

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ  
PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO  
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Glauber da Silva Bernardo

**Proposta de arquitetura para o desenvolvimento de soluções  
embarcadas de internet das coisas aplicadas em redes elétricas  
inteligentes.**

Fortaleza – CE

9 de dezembro de 2019



Glauber da Silva Bernardo

**Proposta de arquitetura para o desenvolvimento de  
soluções embarcadas de internet das coisas aplicadas em  
redes elétricas inteligentes.**

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Linha de Pesquisa: <Engenharia de Software>

Orientador: Prof. Dr. Elias Teodoro da Silva Júnior

Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Pró-Reitoria de Pesquisa, Pós-Graduação e Inovação (PRPI)

Programa de Pós-Graduação em Ciência da Computação (PPGCC)

Fortaleza – CE

9 de dezembro de 2019



Dados Internacionais de Catalogação na Publicação  
Instituto Federal do Ceará - IFCE  
Sistema de Bibliotecas - SIBI  
Ficha catalográfica elaborada pelo SIBI/IFCE, com os dados fornecidos pelo(a) autor(a)

---

- B518p    Bernardo, Glauber da Silva.  
        Proposta de arquitetura para o desenvolvimento de soluções embarcadas de internet das coisas aplicadas em redes elétricas inteligentes. / Glauber da Silva Bernardo. - 2019.  
        88 f. : il. color.
- Dissertação (Mestrado) - Instituto Federal do Ceará, Mestrado em Ciência da Computação, Campus Fortaleza, 2019.  
        Orientação: Prof. Dr. Elias Teodoro da Silva Júnior.
1. Redes Elétricas Inteligentes. 2. Internet das Coisas. 3. Middleware. 4. Redes LPWAN. 5. Aplicações Embarcadas. I. Título.

CDD 004

---

GLAUBER DA SILVA BERNARDO

Proposta de arquitetura para o desenvolvimento de soluções embarcadas de internet das coisas aplicadas em redes elétricas inteligentes.

Dissertação submetida à Coordenação do Curso de Pós-graduação em Ciência da Computação do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação, área de concentração Ciência da Computação.

Aprovada em 28 / 11 / 2019.

BANCA EXAMINADORA



Prof. Dr. Elias Teodoro da Silva Júnior (Orientador)  
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - IFCE



Prof. Dr. Antonio Wendell de Oliveira Rodrigues  
Instituto Federal de Educação, Ciência e Tecnologia do Ceará - PPGCC/IFCE



Prof. Dr. André Luiz Carneiro de Araújo  
Instituto Federal de Educação, Ciência e Tecnologia do Ceará

# Agradecimentos

Aos meus pais, Bernardo e Lúcia, meus tios Airton e Helena, meus irmãos Glaydson e Glayciane e minha dupla Yoko que direta ou indiretamente contribuíram e contribuem para a continuidade do meu caminhar.

Ao meu orientador, professor Elias Teodoro agradeço pelos imensuráveis conhecimentos adquiridos através das aulas, reuniões e orientações. Nestes anos de convivência pude aprender muito sobre como desenvolver uma mentalidade de pesquisador.

Ao Rômulo Garcia, pela confiança em mim depositada para que eu pudesse participar de um projeto de inovação paralelamente ao mestrado, disponibilizando uma bolsa de pesquisa durante todo o período.

Ao professor André Luiz pelo apoio ao disponibilizar sua equipe e equipamentos do NEPEN viabilizando parte dos experimentos.

Aos amigos, Saulo Oliveira que me auxiliou a entrar no mestrado e Anderson Dourado que contribuiu com parte da implementação do código do middleware.

Aos colegas de mestrado e companheiros do LAMP/LES, LIT e LDS pelo apoio. Em especial, ao Paulo Honório sempre prestativo a ajudar, dividir conhecimento e experiências de pesquisa.

A todos os professores do programa de mestrado (PPGCC) do IFCE pelas instruções e contribuições à execução deste trabalho.

O meu mais sincero **Muito Obrigado!**



# Resumo

As Redes Elétricas Inteligentes (*Smart Grids*) empregam vários dispositivos para monitorar, analisar e controlar a própria rede. Esses dispositivos são implantados em grande quantidade em usinas de energia, centros de distribuição e instalações de consumidores. Uma *Smart Grid* requer conectividade, automação e rastreamento para tais dispositivos. A Internet das Coisas (*IoT*) tem sido utilizada para prover esses serviços, contudo, há uma série de desafios a superar. Os principais deles estão relacionados à diversidade de plataformas de hardware e software utilizadas para implementar computadores embarcados que estão mais próximos da infraestrutura da rede elétrica, chamados dispositivos finais. Para reduzir o tempo de desenvolvimento dessas aplicações embarcadas, esta dissertação propõe um *middleware IoT*, denominado *Smart Grids Applications Middleware (SGAM)*, que visa facilitar a criação de soluções para controle, monitoramento e medição em Redes Elétricas Inteligentes. O *middleware SGAM* busca disponibilizar um conjunto de funcionalidades que permitirá ao desenvolvedor focar principalmente na regra de negócio da aplicação. Ele também facilitará a portabilidade das soluções criadas entre diversas plataformas de prototipação existentes. Outro fator considerado é a integração com as redes *LPWAN* provendo comunicação bidirecional às aplicações. Além disso, foi possível trafegar tamanhos de dados superiores às limitações impostas por estas redes por meio do uso de estratégias de fragmentação no envio dos dados. Dois casos de uso são projetados e o *middleware* é avaliado por meio de algumas métricas (memória ocupada e tempo de execução). A partir delas são discutidas as principais características da arquitetura implementada. Os resultados apresentados indicam que a proposta de arquitetura do *middleware SGAM* se mostra promissora para projetar de maneira simples e eficiente aplicações embarcadas para sistemas de controle, monitoramento e medição aplicados em redes elétricas inteligentes.

**Palavras-chave:** Redes Elétricas Inteligentes; Internet das Coisas; *Middleware*; Redes *LPWAN*; Aplicações Embarcadas.



# Abstract

A Smart Grid uses several devices to monitor, analyze and control the power grid resources. These devices are deployed in power plants, distribution centers and consumer facilities in a very large number. Therefore, a Smart Grid needs connectivity, automation, and tracking for such devices. The Internet of Things (IoT) has been used to provide such services, but there are still some challenges to be overcome. The main ones are related to the high diversity of hardware and software platforms used to implement those embedded computers that are closer to the power grid infrastructure. In order to contribute to the development of these embedded applications, this thesis proposes an IoT middleware that facilitates the creation of IoT applications for Smart Grids, called SGAM (Smart Grids Applications Middleware). SGAM purpose is to reduce the development time for embedded solutions for control, monitoring, and measurement in Smart Grids. SGAM middleware aims to provide a set of features that will allow the developer to focus primarily on the business rule of the application. It will also facilitate the portability of solutions built across multiple existing prototyping platforms. Another factor considered is integration with LPWAN networks to provide two-way communication to applications. In addition, it was possible to traffic messages larger than the data size limitations imposed by these networks through the use of fragmentation strategies in the sending of data. Two use cases are designed and middleware is evaluated using some metrics (memory footprint and execution time) that are used to discuss the main features of the implemented architecture. The results presented indicate that the SGAM middleware architecture proposal is promising to simply and efficiently design embedded applications for control, monitoring and measurement systems applied to smart grids.

**Keywords:** Smart Grids; Internet of Things; Middleware; LPWAN; Embedded Applications.



# Lista de ilustrações

|                                                                                                                                                |    |
|------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Figura 1 – Visão geral da utilização das Redes Elétricas Inteligentes. . . . .                                                                 | 27 |
| Figura 2 – Rede elétrica inteligente em 7 domínios. . . . .                                                                                    | 28 |
| Figura 3 – Comparativo entre <i>LPWAN</i> e outras tecnologias. . . . .                                                                        | 31 |
| Figura 4 – Exemplo de topologia de rede <i>LoRaWAN</i> . . . . .                                                                               | 34 |
| Figura 5 – Arquitetura básica do <i>Mbed</i> . . . . .                                                                                         | 38 |
| Figura 6 – Divisão do nó em quatro camadas. . . . .                                                                                            | 49 |
| Figura 7 – Arquitetura do <i>Smart Grid Application Middleware</i> . . . . .                                                                   | 50 |
| Figura 8 – Processo de definição e integração com a plataforma alvo. . . . .                                                                   | 51 |
| Figura 9 – Delimitação de escopo na implementação. . . . .                                                                                     | 52 |
| Figura 10 – Diagrama de classe - <i>Control</i> . . . . .                                                                                      | 53 |
| Figura 11 – Classes geradas para a plataforma <i>Mbed OS</i> . . . . .                                                                         | 55 |
| Figura 12 – Trecho de código dos operadores de inclusão e obtenção do <i>controller</i> . . . . .                                              | 56 |
| Figura 13 – Trecho do código da definição da classe <i>Gyroscope</i> . . . . .                                                                 | 56 |
| Figura 14 – Trecho do código da definição da classe <i>LoRaWanComm</i> . . . . .                                                               | 57 |
| Figura 15 – Trecho do código da <i>API</i> baseada em eventos. . . . .                                                                         | 57 |
| Figura 16 – Diagrama de Classe - <i>Management</i> . . . . .                                                                                   | 58 |
| Figura 17 – Declaração do <i>payload</i> e envio de 10 <i>bytes</i> . . . . .                                                                  | 59 |
| Figura 18 – <i>Single block</i> - Exibição do <i>Payload</i> em formato <i>json</i> . . . . .                                                  | 60 |
| Figura 19 – Declaração do <i>Payload</i> e envio de 100 <i>bytes</i> . . . . .                                                                 | 60 |
| Figura 20 – <i>Composite block</i> - Exibição do <i>Payload</i> em formato <i>json</i> . . . . .                                               | 61 |
| Figura 21 – Exemplo de sistema de monitoramento utilizando dispositivo embarcado. . . . .                                                      | 65 |
| Figura 22 – Diagrama de sequência de obtenção de referência para os objetos <i>control</i> ,<br><i>sensor</i> e <i>communication</i> . . . . . | 66 |
| Figura 23 – Trecho do código da aplicação embarcada - rotina de inicialização da<br>classe controle. . . . .                                   | 67 |
| Figura 24 – Trecho do código da aplicação embarcada - classe sensor. . . . .                                                                   | 68 |
| Figura 25 – Trecho do código da aplicação embarcada - rotina de conexão. . . . .                                                               | 69 |
| Figura 26 – Trecho do código da aplicação embarcada - rotina de escrita. . . . .                                                               | 70 |
| Figura 27 – Trecho do código da aplicação embarcada - rotina de leitura. . . . .                                                               | 70 |
| Figura 28 – Tela de dados da aplicação no <i>TTN</i> (Servidor). . . . .                                                                       | 71 |
| Figura 29 – Exemplo de fragmentação de dados utilizando dispositivo embarcado. . . . .                                                         | 72 |
| Figura 30 – Diagrama de sequência da rotina de fragmentação. . . . .                                                                           | 74 |
| Figura 31 – Trecho de código da aplicação embarcada - fragmentador de dados. . . . .                                                           | 75 |
| Figura 32 – Tempo de resposta do método (tempo x fragmentos). . . . .                                                                          | 78 |
| Figura 33 – Tempo de resposta do método (tempo x dados). . . . .                                                                               | 79 |



# Lista de tabelas

|                                                                                                                             |    |
|-----------------------------------------------------------------------------------------------------------------------------|----|
| Tabela 1 – Comparação dos trabalhos relacionados quanto aos requisitos . . . . .                                            | 46 |
| Tabela 2 – <i>Single Block</i> . . . . .                                                                                    | 59 |
| Tabela 3 – <i>Composite Block</i> . . . . .                                                                                 | 60 |
| Tabela 4 – Recursos utilizados . . . . .                                                                                    | 64 |
| Tabela 5 – Ocupação de memória na geração da aplicação de monitoramento de<br>ativos . . . . .                              | 66 |
| Tabela 6 – Tempo consumido (em microssegundos) do módulo <i>control</i> . . . . .                                           | 67 |
| Tabela 7 – Tempo consumido (em microssegundos) dos sensores. . . . .                                                        | 68 |
| Tabela 8 – Tempo consumido (em microssegundos) na comunicação. . . . .                                                      | 72 |
| Tabela 9 – Ocupação de memória na geração da aplicação de fragmentação de dados                                             | 73 |
| Tabela 10 – Tempo consumido (em microssegundos) na fragmentação. . . . .                                                    | 76 |
| Tabela 11 – Tempo consumido (em microssegundos) na execução do método <i>setMsg</i><br>em diferentes configurações. . . . . | 77 |



# Lista de abreviaturas e siglas

|         |                                                             |
|---------|-------------------------------------------------------------|
| ABraDEE | Associação Brasileira de Distribuidores de Energia Elétrica |
| ABP     | Activation By Personalization                               |
| ADR     | Adaptive Data Rate                                          |
| AES     | Advanced Encryption Standard                                |
| API     | Application Programming Interface                           |
| BLE     | Bluetooth Low Energy                                        |
| CoT     | Cloud of things                                             |
| COSEM   | Companion Specification for Energy Metering                 |
| CSS     | Chirp Spread Spectrum                                       |
| CPU     | Central Processing Unit                                     |
| DoE     | Department of Energy                                        |
| DLMS    | Device Language Message Specification                       |
| EC-GSM  | Extended Coverage – GSM                                     |
| eMTC    | enhanced Machine Type Communication                         |
| FIFO    | First-IN-First-Out                                          |
| FSK     | Frequency Shift Keying                                      |
| GCC     | GNU Compiler Collection                                     |
| GPIO    | General Purpose Input/Output                                |
| GPS     | Global Positioning System                                   |
| GSM     | Global System for Mobile communication                      |
| HAL     | Hardware Abstraction Layer                                  |
| IDE     | Integrated Development Environment                          |
| IOT     | Internet Of Things                                          |

|        |                                    |
|--------|------------------------------------|
| ISM    | Industrial, Scientific and Medical |
| I2C    | Inter-Integrated Circuit           |
| LoRa   | Long Range                         |
| LPWAN  | Low Power Wide Area Network        |
| LTE    | Long Term Evolution                |
| MAC    | Media Access Control               |
| MCU    | MicroController Unit               |
| M2M    | Machine-To-Machine                 |
| NB-IoT | NarrowBand-Internet of Things      |
| NFC    | Near Field Communication           |
| OIC    | Open Interconnect Consortium       |
| OO     | Object-Oriented                    |
| OS     | Operating System                   |
| OTAA   | Over-The-Air Activation            |
| PLC    | Power Line Communication           |
| RAM    | Random Access Memory               |
| RFID   | Radio Frequency IDentification     |
| RO     | Read-Only                          |
| RTOS   | Real Time Operating Systems        |
| RW     | Read-Write                         |
| SG     | Smart Grid                         |
| SGAM   | Smart Grid Application Middleware  |
| SM     | Smart Meters                       |
| SOE    | Sistema Operacional Embarcado      |
| SPI    | Serial Peripheral Interface        |
| SSL    | Secure Sockets Layer               |

|         |                                                     |
|---------|-----------------------------------------------------|
| TI      | Tecnologia da Informação                            |
| TIC     | Tecnologias de Informação e Comunicação             |
| TTN     | The Things Network                                  |
| UART    | Universal Asynchronous Receiver/Transmitter         |
| UML     | Unified Modeling Language                           |
| WBAN    | Wireless Body Area Networks                         |
| WIFI    | Wireless Fidelity                                   |
| WPAN    | Wireless Personal Area Network                      |
| VSAT    | Very Small Aperture Terminal                        |
| ZI      | Zero Initialise                                     |
| 3GPP    | 3rd Generation Partnership Project                  |
| 6LoWPAN | IPv6 over Low power Wireless Personal Area Networks |



# Sumário

|            |                                                                                                                        |           |
|------------|------------------------------------------------------------------------------------------------------------------------|-----------|
| <b>1</b>   | <b>INTRODUÇÃO</b>                                                                                                      | <b>21</b> |
| <b>1.1</b> | <b>Motivação</b>                                                                                                       | <b>23</b> |
| <b>1.2</b> | <b>Objetivos</b>                                                                                                       | <b>24</b> |
| 1.2.1      | Objetivo Geral                                                                                                         | 24        |
| 1.2.2      | Objetivos Específicos                                                                                                  | 24        |
| <b>1.3</b> | <b>Produção científica</b>                                                                                             | <b>24</b> |
| <b>1.4</b> | <b>Estrutura da Dissertação</b>                                                                                        | <b>24</b> |
| <b>2</b>   | <b>FUNDAMENTAÇÃO TEÓRICA</b>                                                                                           | <b>27</b> |
| <b>2.1</b> | <b>Rede Elétrica Inteligente</b>                                                                                       | <b>27</b> |
| 2.1.1      | Integração da Rede Elétricas Inteligente com a Internet das Coisas                                                     | 29        |
| <b>2.2</b> | <b>Redes LPWAN</b>                                                                                                     | <b>30</b> |
| 2.2.1      | Protocolo <i>LoRaWAN</i>                                                                                               | 33        |
| 2.2.2      | Topologia da rede                                                                                                      | 33        |
| 2.2.3      | Dispositivos finais                                                                                                    | 35        |
| <b>2.3</b> | <b>Plataformas Embarcados e <i>Middlewares</i></b>                                                                     | <b>36</b> |
| 2.3.1      | Plataforma ARM Mbed                                                                                                    | 37        |
| 2.3.2      | <i>Middlewares</i>                                                                                                     | 38        |
| <b>2.4</b> | <b>Conclusão</b>                                                                                                       | <b>39</b> |
| <b>3</b>   | <b>TRABALHOS RELACIONADOS</b>                                                                                          | <b>41</b> |
| <b>3.1</b> | <b>Arquiteturas de <i>middlewares</i> de Internet das Coisas</b>                                                       | <b>41</b> |
| <b>3.2</b> | <b>Utilização das Redes <i>LPWAN</i> em soluções <i>IoT</i> para Redes Elétricas Inteligentes</b>                      | <b>42</b> |
| <b>3.3</b> | <b>Explorando sistemas operacionais embarcados visando à diminuição do tempo de projeto e aumento de portabilidade</b> | <b>44</b> |
| <b>3.4</b> | <b>Conclusão</b>                                                                                                       | <b>45</b> |
| <b>4</b>   | <b>ARQUITETURA PROPOSTA</b>                                                                                            | <b>47</b> |
| <b>4.1</b> | <b>Requisitos Funcionais</b>                                                                                           | <b>47</b> |
| 4.1.1      | Características do Nó Final                                                                                            | 48        |
| 4.1.2      | Smart Grid Application Middleware (SGAM)                                                                               | 48        |
| 4.1.3      | Escolha e integração com a plataforma <i>ARM Mbed</i>                                                                  | 50        |
| <b>4.2</b> | <b>Implementação do <i>Middleware</i></b>                                                                              | <b>51</b> |
| 4.2.1      | Módulo de Controle                                                                                                     | 52        |
| 4.2.2      | Módulo de Integração                                                                                                   | 54        |

|       |                                                                       |           |
|-------|-----------------------------------------------------------------------|-----------|
| 4.2.3 | Módulo de Gerenciamento . . . . .                                     | 58        |
| 4.3   | <b>Conclusão . . . . .</b>                                            | <b>61</b> |
| 5     | <b>CASOS DE USO DO MIDDLEWARE . . . . .</b>                           | <b>63</b> |
| 5.1   | <b>Recursos utilizados . . . . .</b>                                  | <b>63</b> |
| 5.2   | <b>Monitoramento Remoto de Ativos . . . . .</b>                       | <b>64</b> |
| 5.2.1 | Ocupação de memória da aplicação de monitoramento de ativos . . . . . | 65        |
| 5.2.2 | Inicialização do módulo control . . . . .                             | 66        |
| 5.2.3 | Utilização dos sensores . . . . .                                     | 68        |
| 5.2.4 | Utilização da comunicação . . . . .                                   | 69        |
| 5.3   | <b>Fragmentador de Dados . . . . .</b>                                | <b>72</b> |
| 5.3.1 | Ocupação de memória da aplicação de fragmentação de dados . . . . .   | 73        |
| 5.3.2 | Rotina de fragmentação . . . . .                                      | 73        |
| 5.3.3 | Investigação do método <i>setMsg()</i> . . . . .                      | 76        |
| 5.4   | <b>Conclusão . . . . .</b>                                            | <b>79</b> |
| 6     | <b>CONCLUSÃO . . . . .</b>                                            | <b>81</b> |
|       | <b>REFERÊNCIAS . . . . .</b>                                          | <b>83</b> |

# 1 Introdução

O conceito de *Smart Grid (SG)* ou Redes Elétricas Inteligentes, cunhado por Amin e Wollenberg (2005), apresenta uma mudança no paradigma do setor elétrico. Leva em conta a necessidade de tornar o sistema de entrega de energia mais interativo, incorporando diferentes fontes de energia na rede, em especial, fontes geradoras descentralizadas, renováveis e intermitentes. Além disso, introduz novos consumidores, como veículos elétricos e melhora a eficiência e o próprio dimensionamento da rede.

Justamente por representar uma mudança de paradigma, trata-se de um conceito com múltiplas definições, que variam de acordo com a perspectiva adotada, tais como: tecnológica, ambiental, socioeconômica ou político-regulatória. Os autores Amin e Wollenberg (2005) que cunharam o termo *Smart Grid* o definem como uma infraestrutura de rede elétrica em larga escala caracterizada por segurança, agilidade e resiliência/robustez que enfrenta novas ameaças e condições não previstas. Na definição do DOE (U.S. Department of Energy) (2010), trata-se de um conceito abrangente, em que se faz uso de diversas tecnologias para que, através da automação e comunicação de toda a rede elétrica, possa propiciar uma infraestrutura mais integrada entre geração, transmissão e distribuição de energia.

Em síntese, e para fins deste trabalho, as Redes Elétricas Inteligentes objetivam otimizar a produção, distribuição e o consumo de energia. Viabilizam a entrada de novos fornecedores e consumidores na rede. Obtém melhorias significativas em monitoramento, gestão, automação e qualidade da energia ofertada, por meio de uma rede elétrica caracterizada pelo uso intensivo das Tecnologias de Informação e Comunicação (TIC).

Desde a criação da primeira rede de energia elétrica não tinham ocorrido inovações significativas na maneira como essa energia era fornecida ao consumidor. No entanto, com a introdução das Tecnologias de Informação e Comunicação nas redes elétricas pôde iniciar um processo retroalimentado de desenvolvimento. Conforme Ministério de Minas e Energia (2010) e Moscatelli (2016), os medidores inteligentes foram implantados em todo o mundo substituindo medidores mecânicos para fornecer leituras de energia mais precisas e remotas. Posteriormente, também foram incluídos serviços de flexibilidade tarifária, detecção de adulteração, monitoramento de qualidade de energia, gerenciamento de picos de energia, conexão/desconexão e muitos outros. Atualmente, as Redes Elétricas Inteligentes possuem vários tipos de dispositivos para monitorar, analisar e controlar a rede. Tais dispositivos de monitoramento são implantados em usinas, linhas de transmissão, torres, centros de distribuição e instalações dos consumidores, e seus números chegam a centenas de milhões ou mesmo bilhões (COLLIER, 2017).

Nos últimos anos, com o crescimento da Internet das Coisas e das tecnologias digitais, a Rede Elétrica Inteligente está viabilizando cada vez mais o conceito de “*smart*”. Conforme descrito em Meng, Ma e Chen (2014), a Internet das Coisas pode ajudar a Rede Elétrica Inteligente, suportando várias funções na geração, armazenamento, transmissão, distribuição e consumo de energia. Também auxilia incorporando dispositivos (como sensores, atuadores e medidores inteligentes), bem como fornecendo conectividade, automação e rastreamento para tais equipamentos. No entanto, há ainda uma série de desafios a serem superados para alavancar a ampla disseminação desse paradigma. Principalmente com relação ao desenvolvimento de aplicações e à alta diversidade de tecnologias de *hardware* e *software* desses ambientes (OBAIDAT et al., 2018), (AFZAL et al., 2017). Dentre esses desafios, destacam-se: (i) a diversidade de plataformas de desenvolvimento e linguagens de programação que dificultam o reuso de código da aplicação em outras soluções, (ii) as limitações de desempenho das linhas de comunicação impossibilitando por vezes, trafegar maiores quantidades de dados, por exemplo, ao tentar realizar leitura de dados de medidores inteligentes e (iii) a heterogeneidade dos diversos *hardwares*, *drivers* e protocolos que impedem uma aplicação atender a todos esses recursos utilizados para comunicar com os dispositivos da rede elétrica.

O desenvolvimento das soluções embarcadas de Internet das Coisas para as Redes Elétricas Inteligentes possuem requisitos em comuns, tais como: funcionalidades de controle, monitoramento e medição dos dispositivos; interface de comunicação com fio e sem fio; comunicação unidirecional ou bidirecional; protocolos de comunicação; suporte a diversos sensores, atuadores e módulos; portabilidade da aplicação entre diferentes plataformas de *hardware* (MOHAMED et al., 2017). A partir da identificação desses requisitos em comuns, esse trabalho propõe unificá-los em um componente de *software* para redução do tempo de projeto, um *middleware*, que seja genérico o suficiente para permitir sua utilização por variados tipos de aplicações feitas nessas redes, e que ao mesmo tempo seja leve para garantir um bom desempenho e boa integração.

Esta dissertação propõe o desenvolvimento de um *middleware IoT*, denominado *Smart Grids Applications Middleware (SGAM)*, para a criação de soluções embarcadas de controle, monitoramento e medição em Redes Elétricas Inteligentes. O *middleware SGAM* busca disponibilizar um conjunto de funcionalidades que permitirá ao desenvolvedor focar principalmente na regra de negócio da aplicação. Também facilitará a portabilidade das soluções criadas entre diversas plataformas de prototipação existentes. Utilizando os recursos disponibilizados pelo *middleware*, o desenvolvedor estará livre para implementar sua aplicação independentemente do seu nível de conhecimento da plataforma alvo, pois estará programando focado na *API* do *middleware*. Com isso, objetiva-se que pressões ligadas ao mercado de consumo também sejam atendidas, tais como: a redução do tempo desde a ideia até a disponibilidade do produto (*time-to-market*) e sua crescente complexidade (SANGIOVANNI-VINCENTELLI et al., 2004). Vale destacar que o *middleware* desse

trabalho executa nos dispositivos finais, ou seja, aqueles computadores embarcados que estão mais próximos dos equipamentos elétricos da rede.

A área das Redes Elétricas Inteligentes, e principalmente as soluções de Internet das Coisas utilizadas nessas redes, estimulam a produção científica pelos desafios tecnológicos que apresentam e pelo potencial de aplicabilidade. O fato de esta ser uma área emergente e multidisciplinar, e de apresentar várias visões com requisitos diferentes, faz com que não existam ainda muitos sistemas disponíveis que tenham respostas para tantas questões em aberto. Isto torna as soluções de Internet das Coisas para Redes Elétricas Inteligentes uma extensa área de pesquisa em: arquiteturas, *frameworks*, redes, protocolos, entre outros.

A Dissertação enquadra-se em uma área de pesquisa bem recente, denominada *Smart Grids*, que vem despertando muito interesse da comunidade científica e de empresas, por revolucionar substancialmente as redes elétricas. No Brasil, a perspectiva de investimentos em equipamentos e softwares de medição, automação, tecnologia da informação (TI), telecomunicações e dispositivos de geração distribuída, podem alcançar de R\$ 46 bilhões a R\$ 91 bilhões até 2030, segundo estudo da Associação Brasileira de Distribuidores de Energia Elétrica (Abradee) (RIVERA; ESPOSITO; TEIXEIRA, 2013).

## 1.1 Motivação

As motivações do trabalho são detalhadas a seguir, em três tópicos.

Em relação à engenharia de software, utilizando-se principalmente dos conceitos de componentes e reuso. Busca-se discutir a criação de uma arquitetura de *software* flexível o suficiente que torne o desenvolvimento de novas soluções um processo mais fácil e barato. Isto, em um cenário de alta heterogeneidade decorrente da diversidade de plataformas de *hardware* e *software* usadas nas Redes Elétricas Inteligentes.

Quanto à criação de um produto funcional, pretende-se disponibilizar um conjunto de funcionalidades que permitirão que desenvolvedores se preocupem principalmente com a regra de negócio de suas aplicações. Na literatura existem diversos protótipos criados e alguns estudos relacionados ao desenvolvimento de *middleware*. Entretanto, poucas das propostas de *middleware* apresentadas foram de fato desenvolvidas e avaliadas. Ao considerar relevante a implementação, pode-se de fato disponibilizar um componente de *software* que possa ser utilizado em campo.

E por fim, sobre o uso da Internet das Coisas nas Redes Elétricas Inteligentes. Hoje, o número de "coisas" que usam eletricidade é maior do que o número de dispositivos conectados à *internet*, e essencialmente tudo o que usa eletricidade pode ser feito mais útil conectando-o à *internet*. Por isso, no futuro, fazendo a tecnologia de Internet das Coisas como um padrão global para as comunicações e a base para a Rede Elétrica Inteligente,

novas portas serão abertas para maximizar as perspectivas para futuras inovações.

## 1.2 Objetivos

### 1.2.1 Objetivo Geral

- Propor o projeto e implementação de um *middleware* para a criação de soluções embarcadas de controle, monitoramento e medição em Redes Elétricas Inteligentes.

### 1.2.2 Objetivos Específicos

- Definir a arquitetura de um *middleware* modular e extensível para suportar a inclusão e alteração de novos sensores, atuadores e protocolos de comunicação;
- Avaliar o *middleware* proposto sobre aspectos críticos ligados a requisitos arquiteturais (recursos de memória e tempo consumidos na execução) e a interação com as redes *LPWANs* visando trafegar tamanhos de dados superiores às limitações impostas por estas redes;
- Disponibilizar um componente de *software* que seja genérico e otimizado o suficiente para garantir um bom desempenho e diminuição do tempo de desenvolvimento de aplicações feitas para Redes Elétricas Inteligentes.

## 1.3 Produção científica

Durante este projeto de mestrado, o seguinte trabalho científico foi aceito e publicado, a saber:

- Bernardo G. S, Cunha A, Silva Jr. E. T; "Arquitetura para o desenvolvimento de soluções embarcadas de internet das coisas aplicadas em redes elétricas inteligentes", IX Brazilian Symposium on Computing Systems Engineering (IX SBESC), Nov, 2019.

## 1.4 Estrutura da Dissertação

O restante deste texto está organizado em mais cinco capítulos, descritos resumidamente a seguir:

No Capítulo 2, como fundamentação teórica, são apresentados os principais conceitos utilizados nesta pesquisa. Inicialmente, a Rede Elétrica Inteligente é explicada e descrita sua integração com a Internet das Coisas. No segundo momento, as redes *LPWANs* são

abordadas, dando ênfase ao protocolo *LoRaWAN*. Na terceira parte, são explorados os conceitos de Plataformas Embarcadas e *middlewares*.

No Capítulo 3 é descrito o estado da arte da criação de arquiteturas de *middlewares IoT*, utilização das redes *LPWANs* em soluções de Internet das Coisas para Redes Elétricas Inteligentes, assim como pesquisas que exploram a utilização de sistemas operacionais embarcados visando diminuição do tempo de projeto e aumento da portabilidade das soluções criadas.

O Capítulo 4 apresenta a proposta de *Middleware SGAM (Smart Grid Application Middleware)* para criação de soluções embarcadas. Descreve seus requisitos, sua arquitetura e seus componentes de integração. Além disso, propõe uma implementação funcional da arquitetura.

O Capítulo 5 apresenta casos de uso que foram implementados para mostrar exemplos de utilização do *middleware SGAM*, assim como, os resultados dos experimentos realizados.

O Capítulo 6 finaliza o trabalho, onde se apresentam as considerações finais da pesquisa e perspectivas de continuidade.



## 2 Fundamentação Teórica

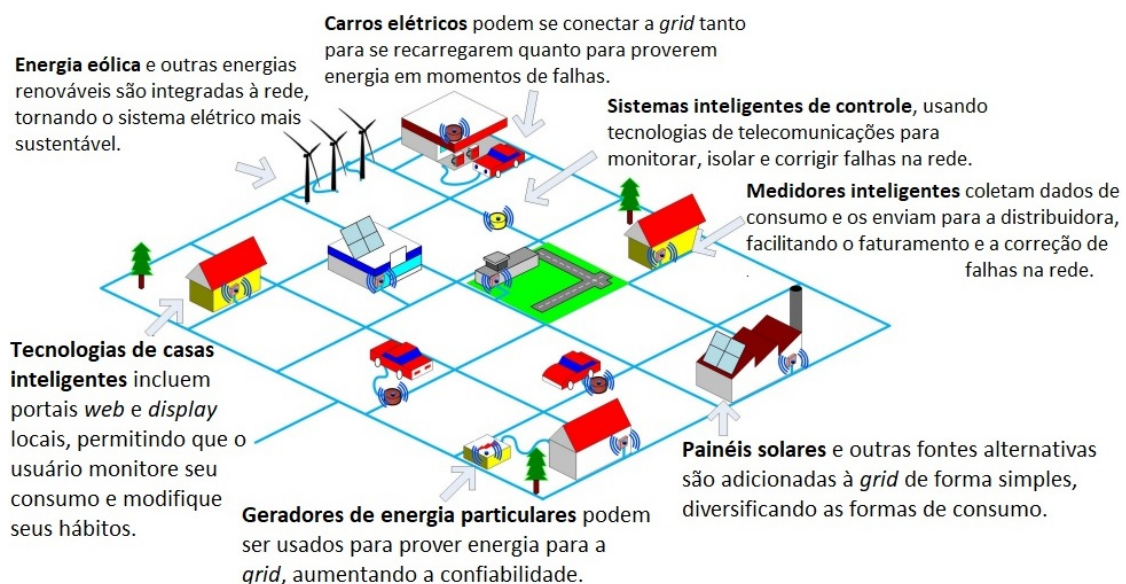
Neste capítulo são explorados os conceitos fundamentais utilizados nesse trabalho, abordando desta forma os temas principais: Rede Elétrica Inteligente, Redes *LPWAN*, Plataformas Embarcadas e *Middlewares*.

### 2.1 Rede Elétrica Inteligente

De acordo com Falcão (2009), a expressão Rede Elétrica Inteligente (*Smart Grid*) deve ser entendida mais como um conceito do que uma tecnologia ou equipamento específico. Ela carrega a ideia da utilização intensiva de tecnologia de informação e comunicação na rede elétrica. E através da possibilidade de comunicação do estado dos diversos componentes da rede, permitirá a implantação de estratégias de controle e otimização da rede de forma muito mais eficiente que as atualmente em uso.

Uma tentativa abrangente de definição do conceito, apresentada por Siddiqui, Hurtado e Parmenter (2008), é que a expressão Rede Elétrica Inteligente pode ser entendida como a sobreposição dos sistemas unificados para prover a informação correta para a entidade correta (equipamentos de uso final, sistemas de controle, consumidores, etc.). É um sistema que otimiza o suprimento de energia, minimizando perdas de várias naturezas, é auto recuperável (*self-healing*), e possibilita o surgimento de uma nova geração de aplicações energeticamente eficientes.

Figura 1 – Visão geral da utilização das Redes Elétricas Inteligentes.

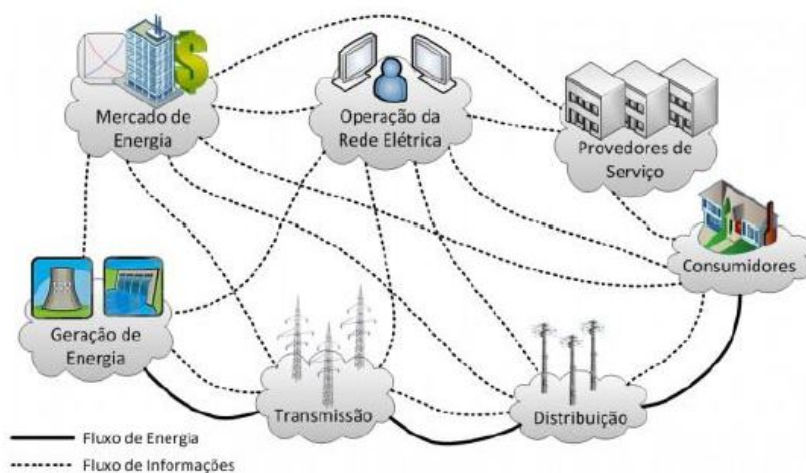


Fonte: Adaptado de (LOPES et al., 2012)

Na Figura 1 são ilustradas as diversas fontes (eólicas, painéis solares, geradores de energia particular), dispositivos (carros elétricos, medidores inteligentes) e sistemas (sistemas de controle, tecnologias de casa inteligente) interagindo na Rede Elétrica Inteligente.

Através de tecnologias digitais, as distribuidoras poderão estabelecer uma comunicação constante com os clientes. Com essa comunicação constante gera-se o controle mais eficiente da energia, a identificação de falhas em tempo real, além do controle dos inúmeros equipamentos que estão conectados à rede, tornando-as redes com maior confiabilidade e qualidade. (LI et al., 2010).

Figura 2 – Rede elétrica inteligente em 7 domínios.



Fonte: Adaptado de (NIST, 2010)

A Figura 2 mostra uma estrutura de Rede Elétrica Inteligente definida em sete domínios diferentes. O domínio de geração está relacionado com a produção em larga escala de energia elétrica, sejam de fontes renováveis ou não renováveis. Este domínio também realiza o armazenamento de energia para o caso de algum contratempo.

Na transmissão, é feito o transporte da energia das centrais geradoras até o cliente, por meio de linhas de transmissão. Neste domínio, o gerador é conectado aos centros através das subestações.

O domínio de distribuição possui as subestações e realiza a distribuição da energia para o cliente final. Aqui é que se devem conectar os medidores inteligentes entre a rede e o consumidor para que possam ser controlados e gerenciados.

O domínio do consumidor é responsável pela comunicação entre usuário e rede, por meio dos *Smart Meters (SM)*, isto é, medidores inteligentes. De acordo com Koponen et al. (2008), os medidores inteligentes são responsáveis pela maior parte das atividades da Rede Elétrica Inteligente. Esses dispositivos processam a todo o momento dados e os enviam para a rede, proporcionando a integração de todo o sistema. Também permitem que os usuários da rede tenham informações em tempo real sobre o seu consumo, favorecendo que

os mesmos se adaptem e melhorem seus hábitos, reduzindo seu consumo e aumentando a eficiência energética. Além disso, outros dispositivos de medição, sensoramento, atuação e geração podem fazer parte desse domínio.

Os demais domínios são operação, mercado e provedor de serviço. No domínio da operação há o gerenciamento e o controle da energia em todas as outras operações, utilizando uma rede de comunicação em duplo sentido, sendo muito importante em tomadas de decisões e correção de erros no sistema. No domínio do mercado existe a coordenação das empresas distribuidoras de energia elétrica, bem como o controle da troca de energia entre subestação e cliente final. Por último, existe o domínio do provedor de serviço que controla todas as ações de empresas terceirizadas.

### 2.1.1 Integração da Rede Elétricas Inteligente com a Internet das Coisas

A Rede Elétrica Inteligente já alcançou ampla adoção em detecção, transmissão e processamento de informações, e agora a tecnologia Internet das Coisas (*Internet of Things, IoT*) desempenha um papel significativo na construção das redes (SALEEM et al., 2017).

A "Internet das Coisas" (do inglês *Internet of Things – IoT*) é um paradigma que preconiza um mundo de objetos físicos embarcados com sensores e atuadores, conectados por redes sem fio e que se comunicam usando a Internet. Moldam uma rede de objetos inteligentes capazes de realizar variados processamentos, capturar variáveis ambientais e reagir a estímulos externos. Esses objetos interconectam-se entre si e com outros recursos (físicos ou virtuais) e podem ser controlados através da Internet. Assim, permitem o surgimento de uma miríade de aplicações que poderão se beneficiar dos novos tipos de dados, serviços e operações disponíveis (ATZORI; IERA; MORABITO, 2010).

A aplicação da Internet das Coisas em Redes Elétricas Inteligentes pode ser encontrada em três principais cenários: (1) Implantar vários dispositivos inteligentes de *IoT* para o monitoramento de estados de equipamentos; (2) Coletar informações de equipamento com a ajuda de seus dispositivos inteligentes de *IoT* conectados por meio de várias tecnologias de comunicação; e (3) Controlar a *Smart Grid* por meio de interfaces de aplicativos.

Os dispositivos de Internet das Coisas de detecção geralmente são compostos de sensores sem fio, identificadores de radiofrequência (*RFIDs*<sup>1</sup>), dispositivos (*machine-to-machine (M2M)* <sup>2</sup>), câmeras, sensores infravermelhos, *scanners a laser*, sistema de posicionamento global (*GPSs*) e dispositivos de coleta de dados. A detecção de informações em uma Rede Elétrica Inteligente pode ser apoiada e aprimorada pela tecnologia *IoT*. Além disso, a tecnologia de Internet das Coisas também desempenha um papel essencial na

<sup>1</sup> <https://www.epc-rfid.info/rfid>

<sup>2</sup> <https://www.ibm.com/blogs/internet-of-things/what-is-m2m-technology/>

implementação de infraestrutura de detecção e transmissão de dados para a Rede Elétrica Inteligente, auxiliando na construção de rede, operação, gerenciamento e monitoramento de segurança, manutenção, coleta de informações, medição, interação do usuário etc.

A Internet das Coisas pode ser aplicada a todos os subsistemas da rede elétrica (geração, transmissão, distribuição e consumo) e aparece como uma solução promissora para aprimorá-los, tornando-se um elemento-chave para a Rede Elétrica Inteligente.

Na área de **geração** de energia, a *IoT* pode ser utilizada para o monitoramento e controle do consumo de energia, unidades, equipamentos, emissão de gases e descarga de poluentes, previsão de uso / produção de energia, armazenamento de energia e conexão de energia, bem como para gerenciar planos de distribuição de energia, armazenamento bombeado, energia eólica, energia de biomassa e centrais fotovoltaicas (SHU-WEN, 2011), (WANG et al., 2012), (BASIT et al., 2017). Na área de **transmissão** de energia, a *IoT* pode ser usada para o monitoramento e controle de linhas de transmissão e subestações, bem como para proteção de torres de transmissão (SHU-WEN, 2011), (WANG et al., 2012). Na área de **distribuição** de energia, a *IoT* pode ser adotada para automatizar a distribuição, bem como no gerenciamento de operações e equipamentos (SALEEM et al., 2017). Na área de **consumo** de energia, a *IoT* pode ser empregada nas casas inteligentes, leituras automáticas de medidores, carregamento de veículos elétricos, coleta de informações sobre o consumo de energia de eletrodomésticos, controle de carga de energia, monitoramento e gerenciamento de eficiência de energia, gerenciamento de demanda de energia e consumo de múltiplas redes (SHU-WEN, 2011), (WANG et al., 2012).

Com o crescimento exponencial de dispositivos conectados à Internet isso tem permitido o surgimento de uma nova geração de aplicativos e serviços. Assim, aplicações que possuem requisitos como: baixa taxa de transferência, longo alcance, baixo atraso e baixo consumo de energia; que não são totalmente atendidos pelas tecnologias de rede atuais. Passam a ser atendidas com a introdução das redes sem fio de baixa potência (*Low Power Wide Area Network - LPWAN*) como um conjunto de novas tecnologias adequadas para *IoT* (RAZA; KULKARNI; SOORIYABANDARA, 2017). Dentre as tecnologias *LPWAN*, a *Long Range (LoRa)* está se consolidando como uma das principais formas de se enviar dados em sistemas de Internet das Coisas (*IoT*), devido a sua capacidade de envio de dados a longas distâncias e baixo gasto energético.

## 2.2 Redes LPWAN

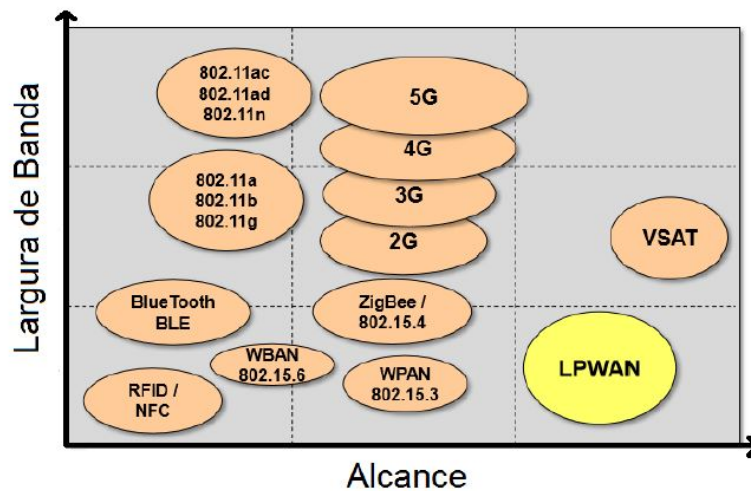
A escolha da tecnologia a ser adotada para a rede de comunicações implantada dependerá de vários fatores. Entre os quais custos envolvidos, distância entre os sensores e medidores até o ponto concentrador de dados e deste, até a rede da concessionária. A topologia física do local, da área de cobertura, a taxa de transmissão e aspectos relacionados

ao desempenho do sistema, atenuação e ruídos.

De acordo com LoRa-Alliance (2015), uma única tecnologia não será capaz de prover soluções para toda a demanda e requisitos de aplicações de IoT. Por muitos anos, tecnologias como ZigBee<sup>3</sup> e *Bluetooth Low Energy (BLE)*<sup>4</sup> têm sido consideradas implementações viáveis de conectividade para IoT (GUBBI et al., 2013), (MIORANDI et al., 2012). Tais tecnologias são caracterizadas por seu baixo consumo de bateria, fator de fundamental importância para dispositivos embarcados, porém apresentam um alcance muito limitado. Para aplicações de cidades inteligentes, por exemplo, que têm como requisito a cobertura da área de uma cidade inteira (ZANELLA et al., 2014), fica evidente a necessidade de uma tecnologia que consiga suprir os requisitos de longo alcance de algumas das aplicações em *IoT*, onde seja possível conectar um dispositivo com a rede independentemente de onde ele tenha que ser posicionado.

Aplicações que necessitam de uma grande área de cobertura e altas taxas de transmissão podem ser atendidas pela tecnologia das redes celulares, devido à sua capilaridade, porém ainda associada a um alto custo de implantação (LORA-ALLIANCE, 2015).

Figura 3 – Comparativo entre *LPWAN* e outras tecnologias.



Fonte: Adaptado de (EGLI, 2015)

A Figura 3 faz um comparativo do alcance e da largura de banda de algumas redes. Em se tratando de *LPWAN*, muito se fala sobre *LoRa*, *Sigfox*<sup>5</sup>, *NB-IoT*<sup>6</sup> e algumas especificações do *LTE(Long Term Evolution)*<sup>7</sup>, que nada mais são do que redes baseadas em *LPWAN*.

<sup>3</sup> <https://zigbee.org/what-is-zigbee/>

<sup>4</sup> <https://www.bluetooth.com/specifications/mesh-specifications/>

<sup>5</sup> <https://www.sigfox.com/en>

<sup>6</sup> <https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/>

<sup>7</sup> <https://searchmobilecomputing.techtarget.com/definition/Long-Term-Evolution-LTE>

As redes *Low Power Wide Area Network (LPWAN)* surgiram para suprir a demanda por uma tecnologia com longo alcance e baixo consumo de energia para aplicações com baixas taxas de transmissão, mantendo um custo de implantação baixo. As principais características da rede *LPWAN* são apresentadas:

**Baixo consumo de bateria:** É de crítica importância para dispositivos finais embarcados que suas baterias tenham uma vida útil longa, visto que seria ecologicamente e economicamente inviável a troca das baterias de bilhões de dispositivos constantemente (BARDYN et al., 2016). Portanto, os dados são transmitidos periodicamente, de tal forma que os dispositivos finais possam permanecer inativos por longos períodos de tempo, reduzindo o consumo de bateria.

**Baixas taxas de transmissão:** As *LPWAN* são projetadas para trabalhar com taxas de transmissão mais baixas, visto que as mensagens enviadas pelos dispositivos finais costumam ser curtas, de 20-256 bytes.

**Comunicação de longo alcance:** O alcance da comunicação deve ser elevado, podendo chegar a dezenas de quilômetros. Um longo alcance não pode ser obtido através do aumento da potência de transmissão considerando a restrição no consumo de bateria já discutida, portanto, além da baixa taxa de modulação, ele é também viabilizado pela forma como a camada física de tais tecnologias é projetada, visando aumentar a sensibilidade de recepção (CENTENARO et al., 2016).

**Baixo custo de implantação:** Os dispositivos finais devem ter um baixo custo, a rede deve ter esforço de implantação reduzido e requerer manutenção mínima. A complexidade do *hardware* e *software* deve ser baixa, significando o uso de arquiteturas e protocolos simples (BARDYN et al., 2016).

**Topologia da rede:** A topologia em estrela é a mais comumente encontrada em redes *LPWAN*, devido a sua simplicidade de implantação e melhor eficiência energética para os dispositivos finais quando comparada a redes *mesh* (topologia de malha (*mesh topology*), por exemplo, Zibgee), as quais são muito utilizadas nas redes de curto alcance como *BLE* e *Zigbee*. Nelas os próprios nós da rede também são encarregados de encaminhar dados (*multihop*) para obter um maior alcance, o que implica num aumento de atividade dos dispositivos finais e também em uma maior complexidade do sistema (CENTENARO et al., 2016).

**Segurança da rede e robustez:** A transmissão de dados de um dispositivo final até o usuário deve ser segura. O operador da rede não deve ser capaz de ter acesso aos dados. Além disso, o sistema deve apresentar robustez contra interferências (BARDYN et al., 2016).

Existem *LPWANs* que operam em faixas de frequência licenciadas, como a *Narrow-*

*Band IoT (NB-IoT)*, *enhanced Machine-Type Communication (eMTC)*<sup>8</sup>, e *Extended Coverage – GSM (EC-GSM)*<sup>9</sup> promovidas pelo *3rd Generation Partnership Project (3GPP)*<sup>10</sup> (BARDYN et al., 2016). Também existem as que operam em faixas de frequência não licenciadas, *Sigfox*, *Ingenu*<sup>11</sup> e *LoRaWAN*, nas bandas industrial, científica e médica (*industrial, scientific and medical - ISM*) de 2.4 GHz, 868/915 MHz, 433 MHz, e 169 MHz dependendo da região de operação (CENTENARO et al., 2016). A seguir, o protocolo *LoRaWAN* é apresentado em mais detalhes.

### 2.2.1 Protocolo *LoRaWAN*

*LoRaWAN* é um padrão de arquitetura de rede aberto definido pelo grupo *LoRa Alliance*, com o objetivo de padronizar a implantação das *LPWAN* no cenário de *IoT*. A camada física de uma rede *LoRaWAN* pode ser implementada tanto com modulação *Frequency-shift keying (FSK)*<sup>12</sup> quanto via modulação *LoRa*<sup>TM</sup>.

*LoRa*<sup>TM</sup> (**Long Range**) é uma modulação para meios sem fio proprietária da *Semtech Corporation*, projetada para operar com longo alcance e baixo consumo de potência. A modulação *LoRa*<sup>TM</sup> é baseada na técnica de espalhamento espectral *Chirp Spread Spectrum (CSS)*<sup>13</sup> para codificar a informação a ser transmitida. A modulação *LoRa*<sup>TM</sup> é independente de *LoRaWAN*, podendo ser utilizada como solução na camada física para outras tecnologias.

### 2.2.2 Topologia da rede

A arquitetura de rede *LoRaWAN* é implantada em topologia estrela de estrelas (*star-of-stars topology*) (similar a uma rede de telefonia celular). A comunicação entre os nós sensores e as estações base passa pela rede sem fio que utiliza a camada física *LoRa*, enquanto a conexão entre os *gateways* e o servidor de rede é tratada em uma rede baseada em *backbone IP*.

Na topologia da rede *LoRaWAN* são especificados quatro elementos: nó final (dispositivo embarcado), *gateway*, servidor de rede e o servidor de aplicação. A Figura 4 mostra o diagrama de blocos da infraestrutura da rede *LoRaWAN*.

Os dispositivos finais é o ponto de partida para uma solução de internet das coisas. Transmite diretamente para todos os *gateways* dentro do alcance, usando *LoRa*. Os *gateways*, por sua vez, estão conectados com um servidor de rede, que realiza a interface da rede *LoRaWAN* com a Internet, de acordo com a Figura 4 (LORA-ALLIANCE, 2015). Os

<sup>8</sup> <https://medium.com/iotforall/emtc-and-nb-iot-2339dd3833e1>

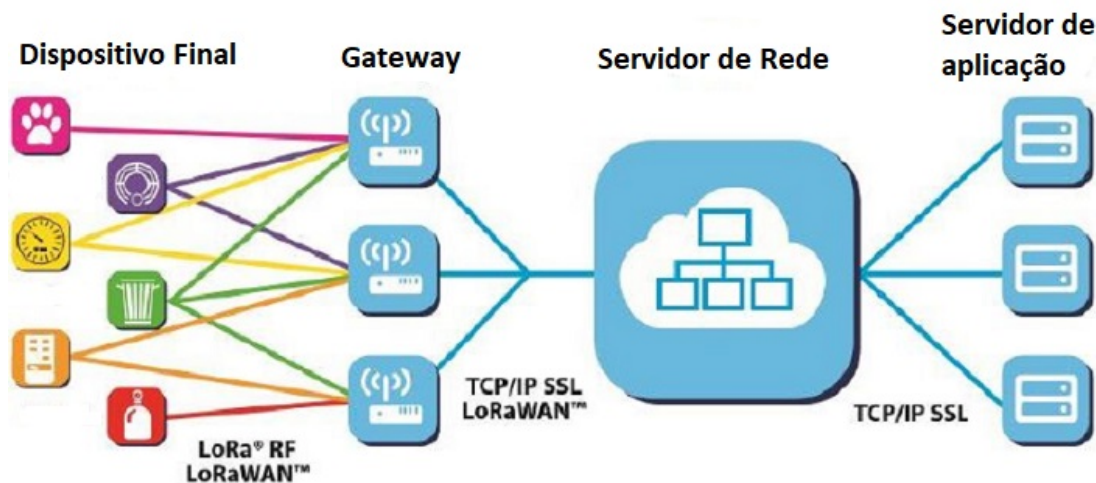
<sup>9</sup> <https://www.gsma.com/iot/extended-coverage-gsm-internet-of-things-ec-gsm-iot/>

<sup>10</sup> <https://www.3gpp.org/about-3gpp>

<sup>11</sup> <https://www.ingenu.com/>

<sup>12</sup> <https://www.sciencedirect.com/topics/engineering/frequency-shift-keying>

<sup>13</sup> <https://journals.sagepub.com/doi/full/10.1155/2015/572861>

Figura 4 – Exemplo de topologia de rede *LoRa* WAN.

Fonte: Adaptado de LoRa-Alliance (2015)

*gateways* devem ser capazes de receber volumes grandes de dados. Uma rede *LoRa* WAN viabiliza isso através da utilização de *Adaptive Data Rate (ADR)* e comunicação multicanal. Os fatores críticos para determinar a capacidade do *gateway* são: número de canais, taxa de transmissão (tempo no ar), comprimento das mensagens e periodicidade de transmissão dos dispositivos finais (LORA-ALLIANCE, 2015). Toda informação recebida pelos múltiplos *gateways* é repassada para o servidor. O servidor de rede (*LoRa* WAN Network Server - LNS) é o responsável por identificar e descartar dados redundantes, realizar verificações de segurança, administrar as taxas de transmissão via *ADR*, etc. Existem algumas soluções proprietárias e *open-sources* de servidores de redes para projetar uma infraestrutura básica *LoRa* WAN tais como, *ChirpStack*<sup>14</sup>, *Loriot*<sup>15</sup>, *Actility*<sup>16</sup> e *The Things Network (TTN)*<sup>17</sup>. O *TTN* é uma rede baseada em *cloud servers* que conecta os *gateways* *LoRa* WAN ao redor do planeta. Assim, cada vez que alguém compra uma estação base *LoRa* e o conecta ao *TTN*, sua cobertura é expandida para todos os usuários (com cobertura gratuita). O diferencial do *TTN* é ser uma rede aberta (*open-source*) e colaborativa (*crowd-sourced*), isso tem auxiliado a sua forte adesão à comunidade de desenvolvedores. Quanto à segurança, a *LoRa* WAN implementa criptografia *Advanced Encryption Standard (AES)*, tanto para garantir a autenticidade dos dispositivos conectados na camada de rede, quanto a privacidade dos dados do usuário final na camada de aplicação (LORA-ALLIANCE, 2015).

<sup>14</sup> <https://www.chirpstack.io/>

<sup>15</sup> <https://www.loriot.io/>

<sup>16</sup> <https://www.actility.com/lorawan-network-server/>

<sup>17</sup> <https://www.thethingsnetwork.org/>

### 2.2.3 Dispositivos finais

Os dispositivos finais (ou nós finais) do *LoRaWAN* (ex: sensores) normalmente usam baixa energia e são alimentados por bateria, portanto, *LoRaWAN* prevê três classes de dispositivos para diferentes perfis de aplicação, oferecendo um câmbio entre duração da bateria e latência no *downlink*. Conforme LoRa-Alliance (2015), as diferentes classes existentes são listadas e explicadas a seguir:

**Classe A - Dispositivos finais bidirecionais** - Dispositivos desta classe permitem comunicação bidirecional, sendo que após toda transmissão feita pelo dispositivo final existem duas janelas de recepção. A transmissão dos dispositivos finais é agendada dependendo das necessidades de cada aplicação, havendo uma pequena variação devido ao protocolo de *Media Access Control (MAC)* do tipo *ALOHA*<sup>18</sup>. Esta classe apresenta a melhor eficiência energética por permitir longos períodos de inatividade do dispositivo final. Caso seja necessário o recebimento de dados do servidor, é preciso aguardar a próxima transmissão agendada pelo dispositivo final, gerando latência no *downlink* proporcional ao período de transmissão da aplicação.

**Classe B - Dispositivos finais bidirecionais com janelas de recepção agendadas** - Além das janelas de recepção já presentes nos dispositivos finais Classe A, são agendadas outras janelas de recepção sincronizadas por um sinal emitido pelo *gateway (beacon)*. Desta forma a latência no *downlink* pode ser reduzida ao custo de uma menor eficiência energética por aumentar o nível de atividade do dispositivo final.

**Classe C - Dispositivos finais com janelas de recepção máxima** - A janela de recepção é praticamente contínua, exceto quando o dispositivo final está transmitindo. A latência no *downlink* se torna mínima, porém a eficiência energética é extremamente reduzida já que o dispositivo está constantemente ativo.

Vale destacar que o nó final (ou dispositivo embarcado) é tipicamente o originador dos dados, e interage com o mundo físico. Os nós normalmente têm: (1) Sensores (usados para detectar os parâmetros variáveis, por exemplo, temperatura, umidade, acelerômetro, GPS); (2) *Transponder LoRa* para transmitir sinais através do método de transmissão via rádio patenteado pela *LoRa*, e (3) opcionalmente um microcontrolador (com memória integrada).

Os sensores podem se conectar ao chip *transponder LoRa* ou o sensor pode ser uma unidade integrada com o chip *transponder LoRa* incorporado. É possível programar os microcontroladores em diversas linguagens. Isso permite que os desenvolvedores usem os dados de sensores (como acelerômetros, temperatura, etc.) e programem diversas aplicações. Nas redes elétricas inteligentes esses dispositivos são aqueles computadores embarcados que estão mais próximos dos equipamentos elétricos da rede.

<sup>18</sup> <https://www.mdpi.com/1424-8220/19/4/838/pdf>

## 2.3 Plataformas Embarcados e *Middlewares*

De acordo com Wolf (2012), um sistema de computador embarcado é qualquer dispositivo que contém um computador programável, mas que não é desenvolvido para ser computador de propósito geral. Entre as aplicações mais comuns estão os eletrodomésticos (geladeira, forno de micro-ondas), telecomunicações (*smartphones*), veículos (freio *ABS*, computadores de bordo) e em controles industriais (robôs, sistemas de monitoramento e diagnóstico de maquinário). Adicionalmente, Marwedel (2006) define sistemas embarcados como sistemas de processamento de informação em produtos fechados. Ou seja, todos os produtos citados contêm, além de outros componentes, um computador que foi planejado, desenvolvido ou programado para desempenhar um conjunto específico de tarefas.

Um sistema embarcado consiste na combinação entre *hardware* e *software* projetados para desempenhar uma determinada aplicação (MARWEDEL, 2006). Devido ao grande espaço de projeto em sistemas embarcados, a diversidade de soluções para a implementação de uma determinada aplicação torna o processo muito complexo. Por isso, a componentização e reuso de plataformas de *hardware* e *software* pode permitir uma redução no espaço de soluções e, portanto, no tempo de projeto.

Para mitigar a complexidade do projeto de *software* embarcado, surge como solução à utilização de Sistema Operacional Embarcado (SOE) para controlar e desempenhar todas as funções no dispositivo. Um SOE dedicado a aplicações de sistemas embarcados deve estar sujeito aos requisitos naturais exigidos por esses sistemas. As principais restrições no projeto são: limite de memória disponível, desempenho do processador e consumo de potência. Um sistema operacional embarcado com foco na internet das coisas é um sistema operacional com baixo *footprint* (consumo de *CPU*, *RAM* e *flash*) e com boas opções de conectividade, principalmente sem fio, modular, escalável, seguro e capaz de gerenciar eficientemente o consumo de energia.

No desenvolvimento de aplicações embarcadas é frequente a dificuldade em usar ferramentas de produtividade, sendo mais comum quando a plataforma alvo é um computador de propósito geral. A maior dificuldade está na pluralidade de plataformas embarcadas e no alto grau de personalização de cada produto.

De acordo com Rahman, Ozcelebi e Lukkien (2016), uma plataforma de programação para Internet das Coisas pode ser definida como um kit de ferramentas para desenvolver aplicativos *IoT* de acordo com um determinado estilo ou método. Uma plataforma de programação *IoT* adota um ou mais protocolos de aplicativo. Fornece um sistema operacional, *APIs* e bibliotecas que implementam serviços sobre esses protocolos. Pode também fornecer ferramentas para desenvolvimento, teste e implantação. O Autor também destaca que existem diversos exemplos de plataformas de programação *IoT*, tais como: *ARM*

*mbed IoT*<sup>19</sup>, *Works com Nest*<sup>20</sup>, *Allseen Alljoyn*<sup>21</sup>, *Open Interconnect Consortium (OIC) IoTivity*<sup>22</sup>, *Eclipse Leshan*<sup>23</sup>, *Eclipse Wakaama*<sup>24</sup>, *OSGi openHAB*<sup>25</sup>, *OpenRemote*<sup>26</sup>, *Apple HomeKit*<sup>27</sup> e *Node-RED*<sup>28</sup>. A lista, não exaustiva, de plataformas mostra-se em crescimento constante.

Nos estudos de Rahman, Ozcelebi e Lukkien (2016) e Sabri, Kriaa e Azzouz (2017) a plataforma *ARM mbed IoT* foi verificada em diferentes aspectos arquiteturais. E nos trabalhos de Bhati (2017), Balsamo et al. (2017), Awasthi et al. (2017) e Gecevic e Bjazic (2017), o *Mbed* foi utilizado para a criação de aplicações voltada para Internet das Coisas devido sua capacidade de abstrair a complexidade do controle do *hardware*, *software* e projeto. A seguir, a plataforma *ARM Mbed* é apresentada.

### 2.3.1 Plataforma ARM Mbed

A plataforma *ARM Mbed IoT* é um *framework* que fornece um sistema operacional embarcado de código aberto (*ARM mbed OS*). Além de um grande número de ferramentas e bibliotecas padrões projetado para desenvolver aplicações *IoT*. Também fornece *drivers* para diferentes protocolos de comunicação existentes, tais como *Wi-Fi*, *Bluetooth Low Energy (BLE)*, *Thread*, *6LoWPAN*<sup>29</sup> e *Ethernet*.

Na figura 5, de cima para baixo, os códigos, bibliotecas e ferramentas disponibilizadas pela comunidade de desenvolvedores de *software* que contribuem com a plataforma *Mbed* são ilustrados. Em seguida, são apresentadas as funcionalidades disponibilizadas pelo SO do *Mbed*. Além das funcionalidades básicas de um sistema operacional, também oferece camadas de segurança e conectividades. O *Mbed OS* usa uma camada de abstração de *hardware* (*HAL*) para suportar as partes mais comuns de um microcontrolador. Essa base facilita a criação de aplicativos em um conjunto comum de interfaces de programação de aplicativos (*API's*). Possui as bibliotecas necessárias e o suporte ao *driver* para periféricos padrão *MCU*, como *I2C (Inter-Integrated Circuit)*, serial e *SPI (Serial Peripheral Interface)*. Por fim, de maneira geral, as interfaces de *hardware* suportadas pelo *framework* são ilustradas.

O *mbed OS* é o componente fundamental desta plataforma, pois inclui todos os recursos necessários para o desenvolvimento de aplicações básicas, incluindo segurança,

<sup>19</sup> <https://www.mbed.com/en/>

<sup>20</sup> <https://nest.com/whats-happening/>

<sup>21</sup> <https://openconnectivity.org/developer/reference-implementation/alljoyn>

<sup>22</sup> <https://iotivity.org/>

<sup>23</sup> <https://www.eclipse.org/leshan/>

<sup>24</sup> <https://www.eclipse.org/wakaama/>

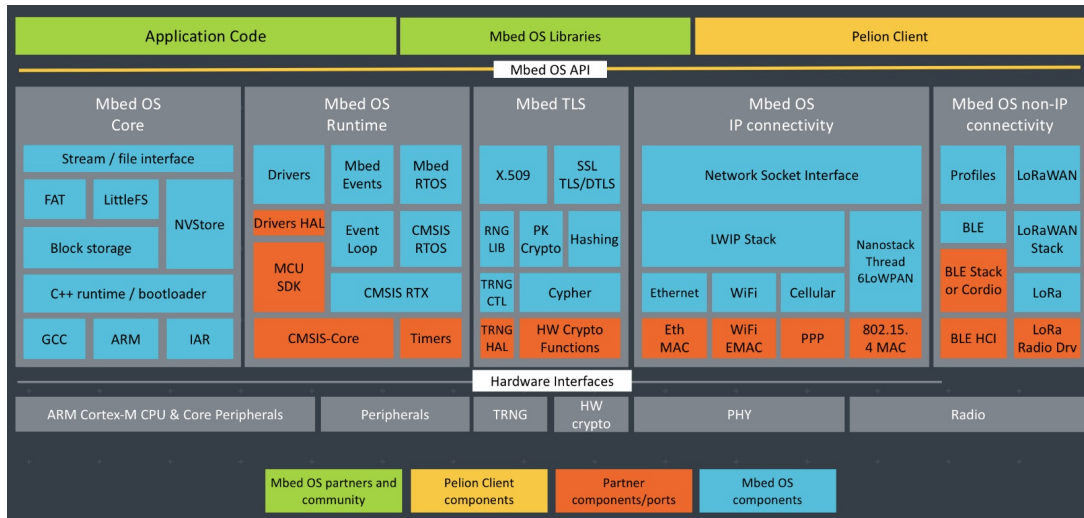
<sup>25</sup> <https://www.openhab.org/v2.4/docs/developer/prerequisites/osgi.html>

<sup>26</sup> <http://www.openremote.com/>

<sup>27</sup> <https://www.apple.com/ios/home/>

<sup>28</sup> <https://nodered.org/>

<sup>29</sup> <https://www.uctec.com/en/6lowpan/6lowpan-w/>

Figura 5 – Arquitetura básica do *Mbed*.

Fonte: (ARM Mbed, 2019a).

conectividade e *drivers* para periféricos e sensores de E/S. O ecossistema *Mbed* disponibiliza um ambiente de desenvolvimento integrado *on-line* (*IDE*) que fornece acesso ao sistema operacional *Mbed* sem exigir pré-configurações específicas. Isso permite a criação rápida de protótipos, sem usar *hardware* de depuração.

O reuso dos componentes da plataforma pode ser ainda reforçado pela adoção de padrões na arquitetura do *software* (*Design Pattern*) e projeto do sistema. Estas demandas também apontam para a necessidade de uma camada intermediária (o *middleware*) que permita ao projetista elevar o nível de abstração, ajudando a alcançar uma redução nos tempos de desenvolvimento de aplicações.

### 2.3.2 Middlewares

O *middleware* é um *software* com múltiplas funcionalidades que fornece serviços para aplicações além das que já são oferecidas pelo sistema operacional. De acordo com Krakowiak (2007), o *middleware* clássico tem como propósito integrar componentes de *software* reusáveis para diminuir o tempo e o esforço despendido no desenvolvimento de aplicações e serviços com alta qualidade. O *middleware* também fornece uma visão uniforme de redes, protocolos e recursos de sistemas operacionais, ajudando a simplificar o desenvolvimento de novas aplicações.

Sendo o *middleware* um conjunto de *softwares* localizado entre as aplicações e o sistema operacional junto com a infraestrutura de rede (MOHAMED et al., 2017). Ocorre que é preciso buscar um compromisso entre a transparência total oferecida pelo conceito de *middleware* (oriundo de sistemas distribuídos clássicos) e a eficiência necessária ao sistema embarcado, em função das limitações de recursos computacionais, de maneira a assegurar

menor custo e melhor uso dos recursos da plataforma. Assim, é comum em sistemas deste porte a sobreposição dos conceitos de *middleware* com o de sistema operacional, oferecendo os dois tipos de recursos em um só pacote.

Esse desafio de caracterizar um *middleware* é destacado por Rodríguez-Molina e Kammen (2018), ao afirmar que é difícil avaliar as infinitudes de propostas de *middleware* para as redes elétricas inteligentes, devido ao fato de que as propostas discutem amplamente sobre o que é *middleware* e o que pode ser considerado como tal. Às vezes, o *middleware* é mencionado como um conceito que não é totalmente implementado, enquanto em outros casos o *middleware* inclui recursos que pertencem a camadas imediatamente superiores e inferiores, como as de redes e aplicativos.

Ainda no que diz respeito às redes elétricas inteligentes, de acordo com a taxonomia apresentada por Rodríguez-Molina e Kammen (2018), existem quatro propriedades diferentes que devem ser levadas em conta para auxiliar a descrição de uma proposta de *middleware*, são elas: disponibilidade de serviços, recursos computacionais, acoplamento de mensagens e distribuição do *middleware*.

**A. Disponibilidade de serviços:** Trata-se a respeito da quantidade de serviços que devem ser oferecidos pelo *middleware*.

**B. Capacidades computacionais:** Leva-se em conta o *hardware* necessário que deve ser usado para executar a proposta nos dispositivos que foram incluídos como parte da implantação do *Smart Grid*.

**C. Acoplamento de mensagem:** Busca-se avaliar a velocidade que as mensagens geradas por uma entidade são consumidas por outra (aquela que se espera recebê-las). Dependendo do caso específico, haverá diferentes necessidades para as mensagens que estão sendo transferidas; como alguns deles devem ser enviados o mais rápido possível, enquanto outros podem ser armazenados até que sejam solicitados por uma parte interessada. Dessa maneira, o acoplamento de mensagens está intimamente associado à necessidade de entregar as informações necessárias.

**D. Distribuição de *middleware*:** Mede-se quantos dispositivos em uma implantação possuem qualquer implementação parcial do *middleware* instalado neles.

## 2.4 Conclusão

Este capítulo descreveu, no primeiro momento, as Redes Elétricas Inteligentes, sua infraestrutura e a integração com a Internet das Coisas. Destaca-se que a *IoT* pode ser aplicada a todos os subsistemas da rede elétrica e aparece como uma solução promissora para aprimorá-los. No segundo momento, as redes *LPWANs* são detalhadas, dando ênfase ao protocolo *LoRAWAN* que é verificado como uma das principais formas de se enviar

dados em sistemas de Internet das Coisas (*IoT*), devido a sua capacidade de envio de dados a longas distâncias e baixo gasto energético. Por último, o conceito de Plataformas Embarcadas é abordado, dando destaque ao sistema operacional embarcado *Mbed OS*, os conceitos de *Middlewares* e suas características, de acordo com a taxonomia apresentada.

Como desafio geral, esta pesquisa busca propor e desenvolver um *middleware* que possa abstrair as diferenças de *hardware* e *software* das diversas plataformas existentes. Além disso, deve fornecer um ponto de acesso, ou seja, uma *API* (*Application Programming Interface*) para ser facilmente utilizada pela aplicação. Quanto aos recursos computacionais consumidos pelo *middleware*, este deve ser leve o suficiente para rodar em plataformas de *hardware* com recursos limitados, pois será executado nos dispositivos finais. A respeito das trocas de mensagem, ocorrerão de forma desacoplada, tendo em vista que a *API* de comunicação é baseada em eventos. E por fim, cada dispositivo final deve executar uma versão atômica (não distribuída) do *middleware* implementado.

## 3 Trabalhos Relacionados

Neste capítulo, a literatura foi revisada sob três categorias de pesquisa, nas quais a solução proposta está na interseção: (i) Arquitetura de *middleware IoT*; (ii) uso das redes *LPWAN* em soluções *IoT* para *smart grids*; e (iii) utilização de sistemas operacionais embarcados visando à diminuição do tempo de projeto e aumento da portabilidade.

### 3.1 Arquiteturas de *middlewares* de Internet das Coisas

Várias plataformas de *middleware* foram propostas e desenvolvidas para resolver diferentes desafios. O trabalho de Obaidat et al. (2018), propõe uma arquitetura de *middleware* sensível (adaptável) ao contexto. Assim, busca-se atender a um grande número de solicitações diversificadas no atual sistema da rede elétrica inteligente, que incluem redes de sensores, comunicações via satélite, *Wi-Fi* e outras redes sem fio possíveis. O *middleware* compatível com o contexto fornece aprimoramentos consideráveis ao gerenciamento de dados e serviços de rede inteligente que utilizam comunicação e computação bidirecional. Além disso, integrou à infraestrutura de comunicações da rede inteligente para processamento eficiente de dados, buscando levar à sustentabilidade e estabilidade, auxiliando assim, no gerenciamento, por exemplo, da casa inteligente. A proposta desta dissertação apresenta um modelo de *middleware* adaptável que descreve de forma genérica entidades responsáveis por interagir com o ambiente e realizar a comunicação de forma bidirecional (ex: periférico, protocolo e comunicação). Bastando o integrador do *middleware* especializá-lo em tempo de projeto, para atender as demandas de sensores, atuadores, protocolos e comunicações existentes.

Os autores em Ramesh et al. (2019), apresentam uma proposta de *middleware*, que executa no servidor, para auxiliar na integração com todos os nós da rede *IoT*. A comunicação utilizada nos dispositivos é *LoRaWAN*. Para lidar com a baixa banda disponibilizada por essa rede, buscou-se utilizar o protocolo da *Google Protobuf*<sup>1</sup>, que não trafega dados em formatos *json*, para diminuir o tamanho do *payload* de dados utilizados. A solução é validada utilizando um caso de uso envolvendo iluminação inteligente. A proposta de *middleware* deste trabalho difere por executar no dispositivo final. Além disso, possui uma estratégia de fragmentação de dados para trafegar grandes lotes de informação pela rede *LoRaWAN*.

Em Mohamed et al. (2017), os autores propõe um *middleware* chamado *Smart-CityWare*. Sua abordagem orientada a serviços busca integrar de forma eficaz os para-

<sup>1</sup> <https://developers.google.com/protocol-buffers>

digmas de computação *Cloud of things (CoT)* e *Fog Computing*. O principal objetivo do *SmartCityWare* é fornecer um ambiente virtual a ser usado para desenvolver e implantar aplicativos de cidades inteligentes. Consiste em um conjunto de serviços e um ambiente de tempo de execução de vários agentes. A plataforma proposta é testada com serviços de intermediação, chamada e localização com uma placa *Arduino* para medição de temperatura e umidade. A proposta de *middleware* desta dissertação utiliza-se especificamente do paradigma da computação na borda da rede (*Fog Computing*) buscando auxiliar o desenvolvimentos de aplicações de controle, monitoramento e medição dos dispositivos existentes na rede elétrica inteligente.

No trabalho realizado em Patti et al. (2014), é apresentado o projeto de um *middleware* orientado a eventos para serviços gerais em *smart grid (SG)*. O *middleware* proposto fornece fácil integração de tecnologias heterogêneas, sem fio e com fio. Possibilita a interoperabilidade independente de *hardware* nessas tecnologias. Ao mesmo tempo, a solução proposta apresenta intrinsecamente uma comunicação segura e confiável entre diferentes atores. Para demonstrar a implantação prática, um estudo de caso foi desenvolvido e seus resultados foram discutidos. O *middleware* proposto nesta dissertação também possui uma arquitetura adaptável e baseada em eventos auxiliando criar aplicações mais resilientes a falhas. Já a integração com as diferentes plataformas de *hardware* é obtida pelo uso de sistemas operacionais embarcados.

Diversas plataformas comerciais, por exemplo, ThingSpeak<sup>2</sup>, NodeRed<sup>3</sup>, Geniot<sup>4</sup> e OpenHAB<sup>5</sup> facilitam a implementação de aplicações *IoT*, e oferecem suporte a várias placas controladoras e serviços. A principal diferença entre o *Middleware SGAM* e outras plataformas de *middleware* propostas é que o *SGAM* auxilia a criação da parte das aplicações que será executada no nó final com ênfase na comunicação por *LPWAN*. Sua arquitetura adaptável favorece a criação de novas aplicações embarcadas de maneira mais simples pela utilização dos conceitos de componentes e reuso de *software* e permite ser implementada para diferentes plataformas.

### 3.2 Utilização das Redes *LPWAN* em soluções *IoT* para Redes Elétricas Inteligentes

Alguns estudos foram realizados e diversas soluções foram proposta integrando aplicações de Internet das Coisas e Redes Elétricas Inteligentes com as redes *LPWAN*. No trabalho elaborado por Filho, Filho e Moreli (2016), afirma-se que as iniciativas de cidades inteligentes têm sido objeto de discussão em vários fóruns em todo o mundo.

---

<sup>2</sup> <https://thingspeak.com/>

<sup>3</sup> <https://nodered.org/>

<sup>4</sup> <https://www.geniot.io/site/>

<sup>5</sup> <https://www.openhab.org/>

E, como plataforma principal, utiliza-se o conceito de Internet das Coisas. Buscando alternativas convergentes de comunicações e infraestrutura, os autores exploraram as principais características de *RF Mesh* e *LoRaWAN*, fazendo uma comparação entre elas e discutindo a adequação do *LoRaWAN* à Rede Elétrica Inteligente. Concluem que para serviços de Internet das Coisas, incluindo Redes Elétricas Inteligentes, a tecnologia emergente *LoRaWAN* representa uma alternativa realmente convergente que também adota padrões abertos e interoperáveis, mas a um custo menor se comparado ao *RF Mesh*. Esse estudo comparativo auxiliou a tomada de decisão para utilizar o protocolo na solução proposta. Assim, o protocolo *LoRaWAN* foi escolhido para fornecer a infraestrutura de conectividade entre os diversos dispositivos da Rede Elétrica Inteligente.

Em Wibisono et al. (2017), os autores mostram que a empresa de energia elétrica nacional da Indonésia inaugurou o "*Bali Eco Smart Grid*" para apoiar o programa de incentivo ao uso de energia renovável, melhorar o atendimento ao cliente e a eficiência nas empresas. Desta forma, a PLN Bali implementou a tecnologia da *LPWAN* no medidor inteligente. Com isso, avaliou a implementação quanto à conexão do cliente, precisão do controle e a eficiência de custos. Primeiro, a taxa de leitura de sucesso do medidor inteligente baseado em *LoRaWAN* foi avaliada enviando a leitura do medidor a cada 30 minutos e comparando com o método convencional. Em segundo lugar, também foi testado à capacidade do medidor inteligente incluindo a ativação e desativação remotas do medidor e controle remoto de limitação de carga. Como resultado, foi observado que a tecnologia *LoRaWAN* se adequou perfeitamente a solução, obtendo precisão no controle dos dados, eficiência dos custos de investimento e operacional. A qualidade no controle dos dados obtido com o protocolo *LoRaWAN* é um requisito bastante relevante para este trabalho, tendo em vista que a proposta do *middleware* é ser uma base confiável para a criação de diversas outras aplicações.

Em Ghiasimonfared et al. (2017), os autores buscaram melhorar o serviço oferecido na gestão de um parque tecnológico com o fornecimento de uma comunicação confiável e escalável com alta capacidade de cobertura. O *LoRa* foi usado como uma camada física proprietária para conectividade *LPWAN*. O desafio fundamental foi desenvolver a interoperabilidade entre as tecnologias *LoRa* / *G3-PLC*<sup>6</sup>. A arquitetura da rede criada é mostrada e explicada sua implementação. O aspecto fundamental desta interface de comunicação é um nó (*Bridge*) que permite a tradução dos dados do *LoRa* ao *G3-PLC* e vice-versa. Neste trabalho, busca-se generalizar a proposta de serviços oferecidos, disponibilizando um *middleware* que executa exclusivamente no nó final e disponibiliza uma *api* de conectividade *LoRa* para o desenvolvimento de diferentes soluções.

No trabalho de Li et al. (2017), os autores propuseram um sistema de leitura de medidores sem fio com o objetivo de obter baixo consumo de energia em leitura remota de

---

<sup>6</sup> <http://www.g3-plc.com/what-is-g3-plc/g3-plc-overview/>

hidrômetros. Foi utilizado um relé multinível e um concentrador baseado na comunicação *LoRa*. O sistema foi composto por medidores de água inteligente sem fio, repetidores de múltiplos estágios, um concentrador de aquisição de dados, uma estação de trabalho e um *data center*. O Medidor inteligente de água possui a função de comunicação sem fio *LoRa* e foi instalado em prédios residenciais altos para monitoramento de medidores de água e status de equipamentos de medição. Com a conclusão do projeto atingiu a medição precisa dos dados do medidor de água, e foi possível transmitir os dados utilizando *LoRa*. Observa-se que o concentrador de dados surge como um elemento relevante neste sistema, pois ele garante que dados lidos em grande quantidade não sejam perdidos. Na arquitetura proposta nesta dissertação, buscará lidar com esse problema de perda de dados aplicando técnicas para fragmentar grandes volumes de dados em blocos menores.

Observa-se com esses trabalhos a aderência da tecnologia *LoRaWAN* com redes que buscam monitoramento remoto com o consumo baixo de energia, justificando assim, a sua escolha para a elaboração do protótipo neste projeto.

### 3.3 Explorando sistemas operacionais embarcados visando à diminuição do tempo de projeto e aumento de portabilidade

Como plataforma de *software*, diversos estudos de Internet das Coisas sugerem a utilização de sistemas operacionais embarcados visando à diminuição do tempo de projeto e aumento da portabilidade das soluções. Na produção de Bhati (2017), um sistema foi proposto para a detecção de queda de idosos. O sistema projetado utilizou um microcontrolador *ARM NXP Mbed LPC1114* responsável pelo processamento principal do protótipo projetado. Esta unidade ajuda na transmissão dos dados coletados de sensores para o aplicativo móvel *Android* desenvolvido com a ajuda de comunicação *Bluetooth* em um curto intervalo. Para validar o sistema de detecção de queda desenvolvido, ele foi testado por quedas intencionais em ambiente de laboratório. O *Mbed* foi utilizado como unidade de processamento principal. Sendo responsável por transmitir os dados coletados dos sensores para o aplicativo móvel. Na proposta desta dissertação, a plataforma *Mbed* também será utilizada como unidade de processamento principal. Além de oferecer suporte ao *middleware* a diversos outros dispositivos.

Como descrito no trabalho de Balsamo et al. (2017), um sistema de coleta de energia foi criado e integrado a um *framework* de Internet das Coisas. A integração foi realizada com o *ARM Mbed OS* com o objetivo de ter disponível um sistema operacional embarcado de código aberto, *drivers* para diferentes protocolos de comunicação existentes e poder embarcar a solução em diferentes plataformas. A solução foi projetada e validada utilizando um microcontrolador da *NXP* de baixa potência. Como já destacado, o *Mbed* funcionou como uma plataforma para disponibilizar recursos de *hardware*, *software* e rede

de forma mais prática e rápida, acelerando o desenvolvimento da solução. No projeto de *middleware* desta dissertação, a integração com os diversos dispositivos e redes também ocorrerão através dos recursos disponibilizados pela plataforma *Mbed*.

Neste artigo Awasthi et al. (2017), uma solução foi desenvolvida para monitorar/testar qualquer tipo de sistema de vácuo. A plataforma *mBed* foi escolhida para manter os custos baixos. Também para proporcionar ao sistema uma maior portabilidade e facilidade de uso. Para esta dissertação, a motivação para utilizar da plataforma *Mbed* se assemelha ao trabalho apresentado. No trabalho de Gecevic e Bjazic (2017), um sistema computacional embarcado foi desenvolvido para o controle, sem contato, do fluxo e temperatura da água para lavar as mãos. O código foi escrito em linguagem de programação orientada a objeto C++ para a plataforma de desenvolvimento *Nucleo-F303RE* que se baseia no microcontrolador *ARM* de 32 bits *Cortex M4*. A programação do microcontrolador foi feita a partir de um ambiente de desenvolvimento integrado *MBED OS*. A escolha da plataforma *Mbed* foi realizada visando manter os códigos-fontes abertos. O código é traduzido usando o compilador *on-line*. Além da proposta apresentada, destaca-se a ampla utilização do ecossistema *Mbed*, que além do sistema operacional, utilizou a placa de desenvolvimento da *nucleo* e o compilador *online*. A placa de desenvolvimento que será utilizada na implementação do protótipo deste trabalho será uma *nucleo*. Esta placa por possuir os conectores no padrão *Arduino*, permite a adição de diversos outros módulos(*shields*, sensores, atuadores) para a realização dos testes.

Observa-se que para reduzir o tempo de projeto e, conseqüentemente, os custos, é crucial que programadores utilizem plataformas que facilitam e aceleram a programação. Para esses fins a plataforma *ARM mbed IoT* vem se mostrando eficaz e bastante utilizada. Assim, o *Mbed* foi escolhido como a plataforma alvo para ser implementado o *middleware* sugerido nesta dissertação.

### 3.4 Conclusão

Este capítulo descreveu o estado da arte das pesquisas relacionadas à criação de arquiteturas de *middleware IoT*, uso das redes *LPWAN* em soluções *IoT* para *smart grids*, assim como, foram apresentados trabalhos que exploraram sistemas operacionais embarcados visando à diminuição do tempo de projeto e aumento da portabilidade. No final de cada seção, um comentário conclusivo foi realizado sobre as leituras, fundamentando os direcionamentos desta pesquisa.

A Tabela 1, mostra a comparação entre os principais trabalhos analisados e uma lista de requisitos desejáveis para um *middleware IoT* no contexto de sistemas embarcados e redes elétricas inteligentes. As duas primeiras colunas da tabela se referem às definições arquiteturais de *software*. A primeira coluna indica a execução da solução de *software* no

próprio nó final e a segunda, discorre a respeito de uma arquitetura de *software* adaptável. A terceira e quarta coluna abordam as características da infraestrutura de comunicação escolhida. Suporte as redes *LPWAN* e utilização de estratégias de otimização do *payload* ou fragmentação de dados para contornar as limitações de largura de banda destas redes. As duas últimas colunas versam sobre o suporte obtido ao utilizar sistemas operacionais embarcados padrões de mercado. Indicando na coluna cinco a obtenção de suporte a recursos de *hardware*, *software* e rede, e na coluna seis, a possibilidade da mesma aplicação rodar em diferentes plataformas de *hardware* (microcontroladores).

Dois aspectos importantes são extraídos da tabela: (1) As pesquisas tendem a ter bastante aderência ao uso das redes *LPWAN* e (2) há uma boa variedade de artefatos direcionados para executar no próprio sistema embarcado (nó final). Destaca-se assim, a relevância em desenvolver uma camada de abstração (*middleware*) que auxilie na criação de novas soluções embarcadas com suporte a essas redes. Observa-se também, que as soluções apresentadas atendem apenas de forma parcial aos requisitos destacados. Neste trabalho, todos os requisitos presentes na Tabela 1 são atendidos pelo *Middleware SGAM*, em contraponto aos estudos encontrados.

Tabela 1 – Comparação dos trabalhos relacionados quanto aos requisitos

| Trabalhos \ Requisitos       | Direcionado para o nó final | Arquitetura adaptável | Suporte LPWAN | Otimização ou fragmentação de dados | Suporte a hardware, software e rede | Rodar em diferentes plataformas HW |
|------------------------------|-----------------------------|-----------------------|---------------|-------------------------------------|-------------------------------------|------------------------------------|
| Obaidat et al. (2018)        |                             | X                     |               |                                     |                                     |                                    |
| Ramesh et al. (2019)         |                             |                       | X             | X                                   |                                     |                                    |
| Mohamed et al. (2017)        | X                           |                       |               |                                     |                                     |                                    |
| Patti et al. (2014)          |                             | X                     |               |                                     |                                     | X                                  |
| Filho, Filho e Moreli (2016) |                             |                       | X             |                                     |                                     |                                    |
| Wibisono et al. (2017)       | X                           |                       | X             |                                     |                                     |                                    |
| Ghiasimonfared et al. (2017) | X                           |                       | X             |                                     |                                     |                                    |
| Li et al. (2017)             | X                           |                       | X             | X                                   |                                     |                                    |
| Bhati (2017)                 |                             |                       |               |                                     | X                                   |                                    |
| Balsamo et al. (2017)        |                             |                       |               |                                     | X                                   | X                                  |
| Awasthi et al. (2017)        |                             |                       |               |                                     |                                     | X                                  |
| Gecevic e Bjazic (2017)      | X                           |                       |               |                                     | X                                   |                                    |

A proposta de arquitetura do *middleware*, requisitos de comunicação e as plataformas de *hardware* e *software* são detalhadas no capítulo seguinte.

## 4 Arquitetura Proposta

Este capítulo apresenta o *middleware* SGAM (*Smart Grid Application Middleware*). Serão fornecidos detalhes de cada módulo, estrutura, geração e aplicabilidade, de modo a constituir uma arquitetura a ser utilizada no desenvolvimento de aplicações para Redes Elétricas Inteligentes. Além disso, uma implementação funcional da arquitetura é oferecida.

### 4.1 Requisitos Funcionais

A abordagem do *Middleware SGAM* foi inicialmente contextualizada na taxonomia (apresentada na subseção 2.3.2) de Rodríguez-Molina e Kammen (2018) para auxiliar a definição de aspectos arquiteturais. Além disso, o desenho arquitetural do *middleware* seguiu a metodologia de projeto orientado a pontos adaptáveis (*Hot Spot Driven Design*), desenvolvida em Wolfgang (1994), que consiste em identificar os pontos fixos e os pontos adaptáveis do domínio de uma aplicação, dando ao *middleware* a capacidade de ser flexível e se moldar às diferentes aplicações e plataformas embarcadas de *software*.

Diferentes casos de uso de controle, monitoramento e medição fornecidos para projetistas de soluções de Internet das Coisas e Redes Elétricas Inteligentes foram pensados. Posteriormente, dois casos de uso foram implementados (Capítulo 5) referentes ao monitoramento de ativos da rede elétrica e envio de quantidade de dados superiores às limitações das redes utilizando a estratégia de fragmentar os dados. Com base nesses resultados, a especificação funcional do *middleware* foi estabelecida:

- Arquitetura modular para que possa ser reutilizado em cada nova implementação;
- *API* com suporte a multilinguagem;
- Disponibilidade de diversos protocolos utilizados em sistemas de medição;
- Comunicação básica, transmissão de dados bidirecional, confiabilidade, segurança e escalabilidade obtidas com as redes *LPWAN*, especificamente *LoRaWAN*;
- Controle dos recursos do sistema, manipulação de dados e escalonamento de tarefas;
- Suporte a diversas plataformas de prototipação;
- Executar exclusivamente nos dispositivos embarcados (*end-node*).

A arquitetura do *middleware SGAM* funcionará sobre a rede *LoRaWAN*, desta forma, utilizará a topologia de rede definida para este protocolo. Conforme a infraestrutura da rede *LoRaWAN* apresentado na subseção 2.2.2 (Figura 4). São especificados quatro elementos: nó final (dispositivo embarcado), *gateway*, servidor de rede e o servidor de aplicação. Atendendo o requisito do *middleware SGAM* rodar exclusivamente nos dispositivos

finais, as seções seguintes caracterizam o *middleware* neste cenário.

#### 4.1.1 Características do Nó Final

Uma solução de Internet das Coisas é caracterizada por muitos dispositivos (nós finais ou de borda) que podem usar algum tipo de *gateway* para se comunicar por meio de uma rede com um servidor corporativo que está executando uma plataforma *IoT* na nuvem. As funções dos dispositivos, *gateways* e plataforma de nuvem são bem definidas e cada uma delas fornece recursos e funcionalidades específicas nas soluções de Internet das Coisas.

O dispositivo embarcado ou nó final é o ponto de partida para uma solução de Internet das Coisas, sendo tipicamente o originador dos dados devido sua interação com o mundo físico. Algumas aplicações, visando a uma redução nos custos ou no uso de energia, por exemplo, utilizam dispositivos (microcontroladores) com muito poucos recursos de memória e desempenho.

A Figura 6 traz um detalhamento da hierarquia de *software* e *hardware* do nó final, visando a uma melhor compreensão da interação do *middleware SGAM* com as demais camadas.

O nó é dividido em quatro camadas:

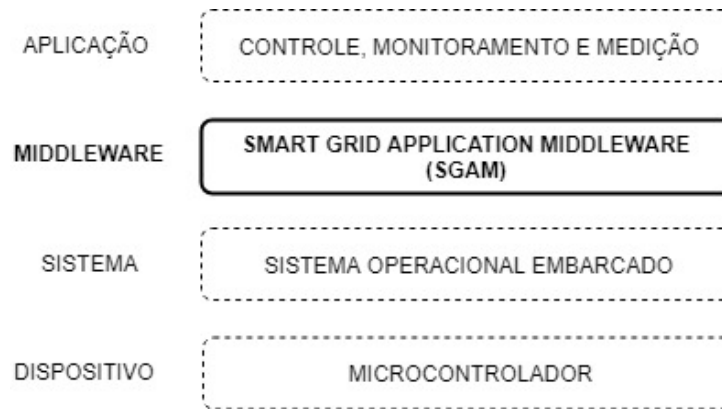
- **Aplicação:** Responsável pela combinação completa de funções e funcionalidades que descrevem bem a regra de negócio de uma aplicação útil ao usuário final;
- **Middleware:** Responsável por disponibilizar uma interface para servir de base para a programação da aplicação do usuário final;
- **Sistema:** O sistema operacional embarcado utilizado para gerenciar os recursos do microcontrolador e seus dispositivos periféricos internos ou externos;
- **Dispositivo:** A camada que descreve os *hardwares* utilizados, podendo englobar recursos e interfaces do próprio microcontrolador ou externas a ele, bem como sensores e atuadores.

A divisão da Figura 6 pretende enfatizar a independência das quatro camadas em relação à criação da aplicação embarcada. Assim, o desenvolvedor cria sua solução com base na *API* disponibilizada pelo *middleware*, o que permitirá que o código da aplicação seja reutilizado em outras plataformas de *hardware* e *software*.

#### 4.1.2 Smart Grid Application Middleware (SGAM)

O *middleware SGAM* foi concebido em quatro módulos principais, oferecendo desde uma *API* padrão multilinguagens até sua integração com a plataforma de *software* utilizada. A arquitetura conceitual do *middleware* é apresentada na Figura 7. Esse modelo

Figura 6 – Divisão do nó em quatro camadas.

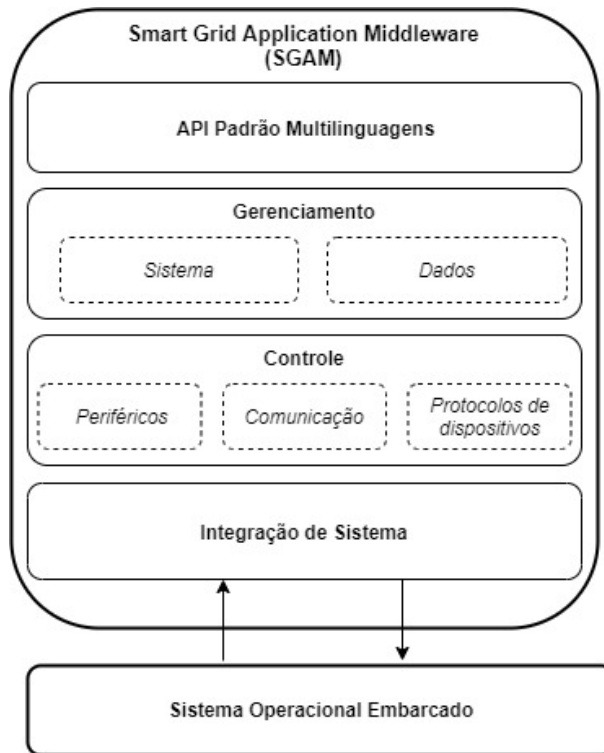


Fonte: Próprio Autor.

conceitual busca ilustrar os aspectos mais significativos do domínio do problema no mundo real. Cada módulo encapsula os detalhes de implementação, ajudando a lidar com a complexidade, diminuindo o acoplamento e, por fim, disponibilizando interfaces claras e bastante coesas para o sistema. Esta elevação no nível de abstração auxilia na redução do tempo de desenvolvimento de aplicações e simplifica o trabalho do projetista em duas tarefas importantes: (1) Reusar componentes de *hardware* e *software* desenvolvidos em projetos anteriores e (2) explorar as diversas possibilidades de implementação visando ganhos em previsibilidade, desempenho, tamanho etc.

A seguir, cada módulo do *middleware SGAM* é apresentado.

- **API Padrão Multilinguagens:** É o módulo responsável por oferecer as funcionalidades de controle, monitoramento e medição dos dispositivos conectados à rede elétrica. Essencialmente será implementada em C/C++, no entanto, também suportará as linguagens *Java*, *Python* e *JavaScript*.
- **Gerenciamento:** Esse módulo engloba um conjunto de serviços e tarefas que permitem gerenciar os recursos do próprio sistema embarcado (nó final da rede) e os dados trafegados. O submódulo sistema oferece funcionalidades de configuração dos dispositivos. No submódulo dados ocorre a conversão de tipos, além de agregação/desagregação de dados em lotes.
- **Controle:** Esse módulo controla o acesso a sensores, atuadores, tipos de comunicação e protocolos. O submódulo periférico descreve um modelo de sensor ou atuador genérico. O submódulo comunicação disponibiliza uma interface geral de conectividade com o servidor. Já o submódulo protocolos de dispositivos deve prover a descrição de diversos tipos de protocolos para interagir com os dispositivos da rede elétrica, por exemplo, o protocolo *DLMS/COSEM* (*Device Language Message Specification/Companion Specification for Energy Metering*),

Figura 7 – Arquitetura do *Smart Grid Application Middleware*.

Fonte: Próprio Autor.

que interage com medidores inteligentes.

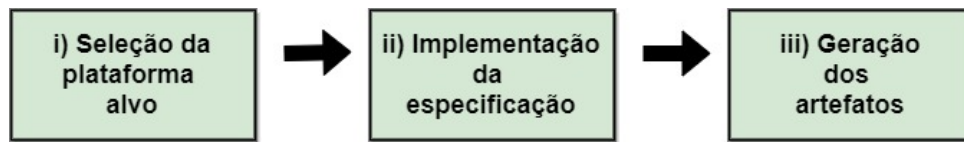
- **Integração de Sistema:** É o módulo responsável pela integração entre o *middleware* e o sistema operacional *IoT* da plataforma de *hardware* utilizada. Este módulo responde pela implementação das camadas superiores (controle, gerenciamento e *API*) e pela "tradução" entre as funcionalidades disponibilizadas pelo *middleware* e o sistema operacional embarcado selecionado, conforme mostrado na figura 7.

#### 4.1.3 Escolha e integração com a plataforma *ARM Mbed*

Com a implementação da especificação do *middleware SGAM* gera-se artefatos (bibliotecas) que, juntamente com a aplicação embarcada, rodam na plataforma de *software/hardware* selecionada. A Figura 8 ilustra o processo de definição e integração com a plataforma de *software*. No primeiro momento (*i*) é definida qual plataforma de *software* será utilizada, por exemplo, *Mbed OS*. O próximo passo, do processo de integração, (*ii*) ocorre pela correta implementação das classes e interfaces que descrevem as funcionalidades presentes nos módulos. As classes e interfaces devem ser corretamente implementadas para garantir a total compatibilidade entre as *API's* do *middleware* e do sistema alvo. Por fim, ao implementá-las adequadamente (*iii*) geram-se artefatos (bibliotecas) que se

tornam específicos para o sistema operacional utilizado. Com o *middleware* gerado suas funcionalidades ficam disponíveis para auxiliar no desenvolvimento de novas soluções embarcadas.

Figura 8 – Processo de definição e integração com a plataforma alvo.



Fonte: Próprio Autor.

Conforme apresentado na subseção 2.3.1, a plataforma *ARM Mbed IoT* fornece um sistema operacional embarcado de código aberto (*ARM mbed OS*). Esse *S.O.* possui um grande número de bibliotecas padrão projetadas para desenvolver aplicações *IoT*, incluindo segurança, conectividade e *drivers* para periféricos e sensores de E/S. Além disso, esse sistema é suportado por um grande número de plataformas de *hardware* de vários fabricantes com processadores que variam de *Cortex-M0* a *Cortex-M7*. Portanto, a camada de sistema (Figura 6) será implementada usando o *ARM Mbed OS*.

Como mencionado nos trabalhos de Zhu (2017) e Hossein et al. (2016), as arquiteturas *ARM* são predominantes em dispositivos móveis e sistemas embarcados. Além disso, mais de 30 empresas fornecedoras de placas e componentes para dispositivos embarcados dão suporte ao *ARM Mbed OS*, tais como: *STMicroelectronics*<sup>1</sup>, *MultiTech*<sup>2</sup>, *NXP*<sup>3</sup>, *Semtech*<sup>4</sup> e outras (ARM Mbed, 2019b). Essa ampla base favorece a criação de aplicações de Internet das Coisas e auxilia a integração com outras soluções com diferentes componentes de *hardwares* e *software*.

## 4.2 Implementação do *Middleware*

A proposta inicial de arquitetura do *middleware SGAM* é abrangente. Neste trabalho, parte da especificação apresentada foi selecionada para a implementação funcional da arquitetura, viabilizando a avaliação experimental do *middleware* proposto. (i) A *API* multilinguagem do *middleware* oferece suporte a quatro linguagens de programação, no entanto, apenas a linguagem C/C++ foi elegida; (ii) A camada de gerenciamento controla funcionalidades ligadas ao próprio sistema e manipulação dos dados. Parte do submódulo dados estará disponível; (iii) A camada controle é responsável por suportar periféricos(sensores, atuadores), comunicação e protocolos de dispositivos da rede elétrica,

<sup>1</sup> [https://www.st.com/content/st\\_com/en.html](https://www.st.com/content/st_com/en.html)

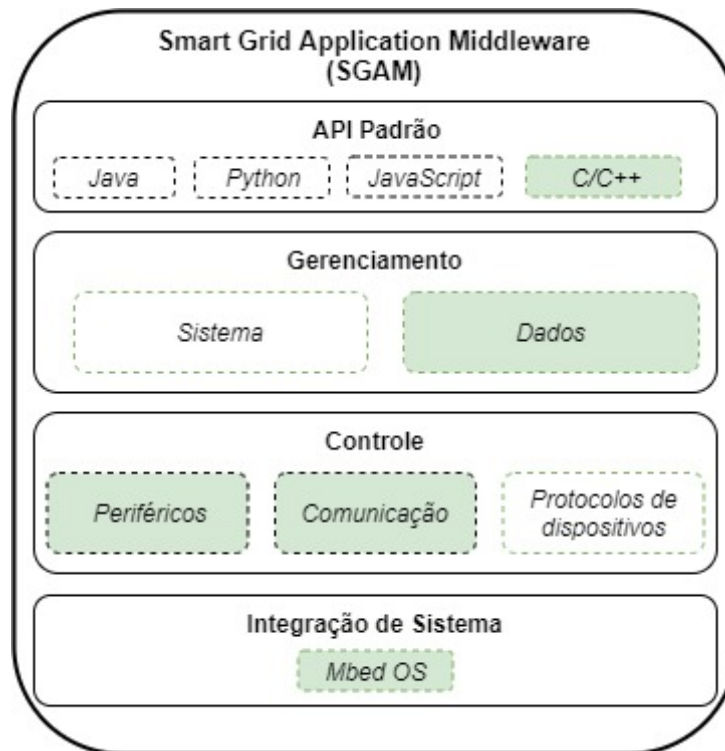
<sup>2</sup> <https://www.multitech.com/>

<sup>3</sup> <https://www.nxp.com/>

<sup>4</sup> <https://www.semtech.com/>

optou-se pelos submódulos periféricos (especificamente sensores) e comunicação; (iv) A camada de integração busca garantir o pleno funcionamento entre o *middleware* e o sistema operacional embarcado utilizado. Esta camada será implementada especificamente para o sistema operacional embarcado *Mbed OS*. A Figura 9 ilustra essa delimitação de escopo na implementação.

Figura 9 – Delimitação de escopo na implementação.



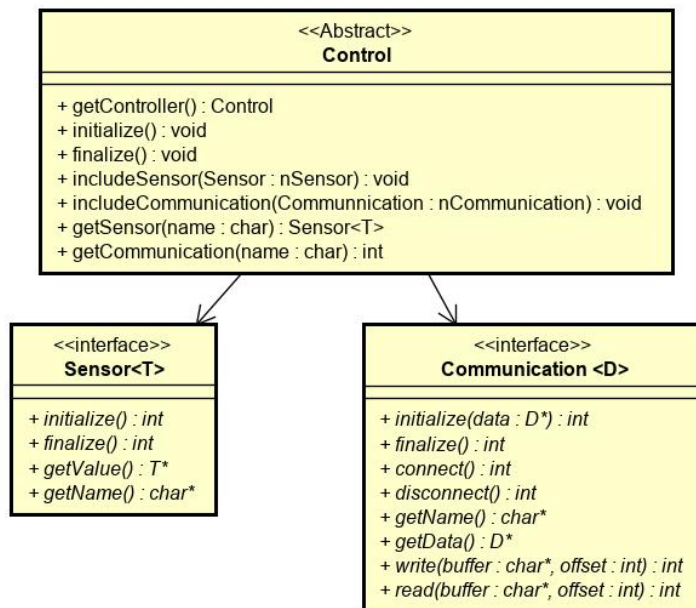
Fonte: Próprio Autor.

A modelagem dos módulos foi realizada a fim de indicar uma representação das principais entidades do *middleware* e seus relacionamentos. O diagrama de classe e a notação da UML<sup>5</sup> 2.0 (*Unified Modeling Language*) foram utilizados para essa representação. Para a geração do código-fonte foi utilizada a linguagem C++, utilizando-se dos conceitos *Object-Oriented (OO)*, *templates* e *maps*. A abordagem de reutilização de *software* baseou-se na estratégia *top-down*, aplicando *design patterns* para sistemas embarcados (DOUGLASS, 2010), como *Factory Method* e *Singleton*.

#### 4.2.1 Módulo de Controle

O módulo de controle possui a classe abstrata *control* e uma descrição genérica de um tipo *sensor* e um tipo *communication*. Como já mencionado, o submódulo protocolo não será implementado. A Figura 10, ilustra o diagrama de classes do módulo Controle. As classes exibidas são *Control*, *Sensor* e *Communication*.

<sup>5</sup> <https://www.uml.org/>

Figura 10 – Diagrama de classe - *Control*.

Fonte: Próprio Autor.

- *Control*: é uma classe abstrata responsável por controlar a inclusão e utilização de novos sensores e tipo da comunicação. A classe *Control* possui sete operadores que devem ser implementados:
  - *getController*: retorna uma instância única ( *Singleton*) da classe *control*;
  - *initialize*: responsável por inicializar os objetos da classe *control*;
  - *finalize*: operador responsável por destruir os objetos da classe *control*;
  - *includeSensor*: inclui a definição de novos tipos de sensores;
  - *includeCommunication*: inclui a definição de novos tipos de comunicação;
  - *getSensor*: obtém a referência do objeto sensor criado;
  - *getCommunication*: obtém a referência do objeto comunicação criado;
- *Sensor*: é uma *interface* que descreve um sensor genérico. A especialização dessa classe gera a descrição de diferentes tipos de sensores; por exemplo, sensor de giroscópio, sensor de temperatura, sensor de posicionamento global (GPS), etc. Quatro operações devem ser implementadas para definir o comportamento do sensor utilizado.
  - *initialize*: operador responsável por inicializar os objetos da classe sensor especializada;
  - *finalize*: operador responsável por destruir os objetos da classe sensor especializada;

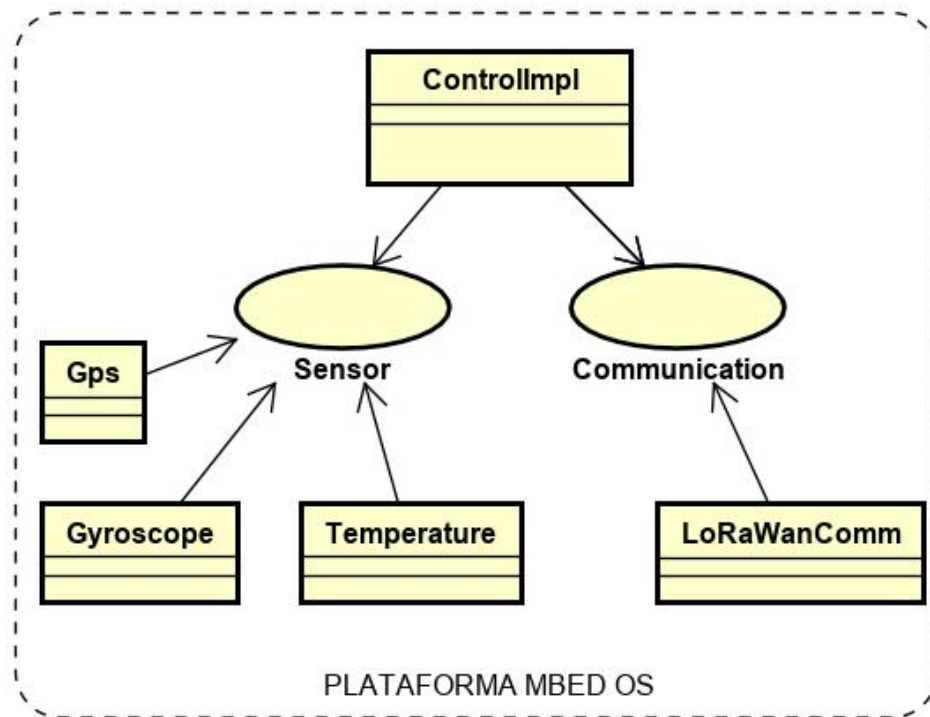
- *getValue*: retorna os dados da classe especializada, ou seja, os dados do sensor selecionado. O tipo  $T^*$  descreve o modelo de dados utilizado;
  - *getName*: retorna o nome do sensor que foi especializado, por exemplo, um sensor *gyroscope*;
- *Communication*: é uma *interface* que descreve uma comunicação genérica. Aponta especificamente para o servidor de dados da aplicação *IoT*. A especialização dessa classe gera a definição de um ou mais tipos de comunicação; por exemplo, comunicação *LoRaWAN*. Oito operadores devem ser implementados para definir o comportamento da comunicação utilizada:
    - *initialize*: operador responsável por receber os objetos de configuração da comunicação e inicializá-los corretamente. O tipo  $D^*$  encapsula os parâmetros de inicialização;
    - *finalize*: operador responsável por destruir os objetos da classe *communication* que foi especializada;
    - *connect*: realiza o procedimento de conexão com a rede da comunicação definida;
    - *disconnect*: responsável por desconectar da rede do tipo de comunicação definida;
    - *getName*: retorna o nome da comunicação que foi especializada, por exemplo, comunicação *LoRaWAN*;
    - *getData*: retorna o objeto de configuração da comunicação. O tipo  $D^*$  descreve o objeto de configuração de comunicação;
    - *write*: escreve/envia os dados trafegados na comunicação;
    - *read*: recebe/ler os dados trafegados na comunicação;

#### 4.2.2 Módulo de Integração

O módulo de integração é a implementação das classes/interfaces do módulo de controle. Desta forma, a implementação da *Control* gera a classe *ControlImpl*, a *sensor* gera as classes *GPS*, *Gyroscope* e *Temperature* e a *communication* gera a classe *LoRaWanComm*. A Figura 11, ilustra as classes geradas a partir do módulo de controle.

Após o primeiro passo de integração com a especialização dessas classes, o próximo passo ocorre no nível do sistema operacional embarcado utilizado. Neste caso, a integração é definida com o *Mbed OS*. Assim, o *middleware SGAM* se utiliza da *API* e *Drivers* do sistema operacional embarcado para ter acesso a recursos como interfaces *UART*, *SPI*, *I2C*, *GPIO* e outras. A seguir, são exibidos trechos de códigos utilizados na implementação do *middleware SGAM*.

A implementação da classe *Control* é realizada na classe *ControlImpl*. Conforme é

Figura 11 – Classes geradas para a plataforma *Mbed OS*.

Fonte: Próprio Autor.

observado na Figura 12, linhas 12 a 14, três tipos de sensores são incluídos para o módulo de controle do *middleware SGAM*, o sensor de temperatura, o sensor de giroscópio e o sensor *GPS*, com suas respectivas interfaces de comunicação ( *A1-Analog IO*; *I2C*; *serial*). A inclusão do tipo de comunicação *LoRaWAN* ocorre na linha 17. Após a inclusão dos sensores e da comunicação, estes objetos estão disponíveis no módulo de controle. Nas linhas 21 a 23, é exibido o método que retorna a referência para o objeto **control** que será utilizado pela aplicação para acessar os recursos do *middleware*.

Na Figura 13, observa-se a definição de um tipo de sensor, o giroscópio. A definição dos tipos *GPS* e *Temperature* ocorrem de forma similar. A classe *Gyroscope* implementa os operadores da interface *Sensor*. Conforme é observado na linha 19, a interação do sensor com a placa de prototipação ocorre pela interface *I2C*. O tipo *GyroscopeData*, linha 25, descreve o modelo de dados que ele deve retornar, no caso, são inteiros que possuem três eixos: *gx*, *gy* e *gz*. Na linha 29, verifica-se a obtenção da referência *mpu* para o *driver* do sensor de giroscópio. E na linha 30, a declaração do método *getMotion* usado pela classe para obter os valores do sensor.

A implementação da interface *Communication* ocorre de forma similar a interface *Sensor*. Na definição da classe *LoRaWanComm*, conforme é observado na Figura 14, ocorre à declaração dos operadores da interface *Communication*. Destaca-se, na linha

Figura 12 – Trecho de código dos operadores de inclusão e obtenção do *controller*.

```

9  ControlImpl::ControlImpl() {
10
11     // add Sensor
12     includeSensor( (Sensor<void*>*) new Temperature(A1) );
13     includeSensor( (Sensor<void*>*) new Gyroscope(i2c) );
14     includeSensor( (Sensor<void*>*) new Gps(serial) );
15
16     // add Communication
17     includeCommunication( (Communication<void*>*) new
18         LoRaWanComm("") );
19
20     // returns single instance
21     Control* ControlImpl::getController() {
22         return this;
23     }

```

Fonte: Próprio Autor.

Figura 13 – Trecho do código da definição da classe *Gyroscope*.

```

17  class Gyroscope: public Sensor<GyroscopeData> {
18  public:
19      Gyroscope(I2C &i2c);
20      virtual ~Gyroscope();
21
22      virtual int initialize();
23      virtual int finalize();
24
25      virtual GyroscopeData* getValue();
26      virtual const char* getName();
27
28  private:
29      MPU6050 mpu;
30      void getMotion(GyroscopeData* data);
31  };

```

Fonte: Próprio Autor.

61, o tipo *LoraData*, que possui o modelo de dados necessário para a inicialização do tipo de comunicação *LoRaWAN*. Essa classe de tipo de dados encapsula os parâmetros de inicialização; por exemplo, o tipo de comunicação utilizada: via rádio *LoRa*, o tipo de conexão: *Over-The-Air Activation (OTAA)* ou *Activation By Personalization (ABP)*, seus parâmetros conforme a conexão, porta de leitura e escrita. O *driver* utilizado para interagir com o rádio do sistema, definido na linha 67, é o *LoRaWANInterface* que fornece uma *API C++* para se conectar a Internet através de uma rede *LoRaWAN*.

O *driver* de comunicação *LoRaWANInterface* é baseado em eventos, auxiliando na estratégia de mitigar bloqueios na execução da aplicação final que utiliza o *middleware*. Por exemplo, ao tentar realizar conexão ou enviar/receber dados pela rede. A Figura 15 ilustra alguns eventos indicativos do *driver*. Os eventos *CONNECTED* e *DISCONNECTED* são indicativos em relação à conexão de rede, mostram quando foi realizada conexão ou quando

Figura 14 – Trecho do código da definição da classe LoRaWanComm.

```

49 class LoRaWanComm: public Communication<LoraData> {
50 public:
51     LoRaWanComm(const char* parameters);
52     virtual ~LoRaWanComm();
53
54     virtual int initialize(LoraData* data);
55     virtual int finalize();
56
57     virtual int connect();
58     virtual int disconnect();
59
60     virtual const char* getName();
61     virtual LoraData* getData();
62
63     virtual int write(const unsigned char* buffer
64                       , int offset);
65     virtual int read(unsigned char* buffer, int
66                     offset);
67
68 private:
69     LoRaWANInterface* lorawan;
70     LoraData* data;
71 };

```

Fonte: Próprio Autor.

foi desconectado, respectivamente. Já o evento *JOIN\_FAILURE* indica falha de conexão (tentativa de *join* na rede *LoRaWAN*); O evento *TX\_DONE* ocorre quando o *middleware* envia dados com sucesso e o evento *RX\_DONE* ocorre quando há recebimento de dados com êxito.

Figura 15 – Trecho do código da API baseada em eventos.

```

3 static void lora_event_handler(lorawan_event_t event) {
4     switch (event) {
5         case CONNECTED:
6             // do something ...
7             break;
8         case DISCONNECTED:
9             // do something ...
10            break;
11        case RX_DONE:
12            // do something ...
13            break;
14        case TX_DONE:
15            // do something ...
16            break;
17        case JOIN_FAILURE:
18            // do something ...
19            break;
20        ...
21    }
22 }
23

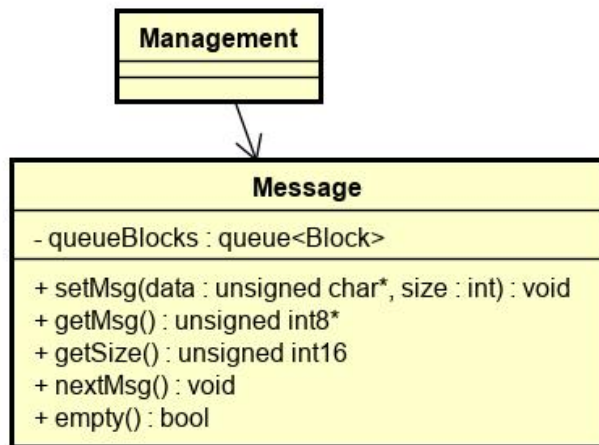
```

Fonte: Próprio Autor.

### 4.2.3 Módulo de Gerenciamento

O módulo de gerenciamento, especificamente o submódulo dados, possui a classe concreta *Message*. Essa classe possui um conjunto de funções que auxiliam o projetista da aplicação final enviar grandes quantidades de dados pela rede *LoRaWAN*. Isso é possível pela estratégia de fragmentar esses dados (mensagens) e agrupá-los em quantidades menores (blocos), identificando cada bloco de maneira que possam ser reagrupados corretamente na aplicação presente no servidor. A Figura 16, ilustra o diagrama de classes do módulo de gerenciamento. A classe *Message* é composta pelo atributo *queueBlocks* e cinco operadores:

Figura 16 – Diagrama de Classe - *Management*.



Fonte: Próprio Autor.

- *queueBlocks*: esse atributo é do tipo *Queue* (fila), ou seja, uma estrutura de dados baseado em *FIFO* (*Fist IN Fist Out*). Essa estrutura é responsável por armazenar os blocos de dados menores que foram gerados no processo de fragmentação.
- *setMsg*: responsável por armazenar os dados em blocos;
- *getMsg*: recupera o bloco de dados atual;
- *getSize*: retorna o tamanho do bloco de dados atual;
- *nextMsg*: recupera o próximo bloco de dados existente na estrutura de dados;
- *empty*: indica se a fila está vazia;

Para utilizar a Classe *Message* é necessário primeiro declarar um objeto do tipo *Message*. Esse objeto deve ter como parâmetro obrigatório o tamanho do *payload*, ou seja, o tamanho do bloco de dados que será transmitido em cada envio pela rede. Devido limitações da própria rede *LoRaWAN*, esse valor pode chegar ao máximo a 256 *bytes* de dados. O próximo passo é utilizar a função *setMsg* para armazenar o *buffer* de dados na estrutura. Com os métodos *getMsg* e *getSize* é possível recuperar o conteúdo e tamanho existente em cada bloco. Para realizar os próximos envios de dados devem-se recuperar

os demais blocos existentes na fila com a função *nextMsg*. Por fim, verifica-se se todos os blocos de dados da fila foram recuperados utilizando o operador *empty*;

De forma particular o método *setMsg* define internamente dois tipos de blocos. O tipo *Single Block* e o tipo *Composite Block*. O tipo *single* é gerado quando os dados a serem enviados pela rede possui um tamanho menor que o tamanho definido para um bloco. Assim um envio é suficiente, não precisando que os dados sejam fragmentados. A Tabela 2 apresenta a estrutura (cabeçalho) do bloco *single*.

Tabela 2 – *Single Block*

| type [t] | dataSize [ds] | data     |
|----------|---------------|----------|
| 1 byte   | 1 byte        | 62 bytes |

- *Single Block*: Ocorre quando a quantidade de dados de envio é menor que o tamanho definido para o bloco;
  - *type[t]*: Indica o tipo de bloco *single* ("s")/ único;
  - *dataSize[ds]*: Indica o tamanho da quantidade de dados;
  - *data*: Armazena os dados propriamente ditos.

Na Figura 17 exibe a declaração do *payload* e exemplo de envio de 10 *bytes* de dados. Na linha 3, o objeto *myPayload(64)* do tipo *Message* é definido e inicializado com valor inteiro 64 *bytes*, indicando o tamanho do bloco padrão de envio de dados. Nesta ilustração, pretende-se enviar 10 *bytes* de dados pela rede, desta forma, como o número de *bytes* a serem enviados é menor que o tamanho do bloco definido, a estrutura *single* é criada internamente. Na linha 4, observam-se os dados em formato hexadecimal, nos quais dois *bytes* são referentes ao cabeçalho e os 10 *bytes* seguintes são referentes aos dados, totalizando 12 *bytes*.

Figura 17 – Declaração do *payload* e envio de 10 *bytes*.

```

2 //Ex: Envio de 10 bytes de dados em um bloco
3 Message myPayload(64); //Declaração
4 myPayload: 73 0A 31 32 33 34 35 36 37 38 39 30 //12 bytes;
```

Fonte: Próprio Autor.

Na Figura 18, a estrutura *single* criada no envio de 10 *bytes* é exibida em formato *.json* auxiliando a visualização de cada campo apresentado na Tabela 2.

Figura 18 – *Single block* - Exibição do *Payload* em formato *json*.

```
{
  "myPayload": {
    "data": "1234567890",
    "dataSize": 10,
    "type": "s"
  }
}
```

Fonte: Próprio Autor.

Quando o tamanho dos dados para serem enviados pela rede é maior do que o tamanho de um bloco (ex: *mypayload(64)*), o *middleware* gera o tipo de bloco *composite*, ou seja, os dados são estruturados com um cabeçalho maior e são transmitidos em dois ou mais envios. O número de blocos gerados e adicionados na fila é definido pela divisão entre o tamanho dos dados para serem enviados pelo tamanho do bloco definido pelo usuário. Todo esse processo ocorre de forma automática e transparente a aplicação. A Tabela 3 apresenta a estrutura (cabeçalho) do bloco *composite*.

Tabela 3 – *Composite Block*

| type [t] | id     | tBlock [tb] | iBlock [ib] | dataSize [ds] | data     |
|----------|--------|-------------|-------------|---------------|----------|
| 1 byte   | 1 byte | 1 byte      | 1 byte      | 1 byte        | 59 bytes |

- *Composite Block*: Ocorre quando a quantidade de dados de envio é maior que o tamanho definido para o bloco;
  - *type[t]*: Indica o tipo de bloco *composite* ("c")/ composto;
  - *id*: Identificador de mensagem. Todos os blocos de uma mensagem tem o mesmo identificador;
  - *tBlock[tb]*: Número total de blocos de uma mensagem;
  - *iBlock[ib]*: Índice do bloco de uma mensagem;
  - *dataSize[ds]*: Indica o tamanho da quantidade de dados;
  - *data*: Armazena os dados propriamente ditos.

Figura 19 – Declaração do *Payload* e envio de 100 bytes.

```
15 //Ex: Envio de 100 bytes de dados em dois blocos
16 Message myPayload(64); //Declaração
17 myPayload: 63 7B 02 01 3B 31 32 33 34 35 36 37
  (...) 36 37 38 39 //64 bytes;
```

Fonte: Próprio Autor.

A Figura 19 exibe um exemplo de declaração do *payload* e exemplo de envio de 100 *bytes* de dados. Na linha 16, o objeto *myPayload(64)* do tipo *Message* é definido e inicializado com valor inteiro 64, indicando o tamanho do bloco padrão de envio de dados. Nesta ilustração, pretende-se enviar 100 *bytes* de dados pela rede, desta forma, como o número de *bytes* a serem enviados é maior que o tamanho do bloco definido (apenas 64 *bytes*), a estrutura *composite* é criada internamente e os dados (juntamente com os cabeçalhos) serão enviados em duas tentativas. Na linha 17, observam-se os dados em formato hexadecimal, nos quais os primeiros cinco *bytes* são referentes ao cabeçalho e os 59 *bytes* seguintes são referentes aos dados, totalizando o limite estabelecido, neste exemplo, de 64 *bytes* por envio. Na Figura 20 são exibidos os campos da estrutura *composite* em formato *.json* conforme descrito na Tabela 3. Dois blocos são gerados, em ambos, os campos *type*, *id*, *tBlock* se repetem em cada bloco. Já os campos *iBlock*, *dataSize* e *data* são diferentes por indicarem, respectivamente o índice, o total de dados e os dados enviados de cada bloco.

Figura 20 – *Composite block* - Exibição do *Payload* em formato *json*.

```
//Bloco 01/02
{
  "myPayload": {
    "data": "12345678901234567890123456789012345678901234567890123456789",
    "dataSize": 59,
    "iBlocks": 1,
    "tBlocks": 2,
    "id": 123,
    "type": "c"
  }
}

//Bloco 02/02
{
  "myPayload": {
    "data": "01234567890123456789012345678901234567890",
    "dataSize": 41,
    "iBlocks": 2,
    "tBlocks": 2,
    "id": 123,
    "type": "c"
  }
}
```

Fonte: Próprio Autor.

## 4.3 Conclusão

Uma visão geral do *Smart Grids Application Middleware (SGAM)* foi apresentada neste capítulo. Inicialmente através de uma modelagem conceitual, seus requisitos, arquitetura e funcionalidades foram discutidos. Também foi abordada sua integração com a plataforma *Mbed*. O desenho arquitetural do *middleware* foi representado por diagramas

de classes, detalhando funcionalidades e relacionamentos visando auxiliar numa maior compreensão da arquitetura proposta.

Com a implementação das classes *ControlImpl*, *LoRaWanComm*, *Gyroscope*, *GPS*, *Temperature* e *Message* foi possível gerar uma versão funcional do *middleware SGAM* para a plataforma de *software Mbed OS*.

## 5 Casos de Uso do Middleware

Este capítulo apresenta casos de uso que foram implementados para mostrar exemplos de utilização do *Middleware SGAM* e auxiliar na avaliação da arquitetura proposta.

Quatro seções compõem este capítulo: *(i)* Inicialmente são apresentados os recursos de *hardware*, *software* e requisitos estabelecidos nos experimentos; *(ii)* em seguida, um caso de uso de monitoramento de ativos da rede elétrica é detalhado; *(iii)* segue-se com a descrição do caso de uso de fragmentação de dados; *(iv)* e na última parte, os resultados obtidos são comparados e discutidos.

### 5.1 Recursos utilizados

A portabilidade das aplicações de monitoramento de ativos e fragmentador de dados para diferentes plataformas de *hardwares* foi verificada através da utilização de três placas controladoras: *STM32L152RE*, *STM32F411RE* e *STM32F429ZI*. Esse suporte para compilar em diferentes plataformas de *hardware* foi obtido pelo emprego do sistema operacional embarcado *Mbed OS*. A versão do *Mbed* utilizado é a *Mbed OS 5*. O processo de compilação e geração das aplicações ocorreu na plataforma *online*<sup>1</sup>, disponibilizada pela *ARM*, utilizando o compilador profissional *ARMCC*<sup>2</sup>.

Destaca-se que se optou pela placa *STM32F429ZI* para a realização dos experimentos e obtenção das métricas deste capítulo. Esta placa possui um processador *ARM 32bits Cortex-M4* com frequência de 180MHz, 2048 KB de memória *Flash* e 256KB *SRAM*.

Para a comunicação com o servidor é usado um transceptor LoRA da *Semtech SX1272 Low Power Long Range*. Os sensores utilizados são: temperatura (*Grove*), giroscópio (*GY-87 MPU 6050*) e de posicionamento global (*GPS ublox NEO6m*). Quanto aos recursos de rede, foi utilizado um *Gateway LoRa RISINGHF 2S008*, configurado no plano de banda Australiana 915MHz, utilizando os canais de 8 a 15 e adicionado ao servidor de rede *LoRaWAN The Things Network (TTN)*. A Tabela 4, resume os recursos de *hardware* e rede utilizados.

Dois tipos de memórias, *Flash* e *RAM*, foram verificados quanto aos recursos computacionais consumidos. No compilador, cada uma dessas memórias possui modelo próprio para uma região de alocação chamado seção. Para código executável a seção é chamada de "*Code*" e as constantes e outros dados de somente leitura são colocados em uma

<sup>1</sup> <https://ide.mbed.com/compiler/>

<sup>2</sup> <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0472k/index.html>

Tabela 4 – Recursos utilizados

| Dispositivos           | Parâmetro/Modelo | Características                                            |
|------------------------|------------------|------------------------------------------------------------|
| Plataformas Embarcadas | STM32L152RE      | ARM 32 bits Cortex-M3, 32 MHz, Flash 512 kb, SRAM 80 kb    |
|                        | STM32F411RE      | ARM 32 bits Cortex-M4, 100 MHz, Flash 512 kb, SRAM 128 kb  |
|                        | STM32F429ZI      | ARM 32 bits Cortex-M4, 180 MHz, Flash 2048 kb, SRAM 256 kb |
| LoRa                   | Rádio            | Semtech SX1272                                             |
|                        | Gateway          | RISINGHF 2S008                                             |
|                        | Servidor de Rede | The Things Network (TTN)                                   |
| Sensores               | Temperatura      | Seed Groove                                                |
|                        | Giroscópio       | Gy-MPU 6050                                                |
|                        | GPS              | GPS ublox Neo6m                                            |

seção chamada "*RO Data*" (*Read-Only*). Essas seções são armazenadas na memória *Flash* do dispositivo. Enquanto que variáveis estáticas e globais inicializadas entram em uma seção chamada "*RW Data*" (*Read-Write*) e as que não são inicializadas pertencem a uma seção chamada "*ZI Data*" (*Zero Initialise*). Essas últimas seções são armazenadas em memória *RAM*. Vale observar que a construção das aplicações para dispositivos embarcados gera pacotes (ou binários) que englobam todos os artefatos envolvidos na solução, por exemplo, aplicação, sistema operacional e qualquer outra camada de *software*. Desta forma, toda métrica obtida nos casos de uso em relação ao consumo de memória, se referem ao código da aplicação acrescido pelos custos do *middleware* e os recursos trazidos pelo sistema operacional embarcado.

Quanto às métricas de tempo consumido na execução do *middleware*, o objeto *Timer*<sup>3</sup> foi utilizado para verificar o tempo de retorno de cada método, em microssegundos. Esse objeto é uma interface de medição precisa disponibilizada pelo próprio *Mbed OS*.

Outro aspecto que merece destaque é que todos os testes de medições de tempo realizados neste capítulo foram repetidos trinta vezes, apresentando as médias e o desvio padrão dos valores obtidos.

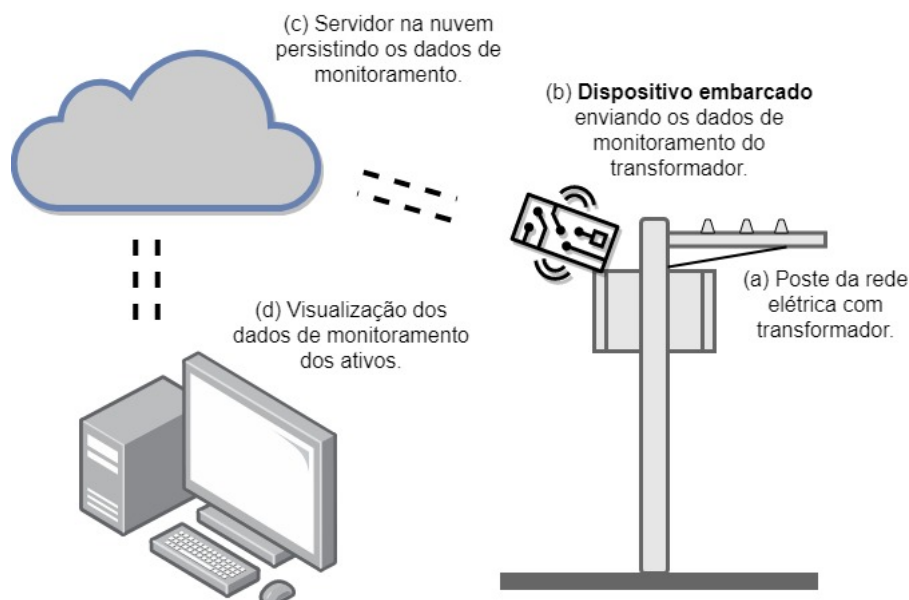
## 5.2 Monitoramento Remoto de Ativos

A Figura 21 ilustra um exemplo de sistema de monitoramento remoto de ativos (equipamentos) de uma rede elétrica utilizando um dispositivo embarcado. No dispositivo é executada uma aplicação que obtêm dados dos sensores e os envia para a nuvem, alimentando um sistema de diagnóstico dos ativos. Com este caso de uso é possível

<sup>3</sup> <https://os.mbed.com/docs/mbed-os/v5.14/apis/timer.html>

demonstrar como o *middleware* pode auxiliar na construção de soluções de monitoramento desses equipamentos.

Figura 21 – Exemplo de sistema de monitoramento utilizando dispositivo embarcado.



Fonte: Próprio Autor.

Neste caso de uso o *middleware* provê acesso a sensores que monitoram grandezas físicas, as quais indicam o estado operacional de ativos em uma rede elétrica, como transformadores e isoladores. Exemplos dessas grandezas são: temperatura, vibração e corrente elétrica. Com esses dados uma aplicação no servidor pode fazer diagnóstico desses ativos ou permitir levantamentos estatísticos. No caso de equipamentos que podem ser removidos, a aplicação pode detectar e alarmar deslocamentos não programados, utilizando giroscópio e/ou *GPS*. Neste exemplo, é possível validar a camada controle do *middleware*, especificamente o submódulo periférico devido à adição e interação com novos sensores conectados ao sistema e o submódulo comunicação pelo tratamento bidirecional de dados feita na rede *LoRaWAN*.

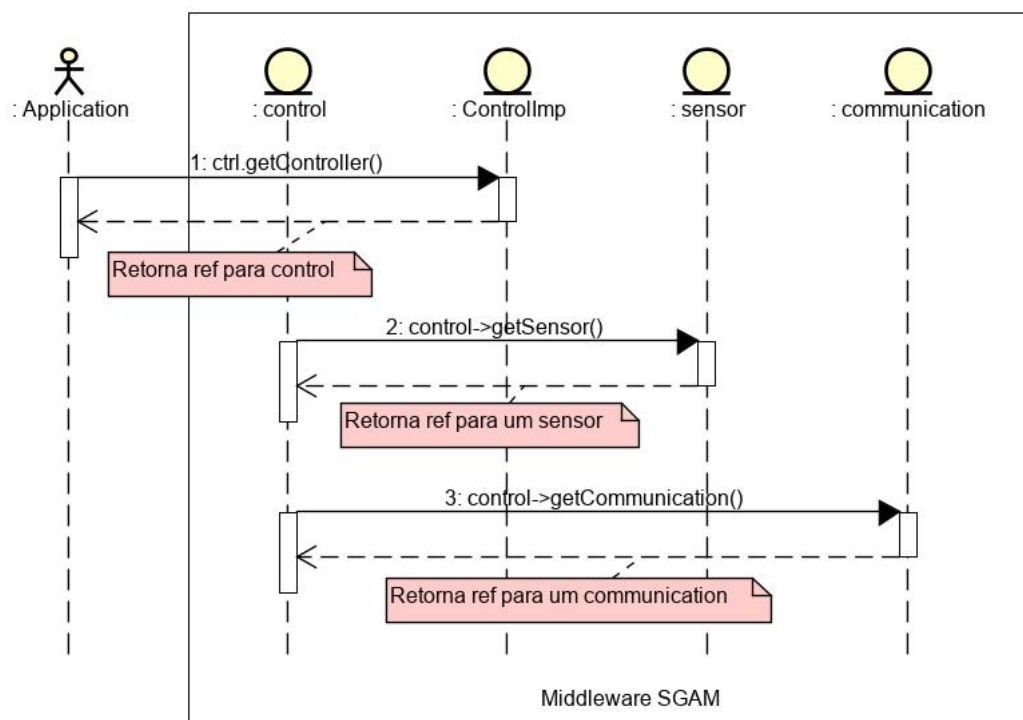
### 5.2.1 Ocupação de memória da aplicação de monitoramento de ativos

As métricas de ocupação de memória *Flash* e *RAM* obtidas na geração da aplicação de monitoramento remoto de ativos são exibidas na Tabela 5. A geração da aplicação foi direcionada para a placa *STM32F429ZI*. Na memória *Flash*, a seção de código (*Code*) consumiu 112,9 kb (90,17%) e a seção de constantes (*RO Data*) consumiu 12,3 kb (9,83%). No qual se chega ao consumo total de 125,2 kb (100%) de memória *Flash*. Quanto à memória *RAM*, as variáveis inicializadas (*RW Data*) da aplicação ocuparam 0,4 kb (1,32%) e as variáveis não inicializadas (*ZI Data*) consumiram 30,1 kb (98,68%). Totalizando uma ocupação de 30,5 kb (100%) de *RAM*.

Tabela 5 – Ocupação de memória na geração da aplicação de monitoramento de ativos

| Placa       | Memória      | Seção          | Tamanho         | %           |
|-------------|--------------|----------------|-----------------|-------------|
| STM32F429ZI | Flash        | <i>Code</i>    | 112,9 kb        | 90,17%      |
|             |              | <i>RO Data</i> | 12,3 kb         | 9,83%       |
|             | <b>TOTAL</b> | -              | <b>125,2 kb</b> | <b>100%</b> |
|             | RAM          | <i>RW Data</i> | 0,4 kb          | 1,32%       |
|             |              | <i>ZI Data</i> | 30,1 kb         | 98,68%      |
|             | <b>TOTAL</b> | -              | <b>30,5 kb</b>  | <b>100%</b> |

### 5.2.2 Inicialização do módulo control

Figura 22 – Diagrama de sequência de obtenção de referência para os objetos *control*, *sensor* e *communication*.

Fonte: Próprio Autor.

O módulo *control* é responsável por prover acesso aos submódulos *sensor* e *communication*. A Figura 22 exibe o diagrama de sequência de inicialização do módulo *control* e os submódulos *sensor* e *communication*. (1) O método *getController()* retorna a referência para o objeto *control*. (2) Com a referência *control* é utilizado o método *getSensor()*, que retorna o objeto do sensor solicitado, por exemplo, *getSensor(gps)* retorna a referência do sensor definido como *gps*. (3) De forma similar, o objeto *control* também possui o método *getCommunication()* que devolve um objeto para o tipo de comunicação implementado

no *middleware*. Neste trabalho, a comunicação implementada é invocada pela chamada *getCommunication(LoRaWAN)*.

Na Figura 23, um trecho do código da aplicação de monitoramento remoto de ativos utilizando o módulo *control* do *middleware* é apresentado. Nas linhas 02 e 03, observa-se a criação das variáveis globais *ctrl* (do tipo *ControlImpl*) e *control* (do tipo *Control*). Na linha quatro a variável *comm* do tipo *communication* é utilizado em rotinas de comunicação posteriores. Na linha 8, obtém-se uma referência do tipo *control* utilizando o método *getController()*. Na linha 9, a módulo controle do *middleware* é inicializado.

Figura 23 – Trecho do código da aplicação embarcada - rotina de inicialização da classe controle.

```

2   ControlImpl ctrl;
3   Control* control = NULL;
4   Communication<LoraData>* comm = NULL;
5
6   int main() {
7
8       control = ctrl.getController();
9       control->initialize();

```

Fonte: Próprio Autor.

A Tabela 6 apresenta o tempo gasto na execução da rotina de inicialização do módulo de controle. Cada função foi verificada individualmente, o método *getController()* teve um tempo médio de resposta de  $2,067\mu s$  e apresentou um desvio padrão de  $0,249\mu s$ . É observado um tempo de resposta rápido nesta função, pois se trata apenas de retornar a referência da classe *control* criada na inicialização da aplicação. Na função *initialize()* o valor de retorno médio foi de  $30230,100\mu s$  (aprox 30 milissegundos) com o valor  $187,163\mu s$  de desvio padrão. É verificado no método *initialize()* um tempo de retorno maior devido às rotinas de inicialização dos sensores e comunicação.

Tabela 6 – Tempo consumido (em microssegundos) do módulo *control*.

| Rotina                          | Função                 | Média( $\mu s$ ) | Desvio Padrão( $\mu s$ ) |
|---------------------------------|------------------------|------------------|--------------------------|
| Inicialização<br>módulo control | <i>getController()</i> | 2,067            | 0,249                    |
|                                 | <i>Initialize()</i>    | 30230,100        | 187,163                  |

### 5.2.3 Utilização dos sensores

A Figura 24 exibe um trecho do código da rotina de leitura de sensores. Na linha 58 é declarada a variável *myPayload* do tipo *CayenneLPP*<sup>4</sup>. Esse tipo é uma biblioteca auxiliar utilizada para reduzir o tamanho do *payload* trafegado na rede *LoRaWAN*. Na linha 60 é obtida a referência (*temp*) para um sensor do tipo temperatura. Na linha 61, a função *getValue()* recupera a grandeza física indicadora da temperatura. Na linha 62, o valor de temperatura é adicionado a variável *myPayload*.

Figura 24 – Trecho do código da aplicação embarcada - classe sensor.

```

56 static void monitoring_module() {
57
58     CayenneLPP myPayload(10);
59
60     Sensor<void*>* temp = control->getSensor("Temperature");
61     float* valor = (float*)temp->getValue();
62     myPayload.addTemperature(1, *valor);

```

Fonte: Próprio Autor.

A Tabela 7 apresenta o tempo de execução da rotina de obtenção de referência e utilização das funções da classe sensor. Além do sensor de temperatura, foram obtidos os tempos gastos na utilização dos sensores *gps* e giroscópio. As funções de obtenção de referência (*getSensor()*) retornaram valores bastante similares para os sensores de *gps* ( $5,933\mu s/0,249\mu s$ ), temperatura ( $6,956\mu s/0,128\mu s$ ) e giroscópio ( $6,136\mu s/0,105\mu s$ ), tanto no tempo médio como na diferença entre as chamadas (desvio padrão). Já na resposta de cada sensor (*getValue()*), houve diferenças significativas. O tempo médio obtido com o retorno do resultado do *gps* foi de  $599339,438\mu s$  (aprox 600 milissegundos), com o sensor de temperatura obteve  $17,030\mu s$  (aprox 0,017 milissegundos) e o sensor de giroscópio foi de  $1593,067\mu s$  (aprox 1,6 milissegundos). O *gps* teve o tempo mais alto de retorno, isso se deve, além do *delay* do próprio sensor, o *driver* de comunicação utilizado que possui tratamento na *string* de retorno. Nos testes foram observados que alguns campos da estrutura da resposta viam vazios ou inválidos, necessitando assim, adequado tratamento.

Tabela 7 – Tempo consumido (em microssegundos) dos sensores.

| Rotina                                           | Função                        | Média( $\mu s$ ) | Desvio Padrão( $\mu s$ ) |
|--------------------------------------------------|-------------------------------|------------------|--------------------------|
| Obtenção da referência e utilização dos sensores | <i>getSensor(GPS)</i>         | 5,933            | 0,249                    |
|                                                  | <i>gps-&gt;getValue()</i>     | 599339,438       | 116,114                  |
|                                                  | <i>getSensor(Temperature)</i> | 6,956            | 0,128                    |
|                                                  | <i>tmp-&gt;getValue()</i>     | 17,030           | 5,385                    |
|                                                  | <i>getSensor(Gyroscope)</i>   | 6,136            | 0,105                    |
|                                                  | <i>gyro-&gt;getValue()</i>    | 1593,067         | 0,249                    |

<sup>4</sup> <https://developers.mydevices.com/cayenne/docs/lora/lora-cayenne-low-power-payload>

### 5.2.4 Utilização da comunicação

A Figura 25 exibe um trecho de código de obtenção da referência do objeto de comunicação e estabelecimento de conexão com a rede. Na linha 16, a função *getCommunication()*, retorna uma comunicação do tipo *LoRaWAN*. Nas linhas 18 e 19, define-se o modo de ativação *Over-the-Air Activation (OTAA)*. Na linha 21, é criada a variável *data*, na qual recebe os parâmetros de conexão, rádio (*semtech sx1272*) e eventos do *mbed*. É preparado o *callback* (linha 22) para tratar os eventos do *Mbed*, na linha 23, o *callback* é adicionado na variável *data*. Na linha 24, a porta onde o *mbed* realizará as leituras e escritas é configurada. Nas linhas 27 a 30, a variável *data* é passada como parâmetro para a função *initialize()*, inicializando a classe comunicação com as configurações definidas. Na linha 32, é solicitada conexão a rede. No modo *OTAA*, a aplicação que está no dispositivo embarcado inicia o procedimento de requisição de conexão (*join request*) com o servidor de rede, depois que o servidor recebe a solicitação, valida os parâmetros de conexão, trata da performance da rede, estabelece as chaves de segurança e finalmente envia uma mensagem de aceite de conexão (*join accept*) ao dispositivo final, caso tudo ocorra com sucesso. Nas linhas 33 a 39, o retorno do método *connect()* é tratado. A linha 41 indica ao *mbed* para despachar sempre os eventos.

Figura 25 – Trecho do código da aplicação embarcada - rotina de conexão.

```

16      comm = (Communication<LoraData*>)control->getCommunication("LoRAWAN");
17
18      lorawan_connect_t connect_params;
19      connect_params.connect_type = LORAWAN_CONNECTION_OTAA;
20
21      LoraData* data = new LoraData(&connect_params, (LoRaRadio*)&radio, &ev_queue);
22      callbacks.events = mbed::callback(lora_event_handler);
23      data->prepareCallBack(callbacks);
24
25      data->lora_port = MBED_CONF_LORA_APP_PORT;
26
27      if( comm->initialize(data) != TRUE) {
28          D_LOG("Inicialização falhou !!\r\n");
29          return -1;
30      }
31
32      lorawan_status_t retcode = (lorawan_status_t) comm->connect();
33      if (retcode == LORAWAN_STATUS_OK ||
34          retcode == LORAWAN_STATUS_CONNECT_IN_PROGRESS) {
35          D_LOG("Connection - In Progress ...\r\n");
36      } else {
37          D_LOG("Connection error, code = %d \r\n", retcode);
38          return -1;
39      }
40
41      ev_queue.dispatch_forever();

```

Fonte: Próprio Autor.

A Figura 26 exibe um trecho do código de envio de dados pela rede. Na linha 59, a referência das configurações de comunicação é recuperada. Na linha 60, adiciona-se uma *flag* para indicar que a mensagem não precisa de confirmação pela rede. Nas linhas 62 a

66, a rotina de escrita é apresentada. Dois parâmetros são exigidos, o conteúdo para ser enviado e seu tamanho. A linha 68 adiciona essa rotina de envio na fila de eventos. Com o parâmetro *tx\_interval* é estabelecido o intervalo de tempo de repetição.

Figura 26 – Trecho do código da aplicação embarcada - rotina de escrita.

```

59     LoraData* data = (LoraData*) comm->getData();
60     data->read_write_flags = MSG_UNCONFIRMED_FLAG;
61
62     int16_t retcode = comm->write(payload.getBuffer(), payload.getSize());
63
64     if (retcode < 0) {
65         return;
66     }
67
68     ev_queue.call_in(tx_interval, monitoring_module);

```

Fonte: Próprio Autor.

A Figura 27 exibe um trecho de código de leitura de dados. Na linha 74, a referência das configurações de comunicação é recuperada. Na linha 75, *flags* para indicar que a leitura pode ser de mensagem com/sem confirmação pela rede são adicionadas. Nas linhas 77 a 82, a rotina de leitura é apresentada. Inicialmente, uma variável para armazenar os dados lidos é criada. Na linha 78, o método *read()* recupera os dados que chegaram pela rede. Na linha 79 a 82, uma rotina para tratar o retorno é mostrada. No contexto da aplicação de monitoramento de ativos, essa mensagem de resposta pode ser utilizada como comando para ativar uma *thread* com o sensor de movimento. Caso haja alguma perturbação nos valores do sensor, este deve iniciar o *GPS* e enviar suas coordenadas de posicionamento.

Figura 27 – Trecho do código da aplicação embarcada - rotina de leitura.

```

72     static void downlink_analyzer()
73     {
74         LoraData* data = (LoraData*) comm->getData();
75         data->read_write_flags = MSG_CONFIRMED_FLAG|MSG_UNCONFIRMED_FLAG;
76
77         uint8_t rx_buffer[50] = { 0 };
78         int16_t retcode = comm->read( rx_buffer, sizeof(rx_buffer) );
79         if (retcode < 0) {
80             D_LOG("Error code %d \r\n", retcode);
81             return;
82         }
83     }

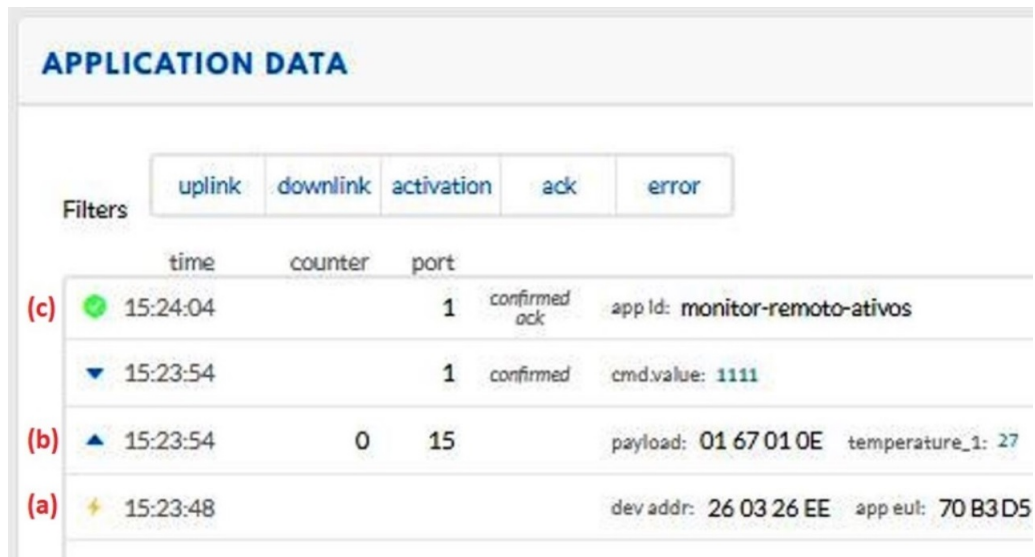
```

Fonte: Próprio Autor.

Para realizar a conexão da aplicação de monitoramento remoto de ativos à rede *LoRaWAN* é necessário adicionar ao servidor de rede (*TTN*) as configurações do *gateway* e do dispositivo. A Figura 28, apresenta a tela de dados da aplicação no *TheThingsNetwork* (*TTN*). A coluna *time* mostra o horário que as mensagens chegam ao servidor. A coluna

*counter* é um contador que indica a quantidade de mensagens. A coluna *port* indica por qual porta está ocorrendo a leitura e a escrita dos dados. Nas últimas colunas apresentam *labels* que descrevem informações, como, valores das chaves, dados e *ids* das aplicações embarcadas que estão vinculadas e comunicando com o *TTN*. Analisando as linhas, na Figura 28(a), a mensagem de confirmação de estabelecimento de conexão com a rede *LoRaWAN* é mostrada. Após a confirmação de conexão, é possível enviar dados ao servidor. Na Figura 28(b), observa-se a chegada do *payload* no servidor de rede (*UPLINK*) na porta 15, com os dados de temperatura no valor de 27°C. Também é verificada, Figura 28(c), a confirmação de leitura da mensagem enviada pelo servidor de rede (*cmd.value:1111*) para o dispositivo embarcado (*DOWNLINK*).

Figura 28 – Tela de dados da aplicação no *TTN* (Servidor).



| APPLICATION DATA |          |                                                                                                                                                                                                 |      |                  |                                         |
|------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------|------------------|-----------------------------------------|
| Filters          |          | <input type="button" value="uplink"/> <input type="button" value="downlink"/> <input type="button" value="activation"/> <input type="button" value="ack"/> <input type="button" value="error"/> |      |                  |                                         |
|                  | time     | counter                                                                                                                                                                                         | port |                  |                                         |
| (c)              | 15:24:04 |                                                                                                                                                                                                 | 1    | confirmed<br>ack | app id: monitor-remoto-ativos           |
|                  | 15:23:54 |                                                                                                                                                                                                 | 1    | confirmed        | cmd.value: 1111                         |
| (b)              | 15:23:54 | 0                                                                                                                                                                                               | 15   |                  | payload: 01 67 01 0E temperature_1: 27  |
| (a)              | 15:23:48 |                                                                                                                                                                                                 |      |                  | dev addr: 26 03 26 EE app eui: 70 B3 D5 |

Fonte: Próprio Autor.

A Tabela 8 apresenta o tempo de execução da rotina de obtenção de referência e utilização das funções da classe comunicação. O método *getCommunication()* apresenta um tempo médio de resposta de 5,967 $\mu$ s e um desvio padrão de 0,249 $\mu$ s. Com a referência da classe comunicação, as demais funcionalidades da classe podem ser utilizadas. O método *initialize()* lida com as configurações de conexão, rádio e eventos do *mbed*, obteve tempo médio de retorno de 139979,203 $\mu$ s (aprox 140 milissegundos) e 23,368 $\mu$ s de desvio. A solicitação de conexão com a rede realizada pela função *connect()* obteve tempo médio de retorno de 32488,683 $\mu$ s (aprox 32 milissegundos) e desvio de 8,321 $\mu$ s. Vale destacar que esse tempo de retorno é apenas da solicitação de conexão. Caso a conexão ocorra com sucesso, um evento é disparado permitindo a aplicação embarcada tratar com as demais rotinas, por exemplo, leitura e escrita de dados na rede.

O método *getData()* é chamado sempre antes das rotinas de escrita e leitura para configurar o objeto de conexão, como se trata apenas de retornar a referência desse objeto,

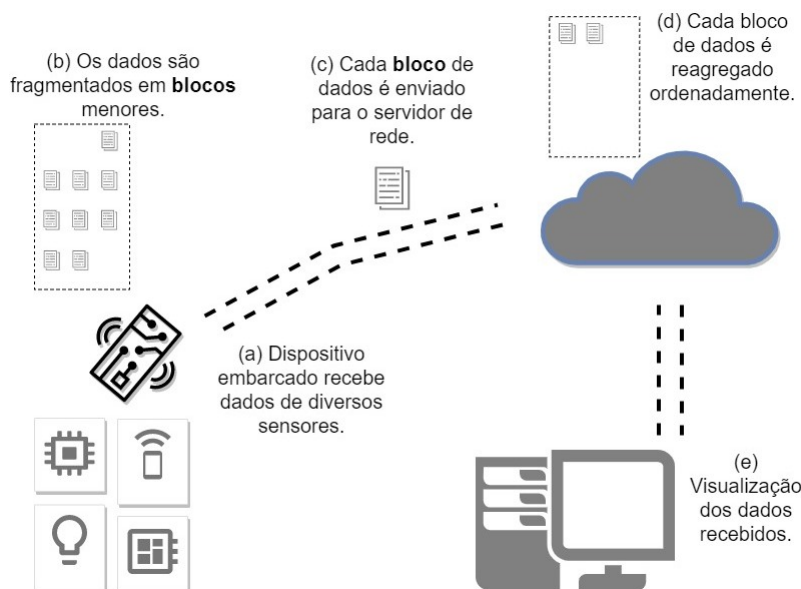
o tempo médio de retorno do método é de  $2,033\mu s$  com desvio de  $0,180\mu s$ . É observado que a rotina de escrita *write()* consome mais tempo que a rotina de leitura *read()* na aplicação. O menor tempo para a rotina de leitura ocorre porque a recepção do *payload(downlink)* é um evento assíncrono tratado pelo *middleware*. Bastando recuperar o ponteiro com os dados já armazenado no dispositivo embarcado. Desta forma, a rotina de escrita apresenta o valor médio de  $1292,721\mu s$  (aprox 1,3 milissegundos) com desvio de  $29,934\mu s$  e o tempo médio de leitura de  $6,003\mu s$  (aprox 0,006 milissegundos) e desvio padrão de  $0,009\mu s$ .

Tabela 8 – Tempo consumido (em microssegundos) na comunicação.

| Rotina                                             | Função                       | Média( $\mu s$ ) | Desvio Padrão( $\mu s$ ) |
|----------------------------------------------------|------------------------------|------------------|--------------------------|
| Obtenção da referência e utilização da comunicação | <i>getCommunication(...)</i> | 5,967            | 0,249                    |
|                                                    | <i>initialize(...)</i>       | 139979,203       | 23,368                   |
|                                                    | <i>connect()</i>             | 32488,683        | 8,321                    |
|                                                    | <i>getData()</i>             | 2,033            | 0,180                    |
|                                                    | <i>write(...)</i>            | 1292,721         | 29,934                   |
|                                                    | <i>read(...)</i>             | 6,003            | 0,009                    |

### 5.3 Fragmentador de Dados

Figura 29 – Exemplo de fragmentação de dados utilizando dispositivo embarcado.



Fonte: Próprio Autor.

Neste caso de uso a aplicação criada utilizará o *middleware* para acessar leituras de diversos sensores ou algum equipamento que retorne uma massa significativa de dados, por exemplo, um medidor inteligente (*Smart Meter*). Com o acesso a esses dados, a aplicação, utilizando o *middleware*, os enviará sobre as limitações (ex: 64 *bytes*) da rede *LoRa WAN*.

A estratégia utilizada para o envio de dados sobre as limitações da rede *LoRaWAN* é fragmentar a mensagem (dados) em pacotes menores e identificáveis para que a mensagem possa ser recuperada no servidor. O caso de uso fragmentador de dados é ilustrado na Figura 29.

Neste caso de uso é possível validar a camada de gerenciamento e controle do *middleware*. Na camada de gerenciamento, o submódulo dados controlará o *payload* trafegado para não ocorrer perdas. E na camada controle, o submódulo comunicação permitirá trafegar esses dados pelas redes *LoRaWAN*.

### 5.3.1 Ocupação de memória da aplicação de fragmentação de dados

As métricas de ocupação de memória *Flash* e *RAM* obtidas na geração da aplicação de fragmentação de dados são exibidas na Tabela 9. A geração da aplicação foi para a plataforma de *hardware STM32F429ZI*. Na memória *Flash*, a seção de código (*Code*) consumiu 113,3 kb (87,76%) e a seção de constantes (*RO Data*) consumiu 15,8 kb (12,24%). Gerando uma ocupação total de 129,1 kb (100%) de memória *Flash*. Quanto à memória *RAM*, as variáveis inicializadas (*RW Data*) da aplicação ocuparam 0,4 kb (1,16%) e as variáveis não inicializadas (*ZI Data*) consumiram 34,2 kb (98,84%). Totalizando uma ocupação de 34,6 kb (100%) de memória *RAM*.

Tabela 9 – Ocupação de memória na geração da aplicação de fragmentação de dados

| Placa       | Memória      | Seção          | Tamanho         | %           |
|-------------|--------------|----------------|-----------------|-------------|
| STM32F429ZI | Flash        | <i>Code</i>    | 113,3 kb        | 87,76%      |
|             |              | <i>RO Data</i> | 15,8 kb         | 12,24%      |
|             | <b>TOTAL</b> | -              | <b>129,1 kb</b> | <b>100%</b> |
|             | RAM          | <i>RW Data</i> | 0,4 kb          | 1,16%       |
|             |              | <i>ZI Data</i> | 34,2 kb         | 98,84%      |
|             | <b>TOTAL</b> | -              | <b>34,6 kb</b>  | <b>100%</b> |

### 5.3.2 Rotina de fragmentação

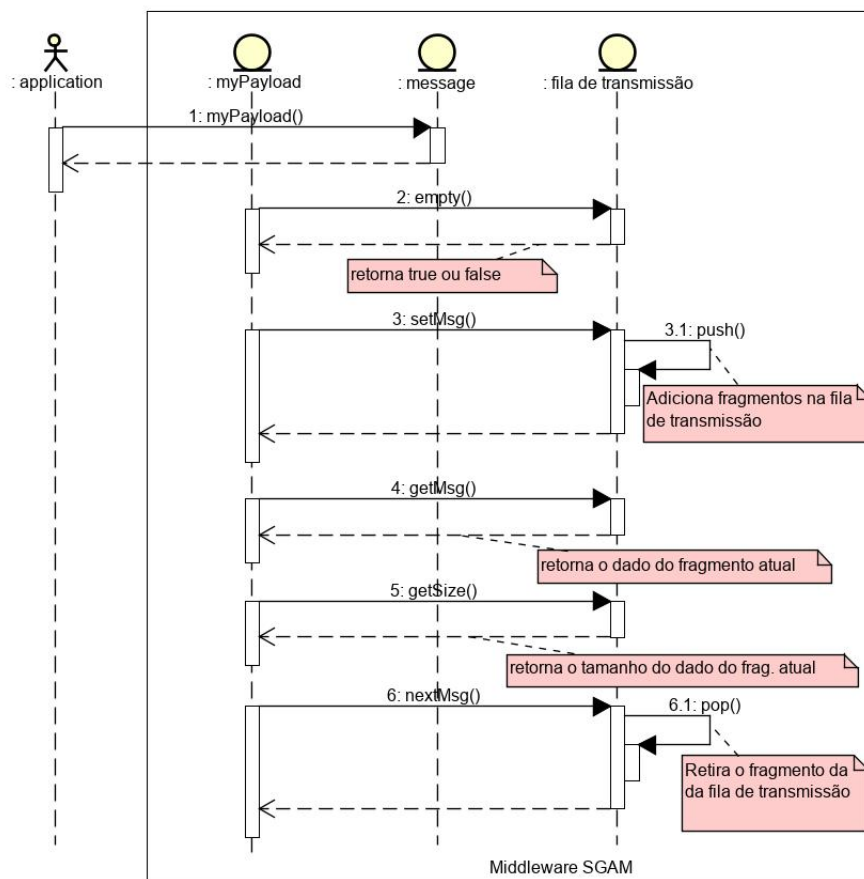
Para utilizar a funcionalidade de fragmentação de dados disponibilizada pelo *middleware* deve-se utilizar a classe *Message*. Esta classe disponibiliza cinco funções para a rotina de fragmentação: *empty()*, *setMsg()*, *getData()*, *getSize()* e *nextMsg()*. Além disso, possui uma estrutura de dados baseada em fila (*Fist IN Fist Out*), na qual é responsável por armazenar os blocos de dados menores que foram gerados no processo de fragmentação. Daqui em diante, os termos estrutura de dados e blocos são substituídos por fila de transmissão e fragmentos, respectivamente. Segue a formalização da nova terminologia:

- **Fila de transmissão:** estrutura de dados baseados em fila;

- **Fragmentos:** blocos de dados menores gerados no processo de fragmentação.

A Figura 30 mostra o diagrama de sequência ilustrando a rotina de fragmentação. (1) A aplicação cria o objeto *myPayload* da classe *message*. Com esse objeto é possível utilizar as demais funcionalidades. (2) O método *empty()* verifica se a fila de transmissão está vazia, retornando *true* ou *false*. (3) O método *setMsg()* é utilizado para fragmentar os dados passados nos parâmetros. Internamente adiciona (*push()*) os fragmentos na fila de transmissão. (4) O método *getMsg()* retorna os dados armazenados no começo da fila de transmissão. (5) O método *getSize()* retorna o tamanho dos dados armazenados no começo da fila de transmissão. (6) O método *nextMsg()* retira (*pop()*) o primeiro fragmento da fila de transmissão, tornando o próximo fragmento como atual (ou primeiro).

Figura 30 – Diagrama de sequência da rotina de fragmentação.



Fonte: Próprio Autor.

Na Figura 31, um trecho de código da aplicação embarcada com a rotina de fragmentação de dados é exibido. Na linha 94, um *array* de *bytes* (inteiros sem sinal de 8 *bits*) com 4001 posições é criado. Na linha seguinte, a variável *myPayload* com o tamanho de 64 *bytes* do tipo *message* é definida. Esse valor de 64 indica o tamanho do fragmento (em *bytes*) que é escrito a cada solicitação de envio para o servidor sobre as limitações da

rede *LoRaWAN*. Nas linhas 99 e 100, duas variáveis auxiliares são criadas: a *tx\_data\_size* para guardar o tamanho total dos dados a serem enviados e *retcode* para controlar o valor de retorno das funções. Nas linhas 102 e 103, o objeto de conexão (*data*) é recuperado e a *flag* de leitura para mensagens que não precisam de confirmação é definida.

Figura 31 – Trecho de código da aplicação embarcada - fragmentador de dados.

```
94  uint8_t tx_data[4001] = {0};
95  Message myPayload(64);
96
97  static void lorawan_send_data() {
98
99      uint16_t tx_data_size;
100     uint8_t retcode;
101
102     LoraData* data = (LoraData*) comm->getData();
103     data->read_write_flags = MSG_UNCONFIRMED_FLAG;
104
105     if (myPayload.empty()) {
106         myPayload.setMsg(tx_data, tx_data_size);
107     }
108
109     retcode = comm->write(myPayload.getMsg(), myPayload.getSize());
110
111     if (retcode < 0) {
112         D_LOG("Error code %d \r\n", retcode);
113         return;
114     }
115
116     myPayload.nextMsg();
117     ev_queue.call_in(tx_interval, lorawan_application);
118 }
```

Fonte: Próprio Autor.

Na linha 105, o método *empty()* é utilizado para verificar se a fila de transmissão da classe *message* está vazia. Caso a fila esteja vazia, a função *setMsg()* é chamada na linha 106. Essa função recebe dois parâmetros, o *tx\_data* que é o total de dados para serem enviados e o *tx\_data\_size* referente ao tamanho desse total de dados. Internamente, o método *setMsg()* fragmenta os dados (*tx\_data[4001]*) de acordo com o tamanho definido para o fragmento (neste exemplo, *myPayload(64)*). Assim, o algoritmo divide o tamanho total dos dados pelo tamanho do *myPayload*, e verifica a parte decimal, gerando um número inteiro que representa o total de fragmentos da fila de transmissão da classe *message*. Neste caso, 4001 *bytes* de dados dividido por 64 *bytes* gera o número inteiro, arredondado para cima, de 63 fragmentos. Essa quantidade representa exatamente a quantidade de fragmentos (*myPayload*) que serão enviados pela rede. Na linha 109, ocorre a rotina de escrita de dados na rede, já apresentada no exemplo anterior, utilizando o método *write()*. O operador *getMsg()* recupera os dados e o método *getSize()* retorna o tamanho, ambos do fragmento atual da fila de transmissão da classe *message*. Entre as linhas 111 e 114 é verificado o retorno da rotina de escrita de dados na rede. Na linha 116, o método *nextMsg()*, atualiza a fila de transmissão da classe *message*. O fragmento atual da fila é retirado, tornando o próximo elemento como fragmento atual. Assim, quando forem

utilizados os métodos *getMsg()* e *getSize()*, os valores retornados serão do novo fragmento que se encontra como primeiro da fila de transmissão. Por fim, na linha 117, a aplicação principal (*lorawan\_application*) é adicionada na fila de eventos do *Mbed*.

O tempo de execução dos métodos da classe *message* é apresentado na Tabela 10. Para a realização dessa métrica foi considerado o seguinte cenário: O total de dados para serem enviados pela rede é de 4001 *bytes*, e o tamanho do *myPayload* (em cada envio) é de 64 *bytes*. O método *empty()* obteve um tempo médio de resposta de 2,100  $\mu$ s e um desvio de 0,300  $\mu$ s. A rotina de fragmentação dos dados com a função *setMsg()* consumiu em média 275,933  $\mu$ s com um desvio de 11,150  $\mu$ s. As funções *getMsg()* e *getSize()*, obtiveram, respectivamente, os valores médios de 7,033  $\mu$ s e 4,100  $\mu$ s com desvios de 1,180  $\mu$ s e 0,966  $\mu$ s. O método *nextMsg()* obteve como tempo médio de resposta o valor de 2,333  $\mu$ s e desvio padrão de 0,471  $\mu$ s.

Tabela 10 – Tempo consumido (em microssegundos) na fragmentação.

| Rotina       | Função             | Média( $\mu$ s) | Desvio Padrão( $\mu$ s) |
|--------------|--------------------|-----------------|-------------------------|
| Fragmentação | <i>empty()</i>     | 2,100           | 0,300                   |
|              | <i>setMsg(...)</i> | 275,933         | 11,150                  |
|              | <i>getMsg()</i>    | 7,033           | 1,180                   |
|              | <i>getSize()</i>   | 4,100           | 0,966                   |
|              | <i>nextMsg()</i>   | 2,333           | 0,471                   |

De modo geral, na Tabela 10, as métricas obtidas apresentam valores abaixo de 7,1  $\mu$ s para o tempo de retorno médio e abaixo de 1,2  $\mu$ s para os valores de desvio padrão. Porém, merece destaque os valores do método *setMsg()* por exibir as medidas de tempo bem acima dos demais métodos. Esse tempo elevado é esperado, tendo em vista, que o método possui mais rotinas internas, como de identificação de tipo de fragmento (*single* ou *composite*), fragmentação dos dados e adição na fila de transmissão. No entanto, vale investigar como o tamanho dos dados, o tamanho dos fragmentos e a quantidade dos fragmentos influenciam no tempo de resposta desse método.

### 5.3.3 Investigação do método *setMsg()*

Inicialmente, um arranjo com diferentes configurações para a quantidade de fragmentos e o tamanho desses fragmentos é elaborado de acordo com a Tabela 11. No lado esquerdo da tabela (nas linhas), diferentes quantidades (2, 4, 8, 16, 32, 64 e 128) de fragmentos são definidas. Na parte superior da tabela (colunas), três diferentes tamanhos (64, 128 e 256 *bytes*) para o fragmento são apresentados. Com esse arranjo foi possível obter diferentes tempos médios de retornos e desvios padrões do método *setMsg()*. Além disso, o tamanho dos dados utilizados para obter essas métricas de tempo é destacado nos

parênteses. Cada rotina de teste ocorreu fixando um tamanho para o fragmento (ex: 64 *bytes*) e executando trinta vezes o método *setMsg()*, obteve-se a média e o desvio padrão do tempo de resposta.

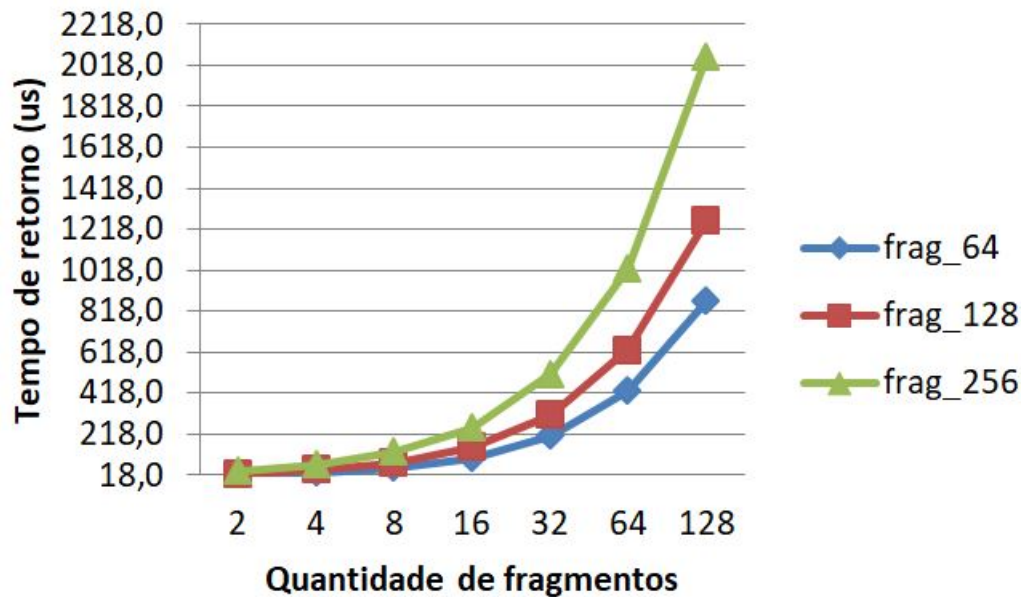
Tabela 11 – Tempo consumido (em microssegundos) na execução do método *setMsg* em diferentes configurações.

|                          |     | Tamanho dos fragmentos                        |                                               |                                               |
|--------------------------|-----|-----------------------------------------------|-----------------------------------------------|-----------------------------------------------|
|                          |     | 64                                            | 128                                           | 256                                           |
|                          |     | <i>Média(μs)/Desvio(μs)</i>                   | <i>Média(μs)/Desvio(μs)</i>                   | <i>Média(μs)/Desvio(μs)</i>                   |
| Quantidade de fragmentos | 002 | 18,100 ± 7,059<br>( 128 bytes )               | 24,067 ± 7,113<br>( 256 bytes )               | <b>36,200 ± 6,935</b><br>( <b>512 bytes</b> ) |
|                          | 004 | 28,700 ± 0,690<br>( 256 bytes )               | <b>41,067 ± 0,680</b><br>( <b>512 bytes</b> ) | 66,000 ± 0,816<br>( 1024 bytes )              |
|                          | 008 | <b>53,067 ± 0,680</b><br>( <b>512 bytes</b> ) | 77,967 ± 0,706<br>( 1024 bytes )              | 127,833 ± 0,745<br>( 2048 bytes )             |
|                          | 016 | 102,000 ± 0,577<br>( 1024 bytes )             | 151,800 ± 0,702<br>( 2048 bytes )             | 251,333 ± 0,745<br>( 4096 bytes )             |
|                          | 032 | 211,433 ± 3,896<br>( 2048 bytes )             | 311,100 ± 4,150<br>( 4096 bytes )             | 510,000 ± 4,150<br>( 8192 bytes )             |
|                          | 064 | 428,533 ± 4,064<br>( 4096 bytes )             | 627,667 ± 4,222<br>( 8192 bytes )             | 1026,033 ± 4,285<br>( 16384 bytes )           |
|                          | 128 | 863,033 ± 4,483<br>( 8192 bytes )             | 1261,367 ± 4,644<br>( 16384 bytes )           | 2058,200 ± 4,460<br>( 32768 bytes )           |

Dois fatores relacionados ao aumento no tempo de retorno do método são observados na Tabela 11. (i) O primeiro ocorre quando se aumenta a quantidade de fragmentos gerados, aumenta-se também o tempo de resposta do método. Por exemplo, observando a coluna com tamanho de fragmento de 64 *bytes* e a linha com quantidade de fragmento 002, o valor médio de retorno foi 18,100  $\mu s$ . Permanecendo na mesma coluna (64 *bytes*) e passando para a próxima linha (004) o valor médio de retorno é de 28,700  $\mu s$ . Nas demais linhas, observam-se esses valores sempre aumentarem. (ii) O outro fator relacionado ao aumento do tempo de retorno do método ocorre quando o tamanho do fragmento é aumentado. Por exemplo, fixando-se na linha com quantidade de fragmento 004 e coluna com tamanho de fragmento 64, o tempo médio de retorno é de 28,700  $\mu s$ . Permanecendo na mesma linha e verificando as demais colunas (128 e 256), os valores são 41,067  $\mu s$  e 66,000  $\mu s$  respectivamente. Esse comportamento de aumento no tempo de resposta é evidenciado no gráfico em linhas da Figura 32. Nesse gráfico, o eixo vertical destaca o tempo de retorno do método ( $\mu s$ ) e o eixo horizontal, a quantidade de fragmentos. O tamanho dos fragmentos é representado pelas linhas frag\_64 (64 *bytes*), frag\_128 (128 *bytes*), e frag\_256 (256 *bytes*). Como é possível observar, o fragmento com tamanho de 256 *bytes* (frag\_256) apresenta o tempo de resposta maior do que os demais à medida que a quantidade de fragmentos aumenta.

Voltando para a Tabela 11, outra métrica que é possível ser observada, refere-se ao

Figura 32 – Tempo de resposta do método (tempo x fragmentos).



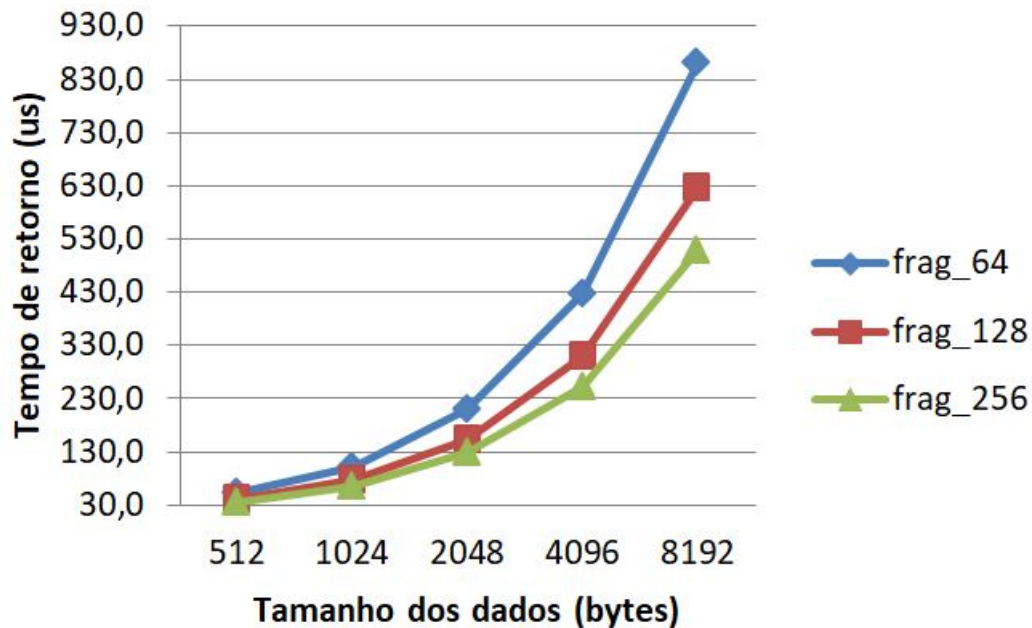
Fonte: Próprio Autor.

tamanho dos dados (em *bytes*) utilizados para gerar os fragmentos. Esses valores estão localizados logo abaixo das métricas de tempo, entre parenteses. Por exemplo, em destaque na coluna com tamanho de fragmento de 64 *bytes* e linha com quantidade de fragmento 008, o valor da quantidade de dados utilizados para a realização da métrica foi de 512 *bytes*. Essa mesma quantidade de *bytes* é utilizada em outras medições, por exemplo, na coluna de tamanho de fragmento de 128 *bytes* e linha com quantidade de fragmentos 004 e também na coluna com tamanho 256 e linha com quantidade 002. Verifica-se também nos demais campos da tabela essa repetição com as seguintes quantidades de dados 1024, 2048, 4096 e 8192 *bytes*. Transferindo esses valores para um novo gráfico é possível melhor visualizá-los.

O gráfico da Figura 33 relaciona o tempo de retorno do método (eixo vertical) com o tamanho dos dados (eixo horizontal). Neste gráfico é observado à medida que aumenta o tamanho dos dados utilizado na rotina de fragmentação, o tempo de retorno do método também aumenta, no entanto, diferente do exposto no gráfico anterior, o fragmento definidos com menor tamanho (frag\_64) tende a apresentar maior tempo de retorno. Esse comportamento é compreendido pelo fato de que quanto menor o tamanho do fragmento, maior será a quantidade de fragmentos gerados e automaticamente, maior será a demanda de recursos computacionais (memória e processamento) para executar o método.

No ponto de vista do usuário do *middleware*, é observado no gráfico da Figura 33, que até o tamanho de dados de 1024 *bytes*, as três configurações de tamanho do fragmento (64, 128 e 256 *bytes*) exercem uma influência semelhante no tempo de retorno do método. Entretanto, acima de 1024 *bytes*, verifica-se que o tamanho do fragmento opera

Figura 33 – Tempo de resposta do método (tempo x dados).



Fonte: Próprio Autor.

maior influência no tempo de retorno desse método. Desta forma, cabe ao projetista da aplicação embarcada que utiliza a rotina de fragmentação encontrar a melhor relação entre o tamanho do fragmento e a quantidade de dados utilizados, tendo em vista o tempo consumido pela quantidade de fragmentos gerados.

## 5.4 Conclusão

Este capítulo apresentou casos de uso que exemplificam a utilização do *Middleware SGAM*. O *middleware* se limitou a versão *Mbed OS 05*. Essa versão foi escolhida por possuir recursos de *multithreads* e mais conectividade auxiliando a criação de aplicações *IoT* complexas. No entanto, com a versão escolhida, perde-se a capacidade de rodar em dispositivos que requerem menos recursos computacionais, por exemplo, que utilizam pouca memória *RAM* e executam em processadores *Cortex-M0*. Quanto à compilação, o *middleware* utilizou três placas controladoras diferentes, verificando sua capacidade de executar em diferentes plataformas de *hardware*. Para a realização dos testes e obtenção das métricas, optou-se pela *STM32F429ZI*. Dois eixos principais foram explorados para avaliar o consumo de recursos computacionais: uso de memória e tempo de execução.

As investigações de memória mostraram o consumo de *RAM* e *Flash* requerido na geração da aplicação de monitoramento de ativos e fragmentação de dados. Verifica-se que a diferença entre os valores de memória obtidos foram proporcionais ao tamanho dos dados utilizados para testar a aplicação de fragmentação. Outro ponto averiguado é que

o *middleware* requisita *64kb* de memória *RAM* para executar essas aplicações, devido principalmente as funções de acesso ao hardware *LoRaWAN* e a versão utilizada do *Mbed OS*.

Quanto aos testes de verificação de tempo de execução. Observou-se que na aplicação de monitoramento remoto, as funções de obtenção de referências (ex: *getControler()*, *getSensor(gps)*, ...), apresentam tempos de retornos bastante aproximados. Nas demais funções do módulo controle, as métricas exibem valores bastante distintos, mas justificáveis, tendo em vista as diferentes funcionalidades de cada método.

As funções da rotina de fragmentação de dados, no geral, apresentam métricas de tempo semelhantes, no entanto, o método *setMsg()* consome mais tempo na execução. Esse método é responsável pelas rotinas de identificação de tipo de fragmento, fragmentação dos dados e adição na fila de transmissão. A investigação realizada se concentrou em verificar a influência do tamanho dos dados, tamanho dos fragmentos e, consequentemente, quantidade de fragmentos no tempo de resposta desse método.

Por fim, buscou-se como principal requisito auxiliar na redução do tempo de desenvolvimento de aplicações e simplificar o trabalho do projetista em duas tarefas importantes: (i) Explorar as diversas possibilidades de implementação visando ganhos em previsibilidade, desempenho e tamanho das aplicações sem perder de vista as limitações de recursos como memória e tempo de processamento e (ii) reusar componentes de *hardware* e *software* desenvolvidos em projetos anteriores. Com as métricas obtidas é possível considerar que a padronização aplicada no desenho da arquitetura do *middleware* auxiliou na obtenção de uma solução leve e reutilizável.

## 6 Conclusão

As Redes Elétricas Inteligentes (*Smart Grids*) empregam vários dispositivos para monitorar, analisar e controlar a própria rede. Esses dispositivos são implantados em grande quantidade em usinas de energia, centros de distribuição e instalações de consumidores. Uma *Smart Grid* requer conectividade, automação e rastreamento para tais dispositivos. A Internet das Coisas (*IoT*) tem sido utilizada para prover esses serviços, contudo, há uma série de desafios a superar. Os principais deles estão relacionados à diversidade de plataformas de hardware e software utilizadas para implementar computadores embarcados que estão mais próximos da infraestrutura da rede elétrica, chamados dispositivos finais.

Para reduzir o tempo de desenvolvimento dessas aplicações embarcadas, esta dissertação propõe um *middleware IoT*, denominado *Smart Grids Applications Middleware (SGAM)*, que visa facilitar a criação de soluções embarcadas para controle, monitoramento e medição em Redes Elétricas Inteligentes. Como recurso de projeto em sistemas embarcados, o *middleware SGAM* busca disponibilizar um conjunto de funcionalidades que permitirá ao desenvolvedor focar principalmente na regra de negócio da aplicação. Também facilitará a portabilidade das soluções criadas entre diversas plataformas de prototipação existentes. Utilizando os recursos disponibilizados pelo *middleware*, o desenvolvedor estará livre para programar sua aplicação independente do seu nível de conhecimento acerca da plataforma alvo, pois estará programando focado na *API* do *middleware*. Com isso, objetiva-se que pressões ligadas ao mercado de consumo também sejam atendidas, como a redução do tempo desde a ideia até a disponibilidade do produto (*time-to-market*) e sua crescente complexidade (SANGIOVANNI-VINCENTELLI et al., 2004).

Quanto à implementação da arquitetura proposta, três aspectos principais foram observados: (1) Um *middleware* leve (com baixo consumo de recursos computacionais) e reutilizável foi gerado. Com o *middleware SGAM*, foi possível projetar, de forma simples, soluções aplicadas para o controle e monitoramento dos dispositivos da rede elétrica. Parte desses aspectos positivos foi alcançada pela utilização de padrões de projeto direcionados a sistemas embarcados. (2) Adicionalmente, foi utilizado um sistema operacional de Internet das Coisas como plataforma para disponibilizar recursos de *hardware*, *software* e rede, acelerando o tempo de desenvolvimento da solução e aumentando a sua portabilidade. Verificou-se a execução do *middleware* no *Mbed OS*, permitindo sua compilação para diferentes placas de prototipação e sua interação com diversos sensores. (3) Outro fator considerado foi a integração com as redes *LPWAN*, especificamente utilizando o protocolo *LoRaWAN*, para prover comunicação bidirecional entre os dispositivos finais e o servidor de rede. Além disso, foi possível trafegar tamanhos de dados superiores às limitações impostas por estas redes por meio do uso de estratégias de fragmentação no envio dos dados.

De maneira geral, os resultados apresentados neste trabalho indicam que a proposta de arquitetura do *middleware SGAM* é promissora para projetar, de maneira simples e eficiente, aplicações embarcadas para sistemas de controle, monitoramento e medição aplicados em redes elétricas inteligentes.

No que diz respeito aos trabalhos futuros, é válido destacar alguns pontos: (i) Somente parte da especificação do *middleware* foi projetada; Logo, há espaço para explorar a implementação e validação dos demais submódulos, sistema e protocolos, incluindo também a camada multilinguagem. (ii) Outros casos de uso podem ser projetados a fim de avaliar de maneira integrada os demais módulos da arquitetura. (iii) Sugere-se, ainda, verificar eventuais limitações quanto ao número de sensores conectados e/ou quais plataformas embarcadas podem ser utilizadas.

# Referências

- AFZAL, M. K. et al. Ieee access special section editorial: The new era of smart cities: Sensors, communication technologies, and applications. *IEEE Access*, IEEE, v. 5, p. 27836–27840, 2017. Citado na página 22.
- AMIN, S. M.; WOLLENBERG, B. F. Toward a smart grid: power delivery for the 21st century. *IEEE Power and Energy Magazine*, v. 3, n. 5, p. 34–41, Sep. 2005. ISSN 1540-7977. Citado na página 21.
- ARM Mbed. *ARM Mbed An introduction to Arm Mbed OS 5, Jan 2019*. 2019. [Online; acessado 03-jan-2019]. Disponível em: <<https://os.mbed.com/docs/v5.11/introduction/index.html>>. Citado na página 38.
- ARM Mbed. *Development boards, Mai 2019*. 2019. [acessado 03-mai-2019]. Disponível em: <<https://os.mbed.com/platforms/>>. Citado na página 51.
- ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer networks*, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Citado na página 29.
- AWASTHI, A. et al. Development of a low-cost, hand-held, remote vacuum profile monitoring system. In: *2017 28th Irish Signals and Systems Conference (ISSC)*. [S.l.: s.n.], 2017. p. 1–5. Citado 3 vezes nas páginas 37, 45 e 46.
- BALSAMO, D. et al. Exploring arm mbed support for transient computing in energy harvesting iot systems. In: *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*. [S.l.: s.n.], 2017. p. 115–120. Citado 3 vezes nas páginas 37, 44 e 46.
- BARDYN, J.-P. et al. Iot: The era of lpwan is starting now. In: IEEE. *European Solid-State Circuits Conference, ESSCIRC Conference 2016: 42nd*. [S.l.], 2016. p. 25–30. Citado 2 vezes nas páginas 32 e 33.
- BASIT, A. et al. Efficient and autonomous energy management techniques for the future smart homes. *IEEE Transactions on Smart Grid*, IEEE, v. 8, n. 2, p. 917–926, 2017. Citado na página 30.
- BHATI, N. mhealth based ubiquitous fall detection for elderly people. In: *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. [S.l.: s.n.], 2017. p. 1–7. Citado 3 vezes nas páginas 37, 44 e 46.
- CENTENARO, M. et al. Long-range communications in unlicensed bands: The rising stars in the iot and smart city scenarios. *IEEE Wireless Communications*, IEEE, v. 23, n. 5, p. 60–67, 2016. Citado 2 vezes nas páginas 32 e 33.
- COLLIER, S. E. The emerging enernet: Convergence of the smart grid with the internet of things. *IEEE Industry Applications Magazine*, IEEE, v. 23, n. 2, p. 12–16, 2017. Citado na página 21.

- DOE (U.S. DEPARTMENT OF ENERGY). *The Smart Grid: an introduction...*: Exploring the imperative of revitalizing america's electric infrastructure. [S.l.], 2010. v. 01. Citado na página 21.
- DOUGLASS, B. P. *Design patterns for embedded systems in C: an embedded software engineering toolkit*. [S.l.]: Elsevier, 2010. Citado na página 52.
- EGLI, P. R. *OVERVIEW OF EMERGING TECHNOLOGIES FOR LOW POWER WIDE AREA NETWORKS IN INTERNET OF THINGS AND M2M SCENARIOS*. 2015. [Online; acessado 14-jan-2019]. Disponível em: <[http://www.indigoo.com/dox/itdp/12\\_MobileWireless/LPWAN.pdf](http://www.indigoo.com/dox/itdp/12_MobileWireless/LPWAN.pdf)>. Citado na página 31.
- FALCÃO, D. M. Smart grids e microrredes: o futuro já é presente. *SIMPÓSIO DE AUTOMAÇÃO DE SISTEMAS ELÉTRICOS*, v. 8, 2009. Citado na página 27.
- FILHO, H. G. S.; FILHO, J. P.; MORELI, V. L. The adequacy of lorawan on smart grids: A comparison with rf mesh technology. In: *2016 IEEE International Smart Cities Conference (ISC2)*. [S.l.: s.n.], 2016. p. 1–6. Citado 2 vezes nas páginas 42 e 46.
- GECEVIC, D.; BIAZIC, T. Contactless control of sanitary water flow and temperature. In: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. [S.l.: s.n.], 2017. p. 1551–1556. Citado 3 vezes nas páginas 37, 45 e 46.
- GHIASIMONFARED, A. et al. Development of a hybrid lora/g3-plc iot sensing network: An application oriented approach. In: *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. [S.l.: s.n.], 2017. p. 503–508. Citado 2 vezes nas páginas 43 e 46.
- GUBBI, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, Elsevier, v. 29, n. 7, p. 1645–1660, 2013. Citado na página 31.
- HOSSEIN, M. et al. Towards code generation for arm cortex-m mcus from sysml activity diagrams. In: *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. [S.l.: s.n.], 2016. p. 970–973. ISSN 2379-447X. Citado na página 51.
- KOPONEN, P. et al. Definition of smart metering and applications and identification of benefits. *Deliverable D3 of the European Smart Metering Alliance ESMA (available at www.esma-home.eu, members area)*, v. 42, 2008. Citado na página 28.
- KRAKOWIAK, S. Middleware architecture with patterns and frameworks. Citeseer, 2007. Citado na página 38.
- LI, F. et al. Smart transmission grid: Vision and framework. *IEEE transactions on Smart Grid*, IEEE, v. 1, n. 2, p. 168–177, 2010. Citado na página 28.
- LI, Y. et al. Research on water meter reading system based on lora communication. In: *2017 IEEE International Conference on Smart Grid and Smart Cities (ICSGSC)*. [S.l.: s.n.], 2017. p. 248–251. Citado 2 vezes nas páginas 43 e 46.
- LOPES, Y. et al. Smart grid e iec 61850: Novos desafios em redes e telecomunicações para o sistema elétrico. *XXX Simpósio Brasileiro de Telecomunicações*, 2012. Citado na página 27.

- LORA-ALLIANCE. *LoRaWAN™ What is it?: A technical overview of lora® and lorawan™ technical marketing workgroup 1.0*. San Ramon, Califórnia, EUA, 2015. Citado 4 vezes nas páginas 31, 33, 34 e 35.
- MARWEDEL, P. *Embedded system design*. [S.l.]: Springer, 2006. v. 1. Citado na página 36.
- MENG, W.; MA, R.; CHEN, H.-H. Smart grid neighborhood area networks: a survey. *IEEE Network*, IEEE, v. 28, n. 1, p. 24–32, 2014. Citado na página 22.
- MINISTÉRIO DE MINAS E ENERGIA. *Relatório, Smart Grid*: Grupo de trabalho de redes elétricas inteligentes, ministério de minas e energia. Brasil, 2010. Citado na página 21.
- MIORANDI, D. et al. Internet of things: Vision, applications and research challenges. *Ad hoc networks*, Elsevier, v. 10, n. 7, p. 1497–1516, 2012. Citado na página 31.
- MOHAMED, N. et al. A service-oriented middleware for cloud of things and fog computing supporting smart city applications. In: IEEE. *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing*. [S.l.], 2017. p. 1–7. Citado 4 vezes nas páginas 22, 38, 41 e 46.
- MOSCATELLI, A. Innovative system on chip platform for smart grids and internet of energy applications. In: *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*. [S.l.: s.n.], 2016. p. 1–2. Citado na página 21.
- NIST. *Guidelines for smart grid cyber security*: smart grid cyber security strategy, architecture, and high-level requirements. [S.l.], 2010. v. 01. Citado na página 28.
- OBAIDAT, M. S. et al. Context-aware middleware architectural framework for intelligent smart grid data management. In: IEEE. *2018 International Conference on Computer, Information and Telecommunication Systems (CITS)*. [S.l.], 2018. p. 1–5. Citado 3 vezes nas páginas 22, 41 e 46.
- PATTI, E. et al. Distributed software infrastructure for general purpose services in smart grid. *IEEE Transactions on Smart Grid*, IEEE, v. 7, n. 2, p. 1156–1163, 2014. Citado 2 vezes nas páginas 42 e 46.
- RAHMAN, L. F.; OZCELEBI, T.; LUKKIEN, J. J. Choosing your iot programming framework: Architectural aspects. In: *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*. [S.l.: s.n.], 2016. p. 293–300. Citado 2 vezes nas páginas 36 e 37.
- RAMESH, R. et al. Interoperable middleware for smartcities streetlighting on lorawan as a case study. In: IEEE. *2019 11th International Conference on Communication Systems & Networks (COMSNETS)*. [S.l.], 2019. p. 550–552. Citado 2 vezes nas páginas 41 e 46.
- RAZA, U.; KULKARNI, P.; SOORIYABANDARA, M. Low power wide area networks: An overview. *IEEE Communications Surveys & Tutorials*, IEEE, v. 19, n. 2, p. 855–873, 2017. Citado na página 30.
- RIVERA, R.; ESPOSITO, A.; TEIXEIRA, I. Redes elétricas inteligentes (smart grid): oportunidade para adensamento produtivo e tecnológico local. *Revista do BNDES, Brasil*, v. 40, 2013. Citado na página 23.

- RODRÍGUEZ-MOLINA, J.; KAMMEN, D. M. Middleware architectures for the smart grid: a survey on the state-of-the-art, taxonomy and main open issues. *IEEE Communications Surveys & Tutorials*, IEEE, v. 20, n. 4, p. 2992–3033, 2018. Citado 2 vezes nas páginas 39 e 47.
- SABRI, C.; KRIAA, L.; AZZOUZ, S. L. Comparison of iot constrained devices operating systems: A survey. In: *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*. [S.l.: s.n.], 2017. p. 369–375. ISSN 2161-5330. Citado na página 37.
- SALEEM, Y. et al. Internet of things-aided smart grid: technologies, architectures, applications, prototypes, and future research directions. *arXiv:1107.3313 - Submitted to Proceedings of the IEEE*, 2017. Citado 2 vezes nas páginas 29 e 30.
- SANGIOVANNI-VINCENTELLI, A. et al. Benefits and challenges for platform-based design. In: ACM. *Proceedings of the 41st annual Design Automation Conference*. [S.l.], 2004. p. 409–414. Citado 2 vezes nas páginas 22 e 81.
- SHU-WEN, W. Research on the key technologies of iot applied on smart grid. In: IEEE. *Electronics, Communications and Control (ICECC), 2011 International Conference on*. [S.l.], 2011. p. 2809–2812. Citado na página 30.
- SIDDIQUI, O.; HURTADO, P.; PARMENTER, K. *The Green Grid Energy Savings and Carbon Emissions Reductions Enabled by a Smart Grid*. [S.l.], 2008. Citado na página 27.
- WANG, Y. et al. Research on application and security protection of internet of things in smart grid. IET, 2012. Citado na página 30.
- WIBISONO, G. et al. Development of advanced metering infrastructure based on lora wan in pln bali toward bali eco smart grid. In: *2017 Saudi Arabia Smart Grid (SASG)*. [S.l.: s.n.], 2017. p. 1–4. Citado 2 vezes nas páginas 43 e 46.
- WOLF, M. *Computers as components: principles of embedded computing system design*. [S.l.]: Elsevier, 2012. Citado na página 36.
- WOLFGANG, P. Design patterns for object-oriented software development. *Reading Mass*, v. 15, 1994. Citado na página 47.
- ZANELLA, A. et al. Internet of things for smart cities. *IEEE Internet of Things journal*, IEEE, v. 1, n. 1, p. 22–32, 2014. Citado na página 31.
- ZHU, W. Teaching assembly programming for arm-based microcontrollers in a professional development kit. In: *2017 IEEE International Conference on Microelectronic Systems Education (MSE)*. [S.l.: s.n.], 2017. p. 23–26. Citado na página 51.