importantes características, como o CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) e suporte à rede MESH. O CSMA/CA reduz a colisões de pacotes no caso em que existem muitos nós na rede. Com este procedimento, o dispositivo não inicializa a transmissão enquanto um outro nó esteja transmitindo. Na arquitetura de rede MESH, dispositivos repassam mensagens para outros dispositivos, até que a mensagem chegue ao destinatário. Nesta configuração é possível que a mensagem chegue a distâncias maiores. O roteamento de pacotes entre dispositivos é normalmente feito por meio do protocolo RPL (*Routing Protocol for Low-Power and Lossy Networks*) [7].

É um padrão muito testado e com um grande tempo de mercado, muitas pilhas de protocolos utilizam como base o IEEE 802.15.4, como: ZigBee, Thread, ,6LoWPAN, WirelessHART, dentre outros [16].

### 2.4.2 6LoWPAN

O 6LoWPAN estabelece uma pilha de protocolos com o objetivo de que os dispositivos com recursos restritos possam utilizar o IPv6 para se conectar à internet. Uma das suas principais funções é a de compressão do cabeçalho do IPv6. Uma mensagem que contenha IPv6 possui um tamanho típico de 1280 bytes. O IEEE 802.15.4 suporta um quadro máximo de até 127 bytes. Portanto, em teoria, não seria possível habilitar um dispositivo IoT que utilize IEEE 802.15.4 como um sistema IPv6. O 6LoWPAN comprime e fragmenta o cabeçalho de pacotes IPv6, possibilitando, assim, que dispositivos restritos possam se beneficiar do IPv6. O 6LoWPAN é normalmente usado junto com o padrão IEEE 802.15.4, mas é uma pilha de protocolos genéricos e pode ser usado em diferentes tecnologias de transmissão [17].

O roteamento de pacotes 6LoWPAN é feito usando o protocolo RPL, criado pelo IETF (*Internet Engineering Task Force*) por meio do RFC 6550 [18]. Este protocolo cria o suporte para redes MESH e é otimizado para dispositivo limitados. No transporte e na aplicação são utilizados geralmente os protocolos UDP e CoAP, respectivamente.

Há uma implementação de *software* aberto desta pilha de protocolos muito popular, que é o sistema operacional Contiki. Esta implementação pode ser executada em dispositivos com recursos limitados. O tamanho total de um *firmware* Contiki-OS é por volta de 100KB e o uso de memória RAM pode ser otimizado para até 10KB. Possui suporte para uma grande quantidade de arquiteturas de hardware de MCUs, como ARM Cortex M3/M4 e TI MSP 430. Há também um simulador de rede de código aberto 6LoWPAN desenvolvido em Java e chamado de COOJA [19].

### 2.4.3 LTE-M Cat 1

É uma tecnologia de rede padronizada pelo 3GPP (*3rd Generation Partnership Project*) que pode ser usada por dispositivos limitados em IoT e se enquadra em redes do tipo LPWAN. Possuem um alcance maior do que redes WPAN e uma maior autonomia de bateria, em contrapartida da taxa de transmissão de dados.

LTE-M (*Long Term Evolution - Machine Type Communication*) Cat 1 suporta até 300Kbit/s em *downlink* e 375Kbit/s em *uplink* em modo *half duplex*, possuindo um baixo consumo de potência, sendo 100mA seu consumo mínimo e em *standby* 8µA. Tem como suporte a FOTA (*Firmware Updates Over the Air*), que possibilita que o *firmware* de um dispositivo IoT possa ser atualizado remotamente, sendo uma característica muito importante para manter a segurança em dispositivos IoT. Este padrão permite que o dispositivo consiga se comunicar com diferentes células de uma torre de rede, sem perder a conexão. Pensemos em uma torre de celular, onde são inseridas várias células com o objetivo de oferecer uma cobertura em 360°. A LTE-M é similar a um celular, onde o usuário mesmo em movimento não perde a sua conexão. Este procedimento é chamado em inglês por *handover*. Isto é fundamental para dispositivos IoT não estáticos inseridos em objetos móveis (como carros, máquinas agrícolas e animais). Há também suporte para comunicação por voz. LTE-M Cat 1 é uma configuração de rede robusta, segura e escalável, que se aproveita dos benefícios que a rede celular oferece. Porém, para que possa ser utilizada, há uma necessidade de algum tipo de contrato com operadoras de rede celular, assim como também depende da disponibilidade da operadora oferecer esta tecnologia na região de interesse [20].

### 2.4.4 NB-IoT

NB-IoT (*Narrowband* IoT) é uma rede do tipo LPWAN também padronizada pelo 3GPP, que utiliza uma largura de banda de 200KHz e possui taxas de comunicação de 27.2Kbits/s de *downlink* de *uplink* 62.5Kbits/s em modo *half-duplex*. Tem um baixo

consumo de potência, apresentando 60mA seu consumo mínimo e em *standby* 8µA e, assim, como o LTE-M Cat 1, beneficia-se de toda a infraestrutura da rede celular. Suas principais diferenças em relação a LTE-M Cat 1 são o fato desta última ser mais indicada para aplicações em tempo real que necessitam uma maior taxa de transmissão de dados e que sejam dispositivos móveis. NB-IoT é o mais indicado para sistemas estáticos em geral.

Uma importante característica que padrões criados pelo 3GPP possuem é o fato de suas frequências usadas serem do tipo licenciadas, o que naturalmente diminui casos de interferência entre sistemas, uma vez que boa parte de todo o ecossistema IoT tendem a usar frequências não licenciadas por motivos de custos [20].

As informações relativas à taxa de dados e consumo de potência foram retiradas de um módulo de desenvolvimento LTE CAT 1 e NB-IoT, chamado SARA-R4 (x2B) series da empresa u-blox.

### 2.4.5 LoRa

LoRa (*Long Range*) é uma tecnologia de camada física patenteada pela empresa SEMTECH, que faz parte da categoria de redes LPWAN. Suas principais caraterísticas são o seu longo alcance (2Km a 10Km em ambiente urbanos e até 20 Km em ambientes rurais) e seu baixo consumo de energia, que de acordo com [21], no modo de recepção possui um consumo de até 23.6mA e no processo de transmissão até 47mA, em modo de baixo consumo de energia, o valor cai para até 1.65µA. A taxa de transmissão do LoRa varia de 300 bps a 37.5Kbps, dependendo do cenário [22].

LoRa é um sistema de modulação baseado na tecnologia chamada CSS (*Chirp Spread Spectrum*), que possui diferentes modos de operação, chamados de SF (S*preading Factor).* Há seis SF diferentes que podem ser usados: SF7, SF8, SF9, SF10, SF11 e SF12. Basicamente, em SF7 é possível alcançar uma maior taxa de transmissão de dados, porém em distâncias mais curtas. Ao se utilizar SF12, o sinal de transmissão é capaz de atingir

distâncias maiores, no entanto com uma taxa de transmissão menor. Os usos de SFs mais baixos possibilitam os alcances de menores níveis de latência. O inverso ocorre com SFs, com valores maiores. LoRa opera em frequências não licenciadas abaixo de 1GHz, com a faixa dependendo de cada país.

Dispositivos LoRa são utilizados tipicamente junto ao protocolo de camada de enlace LoRaWAN, que é um protocolo aberto específico para os rádios LoRa.

### 2.4.6 LoRaWAN

LoRaWAN é um protocolo aberto de enlace (Camada 2) desenvolvido pela LoRa *Alliance*, criado para ser usado junto ao LoRa. Possui topologia de rede estrela, onde vários dispositivos se conectam a um *gateway* LoRaWAN (algumas vezes chamado de concentrador), que repassa, então, estes dados para um servidor LoRaWAN (normalmente utilizando UDP). Por sua vez, o servidor LoRaWAN (LNS – *LoRaWAN Network Server*) envia os dados para servidores de aplicação, tipicamente em MQTT. Alguns dos servidores LoRaWAN mais populares são o TheThings Network [31] e ChirpStack [23].

Uma das atribuições do LNS é validar o acesso do dispositivo e permitir que possa enviar e receber dados. Um dos procedimentos que podem ser usados em redes LoRaWAN para este propósito se chama OTAA (*Over the Air Activation*). Neste processo, dispositivo e servidor precisam ter uma chave chamada AppKey, previamente salvas em ambos. O processo OTAA inicia-se com uma mensagem de JOIN REQUEST por parte do dispositivo, onde a chave AppKey é enviada, o servidor então recebe esta mensagem e, se estiver tudo correto, devolve uma mensagem JOIN ACCEPT para o dispositivo. Assim que a última mensagem é enviada para o dispositivo, é gerada uma chave dentro do LNS chamada AppSKey. Com a mensagem de JOIN ACCEPT que chega ao dispositivo, é gerado a mesma AppSkey. A partir desse ponto, o dispositivo pode enviar mensagens para a aplicação [24].

LoRaWAN possui três modos operacionais para o LoRa: classe A, classe B e classe C; resumidos a seguir:

- **Classe A**: trata-se de dispositivos com comunicação bidirecional, nos quais uma transmissão (*uplink*) é seguida por duas janelas de recepção. Este modo de operação é voltado para dispositivos que necessitam ter um baixo consumo de potência e que apresentam como objetivo principal a aplicação de mensagens de *uplink*. Mensagens de *downlink* enviadas pelo servidor podem ser recebidas pelo dispositivo em uma das duas janelas de recepção que se abrem quando é transmitido um pacote. Estas janelas possuem tempo de 1 e 2 segundos respectivamente [24].
- **Classe B**: são dispositivos com comunicação bidirecional. Quando o dispositivo realiza alguma transmissão, são abertas duas janelas de recepção (da mesma forma que em classe A), porém as novas janelas de recepção podem ser abertas em tempos programados, utilizando um processo chamado de *beacons* [24].
- **Classe C**: neste modo, o dispositivo envia uma mensagem e são abertas duas janelas de recepção. Porém, diferentemente da classe A e B, a segunda janela de recepção não é fechada. Assim o dispositivo está sempre apto a receber comandos do servidor de aplicação. Este modo de operação é ideal para aplicações onde são necessárias atuações constantes no ambiente ou em casos onde dispositivos necessitam de uma comunicação constante com o servidor [24].

O método de acesso ao meio do LoRaWAN é o ALOHA, ou seja, se um dispositivo possui alguma mensagem a ser enviada, ele simplesmente a envia. Isto é um problema no caso de muitos dispositivos, pois pacotes irão colidir com mais frequência. Por isso há restrições da operação de sistemas do tipo ALOHA, que variam de acordo com a regulamentação de cada país. Na Europa (regulação ETSI - *European Telecommunications Standards Institute)* é usado a frequência de 868 MHz, que possui restrições de ciclo de trabalho (em inglês o termo é *duty cycle*). Esta regra se refere ao

máximo tempo que um dispositivo pode transmitir em um determinado canal de frequência. Esta restrição é fixada em 1%, o que implica que um dispositivo pode transmitir por apenas 36 segundos por hora, para cada canal. A regulamentação brasileira é baseada na Australiana e é utilizada a faixa de frequência de 915-928 MHz[4]. Não há restrições de ciclo de trabalho aqui, porém, em algumas regiões, existe um limite do máximo de tempo que um dispositivo pode transmitir em um determinado canal (em inglês o termo é conhecido por *dwell time*), no caso 0.4 segundos. Isto inviabiliza, por exemplo, o uso de SF11 e SF12, pois são configurações que possuem uma latência intrínseca e, por menor que seja o tamanho das suas mensagens, violam a regra dos 0.4 segundos se operando nestes SFs [25].

O LoRaWAN possui um algoritmo chamada de ADR (*Adaptative Data Rate*), que observa a potência do sinal recebido e, ao se perceber que o sinal está fraco (RSSI - *Receive Signal Strength Indicator* está muito baixo), pode mudar o SF da transmissão para um nível maior. O ADR também pode alterar taxas de transmissão de dados e níveis de potência. Portanto, o objetivo do ADR é otimizar a rede LoRaWAN [24].

Ainda vale mencionar que, embora existam compressões de cabeçalhos para IPv6 para serem utilizados em redes LoRa [26], foi utilizado o modelo tradicional LoRa ao longo da pesquisa, isto é, topologia estrela sem IP nos dispositivos. Além do mais, a especificação LwM2M [11] trata de dispositivos LoRaWAN gerenciáveis como sendo não-IP. O esforço de se colocar um cabeçalho IP em um pacote LoRa reduziria o espaço útil para *payload*, limitando as aplicações.

### 2.4.7 SigFox

SigFox é uma rede do tipo LPWAN que utiliza uma largura de banda bem curta de 100Hz, que pode alcançar até 100 bps de *uplink* com um tamanho de pacote de 12

[4] Ao contrário do que acontece na Austrália, a ISM brasileira também permite transmissões na faixa 902-907MHz.

bytes. É possível serem enviadas apenas 14 pacotes por dia. A sua topologia de rede é estrela [22]. A infraestrutura de rede SigFox é proprietária, incluindo *gateways* e servidores de aplicação, que podem ser utilizados por meio de planos de assinaturas. Existe uma grande cobertura global, inclusive no Brasil. A implantação de uma solução em SigFox é rápida, uma vez que toda a infraestrutura está preparada para receber dados nas regiões cobertas [22].

## 2.5 Resumo do capítulo

Este capítulo tratou de alguns pontos específicos das tecnologias que dão suporte à IoT. É possível notar a complexidade de IoT ao ver os seus diversos componentes, porém, independente da tecnologia, a essência mais básica de IoT está em registrar dados de "coisas" (através de sensores e de uma rede de comunicação) e transformar estes dados em informação. Para isto acontecer é necessário a utilização de *middlewares* (*middleware* em IoT são explicados em mais detalhes no próximo capítulo), que de preferência devem ser agnósticos quanto à tecnologia de comunicação. Para abstrair estas necessidades, *middlewares* em geral possuem uma série de componentes que vão desde os protocolos de comunicação até os modernos recursos de virtualização.

No próximo capítulo, além da revisão relacionado a *middlewares*, é mostrado o atual estado de aplicações que utilizam o LwM2M como gerenciamento de dispositivos e como estas aplicações se relacionam com os objetivos deste trabalho.

No Capítulo 4 é proposta uma série de recursos computacionais para se estabelecer os recursos mínimos de gerência de rede para dispositivos com *link* de dados muito restritos, como é o caso, por exemplo, de redes LoRa/LoRaWAN. Os materiais e métodos necessários para tais objetivos são detalhados.

# 3 Estado da arte LwM2M

A necessidade de desenvolver modelos e sistemas para gerências ou interfaces de comunicação entre dispositivos, especialmente os restritos, tem sido uma vertente rica na pesquisa na área de IoT. Por ser nova, o número de sugestões é relativamente grande, uma vez que não se percebe consolidação na área. Neste sentido, este capítulo apresenta os principais esforços de pesquisa envolvendo esta área. Em especial, é dado destaque em iniciativas de integração que usam o LwM2M e como se encaixa no contexto de *middleware* para IoT.

## 3.1 Pesquisas relacionadas a LwM2M

Em Chen *et al.* [27] é proposto um novo modelo de gerenciamento de grupo de dispositivos, uma vez que, com o aumento no número de dispositivos em sistemas IoT, o gerenciamento individual de dispositivos pode se tornar muito complexo. É relatado a utilização do padrão oneM2M junto ao protocolo de gerenciamento LwM2M, que seria estendido, para também ser capaz de realizar o gerenciamento de grupos de dispositivos. Os autores criaram um *software* chamado *Group Handler*, que é instalado no *gateway* LwM2M, que se comunica com os dispositivos oneM2M. O padrão oneM2M foi elaborado por um consórcio de empresas, com o objetivo de criar padrões técnicos com o foco de mitigar questões relacionadas à interoperabilidade, arquitetura, API, segurança, dentre outros aspectos no contexto de IoT. Os padrões produzidos pelo oneM2M podem ser utilizadas em cidades inteligentes, automação e no setor de saúde, dentre outras áreas. O objetivo do oneM2M é remover a fragmentação de IoT, uma vez que é independente da conectividade (Wi-Fi, Bluetooth, LoRa etc.) [28].

Em comparação ao trabalho de [27], o sistema de gerência proposto nesta pesquisa possibilita que dispositivos sejam agrupados de acordo com as necessidades dos usuários, sem a precisão de alteração do *software* instalado no *gateway*, o que aumenta a

complexidade do sistema de Cheng *et al.* [27]. Além disso, possui o foco em dispositivos restritos que não possuem IP, como é o caso do LoRa. A solução de Chen *et al.* [27] não menciona esta situação, todavia o suporte ao padrão oneM2M junto ao LwM2M pode vir a ser muito importante no processo de integração de diferentes sistemas IoT. O sistema criado ao longo desta pesquisa não suporta o padrão oneM2M.

Karagaac *et al.* [29] analisa a integração e interoperabilidade entre Indústria 4.0 e IoT. Para demostrar este processo, os autores propõem integrar o protocolo industrial OPC UA e o LwM2M. Para alcançar este objetivo foi construída uma camada de *software* utilizando containers para demostrar o funcionamento de sua proposta. O OPC UA, criado pela OPC Foundation [30], é um protocolo industrial recente, com o foco na comunicação M2M e na interoperabilidade entre aplicações industriais. Por meio do *software* virtualizado criado, a comunicação entre os dois protocolos se torna transparente. Mensagens de gerenciamento e de aplicação podem transitar de um protocolo ao outro, alcançando, assim, a interoperabilidade.

Na área médica também há uma demanda por interoperabilidade entre dispositivos que possuam diferentes protocolos. Em Li *et al.* [31] é proposta a integração do LwM2M junto à arquitetura CONTINUA [32], que é utilizada dentro do contexto de assistência médica remota. É fundamental que dispositivos médicos remotos sejam gerenciados e monitorados, assim como os *gateway*s nos quais os dispositivos estão conectados. A arquitetura CONTINUA foi criada pelo consórcio *Personal Connected Health Alliance* (PCHA) e foca na confiabilidade entre dados trocados por dispositivos médicos de uso pessoal.

O LwM2M também foi adaptado para ser utilizada no campo de multimídia, como mostra o trabalho proposto por Karagaac *et al.* [33], uma pilha de *software* com o objetivo de criar um protocolo de *streaming* eficiente e compacto para IoMT (*Internet of Multimedia Things*). Além disso, a proposta visa criar uma comunicação M2M global e interoperável, para dispositivos IoMT heterogêneos onde possam comunicar entre si. Os

autores sugerem o uso do protocolo LwM2M, uma vez que possui um modelo de dados bem definido e uniforme em sua representação. Tudo isto é importante para lidar com diferentes tecnologias de multimídia e aplicações de *streaming*. Foram criados objetos e recursos específicos usando o padrão IPSO como referência. A proposta foi validada usando uma rede LPWAN NB-IoT.

Os trabalhos desenvolvidos por Karagaac *et al.* [29], Li *et al.* [31] e Karagaac *et al* [33] mostram que o protocolo LwM2M pode atender diferentes áreas de aplicações dentro de IoT, demonstrando a sua importância e flexibilidade. Embora na solução criada nesta pesquisa não exista suporte para os padrões OPC UA e CONTINUA, o sistema pode ser expandido para atender estas e outras demandas, pois o protocolo LwM2M, juntamente com as operações de gerenciamento propostas neste trabalho, podem ser estendidas para outras aplicações.

O trabalho de Karagaac *et al.* [33] utiliza de maneira prática a tecnologia NB-IoT (tecnologia LPWAN assim como o LoRa) no contexto de multimídia, o que é algo interessante, porque é um tipo de comunicação que pode ser usado em sistemas restritos, assim como o sistema desenvolvida nesta pesquisa, com a diferença fundamental de que o NB-IoT é uma tipo de rede pertencente ao 3GPP que é, consideravelmente, mais complexa que o LoRa. O sistema de gerência desta pesquisa em relação a [33] procurou desenvolver operações de uma forma que os usuários da rede possuam todo o controle da infraestrutura criada, uma vez que a tecnologia de comunicação LoRa permite isto.

A pesquisa realizada por Hoebeke *et al.* [34] propõe o gerenciamento de redes LPWAN que possuem diferentes tecnologias. Em alguns sistemas são necessários configurações e opções de rede diversas. Portanto alguns dispositivos podem possuir múltiplos rádios LPWAN para satisfazer diferentes requisitos. Estes sistemas são chamados de multimodais. Gerenciar estes dispositivos é uma tarefa complexa. Para isto é proposto um operador de rede virtual utilizando containers Docker que unifiquem dispositivos multimodais, de uma forma que redes LPWAN heterogêneas possam ser

gerenciados por meio do LwM2M. O operador virtual da rede criada deve padronizar a forma em com que diferentes redes LPWAN implantadas são gerenciadas. Este processo é chamado de plano de controle. Ele garante que a comunicação entre dispositivos multimodais seja padronizada (plano de dados), independente da tecnologia de rede física empregada. É utilizado na comunicação do plano de controle, por isso foram criados objetos e recurso específicos com referência no padrão IPSO para realizar o gerenciamento entre redes LPWANs.

A pesquisa realizada em [34] é importante, pois é uma tentativa de amenizar a questão da heterogeneidade de sistemas de comunicação em IoT, mais especificamente LPWAN. Este é um tópico abordado no Capítulo 4, onde o sistema de gerência construído pode evoluir para ser capaz de gerenciar outros tipos de rede além de LoRa/LoRaWAN. Isto pode ser feito por meio de *middleware* que pode entender qual é o tipo de tecnologia usado, se, por exemplo, é LoRa, NB-IoT ou Wi-Fi, sem a necessidade de criar um plano de controle e dados, como feito por [35], deixando o sistema construído mais simples. Este *middleware* também seria capaz de identificar de qual aplicação pertence a mensagem enviada pelo dispositivo. Na próxima seção é explicado em detalhes a função de *middleware* dentro de IoT.

Dentro do tópico de gerenciamento de dispositivos, que se utiliza o LwM2M, a maioria das propostas são de integrar o protocolo junto a outros padrões, como o oneM2M, OPC-UA e CONTINUA, e a utilização do protocolo LwM2M em contextos específicos, como em multimídia. Porém, dentro da literatura, existem poucos trabalhos relacionados ao gerenciamento de dispositivos não-IPs utilizando LwM2M, como é o foco deste trabalho. Além disso, o trabalho não é uma mera implementação LwM2M no contexto LoRa, mas é também uma série de operações que uma gerência de dispositivos precisa ter, além de estender para operações da gerência de rede, onde tráfego de dados pode ser manipulado. Por fim, o LwM2M é usado como um meio da gerência extrair e

inserir informações no dispositivo, por isso, também, faz parte do contexto do sistema de gerenciamento.

## 3.2 *Middleware* para IoT

O *middleware* é entendido como um conjunto de recursos de *software* que visa elaborar uma ligação entre os dispositivos (camada de sensores ou física) e as aplicações de IoT, além de ter a função de criar condições para a coleta de dados dos dispositivos, descoberta de recursos, dentre outras funções importantes. A gerência de dispositivos criada nesta pesquisa é um elemento de *middleware,* uma vez que contribui para a organização desta ligação fazendo a abstração dos dispositivos para a aplicação. Nos próximos parágrafos são descritos com mais detalhes dos tipos de *middlewares* existentes.

De acordo com o trabalho de Ngu *et al.* [35], as principais abordagens de *middleware* podem ser classificadas em três categorias: (i) soluções baseadas em serviço (*Service Oriented Architecture* - SoA), (ii) soluções baseadas em nuvem e (iii) soluções baseadas em atores. As principais diferenças entre estes tipos estão associadas a como elas tratam o suporte a novos dispositivos, os tipos de serviços prestados pelos seus *middleware*s e as camadas onde o *middleware* pode ser embarcado ou desenvolvido.

Na arquitetura baseada em serviços, o *middleware* está integralmente presente em um servidor ou estrutura de virtualização. Assim não existe código de *middleware* para ser executada diretamente no dispositivo. Nesta arquitetura, geralmente o *middleware* provê ferramentas simples de visualização de dados brutos coletados pelos dispositivos e limitadas funcionalidades que ajudam na integração entre aplicações ou interpretação de dados. Todas estas funcionalidades são oferecidas como "serviços" ofertados pelo *middleware*. Como exemplo, tem-se o *middleware* 'Hydra' ou 'LinkSmart' [36]. Neles, a abstração de dispositivos heterogêneos (e seu controle) é feita via WS (*Web Services*), independente de qual seja a tecnologia do dispositivo (Wi-Fi, ZigBee, Bluetooth etc.). Os dispositivos são descritos e abstraídos através de uma linguagem SAWSDL (*Semantic*

*Annotation for Web Service Description Language*). Entre seus principais recursos de *middleware*, pode-se citar a descoberta de serviços e dispositivos. Outro exemplo de *middleware* do tipo baseado em serviço é o GSN (*Global Sensor Networks*) [37], que provê meios para integrar e gerenciar dispositivos IoT heterogêneos. Ele é baseado na abstração de "sensor virtual" que torna possível que usuários desenvolvam abstrações de sensores através de descritores em padrão XML. Conceitos como gerenciamento de ciclo de vida, notificação e processamento de eventos são abordados por esse *middleware*. Aplicações externas acessam os sensores virtuais vias chamadas em padrão RESTful ou APIs.

Na arquitetura baseada em atores [38], o *middleware* está distribuído em duas ou mais camadas, nos servidores (virtualizados ou não), no *gateway* (se presente) e também sendo executada dentro do dispositivo. Por isto ela tende a ser mais vantajosa para aplicações que exigem escalabilidade e baixa latência. Desta forma, a computação de algumas atividades pode ser executada em ambas camadas. Assim como na arquitetura baseada em serviços, a arquitetura baseada em atores não necessariamente impõe um determinado padrão de comunicação ou protocolo nos dispositivos, favorecendo a heterogeneidade. Isto é feito através de um modelo de abstração de dispositivo que é independente. Um dos representantes desta classe é o Node-RED [39]. Ele é orientado a eventos que são programados pelo usuário no seu ambiente e possui módulos de computação distribuída que podem ser executados tanto no ambiente de servidores quanto na borda da rede em versões de tamanho limitado. Usa, ainda, uma representação visual para abstrair um dispositivo (que lá é chamado de '*node*'), que é um conjunto de códigos que executam um conjunto de funções específicas dependendo da entrada de dados provida pelo dispositivo ligado ao seu *middleware*. Também apresenta facilidade de integração com vários protocolos (MQTT, AMQP, CoAP etc.), o que facilita a comunicação entre o dispositivo físico e o mundo lógico provido por ele, segundo a programação (geralmente visual, por 'blocos') provida por sua estrutura. Uma série de APIs e bibliotecas são fornecidas para esta integração de comunicação.

A arquitetura baseada em nuvem adota padrões específicos entre os dispositivos para que possam se vincular ao *middleware* e dispor de alguma interoperabilidade. No *middleware* tipo nuvem é provido uma série de APIs (geralmente suportando RESTful) que fazem a ligação entre os dispositivos físicos e as aplicações, assim como a oferta de funcionalidades (armazenamento, processamento, monitoramento, análise etc.). Contudo todos estes elementos (aplicações e dispositivos) devem seguir um padrão comum de formatação de dados, chamadas etc. Se o *middleware* parar por algum motivo, todo o conjunto para. Nas outras arquiteturas, as aplicações podem, em algumas circunstâncias, se comunicarem diretamente com os dispositivos, especialmente se forem IP. Um *middleware* que pode ser encaixado nesta categoria é o Fiware [40]. Ele ajuda a abstrair dispositivos físicos em "*virtual entities*" e a conectar estes sensores a sua estrutura de *middleware* que é flexível e modular (baseada em agentes de IoT e módulos "*enablers*"). Uma vez que o dispositivo consegue se conectar a sua estrutura, permite ao usuário acessar seus dados a qualquer momento por chamadas REST. Os dispositivos podem trabalhar em diferentes protocolos. Para isto, deve ser adicionado a pilha de recursos, um "agente IoT" capaz de converter este protocolo de comunicação do dispositivo em um formato próprio do Fiware conhecido como NGSI (*Next Generation Service Interfaces*). Com base neste formato, o Fiware consegue gravar dados e manter o estado de uma entidade virtual pronta para consulta de uma aplicação, manipular eventos etc. Além disto, sua estrutura permite que escalabilidade em infraestrutura de nuvem.

Do ponto de vista de requerimentos de arquitetura para *middleware* é interessante consultar [41]. Nele também são tratadas as tendências mais recentes de recursos para IoT e soluções em nuvem. Vários *middleware*s disponíveis são analisados sob este enfoque arquitetural e para quais tipos de aplicações são adequados. Como este não é o tema central desta dissertação, o leitor mais interessado pode buscar informações complementares em [42] [43].

Há uma grande quantidade de ferramentas, *middleware* e plataformas para IoT disponíveis. O Anexo A ilustra algumas ferramentas *open-source* para IoT.

## 3.3 Resumo do capítulo

Este capítulo mostrou alguns trabalhos relacionados ao gerenciamento de dispositivos sob o ponto de vista de desenvolvimento para IoT de dispositivos restritos e associados ao padrão LwM2M. O contexto em que estes recursos se inserem dentro do *middleware* também foi tratado. Como esperado, as vertentes de abordagem dos problemas são variadas, contudo os serviços a serem prestados, tanto pela gerência de dispositivos quanto pelo *middleware*, são relativamente similares. O capítulo ilustra que o LwM2M pode ser usado em diferentes domínios em IoT. Isto evidencia a sua natureza genérica.

# 4 Requisitos do sistema de gerenciamento LoRaWAN LwM2M

Neste capítulo é apresentada a estruturação técnica do sistema proposto para o gerenciamento e comissionamento de dispositivos restritos por meio do protocolo padronizado LwM2M. O sistema em si é composto por uma associação de diversos subsistemas que dão suporte tecnológico aos procedimentos de gerenciamento. É importante destacar que, como descrito no Capítulo 1, o objetivo aqui é desenvolver um sistema para o suporte a dispositivos restritos nas operações de gerência e tráfego de dados para aplicações. Neste sentido, boa parte do esforço aqui realizado é dado a descrição do procedimento, análise e a avaliação das etapas que compõem o método proposto de gerenciamento de dispositivos, por meio de uma rede LoRaWAN.

Este capítulo foi subdividido em três partes, a saber: (i) descrever o *setup* montado para o gerenciamento de dispositivos LoRaWAN; (ii) detalhar o modelo de dados, de comunicação e de implementação; e, por fim, (iii) apresentar os cenários de testes e coletas de dados empregados nesta pesquisa para validar e caracterizar o método de gerência oferecido.

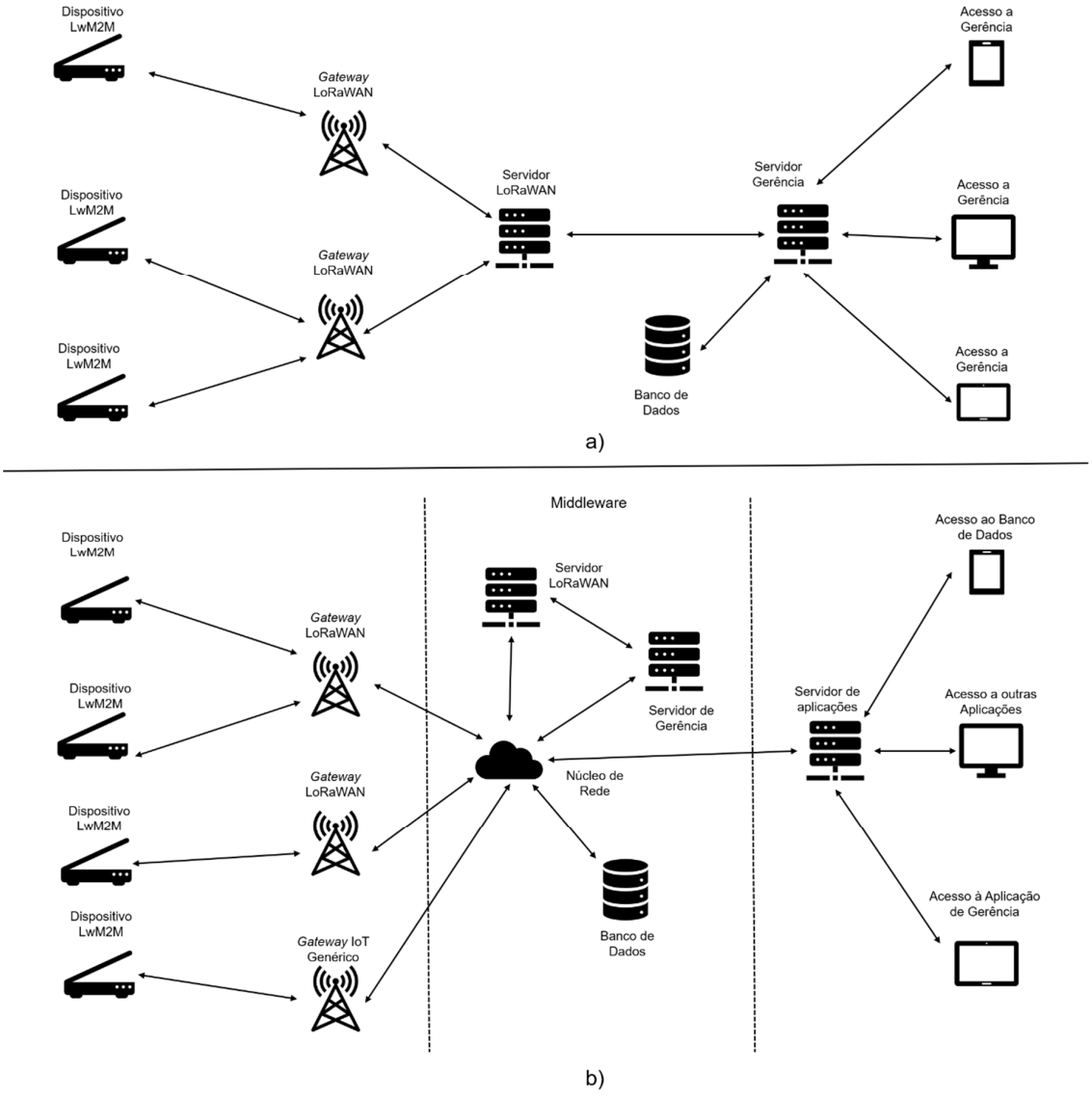
## 4.1 Materiais empregados

### 4.1.1 Rede LoRaWAN

O sistema de gerência construído baseado no LoRaWAN é composto idealmente por uma das configurações vistas nas Figura 9a e b. Nota-se que o serviço de gerência pode estar tanto no mesmo servidor em que é executado o LNS, quanto em outro servidor remoto. Ainda, o servidor de gerência pode receber dados diretamente do LNS ou de um

núcleo de um *middleware* IoT. No caso específico desta pesquisa, o sistema foi preparado para trabalhar com uma configuração similar ao que se vê na Figura 9a. Nela foram usados até 22 dispositivos, dois *gateway*s (modelo Wirnert AU 923MHz, do fabricante Kerlink) e um servidor (modelo PowerEdge XC730XD, do fabricante Dell) para dar suporte à execução do LNS e da plataforma de gerência de dispositivos produzida nesta pesquisa.

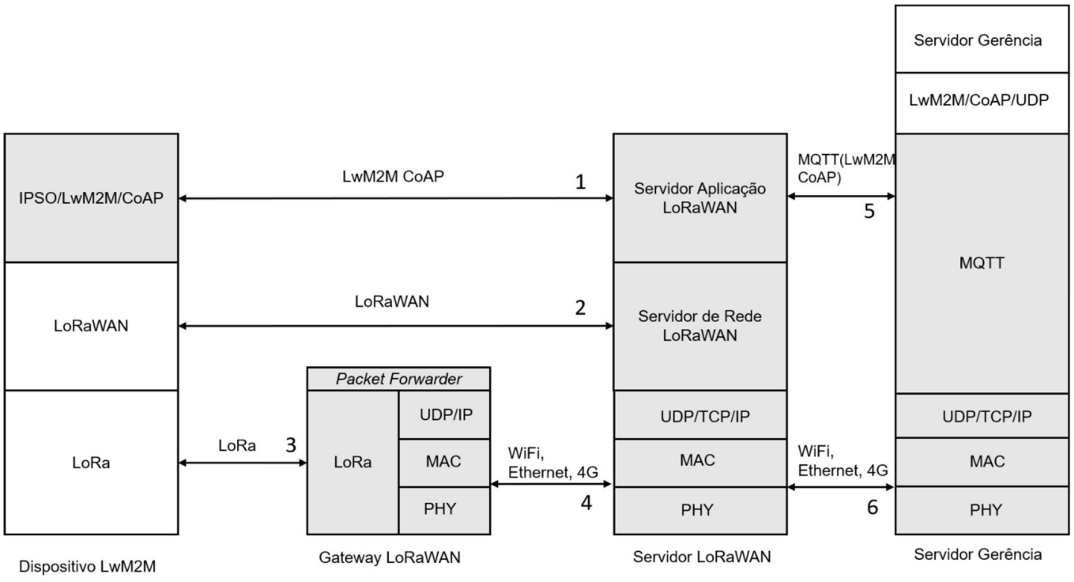
Figura 9 - Elementos da rede considerando (a) uma rede só com um LNS despachando as mensagens para o servidor de gerência b) um sistema com um *middleware* IoT *broker* distribuindo as mensagens subscritas nela




Fonte: o autor

Considerando a Figura 9a e cada um dos seus componentes, foi montada uma pilha de protocolos que é esquematicamente mostrada na Figura 10. Nesta figura é ilustrada a comunicação lógica entre os componentes, de forma a expor com quem cada camada se comunica e qual a função prática de cada uma.

Figura 10 - Fluxo de comunicação lógico entre os componentes do sistema proposto. Os destacados em cinza foram em alguma medida trabalhados aqui nesta pesquisa




Fonte: o autor

As principais implementações desta pesquisa estão associadas ao modelo de dados IPSO [11] e ao protocolo CoAP, ambos vistos logo no topo da pilha de protocolos do "dispositivo LwM2M". Contudo, para que funcionem corretamente dentro do contexto aqui vislumbrado, foi necessário o emprego de protocolos de mais baixo nível, como o LoRaWAN, utilizando a tecnologia de comunicação LoRa.

O próximo dispositivo de rede é o *gateway*. Ele recebe os dados dos dispositivos LoRa e os envia para o LNS, conforme indicado pelo fluxo de comunicação número 3 e 4, respectivamente. Para isto, ele faz um processo de conversão entre pacotes LoRa formatados por quadros LoRaWAN em pacotes IP. Isto é realizado por meio de um *software* denominado de *Packet Forwarder*. Para dar suporte a este *software* de alto

nível, geralmente os *gateway*s são capazes de executar uma versão de sistema operacional (em geral alguma distribuição Linux). Para os *gateways*, os pacotes de aplicação CoAP e LoRaWAN são transparentes, ou seja, não são manipulados por ele, tendo apenas a função de os repassar. A comunicação do *gateway* com o servidor LNS foi feita por meio de conexão ETHERNET, mas normalmente *gateways* podem suportar outros tipos de tecnologias, como Wi-Fi, 4G etc.

O servidor de rede LoRaWAN após receber os pacotes LoRaWAN provenientes do *gateway*, manipula estes dados, verifica chaves, processa JOIN REQUEST, analisa de qual dispositivo os dados pertencem, se estão formatados na forma correta etc. Seguindo o fluxo de comunicação, o servidor de aplicação LNS dispara as mensagens por meio de uma mensagem MQTT PUBLISH, onde estão o *payload* que contém os pacotes CoAP referentes a gerência. No tópico da mensagem MQTT é indicado qual dispositivo está enviando a mensagem. A comunicação lógica entre a camada de LoRaWAN do dispositivo com o servidor de rede LoRaWAN é ilustrado no fluxo de comunicação 2. A comunicação de número 1 ilustra a comunicação entre a camada de aplicação no dispositivo com o servidor de aplicação LoRaWAN, que, apesar de não manipular os pacotes CoAP, é responsável por repassá-los para o servidor de gerência. O servidor LoRaWAN se comunica com o servidor de gerência por meio de conexão ETHERNET, como está ilustrado no fluxo de mensagem número 6 da Figura 9a.

Uma vez que os dados chegam na aplicação de gerência de dispositivos (fluxo de comunicação número 5), a mensagem é descompactada e o seu *payload* é inserido dentro do protocolo UDP (na porta 5683). A partir deste momento, a gerência é capaz de realizar as operações e funcionalidades para a qual foi programada, tomando por base os recursos e formatação LwM2M para gerência de dispositivos. As funções pensadas para a gerência estão explicadas em detalhes na Seção 4.2.

A gerência envia requisições ou comandos aos dispositivos, por meio de mensagens CoAP dentro do protocolo MQTT, utilizando MQTT PUBLISH. A mensagem chega ao

servidor de aplicação do LNS, que, então, as envia para o *gateway* por meio do UDP. Por fim, o *gateway* envia as mensagens para os dispositivos finais.

No sistema atual, a gerência não é capaz de enviar diretamente estes dados ao dispositivo, pois a manipulação do LoRaWAN está concentrada no LNS. A manipulação de pacotes LoRaWAN, a configuração do canal de comunicação e a mudança dinâmica de SF são realizadas por ele. Portanto, no sistema criado, é um elemento obrigatório na solução para gerenciar os rádios LoRa. Em outras tecnologias de comunicação este elemento da solução pode ser eliminado.

A parte em cinza da Figura 9a evidencia o que foi desenvolvido ao longo desta pesquisa, sendo o suporte de aplicações de gerência no dispositivo, assim como a infraestrutura para que a aplicação de gerência seja capaz de acessar os dispositivos.

No outro sistema ilustrado na Figura 9b, este processo é ligeiramente diferente. O *gateway* repassa o pacote UDP para um *middleware* IoT *broker* e este último replica o pacote enviado tanto para o LNS (no caso se o dispositivo for LoRa e estiver utilizando o protocolo LoRaWAN) quanto para o servidor de gerência da rede (se os dados enviados pelo *end-device* são do tipo associados a gerência de dispositivos). Nesta configuração de rede, o sistema conseguiria identificar a tecnologia de comunicação e qual aplicação pertence as mensagens enviadas. Por exemplo, se o dispositivo a ser gerenciado estivesse utilizando NB-IoT, não haveria a necessidade destas mensagens serem enviadas para o servidor LoRaWAN, uma vez que o NB-IoT não utiliza o protocolo LoRaWAN, bastaria que a mensagem fosse enviada para o servidor de gerência e, em seguida, para o servidor de aplicação. No caso da mensagem enviada não for de gerência, mas, sim, de uma outra aplicação que o dispositivo esteja executando, bastaria ao *middleware* por meio do núcleo de rede enviar esta mensagem ao servidor responsável, que não seria o servidor de gerência neste caso. O objetivo da configuração de rede da Figura 9b é criar um sistema que distribua mensagens, de acordo com a tecnologia e o tipo da aplicação, desta forma

as aplicações de mais alto nível não precisariam lidar com estes detalhes. Esta configuração será implantada nos próximos trabalhos.

### 4.1.2 Ferramentas computacionais

A implementação de *firmware* no dispositivo foi feita em linguagem C e usando como base principal algumas importantes implementações *open-source*, como: Cantcoap [44], JSMN [45] e a LoRaWAN [46]. Esta última é uma adaptação de código para microcontroladores STM32 do fabricante STMicroelectronics. O código original foi desenvolvido pela Semtech e se encontra em [47]. Todas elas foram portadas para a aplicação e para o *hardware* do dispositivo utilizado nesta pesquisa.

No LNS foi utilizado a pilha de *software* Chirpstack [23]. No *gateway* foi instalado o *Packet Forwarder* da TTN (*The Things Network*) [48]. Na gerência foi usado o servidor LWM2M-NODE-LIB, disponibilizado e mantido pela TELEFONICA [49]. O servidor LwM2M, conhecido como LESHAN [50], foi utilizando em processos de validação ao longo do projeto. Todas estas implementações de base serão brevemente descritas a seguir:

- **CantCoap**: biblioteca em C++ que manipula pacotes CoAP. Possui foco na simplicidade. É capaz de receber, tratar e enviar pacotes de todos os tipos CoAP, considerando o RFC 7252. Na solução criada o código foi adaptado para linguagem C do microcontrolador usado no dispositivo. Existem várias outras implementações CoAP para dispositivos com recursos limitados, como: Libcoap [51], Contiki-OS [19], Wakaama [52]. Este último, além do CoAP, suporta as operações LwM2M. Contudo sua pilha deve ser executada com algum sistema operacional e se comparadas as outras, tem alto custo computacional. Por meio do CantCoap as operações LwM2M de gerência foram construídas nesta pesquisa para que pudessem ser executadas sem o uso de sistemas operacionais e empregadas em dispositivos mais restritos como o aqui usado.

- ***Packet Forwarder* TTN**: é continuamente executado no *gateway* LoRa, encapsula as mensagens provenientes dos dispositivos LoRa e as envia para o LNS como pacotes UDP/IP. Nele, basta apenas configurar o IP do LNS, que deve receber os pacotes e quais são os canais de frequência LoRa que serão usadas para a comunicação entre dispositivos e *gateway*. Esta informação de configuração também precisa ser compatível com a inserida no LNS. Em específico, este *software* suporta que uma mesma mensagem LoRaWAN possa ser repassada para diferentes LNSs. Essa é uma das suas principais qualidades.
- **JSMN** (jasmine): é uma biblioteca capaz de receber, tratar e enviar dados formatados em JSON. Nesta pesquisa, ela foi inserida no *firmware* do dispositivo. O formato foi escolhido como padrão de comunicação entre dispositivo e aplicações devido ao seu grande uso. Isto facilita futuras ações de interoperabilidade e padronização. A biblioteca JSMN é largamente utilizada, simples e compacta, ideal para sistemas com recursos limitados.
- **LoRaWAN**: é um conjunto de arquivos com exemplos de aplicações LoRaWAN para microcontroladores da STMicroelectronics. Possui suporte para os MCUs de 32 bits STM32L0, STM32L1 e STM32L4. É compatível com a especificação 1.1 do LoRaWAN. Tem suporte para os modos de operação A, B, C e para autenticação OTAA e ABP (*Activation By Personalization*). Consegue operar nas frequências de 433MHZ, 868MHz e 915MHz, além de suporte para diversas rádios LoRA. Toda pilha de protocolos é otimizada para o baixo consumo de energia. O consórcio que criou o LoRaWAN disponibiliza códigos de exemplo que podem ser adaptados para diferentes microcontroladores [53].

- **ChirpStack:** é uma pilha de *softwares* que implementa um LNS genérico. É constituído por: *Gateway Bridge*, *Network Server* e *Application Server*[5]. Os principais componentes deste pacote LNS são, brevemente descritos, como:
    - **Chirpstack *Gateway Bridge*:** é um serviço que converte as mensagens originárias dos *gateways* LoRaWAN em JSON e *protobuf* (um método de serialização de dados estruturados, criado pelo Google). Ele se comunica com o *Network Server*.
    - **ChirpStack *Network Server:*** realiza autenticação, agendamento de *downlinks*, processa comandos MAC LoRaWAN e analisa pacotes duplicados. Se comunica com o *Application Server*.
    - **ChirpStack *Application Server:*** é responsável pelo cadastro de dispositivos que atuam na rede LoRaWAN e pela manipulação de JOIN REQUEST (se tudo estiver correto nesta mensagem ele devolve um JOIN ACCEPT). Possui uma interface web, construída em MATERIAL UI (*framework* baseado em REACT). Nesta interface, usuários, dispositivos e *gateways* podem ser cadastrados, excluídos e gerenciados. Ferramentas externas podem se conectar ao ChirpStack *Application Server* por meio de RESTful. O envio de mensagens (*downlink*) para os dispositivos pode ser realizada por meio de integrações com MQTT e HTTP. Nativamente, todas as mensagens de *uplink* que chegam a esta camada são despachadas por meio de uma mensagem MQTT PUBLISH (no tópico desta mensagem é possível identificar qual dispositivo realizou a transmissão).
- **Eclipse Leshan:** disponibiliza bibliotecas e exemplos em Java, para o desenvolvimento de servidores e clientes LwM2M. Para facilitar o desenvolvimento, oferece uma API com boa parte das funções LwM2M e uma

[5] Além destes, existem ainda dois subsistemas (*Geolocation Server*, Gateway OS), que fazem parte deste pacote, contudo não foram usados nesta pesquisa.

interface gráfica web. Nesta interface são mostrados os dispositivos cadastrados, além de ser possível realizar operações de gerenciamento de acordo com os objetos cadastrados de cada dispositivo (requerer valores de temperatura, humidade etc.).

- **LWM2M-NODE-LIB**: implementa servidor e cliente LwM2M em node.js. Possui a maior parte das funções LwM2M implementadas (leitura, escrita, registro e notificação).

Além destas implementações, foram usadas outras pilhas de *software* ou sistema em seu formato original. Pode-se citar o banco de dados Postgree SQL, Redis e Mosquitto MQTT broker. Todos dando suporte para criação de uma solução completa para gerenciamento de dispositivos restritos.

### 4.1.3 Recursos de hardware

Nesta pesquisa foram usados dois diferentes *hardwares* LoRa (ou *end-devices*), a saber: (i) a placa de desenvolvimento B-L072Z-LRWAN1 (STMicroelectronics) que integra o módulo CMWX1ZZABZ-091 [54] e (ii) um *hardware* próprio projetado na pesquisa de P&D a que esta dissertação está vinculada. O primeiro caso emprega um microprocessador STM32L072CZ (Arm® Cortex®-M0+ com 192 KB de memória Flash e 20 KB de RAM), enquanto que no segundo caso (hardware próprio), usou-se um microprocessador STM32L433RCT6 (Arm® Cortex®-M4 com 256 KB de memória Flash e 64 KB de RAM). A Figura 12 ilustra uma imagem de ambos dispositivos. Em todos os casos, o modem LoRa utilizado foi baseado no módulo SX1276 da Semtech com potência configurável de até 100mW (20dBm) e operando na faixa de 902 a 928 MHz. Ao todo, foram usadas cerca de 5 placas B-L072Z-LRWAN1 e 22 placas próprias.

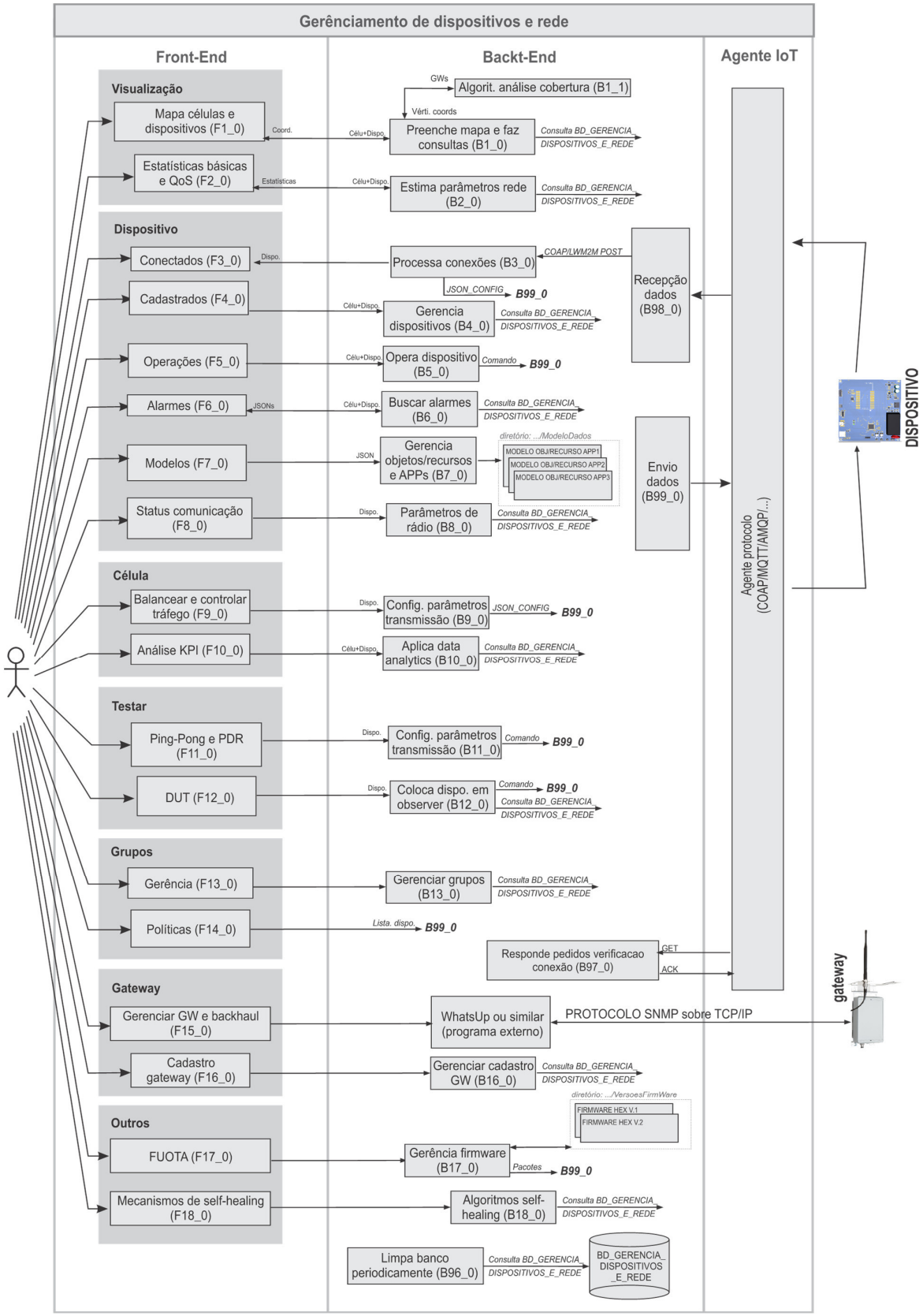
## 4.2 Sistema proposto de gerência de dispositivos restritos

### 4.2.1 Sistema funcional

A definição dos recursos de gerência de dispositivos propostos nesta pesquisa é apresentada através de um sistema idealizado, que é visto no diagrama esquemático da Figura 11, onde se pode tecer algumas descrições sobre os recursos funcionais previstos para cada um dos blocos ilustrados. Esta descrição resumida pode ser vista na Tabela 1.

O sistema de gerência deve ser capaz de dar suporte a qualquer um dos modelos de arquitetura IoT apresentados na Figura 9a e b. Ainda por ser um sistema implementado para *web*, foi usada uma arquitetura de *software* que desacopla a interface gráfica da implementação através das camadas de *front-end* e *back-end*. Uma terceira camada de *software*, designada por "IoT – Agentes" também foi empregada para permitir que nela, futuramente, sejam usados outros protocolos (AMQP, DDS, XMPP, etc) ou tecnologias de comunicação sem interferir com o *back-end*. O *front-end* (ou interface gráfica) é do tipo web (apresentada por um *browser*) e é executado pelo servidor de gerência. A sua implementação foi feita por outra parte de esforço de pesquisa (em andamento) que se associa a esta, cabendo aqui apenas a definição formal de como as informações de gerência devem ser apresentadas na interface gráfica.

Figura 11 - Diagramas de blocos funcionais do sistema proposto




Fonte: o autor

Tabela 1 - Descrição funcional e requerimentos básicos da proposta

| Tag [6] | Funcionalidade | Descrição resumida de funcionalidade do sistema |
|---|---|---|
| F1 e B1 | Mapear visualmente células e dispositivos | • Exibir interface gráfica que permita ao usuário visualizar em mapa localização de dispositivos e elementos de rede (*gateways*, por exemplo).<br>• Estimar a cobertura da rede M2M.<br>• Vincular a cobertura de dispositivos por células.<br>• Flexibilizar a visualização dos elementos da rede para o operador ser capaz de atuar e(ou) avaliar as suas condições. |
| F2 e B2 | Levantamento de parâmetros básicos de QoS | • Levantar parâmetros básicos de QoS e de rádio (de comunicação) de um dado dispositivo. |
| F3 e B3 | Levantamento (em tempo real) de dispositivos conectados à plataforma | • Listar todos os dispositivos (cadastrados ou não) que estão disponíveis para a plataforma e que podem ser operados.<br>• Permitir que dispositivos recém ligados à rede possam automaticamente fazer pedido de cadastro (apresentar-se automaticamente).<br>• Permitir que dispositivos não cadastrados (mas disponíveis) possam ser cadastrados. |
| F4 e B4 | Cadastro de dispositivos | • Apresentar uma interface para o preenchimento de informações relacionadas aos dispositivos como: coordenadas, configurações de ADR, SF etc.<br>• Este cadastro é salvo em um banco de dados e enviado ao dispositivo para que possa executar as políticas atribuídas a ele pelo servidor da aplicação de gerência.<br>• A qualquer momento pode ser requisitado campos individuais ou todo cadastro ao dispositivo, podendo qualquer campo ser alterado. Modificações são enviadas ao dispositivo e serão salvas no banco. |
| F5 e B5 | Operar em dispositivos | • Podem ser enviadas requisições de reset e hibernação, requisições de versão de *software* e hardware e outras operações que a gerência pode exercer no dispositivo. |
| F6 e B6 | Gerenciar alarme de dispositivos de rede | • Nesta tela é possível ver e acompanhar os alarmes gerados pelos dispositivos. Dentre os alarmes incialmente previstos estão o de dispositivo operando em alta temperatura, nível de bateria baixo e indicadores de QoS baixos. |
| F7 e B7 | Modelos de dispositivos e dados. | • Possibilita a criação de modelos genéricos de dispositivos. Eles devem descrever o que um determinado dispositivo pode ser capaz de fazer (quais sensores tem, se podem ser lidos e/ou escritos). A partir deste modelo de dispositivo, o sistema deve reconhecer os recursos de diferentes dispositivos, uma vez que, teoricamente, cada aplicação pode ter um ou mais modelos diferentes de dispositivo. E como o |

[6] Rótulos que associam as funções aqui descritas com o diagrama da Figura 11. O predicado F indica uma função do *front-end* e a letra B corresponde a função do *back-end*, que realiza a funcionalidade prevista nesta equivalente interface gráfica do *front*.

| | | |
|---|---|---|
| | | sistema de gerência deve servir a uma plataforma (ou seja, com uma ou mais aplicações) é importante que o sistema tenha conhecimento dos recursos de cada dispositivo. Isto é feito através desta associação com o modelo de dispositivo. |
| F8 e B8 | Status de comunicação da rede | • Mostrar um gráfico com informações de RSSI, SNR (*Signal Noise Ratio*) e SF dos últimos pacotes que chegaram à gerência, as mensagens podem ser separadas por células(*gateway*s) e dispositivos. |
| F9 e B9 | Balanço e controle de tráfego | • Através de variáveis de controle disparadas pelo servidor de aplicação, pode-se limitar a quantidade de mensagens e frequência de envio de um dado dispositivo com base na análise de seu tráfego e valores de QoS. Tamanhos limites de pacotes também podem ser controlados. |
| F10 e B10 | Análise de indicadores de rede | • Quando a rede escala (milhões de nós) a análise individual de dispositivos fica comprometida. Neste sentido, para tentar facilitar esta análise o operador pode visualizar apenas o quadro de KPI (*Key Parameters Indicator*), onde os principais valores de indicadores de rede (RSSI, SNR, taxa envio/recebimento, falhas conexão, etc) são tabelados e posteriormente agrupados por técnicas de processamento de sinais (*clustering*) para identificar as anomalias da rede (dispositivos potencialmente fora do padrão esperado) |
| F11 e B11 | Ferramentas de disponibilidade (ping e PDR) | • Realiza operação PING no dispositivo para analisar sua disponibilidade e latência.<br>• Faz estimativa de PDR (*Package Delivery Ratio*) de um dado dispositivo para estimar valores médios de taxa de entrega de pacotes, perdas de pacotes e bits corrompidos. |
| F12 e B12 | Teste de dispositivos | • No modo DUT (*Device Under Test*), o dispositivo envia mensagens de maneira periódica por longos períodos (até 7 dias) e frequência ajustável (variando de segundos a minutos). A intenção deste recurso é testar a comunicação de um determinado dispositivo (ou grupo) por longos períodos e avaliar seu desempenho. |
| F13 e B13 | Gerência de grupos | • Grupos são criado no processo de cadastro para categorizar os dispositivos segundo demandas específicas. Assim cada dispositivo pode ser vinculado a um determinado grupo. As operações podem ser aplicadas a todo um grupo flexibilizando a gerência destes dispositivos. Um dispositivo pode pertencer simultaneamente a um ou mais grupos. |
| F14 | Aplicação de políticas a grupos | • Permite que uma operação seja aplicada de uma só vez a todos os dispositivos de um mesmo grupo. Assim, se o operador deseja intervir em todos os dispositivos de um mesmo grupo, basta validar a operação uma só vez que o sistema trata de aplicar esta operação de forma automática a cada um dos dispositivos daquele grupo. |
| F15 | Gerência rede transporte e *gateway*s | • Por se tratar de uma rede IP, já existem inúmeras e maduras plataformas de controle e gerência de rede de enlaces e dispositivos desta natureza. Por isto será aproveitado um sistema existente e comercial para monitorar os *gateway*s e os enlaces. Neste caso, sugere-se o uso do *software* WhatsUp Golden, por ser um sistema web bem conhecido e que pode ser integrado ao sistema aqui proposto. Desta forma, o esforço da plataforma se concentra nos |

| | | |
|---|---|---|
| | | dispositivos restritos. Os dispositivos não restritos (como o *gateway* LoRa) podem ser gerenciado por uma ferramenta de terceiros a ser integrada aqui. |
| F16 e B16 | Cadastro de *gateways* | • *Gateways* são cadastrados no sistema de gerência. As informações são salvas em um banco de dados para uso de outras funções desta plataforma (como visualização em mapas, associação com dispositivos etc.). |
| F17 e B17 | Atualização de *firmware* remota (FUOTA) | • Por vezes é necessário efetuar atualizações de *firmware*. Realizar o processo manual é inviável, principalmente em caso de uma infraestrutura com muitos dispositivos. Por isso é necessária uma forma de atualizar dispositivo de forma remota (observação: isso não foi implementado nesta etapa da pesquisa; implementação futura). |
| F18 e B18 | Mecanismos de *self-healing* para gerência rede | • O dispositivo deve ser capaz de corrigir eventuais problemas que este venha a possuir durante a sua operação ou da rede (observação: isso não implementado nesta etapa da pesquisa; implementação futura). |

Fonte: o autor

Os parâmetros de KPI (que se resumem aos parâmetros de SF, RSSI e SNR, mostradas em F10 e B10 da Tabela 1) são extraídos da comunicação LoRa. Toda vez que um dado gerado pelo protocolo LoRaWAN é enviado ao LNS, existe um cabeçalho designado de "metadados" contendo diversas informações sobre a condição do enlace e seu estado. Dentre estas informações, estão os KPIs estimados pelo *gateway*. É com base nestes metadados que o LNS promove sua política (através do ADR) de gerenciamento do enlace LoRa. Neste sentido, o sistema de gerência aproveita esta peculiaridade de LoRa e captura do LNS e os dados são salvos no bando de dados do servidor de gerência. Na Tabela 2 é ilustrado uma mensagem enviada do LNS para a gerência contendo tanto o *payload*, formado em LwM2M/CoAP, quanto os metadados. Esta mensagem está no formato JSON. É através desta estratégia que se faz a estimação dos KPIs de rede, que podem ajudar o usuário do sistema de gerência de dispositivos na tomada de decisões.

Tabela 2 - Mensagem de metadados recebidos pelo LNS e repassados a gerência

| Variável | Significado | Formato JSON metadados |
|---|---|---|
| applicationID | ID da aplicação criada no LNS | Meta-Dados(KPI): JSON<br>{<br>“applicationID”:”1”,<br>“aplicationName”:”ceb”,<br>“deviceName”:”PlacaCEB_v11_Num4”,<br>“devEUI”: “1350485094377420”,<br>“rxInfo”: [{<br>“gatewayID”:”0000024b08031c25”,<br>“uplinkID”:”52db6e6d-edbe-4d7d-bc42-dd145bc43c26”,<br>“name”:”sm”,<br>“time”:“2020-02-03T12:37:20.784418Z”,<br>“rssi”: -57,<br>“loraSNR”: 10.2,<br>“location:”<br>{“latitude”:-18.91957,<br>“longitude”:-48.25892,<br>“altitude”:875}}],<br>“txInfo”: {“frequency”: 91800000,<br>”dr”:2}<br>“adr:”false<br>“fcnt”: 40,<br>“data”: ZEV4FaH/et4RAbEA/09L<br>} |
| applicationName | Nome da aplicação criada no LNS | |
| deviceName | Nome do dispositivo, vinculado a aplicação criada no LNS | |
| devEUI | Identificador único do dispositivo de 8 bytes | |
| gatewayID | Identificador único do *gateway* | |
| uplinkID | Identificador única de cada transmissão | |
| name | Nome atribuído ao *gateway* | |
| time | Tempo em que a mensagem foi capturada | |
| rssi | *Receive Signal Strength Indicator* da mensagem recebida pelo LNS | |
| loraSNR | *Signal Noise Ratio* da mensagem recebida | |
| latitude | Coordenadas do *gateway* | |
| longitude | Coordenadas do *gateway* | |
| altitude | Altura do gateway | |
| frequency | Frequência de transmissão | |
| dr | SF de transmissão (dr:2 = SF10) | |
| adr | Estado de configuração do ADR | |
| fcnt | Porta LoRaWAN de recepção | |
| data | *Payload* transmitido pelo dispositivo, neste momento se encontra no formato base64 | |

Fonte: o autor

Para entender o alcance dos recursos de gerência propostos, sem levar em consideração a profundidade de cada um, elaborou-se na Tabela 3 uma comparação entre os principais recursos de rede típicos para sistemas baseado no protocolo SNMP (*Simple Network Management Protocol*), com o sistema aqui sugerido para dispositivos restritos, em destaque.

Tabela 3 - Comparação básica de disponibilidade de recursos

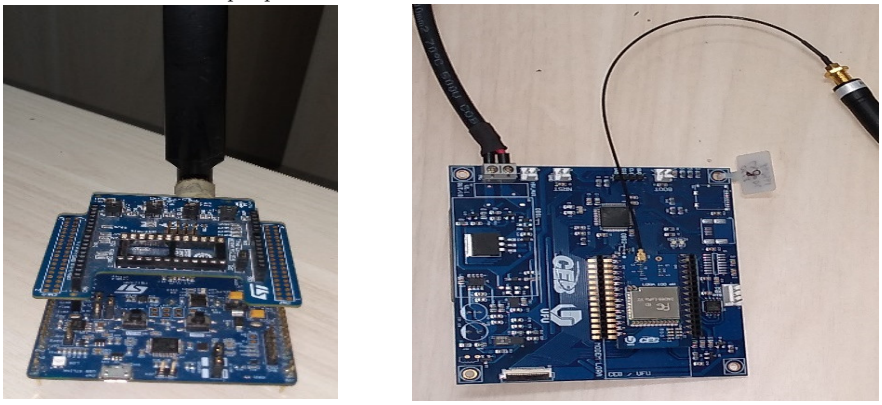
| Recursos típicos | SNMP | Proposto |
|---|---|---|
| Provisionamento de dispositivos | ■ | ■ |
| Análise de tráfego | ■ | ■ |
| Emissão alerta/trap (agente) | ■ | ■ |
| Solicita modificação do agente | ■ | ■ |
| Consulta agente | ■ | ■ |
| Segurança do protocolo | ■ | □ |
| Base de informação | Por tabela MIB | IPSO |
| Reserva de recursos para atendimento a fluxos críticos | ■ | □ |
| Consultar informações de um grupo de objetos | ■ | ■ |
| Análise de congestionamento | ■ | ■ |

Fonte: o autor

### 4.2.2 Recursos de rede, objetos e estrutura de dados da gerência

Como discutido no Capítulo 2, o LwM2M opera funcionalmente os recursos da rede como objetos. Neste sentido, para a aplicação aqui proposta, e com base nos requerimentos funcionais descritos, foram criados alguns objetos que devem suporte à aplicação de gerência de dispositivos. Os objetos e seus recursos construídos seguiram os modelos de referência definidos pelo IPSO e são descritos na Tabela 4.

Figura 12 - Dispositivos finais usados nesta pesquisa. (a) Placa de desenvolvimento da ST para LoRa e (b) projeto próprio de *hardware* usados nesta pesquisa



(a) (b)

Fonte: o autor

Tabela 4 - Objetos LWM2M usados para os principais serviços da gerência de dispositivo

| Operações do dispositivo | Efeito desejado | Objeto | Recurso | Tipo msg CoAP |
|---|---|---|---|---|
| Resetar sem salvar variáveis | Apenas reseta o dispositivo e não recupera seu conteúdo da memória RAM. | 26241 | 1 | POST |
| Resetar salvando variáveis | Reseta o dispositivo salvando as principais variáveis atuais das aplicações na EEPROM para que possam ser devolvidas a RAM após a reinicialização. | | 2 | POST |
| Hibernar 1 dia | Hardware hiberna por 1 dia e depois retorna normalmente. Antes de hibernar, emite confirmação que irá hibernar. | | 3 | POST |
| Hibernar 1 semana | Hardware hiberna por 1 semana e depois retorna normalmente. Antes de hibernar, emite confirmação que irá hibernar. | | 4 | POST |
| Hibernar 1 mês | Hardware hiberna por 1 mês e depois retorna normalmente. Antes de hibernar, emite confirmação que irá hibernar. | | 5 | POST |
| Solicita dados gerais de configuração do dispositivo | Dispositivo retorna dados gerais de configuração do dispositivo como limites de números de pacote diários, qual SF está operando no momento etc. Conforme visto, mais a frente, nesta seção, estes dados estão na estrutura JSON_ESTRUTURA_CONFIG_DISPOSITIVO. | 26242 | 1-17 | GET |
| Muda uma configuração do dispositivo | O dispositivo tem muitas variáveis que podem ser configuradas (ou parametrizáveis) por este recurso. Elas são designadas pelo usuário da aplicação de gerência, sendo executada no servidor que envia o nome da variável ou parâmetro a ser mudado acompanhado de seu novo valor. Estes devem ser um (ou mais) dos campos da estrutura JSON_ESTRUTURA_CONFIG_DISPOSITIVOS. | 26242 | 1-17 | PUT |
| Coloca em DUT (*device under test*) | P_freq_msg = estipula a frequência das mensagens (em minutos). | 26243 | 1 | PUT |
| | P_duracao_do_teste = estipula, em dias, quanto tempo o dispositivo deve estar em DUT. | | 2 | OBSERVER |
| Emitir alarmes | Através do sensor de temperatura do processador, estima-se a temperatura de operação do hardware do dispositivo. Quando | 26244 | 1 | OBSERVER |

| | | | | |
|---|---|---|---|---|
| | ela passar de determinador valor, deve ser emitido uma mensagem de alarme. Este recurso também permite que o servidor consulte o valor de temperatura de operação do dispositivo. | | | |
| | Revela o nível de tensão da alimentação elétrica do dispositivo: se ela tiver abaixo de um determinado valor, deve ser emitido uma mensagem de alarme. Este recurso também permite que o servidor consulte o valor de alimentação. Na falta de energia também é emitido um alarme quando o hardware permitir uma autonomia de energia de pelo menos alguns segundos (armazena carga). | | 2 | OBSERVER |
| PING/PDR | Ping simples. | 26245 | 1 | GET |
| | Múltiplos PING (ou seja, PDR). | | 2 | GET |
| QoS (das últimas 24h) | Quantidade pacotes enviados, últimas 24 horas. | 26246 | 1 | GET |
| | Quantidade de perdas de conexões e reconexões que falharam. | | 2 | GET |
| | Quantidade de *resets* do dispositivo por watchDogTimer . | | 3 | GET |
| | Limite de quantidade de mensagens por dia do dispositivo (vinculado a variável p_limite_msg_dia). | | 4 | GET |
| | Indica o tamanho máximo do pacote (vinculado a variável p_limite_bytes_msg | | 5 | GET |
| | Variável definida pelo servidor que vai limitar a quantidade de envio de pacotes tomando como referência de valor nominal a referência p_limite_msg_dia. (Diminiu o valor de p_limite_msg_dia em 25%, 50%, 75%). Está vinculado a variável p_disponibilidade_envio. | | 6 | GET |

Fonte: o autor

O recurso DUT foi especialmente idealizado para permitir, por exemplo, que um dispositivo fosse deixado por um longo (e frequente) período de teste automático. Isto é importante para dispositivos que poderão ser instalados em lugares de baixa acessibilidade e antes de ir a campo, necessitando ser testado funcionalmente. Logo um de seus intuitos é verificar o desempenho do dispositivo e da rede onde será instalado

antes que o dispositivo seja efetivamente disponibilizado para o serviço/aplicação final. Pode também servir para avaliar a disponibilidade da rede no local onde o dispositivo foi instalado. Quando o dispositivo está em DUT, é programado para enviar uma sequência de pacotes que contém um contador. O servidor monta um gráfico onde mostra os pacotes recebidos em sua ordem e indica os pacotes faltantes. Os recursos de dispositivo listados na Tabela 4 geralmente usam estruturas de dados trocadas entre o dispositivo e a aplicação de gerência no servidor para se obter a funcionalidade desejada. Esta estrutura de pacotes de dados usados para algumas das operações listadas anteriormente são mostradas na Tabela 5.

A formatação destes pacotes é feita em padrão JSON. Embora seja um padrão muito conhecido para a ambiente *web*, onde se executa a aplicação de gerência, não se pode afirmar o mesmo para o *firmware* onde as tecnologias de programação ainda são de mais "baixo nível". Mesmo assim, buscando interoperabilidade e universalidade, este padrão de formatação de dados também foi implementado para ser aceito pelo dispositivo.

Tabela 5 - Estrutura JSON de dados usada para troca de mensagens

| | Variável | Significado | Formato JSON |
|---|---|---|---|
| JSON_ESTRUTURA_CONFIG_DISPOSITIVO_LORA | p_coordenadas_device | Contém latitude e longitude do dispositivo. | Objeto 26242<br>{<br>**(1)**"p_coordenadas_device"=[0,0],<br>**(2)**"p_grupo_disp_gerencia"=0,<br>**(3)**"p_subbanda"=" ",<br>**(4)**"p_porta"=" ",<br>**(5)** "p_AppEUI" = ""<br>**(6)** "p_tecnologia_comuicacao" = ""<br>**(7)**"p_classe_LoRa_operacao"=" ",<br>**(8)**"p_LoRa_ADR_habilitado"=0 ,<br>**(9)**"p_LoRa_SF"=0,<br>**(10)**"p_limite_msg_dia"=0,<br>**(11)**"p_limite_bytes_msg"=0,<br>**(12)**"p_disponibilidade_envio"=0, |
| | p_grupo_disp_gerencia | Cada dispositivo pode ser adicionado a um grupo a ser definido pelo gerente da rede. | |
| | p_subbanda | Define a sub-banda que o dispositivo deve operar. | |
| | p_porta | Define a porta de operação desta aplicação (de gerência). | |
| | p_AppEUI | Chave LoRa de autenticação da aplicação. | |
| | p_tecnologia_comuicacao | Como o sistema é agnóstico, pode permitir outras tecnologias que não sejam LoRa. | |

| | | | |
|---|---|---|---|
| | p_classe_LoRa_operacao | Define qual é a classe de operação da comunicação. | **(13)**"p_versao_*firmware*"="0",<br>**(14)**"p_permissao_servicos_hab"=0,<br>**(15)**"p_comunicacao_habilitada"=0,<br>**(16)**"p_versao_hardware" = "0"<br>} |
| | p_LoRa_ADR_habilitado | Habilita ou desabilita o ADR. | |
| | p_LoRa_SF | Define o spread fator do LoRa quando o ADR é desligado. | |
| | p_limite_msg_dia | Limita a quantidade de mensagens que pode enviar em 24h. | |
| | p_limite_bytes_msg | Define tamanho máximo do pacote (está ligado a diferentes SF e a *time on air*). | |
| | p_disponibilidade_envio | Variável que o servidor configura para indicar ao dispositivo facilidade de envios de pacotes no canal. As opções são:<br>0=pode enviar à vontade respeitando limites<br>1 = deve cair pela metade os limites<br>2 = deve cair em ¼ os limites de envio. | |
| | p_versao_*firmware* | Indicação de qual versão de *firmware* está sendo executado. | |
| | p_permissao_servicos_hab | Usada para desabilitar o dispositivo fazendo com que ele evite o envio de mensagens de aplicações. A aplicação de gerência continua sendo executada para reverter este estado. | |
| | p_comunicacao_habilitada | Quando o dispositivo conseguir fazer um *login* em algum LNS, este campo deverá valer 1 indicando que tem uma conexão ativa. | |
| | p_versao_hardware | Indicação de qual versão de *hardware* está sendo utilizada. | |
| JSON_QOS | p_enviados_acumulados | Este é um contador que envia a quantidade de mensagens enviadas pelo dispositivo. Este contador é acumulativo. | Objeto 26246<br>{<br>**(1)**"p_pacotes_enviados_acumulados"= "", |

| | | | |
|---|---|---|---|
| | p_num_perdas_conexoes_acumulados | Toda vez que o dispositivo perde um *join* ele incrementa esta variável que é acumulativa (como a variável p_contador_pacotes_enviados_acumulados). | **(2)** _num_perdas_conexoes_acumulados = “”, **(3** p_num_tentativas_falhas_conexao ”=” ”, **(4)**” p_numero_resets”=” ”, } |
| | p_num_tentativas_falhas_conexao | Toda vez que tenta fazer um JOIN e não consegue incrementar esta variável. | |
| | p_numero_resets | Contabiliza quantas vezes este *hardware* foi resetado (fora o WDT). | |
| JSON_DUT | p_contador | É um contador incremental que é usado para ver se perdeu ou não um dado, *timestamp* é inserido pelo servidor. | Objeto 26243 { **(1)** “p_timeStamp”=”” **(2)**”p_contador” } |

Fonte: o autor

Para entender melhor como funciona esta interação entre o aplicativo que está sendo executado (no *back end*) do sistema de gerência com o dispositivo, imagine o exemplo: o gerente da rede quer forçar o dispositivo a operar em SF 11, para isto, acontece a seguinte troca de mensagens no formato LwM2M:

- A aplicação envia uma mensagem LwM2M WRITE (CoAP PUT) na URI 26242/0/9, que de acordo com a Tabela 5, corresponde ao recurso de SF do objeto 26242. No *payload* da mensagem é enviado o valor {“data”: “11”}, que como pode ser observado, está no formato JSON.
- Uma vez que esta mensagem chega no dispositivo, este sabe que corresponde a uma atualização do parâmetro de SF, pois pela a URI da mensagem recebida ele tem conhecimento que 26242/0/9, corresponde a SF, e pelo fato da mensagem ser do tipo LwM2M WRITE que significa uma atualização. A mensagem recebida possui um *payload,* no caso do tipo JSON (o formato da mensagem é inserido no campo de opções de uma mensagem CoAP e o *payload* enviado deve ser o mesmo informado no campo de opções), a partir do *payload* o dispositivo realiza uma

operação chamada “PARSE_JSON” e retira o valor 11, correspondente ao SF11, em seguida o rádio do dispositivo LoRa é alterado para teste valor.

- Uma mensagem de confirmação (ACK CHANGED) é retornada ao servidor, indicando que o SF foi alterado com sucesso.

Um outro exemplo do mapeamento de dados, seria o caso de o operador de rede desejar saber qual seria o SF de operação de um determinado dispositivo:

- A aplicação de gerência envia uma mensagem LwM2M READ (CoAP GET) na URI 26242/0/9, lembrando que não há *payload* em requisições do tipo GET.
- Quando a mensagem chega ao dispositivo, ele verifica que se trata de uma extração de informação, pelo fato de ser uma mensagem do tipo LwM2M READ, em seguida ele constata que a requisição é do recurso de SF, pelo fato de ser na URI 26242/0/9.
- Por fim, uma mensagem ACK CONTENT é enviado ao servidor de gerência. Nesta mensagem é enviado um *payload* JSON que informa o SF de operação do dispositivo, considerando o exemplo anterior seria {“data”:”11”}.

É por meio de operações como estas e trocas de mensagens CoAP (ilustradas anteriormente) que acontece o processo de gerenciamento remoto de dispositivos. A criação (ou personalização) de outras operações de gerência deve seguir o mesmo trâmite.

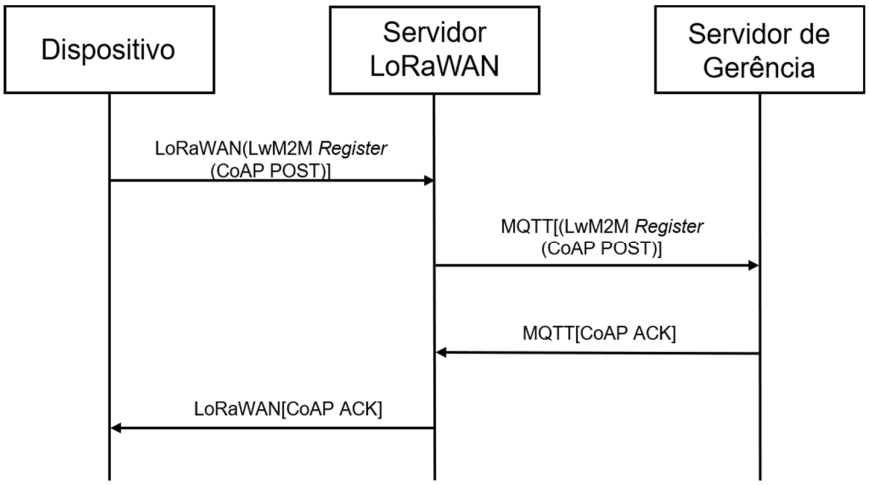
### 4.2.3 Modelo de comunicação usando LoRa e LoRaWAN

Logo no início da autenticação do dispositivo na rede, quando ele é energizado[7], e é feita a autenticação do dispositivo no LNS (ocorreu um JOIN ACCEPT), o *firmware* de gerência de dispositivo passa ao servidor de gerência para quais objetos o dispositivo possui. Esta ação é feita por meio da operação LwM2M REGISTER (usando um CoAP POST). Se a gerência aceitar os objetos em questão (eles precisam necessariamente existir

[7] Neste caso, o dispositivo foi programado para que toda vez que for inicializado tente se conectar automaticamente à rede LoRa disponível.

no servidor), ele devolve uma mensagem CoAP ACK. Após a recepção desta mensagem, o dispositivo deixa de enviar mensagens de registro (elas são enviadas até a chegada de confirmação de recepção por parte do servidor). A partir deste ponto, operações relacionadas ao gerenciamento podem ser realizadas. O fluxo de cadastro é ilustrado na Figura 13. Como declarado anteriormente, a arquitetura de comunicação entre o servidor de aplicação e o LNS é feita por MQTT.

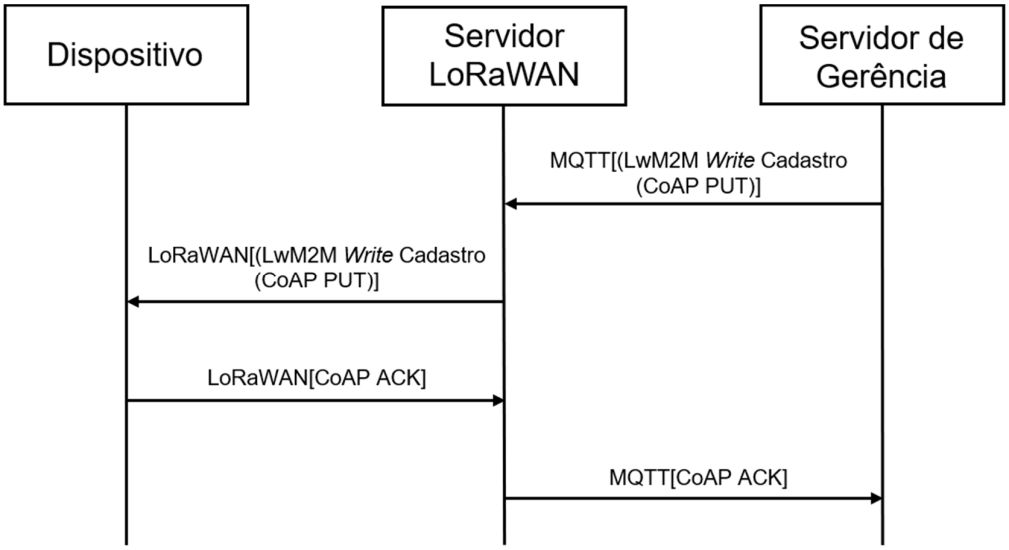
Figura 13 – Exemplo de um registro de dispositivo no padrão LwM2M, dentro do sistema criado




Fonte: o autor

Os dispositivos registrados pelo LwM2M são mostrados na interface gráfica da gerência. Logo que pedem registro, o sistema permite que eles possam ser cadastrados no banco de dados (BD) da aplicação. Após o preenchimento das informações de cadastro, são enviadas ao dispositivo no formato JSON. Alguns campos são informativos e outros configuram o modo de operação do dispositivo, como quantidade de mensagens que podem ser enviadas por dia, habilitar ou não a comunicação, mudar SF, habilitar ou desabilitar ADR, dentre outras opções. Mensagens de cadastro e de atualizações são realizados via LwM2M WRITE (CoAP PUT), como mostrado na Figura 14. Feito isto, o dispositivo responde ao servidor com uma mensagem ACK CHANGED. As atualizações podem ser realizadas em todo cadastro ou em recursos individuais.

Figura 14 - Diagrama de interação para as mensagens de cadastro de dispositivo em padrão LwM2M

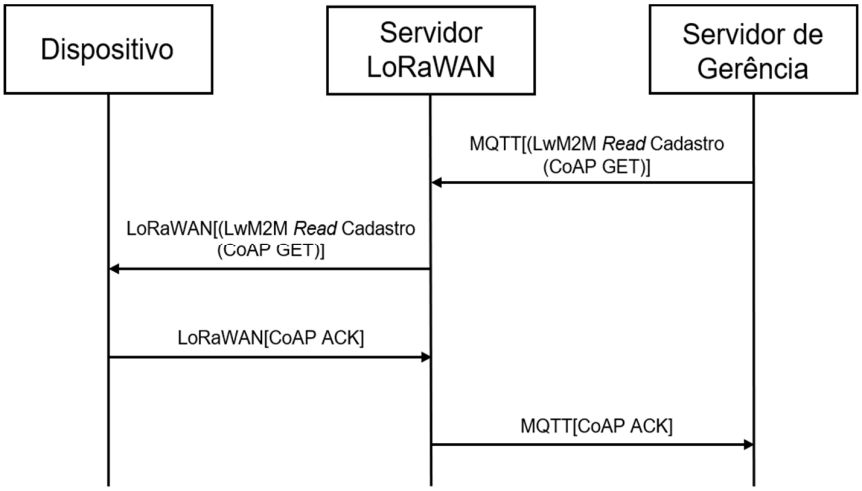



Fonte: o autor

A Figura 15 ilustra a consulta que o servidor faz ao dispositivo para trocar dados de cadastro. A operação de leitura é feita via LwM2M READ (CoAP GET). O dispositivo retorna um ACK CONTENT onde no campo de *payload* desta mensagem se encontra a resposta da requisição feita. Podem ser realizadas requisições de todo o campo de cadastro (26242/0) ou apenas em recursos individuais (26242/0/1, por exemplo).

As operações de DUT e ALARME são feitas por meio da operação LwM2M NOTIFY. Mensagens NON enviadas possuem o *payload* com um contador para identificar perdas para o caso do DUT. No alarme, o dispositivo envia mensagens de notificação apenas quando o nível pré-estabelecido do alarme for ultrapassado (como temperatura alta ou um nível de bateria muito baixo). A operação do alarme também é feita por meio do LwM2M NOTIFY, que é ilustrada na Figura 16 (notificação do LwM2M).

Figura 15 - Requisição do servidor em relação ao cadastro



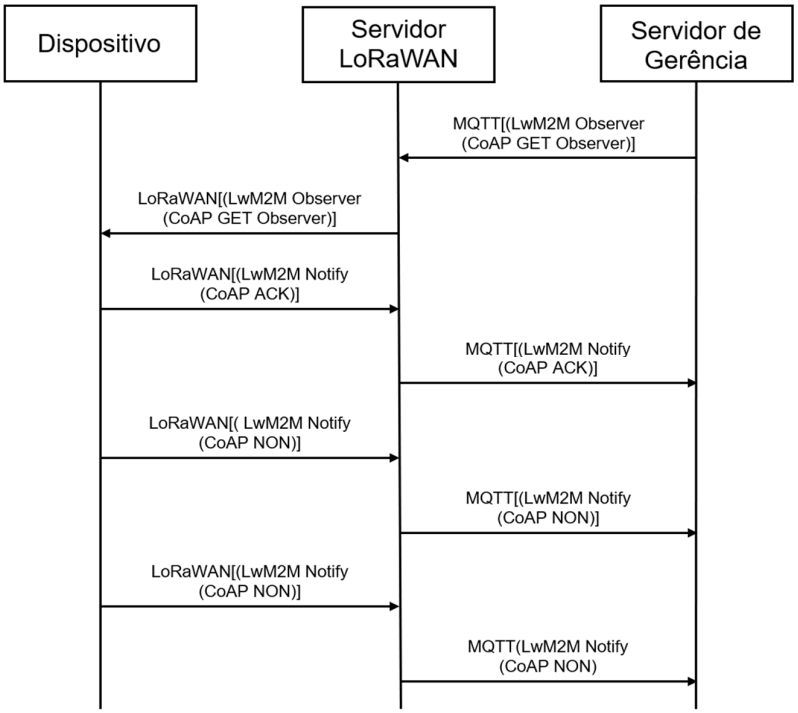

Fonte: o autor

Antes do processo LwM2M NOTIFY, na operação DUT é enviado, por meio de LwM2M WRITE, a quantidade de dias que a operação DUT será executada pelo dispositivo.

As operações de RESET e HIBERNAÇÃO, por exemplo, são realizadas por meio de um pacote LwM2M tipo EXECUTE (CoAP POST), conforme ilustração da Figura 17.

O recurso de PING é utilizado para verificar se o dispositivo está respondendo na rede. Neste caso, o servidor envia um comando LwM2M READ no objeto 26245, recurso de número 1, e aguarda a resposta do dispositivo. A resposta é um ACK do tipo CONTENT.

Figura 16 - Diagrama de interação para as mensagens padrão LwM2M do tipo NOTIFY utilizada para DUT e ALARMES

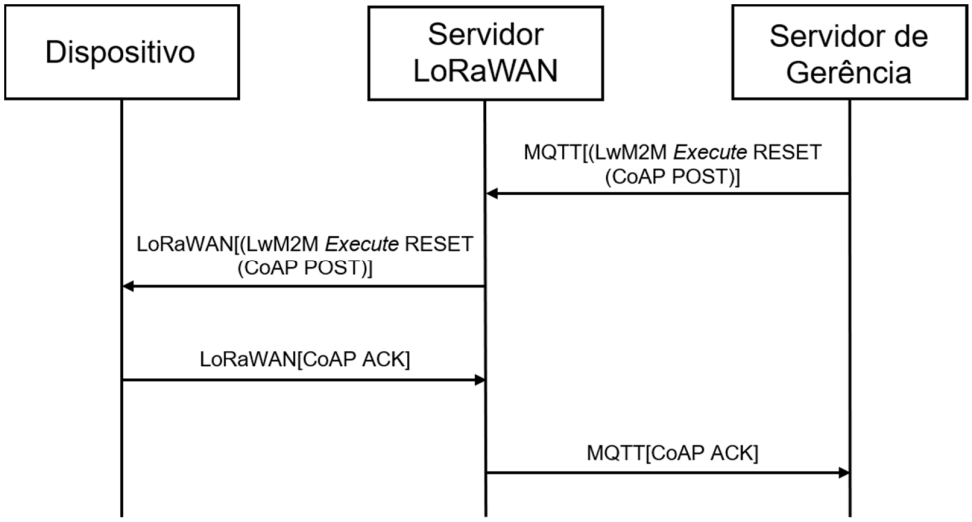



Fonte: o autor

A operação de PDR (*Packet Delivery Ratio*) é derivada do PING, pois é uma sequência de requisições, que visa analisar a disponibilidade da comunicação entre dispositivo e servidor. O usuário da gerência insere o número de pacotes que deseja testar a PDR, em seguida o sistema envia a quantidade requisições PING configuradas, e a gerência, então, aguarda os ACKs destas mensagens. No final é mostrada a relação entre a quantidade de pacotes enviados e respostas recebidas. Se foram enviados 10 pacotes, por exemplo, e chegaram 10 ACKs, a PDR é de 100%; se chegarem 5 ACKs, a PDR seria de 50%.

O objeto QoS 26246 é requisitado pelo servidor uma vez por dia, por meio de uma operação LwM2M READ.

Figura 17 - Diagramas de interação para as operações de reset e hibernação




Fonte: o autor

### 4.2.4 Diferenças entre o sistema proposto e o protocolo LwM2M

O protocolo LwM2M estabeleceu a forma como foi feita a comunicação entre o servidor de gerência com o dispositivo, ambos executando o protocolo CoAP. Como foi ilustrado, determinadas ações foram feitas com CoAP GET, outras com CoAP POST e assim por diante. Além disso, mensagens enviadas pelo dispositivo e servidor precisaram obedecer estritamente ao protocolo LwM2M. Embora a implementação LwM2M foi de fato realizada no dispositivo e também no servidor de gerência, o sistema de gerência criou uma série de objetos e funções para atender o que se imagina de um gerência de dispositivo restritos não IP.

O mapeamento de objetos e recursos foram criados tendo como referência o IPSO *objets*. Dentro da sua especificação é estabelecido que, da numeração 26241 até 32768, objetos podem ser criados e reusados por diferentes indivíduos. Com relação à numeração dos recursos, o IPSO estabelece que, dentro da numeração de 0 até 2047, recursos podem ser criados e reusados em diferentes objetos, como pode ser observado na Tabela 4. Tudo isto foi obedecido.

Vários objetos foram criados e seus recursos estabelecidos pelo IPSO, como um sensor de temperatura genérico que é mostrado na Seção 2.3.2. Há centenas de objetos criados, assim como os seus recursos. As operações que estes objetos alcançam são diversos, desde casos genéricos de sensores como acelerômetro, magnetômetro e sensores de pressão, até casos mais específicos, como um objeto para manipular comandos AT, criado pela empresa CISCO, no caso o objeto 10251 [11]. O IPSO também provê uma forma para que os indivíduos ou as empresas possam comprar uma determinada numeração de objetos e recursos, como foi o caso citado da CISCO.

Nas operações criadas para o sistema de gerência proposto, alguns deles já existem na nomenclatura do IPSO; por exemplo, o objeto número 3, no recurso 4, estabelece a operação *reset*. Porém, durante a pesquisa, esta operação foi estabelecida dentro do objeto 26241, recurso 1. Ressalta-se que não é apenas esta operação, outras também já existem dentro do IPSO. O objeto 3 estabelecido pela IPSO possui 22 recursos, muitos destes não seriam necessários nas operações de gerência definidas neste trabalho. Outras funções existentes dentro da proposta de pesquisa estão espalhadas por diversos objetos criado pelo IPSO. A escolha durante a pesquisa foi, portanto, condensar o máximo de operações (recursos) dentro de poucos objetos, nos quais se teria a certeza de que serão utilizados, uma vez que o sistema de gerência é otimizado para dispositivos restritos nos quais processamento e memória não podem ser desperdiçados.

Outros recursos, como a parametrização da comunicação LoRA (mudança de ADR, SF e classe operacional), foram criados ao longo da pesquisa. O objeto que provê estes recursos, de número 26242, possui o recurso de coordenadas geográficas, que é o recurso 1; que por ser bastante comum, já existia no IPSO, no objeto 6 recurso 1. Portanto, se não tivéssemos criado um objeto com todas as informações e operações LoRa (26246) em um único objeto, teríamos que utilizar vários de forma desnecessária.

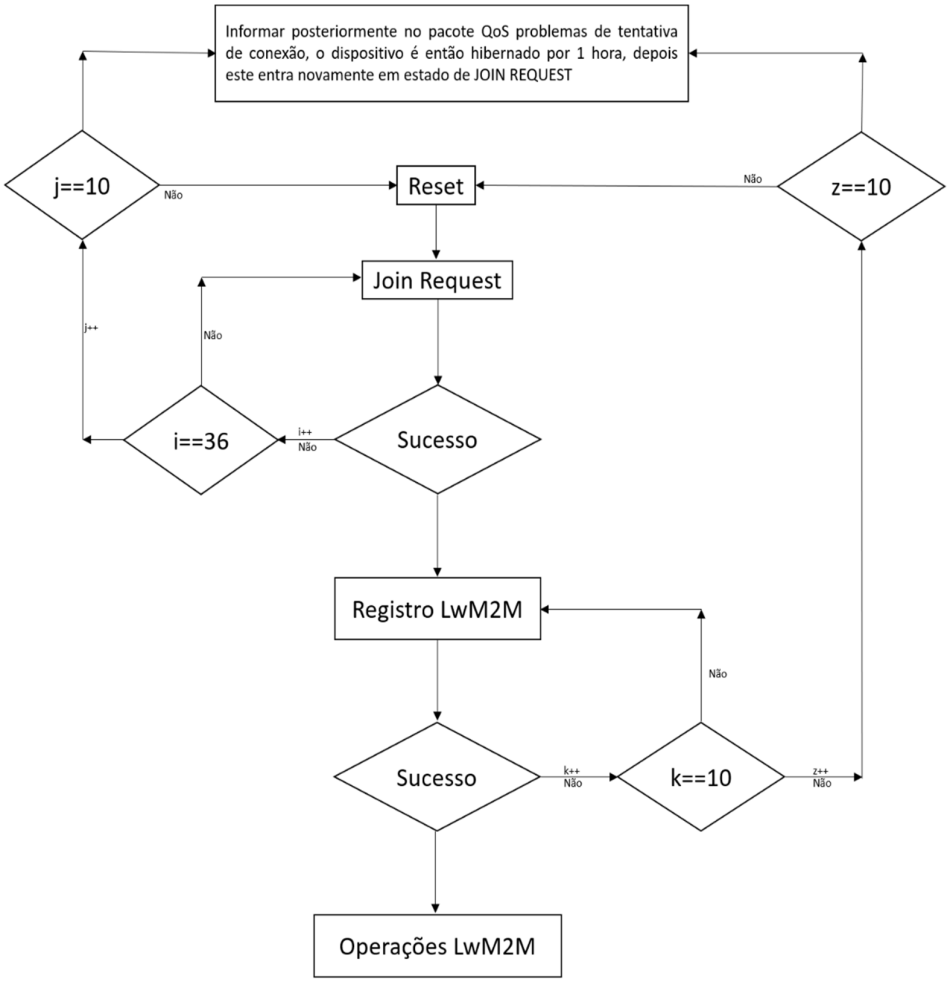
### 4.2.5 Estados de comunicação da gerência no dispositivo

Como era de se esperar, para que a gerência seja efetiva, o dispositivo deve estar constantemente conectado. Por isto, logo após iniciado o dispositivo, deve-se solicitar conexão/autenticação LoRa a um servidor LNS. O dispositivo inicializa com mensagens JOIN REQUEST enviadas pelo dispositivo ao LNS. Se obter sucesso nesta tarefa, o dispositivo começa a enviar mensagens de registro ao servidor de gerência. Se houver falha no processo de JOIN REQUEST, o dispositivo tenta novamente por 36 vezes (aproximadamente 6 minutos). O mesmo procedimento é feito para eventuais problemas no registro. Neste caso, o dispositivo tenta se registar no máximo 10 vezes. Após estas falhas consecutivas, é reiniciado. Em casos extremos onde nem o processo de JOIN ou REGISTRO é concluído, o dispositivo é configurado para hibernar por um tempo de 1 hora suas informações a respeito das falhas, que também são salvas. Posteriormente, quando sair do estado de hibernação, ele retorna o processo de JOIN REQUEST. A Figura 18 ilustra os estados deste procedimento.

Após o término das operações previstas, o dispositivo vai para um estado de "espera" onde deverá estar pronto para receber a qualquer momento (*i.e.*, assíncrono) uma requisição da gerência ou tratar algum evento associado às funções locais. As requisições que o dispositivo aguarda da gerência são do tipo LwM2M WRITE, READ e NOTIFY.

Uma função local que o dispositivo realiza é o controle de LIFETIME. Dispositivos e gerência precisam ter conhecimento se estão acessíveis entre si, ou seja, se o canal de dados está disponível de maneira bidirecional. Para manter o enlace de comunicação funcional, o dispositivo LwM2M envia uma mensagem POST LIFETIME de forma periódica. Após receber a mensagem, a gerência renova o dispositivo por um determinado tempo (este tempo é informado na mensagem de registro) e devolve um ACK ao dispositivo.

Figura 18 - Algoritmo de inicialização (no *firmware*) do serviço de gerência de dispositivo




Fonte: o autor

Se ocorrerem 3 perdas de ACK, é considerado que a conexão foi perdida, então o dispositivo é reiniciado, sendo necessário que seja registrado novamente. Este processo é automático no dispositivo. No servidor, passado o tempo estabelecido da mensagem de registro sem que tenha chegado uma mensagem de POST LIFETIME, o dispositivo é desconectado. Na mensagem de registro é enviado um tempo de 5400 segundos (1 hora e meia), ou seja, após este tempo que o dispositivo não atualiza seu status, o servidor o excluí do sistema. De 30 em 30 minutos o dispositivo envia uma mensagem de POST LIFETIME, para que o servidor renove o *status* de conectado do dispositivo. Portanto após 90 minutos que o dispositivo fique sem receber um ACK do servidor, ele será automaticamente reiniciado. Desta forma, o servidor e o dispositivo possuem o

conhecimento se estão acessíveis entre si. Estes valores de tempo foram escolhidos durante a pesquisa, de forma que não impactasse no desempenho dos dispositivos restritos.

## 4.3 Experimentos e validação

Foram desenvolvidos alguns tipos de testes para validar a estrutura básica construída. Estes testes estão resumidos na Tabela 6:

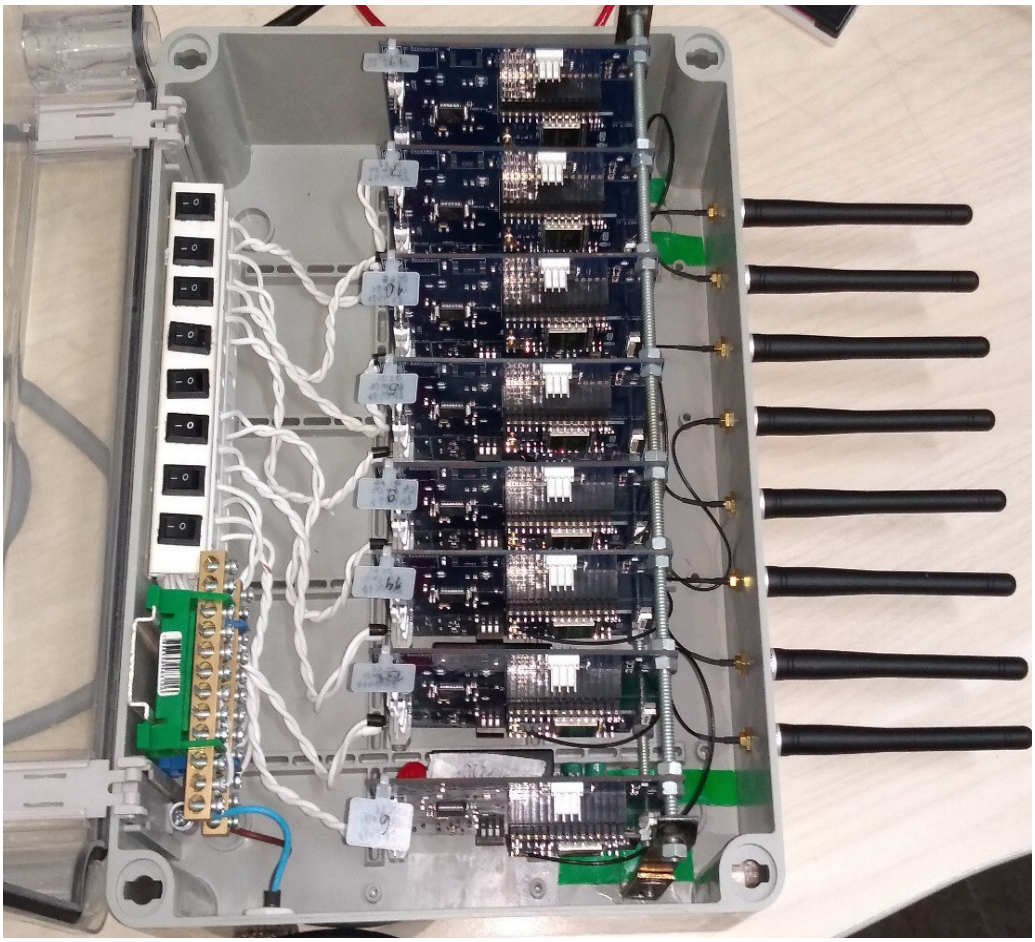
Tabela 6 - Resumo dos experimentos realizados com o sistema construído

| | Descrição resumida do experimento | Nº cenários testados | Objetivo(s) do experimento |
|---|---|---|---|
| 1 | Comissionamento de dispositivos | 6 | • Avaliar a capacidade do sistema em identificar automaticamente novos dispositivos.<br>• Avaliar a facilidade e efetividade no cadastramento e observação de dispositivos (DUT). |
| 2 | Operações em dispositivos e análise de tráfego | 3 | • Avaliar a capacidade do sistema em fazer efetivamente operações remotas nos dispositivos.<br>• Avaliar a capacidade do sistema em fazer uma estimativa do tráfego de cada dispositivo. |
| 3 | Estresse de serviço e de ambiente | 2 | • Avaliar os limites de operação da comunicação com o dispositivo.<br>• Considerar como o sistema reage a fatores externos (temperatura e alimentação elétrica). |

Fonte: o autor

Para todos os testes foram levantados estatisticamente a quantidade de sucessos e fracassos. Tentou-se, ainda, variar as condições de operação dos dispositivos. A Figura 19 ilustra os dispositivos usados para os experimentos propostos, que estão descritos nos próximos subtópicos. Todos eles foram executados na cidade de Uberlândia durante variados turnos, períodos e condições ambientais.

Figura 19 - Conjunto de dispositivos LoRa usados para testar as capacidades do sistema de gerência desenvolvido



Fonte: o autor

### 4.3.1 Experimento 1: comissionamento de dispositivos

Neste experimento, os dispositivos foram ligados e desligados várias vezes (hora aleatoriamente, hora simultaneamente), para quantizar a taxa de sucesso de registro e habilitação dos dispositivos para a plataforma. O número de falhas também foi levantado. Foram considerados os seguintes cenários, sendo que cada um deles foi repetido 10 vezes:

- Cenário 1A: 9 dispositivos juntos fisicamente (8 dentro de uma caixa mais um ao lado) e ligados simultaneamente;
- Cenário 1B: 22 dispositivos juntos fisicamente (duas caixas contendo 8 dispositivos mais 6 dispositivos ao lado) e ligados simultaneamente;

- Cenário 1C: 5 dispositivos espalhados geograficamente mais 9 dispositivos próximos a um *gateway* ligados ao mesmo tempo;
- Cenário 1D: 8 dispositivos juntos fisicamente (condicionados em uma caixa) e ligados aleatoriamente dentro de um intervalo de tempo de até 2 minutos;
- Cenário 1E: 5 dispositivos espalhados geograficamente mais 9 dispositivos próximos a um *gateway* e ligados aleatoriamente.
- Cenário 1F: um dispositivo operando em ambiente com considerável interferência de equipamento de potência (interferência eletromagnética). No caso, os dispositivos foram posicionados a até 50cm de um transformador de 13,8KV e 300KVA. Foi realizado uma operação DUT neste ambiente.

### 4.3.2 Experimento 2: operações em dispositivos LoRa, análise de tráfego e outros recursos da plataforma idealizada

Neste experimento foi avaliada a capacidade do sistema em executar suas operações previstas em diferentes cenários. Embora a quantidade de dispositivos seja relativamente limitada, foi reconstituída a estimativa de tráfego de cada dispositivo e avaliada se ela corresponde com a registrada no servidor. Os cenários de testes são os seguintes para este experimento:

- Cenário 2A: 5 dispositivos espalhados geograficamente foram simultaneamente colocados em DUT por um período de 2 dias, com frequência de mensagens de 1 minuto, espalhados por diferentes pontos dentro do raio de cobertura de um *gateway*.
- Cenário 2B: como o sistema proposto tem capacidade de registro dos KPI dos últimos pacotes do dispositivo, foram divididos em grupos menores e colocados a distâncias gradativas do *gateway* para que seus KPI e parâmetros de rádio variassem. Foi avaliada a capacidade do sistema para registrar este KPIs e as

variações que são usadas para informar ao gerente da rede a condições da comunicação de cada dispositivo.

- Cenário 2C: neste cenário foram avaliadas outras operações de dispositivos. Elas foram: (i) mudar o SF e (ii) aplicar PDR (de 10 pacotes) aos dispositivos. Cada aplicação foi repetida 10 vezes em cada dispositivo e avaliada a taxa de sucesso destas operações nos dispositivos. As operações foram aplicadas utilizando a operação por grupos.

### 4.3.3 Experimento 3: estresse e disponibilidade de serviço

Neste cenário são avaliados alguns aspectos que podem ser fatores estressantes ao serviço de gerência de dispositivo construído ou ao dispositivo fisicamente. Como a implementação do *software* executado no servidor não foi feito nesta pesquisa, ele não será avaliado. Foram estes os cenários testados:

- Cenário 3A: neste cenário dois tipos de alarmes são testados: a (i) queda da alimentação elétrica e a (ii) fora de faixa de operação em temperatura. Todos esses fatores são controlados. Foi avaliada a capacidade do sistema em gerar estes alarmes. Cada dispositivo foi submetido a estas condições 10 vezes e foram contabilizadas quantas delas foram registradas no servidor.
- Cenário 3B: neste cenário é avaliado a capacidade de tratar o tráfego da rede. O *gateway* tem capacidade de processar fisicamente cerca de 6.000 mensagens por hora, pelo menos. Foi gerado um grande tráfego de rede com 21 dispositivos enviando mensagens de 10 em 10 segundos, durante 2 horas. Foi contabilizado no servidor a quantidade de pacotes recebidas durante este período.

## 4.4 Resumo do capítulo

Foram apresentados nesse capítulo detalhes metodológicos de como o trabalho foi tecnicamente desenvolvido e quais elementos foram usados para sua implementação e

testes. Em especial, foi dado maior ênfase aos procedimentos utilizados para se executar o sistema de gerência definido. O fluxo de informação da rede e elementos de infraestrutura também foram tratados, assim como o LwM2M, IPSO e CoAP foram usados e como podem servir à tecnologia LoRa para gerenciamento destes tipos de dispositivos. Foram apresentados, ainda, vários cenários de testes. No próximo capítulo estão exibidos os resultados dos testes e feitas as devidas conclusões que a experiência deste trabalho permitiu.

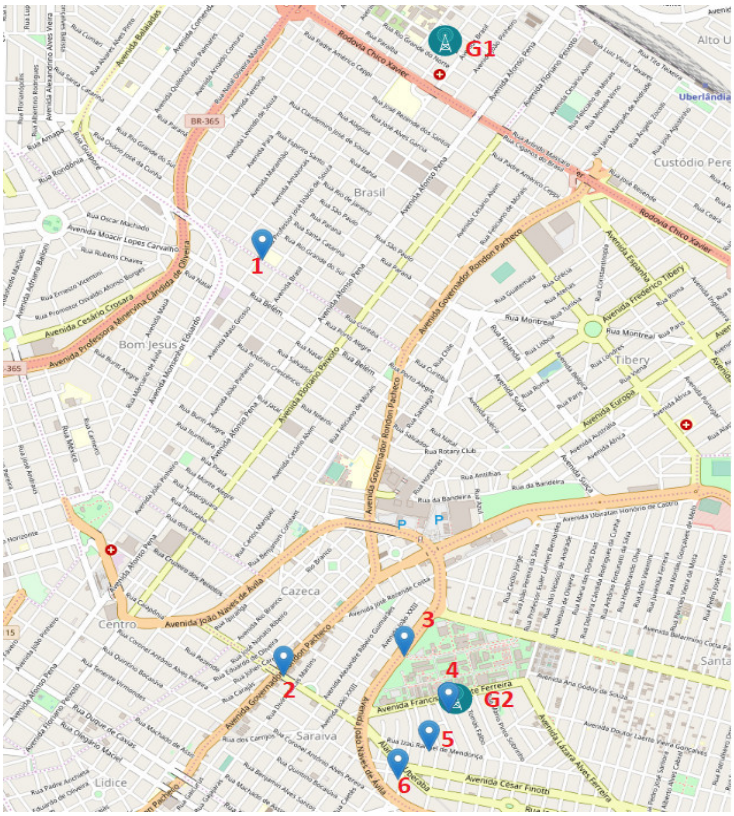
# 5 Resultados e discussões

Este capítulo descreve os resultados dos experimentos realizados segundo os apontamentos feitos no Capítulo 4. Foram realizados diversos testes, onde foi explorado o sistema de gerência de dispositivos proposto. Além dos resultados, foram feitas discussões com base nos experimentos.

## 5.1 Resultados

A configuração dos testes foi da seguinte forma: a caixa com os *end-devices*, ilustrada na Figura 19, foi posicionada a poucos metros de um *gateway* LoRaWAN e a alguns quilômetros (~4,2km) de outro *gateway*. Um outro dispositivo (do mesmo modelo dos que estão na caixa) também foi posicionado ao lado desta caixa, totalizando assim 9 dispositivos. Outros 5 foram dispostos ao longo da cidade de Uberlândia, desta forma, em alguns testes, foram utilizadas 14 placas (dispositivos). Todos os dispositivos utilizados nos testes possuem a mesma configuração de *hardware* e *software*. A Figura 20 ilustra estes dispositivos sendo visualizados pela aplicação da gerência. Apenas 6 pontos são visíveis, pois no ponto 4 (em vermelho), há 9 dispositivos (oito dentro da caixa e mais um que se encontra fora da caixa), dentro de um curto espaço. O ícone em formato de torre são os dois *gateways* LoRaWAN utilizados na pesquisa. A Tabela 7 ilustra a distância de cada um dos dispositivos em relação aos *gateways,* a altitude em que os dispositivos se encontram (valor aproximado) e se eles possuem visada para os *gateways*. As alturas dos *gateways* G1 e G2 mostrados na Figura 20, foram: 950 metros e 855 metros respectivamente.

Figura 20 - Espalhamentos dos dispositivos utilizados nos testes




Fonte: o autor

Tabela 7 - Distância em metros de cada dispositivo em relação aos gateways, altura dos dispositivos e status de visada

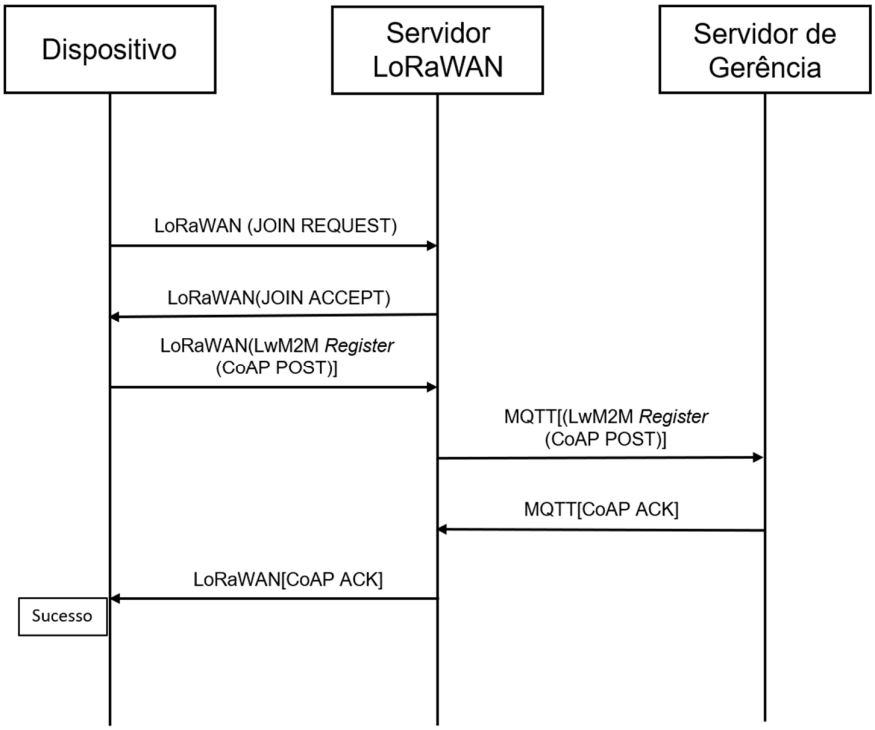
| Dispositivo | Distância até G1 | Distância até G2 | Altura do dispositivo (s) | Visada |
|---|---|---|---|---|
| 1 | 1462m | 2385m | 888 m | Não |
| 2 | 3620m | 859m | 835 m | Não |
| 3 | 3437m | 382m | 850 m | Não |
| 4 (9 dispositivos) | 3746m | ~10m | 873 m | Não |
| 5 | 3792m | 244m | 872 m | Não |
| 6 | 4135m | 455m | 868 m | Sim |

Fonte: o autor

O primeiro teste realizado foi o cenário 1A, onde os dispositivos (número 4, Tabela 7) foram concentrados geograficamente próximos ao *gateway* G2 e tentaram se registrar na plataforma de gerência logo após ligados. Eles foram ligados simultaneamente. O estado de "sucesso" é considerado quando o dispositivo realiza um JOIN REQUEST no

servidor LNS e, então, recebe uma mensagem de JOIN ACCEPT e, em seguida, envia uma mensagem LwM2M REGISTRO ao servidor de gerência, recebendo uma mensagem CoAP ACK. A Figura 21 ilustra o procedimento. Esta é uma funcionalidade de *plug and play* do sistema construído, onde basta o usuário ligar o dispositivo na energia e ele se encarrega de todo o resto.

Figura 21 - Fluxo de comunicação JOIN REQUEST e LwM2M REGISTER




Fonte: o autor

Assim que o dispositivo recebe a mensagem CoAP ACK da requisição de REGISTRO, ele para de enviar requisições deste tipo e fica pronto para ser gerenciado. Na solução construída, o sistema de gerência só recebe mensagens de registro após conexão estabelecida com o LNS.

O teste foi realizado 10 vezes consecutivas, com 9 placas (número 4 da Tabela 7). Foi verificado em cada uma destas 10 vezes, a quantidade de sucessos, o tempo gasto até o dispositivo atingir o estado de sucesso e quantas vezes ele enviou a mensagem de JOIN REQUEST até atingir o estado de sucesso. Para cada um dos 10 testes, os resultados de

cada dispositivo foram compilados e foi realizada uma média, o resultado é resumido na Tabela 8.

Tabela 8 - Resultado do cenário 1A

| Quantidade de sucessos média | Tempo médio para dispositivos atingirem o estado de sucesso (aproximado) | Quantidade média de JOIN REQUEST |
|---|---|---|
| 100% | ~228.4s | 10 |

Fonte: o autor

No cenário 1B, 22 placas foram ligadas por 10 vezes consecutivas, assim como no teste 1A. A Tabela 9 mostra o resultado médio da taxa de sucesso (JOIN REQUEST seguido de LwM2M REGISTER), quantidade média de JOIN REQUEST por teste e o tempo médio para atingir o sucesso. Todos estes dispositivos se encontravam próximos ao *gateway* G2 e foram ligados simultaneamente.

Tabela 9 - JOIN REQUEST e tempo médio para os testes do cenário 1B

| Quantidade de sucessos média | Tempo médio para dispositivos atingirem o estado de sucesso (aproximado) | Quantidade média de JOIN REQUEST |
|---|---|---|
| 100% | ~586s | 24.42 |

Fonte: o autor

No cenário 1C, parte dos dispositivos foram espalhados geograficamente, de acordo com a Figura 20, e outros 9 dispositivos (número 4, Tabela 7) estão próximo do *gateway* G2. Neste caso, o espalhamento foi feito para tentar simular diferentes condições de potência de sinal e ver como o sistema de gerência reagia aos casos (e não a densidade de dispositivos). Estes 14 dispositivos, no total, foram ligados ao mesmo tempo por 10 vezes consecutivas (o processo de ligar os dispositivos ao mesmo foi realizado manualmente por pessoas). O resultado mostra que nem sempre foi possível obter uma taxa de 100% de sucesso, pois o termo sucesso neste teste tem o mesmo significado dos testes 1A e 1B. É considerado falha quando os dispositivos enviam mensagens de JOIN REQUEST e não receberam a mensagem de JOIN ACCEPT, ou quando os *end-devices*

enviam mensagens de LwM2M REGISTER e não recebem a mensagem de CoAP ACK do servidor de gerência. Em resumo, se o dispositivo não conseguiu atingir o estado de "sucesso" em 6 minutos é considerado falha, mesmo que, potencialmente, possam conseguir o sucesso em tentativas posteriores. Apenas falhas do tipo JOIN REQUEST e JOIN ACCEPT aconteceram. O resumo dos resultados é ilustrado na Tabela 10, onde em cada uma das 10 vezes que os 14 dispositivo foram ligados é compilado o resultado, por exemplo, o teste 1 possui 85.7% de taxa de sucesso, isto significa que dos 14 dispositivo que estavam no teste, 12 conseguiram chegar ao estado de sucesso em 6 minutos (lembrando que dentro de 6 minutos se o dispositivo não conseguir realizar o registro no servidor de gerência, ele é reiniciado conforme é explicado na Seção 4.2.6).

Tabela 10 - Resultado do cenário 1C

| Repetição | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Taxa de Sucesso | 85.7% | 92.8% | 92.8% | 100% | 100% | 85.7% | 100% | 100% | 100% | 85.7% |

Fonte: o autor

No cenário 1D, dispositivos (número 4, Tabela 7) foram ligados e desligados aleatoriamente dentro de um intervalo de 2 minutos por 10 vezes consecutivas, o resultado deste teste foi 100% de reconhecimento de registros (sucesso).

No cenário 1E, todos os dispositivos da Tabela 7 foram ligados e desligados de forma aleatória e de maneira remota pelo próprio sistema de gerência, por meio do comando LwM2M RESET. Foram criados dois grupos, um para os dispositivos próximos do *gateway* G2 (Figura 19) e um outro grupo para os dispositivos espalhados (1,2,3,5 - Figura 20), que foram criados dentro da aplicação de gerência. A ordem deste teste foi: reiniciar todos ao mesmo tempo, reiniciar em grupo e individualmente. Estes testes foram repetidos nesta ordem até atingirem um total de 10 tentativas. Dentro do tempo de 12 minutos, todos os dispositivos conseguiram novamente entrar na gerência automaticamente (JOIN REQUEST seguido de LwM2M REGISTRO com sucesso), alcançando um sucesso de operação de 100% em todos os testes.

O cenário 1F analisa o dispositivo exposto ao ar livre, ao lado de um transformador de tensão de 500KVA operando com 13.800V no primário, como mostra a Figura 22. A alimentação elétrica foi tirada diretamente no secundário do transformador.

Foi realizada uma operação DUT neste dispositivo durante um período de 2 dias, 12 horas e 28 minutos. Ele foi configurado para que o envio de mensagens fosse de minuto em minuto, com SF fixado em SF10. No total, foram transmitidas 3.623 mensagens. Destas, chegaram ao servidor 3.533, ou seja, uma assertividade de 97.5%. O transformador em questão se situa próxima do gateway G2, contudo, ambos os *gateways* receberam seus dados, ainda que houvesse prédios e árvores obstruindo a visada direta.

No cenário 2A, realizou-se o DUT para os dispositivos espalhados (1, 2, 3 e 5 da Figura 20). A Figura 23 e Figura 24 ilustram o melhor e o pior desempenho desta operação, respectivamente. O melhor resultado foi no dispositivo 6 e o pior no dispositivo 2. Como a quantidade de dados transmitidos foi grande, foi ilustrado graficamente apenas a última hora de cada teste para se ilustrar a "tendência".

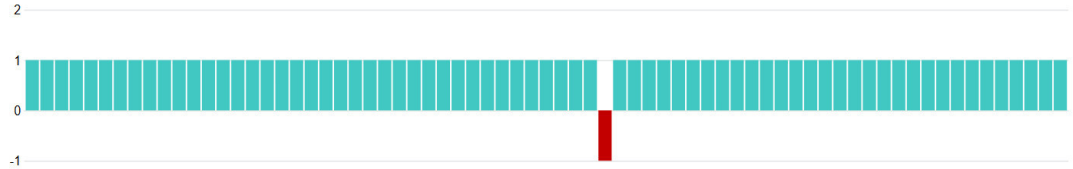
A Tabela 11 mostra a quantidade total de pacotes recebidos na gerência, e o número total de mensagens que cada dispositivo deveria ter transmitido. A operação de DUT foi executada por 2 dias e 10 horas de forma consecutiva. Mensagens de DUT foram configuradas para serem enviadas de minuto a minuto em SF10.

Nas mensagens de DUT enviados há um contador, que a cada mensagem enviada foi somado o valor 1 (isto é feito no dispositivo), desta forma que foi feita a análise gráfica do DUT. Nas Figura 23 e Figura 24, barras azuis significam pacotes recebidos corretamente, por exemplo, pacote 1, 2, 3 recebidos certos. A barra vermelha indica quantos pacotes foram perdidos, por exemplo, pacote 1, 2 ,3 e 5 recebidos, isto significa que vão haver três barras azuis consecutivas representando 1, 2, 3 e depois teremos uma barra vermelha com valor de 2, porque foram perdidos 2 pacotes.

Figura 22 - Dispositivo colocado ao ar livre ao lado de um transformador

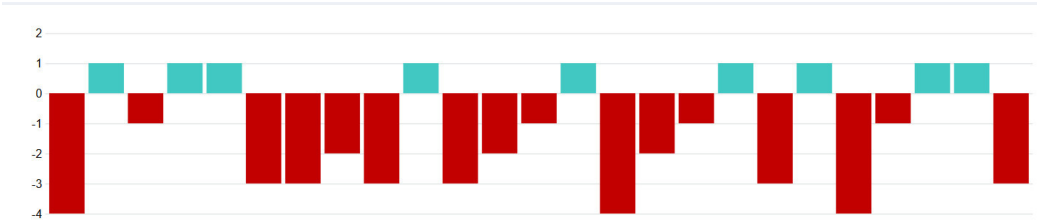

Fonte: o autor

Figura 23 – Última hora de teste do DUT no melhor caso, dispositivo 6




Fonte: o autor.

Figura 24 - Última hora de teste do DUT no pior caso, dispositivo 2.




Fonte: o autor

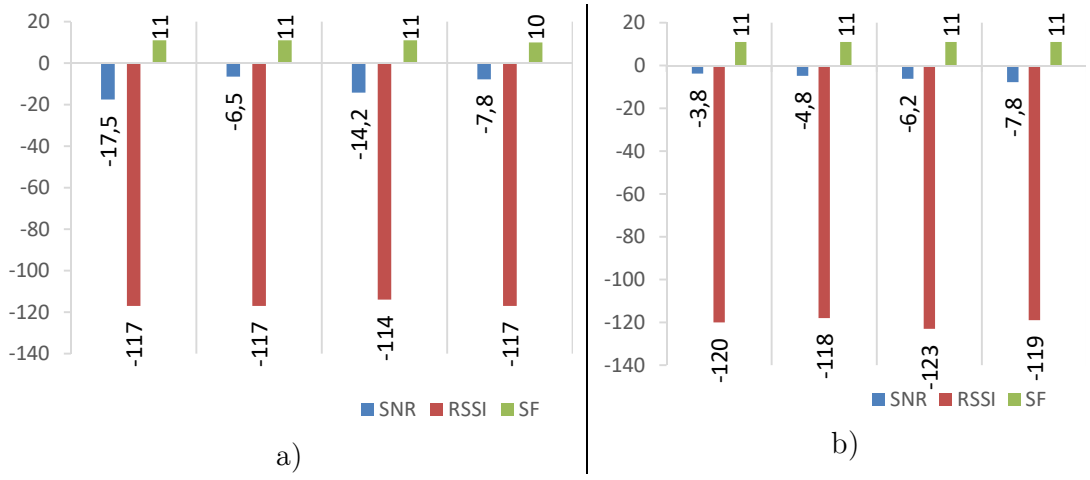
Tabela 11 - Relação de pacotes chegados e perdidos no DUT, para cada dispositivo

| Dispositivo | Total de pacotes que foram recebidos | Total de pacotes que deveriam ser enviados | Taxa de pacote recebidos (%) |
|---|---|---|---|
| 1 | 2576 | 3094 | 83% |
| 2 | 2022 | 3475 | 58% |
| 3 | 2922 | 3489 | 83.7% |
| 5 | 3081 | 3489 | 88.3% |
| 6 | 3400 | 3487 | 97.5% |

Fonte: o autor

Em conformidade com o cenário 2B, a Figura 25 ilustra os parâmetros de KPI (SNR, RSSI, SF) do dispositivo 2, dos últimos 4 pacotes recebidos pelos dois gateways. Para qualquer dispositivo cadastrado é possível verificar o nível de sinal dos últimos pacotes que chegaram nas células (gateway). Dentro da gerência é possível visualizar os gráficos de SNR, RSSI, SF, separados por dispositivo e gateway, porém, para uma melhor visualização, o gráfico aqui mostrado foi feito no software Excel, com base nos dados da gerência.

Figura 25 - Nível do sinal do dispositivo 2 no gateway G1 a) e G2 b) respectivamente, da esquerda para direita informação de SNR, RSSI e SF.




Fonte: o autor

No cenário 2C foram criados dois grupos iguais aos do cenário 1E. O resultado da mudança de SF para os dispositivos em ambos os grupos durante as 10 repetições, obteve 100% de sucesso. Na Figura 25a é possível verificar esta mudança de SF, que ocorreu no

dispositivo 2. Ainda dentro do teste 2C, a Tabela 12 ilustra o resultado de PDR para dispositivos espalhados (1, 2, 3, 5 Figura 20).

Tabela 12 - Resultado da PDR para dispositivos espalhados

| Dispositivos | 1 | 2 | 3 | 5 | 6 |
|---|---|---|---|---|---|
| PDR (%) | 90 | 89 | 98 | 98 | 100 |
| Latência(segundos) | 17.998 | 29.618 | 21.918 | 19,47 | 19.1 |

Fonte: o autor

A Tabela 13 corresponde aos valores de PDR dos dispositivos, de número 4 (Tabela 7) que se situam próximos ao *gateway* G2. O teste de PDR foi realizado com 10 pacotes, tanto para os dispositivo espalhados como os próximos do *gateway*.

Tabela 13 - Resultado da PDR para dispositivos dentro da caixa

| Dispositivos | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| PDR (%) | 98 | 95 | 97 | 91 | 91 | 96 | 93 | 96 |
| Latência(segundos) | 24.64 | 24.354 | 22.92 | 27.346 | 24.859 | 24.899 | 26.125 | 29.484 |

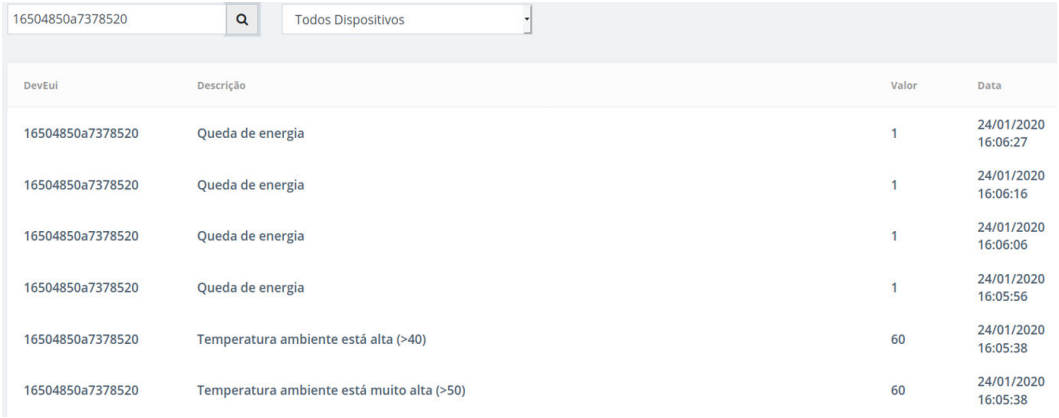
Fonte: o autor

No teste 3A foram testados os alarmes que podem ser configurados no dispositivo. Foi simulado um alarme de temperatura, onde a temperatura operacional do dispositivo superou um limite pré-definido e outro alarme foi configurado para alertar o usuário da gerência sobre uma queda de energia[8]. O alerta de queda de energia foi criada simplesmente retirando a fonte de alimentação primária do dispositivo (rede AC). Os testes foram realizados com os dispositivos dentro da caixa e próximos ao *gateway* G2 (número 4, Tabela 7). O alarme de temperatura foi disparado antes dos dispositivos serem retirados da tomada.

[8] Como o dispositivo tinha um circuito de '*last gasp*' baseado em carga capacitiva, ele tinha uma autonomia média de 2 minutos mesmo após a queda de sua energia. Este tempo era usado para enviar mensagens de alerta ao servidor de gerência de dispositivos.

A Figura 26 ilustra os alertas na gerência enviados pelos dispositivos. Nas 10 oportunidades em que os alarmes foram gerados, eles chegaram e foram mostrados pelo servidor de gerência em todos os casos.

Figura 26 - Mensagens de alarme disparados por um dispositivo mostrados no servidor

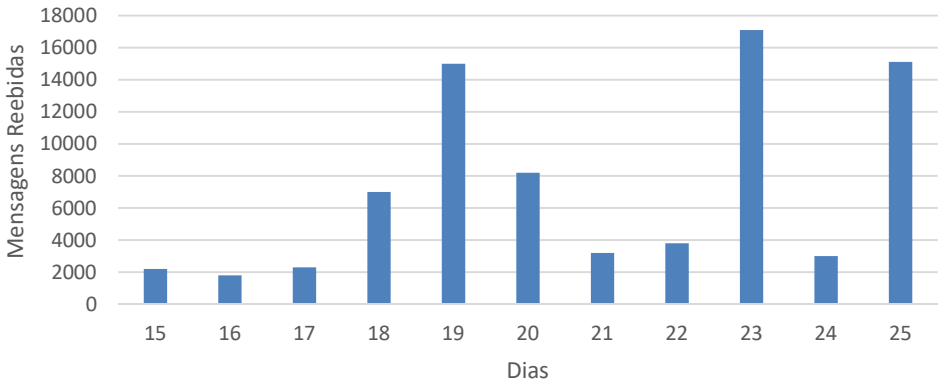



Fonte: o autor

O último cenário (3B) analisa de forma parcial os limites do sistema. Nele, 21 placas foram configuradas para transmitir pacotes a cada 10 segundos durante aproximadamente duas horas. Durante este período, chegaram, aproximadamente, um total de 11.254 mensagens. O total teórico (livre de perdas) seria 15.120. A Figura 27 ilustra o gráfico do número total de mensagens recebidos pelo *gateway* por dia. A última barra à direita corresponde ao momento em que cenário 3B foi realizado (25 de Janeiro). Inicialmente, o *gateway* estava na contagem de 3856 e após as duas horas se encontrava com o resultado de 15.110. Quando uma mensagem chega ao sistema de gerência, seu cabeçalho indica qual *gateway* a registrou e a gerência contabiliza a quantidade de mensagens que chegam por dia nos *gateways*. Neste caso, percebe-se que a quantidade de mensagens não foi suficiente para causar limitações no sistema de gerência. O gráfico da Figura 27 corresponde ao *gateway* G2. Todos os dispositivos estavam próximos deste *gateway* e utilizaram o SF10 para realizar o teste. Dentro da gerência há o gráfico de

mensagens diárias por *gateway*, porém, para uma melhor visualização, o gráfico aqui mostrado foi feito no *software* Excel, com base nos dados da gerência.

Figura 27 - Gráfico de mensagens diárias recebidas pelo *gateway* G2, no período 15 a 25 de janeiro




Fonte: o autor

## 5.2 Discussões

Os testes preliminares reportados aqui e executados exclusivamente com tecnologia LoRa apontam para uma solução de gerência capaz de operar os dispositivos em diferentes circunstâncias.

Os testes DUT são muito importantes para que o operador analise a qualidade da rede e/ou o bom funcionamento do dispositivo (pré-comissionamento). A partir dos testes, foi possível notar que o dispositivo 2 perdeu uma grande quantidade de pacotes. Ao se analisar a PDR do mesmo dispositivo, verifica-se que possui a maior latência e a maior taxa de perda de dados, se comparado aos outros dispositivos. Por fim, o seu nível de sinal mostra que ele possui o pior SNR e RSSI em relação a todos os outros dispositivos que participaram do DUT. Tais observações só puderam ser constatadas graças ao sistema proposto. Ao mesmo tempo, o sistema desenvolvido possui várias formas de o operador isolar e tentar encontrar eventuais problemas nos dispositivos. Uma forma de tentar aliviar o problema seria, por exemplo, mudar o SF de operação do dispositivo, ou mesmo ligar o ADR.

Embora o dispositivo 2 tenha perdido muitos pacotes no teste de DUT, o dispositivo estava responsivo nos outros testes e sempre realizou JOIN REQUEST com sucesso na rede LoRaWAN. Isto é possível, porque a tentativa de JOIN REQUEST é realizada várias vezes, até que se consiga alcançar o JOIN ACCEPT. Da mesma forma, uma requisição CoAP possui requisições configuradas com um *timeout* de 135 segundos, ou seja, uma resposta não recebida diante de uma requisição do servidor será novamente enviada pela gerência até que se obtenha a resposta (em até 135 segundos). Portanto é provável que o dispositivo consiga em algum momento responder e enviar requisições com sucesso. Mensagens DUT, assim como mensagens de alarme, são do tipo CoAP NON, por isso não apresentam mensagem de confirmação. Isto explica a grande quantidade de perda de pacotes. O resultado ruim de 2 comparado ao dos outros dispositivos tem relação direta pelo o seu posicionamento em relação aos *gateways*. Ele se encontra na posição mais baixa em relação aos *gateways*, se comparados aos outros dispositivos, além de não possuir visada. Por exemplo, o dispositivo 6 possui visada para o *gateway* e teve um dos melhores resultado em DUT. Entretanto, para determinadas aplicações de telemetria, uma taxa de 58% de chegada de pacotes pode ser aceitável., ou seja, o dispositivo 2 de fato está ruim se comparado aos outros dispositivos, mas o seu desempenho pode ser aceitável a depender da aplicação.

O cenário de teste 1F demostra a resiliência do LoRa mesmo em ambientes extremos, como sol, chuva e equipamentos que possam emitir algum tipo de interferência. O teste 3B também mostra que os *gateways* são capazes de receber muitas mensagens em um curto período.

Os testes 1A e 1B ilustram algo conhecido em redes LoRaWAN [55]: a dificuldade que o protocolo possui em lidar com o aumento no número de dispositivos, especialmente quando disputam canais simultaneamente. Como pode ser observado nos testes 1A e 1B, à medida que se aumenta o número de dispositivos, apresentam mais dificuldade de entrar na gerência. O problema, como já ilustrado, não é o processo de realizar LwM2M

REGISTER, mas, sim, no procedimento de JOIN REQUEST e JOIN ACCEPT, quando há muitos dispositivos tentando fazer JOIN REQUEST, que, normalmente, demoram a receber a mensagem de JOIN ACCEPT. Dentro da gerência, a quantidade de vezes que o dispositivo tentou realizar JOIN REQUEST é retornado pelo objeto 26246, recurso de número 2, onde está informação é extraída por meio de LwM2M READ (CoAP GET). Mesmo quando já estão na rede, múltiplas mensagens simultâneas levam a perdas de dados. O procedimento 2C ilustra esta questão que concerne a operação de PDR, o que leva um tempo maior para ser completada quando existem mais dispositivo fazendo este processo simultaneamente. Uma alternativa para tentar amenizar este problema seria ligar o ADR, embora isto resolveria o problema apenas para o caso do dispositivo já estiver dentro da rede LoRaWAN (ter recebido a mensagem de JOIN ACCEPT), porque o processo de JOIN REQUEST é feito geralmente em SF10. Uma outra opção seria mudar estaticamente os SFs de transmissão de dispositivos próximos a um *gateway* (para o caso do dispositivo já estiver dentro da rede LoRaWAN) ou até mesmo inserir algum processo aleatório de tempo em que os dispositivos enviam mensagens a gerência e fazem JOIN REQUEST na rede em tempos variáveis durante a operação dos dispositivos. Contudo, como o foco deste projeto não é estudar o desempenho da tecnologia LoRa, esta discussão está além do escopo deste trabalho. O importante é observar que estas análises, típicas para gerência e diagnóstico de operação, são facilitadas pelo sistema proposto.

No que diz respeito à pertinência do sistema desenvolvido como suporte para gerência em IoT, observa-se que a solução foi capaz de operar na maior parte das condições os dispositivos, mesmo quando estes estavam em enlaces degradados (considerável perca de pacotes). Naturalmente, existem aplicações que são tolerantes à percas e a altas latências, outras não. Mas o fato de se prover ao gerente da rede uma ferramenta capaz de parametrizar, tanto a camada física de comunicação quanto inserir políticas de uso do enlace e gerar históricos de informações de desempenho, são de grande

importância, pois ela vai muito além da questão da manutenção da rede, uma vez que também possibilita determinar se aquela condição de uso é suficiente para uma dada aplicação. Ao mesmo tempo, também oferecem informações básicas sobre cobertura e disponibilidade. Contudo o maior ganho que se promove não são só estes recursos (que eventualmente podem ser encontrados em sistemas comerciais), mas também na criação de uma estrutura lógica e física para se inserir novas ferramentas de operação de rede personalizadas, de modo a atender demandas específicas e, ainda assim, manter um padrão. O uso de padronização LwM2M, de estruturação de recursos como objetos IPSO, de formatação de dados em formato JSON e de um sistema modular de gerência possibilitam a abertura para esta personalização.

## 5.3 Resumo e discussão geral do capítulo

Neste capítulo foram apresentados os resultados obtidos nos testes realizados em cenários diversos. Foi mostrado o comportamento dos dispositivos frente ao sistema de gerência e suas interações com ele em diferentes configurações e cenários, ainda que a quantidade de dispositivos fosse limitada. Operações RESET remotos foram feitas de forma natural. O funcionamento do *firmware* nos dispositivos LwM2M também foi demonstrado. Valores de KPI e resultado de PDR e DUT também foram analisados e recuperados dos dispositivos. Por fim, um teste de um volume considerável de mensagens foi realizado, para analisar a solução proposta em condições com tráfegos mais intensos. A solução de gerência também ajudou a demonstrar, através de seu curso de DUT, que a LoRa tem bom potencial de aplicação para operação próximos a dispositivos de potência (neste caso, um transformador elétrico).

# 6 Conclusão e trabalhos futuros

Este trabalho propôs e implementou uma arquitetura genérica de gerenciamento de dispositivos limitados. Contudo, por questões de limitações desta pesquisa, foram conduzidos testes só com tecnologia LoRa. Apesar de suas limitações, os resultados preliminares apontam que:

- O sistema proposto foi capaz de implementar os principais recursos de gerência de dispositivos, incluindo recursos específicos, como o DUT para *smart grids* (e afins) em diferentes cenários práticos, atingindo, com frequência, taxas de sucesso próximas de 100%.
- As limitações de enlace LoRa não impediram o uso do protocolo LwM2M/CoAP para gerência. Além do mais, esta padronização foi capaz de atender todos os cenários de demandas funcionais requeridos.
- Além de poder parametrizar a camada física, as funcionalidades propostas foram capazes de fazer levantamento básico de QoS e de KPIs de cada dispositivo. Isto contribui para a gerência não só do dispositivo, mas para a rede como um todo na medida em que possibilita diagnósticos mais precisos e intervenções mais pontuais de operação. Todo este conjunto de dados foi avaliado como importante para as tarefas de gerência da rede aqui montada.
- Do ponto de vista da análise dos resultados experimentais conduzidos com o sistema criado em associação com a tecnologia LoRa, notou-se que para aplicações não críticas e tolerantes à latência (na ordem de dezenas de segundos), o modelo proposto pode também ser usado para aplicações desta área, uma vez que o

LwM2M dá igual suporte ao tráfego de dados para aplicações, sem fazer distinção entre aplicação e gerência[9].

Considerando que as tecnologias de IoT aqui tratadas são relativamente recentes e que o trabalho desenvolvido nessa dissertação é apenas um esforço inicial para desenvolvimento de uma ferramenta ainda mais robusta, é preciso citar melhoramentos que possam contribuir para este caminho:

- Testar o sistema proposto com outras tecnologias usuais (Wi-Fi, LTE, ZigBee etc.), criando, assim, novos objetos e recursos para estas novas tecnologias.
- Implantação de FUOTA.
- Embarcar no *firmware* a capacidade de fragmentação e recomposição de pacotes para permitir aplicações que possuem pacotes maiores.
- No caso específico de gerência de dispositivos LoRa, desenvolver estratégia de alocação de canal alternativa a usada pelo ADR no LNS. Isto possibilitará não só o uso mais eficiente do canal, mas também a remoção da 'dependência' do LNS.
- Desenvolvimento de mensagens de *multicast* e *broadcast* nativas usando o LoRaWAN.
- Implementar agendamento de tarefas de gerência.
- Configuração de protocolos de segurança, como o OSCORE (*Object Security for Constrained RESTful Environments*) [56].
- Análise do impacto do consumo de energia, de acordo com o tamanho dos pacotes.

[9] A gerência em si pode ser vista como uma aplicação que usa transferência de dados para se fazer a gestão do dispositivo.

# Anexo A: ferramentas e plataformas de IoT

A Tabela 14 apresenta uma análise comparativa entre os principais *middlewares* (de código aberto) e seus recursos.

Tabela 14 - Plataformas IoT de Código aberto

| | Kapua | Kaa | TTN | Fiware | ThingsBoard | OpenRemote | IoTivity |
|---|---|---|---|---|---|---|---|
| Ambiente | Java | Java | Go/Python/ JS | - | Java | Java | C/C++ |
| Plataforma de gerenciamento IoT | Sim | Sim | Sim | Sim | Sim | Sim | Sim |
| Monitoramento | Sim | Sim | Sim | Não | Sim | Sim | Não |
| Registro de identidade | Sim | Sim | Sim | Sim | Sim | Sim | Não |
| Micro services em tempo de execução | Não | Não | Não | Não | Não | Não | Não |
| Atualização Remota | Sim | Sim | Não | Não | Não | Sim (para controladores) | Não |
| Motores de Regra | Não | Não | Não | Não | Sim | Sim | Não |
| Alertas | Não | Sim | Sim | Sim | Sim | Sim | Não |
| Armazenamento de dados | Sim | Sim | Sim | Sim | Sim | Sim | Não |
| Anotação Semântica | Não | Sim | Não | Não | Sim (via atributos) | Não | Não |
| *Machine Learning* | Não | Não | Não | Não | Não | Não | Não |
| Análise em tempo real | Sim | Sim | Não | Sim | Sim | Não | Não |
| Registro e gerenciamento de dispositivos | Sim | Sim | Sim | Sim | Sim | Sim | Sim |
| Codificação/ Decodificação | Não | Por SDK | Sim | Sim | Via extensões | Sim | Sim |
| Gerenciamento de rede e comunicação | Sim | Sim | Não | Sim | Não | Não | Sim |
| Interfaces de alto nível suportadas | REST | REST | MQTT, REST | Sim | REST | REST API | REST API |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Interfaces de baixo nível suportadas | MQTT, CoAP | SDK | LoRaWAN | MQTT, CoAP | MQTT, CoAP, HTTP | Proprietário | CoAP, BLE, etc |
| Segurança | Sim | Sim | Sim | Sim | Sim | Sim | Sim |
| Plataforma de gerenciamento IoT | Sim | Sim | Sim | Sim | Sim | Sim | Sim |
| Monitoramento | Sim | Sim | Sim | Sim | Sim | Sim | Sim |
| Registro de identidade | Não | Não | Sim | Não | Não | Não | Sim |
| Micro services em tempo de execução | Não | Não | Não | Não | Não | Não | Não |
| Atualização Remota | Não | Não | Não | Não | Não | Não | Sim |
| Motores de Regra | Não | Não | Não | Não | Não | Não | Não |
| Alertas | Utilizando Plugins | Sim | Não | Sim | Não | Não | Sim |
| Armazenamento de dados | Sim | Sim | Sim | Sim | Não | Sim | Sim |
| Anotação Semântica | Via extensões | Não | Não | Não | Não | Não | Não |
| Machine Learning | Via extensões | Não | Não | Não | Não | Não | Não |
| Análise em tempo real | Sim | Não (livre) | Sim | Não | Sim (Stream) | Sim | Sim |
| Registro e gerenciamento de dispositivos | Sim | Sim | Sim | Não | Sim | Não | Sim |
| Codificação/ Decodificação | Via extensões | Não | Não | Não | Via extensões | Não | Não |
| Gerenciamento de rede e comunicação | Sim | Sim | Sim | Não | Via Scouts | Sim | Não |
| Interfaces de alto nível suportadas | REST API, MQTT | REST API | REST API | REST API, MQTT | REST API | REST API | REST API |
| Interfaces de baixo nível suportadas | WebSocket ou MQTT | MQTT,AMQP , Stop, WS | MQTT, CoAP | REST API | REST API | REST API | MQTT, WSo2 |
| Segurança | Sim | Sim | Sim | Sim | Sim | Sim | Sim |

Fonte: [57]

# Referências


[1] Cisco, "Internet of Things," 2016. [Online]. Disponível em: https://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/big-data/solution-overview-c22-740248.html. [Acesso em 19 Janeiro 2020].

[2] K. Delaney. e E. Levy, "Internet of Things: Challenges, Breakthroughs and Best Practices," CISCO, 2017. [Online]. Disponível em: https://connectedfutures.cisco.com/report/internet-of-things-challenges-breakthroughs-and-best-practices/. [Acesso em 19 Janeiro 2020].

[3] BNDES, "2B - Relatório Roadmap Tecnológico completo," 2017.

[4] A. Zaslavsky, C. Perera e D. Georgakopoulos, "Sensing as a Service and Big Data," em *International Conference on Advances in Cloud Computing (ACC)*, Bangalore, 2012.

[5] O. Salman, I. Elhajj, A. Chehab e A. Kayssi, "IoT survey: An SDN and fog computing perspective," *Computer Networks,* vol. 143, pp. 221-246, 9 Outubro 2018. https://doi.org/10.1016/j.comnet.2018.07.020.

[6] M. B. M. Noor e W. H. Hassan, "Current research on Internet of Things (IoT) security: A survey," *Computer Networks,* pp. 283-294, 15 Janeiro 2019. https://doi.org/10.1016/j.comnet.2018.11.025.

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari e M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communication Surveys & Tutorials,* vol. 17, pp. 2347-2376, 2015. https://doi.org/10.1109/COMST.2015.2444095.

[8] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang e W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things Journal,* vol. 5, pp. 1125-1142, Outubro 2017. https://doi.org/10.1109/JIOT.2017.2683200.

[9] "The Constrained Application Protocol (CoAP)," IETF, Junho 2014. [Online]. Disponível em: https://tools.ietf.org/html/rfc7252. [Acesso em 19 Janeiro 2020].

[10] OMASpecWorks, "Lightweight Machine to Machine Technical Specification:Core," OMA, 18 Junho 2018. [Online]. Disponível em: http://openmobilealliance.org/RELEASE/LightweightM2M/V1_1-20180612-C/OMA-TS-LightweightM2M_Core-V1_1-20180612-C.pdf. [Acesso em 19 Janeiro 2020].

[11] OMASpecWorks, "OMA LightweightM2M (LwM2M) Object and Resource Registry," OMA, [Online]. Disponível em:

http://openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html. [Acesso em 19 Janeiro 2020].

[12] S. Rao, D. Chendanda, C. Deshpande e L. V, "Implementing LWM2M in Constrained IoT Devices," em *2015 IEEE Conference on Wireless Sensors (ICWiSe)*, Melaka, 2015. https://doi.org/10.1109/ICWISE.2015.7380353.

[13] "MQTT," [Online]. Disponível em: http://mqtt.org/. [Acesso em 19 Janeiro 2020].

[14] "MQTT for Sensor Networks – MQTT-SN," 2 Dezembro 2013. [Online]. Disponível em: https://mqtt.org/tag/mqtt-sn. [Acesso em 19 Janeiro 2020].

[15] D. Thangavel, X. Ma, A. Valera, H. Tan e C. K. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," em *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Singapore, 2014. https://doi.org/10.1109/ISSNIP.2014.6827678.

[16] A. Triantafyllou, P. Sarigiannidis e T. D. Lagkas, "Network Protocols, Schemes, and Mechanisms for Internet of Things (IoT): Features, Open Challenges, and Trends," *Wireless Communications and Mobile Computing,* vol. 2018, pp. 1-24, 13 Setembro 2018. https://doi.org/10.1155/2018/5349894.

[17] "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header," IETF, Abril 2017. [Online]. Disponível em: https://tools.ietf.org/html/rfc8138. [Acesso em 19 Janeiro 2020].

[18] "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," IETF, Março 2012. [Online]. Disponível em: https://tools.ietf.org/html/rfc6550. [Acesso em 19 Janeiro 2020].

[19] Contiki-OS, "The Contiki Operating System," [Online]. Disponível em: https://github.com/contiki-os/contiki. [Acesso em 19 Janeiro 2020].

[20] u-blox, "SARA-R4 (x2B) seriesMulti-band LTE-M / NB-IoT and EGPRS modules," 2019. [Online]. Disponível em: https://www.u-blox.com/sites/default/files/SARA-R4-x2B_ProductSummary_%28UBX-16019228%29.pdf.

[21] muRata, "Sug-G Module Data Sheet," 2018. [Online]. Disponível em: https://wireless.murata.com/pub/RFM/data/type_abz.pdf.

[22] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melià-Seguí e T. Watteyne, "Understanding the Limits of LoRaWAN," *IEEE Communications Magazine,* vol. 55, nº 9, pp. 34-40, Setembro 2017. https://doi.org/10.1109/MCOM.2017.1600613.

[23] ChirpStack, "ChirpStack, open-source LoRaWAN® Network Server stack," [Online]. Disponível em: https://www.chirpstack.io/. [Acesso em 19 Janeiro 2020].

[24] LoRa Alliance, "LoRaWAN™1.0.3 Specification," Junho 2018. [Online]. Disponível em: https://lora-alliance.org/sites/default/files/2018-07/lorawan1.0.3.pdf. [Acesso em 19 Janeiro 2020].

[25] LoRa Alliance, "LoRaWAN™1.1 Regional Parameters," Janeiro 2018. [Online]. Disponível em: https://lora-alliance.org/sites/default/files/2018-04/lorawantm_regional_parameters_v1.1rb_-_final.pdf. [Acesso em 19 Janeiro 2020].

[26] IETF, "Static Context Header Compression (SCHC) and fragmentation for LPWAN, application to UDP/IPv6," [Online]. Disponível em: https://datatracker.ietf.org/doc/draft-ietf-lpwan-ipv6-static-context-hc/. [Acesso em 19 Janeiro 2020].

[27] H. C. Chen e F. J. Lin, "Efficient Device Group Management in oneM2M," em *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Singapore, 2018. https://doi.org/10.1109/WF-IoT.2018.8355131.

[28] oneM2M, "Standards for M2M and the Internet of Things," oneM2M, [Online]. Disponível em: http://www.onem2m.org/. [Acesso em 13 Março 2020].

[29] A. Karaagac, N. Verbeeck e J. Hoebeke, "The Integration of LwM2M and OPC UA: An Interoperability Approach for Industrial IoT," em *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, Limerick, 2019. https://doi.org/10.1109/WF-IoT.2019.8767209.

[30] " The Industrial Interoperability Standard™," OPC Foundation, [Online]. Disponível em: https://opcfoundation.org/. [Acesso em 19 Janeiro 2020].

[31] M. Li, E. Moll e C. M. Chituc, "IoT for Healthcare: An architecture and prototype implementation for the remote e-health device management using Continua and LwM2M protocols," em *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, Washington, 2018. http://doi.org/10.1109/IECON.2018.8591635.

[32] "Personal Connected Health Alliance," [Online]. Disponível em: https://www.pchalliance.org/. [Acesso em 19 Janeiro 2020].

[33] A. Karaagac, E. Dalipi, P. Crombez, P. E. D. e J. Hoebeke, "Light-weight streaming protocol for the Internet of Multimedia Things: Voice streaming over NB-IoT," *Pervasive and Mobile Computing,* vol. 59, pp. 1-16, Outubro 2019. https://doi.org/10.1016/j.pmcj.2019.101044.

[34] J. Hoebeke, J. Haxhibeqiri, B. Moons, M. V. Eeghen, J. Rossey, A. Karagaac e J. Famaey, "A Cloud-based Virtual Network Operator for Managing Multimodal LPWA Networks and Devices," em *2018 3rd Cloudification of the Internet of Things (CIoT)*, Paris, 2018. https://doi.org/10.1109/CIOT.2018.8627134.

[35] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal e Q. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling Technologies," *IEEE Internet of Things Journal,* vol. 4, pp. 1-20, Fevereiro 2017. http://doi.org/10.1109/JIOT.2016.2615180.

[36] M. Eisenhauser, P. Rosengren e Antolin, "A development platform for integrating wireless devices and sensors into ambient intelligence," em *Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, Roma, 2009. http://doi.org/10.1109/SAHCNW.2009.5172913.

[37] K. Aberer, M. Hauswirth e A. Salehi, “The Global Sensor Networks middleware for efficient and flexible deployment and interconnection of sensor networks,” 2006.

[38] TerraSwarm, “The TerraSwarm Research Center,” 2013. [Online]. Disponível em: https://ptolemy.berkeley.edu/projects/terraswarm/. [Acesso em 19 Janeiro 2020].

[39] Node-Red, “Low-code programming for event-driven applications,” [Online]. Disponível em: https://nodered.org/. [Acesso em 19 Janeiro 2020].

[40] FIWARE, “FIWARE: The open source platform for our smart digital future,” [Online]. Disponível em: https://www.fiware.org/. [Acesso em 19 Janeiro 2020].

[41] M. A. Razzaque, M. Milojevic-Jevric, A. Palada e S. Clarke, “Middleware for Internet of Things: A survey,” *IEEE Internet of Things Journal,* vol. 3, pp. 70-95, Fevereiro 2016. https://doi.org/10.1109/JIOT.2015.2498900.

[42] S. Bandyopadhyay, M. Sengupta, S. Maiti e S. Dutta, “Role of Middleware for Internet of Things: A Study,” *International Journal of Computer Science & Engineering Survey,* vol. 2, pp. 94-105, Agosto 2011.

[43] J. Dhas e P. Jeyanthi, “A Review on Internet of Things Protocol and Service Oriented Middleware,” em *International Conference on Communication and Signal Processing (ICCSP)*, Chennai, 2019. https://doi.org/10.1109/ICCSP.2019.8698088.

[44] cantcoap, “cantcoap,” [Online]. Disponível em: https://github.com/staropram/cantcoap. [Acesso em 19 Janeiro 2020].

[45] JSMN, “JSMN,” [Online]. Disponível em: https://github.com/staropram/cantcoap. [Acesso em 19 Janeiro 2020].

[46] ST Microelectronics, “I-CUBE-LRWAN,” [Online]. Disponível em: https://www.st.com/en/embedded-software/i-cube-lrwan.html. [Acesso em 19 Janeiro 2020].

[47] Semtech, “Reference implementation and documentation of a LoRa network node,” 17 Julho 2019. [Online]. Disponível em: https://github.com/Lora-net/LoRaMac-node. [Acesso em 10 Março 2020].

[48] “ Install the TTN Packet Forwarder on a Kerlink IoT Station,” [Online]. Disponível em: https://github.com/virtualguy/packet_forwarder-1/blob/master/docs/INSTALL_INSTRUCTIONS/KERLINK.md#install. [Acesso em 19 Janeiro 2020].

[49] lwm2m-node-lib. [Online]. Disponível em: https://github.com/telefonicaid/lwm2m-node-lib. [Acesso em 19 Janeiro 2020].

[50] LESHAN, “Eclipse Leshan is an OMA Lightweight M2M (LWM2M) implementation in Java,” [Online]. Disponível em: https://github.com/eclipse/leshan. [Acesso em 19 Janeiro 2020].

[51] libCoAP, “C-Implementation of CoAP,” [Online]. Disponível em: https://libcoap.net/. [Acesso em 19 Janeiro 2020].

[52] Wakaama, “ OMA Lightweight M2M C implementation designed to be portable on POSIX compliant systems,” [Online]. Disponível em: https://www.eclipse.org/wakaama/. [Acesso em 19 Janeiro 2020].

[53] LoRaMac-node, “LoRaWAN endpoint stack implementation and example projects,” Semtech, [Online]. Disponível em: https://github.com/Lora-net/LoRaMac-node. [Acesso em 19 Janeiro 2020].

[54] ST Microelectronics, “B-L072Z-LRWAN1,” ST Microelectronics, [Online]. Disponível em: https://www.st.com/en/evaluation-tools/b-l072z-lrwan1.html. [Acesso em 19 Janeiro 2020].

[55] F. V. D. Abeele, J. Haxhibeqiri, I. Moerman e J. Hoebeke, “Scalability analysis of large-scale LoRaWAN networks in ns-3,” *IEEE Internet of Things Journal,* vol. 4, nº 6, pp. 2186 - 2198, 2017. http://doi.org/10.1109/JIOT.2017.2768498.

[56] IETF, “Object Security for Constrained RESTful Environments (OSCORE),” IETF, Julho 2019. [Online]. Disponível em: https://tools.ietf.org/html/rfc8613. [Acesso em 15 Março 2020].

[57] C. Tranoris e S. Denazis, “Open Source Software solutions implementing a reference IoT architecture from the Things and Edge to the Cloud,” Patras, 2018.