UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE TELEINFORMÁTICA

MESTRADO ACADÊMICO EM ENGENHARIA DE TELEINFORMÁTICA

JOSE DANILO DA SILVA COUTINHO FILHO

FUOTA-IOT: UMA PROPOSTA DE APRIMORAMENTO DO PADRÃO DLMS PARA ATUALIZAÇÃO DE FIRMWARE EM DISPOSITIVOS DE INTERNET DAS COISAS

FORTALEZA

2022

JOSE DANILO DA SILVA COUTINHO FILHO

FUOTA-IOT: UMA PROPOSTA DE APRIMORAMENTO DO PADRÃO DLMS PARA ATUALIZAÇÃO DE FIRMWARE EM DISPOSITIVOS DE INTERNET DAS COISAS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia de Teleinformática do Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia de Teleinformática. Área de Concentração: Engenharia de Teleinformática

Orientador: Prof. Dr. Paulo César Cortez

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C896f Coutinho Filho, José Danilo da Silva.
FUOTA-IoT : Uma proposta de aprimoramento do padrão DLMS para atualização de firmware em dispositivos de internet das coisas / José Danilo da Silva Coutinho Filho. – 2022.
79 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia de Teleinformática, Fortaleza, 2022.
Orientação: Prof. Dr. Paulo César Cortez.

1. LoRaWAN. 2. FUOTA. 3. DLMS. 4. multicast. 5. data block. I. Título.

CDD 621.38

JOSE DANILO DA SILVA COUTINHO FILHO

FUOTA-IOT: UMA PROPOSTA DE APRIMORAMENTO DO PADRÃO DLMS PARA ATUALIZAÇÃO DE FIRMWARE EM DISPOSITIVOS DE INTERNET DAS COISAS

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia de Teleinformática do Programa de Pós-Graduação em Engenharia de Teleinformática do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia de Teleinformática. Área de Concentração: Engenharia de Teleinformática

Aprovada em: 29 de julho de 2022

BANCA EXAMINADORA

Prof. Dr. Paulo César Cortez (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Jarbas Aryel Nunes da Silveira
Universidade Federal do Ceará (UFC)

Prof. Dr. Alexandre Augusto da Penha Coelho
Universidade Federal do Ceará (UFC)

Prof. Dr. Auzuir Ripardo de Alexandria
Instituto Federal do Ceará (IFCE)

Prof. Dr. Plácido Rogério Pinheiro
Universidade de Fortaleza (UNIFOR)

À Deus, à minha família, à minha esposa e aos professores que contribuíram na minha formação ao longo destes longos anos...

AGRADECIMENTOS

Concluir um curso de graduação é um sonho da grande maioria das pessoas, já tive esta oportunidade e agora concluo mais uma importante etapa no meu processo de formação e busca de conhecimento. Para chegar aqui, precisei da ajuda de diversas pessoas, as quais gostaria de agradecer.

Agradeço a minha mãe, que sempre me apoiou na busca pelo conhecimento, que apesar das dificuldades financeiras, com muito esforço me ajudou a sair do interior do estado para estudar na capital em uma instituição federal de renome como a UFC.

Agradeço a minha esposa, que me apoiou todos os momentos, acreditou no meu potencial, escutou as reclamações sobre as dificuldades do curso, mas nunca me deixou desanimar.

Agradeço aos demais membros da minha família que de alguma forma contribuíram nesta minha jornada.

Ao professor Ricardo Jardel, que além de orientar, foi um grande amigo e aconselhou a respeito de diversas coisas ao longo da vida acadêmica e pessoal.

Ao professor Paulo César Cortez, orientador, que tirou dúvidas, contribuindo sempre no aperfeiçoamento do trabalho.

Agradeço a todos os professores com os quais tive a oportunidade de aprender ao longo desses anos na UFC.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

Por fim, sou grato a todos que de alguma forma, direta ou indiretamente, contribuíram para a realização deste trabalho ou para minha formação.

"Mais do que máquinas precisamos de humanidade. Mais do que inteligência precisamos de afeição e doçura. Sem essas virtudes a vida será de violência e tudo estará perdido."

(Charles Chaplin, O Grande Ditador (1940))

RESUMO

A Internet das Coisas (*IOT*) prosperou bastante na última década, o que acarretou no surgimento expressivo de novas soluções e como resultado, milhões de novos dispositivos foram instalados. Com a perspectiva de crescimento ainda maior, algumas preocupações surgem relacionadas a segurança e manutenção destes dispositivos, o desejável é que as boas práticas de engenharia de *software* sejam incorporadas no processo de desenvolvimento e manutenção destas novas soluções onde se almeja manter o desenvolvimento contínuo. Para que atualizações sejam entregues a estes dispositivos, sejam correções de *bugs* ou novas funcionalidades, o processo de atualização de *firmware* se faz necessário. Devido as características destes dispositivos, o processo de atualização mais utilizado é o *Firmware Update Over the Air - FUOTA*, que permite utilizar tecnologias de comunicação sem fio para a atualização remota. Este trabalho tem como objetivo esboçar, programar e validar um modelo eficiente de transmissão de *firmware* para dispositivos de *IoT* que permita independência de tecnologia de comunicação, podendo ser adotado como padrão para qualquer solução de *IoT* devido as configurações que otimizam consumo de dados, tempo de transmissão e taxa de sucesso no processo. O trabalho também objetiva apresentar fundamentos principais de *IoT*, Redes de sensores sem fio, DLMS e os desafios envolvidos no processo de atualização. No decorrer deste trabalho será possível concluir que é possível determinar uma padronização no processo de atualização de *firmware* sem subutilizar a tecnologia de comunicação selecionada, uma contribuição importante para o processo de desenvolvimento de novas soluções que permite redução no tempo de desenvolvimento além de redução de custos de desenvolvimento e manutenção dos dispositivos.

Palavras-chave: LoRaWAN; FUOTA; DLMS; multicast, data block.

ABSTRACT

The Internet of Things (*IOT*) has prospered a lot in the last decade, which has led to the significant emergence of new solutions and, as a result, millions of new devices have been installed. With the prospect of even greater growth, some concerns arise related to the safety and maintenance of these devices, what is desirable is that good *software* engineering practices be incorporated into the development and maintenance process of these new solutions where continuous development is intended. For updates to be delivered to these devices, whether corrections of *bugs* or new features, the *firmware* update process is necessary. Due to the characteristics of these devices, the most used update process is *Firmware Update Over the Air - FUOTA*, which allows using wireless communication technologies for remote updating. This work aims to sketch, program and validate an efficient transmission model from *firmware* to *IoT* devices that allows independence of communication technology, which can be adopted as a standard for any *IoT solution.* due to settings that optimize data consumption, transmission time and success rate in the process. The work also aims to present the main fundamentals of *IoT*, Wireless Sensor Networks, DLMS and the challenges involved in the update process. In the course of this work, it will be possible to conclude that it is possible to determine a standardization in the *firmware* update process without underusing the selected communication technology, an important contribution to the process of developing new solutions that allows a reduction in development time in addition to reduction of device development and maintenance costs.

Keywords: LoRaWAN; FUOTA; DLMS; multicast, data block.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Internet das Coisas (*IoT*) mudou drasticamente a perspectiva que se tinha sobre conectividade, houve um rápido crescimento na implantação de novos dispositivos nos últimos 10 (dez) anos e a perspectiva de crescimento nos próximos anos é ainda maior. Estes dispositivos atualmente são desenvolvidos para solucionar uma série de problemas e prometem uma grande durabilidade, principalmente com os avanços tecnológicos em relação a circuitos de baixo consumo e meios de coleta de energia do ambiente (*Harvest Energy*).

Com o aumento de dispositivos conectados uma preocupação que surge está relacionada à segurança, logo estes dispositivos precisam garantir uma comunicação segura e confiável junto aos servidores de aplicação. Dentro desta atual realidade o desenvolvimento de dispositivos de *IoT* deve ser cada vez mais rígido quanto às boas práticas de engenharia de *software*, no qual destaca-se o desenvolvimento contínuo. Examinando a estratégia de desenvolvimento contínuo, os objetivos são bem compreensíveis: agregar novos recursos com liberações de atualizações, correção de *bugs* com *patchs* de solução e reestruturação visando melhoria dos processos ou consumo de recursos. Para que estas novas funcionalidades, correções ou melhorias cheguem aos dispositivos *IoT* é necessário realizar o processo de atualização do *firmware*.

Uma característica dos dispositivos *IoT* é que eles podem ser instalados praticamente em qualquer lugar para solucionar qualquer problema, logo, o acesso a estes dispositivos para uma atualização manual muito provavelmente é uma alternativa inviável. Para possibilitar a atualização de dispositivos sem que seja necessário um acesso direto ao mesmo, existe um método de atualização denominado *FUOTA - Firmware Update Over the Air* (atualização de *firmware* pelo ar, em tradução literal), este método permite que as dificuldades anteriormente citadas sejam superadas e o requisito de desenvolvimento contínuo seja garantido no processo de manutenção destes dispositivos em campo durante todo o tempo de vida do mesmo.

Com o crescimento de soluções *IoT* no mercado, os fabricantes de microcontroladores e transceptors de comunicação diversificaram seu portfolio de produtos, o que para equipes de desenvolvimento foi um fator positivo, dado que com mais opções, maiores são as chances de selecionar componentes que atendam melhor aos requisitos da solução em desenvolvimento. Um fator complicador é o fato da grande heterogeneidade dos dispositivos, com essa diversidade, padronizar qualquer processo é uma atividade muito complexa.

Tomando como exemplo o fluxo de atualização de firmware, cada fabricante disponibiliza um *framework* que possibilite a fácil execução desse processo em seus dispositivos, mas

nesse ponto surge um problema, manter um grande número de protocolos e fluxos diferentes para os mais diversos dispositivos instalados em campo.

Atuando sobre esta lacuna, buscou-se uma forma de padronização do processo tomando como base o padrão *DLMS*, uma especificação robusta, mantida por uma entidade internacional e com cooperação de diversos parceiros, que já é utilizada em mais de 60 países com foco em dispositivos de telemetria.

A característica principal da especificação é a adaptabilidade, logo, com ela é possível descrever a aplicação de praticamente qualquer solução de *IoT* existente, desta forma, uma implementação deste padrão foi realizada, com propostas de melhoria, visando preencher lacunas do processo de atualização de imagem original, observando possíveis pontos de falhas, ou etapas passíveis de otimização.

Nesta dissertação é apresentado alguns fundamentos de *IoT* e Redes de sensores sem fio, assim como as principais tecnologias de comunicação utilizadas, com foco na tecnologia *LoRaWAN*. Também é apresentado o contexto e dificuldades no processo de atualização de firmware para este tipo de dispositivo, pautando um protocolo existente e levantando seus pontos fortes e fracos. Desta forma, pretende-se com esta dissertação atingir os seguintes objetivos:

1.1 Objetivo principal

Apresentar um protocolo unificado para realizar *FUOTA* passível de embarcar em qualquer microcontrolador, com independência de mídia de comunicação, com suporte a atualização do próprio dispositivo ou dispositivos terceiros, com dispositivo de segurança e configurações que permitam maior adaptabilidade a tecnologia de comunicação utilizada evitando a subutilização.

1.2 Objetivos específicos

- Apresentar os principais fundamentos de Internet das Coisas e Redes de sensores sem fio;
- Apresentar os principais fundamentos da tecnologia *LoRa*, principal tecnologia utilizada nos ensaios de atualização de *firmware*;
- Apresentar os fundamentos do padrão *DLMS*, especificação base para a solução apresentada;
- Comparar a solução apresentada com outro protocolo mais utilizado no mercado, demons-

trando sua capacidade;

- Identificar outros elementos que possibilitem atualização e melhorias para o protocolo apresentado visando maximizar seu potencial.

Devido a heterogeneidade dos dispositivos *IoT* (diversidade de microcontroladores, tecnologias de comunicação, protocolos), pretende-se ao fim desta dissertação responder duas perguntas cruciais: se é possível padronizar o processo de atualização de *firmware* pelo ar e se com esta padronização algum diferencial da tecnologia de comunicação utilizada pode ser subutilizada, reduzindo as vantagens da padronização.

2 FUNDAMENTAÇÃO TEÓRICA

Alguns fundamentos de Internet das Coisas e Rede de sensores sem fio são apresentados neste capítulo, assim como as tecnologias de comunicação atualmente utilizadas para estes tipos de aplicações. Também são apresentados os conceitos que embasam os requisitos da solução proposta, além dos desafios atuais no processo de firmware update. O capítulo é concluído fazendo uma breve apresentação sobre o padrão DLMS, alicerce da proposta apresentada.

2.1 Internet das Coisas

A internet revolucionou os meios de comunicação ajudando as pessoas a se conectarem a qualquer momento a uma grande quantidade de informações disponíveis e que estão em constante expansão. Além de contribuir no processo de conexão entre pessoas para as mais diversas finalidades (uso pessoal ou profissional), a internet passou a ser usada para conectar pessoas a dispositivos físicos e conectar dispositivos a outros dispositivos.

Internet das Coisas, do inglês *Internet of Things*, ainda conhecida pela sigla do inglês *IoT*, é um conceito da área da computação que se refere à interconexão de objetos comuns com a internet. Em outras palavras, *IoT* é uma rede de objetos conectados capazes de trocar dados, na maior parte do tempo sem interferência humana (SINGH; SINGH, 2015). O termo *IoT* foi cunhado por Kevin Ashton em 1999 e veio evoluindo desde então, com aplicações em diferentes áreas e com um alto crescimento do número de aplicações. Apesar da grande evolução nos últimos anos os produtos e serviços de *IoT* ainda enfrenta alguns desafios tecnológicos.

Do ponto de vista técnico, a *IoT* tem oito grandes desafios a enfrentar (ROUTH; PAL, 2018), sendo eles:

- Segurança - à medida que a quantidade de dispositivos conectados cresce, a quantidade de vulnerabilidades também aumenta. A segurança é o pilar de toda aplicação para internet segundo a *Internet Society (ISOC)*, o que não é diferente para a *IoT*, sendo este, o desafio mais significativo (GRAMMATIKIS *et al.*, 2019);
- Privacidade - diversos cenários de *IoT* esconde em sua essência a coleta de dados constante e de maneira imperceptível pelos usuários. Os usuários tem um comportamento normal de confiança sobre os dispositivos *IoT*, supondo que estas aplicações respeitem sua privacidade. Para a *ISOC*, para o sucesso da *IoT*, faz-se necessário o respeito as escolhas de privacidade de cada usuário em uma ampla gama de exceções (ROSE *et al.*, 2015);

- Conectividade - com o surgimento constante de novas soluções, conectar tantos dispositivos continua sendo um grande desafio, os modelos e estruturas de rede foram evoluindo, mas ainda existem gargalos. Para fugir do clássico modelo cliente-servidor, o modelo de computação em névoa (do inglês *fog computing*), tem elevado o potencial das aplicações aprimorando a largura de banda e colaborando para atender as necessidades dos usuários (RABAY'A *et al.*, 2019);
- Compatibilidade - conceitualmente a *IoT* deveria permitir um ambiente de interconexão entre os mais diversos dispositivos, todavia a realidade ainda é outra, devido a incompatibilidade dos dispositivos devido a variedade de tecnologias e pilhas de protocolo disponíveis. Algumas tecnologias utilizadas 5 anos atrás já se tornaram obsoletas e deram lugar a outras que permitem maior compatibilidade aos dispositivos nos próximos anos (SUMAN *et al.*, 2019);
- Complexidade - para usuários comúns a *IoT* parece não ser algo tão complexo, todavia, por trás daquele dispositivo aparentemente simples existe uma grande integração de camadas de software, hardware e outros sistemas que permitem o devido funcionamento além de facilitar o serviço de conexão e gerenciamento dos dados. Desta forma, todos os dispositivos conectados podem ter padrões e protocolos de trabalho diferentes e difíceis de gerenciar nessa complexa arquitetura heterogênea que é a *IoT* (TALWANA; HUA, 2016);
- Gestão de dados - além de atuadores, os dispositivos de *IoT* tem uma outra grande funcionalidade, coletar dados para geração de informação. A utilização em tempo real de grandes fluxos de dados de IoT propõe uma mudança de paradigma para uma nova arquitetura distribuída, visto que continuar apostando em uma arquitetura centralizada baseada na nuvem, tende a gerar atrasos no fornecimento de serviços (YASUMOTO *et al.*, 2016);
- Fluxo de dados - devido a grande quantidade de dispositivos conectados, a troca de informações consome uma grande largura de banda o que ocasiona congestionamento de tráfego de dados e colisões. Definir estratégias é uma ação necessária para garantir que ocorra um uso efetivo da largura de banda alocada e dados não sejam perdidos, ou custos de planos de dados crescam desproporcionalmente devido o uso escessivo de replicação;
- Recurso energético limitado - qualquer dispositivo elétrico/eletrônico necessita de energia para funcionar, já os dispositivos *IoT* comumente são projetados para serem pequenos, quase impercecptíveis no ambiente. A tecnologia de armazenamento de energia não

evoluiu ao ponto de termos hoje fontes de energia bastante pequenas e que sejam capazes de fornecer uma alta capacidade de carga durante muito tempo (meses ou anos). Diversos trabalhos focam no tempo de vida do dispositivo tentando fazer uso de tecnologias de baixo consumo (*low power*) ou coleta de energia (*harvesting energy*), embora ainda exista um descompasso entre os sistemas de coleta e a energia demandada pelos dispositivos (RUAN *et al.*, 2017). Otimizar o consumo de energia continua sendo um dos principais desafios atuais.

2.2 Rede de sensores sem fio

O termo *IoT* é por vezes usada de forma comutável com o termo rede de sensores sem fio (RSSF), do inglês *wireless sensor networks - WSN*. *WSN* é uma rede de sensores que monitora e registra estados físicos (ambientais por exemplo) em uma unidade centralizadora, estes registros são analisados e geram ações de atuadores (que também estão em rede) com algum propósito no meio o qual estão inseridos. Embora *IoT* e *WSN* possuam semelhanças, estes termos não representam exatamente a mesma coisa.

Na *IoT* os dispositivos são projetados para ingressar na rede imediatamente, na *WSN* os nós podem ou não estar conectados a internet, dado que os nós roteiam seus dados entre eles até nós coletores. *WSN* é um tipo de rede *adhoc*, logo, existe a constante mudança de topologia, os nós são móveis e o roteamento é desafiador, devido aos nós roteadores não serem fixos.

Com estas informações a *WSN* pode ser considerada um subconjunto da *IoT*, visto que *WSN* refere-se exclusivamente à rede de sensores e atuadores, enquanto a *IoT* propõe-se a prover conectividade a nós sensores individualmente, embarcando inteligência artificial/aprendizagem de máquina, análise de *big data* e amazenamento/computação em nuvem.

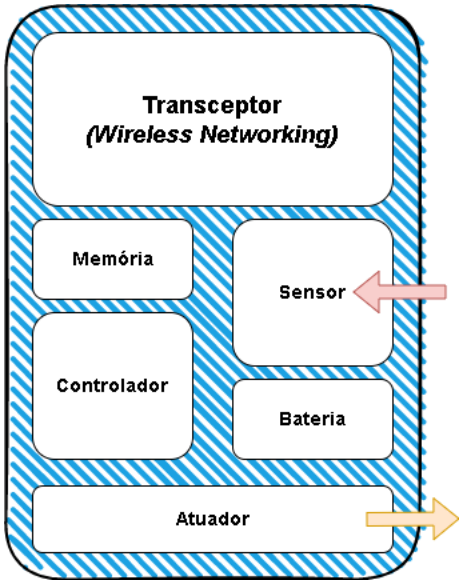
A *WSN* pode ser aplicada em diversas áreas, como implementações militares, de saúde, na agricultura, automação residencial ou industrial, dentre outras. Quanto a atuação das *WSN* elas podem ser classificadas em três tipos (BAGHYALAKSHMI *et al.*, 2011):

- Contínuo - neste modelo os sensores enviam dados periodicamente para o coletor;
- Sob demanda - neste modelo os sensores fazem constante monitoramento das variáveis físicas as quais foi programado para aferir e armazena os dados, todavia, o envio é feito exclusivamente quando solicitado;
- Orientado a eventos - neste modelo assim como no anterior, as variáveis físicas também são constantemente monitoradas, entretanto os dispositivos são configurados com limiares

os quais determinam quando um determinado evento ocorre, e isto determina que o nó sensor deve enviar os dados para a estação base.

As *WSN* possuem algumas características gerais: alta granularidade, baixo custo, facilidade de implementação e flexibilidade. Estas características contribuem para que *WSN* sejam implantados em locais inóspitos, de difícil acesso e em condições ambientais extremas. Um nó sensor é comumente composto por 5 componentes principais os quais podem ser observados na Figura 1: um transceptor de comunicação, microcontrolador, sensores e atuadores e fonte de energia (bateria), alguns autores citam um componente adicional, a memória, que na maioria dos casos é a própria memória do microcontrolador.

Figura 1 – Topologia de um nó sensor.



Fonte: Figura do autor.

Os dispositivos que compõem a *WSN* podem assumir seis diferentes papéis (BORGES *et al.*, 2014):

- Nó sensor - compreende apenas o controlador, transceptor de comunicação, fonte de energia e sensores, sua função é apenas monitorar as grandezas físicas;
- Nó de sincronismo - pode ser um nó sensor, todavia, possui a capacidade de processamento ampliada, recebe dados de todos os nós sensores da *WSN*, concomitantemente, gerencia a rede;
- Nó sensor e atuador - incorpora atuadores ao primeiro modelo da lista;
- Nó âncora - nó com localização conhecida que suporta os demais dispositivos da rede no processo de localização;
- Cabeça do cluster - nó ao qual todos os dispositivos do cluster (grupo de dispositivos em

uma determinada área geográfica) enviam os dados coletados. Este nós é o responsável pelo envio ao nó sorvedouro e também atua como repassador de solicitações de fora do clustes (comandos enviados para um grupo de dispositivos);

- Nó gateway - é o nó responsável pela conexão e entrega de dados para outras redes de comunicação.

2.3 Tecnologias LPWAN

As *LPWANs*, do inglês, *low power wide area networks* (ou redes de área ampla de baixa potência, em tradução literal), ganharam destaque devido sua capacidade de ofertar conectividade acessível para dispositivos de baixa potência que se encontrem distribuídos em grandes áreas geográficas. Este foi um grande passo para concretizar a implementação da *IoT*, já que as *LPWANs* substituíram as tecnologias convencionais de comunicação sem fio de curto alcance e tecnologias celulares já consolidadas (RAZA *et al.*, 2017).

Em sua concepção a *IoT*, ganhou destaque por poder ser aplicado praticamente em qualquer setor de negócios, desde cidades inteligentes, medição inteligente, agricultura, automação residencial e industrial, dentre outros. Todavia, para atender a estes propósitos, as redes sem fio convencionais como *Bluetooth*, *GSM*, *ZigBee*, *Wi-Fi*, etc, não tinham potencial para dispositivos projetados para baixo consumo de energia, ou que pudessem ser movimentados livremente e instalados a centenas de metros ou até quilômetros de uma estação base (um requisito primordial para aplicações de cidades inteligentes e logística por exemplo) (XIONG *et al.*, 2015).

O grande diferencial das tecnologias *LPWANs* que surgiram nos últimos anos é o fenomenal alcance de algumas dezenas de quilòmetros e vida útil de bateria de dez anos ou mais (PETAJAJARVI *et al.*, 2015), todavia, para obter estes resultados foi necessário renunciar as altas taxas de dados (atuando na ordem de dezenas de kilobits por segundo) e lidar com altas latências (comumente na ordem de segundos podendo atingir até minutos). Desta forma, é perceptível que as tecnologias *LPWANs* não podem ser consideradas em aplicações não tolerantes a atrasos.

2.3.1 Diferenciais tecnológicos

2.3.1.1 Longo alcance

Um diferencial tecnológico já citado é o fato das *LPWANs* conseguirem um amplo alcance geográfico de cobertura e alta capacidade de propagar o sinal em locais de difícil acesso (leia-se porões, sub-solo e áreas que possuam muitas barreiras físicas). Salvo algumas exceções, a maioria das *LPWANs* fazem uso da banda *Sub-GHz* que permite uma comunicação robusta. A vantagem em utilizar frequências mais baixas é que a tecnologia experimenta menos atenuação e desvanecimento que possam ser causados por obstáculos físicos densos, além de não enfrentar um alto congestionamento como é o caso da banda de 2,4 *GHz* (usada pela maioria das tecnologias sem fio convencionais) (RAZA *et al.*, 2017).

Outra característica que possibilita o longo alcance são as técnicas de modulação, as tecnologias *LPWANs* são projetadas para prover 150 ± 10 *dB* o que permite um alcance de algumas dezenas de quilômetros em área rural e até dez quilômetros em área urbana, para isso é reduzida a taxa de dados e modulação para empregar mais energia em cada bit transmitido. Outra técnica utilizadas pelas tecnologias *LPWANs* é o espalhamento de espectro onde se espalha um sinal de banda estreita por uma banda de frequência mais ampla, mantendo a mesma densidade de potência. A transmissão é um sinal semelhante a um ruído, isto torna a tecnologia mais robusta e resistente a interferências (NGUYEN *et al.*, 2019).

2.3.1.2 Baixo custo

Visando o gigantesco crescimento que se previa para a *IoT*, para que o sucesso comercial das tecnologias *LPWANs* fossem atingidos, fazia-se necessário reduzir a complexidade do hardware. Com a evolução tecnológica os fabricanntes de chips obtiveram bons resultados mantendo o custo de hardware abaixo de US$5 (cinco dólares) (RAZA *et al.*, 2017). Outro fator decisivo no quesito custo e que influencia diretamente a tomada de decisão de aderir a uma tecnologia *LPWAN* é o custo de implementação de infraestrutura mínima. Diferente de tecnologias sem fio de curto alcance (que necessitam de uma implementação densa de infraestrutura como gateways), nas tecnologias *LPWANs* uma única estação base é capaz de conectar alguns milhares de dispositivos distribuídos em uma ampla região geográfica.

2.3.1.3 Qualidade do serviço

Por atender uma grande diversidade de aplicações (tolerantes e não tolerantes a atrasos, com fonte de energia limitada e ilimitada) as *LPWANs* devem fornecer algum tipo de qualidade de serviço (*QoS*), mas não exceder nas técnicas para garantir esta qualidade de serviço de forma que o uso de determinada tecnologia *LPWAN* se torne completamente inviável em um tipo de aplicação.

Estas tecnologias fornecem *QoS* limitada e esse continua sendo um grande desafio, o principal motivo se dá o fato dos protocolos serem tipicamente *ALOHA* puro (permite que um host utilize o meio de transmissão sempre que tiver necessidade), embora isto represente uma série de benefícios (como simplicidade do protocolo, do *software* embarcado, economia de energia, dentre outros), esse processo de troca de mensagens descoordenadas normalmente implica no acréscimo da probabilidade de colisões de pacotes, o que ocasiona a perda de dados.

Alguns instrumentos podem ser utilizados para mitigar o problema relatado, por exemplo a implementação da camada *MAC* (*media access control*) (DVORNIKOV *et al.*, 2017), confirmação de mensagens recebidas e tentativas de retransmissão (SISINNI *et al.*, 2020), o problema destas técnicas é o fato de aumentar consideravelmente o tempo no ar, o que implica em acréscimo de consumo de energia. Desta forma, em aplicações no mundo real, na maioria das vezes a *QoS* é ignorada na maioria das aplicações, principalmente quando se deseja bateria de longa duração.

2.3.1.4 Baixo consumo de energia

O baixo consumo de energia é um dos principais requisitos na maioria das implementações de *IoT*, alguns fabricantes de soluções, visando uma menor frequência de manutenção dos dispositivos, os projetam de forma a possuírem uma vida útil de bateria de até dez anos (LU *et al.*, 2017), o que de maneira explícita, invalida a necessidade de manutenção, dado que normalmente são dispositivos simples, de baixo custo e que após este tempo de uso já se tornam obsoletos, induzindo a substituição por uma versão aprimorada do dispositivo.

Para atingir a economia de energia nos dispositivos algumas abordagens são exploradas:

- Arquitetura de hardware - a escolha do hardware é a etapa inicial, escolher controladores otimizados para baixo consumo (URSUţIU *et al.*, 2012), uso de chaves para desligar

completamente dispositivos externos não utilizados temporariamente para otimizar o consumo, seleção minunciosa da tecnologia *LPWAN* com módulos (transceptor) otimizados para baixo consumo, estes são passos fundamentais, alguns fabricantes visando facilitar a aderência a uma determinada tecnologia têm lançado no mercado módulos que integram o controlador e transceptor em um único *chip* (KIM *et al.*, 2014);

- Topologia de rede - como visto anteriormente, a arquitetura de malha reduz a necessidade de infraestrutura complexa (*WSN*), todavia, a medida que uma transmissão necessita de uma quantidade alta de saltos até alcançar o gateway, a tendência é que alguns nós da rede sejam sobrecarregados e tenham sua bateria drenada mais rapidamente (RAZA *et al.*, 2017). As novas tecnologias *LPWANs* superam este tipo de problema conectando os dispositivos finais diretamente as estações base, a topologia resultante é a do tipo estrela, neste modelo se obteve menor consumo e rapidez no processo de troca de pacotes;
- Complexidade da aplicação - reduzir o tempo do dispositivo em execução (leia-se fora do modo de baixo consumo) é o foco principal das equipes de desenvolimento de novas soluções, desta forma, a arquitetura da solução proposta deve ser modelada de forma a priorizar processamentos mais complexos na nuvem, o dispositivo deve ter atividades simples e objetivas, coletar dados e enviar, qualquer pós-processamento para atender requisitos de usuário serão realizados em software.

2.3.1.5 Escalabilidade

As soluções de *IoT*, em algumas áreas de negócios como cidades inteligentes por exemplo, podem facilmente alcançar altos números de dispositivos ativos (dezenas de milhares). Para suportar esta grande quantidade de dispositivos algumas técnicas são utilizadas, como por exemplo, técnicas de diversificação (EGGERS *et al.*, 2020), onde se emprega comunicação multicanal e multi-antena paralelizando transmissões e recepção com dispositivos. Esta técnica ainda agrega robustez, tornando a comunicação resiliente à interferência por conta do uso de vários canais.

Em algumas tecnologias, a única forma de cobrir uma grande área geográfica é intensificando a densidade de estações base, todavia esta implementação tende a causar interferência entre dispositivos finais, para que dispositivos mais distantes ainda consigam se comunicar com a estação base e os mais próximos se comuniquem mas não despedicem energia, técnicas de seleção de canal adaptativo (YU *et al.*, 2020) e taxa de dados são implementadas. Nestas

tecnologias os dispositivos selecionam os melhores canais para alcançar maiores distâncias de forma confiável e controlam a potência e taxa de dados de forma a se adaptarem ao ambiente onde estão inseridos interferindo menos no canal de comunicação em comparação as tecnologias onde este tipo de controle não é implementado.

2.3.2 LoRaWAN

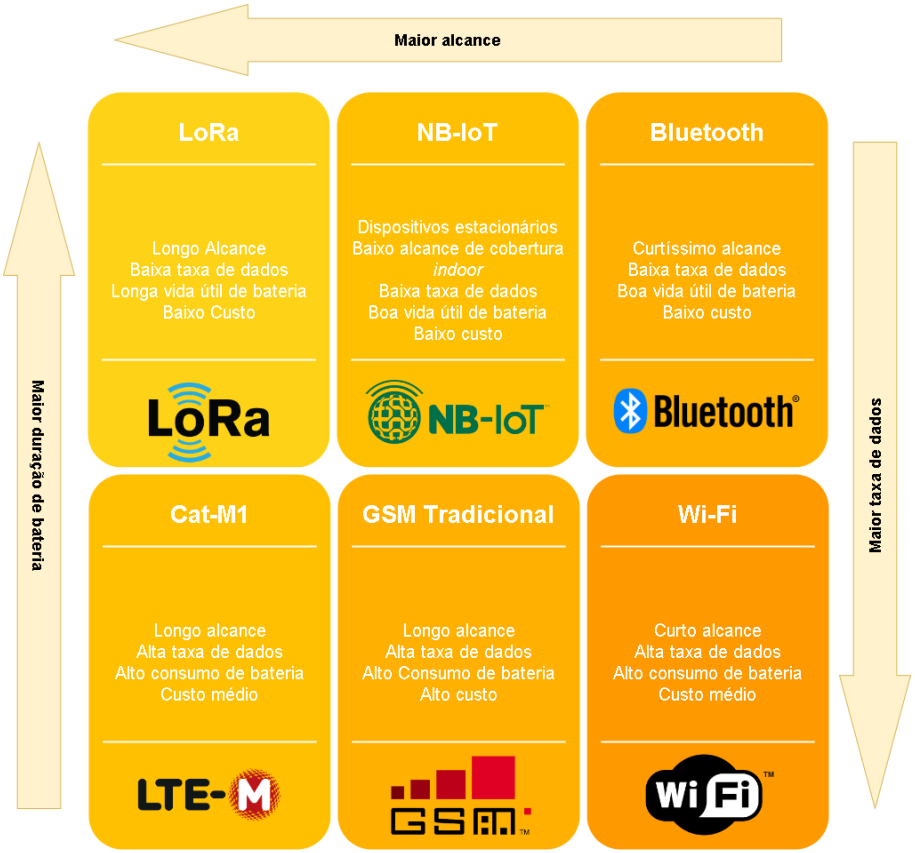
LoRaWAN do inglês *Long Range Wide-Area Network*, é um protocolo de rede aberto que oferece serviços seguros de comunicação bidirecional, mobilidade e localização padronizados e mantidos pela *LoRa Alliance* (SEMTECH-CORPORATION, 2020). *LoRa* é uma tecnologia de modulação de *RF* (rádio frequência) para redes de longa distância (*LPWANs*) de baixa portência. O nome *LoRa* se deve ao fato de uma das principais características desta tecnologia ser o longo alcance dos *links* de dados, permitindo até cinco quilômetros em áreas urbanas e quinze quilômetros ou mais em áreas rurais com visada direta, mas existindo hoje um recorde mundial estabelecido com a surpreendente distância de 766 *Km* (setecentos e sessenta e seis quilômetros) com uso de balões (LORA-ALLIANCE, 2020). Outra característica chave é o baixo requisito de energia, o que permite operar com baterias por até 10 anos.

Esta tecnologia atua sobre banda não licenciada que modula os sinais na banda de *SubGHz ISM* (AYOUB *et al.*, 2019) - *ISM* (do inglês industrial, scientific and medical), em uma porção do espectro de rádio reservada para estes fins usando a técnica de espalhamento de espectro. Foi inicialmente desenvolvida pela *Cyelo* e comercializada pela *Semtech*, *Microchip*, dentre outros.

Na figura 2 é possível comparar as diferenças, vantagens e desvantagens entre *LoRa* e outras tecnologias de rede sem fio comumente usadas em soluções tradicionais de conectividade máquina-a-máquina (*M2M*) e soluções de *IoT*. Uma característica de destaque da tecnologia *LoRa* sobre as demais é o consumo de energia, a energia necessária para transmitir um pacote de dados é mínima e quando o dispositivo está inativo, o consumo de energia é medido em *miliwatts* (*mW*), permitindo que a bateria do dispositivo permita o seu funcionamento por vários anos (SEMTECH-CORPORATION, 2020).

A partir daqui, será realizado uma análise mais minunciosa sobre a tecnologia *LoRa*, a *LPWAN* selecionada para protocolo de design de transmissão da imagem de firmware no processo de *FUOTA* (*firmware update over-the-air*).

Figura 2 – Comparação entre tecnologias de rádio frequência.



Fonte: Figura do autor.

2.3.2.1 LoRa PHY

2.3.2.1.1 Chirp Spread Spectrum Modulation

A tecnologia *LoRa* é baseada na modulação *Chirp Spread Spectrum - CSS*, que utiliza espalhamento ortogonal, ao invés das modulações convencionais (*QAM - Quadrature Amplitude Modulation*, *PSK - Phase Shift Keying* e *FSK - Frequency Shift Keying*), fator este que habilita taxas de dados variáveis e correção de erro de encaminhamento. Este tipo de modulação comprovadamente resiste melhor ao desvanecimento (*fading*) e ao ruído (NGUYEN *et al.*, 2020).

2.3.2.1.2 Code Rate and Forward Error Correction

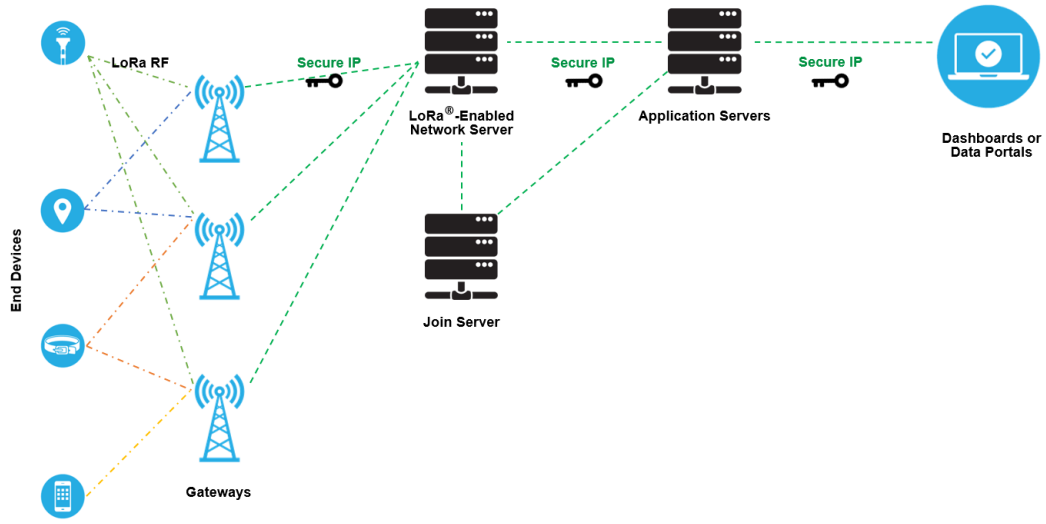
Coding Rate é uma forma de *forward error correction*, este métodos permite que *bits* corrompidos devido a interferência sejam recuperados, todavia este método exige uma pequena sobrecarga de codificação nos dados transmitidos. Para melhorar a robustez do sinal transmitido, a modulação *LoRa* inclui um esquema de correção de erro variável, para cada quatro *bits* de informação enviados é incluído um quinto *bit* de informação de paridade, mas este *CR*

pode variar de *4/5* e *4/8*, aumentar o *CR* garante maior confiabilidade, melhora a resiliência a *bits* corrompidos, todavia o preço pago é alto, dada a característica das tecnologias *LPWAN* essa mudança aumenta o tempo no ar e consequentemente existe um acréscimo no consumo de energia (YIM *et al.*, 2018).

2.3.2.1.3 Topologia

A arquitetura de uma rede *LoRaWAN* típica é composta por no mínimo três componentes principais, os dispositivos finais (*end devices*), os *gateways* e o servidor *LoRaWAN* (*LNS - LoRaWAN Network Server*). Na figura 3 é possível observar uma típica implementação *LoRaWAN* de ponta a ponta.

Figura 3 – Implementação típica de rede LoRaWAN.



Fonte: (SEMTECH-CORPORATION, 2020)

O primeiro componente desta grande rede é o dispositivo final, estes são os nós distribuídos geograficamente ao redor da infraestrutura de *gateways*, são dispositivos com recursos limitados, coletam dados e enviam por meio dos *gateways* até os servidores. Na maioria das aplicações *IoT*, a energia elétrica que alimenta os nós sensores são fornecidas por pequenas baterias, algumas soluções como na área de *smart grid* (rede elétrica inteligente) tem o benefício de possuir uma fonte de energia initerrupta fornecida pela própria rede elétrica, sendo esta uma das poucas soluções que não necessitam ser otimizadas para baixo consumo de energia.

O segundo componente da rede é o *gateway LoRaWAN*, este é o ponto no qual os dispositivos finais se conectam formando uma topologia do tipo estrela. Em uma mesma

área podem existir mais de um *gateway*, estes comumente são do tipo *half-duplex* (toda a capacidade do canal é dada ao dispositivo que estiver transmitindo no momento), existem gateways com multicanais o que permite uma maior capacidade de *uplink/downlink*, todavia o custo de aquisição é alto o que pode se tornar um impeditivo dependendo da aplicação (SMITH, 2017). Os *gateways* conectam aos servidores de rede utilizando conexão *IP* padrão, podendo o tráfego ser provisionado via *Wi-Fi, Ethernet* com fio ou tecnologias móveis como *3G*.

Finalmente, o servidor de rede *LoraWAN* é o componente da outra extremidade de uma implementação com esta tecnologia (quando se fala apenas na infraestrutura de rede, podendo existir ainda servidores de aplicação, dentre outros). Este servidor tem a responsabilidade de coordenar a entrega de pacotes de dispositivos finais aos seus respectivos servidores de aplicação (um mesmo servidor de rede pode atender a diversos servidores de aplicação). O *LNS* autentica os pacotes de dados recebidos e agenda os dados a serem enviados (*downlinks*) para dispositivos finais através dos gateways.

Na figura 3 é possível observar o componente intitulado servidor de aplicação, ele é o responsável por manipular, gerenciar e interpretar com segurança os dados dos dispositivos finais de uma determinada aplicação (SEMTECH-CORPORATION, 2020). Qualquer comunicação no sentido dos dispositivos finais (*downlink*) também é gerado por este componente, sendo ele o gerador apenas da carga útil (dados), todas as informações de criptografia, autenticidade ficando a cargo do *LNS* (exceto quando determinado que o servidor de aplicação é o responsável pela criptografia, ou se mais camadas de segurança forem somadas).

O processo de ativação de um dispositivo na rede pode se dar de duas maneiras, *ABP - Activation by personalization* ou *OTAA - Over-the-Air-Activation*. Na figura 3 é possível observar um terceiro servidor denominado *Join Server* (ou servidor de junção), este servidor contém as informações necessárias para processar as solicitações de junção (*Join Request*) e gerar os pacotes de aceitação de junção *Join Response*. Comumente os servidores *LNS* (quando se contrata infraestrutura de terceiros) tornam este servidor de *join* transparente ao usuário, de forma que aparentemente o próprio *LNS* fosse o responsável por manter estes dados.

2.3.2.1.4 Ativação

Como citado anteriormente, a ativação de um dispositivo final pode ser *ABP* ou *OTAA*. Para o processo de junção algumas informações são trocadas entre os atores do processo de junção. O *DevEUI*, por exemplo, é um número exclusivo de 64 *bits* que é atribuído a um

dispositivo final pelo fabricante, ele o identifica globalmente. O *DevAddr* é um outro identificador do dispositivo, mas desta vez na rede local, é um valor de 32 *bits* que pode ou não ser exclusivo na rede.

ABP: Neste tipo de ativação os dados para associação são fixas, além dos campos anteriormente citados, existem as chaves de sessão da rede *(Network session key)* e chave de sessão da aplicação *(Application session key)*, ambas chaves de 128 *bits*. Todas as informações de ativação permanecem as mesmas durante toda a vida útil de um dispositivo final, neste modo de ativação o passo de solicitação de junção é exluído, embora seja um método de ativação mais simples, tem suas limitações e fragilidades de segurança.

OTAA: Em geral não existem desvangatens em usar a ativação do tipo OTAA, é mais segura, não limita o dispositivo em relação ao contador de frames, todavia exige que o dispositivo esteja dentro da cobertura de rede de forma a evitar tentativas infinitas de junção sem sucesso. Neste tipo de ativação o dispositivo final possui chaves exclusivas de rede e aplicação, a partir destas chaves são geradas as chaves de sessão (tanto de rede como aplicação), que tem vida útil (até a próxima solicitação de junção) e um endereço do dispositivo que também é temporário.

2.3.2.1.5 Classes

A tecnologia *LoRaWAN* define três classes de dispositivos, classe A, B e C. De acordo com a especificação, todo dispositivo LoRaWAN deve implementar a classe A, logo as classes B e C são extensão da especificação da classe A.

Classe A: Quando o dispositivo está configurado nesta classe, sempre a comunicação é iniciada pelo dispositivo final, onde este pode enviar um *uplink* a qualquer momento, e logo após a transmissão o dispositivo abre duas janelas de recepção (*downlink*) denominadas RX1 e RX2, caso o *downlink* não ocorra dentro destas janelas uma nova tentativa de *downlink* só pode ocorrer após um novo *uplink* do dispositivo. Vale ressaltar que o dispositivo não pode receber dois *downlinks* consecutivos, aproveitando as janelas de recepção RX1 e RX2, caso o downlink ocorra na janela RX1, a janela RX2 não será aberta. Dispositivos são configurados como classe A normalmente quando se pretende ter uma maior autonomia de bateria e as aplicações não possuem um alto fluxo de dados.

Classe B: Assim como dispositivos classe A, os dispositivos configurados como classe B abrem janelas programadas para receber *downlinks* além das janelas RX1 e RX2 que são abertas logo após um *uplink*. Estes dispositivos tem latência menor que dispositivos configurados como classe A por serem acessíveis em horários programados sem a necessidade do envio de um *uplink* para que janelas de downlink sejam abertas.

Classe C: No caso dos dispositivos classe C a janela de recepção é mantida aberta constantemente, exceto quando o dispositivo está transmitindo. Esta configuração permite uma comunicação com baixíssima latência, todavia o consumo de energia é comprometido devido a necessidade do dispositivo ter que ficar constantemente no modo normal.

2.3.3 Tecnologias LTE

Tecnologias *LTE* (do inglês *long term evolution*) são padrões de redes de comunicação móveis que foram concebidas para manter a compatibilidade com *GSM* e *HSPA*, podendo atingir altas velocidades de *downlink* e *uplink* (150Mb/s e 50Mb/s respectivamente). Nesta quarta geração da tecnologia de telefonia móvel duas tecnologias se destacam para uso em aplicações de *IoT*, o *CAT-M1* que é utilizado geralmente em aplicações móveis e o *NB-IoT* em aplicações fixas, ambas se utilizam da mesma rede (*3G/4G*) e serão melhor abordados a seguir.

2.3.3.1 NB-IoT

A *NB-IoT* do inglês *Narrowband Internet of Things* é um tipo de *LPWAN* especificada pela *3GPP* em meados de 2016. É um padrão de tecnologia que permite um longo alcance com baixo consumo de energia, todavia, é indicada para aplicações que não demandam um alto volume de dados podendo atingir picos de *uplink* de 66 *Kbit/s* (multi-tom) | 16,9 *Kbit/s* (tom único) e taxas de *downlink* na ordem de 26 *Kbit/s*.

2.3.3.2 CAT-M1

A *Cat-M1* é um tipo de *LPWAN* especificada pela *3GPP* como parte da 13ª edição do padrão *LTE*, também pode ser chamada como *LTE Cat-M*, podendo ser visto desta forma em algumas literaturas. Esta tecnologia complementa a NB-IoT, podendo fornecer taxas de dados superiores de cerca de 1 *Mbps* de *uplink* e *downlink* com latência de 10 a 15 *ms*.

Esta tecnologia teve diversas melhorias em relação a *CAT1*, como a largura de banda

total necessária que antes era 20 *MHz* passando a 1,4 *MHz*, redução significante da complexidade, dentre outros.

2.4 Desafios do processo de Firmware Update

FUOTA (do inglês *Firmware Update Over The Air*) é o termo utilizado para definir o processo de atualização do *firmware* de dispositivos em um meio sem fio. A atualização de *firmware* é um recurso imprescindível para os sistemas embarcados, eles permitem que *patches* de segurança sejam implementados, novas funcionalidades sejam incorporadas ou otimização das funcionalidades já existentes sejam realizadas, tudo isso com o mínimo de intervenção humana (JONGBOOM; STOKKING, 2018).

Durante a explanação acerca da tecnologia *LoRaWAN* e *IoT* foi citado a longa vida útil dos dispositivos, podendo alcançar até 10 anos. Ter uma longa vida útil do *hardware* vai de encontro com o ciclo de vida do *firmware*, não só pela evolução das necessidades das aplicações, como a própria evolução das pilhas de protocolos embarcadas tomando como exemplo a *LoRa Stack*, desta forma o *FUOTA* garante que os dispositivos podem se manter atualizados ao longo de toda sua vida útil (ABDELFADEEL *et al.*, 2020).

A atualização de uma imagem de *firmware* é um processo que demanda o envio de uma grande quantidade de dados (podendo variar entre algumas dezenas de *kilobytes* para imagens pequenas ou atualizações parciais podendo chegar a algumas centenas de *kilobytes* para atualizações completas). Devido as limitações da tecnologia *LoRaWAN* transferir uma grande quantidade de dados é um impeditivo, dada a baixa taxa de dados que a tecnologia permite (ADELANTADO *et al.*, 2017).

Para que a transmissão de imagem de *firmware* seja eficiente algumas necessidades precisam ser atendidas (ABDELFADEEL *et al.*, 2020):

- Mecanismos para transmitir os blocos do *firmware* (*downlink*) sem a necessidade de requisições (*uplink*) serem enviadas primeiro, otimizando o ciclo de trabalho dos dispositivos e consequentemente seu consumo de energia;
- Mecanismos para baixar um bloco de big data e recuperar as perdas de blocos de *firmware* sem congestionar a rede com requisição de pacotes perdidos;
- Suporte *multicast* para otimizar o ciclo de trabalho dos *gateways* transmitindo blocos de *firmware* para diversos dispositivos em conjunto.

Devido a estas demandas a *LoRa Alliance* elaborou novas especificações para cobrir

tópicos de fragmentação, sincronização e *multicast* (CATALANO *et al.*, 2019).

2.4.1 Multicast

O multicast é o processo no gual um grupo de dispositivos pode receber o mesmo *downlink* com uma única transmissão. Como já citado, os dispositivos *LoRa* podem ser classe A, B ou C, no caso específico da classe A (que possui apenas duas janelas de recepção logo após o *uplink*, denominadas RX1 e RX2) a especificação *multicast* define um comando que configura o dispositivo como classe C ou classe B temporariamente, podendo retornar a classe A após determinado tempo ou envio de novo comando. Para a configuração *multicast* os comandos especificados são enviados de modo *unicast* na porta 200 do dispositivo, sendo eles (CATALANO *et al.*, 2018):

- McGroupSetupReq - uma requisição para configurar o grupo multicast no dispositivo, neste pacote é enviado a chave de segurança multicast que será usada por todos os dispositivos deste grupo para derivar a chave de rede e de aplicação (chaves usadas para criptografia e autenticação das mensagens). No mesmo pacote ainda contém o endereço compartilhado pelo grupo (assim como o *Device Address*, este endereço define o grupo dentro da rede no qual está inserido);
- McGroupSetupAns - é uma resposta enviada pelo dispositivo para confirmar a configuração multicast.
- McClassCSessionReq - uma requisição enviada para indicar que o grupo anteriormente configurado é da classe C, neste pacote ainda é informado o tempo de sessão, taxa de dados e canal;
- McClassBSessionReq - semelhante ao anterior, indica que o grupo configurado é da classe B, todavia, inclui a informação de periodicidade do *ping do slot* da classe B;
- McClassCSessionAns e McClassBSessionAns - é a resposta enviada individuamente para indicar a configuração da classe e sessão configurada para o grupo.

2.4.2 Fragmentação - Data Block

O algoritmo de fragmentação (*Data Block*) proposto na especificação da *LoRa Alliance* adiciona um código de correção de erro à imagem original do *firmware* antes do envio. Isso permite a recuperação de uma determinada porção das transmissões perdidas sem a necessidade de solicitar a retransmissão dos fragmentos perdidos. O processo consiste na

fragmentação da imagem em fragmentos de tamanhos iguais, posteriormente é adicionado fragmentos de redundância que são submetidos a lógica *XOR* em alguns framentos redundantes para reconstruir os fragmentos ausentes. Com a lógica a estimativa é que com cerca de 5% de redundância adicionada à imagem original permite que os dispositivos finais percam cerca de 5% das transmissões e ainda assim sejam capazes de reconstruir o *firmware* original (FUOTA-WORKING-GROUP, 2018).

Um problema inerente ao algoritmo anteriormente citado é que para perdas de pacotes maiores que o programado, percebe-se que é inviabilizado o processo de atualização do *firmware* daquele dispositivo (ou grupo de dispositivos). Desta forma o percentual de redundância deve ser acrescido de forma a ter uma maior taxa de sucesso no processo de atualização.

2.4.3 Sincronização

O comando *AppTimeReq* foi especificado para que os dispositivos *LoRaWAN* sincronizem seus relógios. Como a manutenção do tempo não é confiável em determinadas aplicações a especificação determinou uma maneira de corrigir os *drifts* no relógio dos dispositivos, proporcionando a disponibilização de um relógio *GPS* preciso que possa ser usado para atualizar constantemente o relógio dos dispositivos (FUOTA-WORKIN-GROUP *et al.*, 2018). Os comandos relacionados a esta especificação são enviados na porta 202 para distinguir das portas de aplicação. Todavia esta é uma implementação não obrigatória desde que a aplicação seja capaz de garantir a confiabilidade do relógio do dispositivo final.

2.5 Padrão DLMS

O padrão *DLMS - Device Language Message Specification* é uma linguagem globalmente aceita para dispositivos de medição inteligente, mas que pode ser facilmente adaptável a qualquer aplicação *IoT* devido suas características de interoperabilidade, eficiência e segurança. Esta padronização foi publicada em 2002 internacionalmente, a coletânea da especificação é dividida em cores (*blue, green, yellow e white book*) cada uma cobrindo um estágio do padrão.

O *DLMS* especifica os perfis de comunicação específicos da tecnologia utilizada (mídia física, seja sem fio ou física), o modelo de dados (desde a estruturação dos dados compartilhados, com perfis de acesso) e o protocolo de mensagens (determinando o formato da mensagem de cada serviço disponibilizado). Este padrão é usado em mais de 60 países e

implementado por mais de 150 fornecedores somando a quantidade de mais de 1.000 tipos de dispositivos certificados, podendo utilizar qualquer rede para fornecer solução de comunicação completa e eficiente.

2.5.1 Modelagem

O modelo de dados especificado no padrão *DLMS* é orientado a objetos, neste padrão existem 3 componentes principais, seguem:

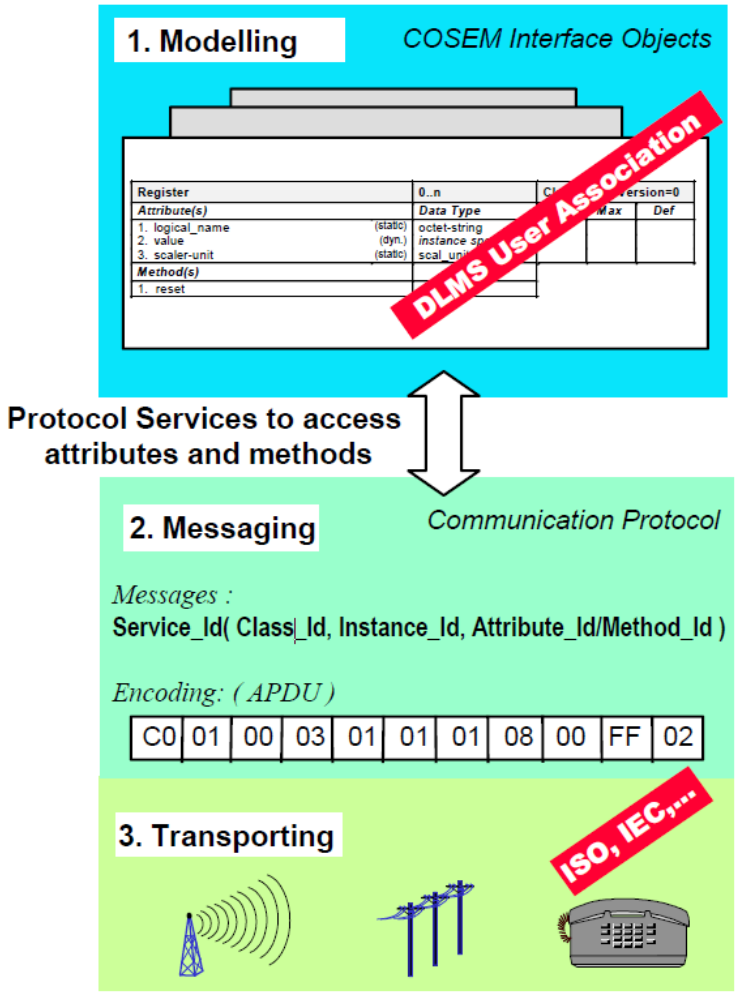
- COSEM - Companion Specification for Energy Metering, um modelo de objeto que pode ser utilizado na prática para a descrição de qualquer aplicação;
- DLMS - o protocolo da camada de aplicação que converte os dados mantidos pelos objetos criados na aplicação em mensagens que são trocadas pelos serviços;
- OBIS - Object Identification System, um gigantesco sistema de terminologias utilizados para classificação e criação de objetos.

A especificação *DLMS/COSEM* possuia uma linha de abordagem composta por três etapas as quais podem ser observadas na figura 4.

A modelagem deste padrão fornece blocos genéricos de construção para modelar qualquer funcionalidade, o modelo não contempla características internas específicas de implementação, o que garante a interoperabilidade já citada. O primeiro estágio listado na figura 4 (Modelagem), é especificado no *Blue Book*, neste livro se encontra as classes de interface (*IC*) do *COSEM*, seguido do padrão de reconhecimento *OBIS* o qual é utilizado para identificar instâncias das classes implementadas que podem ser também chamadas de objetos, estes por fim são as funcionalidades da aplicação.

Os estágios 2 e 3 são especificados no *Green Book*, neste livro é apresentado a *DLMS/COSEM Aplication Layer*, que especifica os serviços que podem ser utilizados para estabelecer uma conexão lógica entre um cliente e um servidor (1 ou mais), permitindo o acesso aos atributos e métodos dos objetos *COSEM*. Também é especificado como as mensagens da camada de aplicação podem ser transportadas por vários meios de comunicação, cada perfil de comunicação carrega seus protocolos específicos para suportar o *DLMS/COSEM Aplication Layer*. Vale ressaltar que qualquer implantação em larga escala demanda fortes sistemas de segurança provendo proteção e privacidade dos dados e a garantia da prestação de serviços aos consumidores, não é diferente para os sistemas de medição de energia, logo, neste livro também é abordado a pilha de segurança implementada na especificação.

Figura 4 – Abordagem de três etapas DLMS/COSEM.



Fonte: DLMS/COSEM Architecture and Protocols, 2020.

O *DLMS* fornece ferramentas de segurança integrados em várias camadas, desde mecanismos de identificação e autenticação de clientes e servidores, até direitos de acesso a atributos e métodos específicos de objetos implementados na aplicação. Os dados cifrados podem ser adaptados conforme cada implementação, deixando a cargo do desenvolvedor decidir qual algorítmo de criptografia utilizar, todavia, destaca-se todo o suporte que a especificação fornece para que as adaptações sejam feitas de maneira mais facilitada.

Esta dissertação não tem como objetivo se aprofundar no padrão *DLMS COSEM*, limitando-se a apresentar características principais do mesmo e focar no uso deste como ferramenta para padronização do processo de *FUOTA*, tomando como ponto de partida a implementação dos serviços básicos de leitura e escrita dos objetos (*read e write*) e o objeto principal para o processo de atualização de *firmware*, o *image transfer* especificado no livro azul da especificação *DLMS*.

2.5.2 Estrutura de Objetos

A Seção 4 do livro azul apresenta um panorama resumido de como os objetos de interface (instância de um *IC*) é usado para fins de comunicação entre cliente e servidor. O documento faz uso da técnica de modelagem de objetos onde um objeto é uma coletânea de atributos e métodos, na figura 5 é possível observar o modelo geral de um *IC*.

Figura 5 – Modelo geral de uma *IC - DLMS/COSEM*.

Class name		**Cardinality**	**class_id, version**			
Attributes		***Data type***	***Min.***	***Max.***	***Def.***	**Short name**
1. logical_name	(static)	octet-string				x
2. ...	(...)	...				x + 0x...
3. ...	(...)	...				x + 0x...
Specific methods (if required)		***m/o***				
1.		...				x + 0x...
2.		...				x + 0x...
3.		...				x + 0x...

Fonte: DLMS/COSEM Architecture and Protocols, 2020.

O atributo representa as características principais de um objeto, alterar o valor de um atributo pode afetar o comportamento do objeto acessado. De acordo com a especificação, o primeiro atributo de qualquer objeto é o *logical name*, este atributo é parte da identificação do objeto. Um servidor DLMS/COSEM pode prever duas formas de identificação dos objetos, o já citado *logical name - LN* e o *short name - SN*, a opção por um ou outro se dá a adaptabilidade as mais diversas tecnologias de comunicação que podem ser usadas, enquanto o *LN* usa 6 octetos para identificação, o *SN* usa apenas 2 octetos, o que trás benefícios na redução do uso de dados em determinadas ocasiões.

Objetos que compartilharem características usuais são generalizados como um *IC* usando uma identificação denominada *class id*. Em um *IC*, atributos e métodos são descritos uma vez para todos os objetos. Para permitir uma maior flexibilidade, os fabricantes podem adicionar métodos e atributos proprietários a qualquer objeto, todavia, os atributos e métodos obrigatórios devem ser preservados, mesmo quando não utilizados em sua completude.

Conforme a especificação DLMS/COSEM foi evoluindo ao longo do tempo, algumas *IC* foram melhoradas, o que gerou revisões, todo *class id*, possui um identificador *revision* associado, que permite que clientes e servidores diferentes possam se comunicar, mantendo

compatibilidade entre versões. Cada atributo também possui um tipo de dado associado, tentar alterar o valor para um tipo de dado não suportado ocasionará uma mensagem de erro na execução do serviço, mas é uma falha segura e programada preservando a integridade dos dados originais. Os tipos de dados suportados pelo *DLMS/COSEM* podem ser visualizados na Figura 6.

Figura 6 – Tipos de dados usados por atributos de objetos *COSEM*.

Type description	Tag [a]	Definition	Value range
-- simple data types			
null-data	[0]		
boolean	[3]	boolean	TRUE or FALSE
bit-string	[4]	An ordered sequence of boolean values	
double-long	[5]	Integer32	-2 147 483 648... 2 147 483 647
double-long-unsigned	[6]	Unsigned32	0...4 294 967 295
	[7]	Tag of the "floating-point" type in IEC 61334-4-41:1996, not usable in DLMS/COSEM. See tags [23] and [24]	
octet-string	[9]	An ordered sequence of octets (8 bit bytes)	
visible-string	[10]	An ordered sequence of ASCII characters	
	[11]	Tag of the "time" type in IEC 61334-4-41:1996, not usable in DLMS/COSEM. See tag [27]	
utf8-string	[12]	An ordered sequence of characters encoded as UTF-8	
bcd	[13]	binary coded decimal	
integer	[15]	Integer8	-128...127
long	[16]	Integer16	-32 768...32 767
unsigned	[17]	Unsigned8	0...255
long-unsigned	[18]	Unsigned16	0...65 535
long64	[20]	Integer64	$-2^{63}...2^{63}-1$
long64-unsigned	[21]	Unsigned64	$0...2^{64}-1$
enum	[22]	The elements of the enumeration type are defined in the *Attribute description* or *Method description* section of a COSEM IC specification.	0...255
float32	[23]	OCTET STRING (SIZE(4))	For formatting, see 4.1.6.2.
float64	[24]	OCTET STRING (SIZE(8))	
date-time	[25]	OCTET STRING SIZE(12))	For formatting, see 4.1.6.1.
date	[26]	OCTET STRING (SIZE(5))	
time	[27]	OCTET STRING (SIZE(4))	
-- complex data types			
array	[1]	The elements of the array are defined in the *Attribute* or *Method description* section of a COSEM IC specification.	
structure	[2]	The elements of the structure are defined in the *Attribute or Method description* section of a COSEM IC specification.	
compact array	[19]	Provides an alternative, compact encoding of complex data.	
-- CHOICE		For some COSEM interface objects attributes, the data type may be chosen at instantiation, in the implementation phase of the COSEM server. The server always shall send back the data type and the value of each attribute, so that together with the logical name an unambiguous interpretation is ensured. The list of possible data types is defined in the "Attribute description" section of a COSEM IC specification.	
[a] The tags are as defined in Clause 9.5 of DLMS UA 1000-2 Ed. 8.0:2014.			

Fonte: DLMS/COSEM Architecture and Protocols, 2020.

2.5.3 Serviços

O modelo de interface *COSEM* especificado no *DLMS UA 1000-1* é projetado para uso com uma variedade de perfis de comunicação para trocar dados entre cliente e servidor, para cada um destes perfis a camada de aplicação *DLMS/COSEM AL* fornece uma séries de serviços denominados *xDLMS*, com estes serviços os atributos e métodos de objetos *COSEM* podem ser acessados.

Para servidores que implementam a forma de identificação *logical name*, os serviços disponibilizados são:

- Get - utilizado para solicitar dados de um determinado atributo de um objeto *COSEM*;
- Set - utilizado para alterar dados de um determinado atributo de um objeto *COSEM*;
- Action - utilizado para ativar métodos de um determinado objeto *COSEM*.

Para servidores que implementam a forma de identificação *short name*, os serviçõe disponibilizados são:

- Read - utilizado para solicitar dados de um determinado atributo de um objeto *COSEM*;
- Write - utilizado para alterar dados de um determinado atributo de um objeto *COSEM* e também para ativar métodos de um determinado objeto *COSEM*.

Além destes serviços já citados o servidor *DLMS* também pode independente da forma de identificação implementada disponibilizar os seguintes serviços:

- Data notification - serviço em que o servidor envia dados periodicamente sem a necessidade de uma solicitação de leitura;
- Event notification - serviço em que o servidor reporta um evento (podendo incluir dados complementares) sem a necessidade de uma solicitação de leitura;
- Information report - serviço em que o servidor envia informações diversas (implementação de usuário) sem a necessidade de uma solicitação de leitura.

Com os serviços citados e os objetos descritos na especificação, o servidor *DLMS* é capaz de atender qualquer requisito de aplicação necessária, dada sua grande versatilidade.

3 METODOLOGIA DE DESENVOLVIMENTO

Neste capítulo apresenta-se o protocolo proposto para realização de uma atualização de *firmware* que pode ser aplicado para atualização do próprio dispositivo ou dispositivos terceiros (*third party*). Desta forma o capítulo divide-se em: Considerações, descrevendo as principais considerações para escolhas na definição do design do protocolo; Design do Protocolo, descrevendo as características principais, funcionalidades, dispositivos de segurança, configurações possíveis para propiciar maior adaptabilidade ao meio físico de comunicação e limitações; Materiais, descrevendo as ferramentas e materiais utilizados para a montagem do *setup* de validação; Avaliação, apresentação de outro protocolo já utilizado que atende o mesmo fim do protocolo proposto; e Métricas, apresentando as métricas utilizadas para comparação do protocolo proposto com o protocolo apresentado, identificando as vantagens e desvantagens de cada protocolo.

3.1 Considerações

Conforme apresentado no Capítulo 1 e 2, devido às características da *IoT* (grande variedade de dispositivos, protocolos, arquiteturas), determinar uma solução única para qualquer problema de projeto é um processo complexo. No caso da atualização de *firmware* não é diferente, cada protocolo ou tecnologia de comunicação pode fornecer um *framework* que possibilite uma maior facilidade no processo de *FUOTA*, todavia uma mesma organização pode possuir uma ou mais soluções com tecnologias e protocolos diferentes. Padronizar o processo pode reduzir de maneira relevante o tempo e custos de desenvolvimento, infraestrutura alocada, pessoal alocado, dentre outros.

Tomando como exemplo uma solução como os medidores de energia elétrica, embora os mesmos normalmente não sejam dotados de módulos de comunicação, podem receber *NIC's - Network Interface Card's* que possibilitem que o dispositivo possa se comunicar com servidores para envio e recebimento de dados (de medição) e até a atualização de *firmware*. Se um fabricante pretende fornecer mais de um tipo de tecnologia de comunicação (*LoRa, NB-IoT, Cat-M1, BLE*, dentre outros), muito provavelmente para cada solução um protocolo específico deveria ser implementado para gerenciar a transmissão da imagem de *firmware*.

O protocolo proposto leva em consideração que qualquer meio físico pode ser utilizado para a transmissão de uma imagem de *firmware*, sejam eles protocolos de comunicação

sem fio (*LoRa*, tecnologias *LTE*, *BLE*, etc), comunicação serial, dentre outros. Desta forma, o protocolo necessita de flexibilidade de configuração dada as limitações de cada tecnologia (taxa de dados, interferências físicas que ocasionem perda de pacotes, e até tolerância a ataques como *bit-fliping, Man-in-the-middle*, etc).

As principais características elencadas no protocolo proposto são:

- Interoperabilidade - pode ser utilizado em qualquer tipo de aplicação inteligente, com os mais diversos protocolos de comunicação;
- Eficiência - com as possibilidades de configurações disponíveis, pode maximizar o potencial da tecnologia de comunicação utilizada, evitando que alguma tecnologia seja sub-utilizada, maximizar a taxa de sucesso, e em alguns casos reduzir o consumo de dados e energia com a otimização proporcionada pelas configurações personalizadas;
- Segurança - camada adicional que pode ou não ser utilizada de acordo com a tecnologia de comunicação selecionada, agregando uma maior proteção no processo de atualização.

3.2 Design do Protocolo

Para dar início a descrição do design do protocolo, inicialmente pretende-se apresentar o objeto principal que será usado para a transferência de imagem, a partir desse objeto, serão apresentadas melhorias e casos de uso que permitem um melhor aproveitamento do potencial da especificação utilizada como referência.

3.2.1 Image Transfer

Conforme apresentado no Capítulo 2, o livro azul da especificação *DLMS* insere a definição das classes de interface (*ICs - Interface Classes*) e cita como elas são convertidas em objetos (instância do *IC*), que em sequência possuem seus atributos e métodos. Para a execução da transferência de imagem, o padrão *DLMS* possui um *IC* denominado *Image Transfer*, instâncias dessa classe (objetos) são enumeradas com o *class id* de número 18 e até a edição 14 da especificação usava-se a versão 0 (zero), que implementa 7 atributos e 4 métodos como pode ser observado na Figura 7.

Os 7 atributos implementados pelo objeto de transferência de imagem guardam informações importantes sobre o andamento do processo, já os métodos são as ações executadas para atingir o objetivo final, a atualização da imagem. A seguir cada um dos atributos é descrito:

Figura 7 – Especificação da *IC Image transfer*

Image transfer		**0...n**	**class_id = 18, version = 0**			
Attributes		***Data type***	***Min.***	***Max.***	***Def.***	**Short name**
1. logical_name	(static)	octet-string				x
2. image_block_size	(static)	double-long-unsigned				x + 0x08
3. image_transferred_blocks_status	(dyn.)	bit-string				x + 0x10
4. image_first_not_transferred_block_number	(dyn.)	double-long-unsigned				x + 0x18
5. image_transfer_enabled	(static)	boolean				x + 0x20
6. image_transfer_status	(dyn.)	enum				x + 0x28
7. image_to_activate_info	(dyn.)	array				x + 0x30
Specific methods		***m/o***				
1. image_transfer_initiate (data)		m				x + 0x40
2. image_block_transfer (data)		m				x + 0x48
3. image_verify (data)		m				x + 0x50
4. image_activate (data)		m				x + 0x58

Fonte: COSEM Interface Classes and OBIS Object Identification System, 2020.

- *Logical name* - identifica a instância do objeto *image transfer*;
- *Image block size* - guarda o valor do tamanho do bloco da imagem que pode ser tratado, este valor não pode exceder o tamanho máximo que servidor suporta, logo esse valor é uma propriedade do servidor;
- *Image transferred blocks status* - guarda informações sobre o *status* de transferência de cada bloco de imagem, um *array de bits* é instanciado para representar a quantidade de blocos da imagem completa e para cada bloco um valor individual pode ser atribuído (0 = não transferido, 1 = transferido);
- *Image first not transferred block number* - guarda o número do primeito bloco não transferido (mapeado no *array de bits* citado no item anterior) e ao fim do processo o valor retornado deve ser igual (ou superior +1, de acordo com a implementação) ao número total de blocos calculados (obtido a partir da divisão do tamanho da imagem pelo tamanho do bloco da imagem);
- *Image transfer enabled* - guarda a informação se o processo de transferência está habilitado ou não (*FALSE* = desabilitado, *TRUE* = habilitado), a invocação dos métodos de transferência de blocos retornarão erro se o processo não for habilitado e a habilitação é realizada a partir do método 1 ,*Image Transfer Initiate*;
- *Image transfer status* - guarda o *status* atual do processo de transferência, os valores possíveis são enumerados de 0 a 7, sendo eles: *image transfer not initiated, image transfer initiated, image verification initiated, image verification successful, image verification*

failed, image activation initiated, image activation successful e image activation failed;

- *Image to activate info* - guarda as informações relacionadas a imagem transferida (e pronta para ativação).

Ainda sobre o último objeto, as informações da imagem pronta para ativação são geradas como resultado do processo de verificação da imagem, em edições anteriores da especificação *DLMS* um dos passos obrigatórios era a confirmação dessas informações, o qual se tornou opcional em edições posteriores. Caso as informações sejam checadas antes da ativação, este objeto deve retornar uma estrutura contendo o tamanho da imagem para ativação, a identificação e a assinatura da imagem, a seguir uma descrição mais completa.

- *Image to activate size* - é o tamanho da imagem a ser ativada apresentada na forma de dados como octetos;
- *Image to activate id* - é a identificação da imagem a ser ativada, este campo é adaptável na implementação, podendo carregar informações como fabricante, tipo de dispositivo, versão, revisão, *patch* de correções, dentre outros dados;
- *Image to activate signature* - é a assinatura da imagem a ser ativada, o algoritmo para assinar as imagens de firmware não são abordados na especificação, ficando a cargo do fabricante definir o que melhor atende sua aplicação, levando em considerações fatores como robustez, garantia de confidencialidade, integridade, mas também considerando fatores como complexidade para embarcar em dispositivos com baixo poder de processamento, pouca memória e consumo de dados ao transferir pacotes.

Quanto aos métodos, como pode ser observado na Figura 7, a instância do objeto *image transfer*, implementa 4 métodos básicos, seguem:

- *Image transfer initiate* - método que inicia o processo de transferência de imagem, carrega uma estrutura contendo duas informações principais para o início do processo, a identificação da imagem (apresentada em octetos) e o tamanho da imagem (também apresentada em octetos não ultrapassando o tamanho máximo de um tipo *double-long-unsigned*. Após invocar com sucesso este método o objeto *image transfer status* é setado para 1, um *array de bits* é inicializado com tamanho definido igual ao tamanho da imagem (e valor zerado), e o primeiro bloco não transferido é setado para o valor 0 (zero);
- *Image block transfer* - método que ao ser invocado transfere um bloco da imagem, o pacote de dados é composto por uma estrutura contendo o número do bloco e o valor do bloco (bloco da imagem de *firmware*). Após invocar com sucesso este método o *bit*

correspondente no *array de bits* que representa a imagem completa é setado para o valor 1 (um) e o primeiro bloco não transferido é atualizado para o valor subsequente ao bloco recebido;

- *Image verify* - método que verifica a integridade da imagem antes da ativação, podendo retornar sucesso, falha temporária, ou outras razões (implementação do usuário), caso o resultado do processo seja sucesso o objeto *Image to activate info* irá guardar as informações da imagem já citadas no tópico relacionado, caso o resultado seja de falha além do retorno à invocação do método, informação do *status* ainda poderá ser coletado no objeto *image transfer status*;
- *Image activate* - método que ativa a imagem transferida, caso a etapa de verificação da imagem não tenha sido realizada, o método de ativação realiza a etapa adicional de verificar a imagem antes da ativação. Assim como o método anterior, a invocação pode retornar sucesso, falha temporário ou outras razões (implementação do usuário). Tanto em caso de sucesso como falha, a leitura do objeto *image transfer status* pode ser realizado para confirmação ou recuperação do resultado do ultimo processo realizado;

O processo de atualização de imagem de *firmware* pode ser executada seguindo um *script* simples composto por 7 etapas:

- 1 - obtenção do tamanho do bloco da imagem pelo cliente que irá realizar a transferência (etapa opcional);
- 2 - inicialização do processo de transferência da imagem pelo cliente;
- 3 - transferência dos blocos da imagem do cliente para o servidor;
- 4 - cliente checa se toda a imagem foi transferida ou se é necessário a recuperação de blocos;
- 5 - servidor verifica integridade da imagem (solicitada pelo cliente, ou por conta própria caso o cliente ignore a etapa de verificação);
- 6 - cliente verifica as informações da imagem para ativação (etapa opcional);
- 7 - servidor ativa a imagem (a ativação sempre é solicitada pelo cliente).

3.2.2 Push

Para a otimização do processo de *firmware update* uma segunda *IC* também especificada no livro azul é bastante útil, o *Push*. Esta *IC* configurável permite que mensagens *DLMS* possam ser enviadas para um destinatário sem que uma solicitação explícita ocorra. O *Push* pode

ser inicializado a partir de um gatilho como:

- um agendamento;
- um valor monitorado que atinge limiares superior ou inferior;
- um evento local.

A modelagem do *Push* é direcionada a eventos, cada caso de uso (evento assíncrono de envio de dados) requer uma instância de um *push setup*, esta configuração é composta por um *push object list* (lista de objetos implementados que serão reportados nesse *push*), configurações de envio (define parâmetros de randomização, retentativas, dentre outros), por fim um *push method* que executa a ação do envio configurado por meio do serviço de *data notification*.

Na Figura 8 é possível observar todos os objetos e métodos implementados pelo *IC - Push setup*, identificado pelo *class id* de número 40 e *version* 0 (zero), optou-se por apresentar a versão 0 (zero) deste *IC* (da edição 12 da especificação publicada em 2014) por conta da sua simplicidade em comparação a última versão (2), que implementa outros objetos e métodos que são opcionais.

Figura 8 – Especificação da *IC Push setup*

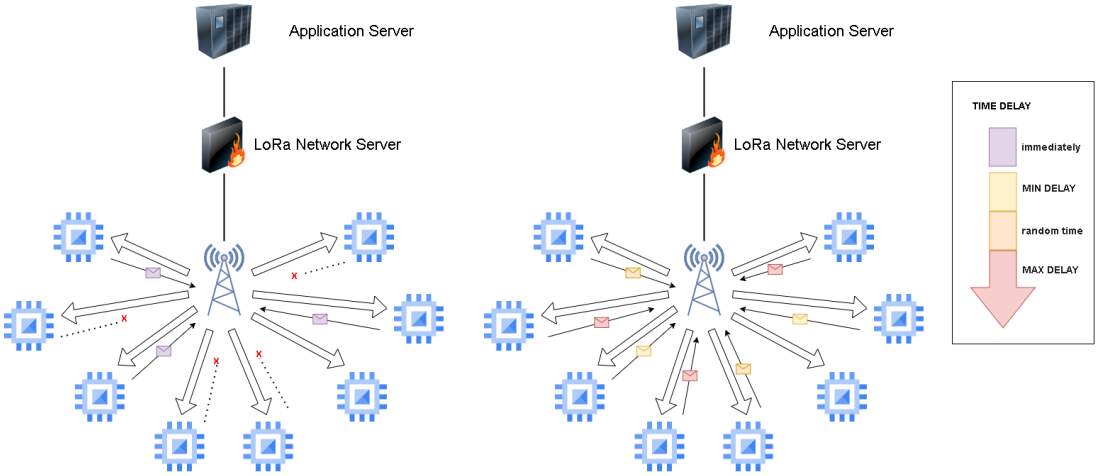
Push setup		**0...n**	**class_id = 40, version = 0**			
Attribute (s)		***Data type***	***Min.***	***Max.***	***Def.***	**Short name**
1. logical_name	(static)	octet-string				x
2. push_object_list	(static)	array				x + 0x08
3. send_destination_and_method	(static)	structure				x + 0x10
4. communication_window	(static)	array				x + 0x18
5. randomisation_start_interval	(static)	long-unsigned				x + 0x20
6. number_of_retries	(static)	unsigned				x + 0x28
7. repetition_delay	(static)	long-unsigned				x + 0x30
Specific methods		***m/o***				
1. push (data)		m				x + 0x38

Fonte: COSEM Interface Classes and OBIS Object Identification System, 2014.

Os atributos *randomisation start interval, number of retries* e *repetition delay* estão diretamente relacionadas a estratégia que será abordada posteriormente para otimização do processo de transferência de imagem, estes parâmetros permitem que a operação simultânea de uma grande quantidade de dispositivos não ocasione colisões na camada de comunicação. Partindo do pressuposto que embora um evento de push possa ocorrer simultaneamente em diversos dispositivos, o uso de valores randômicos para atraso no envio dos dados pode ser aplicado, aumentando a taxa de sucesso dos dados recebidos pelo servidor de rede.

Na Figura 9 é possível observar como a estratégia de resposta com atraso randômico pode contribuir para a maximização da taxa de sucesso do processo. Com uma pequena quantidade de dispositivos, ao se encaminhar um comando *multicast* ao grupo, mesmo que todos os dispositivos respondam imediatamente, um gateway com múltiplos canais é apto a tratar todas as respostas simultâneas. Todavia, conforme o número de dispositivos cresce neste grupo, a taxa de sucesso tende a reduzir, dado que as mensagens de confirmação/solicitação são perdidas (pacotes perdidos ilustrados com o "x"), o uso de respostas com atraso randômico permite um balanceamento do tempo de uso dos canais de comunicação, evitando as perdas de pacotes e necessidade de retransmissões (envio da resposta com tempo randômico dentro de um valor máximo pré-definido, ilustrado com cores dentro da escala mínima/máxima).

Figura 9 – Envio multicast com resposta imediata x resposta com delay randômico



Fonte: Figura do autor.

3.2.2.1 Processo de atualização otimizado

Conforme apresentado na Seção 3.2.1 (*Image Transfer*), o processo de atualização de um dispositivo pode ser executado seguindo 7 etapas simplificadas. O processo original embora robusto ainda é passível de otimizações, principalmente nos quesitos economia de dados, taxa de efetividade e escalabilidade. Para melhor ilustrar o processo as 7 etapas já apresentadas foram organizadas em um fluxo que pode ser observado nas Figuras 10 e 11.

3.2.2.2 Pontos de falha

A partir desta seção, alguns pontos do processo originalmente proposto serão analisados com maior critério levantando possíveis pontos de falha e melhorias propostas que

possibilitem a maximização da taxa de efetividade do processo de atualização de firmware.

De acordo com o fluxo de atualização de firmware apresentado pela especificação tomada como referência, apenas as etapas 2 e 3 (início do processo de transferência e transferência de blocos, respectivamente), podem ser utilizados no modo *multicast* (transferência para diversos dispositivos simultâneamente).

3.2.2.2.1 Diversidade de versões de *firmware*

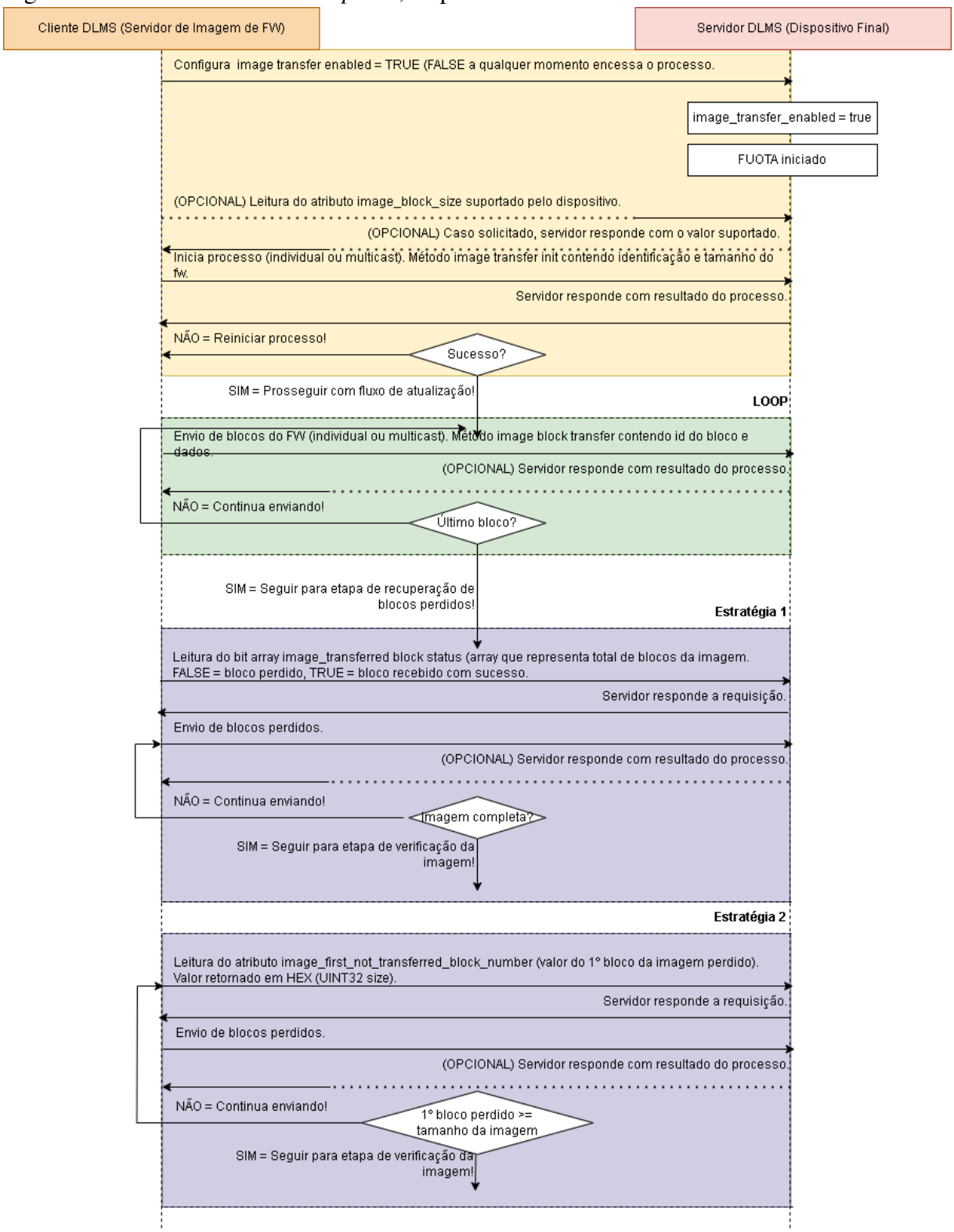
A etapa 1 do fluxo proposto na especificação é classificada como opcional, todavia é possível perceber um ponto de falha ao se adotar a estratégia de não obtenção do tamanho do bloco da imagem suportado, dado que em campo, comumente diversas versões de *hardware e firmware* de dispositivos podem coexistir, logo a configuração e/ou leitura inicial deste valor (que é editável para permitir maior adaptabilidade) torna-se uma etapa crucial. Apenas ler ou configurar este atributo já poderia ser uma solução simples, todavia em uma aplicação comum, esta etapa deveria ser realizada individualmente, dado que o envio multicast do comando de leitura e/ou configuração ocasionaria uma alta taxa de perda de pacotes de resposta (necessários para a confirmação do processo), reduzindo a taxa de efetividade e aumentando o uso de dados com retentativas.

Desta forma propõe-se uma estratégia mais elaborada que garante economia de dados e maior taxa de sucesso com uso do *IC push* apresentado anteriormente.

A proposta para o problema listado envolve tornar obrigatório a etapa 1, normalizando o valor do tamanho de bloco entre todos os dispositivos que serão atualizados e em seguida fazendo a leitura do valor como etapa de confirmação. Ambos os passos podem ser realizados como comando de massa (*multicast*), fazendo a implementação de um método adicional que retorne o valor do tamanho de bloco configurado, mas com delay aleatório, esta funcionalidade é possível estendendo o objeto *image transfer* em conjunto com a funcionalidade do *push*, onde o método de leitura receberia os parâmetros de *randomisation start interval*, *number of retries* e *repetition delay*.

Com esta implementação, independente se a atualização é de algumas dezenas, ou alguns milhares de dispositivos, o parâmetro de randomização passado como argumento do método, impediria que todos os dispositivos respondessem ao mesmo tempo a solicitação multicast, respondendo todos dentro de um intervalo aleatório configurado, e com possibilidade de repetições forçadas para garantir que a resposta chegue ao solicitante.

Figura 10 – Fluxo do *Firmware Update*, etapas de 1 a 4.

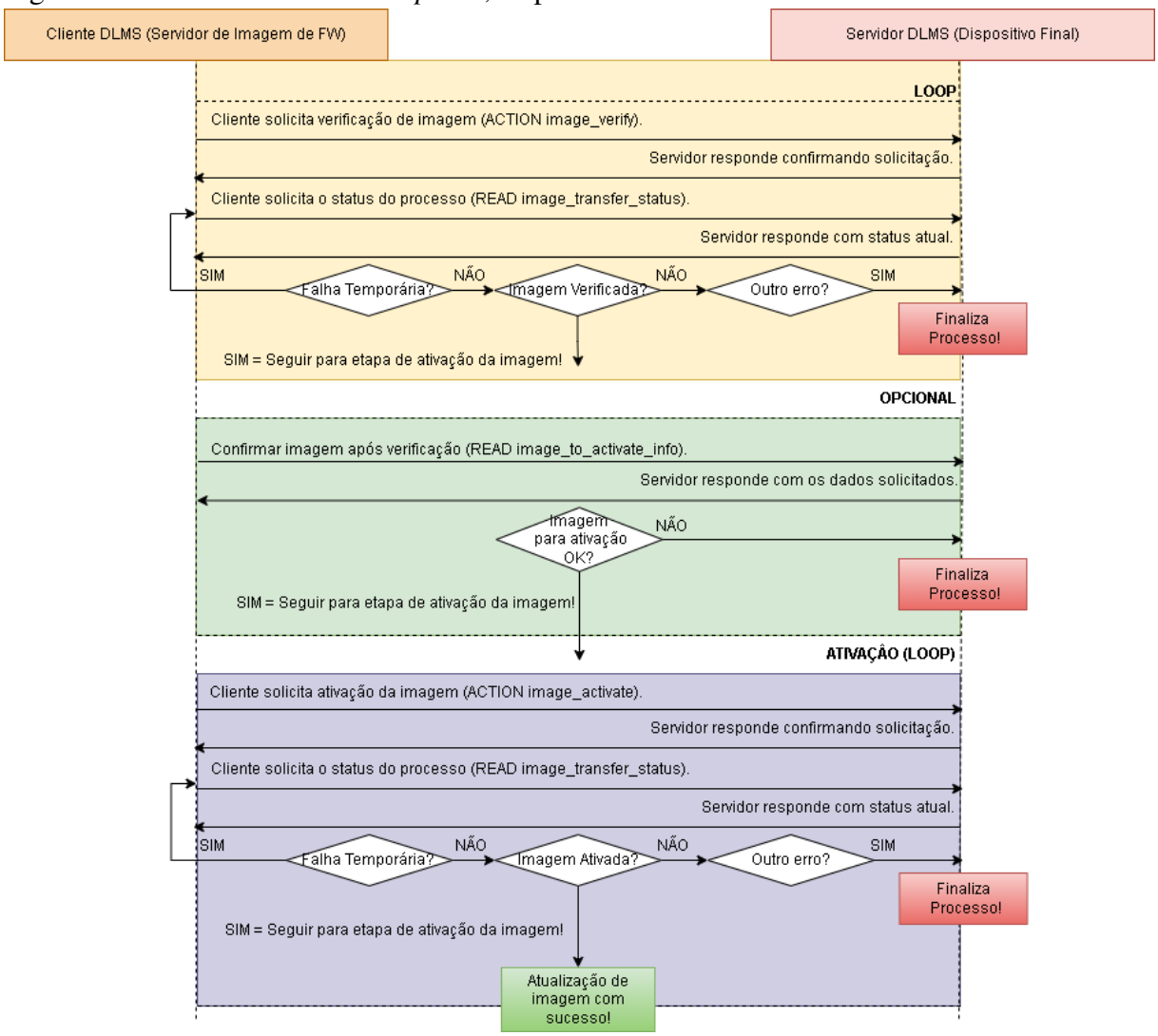


Fonte: Figura do autor.

3.2.2.2.2 Perda de pacotes de confirmação em comandos multicast

A etapa 2, inicialização do processo de transferência da imagem, é uma das etapas listadas como possível de ser realizada em massa (assim como a de transferência de blocos, etapa

Figura 11 – Fluxo do *Firmware Update*, etapas de 5 a 7.



Fonte: Figura do autor.

3). É possível observar um ponto de falha que pode reduzir drasticamente a taxa de efetividade do processo de atualização, dado que para poucos dispositivos e muitos gateways dentro da área de cobertura existe a possibilidade que todas as respostas sejam tratadas pelo receptor.

Todavia, para o caso de uma grande quantidade de dispositivos (o mais comum em uma aplicação real) existe a limitação dos gateways que ao tratar a recepção de uma transmissão recebida em um de seus canais, os demais pacotes que chegam são perdidos (o uso de gateways com mais canais embora mais caro, também possui limitações, gateways mais caros ainda não passam de algumas dezenas de canais). Mesmo que a camada física possua a capacidade de dar vazão a todas as solicitações, ainda existe o gargalo da aplicação, que também deveria ser capaz de tratar algumas centenas ou milhares de respostas simultaneamente.

Assim como na proposta anterior, uma alteração no método 1 *image transfer initiate* do objeto *image transfer*, permitiria em associação com o *push*, a alteração da resposta imediata,

para uma resposta com tempo randômico, com repetições programadas e com *delay* entre as repetições, garantindo que a taxa de sucesso no processo seja maximizada, e tanto a camada de comunicação, como a de aplicação, não sejam sobrecarregadas no caso de um comando em massa para uma grande quantidade de dispositivos.

3.2.2.3 Pontos de melhoria

3.2.2.3.1 Recuperação de pacotes - Estratégia 1

A etapa 4, recuperação de blocos perdidos é listada como uma etapa que deve ser realizada individualmente (*unicast*), propõe-se que esta etapa seja realizada em massa (*multicast*), todavia com alterações no fluxo visando otimização do processo com significante redução de *downlinks* e *uplinks*.

Na proposta os dispositivos que passam por processo de atualização são agrupados em subgrupos de aplicação *multicast* divididos em regiões geográficas (dentro de uma cidade por exemplo, podem existir zonas, bairros, ruas, quadras) e randomicamente, um dispositivo de cada grupo será solicitado o *bit array* que representa os blocos recebidos e não recebidos da imagem.

Com operações lógicas simples *(OR)*, o servidor pode montar uma lista de blocos que devem ser retransmitidos, e como possivelmente, estes blocos foram perdidos por mais de um dispositivo, a retransmissão pode ser realizada *multicast*, reduzindo drasticamente o consumo de dados, colisões e tempo de atualização de todos os dispositivos globalmente. Após o envio dos blocos perdidos uma nova solicitação do *bit array* deve ser realizada para novos dispositivos aleatórios selecionados dentro dos subgrupos, até que os dispositivos aleatórios retornem que todos os blocos da imagem foram recebidos.

Importante ressaltar que, com uso de pacotes de dados contratados de redes públicas, o custo de um *downlink multicast* para uma quantidade ilimitada de dispotivos é o mesmo que um *downlink unicast*, logo esta otimização reduz drasticamente os custos de manutenção dos dispositivos em campo.

3.2.2.3.2 Confirmação de imagem completa

Tomando como referência a estratégia 1 da etapa 4 citado anteriormente, vale salientar que devido a aleatoriedade é possível que os dispositivos randomicamente selecionados sejam *outliers* (pontos fora da curva), podendo assim existir dispositivos dentro dos subgrupos

que ainda possuem blocos faltantes da imagem. Desta forma ainda é necessário um passo adicional (existente também no fluxo comum de atualização), a confirmação de que a imagem está completa, esta confirmação pode ser realizada solicitando novamente o *bit array* de cada dispositivo, todavia isto pode gerar um *overhead* da rede dependendo do tamanho da imagem.

A estratégia 2 do fluxo comum propõe a leitura do atributo *image first not transferred block number*, uma adaptação com criação do método de leitura deste atributo com atraso aleatório é proposto, desta forma, um comando *multicast* pode ser enviado para todos os dispositivos, e o servidor pode montar uma nova lista de blocos perdidos que podem ser enviados multicast. Novas rodadas podem ser realizadas até que todos os dispositivos retornem que a imagem foi completamente transmitida.

3.2.2.3.3 Recuperação de pacotes - Estratégia 2

Como já citado, a etapa de recuperação é listada na especificação como uma etapa *unicast*, que pode ser realizada com duas estratégias que entregam o mesmo resultado: leitura do atributo que retorna o *bit array* completo que representa a imagem ou leitura do ultimo bloco não transferido. A estratégia 1 citada anteriormente é uma combinação das duas estratégias, beneficiando a redução do uso de dados, dado que a estratégia 1 melhorada permite um avanço significativo no tempo do processo de recuperação, além da quantidade de downlinks dada a otimização já citada.

O uso apenas da estratégia 2 com a otimização citada no tópico anterior (método com atraso aleatório para leitura do ultimo bloco transferido), já pode sozinha entregar uma redução considerável no tempo e consumo de dados, ficando a cargo do desenvolvedor analisar os custos computacionais para o servidor de aplicação que trata o envio das imagens.

3.2.2.3.4 Verificação da imagem

A etapa de verificação da imagem é listada na especificação como uma etapa individual, propõe-se a criação de um método com resposta com atraso aleatório, otimizando o consumo de dados com um único *downlink multicast* e evitando o *overhead* da rede.

3.2.2.3.5 Ativação da imagem

A etapa de ativação da imagem é outro comando listado na especificação como uma etapa individual, propõe-se a criação de um método com resposta com atraso aleatório, optimizando o consumo de dados com um único *downlink multicast* e evitando o *overhead* da rede.

3.2.2.3.6 Confirmação da ativação da imagem

A etapa de confirmação de ativação da imagem é listada na especificação como uma etapa individual onde se realiza *downlinks* constantes (*pooling*) solicitando o *status* da operação de ativação, propõe-se a criação de um método de resposta com atraso aleatório, optimizando o consumo de dados com um único *downlink multicast* e evitando o *overhead* da rede. Propõ-se ainda a criação de um método que pode ser acionado automaticamente após o salto do bootloader para a aplicação, informando o *status* do processo sem necessidade de leitura prévia por parte do servidor, este envio podendo ainda ser configurado com atraso aleatório e repetições forçadas, aumentando as chances de que a informação chegue até a aplicação.

3.2.2.3.7 Método adicional - blocos perdidos

Visando um aumento de performance na etapa de recuperação de pacotes, tomando como exemplo o caso do uso da rede *LoRaWAN*, o *payload* útil de *uplink* gira em torno 51 *bytes*, ao se realizar a leitura do ultimo bloco não recebido, boa parte é desperdiçada, dado que o dispositivo informa apenas uma informação que contem 4 *bytes* (número do último bloco não recebido).

Embora o *DLMS/COSEM* possua um cabeçalho que também consome parte deste *payload* útil, propõe-se a otimização com a criação de um método que retorne os 10 últimos blocos não recebidos (totalizando 40 *bytes* de dados), restando ainda 11 *bytes* para o cabeçalho do protocolo. Esta estratégia permite uma melhoria significativa no processo de recuperação tanto na estratégia 1 como na 2, reduzindo o número de *downlinks* realizados o que impacta no tempo global do processo de otimização e custos de manutenção da rede.

Este método pode ser criado com parâmetro de entrada, desta forma, se agrega adaptabilidade ao tipo de tecnologia de comunicação utilizada, no exemplo citado temos um pior caso da rede *LoRa* (que limita os *uplinks* a 51 *bytes*), mas no uso de tecnologias *LTE*, podemos

ter uma taxa de dados maior com limitação de 1500 *bytes* aproximadamente, dependendo da tecnologia e do módulo utilizado (varia para cada fabricante).

3.2.2.4 Resumo do processo de atualização otimizado

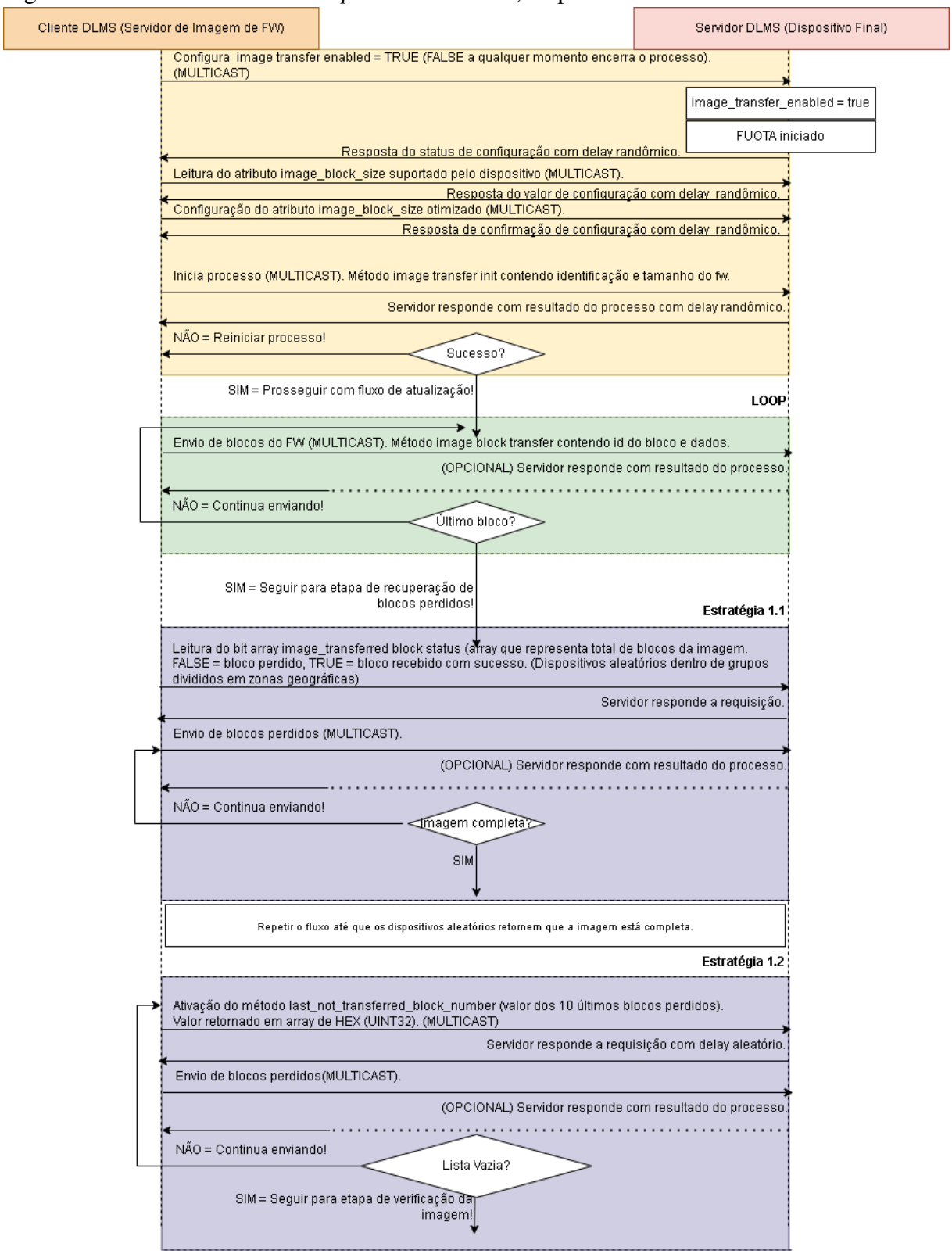
Nas Figuras 12 e 13 é possível observar como ficou o processo de atualização otimizado, todas as 7 etapas originalmente propostas no padrão *DLMS* foram adaptadas para serem executadas via comando *multicast* reduzindo tempo e consumo de dados amparado pelo uso de configurações de randomicidade e criação de métodos adicionais conforme já citado. A método de recuperação de blocos perdidos pode ser executada apenas com a estratégia 1.2, ou com a sequência de estratégias 1.1 seguido da estratégia 1.2, acelerando o processo e reduzindo o uso de dados.

3.3 Materiais

Como já descrito ao longo do texto, o padrão apresentado é interoperável, logo independente da plataforma selecionada as funcionalidades podem ser garantidas desde que a especificação seja respeitada durante a implementação. Atualmente no mercado, quando se trata de sistemas embarcados, boa parte das implementações fazem uso de microcontroladores de arquitetura *ARM*.

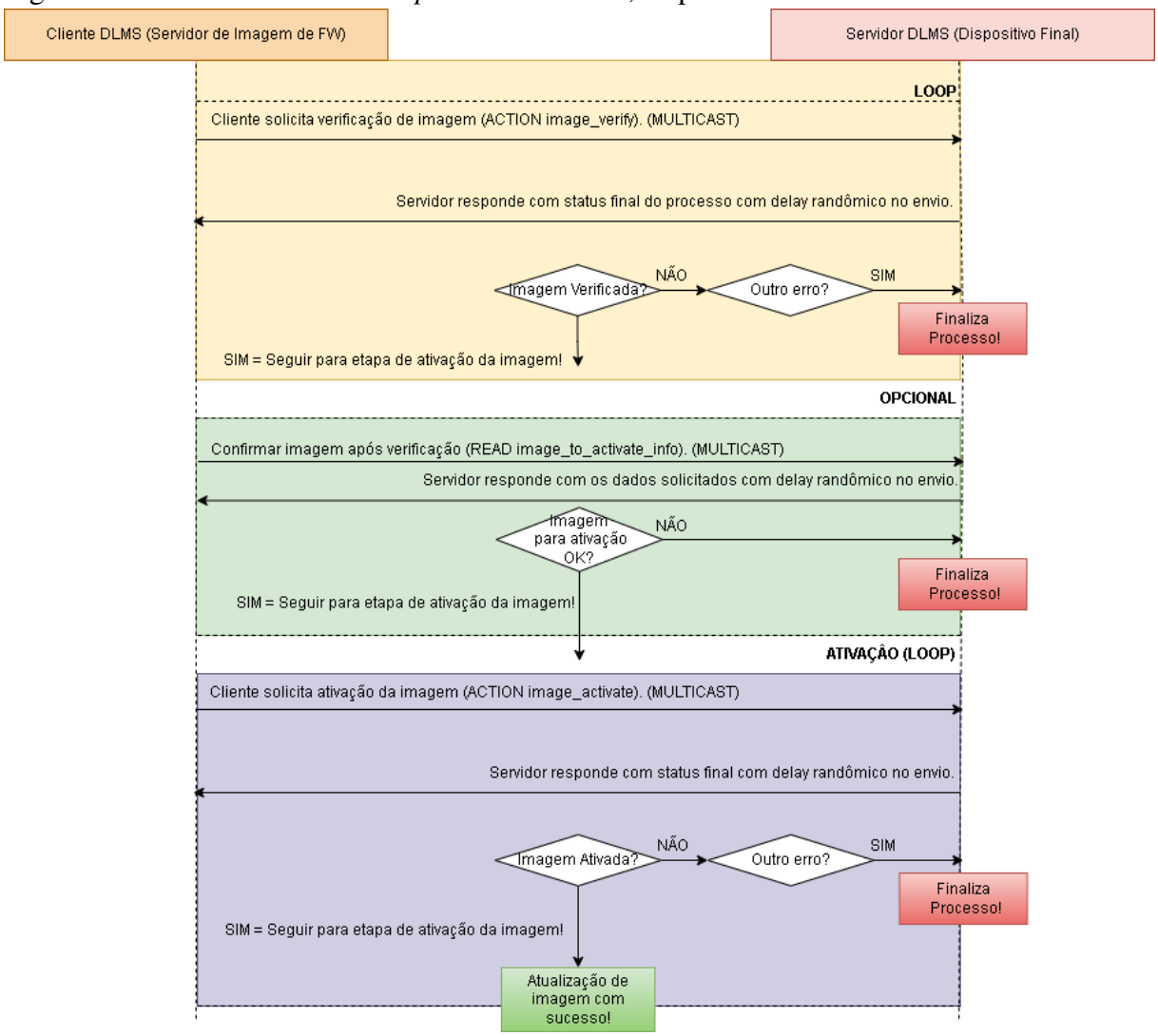
Para definir quais materiais seriam adotados para uso nos experimentos, pesou sobre a decisão o fato de o autor ter trabalhado na implementação do padrão apresentado para uma empresa cliente. A empresa atualmente desenvolve uma série de produtos de medição de água, luz, gás, além de outras soluções de internete das coisas que fazem uso de microcontroladores de arquitetura *ARM*, mais especificamente, fabricadas pela *ST Microelectronics*. Desta forma foram usadas placas desenvolvidas para aplicações finais que são apresentadas no Apêndice A. Vale ressaltar que as placas utilizadas são de dispositivos em produção, ou seja, são produtos que atualmente são comercializados e que possuem algumas centenas de unidades ou dezenas de milhares de dispositivos instalados em campo (dependendo da solução), o que fortalece a capacidade da implementação apresentada.

Figura 12 – Fluxo do *Firmware Update* modificado, etapas de 1 a 4.



Fonte: Figura do autor.

Figura 13 – Fluxo do *Firmware Update* modificado, etapas de 5 a 7.



Fonte: Figura do autor.

3.4 Avaliação/Análise

Durante o texto foi relatado a falta de padronização no processo de atualização de *firmware*, cada fabricante tem a liberdade para criar seu próprio protocolo, a *LoRa Alliance* em 2019 tomou a iniciativa de especificar um modelo, alguns *players* como *SENET e MURATA* já incorporam este modelo em suas interfaces.

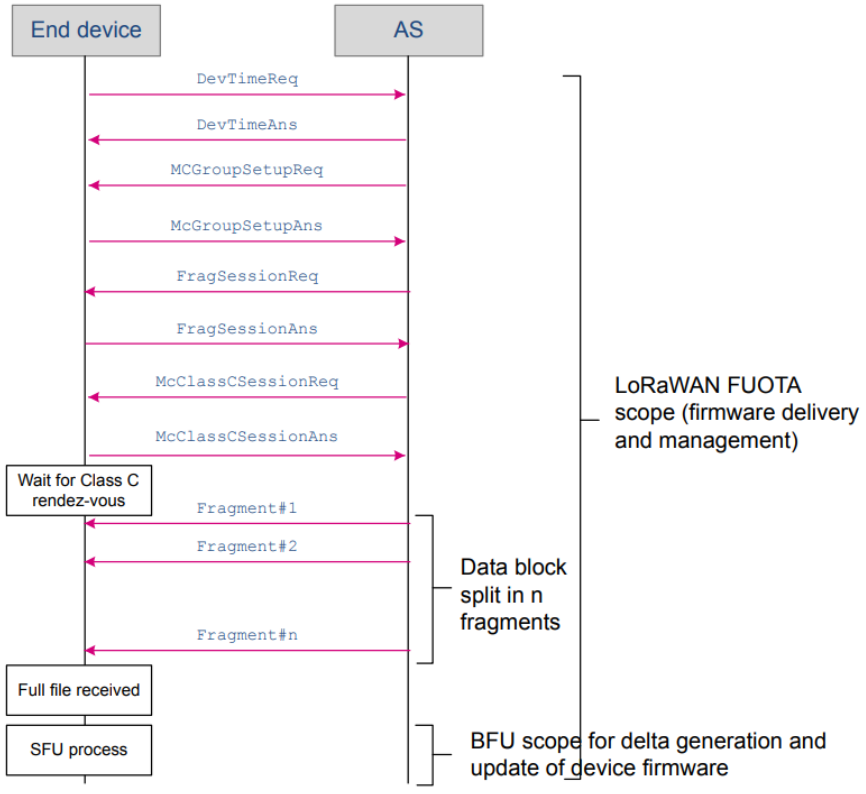
A própria *ST Microelectronics* já fornece *frameworks* que seguem o modelo citado para algumas famílias de controladores, como é o caso da família *STM32WL*, a biblioteca fornecida implementa funcionalidades já apresentadas anteriormente, como as especificações de *multicast e clock sync*, agregando mais recentemente as funcionalidades da especificação *Fragmented Data Block Transport*, responsável pelo gerenciamento da transmissão da imagem. Esta biblioteca fornece ainda um *bootloader* que alem de gerenciar a inicialização da aplicação

principal, pode processar a substituição da imagem por uma nova após o processo de transmissão.

Todo o processo é apresentado em *application notes*, para os dispositivos da família *WL* por exemplo, é disponibilizado o documento *AN5554 - LoRaWAN® firmware update over the air with STM32CubeWL*, que explica como utilizar o modelo geral do *FUOTA*, fornecendo os componentes necessários para a execução.

Uma consideração importante que deve ser feita, refere-se ao fato de que até o momento, apenas foi descrito o processo de atualização propriamente dito, considerando que todas as configurações relacionadas a classe do dispositivo *LoRa* e configuração do grupo *multicast* já foram realizadas previamente. O *application note* citado aborda esta configuração (e pode ser tomada como referência para mais detalhes). Para os ensaios serão considerados dispositivos configurados como classe C, mas o modelo apresentado não impede que dispositivos classe A e B possam usar o mesmo método, visto que para a recepção de blocos, os dispositivos mudam para classe C para a transmissão da imagem em modo contínuo.

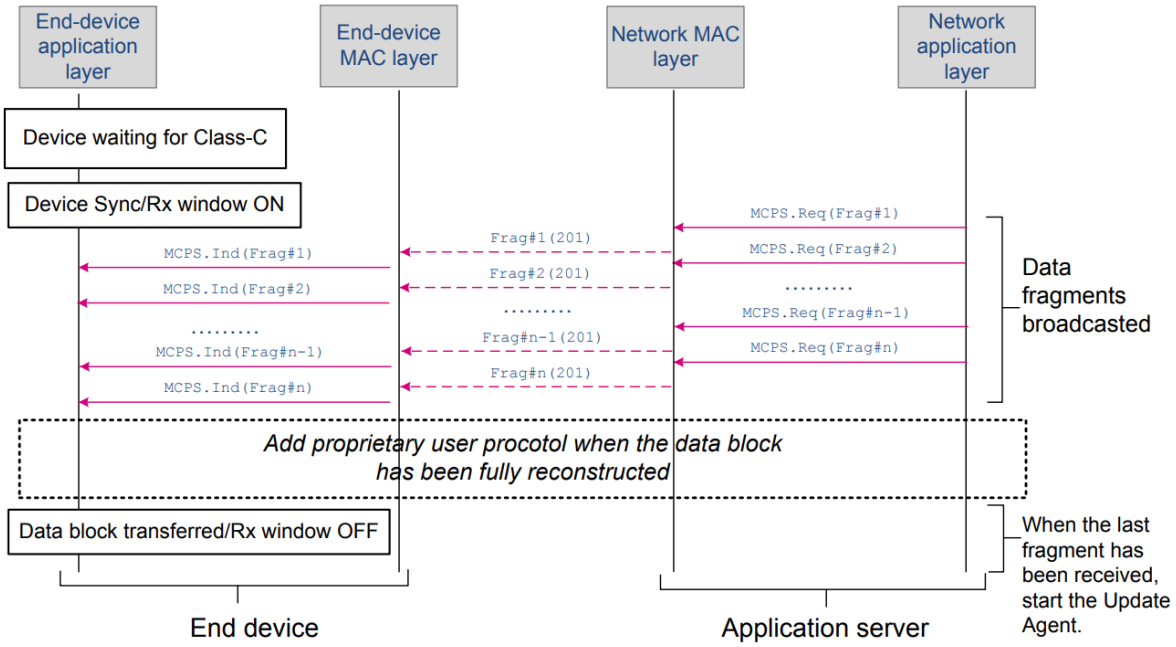
Figura 14 – Fluxo completo do Firmware Update



Fonte: ST Microelectronics, AN5554.

Na Figura 14 é possível observar o fluxo de atualização de *firmware* executado pelo *framework* da *ST*, neste pode ser observado que as configurações de rede para possibilitar

Figura 15 – Fluxo de transmissão de blocos para Firmware Update



Fonte: ST Microelectronics, AN5554.

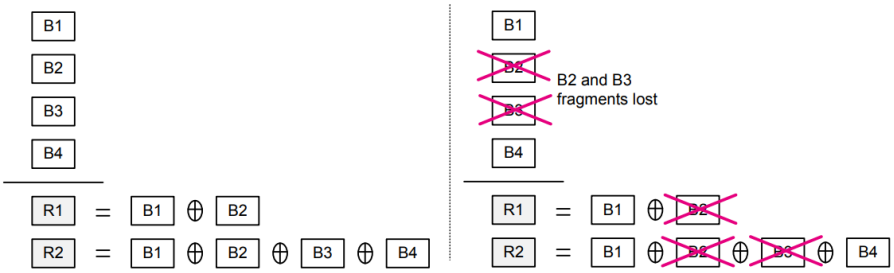
comandos *multicast* e mudança para classe C são as etapas iniciais (o padrão apresentado pressupõe que esta configuração foi realizada previamente, dado que o foco é o processo de transmissão e recuperação de blocos, culminando com a ativação da imagem) seguidas da transmissão da imagem e ativação da mesma.

Na Figura 15 é possível ter uma visão micro do processo de transmissão de blocos (principal diferenciação do padrão proposto para o protocolo agora apresentado, e tantos outros do mercado), a fragmentação é realizada pela própria pilha fornecida, todavia se faz necessário um protocolo proprietário adicional, visto que a especificação *Fragmented Data Transport specification* não fornece uma maneira de informar ao servidor que todos os blocos foram transmitidos corretamente.

Como é inerente as tecnologias *LPWAN* a possibilidade de perda de pacotes (seja por atrasos, pacotes corrompidos, indisponibilidade de *gateways*, etc), o protocolo agrega um algoritmo de redundância como ferramenta para garantir a recepção da imagem completa. Desta forma após a transmissão completa, o dispositivo mantém sua janela de recepção aberta para receber um número "X"de blocos de redundância. Assim, após uma sessão de *FUOTA* o dispositivo final pode tentar montar a imagem completa usando os fragmentos de redundância recebidos.

O servidor de aplicação é o responsável por fragmentar o *firmware* a ser enviado, assim como gerar os fragmentos de redundância. O número máximo de fragmentos de redundância

Figura 16 – Princípio do algoritmo de reconstrução



Fonte: ST Microelectronics, AN5554.

pode ser equivalente ao número total de fragmentos da imagem, limitar a uma quantidade menor de fragmentos de redundância é de responsabilidade do servidor de aplicação, alertando-se que quanto menor a quantidade de fragmentos de redundância, menores serão as chances de que o dispositivo consiga montar a imagem completa no fim do processo.

O documento fornecido pela *ST* fornece um algoritmo em linguagem *Python* que gera 100% de redundância, ou seja, o dobro de dados é enviado para que o dispositivo tenha maiores chances de sucesso no processo completo. Este algoritmo gera uma matriz de paridade onde cada fragmento de redundância é composto por alguns dos fragmentos enviados com operações *XOR* entre sí.

Na Figura 16 é possível ver um exemplo do algoritmo de reconstrução, neste processo a pilha faz uso do algoritmo *LDPC - Low-Density Parity-Check* (em português, código de verificação de paridade de baixa densidade), que está contido da especificação referente ao processo de fragmentação.

No exemplo 4 blocos são enviados (B1-B4) e posteriormente dois fragmentos de redundância (R1-R2). Na ocorrência de perda dos blocos B2 e B3, a redundância R1 contém B1 e B2, e R2 contém B1, B2, B3 e B4. Logo é possível obter B2 em uma operação *XOR* entre B1 e R1, assim como B3 pode ser obtido com uma operação *XOR* entre R1, R2 e B4.

É perceptível que o fragmento R1 é uma operação *XOR* simples e requer pouco tempo computacional para resolver se comparado ao segundo. Este é um fator importante no processo, um algoritmo que pode aumentar bruscamente a complexidade das operações pode se tornar um impeditivo para dispositivos com limitações de poder de processamento, memória e energia.

Neste caso específico temos a complexidade θn^2.

Diversos outros fatores podem ser levantados em relação ao processo, principalmente quando imagens de *firmware* podem alcançar algumas centenas de *kilobytes*, dividir esta imagem

em pequenos blocos de 32 *bytes* (devido limitações da rede LoRaWAN) faz com que se obtenha alguns milhares de fragmentos a serem transmitidos, este valor multiplicado por 2, devido aos fragmentos de redundância.

Por fim, vale salientar, que mesmo após todo o processo, existe a possibilidade de o dispositivo não conseguir recuperar a imagem completa, algo completamente possível em um caso real, se ocorrer uma alta perda de pacotes (ocasionado por problemas técnicos por exemplo) e no pior dos casos, os fragmentos de redundância que continham o bloco necessário também terem sido perdidos. Como já citado, o protocolo não fornece um modo de que o servidor conheça o *status* da imagem no dispositivo final, logo todo o processo deve ser reinicializado o que gera altos custos de consumo de dados, gasto de energia do dispositivo (problema crítico quando dispositivos de classe A ou B) e tempo.

3.5 Métricas

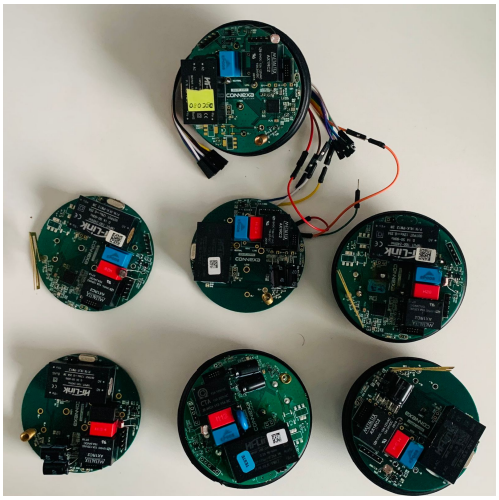
Com o dispositivo programado com a implementação do código, devidamente cadastrado no servidor, alguns ensaios iniciais podem ser realizados em busca de responder alguns questionamentos relativos a efetividade da proposta. Para responder a estes questionamentos algumas métricas podem ser utilizadas, algumas métricas podem ser mais adequadas para um tipo de tecnologia de comunicação, mas não muito conveniente para outras tecnologias. Como os ensaios foram focados no uso da rede *LoRa* as seguintes métricas foram listadas.

- Total de *downlinks*;
- Total de *uplinks*;
- Taxa de eficácia do processo de atualização;
- Tempo total do processo de atualização.

Cada experimento foi realizado 10 vezes inicialmente com 1 dispositivo, posteriormente com 3 dispositivos e finalmente com 7 dispositivos, na Figura 17 é possível observar os dispositivos selecionados para os ensaios. Os dados obtidos serão apresentados em uma tabela comparativa entre os algoritmos avaliados para posterior análise dos resultados.

Embora não exista um protocolo bem definido para atualização de firmware da aplicação com uso de redes *LTE* (*CAT-M, NB-IoT*), como ocorre com a especificação *LoRaWAN*, alguns ensaios foram realizados com dispositivos que embarcam módulos de tecnologias *LTE* com o objetivo inicial de demonstrar a interoperabilidade do padrão, agregando total independência da mídia física para transmissão da imagem. Na Figura 18 é possível observar o dispositivo

Figura 17 – Dispositivos utilizados nos ensaios.



Fonte: Figura do autor.

utilizado, este dispositivo faz uso de um microcontrolador *STM32WB55* e na placa podemos destacar como principal diferença em relação a versão *LoRaWAN*, o módulo *NB-IoT* do fabricante *TELIT* e uso de antena externa para garantir melhor captação do sinal *LTE*.

Figura 18 – Fotocélula NB-IoT.

Fonte: Figura do autor.

Outro ponto que foi verificado com os ensaios com tecnologia *LTE* foram relacionadas a possibilidade de configuração do tamanho máximo do bloco, já que diferente da tecnologia *LoRa* que se limita a 51 *bytes* por *downlink*, com as tecnologias *LTE* é possível fazer envio de pacotes de até 1,5 *KB*. O que se espera demonstrar é que, com a flexibilidade ofertada pelo protocolo, as configurações permitem otimizar o uso da rede com significativa redução do tempo de atualização.

4 RESULTADOS

Neste capítulo são apresentados os resultados dos ensaios realizados para validar a efetividade do protocolo proposto. Para estes ensaios foi gerado um *firmware* denominado *blink_led_v1r0*, esta aplicação como o próprio nome determina, irá apresentar um padrão de alteração pré-determinado de alteração no *led* do dispositivo. Esta imagem gerada tem aproximadamente 10 KB de tamanho (mais precisamente 10.052 *bytes*), já incluso a assinatura adicionada no fim da imagem. O processo utilizando a rede *LoRa* foi configurado com tamanho de bloco de 32 *bytes*, logo 315 blocos devem ser agendados para que a imagem completa seja transferida.

Visando inicialmente avaliar o tempo total de atualização (seja com sucesso ou falha) dos dispositivos, foram realizados 10 ensaios com 1, 3 e 7 dispositivos, os ensaios são melhores descritos a seguir:

- Ensaio 1 - uso do *framework* disponibilizado pela *ST* com algoritmo de redundância;
- Ensaio 2 - uso do algoritmo padrão proposto na especificação *DLMS*;
- Ensaio 3 - uso do algoritmo *DLMS* modificado conforme proposta.

Na Tabela 1 é possível observar os resultados dos primeiros ensaios realizados com o objetivo inicial de avaliar o ganho em tempo no processo de atualização. No uso do *framework* é possível observar que o acréscimo de dispositivos no processo de atualização não tende a incrementar significantemente o tempo total de atualização, dado que sempre a quantidade de blocos transmitidos serão iguais, portanto a variação tende a ser relacionado a alguma indisponibilidade ou atrasos nas solicitações ao servidor *LNS*. No uso do padrão *DLMS* é perceptível que existe um incremento no tempo total de atualização conforme mais dispositivos foram intruduzidos nos ensaios posteriores, isso se deve ao fato de as perdas de pacotes que ocasionalmente ocorrem, na etapa de recuperação serem realizadas de maneira individual. Por fim, no uso do padrão *DLMS* modificado, o incremento no tempo total de atualização conforme

Tabela 1 – Tabela de comparação entre algoritmos de atualização de firmware, tempo total de atualização em minutos.

Algoritmo	1 Dispositivo	3 Dispositivos	7 Dispositivos
Framework	109m	112m	111m
DLMS	59m	71m	98m
DLMS Modificado	59m	61m	67m

Tabela 2 – Tabela de comparação entre algoritmos de atualização de firmware, efetividade da atualização.

Algoritmo	1 Dispositivo	3 Dispositivos	7 Dispositivos
Framework	1S	3S	2S 5F
DLMS	1S	3S	7S
DLMS Modificado	1S	3S	7S

Tabela 3 – Tabela de comparação entre algoritmos de atualização de firmware, total de downlinks.

Algoritmo	1 Dispositivo	3 Dispositivos	7 Dispositivos
Framework	637	632	635
DLMS	353	426	576
DLMS Modificado	354	367	402

novos dispositivos são adicionados ao processo é bem menor se comparado ao algoritmo anterior, isso se deve ao fato do processo de recuperação de blocos ser otimizado, onde se reenvia *multicast* os blocos perdidos relatados por um dos dispositivos do grupo, restando poucos pacotes a serem recuperados aos demais dispositivos.

A partir dos ensaios anteriores ainda foram possíveis obter resultados de efetividade do processo, ou seja, quantos dispositivos foram atualizados com sucesso (S) ou falha (F). Na Tabela 2 é possível observar que os resultados foram iguais para os três algoritmos ao atualizar 1 ou 3 dispositivos simultaneamente, todavia, quando o processo foi realizado com 7 dispositivos, o *framework* não obteve bom resultado, conseguindo atualizar com sucesso apenas 30% dos dispositivos. É perceptível que sem um algoritmo de confirmação da completude da imagem implementada adicionalmente sobre a camada do *framework*, conforme novos dispositivos forem adicionados, a taxa de sucesso tende a diminuir. De maneira diferente, ao usar o *DLMS* obtem-se a vantagem de já existir um algoritmo bem definido e seguro que permite alcançar o sucesso de todos os dispositivos, exceto quando algum evento adverso relacionado ao dispositivo ou a rede impedir que o processo seja concluído.

Na Tabela 3 é possível observar os resultados relacionados ao consumo de dados, esta tabela é importante para mensurar o custo de manutenção da rede, dado que na contratação de dados, o custo de *uplink* tende a ser bem pequeno em relação ao de *downlink*, logo reduzir a quantidade de *downlinks* é mais relevante para o processo de atualização. Nesta tabela é perceptível que no uso do *framework* a variação é insignificante, podendo ser atrelado a problemas ao solicitar o agendamento de *downlink* junto ao *LNS*. No uso do *DLMS* padrão,

Tabela 4 – Tabela de comparação entre algoritmos de atualização de firmware, total de uplinks.

Algoritmo	1 Dispositivo	3 Dispositivos	7 Dispositivos
Framework	0	0	0
DLMS	45	122	290
DLMS Modificado	44	82	150

Tabela 5 – Tabela Comparativa - Ganhos com a implementação do DLMS modificado (ensaio com 7 dispositivos).

||	Taxa de sucesso	Total de Uplinks	Total de Downlinks	Tempo total
Framework	+250%	0%	-36,69%	-39,64%
DLMS	0%	-48,28%	-30,21%	-31,63%

percebe-se que existe um incremento na quantidade de *downlinks* conforme mais dispositivos são inseridos no processo de atualização. No uso do *DLMS* modificado existe o incremento na quantidade total de *downlinks*, mas o valor tende a crescer de maneira mais equilibrada mesmo com o acréscimo de novos dispositivos.

Por fim ainda foi possível obter informações a respeito do total de *uplinks* necessários para o total processo de atualização com cada um dos três algoritmos. Na Tabela 4 é possível observar que o *framework* não possui uma quantidade de *uplinks* computada, isso se deve ao fato de não existir um algoritmo implementado para verificação da completude da imagem ou envio de fragmentos faltantes (ficando a cargo do desenvolvedor inserir esta camada por conta própria), logo, se a imagem não foi montada completamente mesmo com o algoritmo de reconstrução, o processo termina com falha conforme já observado.

Na Tabela 4 ainda é possível observar que nas duas variações do *DLMS* conforme se insere novos dispositivos, ocorre um incremento na quantidade de *uplinks* para que a atualização de *firmware* ocorra com sucesso, todavia, assim como ocorre na Tabela 3, este incremento é mais discreto na execução do *DLMS* modificado, devido a otimização na etapa de recuperação de blocos que reduz a quantidade de solicitações questionando os blocos de imagem que não foram recebidos.

Na Tabela 5 é possível observar os ganhos obtidos com o uso da proposta, foi possível alcançar uma redução de 30 a 40% no tempo total do processo de atualização (tomando como base o tempo até q o último dispositivo seja atualizado ou falhe). Ainda foi possível observar uma redução de 33% em média no consumo de dados tomando como referência a quantidade de *downlinks*, e quase 50% de redução no consumo de *uplinks* quando se compara a proposta ao

Tabela 6 – Tabela de resultados, firmware update com uso do NB-IoT (bloco = 256 bytes).

Métrica	Resultado Obtido
Uplinks	14
Downlinks	48
Tempo Total	16m

Tabela 7 – Tabela de resultados, firmware update com uso do NB-IoT (bloco = 1024 bytes).

Métrica	Resultado Obtido
Uplinks	11
Downlinks	21
Tempo Total	7m

processo padrão de atualização. Quanto a taxa de sucesso, embora não seja possível observar ganhos no uso da proposta quando comparada ao processo padrão de atualização, estima-se que os resultados sejam significativos com grandes quantidades de dispositivos, considerando-se experiências não relatadas, mas acumuladas durante o desenvolvimento de projetos para empresas clientes.

Realizou-se ainda um ensaio com a rede *NB-IoT*, visando demonstrar a adaptabilidade do padrão apresentado a qualquer tipo de tecnologia selecionada, se realizou o processo de atualização de *firmware* com um dispositivo (devido as limitações de placas e planos de dados disponíveis), mas buscou-se demonstrar que a personalização das configurações possíveis, permitiram o uso dos potenciais da tecnologia reduzindo significativamente o tempo total de atualização.

Na Tabela 6 é possível observar os resultados obtidos no processo de atualização de firmware de um dispositivo via *NB-IoT* com o *DLMS* modificado e com um tamanho de bloco limitado a 256 *bytes* (a tecnologia permite pacotes de até 1,5 KB). É possível observar que com apenas 48 *downlinks* foi possível enviar a imagem completa, contra 350 em média na tecnologia *LoRa*. Da mesma forma, os *uplinks* necessários para todo o processo (incluindo a recuperação de blocos) reduziram significantemente correspondendo a cerca de 25% de *uplinks* necessários com a outra tecnologia analisada. Por fim, como esperado, o tempo total reduziu drasticamente atingindo a marca de 16 minutos conta aproximadamente 1 hora via rede *LoRa*.

Por fim, no último experimento, para demonstrar que os resultados ainda poderiam ser melhores se as configurações fossem otimizadas para a tecnologia utilizada atualmente,

reconfigurando o tamanho de bloco para 1024 bytes, foi possível reduzir para menos da metade a quantidade de tempo necessário para a atualização completa da imagem de firmware do dispositivo, com significativa redução na quantidade de *downlinks*, todavia, vale salientar que neste tipo de tecnologia, a precificação de consumo de dados não ocorre por quantidade de *downlinks/uplinks*, mas pela quantidade de dados trafegados, logo, o ganho efetivo é relacionado apenas ao tempo de execução. Estes resultados podem ser observados na Tabela 7.

5 CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação se propôs a responder duas questões principais:

1 - É possível padronizar o processo de atualização de *firmware* pelo ar? Com base no dados apresentados, pode-se concluir que sim, é possível ter uma padronização no processo de atualização de firmware não só pelo ar, mas em qualquer meio de comunicação. O padrão *DLMS* se mostra muito robusto, não à toa é um protocolo implementado em mais de 1000 dispositivos certificados além de ser usado em mais de 60 países. O padrão permite não apenas atualizar o próprio dispositivo, mas também dispositivos terceiros contanto que ambos utilizem o mesmo padrão reforçando a padronização no processo. Além dos pontos listados ainda pode ser levantado como diferencial a fácil configuração de dispositivos de segurança permitindo que o dispositivo atenda este requisito essencial atualmente.

2 - Com esta padronização algum diferencial da tecnologia de comunicação utilizada pode ser subutilizada, reduzindo as vantagens da padronização? Durante os ensaios também foi perceptível que por conta da adaptabilidade do protocolo é possível extrair o máximo da tecnologia de comunicação selecionada, logo dentre as vantagens de padronizar pode ser ressaltado benefícios como: menor tempo de desenvolvimento, menor custo de desenvolvimento, menor custo de manutenção, previsibilidade e maximização dos potenciais da rede no processo devido a configuração personalizada.

Para responder as questões anteriores três algoritmos foram comparados, o primeiro baseado na especificação publicada pela *LoRa Alliance*, o segundo baseado na especificação do padrão *DLMS* e o terceiro uma proposta de melhoria no fluxo do anterior, agregando maior robustez e opções de configuração que proporcionaram economia de dados e de tempo na execução do processo de atualização.

Os ensaios com uso do primeiro algoritmo baseado na especificação não foram totalmente conclusivos, já que o algoritmo não apresenta um método de confirmação que a imagem foi transmitida e em casos em que a rede apresente grande instabilidade é bem provável que o processo falhe dado que o dispositivo não será capaz de remontar a imagem. Uma implementação que preenchesse esta lacuna poderia ter sido realizada, mas este não era o objetivo da dissertação, dado que o *framework* utilizado é bastante engessado não permitindo fácil portabilidade para outros controladores de outras famílias do mesmo fabricante, muito menos para controladores de terceiros. Ponto negativo em relação ao módulo *DLMS* que é facilmente portável para qualquer controlador dado que o protocolo propriamente dito abstrai as

camadas de hardware, bastando que os *drivers* para comunicação e uso de flash sejam reescritos respeitando a regra de instanciação dos mesmos.

O uso do padrão *DLMS* já garante um grande avanço, dada a robustez do protocolo. É perceptível nos resultados que a taxa de sucesso no processo de atualização de *firmware* não se altera (ao menos com pequenas amostras) ao usar o protocolo padrão ou modificado, todavia o consumo de dados e o tempo total de execução sofreram uma redução significativa de até 36% e 39% respectivamente.

Ainda relacionado a comparação das duas versões do protocolo *DLMS*, pode se estimar que as alterações podem surtir um grande efeito relacionado a taxa de sucesso na execução da atualização de grandes quantidades de dispositivos simultaneamente (leia-se centenas ou milhares), dado que sem o uso de comandos *multicast* com respostas com atraso, a possibilidade de ocasionar gargalos na rede cresce o que levari muitos dispositivos a falhar no processo por aparente falta de resposta (em qualquer implementação é necessário estipular uma quantidade máxima de retentativas para evitar gastos desnecessários de recursos).

Como os ensaios foram executados principalmente com uso da tecnologia *LoRa*, foi possível entender os principais fundamentos desta tecnologia elencando seus pontos fortes e fracos, permitindo um maior entendimento sobre o funcionamento não só desta tecnologia, mas também sobre os fundamento da Internet das Coisas.

5.1 Trabalhos Futuros

A partir da problematização apresentada e dos ensaios realizados alguns trabalhos em potencial podem ser vislumbrados como pesquisas futuras.

5.1.1 Delta Update

O *firmware update delta* é uma atualização que consiste na transmissão apenas das partes da imagem que sofreram alterações, não sendo necessário a transmissão completa para atualizar o dispositivo. Este tipo de atualização pode promover significativa redução no tempo total do processo além de reduzir o consumo de dados.

5.1.2 LoRa

As estratégias a seguir referem-se exclusivamente a dispositivos com tecnologia *LoRa*.

5.1.2.1 Dispositivo Classe B

Os ensaios foram realizados com dispositivos classe C, como exposto no decorrer da dissertação a configuração de uma sessão *multicast* pode ser para um grupo classe B ou C. Quando o dispositivo fica constantemente em classe C o consumo de energia é um pouco maior devido a janela de recepção ficar constantemente aberta. No caso dos dispositivos classe B, além das janelas RX1 e RX2 o dispositivo pode sair do modo de baixo consumo em horários programados para abrir nova janela de recepção. Validar este processo com dispositivos classe A que mudam para classe B durante o processo de atualização avaliando o consumo de energia é um estudo válido para reforçar a versatilidade do modelo proposto.

Esta é uma estratégia que pode ser combinada ao *firmware update delta* para dispositivos com fonte de energia limitada, otimizando o processo e garantindo ainda uma boa vida útil da bateria.

5.1.2.2 Retransmissão de pacotes entre dispositivos

Por fim, uma última estratégia que pode ser favorável aos dispositivos classe C (que não possuem restrição de energia) é o processo de retransmissão de blocos perdidos entre dispositivos. Esta estratégia pode permitir uma significativa redução no consumo de dados quando se faz a contratação de planos de dados de uma rede pública. Como a frequência de *Sub-GHz* utilizada pela tecnologia *LoRa* é não licenciada, ainda existem faixas de frequência que podem ser usadas para esta comunicação entre dispositivos sem afetar a rede a qual fazem parte.

Com estas considerações, percebe-se que ainda existe muito a ser explorado nesta área para conceber um mecanismo totalmente funcional e padronizado para atualização de dispositivos, espera-se que esta dissertação possa contribuir com outras pesquisas dentro do mesmo escopo permitindo a evolução destas aplicações no mercado.

REFERÊNCIAS

ABDELFADEEL, K.; FARRELL, T.; MCDONALD, D.; PESCH, D. How to make firmware updates over lorawan possible. In: INTERNATIONAL SYMPOSIUM ON A WORLD OF WIRELESS, MOBILE AND MULTIMEDIA NETWORKS (WOWMOM), 21., 2020, Pisa. **Proceedings**. Pisa: IEEE, 2020. p. 16–25. Disponível em: <https://ieeexplore.ieee.org/document/9217759>. Acesso em: 10 mar. 2022.

ADELANTADO, F.; VILAJOSANA, X.; TUSET-PEIRO, P.; MARTINEZ, B.; MELIA-SEGUI, J.; WATTEYNE, T. Understanding the limits of lorawan. **IEEE Communications magazine**, IEEE, New York, v. 55, n. 9, p. 34–40, 2017.

AYOUB, W.; SAMHAT, A. E.; NOUVEL, F.; MROUE, M.; PRéVOTET, J.-C. Internet of mobile things: Overview of lorawan, dash7, and nb-iot in lpwans standards and supported mobility. **IEEE Communications Surveys Tutorials**, New York, v. 21, n. 2, p. 1561–1581, 2019.

BAGHYALAKSHMI, D.; EBENEZER, J.; SATYAMURTY, S. A. V. Wsn based temperature monitoring for high performance computing cluster. In: **2011 International Conference on Recent Trends in Information Technology (ICRTIT)**. [S.l.: s.n.], 2011. p. 1105–1110.

BARR, M. **Embedded C Coding Standard**. CreateSpace Independent Publishing Platform, 2018. ISBN 9781721127986. Disponível em: <https://books.google.com.br/books?id=XX3FugEACAAJ>. Acesso em: 23 jan. 2022.

BORGES, L. M.; VELEZ, F. J.; LEBRES, A. S. Survey on the characterization and classification of wireless sensor network applications. **IEEE Communications Surveys Tutorials**, New York, v. 16, n. 4, p. 1860–1890, 2014.

CATALANO, J.; COUPIGNY, J.; KUYPER, M.; SORNIN, N.; YEGIN, A. **LoRaWAN Remote Multicast Setup Specification v1. 0.0**. LoRa Alliance, 2018. Disponível em: <https://lora-alliance.org/wp-content/uploads/2020/11/remote_multicast_setup_v1.0.0.pdf>. Acesso em: 31 jan. 2022.

CATALANO, J.; COUPIGNY, J.; KUYPER, M.; SORNIN, N.; YEGIN, A. **Fuota process summary technical recommendation**. LoRa Alliance, 2019. Disponível em: <https://lora-alliance.org/wp-content/uploads/2020/11/tr002-fuota_process_summary-v1.0.0.pdf>. Acesso em: 30 jan. 2022.

DOUGLASS, B. **Design Patterns for Embedded Systems in C: An Embedded Software Engineering Toolkit**. Newnes/Elsevier, 2011. ISBN 9781856177078. Disponível em: <https://books.google.com.br/books?id=8cgNlQEACAAJ>.

DVORNIKOV, A.; ABRAMOV, P.; EFREMOV, S.; VOSKOV, L. Qos metrics measurement in long range iot networks. In: CONFERENCE ON BUSINESS INFORMATICS (CBI), 19., 2017, Thessaloniki. **Proceedings**. Thessaloniki: IEEE, 2017. p. 15–20. Disponível em: <https://ieeexplore.ieee.org/document/8012393>. Acesso em: 12 mar. 2022.

EGGERS, P.; KOVACS, I.; OLESEN, K.; KUIJPERS, G. Measurements of wideband multi-element transmit-receive diversity channels in the umts-band. In: VEHICULAR TECHNOLOGY CONFERENCE, 52., 2020, Båstad. **Proceedings**. Båstad: IEEE, 2020. p. 1683–1689. Disponível em: <https://ieeexplore.ieee.org/document/886112>. Acesso em: 16 mar. 2022.

FUOTA-WORKIN-GROUP; COMMITTEE, L. A. T. *et al.* Lorawan application layer clock synchronization specification v1.0.0. **Lora Alliance**, 2018.

FUOTA-WORKING-GROUP, o. t. L. A. T. C. **LoRaWAN Fragmented Data Block Transport Specification**. [S.l.]: LoRa Alliance Fremont, CA, USA, 2018.

GRAMMATIKIS, P. R.; SARIGIANNIDIS, P.; MOSCHOLIOS, I. Securing the internet of things: challenges, threats and solutions. **Internet Things**, Amesterdã, v. 5, n. 1, p. 41–70, 2019.

JONGBOOM, J.; STOKKING, J. Enabling firmware updates over lpwans. In: EMBEDDED WORLD CONF., 16., 2018, Nuremberg. **Proceedings**. Nuremberg: Embedded World EU, 2018. Disponível em: <http://janjongboom.com/downloads/ew2018-paper.pdf>. Acesso em: 20 mar. 2022.

KIM, E.-H.; KANG, H.; PYO, C. S.; KIM, C. A cmos single-chip transceiver design for 700/800/900mhz wireless sensor network applications. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY CONVERGENCE (ICTC), 5., 2014, Busan. **Proceedings**. Busan: IEEE, 2014. p. 699–700. Disponível em: <https://ieeexplore.ieee.org/document/6983260>. Acesso em: 10 mar. 2022.

LLVM-PROJECT. **LLVM language reference manual for LLVM**. 2022. Disponível em: <https://llvm.org/docs/CodingStandards.html>. Acesso em: 14 jan. 2022.

LORA-ALLIANCE. **LoRaWAN® distance world record broken, twice. 766 km (476 miles) using 25mW transmission power**. 2020. Disponível em: <https://lora-alliance.org/lorawan-news/lorawanr-distance-world-record-broken-twice-766-km-476-miles-using-25mw-transmission/>. Acesso em: 27 jan. 2022.

LU, X.; KIM, I. H.; XHAFA, A.; ZHOU, J.; TSAI, K. Reaching 10-years of battery life for industrial iot wireless sensor networks. In: SYMPOSIUM ON VLSI CIRCUITS, 21., 2017, Kyoto. **Proceedings**. Kyoto: IEEE, 2017. p. C66–C67. Disponível em: <https://ieeexplore.ieee.org/document/8008550>. Acesso em: 23 mar. 2022.

NGUYEN, T. H.; JUNG, W.-S.; TU, L. T.; CHIEN, T. V.; YOO, D.; RO, S. Performance analysis and optimization of the coverage probability in dual hop lora networks with different fading channels. **IEEE Access**, New York, v. 8, p. 107087–107102, 2020.

NGUYEN, T. T.; NGUYEN, H. H.; BARTON, R.; GROSSETETE, P. Efficient design of chirp spread spectrum modulation for low-power wide-area networks. **IEEE Internet of Things Journal**, New York, v. 6, n. 6, p. 9503–9515, 2019.

PETAJAJARVI, J.; MIKHAYLOV, K.; ROIVAINEN, A.; HANNINEN, T.; PETTISSALO, M. On the coverage of lpwans: range evaluation and channel attenuation model for lora technology. In: INTERNATIONAL CONFERENCE ON ITS TELECOMMUNICATIONS (ITST), 14., 2015, Copenhagen. **Proceedings**. Copenhagen: IEEE, 2015. p. 55–59. Disponível em: <https://ieeexplore.ieee.org/document/7377400>. Acesso em: 25 mar. 2022.

RABAY'A, A.; SCHLEICHER, E.; GRAFFI, K. Fog computing with p2p: Enhancing fog computing bandwidth for iot scenarios. In: INTERNATIONAL CONFERENCE ON INTERNET OF THINGS (ITHINGS), 12., 2019, Atlanta. **Proceedings**. Atlanta: IEEE, 2019. p. 82–89. Disponível em: <https://ieeexplore.ieee.org/document/8875437>. Acesso em: 10 fev. 2022.

RAZA, U.; KULKARNI, P.; SOORIYABANDARA, M. Low power wide area networks: An overview. **IEEE Communications Surveys Tutorials**, v. 19, n. 2, p. 855–873, 2017.

ROSE, K.; ELDRIDGE, S.; CHAPIN, L. **The Internet of Things an Overview**: Understanding the issues and challenges of a more connected world. The Internet Society (ISOC), 2015. Disponível em: <https://www.internetsociety.org/wp-content/uploads/2017/08/ISOC-IoT-Overview-20151221-en.pdf>. Acesso em: 21 jan. 2022.

ROUTH, K.; PAL, T. A survey on technological, business and societal aspects of internet of things by q3. In: INTERNATIONAL CONFERENCE ON INTERNET OF THINGS: SMART INNOVATION AND USAGES (IOT-SIU), 3., 2018, Bhimtal. **Proceedings**. Bhimtal: IEEE, 2018. p. 1–4. Disponível em: <https://ieeexplore.ieee.org/document/8519898>. Acesso em: 12 fev. 2022.

RUAN, T.; CHEW, Z. J.; ZHU, M. Energy-aware approaches for energy harvesting powered wireless sensor nodes. **IEEE Sensors Journal**, New York, v. 17, n. 7, p. 2165–2173, 2017.

SEMTECH-CORPORATION. **LoRa and LoRaWAN**: A technical overview. 2020. Disponível em: <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/>. Acesso em: 1 mar. 2022.

SINGH, S.; SINGH, N. Internet of things (iot): Security challenges, business opportunities amp; reference architecture for e-commerce. In: INTERNATIONAL CONFERENCE ON GREEN COMPUTING AND INTERNET OF THINGS (ICGCIOT), 1., 2015, Greater Noida. **Proceedings**. Greater Noida: IEEE, 2015. p. 1577–1581. Disponível em: <https://ieeexplore.ieee.org/document/7380718>. Acesso em: 15 fev. 2022.

SISINNI, E.; CARVALHO, D. F.; FERRARI, P. Emergency communication in iot scenarios by means of a transparent lorawan enhancement. **IEEE Internet of Things Journal**, New York, v. 7, n. 10, p. 10684–10694, 2020.

SMITH, B. **LoRaWAN and the Internet of Things**. 2017. Disponível em: <https://www.thethingsnetwork.org/community/pittsburgh/post/lorawan-and-the-internet-of-things>. Acesso em: 7 fev. 2022.

SUMAN, S.; PERUMAL, T.; MUSTAPHA, N.; YAAKOB, R. Device verification and compatibility for heterogeneous semantic iot systems. In: INTERNATIONAL CONFERENCE AND WORKSHOPS ON RECENT ADVANCES AND INNOVATIONS IN ENGINEERING (ICRAIE), 4., 2019, Kedah. **Proceedings**. Kedah: IEEE, 2019. p. 1–3. Disponível em: <https://ieeexplore.ieee.org/document/9037767>. Acesso em: 9 mar. 2022.

TALWANA, J. C.; HUA, H. J. Smart world of internet of things (iot) and its security concerns. In: INTERNATIONAL CONFERENCE ON INTERNET OF THINGS (ITHINGS), 9., 2016, Chengdu. **Proceedings**. Chengdu: IEEE, 2016. p. 240–245. Disponível em: <https://ieeexplore.ieee.org/document/7917092>. Acesso em: 5 mar. 2022.

URSUţIU, D.; NASCOV, V.; SAMOILă, C.; MOGA, M. Microcontroller technologies in low power applications. In: INTERNATIONAL CONFERENCE ON INTERACTIVE COLLABORATIVE LEARNING (ICL), 15., 2012, Villach. **Proceedings**. Villach: IEEE, 2012. p. 1–5. Disponível em: <https://ieeexplore.ieee.org/document/6402096>. Acesso em: 4 mar. 2022.

XIONG, X.; ZHENG, K.; XU, R.; XIANG, W.; CHATZIMISIOS, P. Low power wide area machine-to-machine networks: key techniques and prototype. **IEEE Communications Magazine**, New York, v. 53, n. 9, p. 64–71, 2015.

YASUMOTO, K.; YAMAGUCHI, H.; SHIGENO, H. Survey of real-time processing technologies of iot data streams. **Journal of Information Processing**, Tokyo, v. 24, n. 2, p. 195–202, 2016.

YIM, D.; CHUNG, J.; CHO, Y.; SONG, H.; JIN, D.; KIM, S.; KO, S.; SMITH, A.; RIEGSECKER, A. An experimental lora performance evaluation in tree farm. In: SENSORS APPLICATIONS SYMPOSIUM (SAS), 1., 2018, Seoul. **Proceedings**. Seoul: IEEE, 2018. p. 1–6. Disponível em: <https://ieeexplore.ieee.org/document/8336764>. Acesso em: 2 mar. 2022.

YU, Y.; MROUEH, L.; DUCHEMIN, D.; GOURSAUD, C.; VIVIER, G.; GORCE, J.-M.; TERRé, M. Adaptive multi-channels allocation in lora networks. **IEEE Access**, New York, v. 8, p. 214177–214189, 2020.

APÊNDICE A – MATERIAIS UTILIZADOS

A.1 Hardware

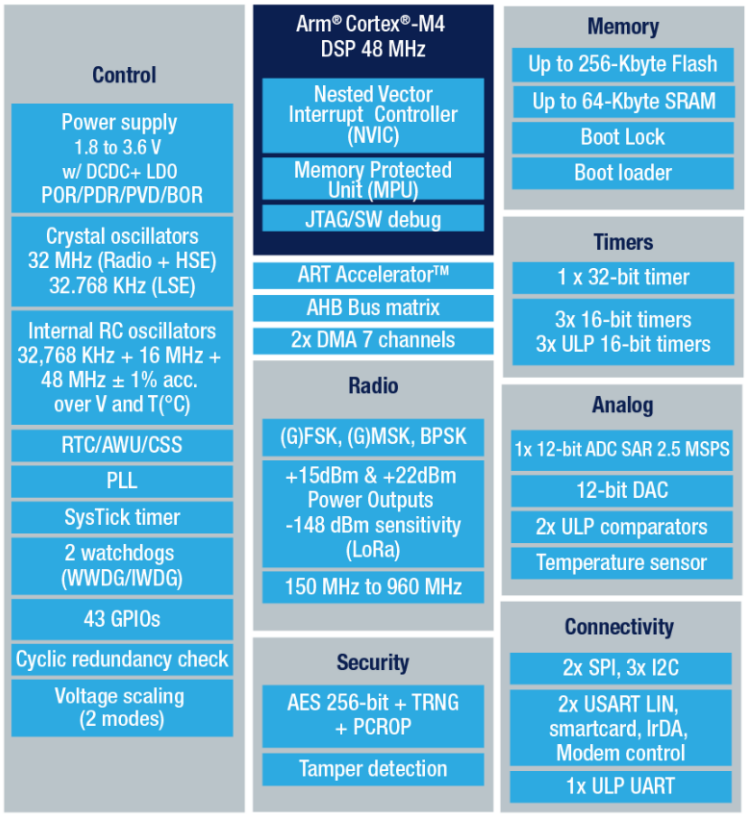
Do ponto de vista de previsibilidade, atualmente as soluções que embarcam o padrão apresentado fazem uso de controladores de arquitetura *ARM*, fabricado pela *ST Microelectronics*, das famílias *STM32WB, STM32WLE e STM32G0*. Em ordem de apresentação, tem-se um *system-on-chip* que integra um microcontrolador de alto desempenho de 2 núcleos (*M0+ e M4*) associado a um transceptor de radio *LoRa* com capacidade até 10x mais de armazenamento se comparado ao controlador da família *G0*. Em seguida tem-se um segundo *sistem-on-chip* de menor capacidade, com um núcleo *M4* e memória *flash* e *RAM* equivalente a 1/4 do anterior. Por fim, tem-se um microcontrolador *Cortex M0+* que dependendo da aplicação foi associado a um transceptor *LoRa* da linha *SX12xx* e em algumas aplicações que fazem uso de tecnologias *LTE* (*NB-IoT, CAT-M1*), foi associado a módulos de comunicação de fabricantes como *Thales*, *Telit* ou *Quectel*.

Para os experimentos realizados optou-se por escolher uma placa que embarca um *IC (Integrated Circuits) STM32WLE5CC*, este controlador atualmente é utilizado em 4 produtos na linha de produção da empresa cliente, como soluções para iluminação pública e medidores de flúidos. Este controlador mediano possui boas características de configuração, como pode ser observado no diagrama do circuito na figura 19.

Este controlador foi selecionado com a intenção que pudesse ser facilmente integrado a sistemas existentes para permitir que sensores e atuadores se comuniquem com uso da *LoRaWAN*. Além disso, dentre as características deste microcontrolador, ainda se pode destacar:

- Memória - com até 256 *KB* de armazenamento, é possível desenvolver um bootloader robusto para verificação da imagem com algoritmos de criptografia mais seguros, além de maior área para desenvolvimento da aplicação final e armazenamento de dados;
- Conectividade - possui diversas opções como *SPI, I2C* e serial, possibilitando o uso de outros circuitos integrados para as mais diversas implementações;
- Segurança - possui acelerador em *hardware* para algoritmos de criptografia como *AES 128/256*, possui detecção de adulteração para relógio;

Figura 19 – Diagrama do circuito - STM32WLE5CC



Fonte: ST Microelectronics, folha de dados.

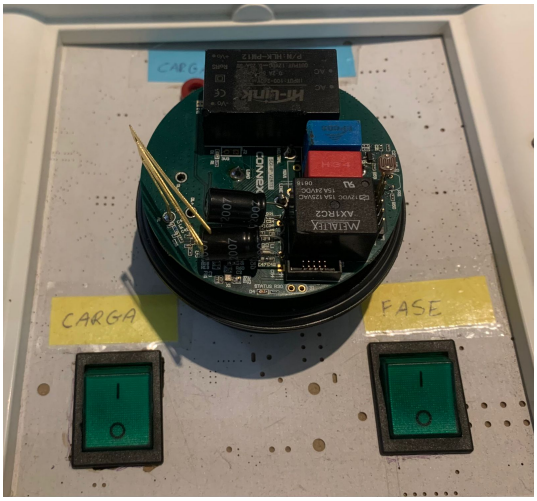
A.1.1 Dispositivo para ensaios

Para os ensaios deste trabalho foi selecionado uma solução desenvolvida para iluminação pública, este produto embarca um microcontrolador *STM32WLE* com transceptor *LoRa* embutido, seu firmware foi desenvolvido com base nos padrões já apresentados, possuindo em sua aplicação um servidor *DLMS* apto a gerenciar o processo de atualização de *firmwate*. Na figura 20 é possível ter uma visão superior do dispositivo sem encapsulamento, acoplado em uma base de testes que fornece a tensão de rede 110/220 V simulando uma luminária. Na figura 21 é possível observar o mesmo dispositivo agora com seu case de acrílico que fornece proteção contra água e poeira, além de atuar como difusor para o sensor de luminosidade utilizado para controle automatizado da luminária (dimerização).

A.2 IDE e ferramenta de DEBUG

Para desenvolvimento com os microcontroladores *ST*, a ferramenta mais atual disponibilizado pelo fabricante é o kit de desenvolvimento *STM32CUBE*, que possui a aplicação *MX* para rápida configuração de periféricos de *hardware* com interface gráfica amigável como pode

Figura 20 – Imagem superior do dispositivo fotocélula sem case de proteção.



Fonte: Figura do autor.

Figura 21 – Imagem frontal do dispositivo fotocélula com case de proteção.

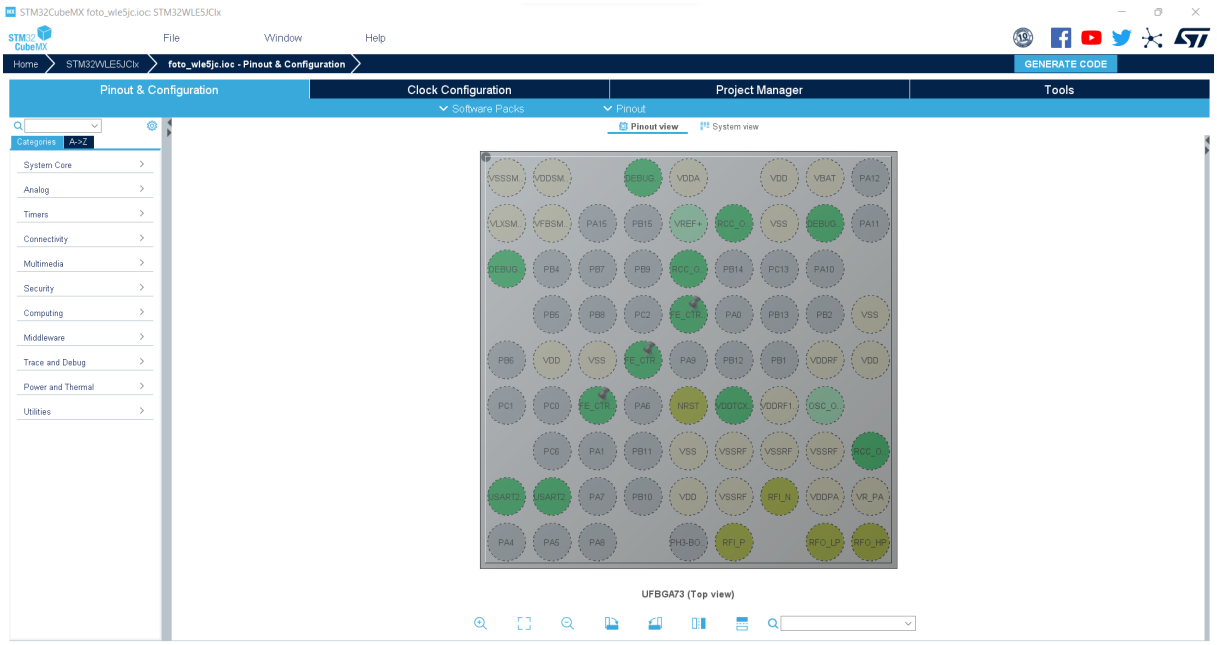
Fonte: Figura do autor.

ser observado na figura 22. Este software pode ser utilizado como base para a geração do projeto para desenvolvimento em uma série de *IDE's*, não se limitando a fornecida pelo fabricante.

A segunda aplicação utilizada é o *STM32CUBE IDE*, um ambiente de desenvolvimento integrado que permite maior facilidade para configuração do projeto gerado pela aplicação anterior. Nesta aplicação além do desenvolvimento, também é possível realizar o *debug* com a ferramenta necessária. Na figura 23 é possível observar a interface da aplicação.

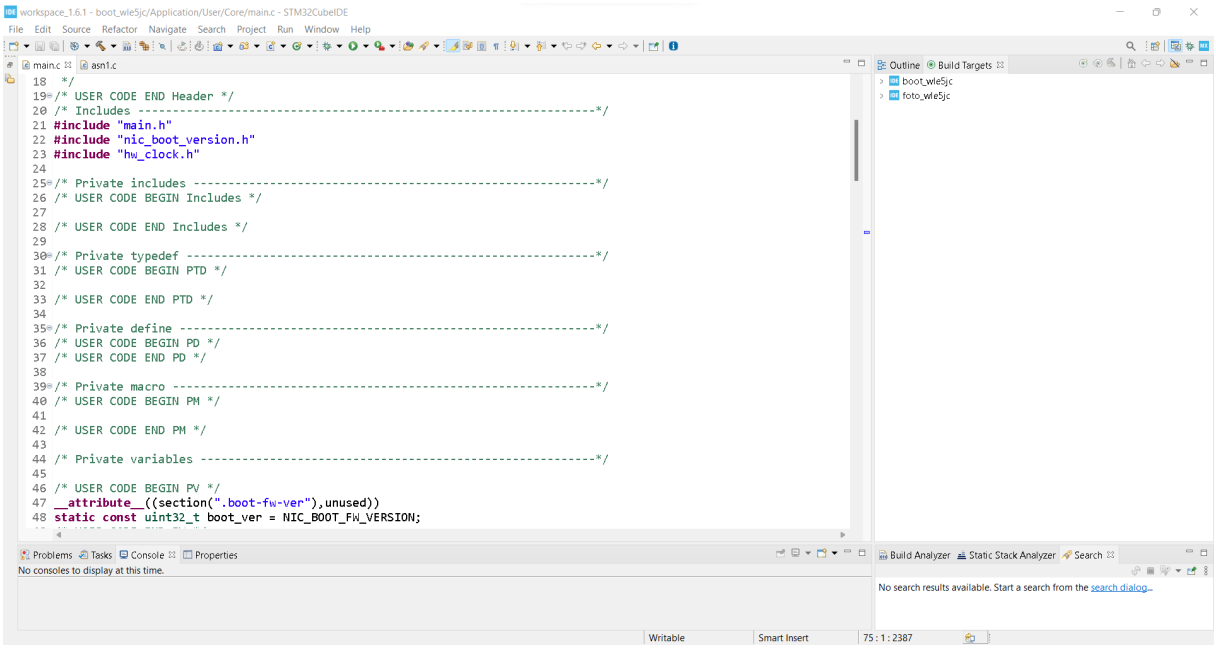
Por fim, como ferramenta de *debug*, optou-se por utilizar o *J-Link* versão *EDU*, uma ferramenta fornecida pela *Segger*, que é suportada pela *IDE* utilizada e que dá suporte a uma vasta gama de processadores e ambientes de desenvolvimento, com um grande diferencial contra a ferramenta da *ST*, possibilita *breakpoints* ilimitados. O fornecedor da ferramenta disponibiliza

Figura 22 – Software STM32CUBE MX



Fonte: Figura do autor.

Figura 23 – Software STM32CUBE IDE



Fonte: Figura do autor.

ainda um suíte de aplicativos que facilita a gravação, leitura da *flash*, proteção da memória, console para impressão de informações do debug, dentre outras ferramentas. Na figura 24 é possível observar a ferramenta citada.

Figura 24 – Sonda de debug Segger J-Link

Fonte: Segger, folha de dados.

A.3 Software

Para a comunicação com os dispositivos finais foram utilizados servidores executados em instâncias *AWS - Amazon WEB Sercice*, este servidor faz uso de uma *API (Application Programming Interface)* para se conectar ao *LNS (LoRa Network Server)* utilizado (podendo ser público como *Everynet*, ou privado como *Chirpstack*), tratando os *uplinks* e agendando pacotes de *downlink* permitindo as leituras e ações necessárias para o sucesso da atualização de imagem.

O protocolo implementado no servidor de aplicação segue a risca a especificação *DLMS/COSEM*, com as devidas modificações propostas que permitem a otimização de performance. Linguagens utilizadas, configurações, arquitetura da infraestrutura e outros pontos relacionados a aplicação não são foco deste trabalho, foram implementadas por terceiros seguindo o especificado e toma-se como pressuposto que a especificação foi seguida, dada a aprovação entre cliente e servidor *DLMS* na troca de dados.

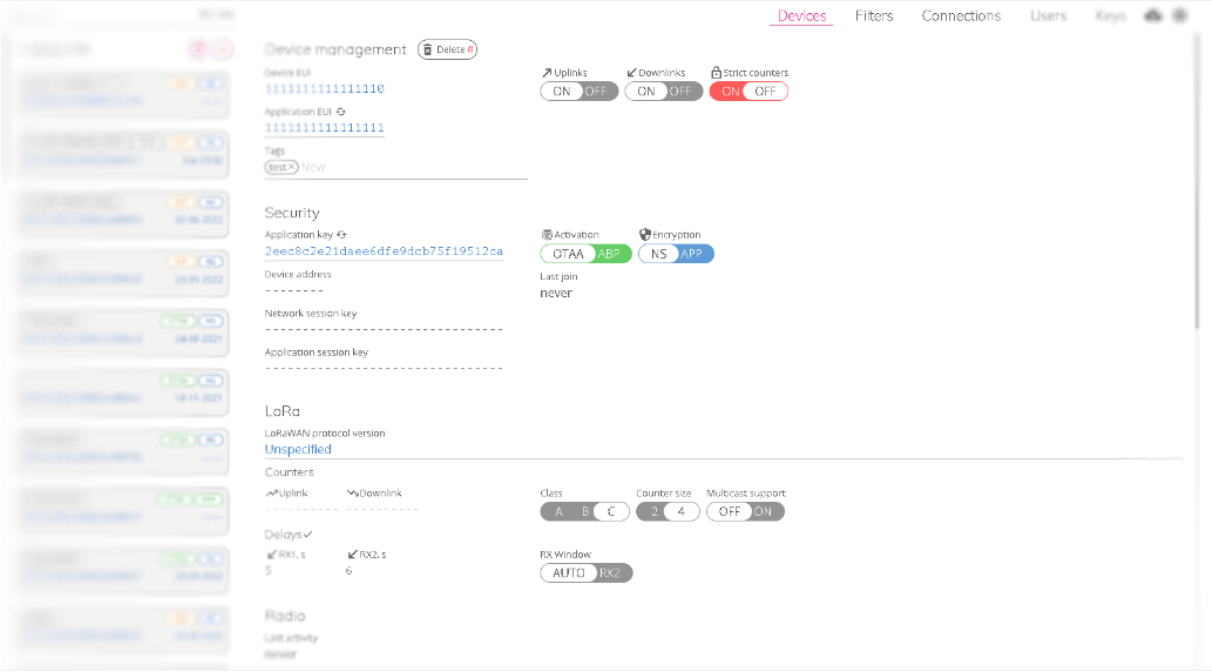
Para alguns ensasios específicos visando avaliar otimização do tempo, alguns *scripts* que agendam a transmissão dos blocos foram descritos em *Python*, podendo desta forma comparar eficiência de mudanças que podem ser realizadas nas estratégias apresentadas.

A.4 Infraestrutura de rede

Para os ensaios foi utilizado a rede pública *LoRa* da *ATC*, o servidor de rede *LoRa* (*LNS - LoRa Network Server*) que possibilida o cadastro dos dispositivos, criar conexões que possibilitem a integração com o servidor de aplicação dentre outras funcionalidades é o *Everynet*, na figura 25 é possível ter uma visão sobre o painel de controle deste servidor onde um dispositivo

fictício está em foco apresentando suas configurações e chaves de segurança (a imagem tem um desfoque aplicado na barra lateral esquerda que agrupa os dispositivos cadastrados visando a proteção de informações de dispositivos reais atualmente instalados em campo).

Figura 25 – Servidor de rede LoRa - Everynet, cadastro de dispositivo.



Fonte: ATC.

A *ATC* possua uma ampla infrafestrutura de rede instalada em todo o país com *gateways outdoor* espalhados em todas as capitais e nas principais cidades. Para garantir uma melhor cobertura de sinal que possibilite gerar variações nos ensaios, além da infraestrutura de rede já instalada, fez-se uso de um *gateway indoor* configurado para funcionar na rede da *ATC*. Na figura 26 é possível observar o dispositivo citado, denominado *IoT Femto Gateway*, o dispositivo utilizado pode ser encontrado pelo modelo *WLRGFM-100*.

Figura 26 – IoT Femto Gateway.

Fonte: Figura do autor.

A.5 Firmware

O *firmware* embarcado nas soluções foi desenvolvido baseado em algumas regras, estas regras estão especificadas em livros e documentos públicos e privados (no caso da especificação *DLMS/COSEM* que deve ser adquirida), as principais regras e especificações que nortearam o desenvolvimento são:

- *Blue Book Cosem Interface Classes and OBIS Object Identification System*;
- *Green Book DLMS/COSEM Architecture and Protocols*;
- *Barr Group Embedded C Coding Standard*;
- *Design Patterns for Embedded Systems in C: An Embedded Software Enginnering Toolkit*;
- *LLVM Coding Standards*.

Os dois primeiros livros especificam o padrão utilizado e adaptado para o processo de atualização de *firmware*. O primeiro livro de padrão de codificação (BARR, 2018), referência na área, foi adotado visando a minimização de *bugs* no *firmware*, agregando ainda capacidade de manutenção e portabilidade, requisitos necessários devido ao código embarcar uma série de soluções com controladores diversificados.

O livro de padrões de *design* (DOUGLASS, 2011) foi adotado devido as diversas recomendações de programação em linguagem C contidas, com exemplos e métodos de mitigação de riscos apresentadas, além de considerações sobre a programação para dispositivos com recursos limitados e inclusão de tópicos relevantes sobre máquinas de estado (altamente relevante devido a grande quantidade de código produzida com diversos estados possíveis).

Ainda foram consideradas as boas práticas apontadas nos padrões do projeto *LLVM* (LLVM-PROJECT, 2022), foram adotadas práticas relacionadas a escrita do código visando simplificação e legibilidade, ficando estilo de formatação e práticas de segurança para os documentos citados anteriormente.

A.5.1 Bootloader

O *bootloader* foi desenvolvido com duas funções básicas a serem realizadas: inicializar a aplicação principal e instalar uma nova aplicação (*firmware update*). Sempre que ocorrer uma reinicialização do dispositivo o *bootloader* é o ponto de partida, ele é responsável por verificar a integridade da imagem atual (o algoritmo utilizado pode ser substituido, ficando totalmente adaptável a cada aplicação), se o resultado do processo for sucesso, *bootloader* salta

para a aplicação principal.

Se existir um processo de ativação de imagem em andamento, as verificações de autenticidade e integridade da nova imagem são realizadas, caso o resultado do processo for sucesso, a imagem atual é copiada para um setor seguro da flash externa ou interna (dependendo das definições da aplicação) e a imagem nova é copiada sobrescrevendo a aplicação anterior. O salvamento da imagem atual como *backup* é opcional, sendo um ítem de segurança que pode agregar robustez em aplicações que não possuem altas restrições de memória. Ao fim de todo o processo, no caso de falhas, o bootloader é capaz de indicar via *IHM* o código de erro relacionado.

Na solução utilizada para os experimentos, o algoritmo de assinatura digital empregado é o *ECDSA - Eliptic Curve Digital Signature Algorithm*, um algoritmo de criptografia de curva eliptica, nesta implementação específica é utilizado os parâmetros *secp256k1*.

A.5.2 Aplicação

A aplicação foi desenvolvida com forte direcionamento a modularização, todos os módulos hoje implementados são inicializados na etapa de configuração dos módulos, e são percorridos um após o outro em um *loop* infinito. Desta forma é possível deduzir que a quebra dos processos em pequenas atividades é o ideal para que a aplicação seja executada de forma flúida e não blocante.

A aplicação possui como principais módulos:

- IHM - *feedback* visual do *status* do dispositivo e possíveis erros;
- *DLMS Server* - Responsável por tratar as requisições, agendar envio de dados síncronos e assíncronos;
- *Common Network* - Módulo de abstração para a camada de rede (*LoRa, NB-IoT, etc*;
- *Aplicação* - Máquina de estados principais que coordena a interação dos módulos principais;
- *Utilities* - Utilidades que variam de acordo com a solução, tomando como exemplo soluções de controle de iluminação pública podem exister os módulos de controle e dimerização da luminária, módulo de medição de energia, dentre outros.