UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA DE TELEINFORMÁTICA

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

ARTHUR PINTO BEZERRA

IMPLEMENTANDO UMA ATUALIZAÇÃO REMOTA DE FIRMWARE PARA SISTEMA EMBARCADO ATRAVÉS DO PROTOCOLO MQTT

FORTALEZA

2024

ARTHUR PINTO BEZERRA

IMPLEMENTANDO UMA ATUALIZAÇÃO REMOTA DE FIRMWARE PARA SISTEMA EMBARCADO ATRAVÉS DO PROTOCOLO MQTT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Alexandre Augusto da Penha Coelho.

FORTALEZA

2024

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)

B469i Bezerra, Arthur Pinto.
Implementando uma atualização remota de firmware para sistema embarcado através do protocolo MQTTS / Arthur Pinto Bezerra. – 2024.
49 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia de Computação, Fortaleza, 2024.
Orientação: Prof. Dr. Alexandre Augusto da Penha Coelho.

1. Atualização remota. 2. Modem. 3. Microcontrolador. 4. MQTT. 5. IoT. I. Título.
CDD 621.39

ARTHUR PINTO BEZERRA

IMPLEMENTANDO UMA ATUALIZAÇÃO REMOTA DE FIRMWARE PARA SISTEMA EMBARCADO ATRAVÉS DO PROTOCOLO MQTT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em: 30 de Setembro de 2024

BANCA EXAMINADORA

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

Eng. Dr. David Freitas Moura Mota
Universidade Federal do Ceará (UFC)

Eng. Me. Alexandre Almeida da Silva
Universidade Federal do Ceará (UFC)

Dedico este trabalho à minha família, especialmente à minha mãe Mariangela, meu pai Romulo e meu irmão Aramir, que sempre me apoiaram e incentivaram a seguir meus sonhos. Agradeço também aos amigos que estiveram ao meu lado, torcendo por cada conquista ao longo dessa jornada.

AGRADECIMENTOS

Agradeço, primeiramente, a minha família, mas principalmente a minha mãe, Mariangela, e a meu pai, Rômulo, por nunca terem desistido de incentivar meus estudos e por sempre me darem amor e ensinamentos responsáveis pela formação da pessoa que sou hoje. À todos os meus amigos que conheci no colégio Antares. Em especial, ao Lucca Lemos Costa Guerra e ao Raniere Paulino de Medeiros Filho, que me acompanharam durante o fim do ensino médio e graduação, por sempre acreditarem no meu potencial e me motivarem a ser melhor, não medindo esforços para estar ao meu lado.

Ao Prof. Dr.Alexandre Augusto da Penha Coelho, pela confiança e orientação que tornou este trabalho possível.

Ao meu amigo Lucas Silva Nogueira, com quem compartilhei inúmeras disciplinas, projetos no LESC e discussões sobre o TCC, agradeço pelo companheirismo e apoio incondicional ao longo da jornada.

Aos colegas do LESC, que colaboraram durante os projetos e foram fundamentais para meu aprendizado e desenvolvimento acadêmico.

"Se você não arriscar, não poderá criar um futuro." (Eiichiro Oda)

RESUMO

Atualmente, os dispositivos inteligentes desempenham um papel fundamental em ambientes de Internet das Coisas (IoT), estando presentes em diversos contextos, como residências, escritórios, lojas, e cidades inteligentes. Além desses espaços urbanos, sua aplicação se estende também a ambientes naturais, como florestas e oceanos, onde sensores e dispositivos conectados monitoram e coletam dados em tempo real. Esses dispositivos inteligentes dependem de protocolos de comunicação para interagir entre si, permitindo a troca de informações e o controle remoto. Esses protocolos viabilizam a manipulação e transmissão de dados coletados por sensores ou em resposta a eventos, facilitando a operação de aplicativos conectados. Entre as funcionalidades possibilitadas está o FOTA (Firmware Over the Air), que permite a atualização remota do firmware dos dispositivos, garantindo a manutenção, correção de falhas e inclusão de novos recursos, sem a necessidade de acesso físico aos dispositivos, otimizando sua performance e segurança. Neste contexto, o presente trabalho apresenta a implementação de um sistema de atualização remota de firmware (FOTA) para o microcontrolador CC1312, utilizando o protocolo de comunicação MQTT via comandos AT, os quais são instruções enviadas ao modem para controlar diversas operações de comunicação, para o modem Cinterion® EXS82. Este trabalho contempla o desenvolvimento do firmware do microcontrolador feito na linguagem de programação C, uma aplicação desenvolvida em Python para distribuir a imagem, a conexão entre eles e o fluxo de transferência da imagem. A solução demonstrou eficiência para blocos de até 256 bytes, porém apresentou falhas no cálculo de CRC para blocos superiores a 512 bytes, comprometendo a integridade da imagem. Soluções como a fragmentação mais robusta de pacotes, retransmissão de pacotes corrompidos e verificação incremental de CRC são sugeridas para melhorar o processo.

Palavras-chave: MQTT; OTA; IoT; Atualização remota; Modem; Microcontrolador.

ABSTRACT

Currently, smart devices play a fundamental role in Internet of Things (IoT) environments, being present in various contexts such as homes, offices, stores, and smart cities. In addition to these urban spaces, their application also extends to natural environments, such as forests and oceans, where connected sensors and devices monitor and collect data in real-time. These smart devices rely on communication protocols to interact with each other, allowing for the exchange of information and remote control. These protocols facilitate the manipulation and transmission of data collected by sensors or in response to events, enhancing the operation of connected applications. Among the functionalities enabled is FOTA (Firmware Over the Air), which allows for the remote update of device firmware, ensuring maintenance, bug fixes, and the addition of new features without the need for physical access to the devices, optimizing their performance and security. In this context, this work presents the implementation of a remote firmware update system (FOTA) for the CC1312 microcontroller, using the MQTT communication protocol via AT commands, which are instructions sent to the Cinterion® EXS82 modem to control various communication operations. This work encompasses the development of the microcontroller firmware in the C programming language, an application developed in Python for distributing the image, the connection between them, and the image transfer flow. The solution demonstrated efficiency for blocks of up to 256 bytes; however, it showed failures in CRC calculation for blocks larger than 512 bytes, compromising the integrity of the image. Solutions such as more robust packet fragmentation, retransmission of corrupted packets, and incremental CRC checks are suggested to improve the process. **Keywords**: MQTT; OTA; IoT; Remote update; Modem; Microcontroller.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AT	Attention
BIM	Boot Image Manager
CAT-M	Category M
CRC	Cyclic Redundancy Check
FOTA	Firmware Over-The-Air
IMEI	Identidade Internacional de Equipamento Móvel
IoT	Internet das Coisas
LESC	Laboratório de Engenharia de Sistemas de Computação
M2M	Máquina para Máquina
MCU	Microcontroller Unit
MQTT	Message Queuing Telemetry Transport
NIC	Network Interface Card
OAD	Over The Air Download
RAM	Memória de Acesso Aleatório
ROM	Memória Somente de Leitura
SIM	Subscriber Identity Module
UART	Universal Asynchronous Receiver/Transmitter
UFC	Universidade Federal do Ceará

SUMÁRIO

1 INTRODUÇÃO

1.1 Justificativa e Motivação

Na era da Internet das Coisas (IoT), a conectividade de dispositivos em tempo real é essencial para a automação, monitoramento e controle de sistemas remotos. A IoT possui uma ampla gama de dispositivos heterogêneos com requisitos em constante mudança e atualizações de funcionalidades, o que aumenta a necessidade de atualizações de *firmware* por meio de atualização remota (DODDAPANENI *et al.*, 2017). Nesse contexto, a capacidade de atualizar o *firmware* de dispositivos em campo, sem a necessidade de intervenção física, torna-se uma funcionalidade crítica. A técnica conhecida como Over The Air Download (OAD) permite essa atualização remota, proporcionando flexibilidade, segurança e redução de custos operacionais.

Este trabalho apresenta o desenvolvimento de um sistema de atualização remota de *firmware* (FOTA) para uma Network Interface Card (NIC) utilizando o protocolo Message Queuing Telemetry Transport (MQTT), um protocolo de comunicação leve e eficiente, amplamente utilizado em aplicações IoT.

A motivação para este trabalho surgiu da necessidade de uma solução de atualização de *firmware* para NICs utilizadas em um projeto entre o Laboratório de Engenharia de Sistemas de Computação (LESC) e a empresa OneRF, no qual, de forma simplificada, o *firmware* na NIC conversa periodicamente com um medidor de energia e modem. Essas NICs são responsáveis por coletar dados dos medidores e transmiti-los via MQTT utilizando a rede Category M (CAT-M). A implementação do OAD garante que as NICs possam ser atualizadas remotamente, mantendo-se seguras e operacionais sem a necessidade de deslocamento físico até o local dos medidores, otimizando recursos e melhorando a eficiência do sistema.

1.2 Objetivos

1.2.1 Objetivo Geral

O objetivo desse trabalho é implementar um sistema de atualização remota de *firmware* para um sistema embarcado que utiliza o microcontrolador CC1312, via protocolo de comunicação MQTT.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Estabelecer a conexão do modem Cinterion® EXS82 com o *broker* MQTT a partir de comandos Attention (AT);
- Desenvolver um script para conversão de uma imagem binária em blocos no formato base 64 e enviá-los via MQTT;
- Desenvolver um *firmware* para o microcontrolador que seja capaz de receber os blocos, e montar uma imagem binária válida;
- Analisar o funcionamento em campo e a eficiência do processo variando a quantidade de bytes por bloco.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por objetivo expor uma fundamentação que possibilite a compreensão dos elementos teóricos que embasam este trabalho. São abordados conceitos referentes aos componentes físicos utilizados nas Seções 2.1 e 2.2, que tratam, respectivamente, do microcontrolador e do modem. Na Seção 2.3 é explicado a teoria do funcionamento do protocolo de comunicação MQTT. Em seguida, na Seção 2.4 é explanado o sistema operacional utilizado, Contiki-ng. Por fim, a Seção 2.5 contém a teoria por trás da estrutura do OAD.

2.1 Sistemas Embarcados e Microcontroladores

Os sistemas embarcados são combinações de hardware e software projetadas para realizar uma função específica ou um conjunto de funções dentro de um sistema maior. Eles são chamados de "embarcados"porque estão integrados em outros dispositivos e muitas vezes não são visíveis para o usuário final. A principal característica de um sistema embarcado é sua capacidade de operar em tempo real, fornecendo respostas rápidas e confiáveis, essenciais para aplicações críticas como automóveis, aeronaves, eletrodomésticos, dispositivos médicos e até brinquedos.

No núcleo de muitos sistemas embarcados está o microcontrolador. Um microcontrolador é um pequeno computador em um único circuito integrado que contém um processador, memória e periféricos de entrada/saída (I/O). Diferente dos microprocessadores, que são usados em computadores pessoais e servidores, os microcontroladores são otimizados para controle de tarefas específicas, tornando-os extremamente eficientes para aplicações dedicadas (PEREIRA,).

2.1.1 Componentes Principais

- **Processador (CPU)**: A unidade central de processamento (CPU) é responsável pela execução das instruções do programa. Ela realiza operações aritméticas, lógicas e de controle com base nas instruções codificadas no software.
- **Memória**: Existem dois tipos principais de memória em um microcontrolador:
  - Memória Somente de Leitura (ROM): Usada para armazenar o firmware, ou seja, o software que o microcontrolador executa. Esse tipo de memória não é volátil, ou seja, mantém os dados mesmo quando a energia é desligada.
  - Memória de Acesso Aleatório (RAM): Usada para armazenar dados temporários

durante a execução dos programas. É volátil, ou seja, perde os dados quando a energia é desligada.

– **Periféricos de I/O**: Esses componentes permitem a interação do microcontrolador com o mundo exterior. Eles incluem portas digitais e analógicas, interfaces de comunicação (como UART, SPI, I2C), temporizadores, contadores e conversores A/D (analógico para digital) e D/A (digital para analógico) (PEREIRA,).

2.1.2 Funcionamento

O microcontrolador executa um programa pré-definido armazenado em sua ROM. Esse programa é composto de uma série de instruções que são processadas pela CPU. A CPU lê as instruções da ROM, executa cálculos, armazena resultados temporários na RAM e se comunica com os periféricos para controlar dispositivos externos.

2.2 Modem

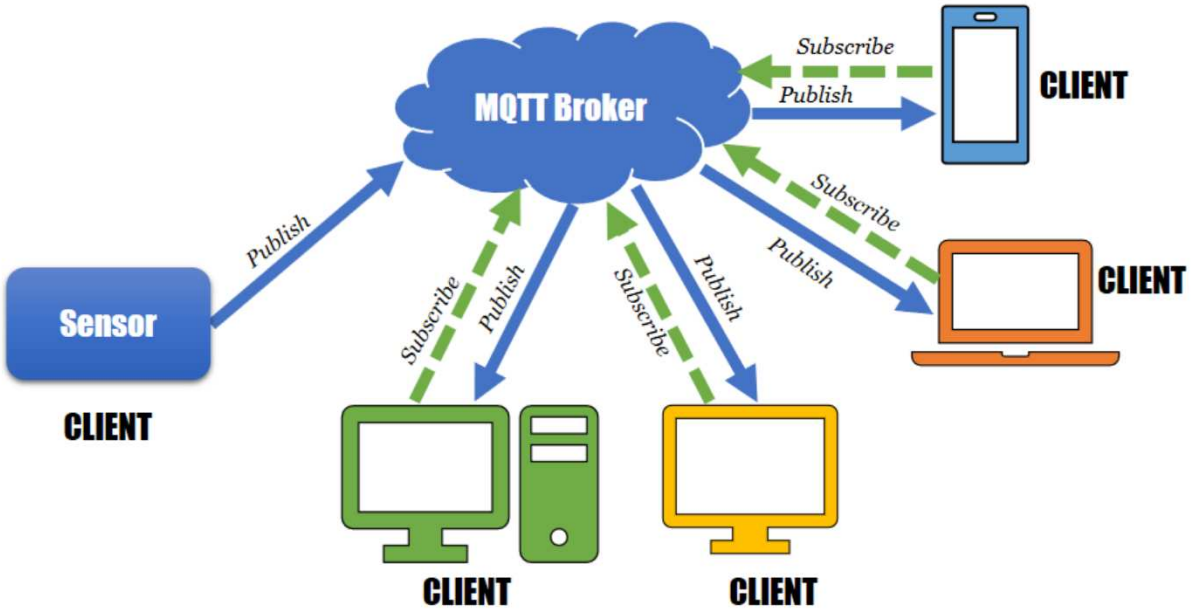
Nesse contexto, o modem (abreviação de modulador-demodulador) é um dispositivo que converte sinais digitais em sinais analógicos e vice-versa, permitindo a comunicação entre o sistema embarcado e outras redes, como a internet ou redes de telecomunicações. São frequentemente usados para permitir que o dispositivo se conecte a redes externas para transmitir ou receber dados, como em aplicações de IoT, nas quais dispositivos precisam comunicar dados coletados para um servidor remoto. Esses módulos geralmente possuem uma interface de comandos AT e são fáceis de usar a partir de um Microcontroller Unit (MCU) via Universal Asynchronous Receiver/Transmitter (UART). Com os comandos AT suportados, o MCU pode se conectar a uma rede Wi-Fi local ou a um servidor na Internet via um protocolo de comunicação, como MQTT ou TCP, e trocar algumas mensagens (ZHANG *et al.*, 2017).

2.3 MQTT

O MQTT (Message Queuing Telemetry Transport) é um protocolo de comunicação que é leve, aberto, simples e projetado para ser fácil de implementar. Essas características o tornam ideal para uso em muitas situações, incluindo ambientes restritos, como na comunicação em contextos de Máquina para Máquina (M2M) e IoT, no qual um código pequeno é necessário e/ou a largura de banda da rede é limitada. (OASIS Committee, 2014)

Seu funcionamento é baseado em uma arquitetura cliente-servidor, no qual os dispositivos atuam como clientes que publicam ou assinam mensagens, enquanto um servidor central, conhecido como *broker*, gerencia a troca de dados entre eles. O MQTT adota o modelo de publicação/assinatura (*publish*/*subscribe*) que difere dos tradicionais sistemas ponto a ponto. Nesse modelo, os clientes podem publicar mensagens em tópicos específicos, que funcionam como canais organizados hierarquicamente para categorizar as mensagens. Outros clientes, ao assinarem esses tópicos, recebem automaticamente as mensagens publicadas neles, facilitando a comunicação entre dispositivos sem a necessidade de conexão direta entre os mesmos.

Figura 1 – O modelo de Publicação/Assinatura do Mosquitto MQTT.



Fonte: Innovations (2024).

2.4 Sistemas Operacionais

Um sistema operacional (SO) é um software essencial que gerencia o hardware do computador e fornece serviços comuns para programas de aplicação. Ele atua como um intermediário entre o usuário e o hardware do computador, garantindo que os recursos do sistema sejam utilizados de maneira eficiente e segura.

2.4.1 Definição e Função

De acordo com Tanenbaum e Bos (2015), um sistema operacional pode ser definido como "um programa que age como uma interface entre o hardware do computador e o usuário".

Ele controla e coordena o uso do hardware entre os vários programas de aplicação para diferentes usuários. Isso inclui a gestão da memória, dos processos, dos dispositivos de entrada/saída e dos sistemas de arquivos (TANENBAUM; BOS, 2015).

2.4.2 Componentes Principais

Os componentes principais de um sistema operacional incluem:

1. **Gerenciamento de Processos**: Responsável pela criação, escalonamento e término dos processos. O SO deve gerenciar os estados dos processos e a comunicação entre eles.

2. **Gerenciamento de Memória**: Controla a alocação e desalocação da memória para os processos. Isso inclui a gestão da memória física e da memória virtual.

3. **Gerenciamento de Arquivos**: Cuida da criação, exclusão e manipulação de arquivos e diretórios. O SO também deve garantir a integridade dos dados armazenados.

4. **Gerenciamento de Dispositivos de Entrada/Saída**: Coordena a comunicação entre o sistema e os dispositivos de hardware, como discos rígidos, impressoras e interfaces de rede.

5. **Segurança e Proteção**: Assegura que os recursos do sistema sejam utilizados de forma segura e protegida. O SO implementa mecanismos de autenticação e autorização para controlar o acesso aos recursos do sistema.

2.4.3 Funcionamento

O funcionamento de um sistema operacional envolve a execução de várias funções críticas. O SO deve fornecer uma interface de usuário que permita a interação com o sistema de forma intuitiva e eficiente. Ele deve garantir que os processos sejam executados de maneira ordenada e sem interferências indevidas, utilizando técnicas de escalonamento e controle de concorrência (SILBERSCHATZ PETER B. GALVIN, 2018).

Além disso, o SO deve gerenciar os recursos do sistema de forma eficiente, garantindo que a CPU, a memória e os dispositivos de entrada/saída sejam alocados de acordo com as necessidades dos processos. Isso inclui a implementação de algoritmos de gerenciamento de memória, como paginação e segmentação, e de estratégias de escalonamento, como FIFO (First-In, First-Out) e RR (Round Robin) (TANENBAUM; BOS, 2015)

2.5 OAD

OAD é um método de atualização de *firmware* que permite que a imagem do *firmware* em execução em um dispositivo seja atualizada através do ar, ao mesmo tempo que oferece proteção contra perda de energia (Texas Instruments, s.d.).

2.5.1 Topologia

Dois dispositivos sem fio são necessários para realizar um OAD. Um é o *OAD Target* (Alvo do OAD), que é o dispositivo que recebe a imagem. O alvo do OAD é responsável por implementar a camada de transporte específica do protocolo da pilha que é usada para enviar e receber dados da imagem OAD. Já o *OAD Distributor* (Distribuidor do OAD) é responsável por fragmentar a nova imagem de *firmware* em pacotes específicos da pilha de protocolo e enviá-los.

2.5.2 Passos

O processo de atualização pode ser resumido em quatro passos:

1. Criar uma imagem binária OAD para o alvo.
2. Carregar a imagem alvo no dispositivo distribuidor.
3. Transferir a imagem do distribuidor para o dispositivo alvo.
4. Substituir a imagem de *firmware* existente.

2.5.3 OAD IMAGE HEADER

Todas as imagens de *firmware* entregues via OAD contêm um cabeçalho de imagem. As informações no cabeçalho da imagem são usadas pela aplicação para determinar a adequação de uma imagem para download ou carregamento.

2.5.3.1 CRC

O Cyclic Redundancy Check (CRC) é um meio de verificar a integridade de dados. Para o caso de uma imagem, isso deve ser feito em duas etapas. Primeiro, o CRC deve ser calculado quando ela é gerada pela cadeia de ferramentas (pela Ferramenta de Imagem OAD), e será armazenado no campo CRC dentro do cabeçalho da imagem. Posteriormente, uma vez que o alvo tenha recebido a imagem OAD, o CRC será recalculado para determinar se a imagem foi

corrompida durante a transferência. Se o CRC for equivalente antes e depois do OAD, o alvo pode assumir que a imagem não foi corrompida durante o envio.

2.5.4 BIM

O Boot Image Manager (BIM) reside no alvo do OAD e é responsável por carregar novas imagens após o término do download. O BIM é executado após um reset do dispositivo e determina se uma atualização de *firmware* precisa ser aplicada. Se nenhuma atualização estiver sendo aplicada, então o BIM transferirá a execução do programa para a imagem principal da aplicação. Ele é uma aplicação totalmente executável que é independente de qualquer pilha de protocolo de alto nível ou aplicação do usuário e é garantido para ser executado na inicialização.

Ademais, permite tolerância a falhas de perda de energia durante o OAD. Se a energia do dispositivo for perdida durante o OAD, o BIM ainda será capaz de executar a partir do reset e reverter para uma imagem funcional se uma estiver disponível. O BIM é projetado para residir permanentemente no dispositivo embarcado e não pode ser atualizado através do processo OAD. Ele produzirá uma imagem totalmente executável que deve ser mesclada com a imagem da aplicação do usuário para criar um sistema de *firmware* funcional habilitado para OAD. Em geral, o BIM é responsável por encontrar e analisar o cabeçalho da imagem OAD. Em um nível alto, o BIM faz o seguinte:

1. Verificar se há novas imagens transferidas de um OAD recente. Se disponível, copiá-las para seu local alvo na memória flash interna.
2. Localizar dinamicamente o ponto de entrada da imagem válida e pular para ele.

2.5.5 Memória

A memória de um sistema embarcado pode ser dividida em dois tipos principais: memória interna e memória externa. A memória interna, geralmente composta de *flash* e RAM, está diretamente integrada ao microcontrolador e é usada para armazenar o firmware e executar tarefas do sistema. A RAM é volátil, perdendo seu conteúdo quando o dispositivo é desligado, enquanto a memória *flash* interna armazena o firmware atual e dados essenciais.

Por outro lado, a memória externa é tipicamente uma memória não volátil adicional conectada ao microcontrolador. No contexto do OAD, a memória externa desempenha um papel crucial ao atuar como um repositório temporário para armazenar os blocos de firmware recebidos durante o processo de atualização. Ela permite que grandes volumes de dados sejam gerenciados

sem sobrecarregar a memória interna do sistema.

Durante o processo de OAD, os blocos de firmware são transferidos para a memória externa em blocos, onde permanecem até que todos os blocos sejam validados e o processo seja concluído. Somente após essa verificação, o firmware armazenado na memória externa é transferido para a memória interna do microcontrolador, garantindo uma atualização segura e livre de falhas.

2.6 Base64

Base64 é um método de codificação que transforma dados binários em uma representação de texto ASCII, utilizando um conjunto de 64 caracteres imprimíveis (A-Z, a-z, 0-9, +, e /) e o caractere "="como preenchimento, se necessário. permitindo que esses dados sejam facilmente transmitidos por sistemas que lidam com dados de texto. É amplamente utilizado para codificar dados binários em formatos que podem ser manipulados ou transmitidos por sistemas que não suportam diretamente dados binários, como em e-mails ou URLs (Mozilla Developer Network, n.d.).

3 MATERIAIS E MÉTODOS

Esta seção detalha os componentes e os procedimentos utilizados para implementar o sistema de atualização remota de *firmware* via protocolo MQTT. Inicialmente, são descritos o hardware empregado no projeto na Seção 3.1, que inclui o microcontrolador e o modem utilizados. Em seguida, a Seção 3.2 aborda a utilização do sistema operacional Contiki-NG, fundamental para a implementação do projeto. A Seção 3.3 trata do método de comunicação entre o microcontrolador e o modem, realizado através da interface UART. Os principais comandos AT utilizados para a comunicação com o modem, que permitem a configuração dos parâmetros de rede e a execução das operações de publicação e subscrição no *broker* MQTT, são apresentados na Seção 3.4. Adicionalmente, na Seção 3.5, são discutidas as configurações necessárias do modem. O papel do distribuidor do OAD é descrito na Seção 3.6, enquanto o alvo do OAD é abordado na Seção 3.7. A Seção 3.8 discute o processo de preparo do *firmware*, seguido pela Seção 3.9, que detalha o fluxo de download e tratamento dos dados. Por fim, fluxo do OAD e a lógica de download dos blocos são detalhados na Seção 3.10.

3.1 Hardware Utilizado

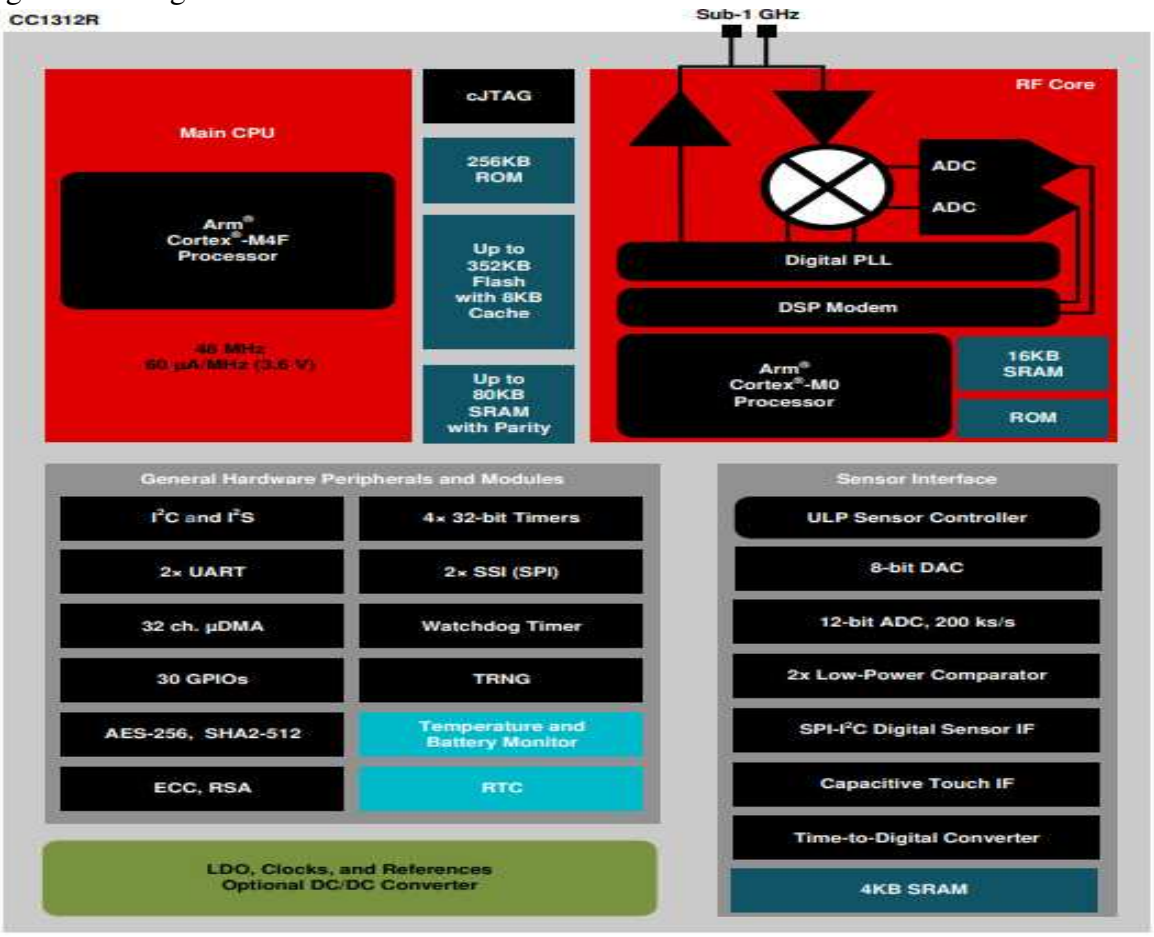
3.1.1 MCU SimpleLink™ CC1312R

O dispositivo SimpleLink™ CC1312R, utilizado neste trabalho, é um microcontrolador Sub-1 GHz que suporta, por exemplo, objetos inteligentes habilitados para IPv6 e sistemas proprietários, incluindo o TI 15.4-Stack. O dispositivo é otimizado para comunicação sem fio de baixo consumo de energia e sensoriamento avançado em sistemas de segurança predial, medidores inteligentes, dispositivos médicos, redes com fio, eletrônicos portáteis, sistemas de *home theater* e entretenimento, e mercados de periféricos conectados (Texas Instruments, 2020). Na figura 2 consta o diagrama de blocos deste microcontrolador.

3.1.2 Modem Cinterion® EXS82

O modem utilizado neste projeto foi o Cinterion® EXS82, que é um módulo de comunicação IoT avançado que oferece conectividade de baixa potência (LPWA) para dispositivos industriais. Suportando LTE-M, NB-IoT, e fallback opcional para 2G, ele é ideal para dispositivos de baixa potência em locais remotos, como medidores inteligentes e rastreadores de ativos. Ele

Figura 2 – Diagrama de Blocos do CC1312R



Fonte: (Texas Instruments, 2020, p. 3).

conta com recursos como PSM e eDRX para eficiência energética, além de uma arquitetura de segurança robusta, incluindo um eSIM integrado que gerencia a autenticação e a conexão segura com redes celulares. O modem suporta atualizações de *firmware* incremental via Firmware Over-The-Air (FOTA), que se refere ao conceito geral de atualização de *firmware* sem fio, o que é essencial para manter as soluções IoT atualizadas sem necessidade de substituição total do *firmware*, economizando energia e largura de banda. O EXS82 também possui suporte a diversas interfaces como USB, SPI, I2C, e GPIO, facilitando sua integração com outros componentes em um sistema embarcado.(Telit, 2024)

Para garantir o funcionamento do modem, foi-se necessário conectar uma antena e um Subscriber Identity Module (SIM) card com dados de Internet para que seja possível acessar a rede CAT-M.

Figura 3 – Imagem do Modem Cinterion® EXS82-W

Fonte: Telit (2024).

3.1.3 SIM Card NLT

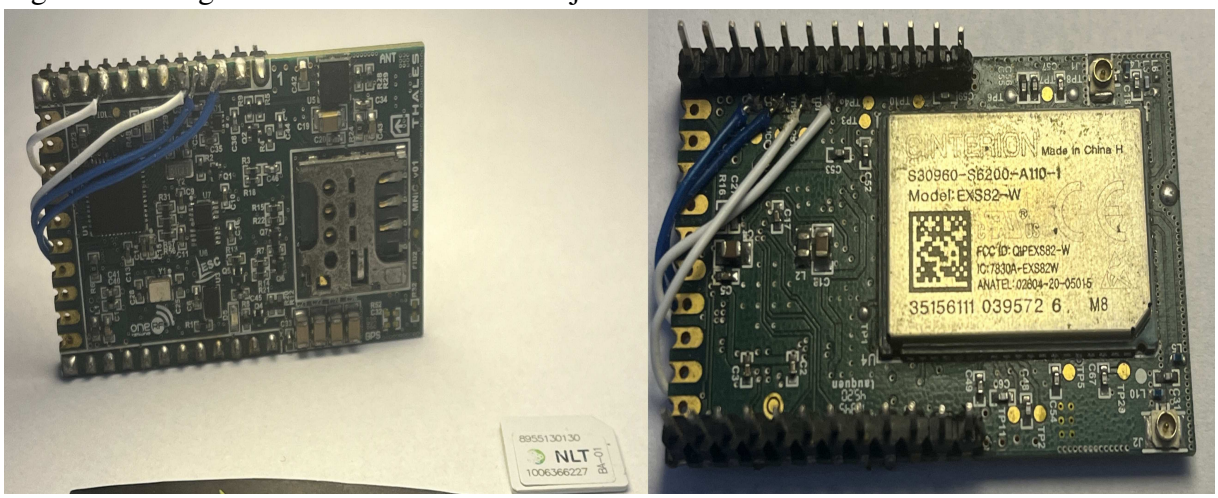
O SIM utilizado neste trabalho foi uma unidade da operadora móvel virtual NLT Telecom, autorizada pela ANATEL , que utiliza topologia de conectividade com a infraestrutura de redes de acesso da Vivo. O chip da NLT Telecom é uma solução especializada voltada para a conectividade em dispositivos IoT. Ele é projetado para operar em redes de comunicação celular, como LTE-M e NB-IoT , que oferecem cobertura estendida e baixo consumo de energia, características essenciais para aplicações em IoT. A NLT Telecom fornece uma infraestrutura robusta e confiável, que inclui suporte para redes de alta disponibilidade e segurança, garantindo que dispositivos conectados possam operar de forma eficiente e contínua em ambientes diversos, desde áreas urbanas até locais remotos (NLT Telecom, s.d.).

3.1.4 Hardware de Teste

Neste trabalho foi utilizado a NIC desenvolvida no Projeto MNIC realizado entre o LESC da Universidade Federal do Ceará (UFC) e a OneRF. A alimentação dessa unidade foi feita a partir de uma fonte chaveada que possui tensão de saída de 5V DC e corrente de saída

de 3A. Por fim, a antena adquirida para testes foi a Taoglas MFX3, a qual é designada para aplicações NB-IoT e CAT M1.

Figura 4 – Imagem da NIC Utilizada no Projeto MNIC



Fonte: Autor.

3.2 Contiki-NG

O Contiki-NG é um sistema operacional de código aberto voltado para sistemas embarcados de baixa potência, amplamente utilizado em dispositivos de Internet das Coisas (IoT). Ele é ideal para o desenvolvimento de aplicações com recursos limitados, como os sistemas de atualização de firmware abordados neste trabalho (OIKONOMOU *et al.*, 2022).

Neste projeto, o Contiki-NG foi empregado para gerenciar a comunicação entre o microcontrolador e o modem, permitindo a integração eficiente do protocolo MQTT no processo de Over-the-Air Download (OAD).

3.3 Comunicação via UART

A comunicação entre o modem e o microcontrolador CC1312 foi realizada via UART1, com um baudrate de 115200, que representa a taxa de transmissão de dados, indicando quantos bits são transmitidos por segundo. A leitura dos dados pela interface serial, utilizada para registrar logs de eventos, foi feita através da UART0, com um baudrate de 2400. A transmissão dos comandos, formatados como cadeias de caracteres, foi implementada a partir da chamada de função de escrita baseada na biblioteca do Contiki-NG.

3.4 Comandos AT

A Tabela 1 condensa todos os comandos AT utilizados neste trabalho, junto de uma descrição sucinta do que cada um realiza.

Tabela 1 – Comandos AT

Comando	Descrição
CGDCONT	Define parâmetros de contexto PDP para conexão de dados
SICA	Ativa ou desativa uma conexão de serviço de Internet usando um contexto PDP
SISS	Especifica perfis de serviço de Internet para controlar conexões de dados.
SISO	Inicia um serviço de Internet configurado previamente com SISS, após ativar o portador com SICA
SISW	Inicia uma operação de escrita (upload)
SISR	Realiza uma operação de leitura (download)
SXRAT	Especifica as Tecnologias de Acesso por Rádio (RAT) para seleção e registro de rede
SISC	Encerra a conexão TCP/IP com o peer remoto estabelecida pelo comando ATˆSISO
CGSN	Retorna o número de série do dispositivo
SGAUTH	Utilizado para definir o tipo de autenticação para conexões PDP−IP

Fonte: Autor, adaptado de (GS M2M, 2020).

3.5 Configuração do Modem

Neste trabalho o microcontrolador alterna entre determinados estados para efetivamente concluir o FOTA. A partir deles é possível determinar três atividades principais em relação ao modem:

- Configurar parâmetros do modem;
- Ativar modo Subscribe do modem;
- Ativar modo Publish do modem.

Em primeiro momento, para que seja possível realizar o download de dados da nova imagem de *firmware* via MQTT, foi necessário desenvolver uma sequência de comandos AT para que o modem consiga estabelecer a conexão com o *broker* e seja capaz tanto de publicar mensagens, como também escutar e salvar mensagens enviadas por outros dispositivos.

Os detalhes sobre o *broker* utilizado no sistema de atualização de *firmware* são informações sensíveis e confidenciais da empresa, e por essa razão, não serão divulgados nesta seção. Contudo, vale ressaltar que, para fins de testes e desenvolvimento, seria possível utilizar *brokers* MQTT gratuitos disponíveis no mercado. No entanto, para a transmissão de dados sensíveis, como imagens de *firmware*, a utilização de um *broker* seguro e privado é fortemente recomendada, a fim de garantir a segurança e integridade dos dados durante o processo de

atualização, especialmente em ambientes de produção.

3.5.1 Configurar Parametros Gerais

Nesta seção o microcontrolador realiza uma sequência de comandos AT para configurar os parâmetros do modem necessários. Após concluir este passo, o modem está apto tanto para ser configurado para se subscrever em um tópico, quanto para publicar.

1. AT^SXRAT= 12,7,0: Seleciona a tecnologia de acesso radioelétrico (RAT), preferindo CAT-M com fallback para NB-IoT e 2G.
2. AT+CGDCONT= 1,"IP","nlt.com.br": Define o Ponto de Acesso à Rede (APN) para a conexão da nlt.
3. AT^SGAUTH= 1,1,"nlt","nlt": Define o tipo de autenticação para conexões PDP-IP, especificando usuário e senha da APN.
4. AT^SICA=0,1: Garante que a conexão esteja desativada inicialmente.
5. AT^SISS=1,srvType,"Mqtt": Define o tipo de serviço como MQTT para o perfil de serviço 1.
6. AT^SISS=1,conId,"1": Define o ID de conexão como "1".
7. AT^SISS=1,address,"mqtt://user:passwd@host:port": Define o endereço do *broker* MQTT, com usuario, senha e porta de acesso.

3.5.2 Modo Subscribe

Esta sequencia compreende o processo de configurar o modem para o modo de *subscribe*, aguardar o envio de uma mensagem e, posteriormente, a sua leitura. Os dados salvos serão em seguidas tratados para armazenar os blocos de imagem.

1. AT^SISS=1,cmd,"subscribe": Define o comando para subscrever a um tópico MQTT.
2. AT^SISS=1,clientId,351561110395726: Define o ID do cliente MQTT.
3. AT^SISS=1,topicFilter,"351561110395726/oad": Define o filtro de tópico MQTT para receber mensagens.
4. AT^SICA=1,1: Ativa a conexão.
5. AT^SISO=1,2: Abre a conexão TCP/IP para o serviço.
6. AT^SISR=1,422: Lê o buffer com um tamanho específico para receber mensagens.
7. AT^SISC=1,1: Fecha a conexão após a leitura das mensagens.

3.5.3 Modo Publish

Esta sequencia compreende o processo de configurar o modem para o modo de *publish*, e, em seguida, o envio de uma mensagem. Os dados a serem enviados consistem na requisição de um bloco para o servidor Distribuidor do OAD.

1. ATˆSISS=1,cmd,"publish": Define o comando para publicar uma mensagem.
2. ATˆSISS=1,hcContLen, x: Define o tamanho do conteúdo da mensagem para x, onde x é um número qualquer.
3. ATˆSISS=1,clientId,351561110395726: Define o ID do cliente MQTT.
4. ATˆSISS=1,Topic,"oad": Define o tópico MQTT onde a mensagem será publicada.
5. ATˆSICA=1,1: Ativa a conexão.
6. ATˆSISO=1,2: Abre a conexão TCP/IP para o serviço.
7. ATˆSISW=1,x: Envia o conteúdo da mensagem de tamanho definido pelo hcContLen.
8. ATˆSISC=1,1: Fecha a conexão após o envio da mensagem.
9. ATˆSICA=0,1: Desativa a conexão para garantir que não fique aberta desnecessariamente.

3.6 Distribuidor OAD

O distribuidor OAD é um componente essencial no processo de atualização remota de *firmware* (OAD) desenvolvido na linguagem de programação Python, utilizando o protocolo MQTT. Ele foi projetado para gerenciar a distribuição de novos *firmwares* para dispositivos IoT, garantindo que as atualizações sejam realizadas de maneira segura e eficiente.

3.6.1 Funcionalidades da Aplicação

1. **Conexão ao Broker MQTT**: O aplicação começa estabelecendo uma conexão com o *broker* MQTT, utilizando credenciais específicas. Ele se inscreve em um tópico designado para receber pedidos de atualização dos dispositivos. A conexão é mantida ativa durante todo o processo de atualização.
2. **Preparo do Firmware**: O *firmware* a ser distribuído é lido a partir de um arquivo binário (`mqtt-fota_oad.bin`). Esse arquivo é dividido em blocos de 256 bytes, que são armazenados em um dicionário para facilitar o acesso e envio subsequente. Cada bloco é preparado com um índice que facilita a solicitação e o envio corretos dos dados.
3. **Recebimento e Resposta a Solicitações**: A aplicação fica atento às mensagens publicadas

no tópico inscrito. Quando uma solicitação de bloco de *firmware* é recebida, a aplicação identifica o número do bloco solicitado, acessa o bloco correspondente no dicionário e o codifica em base64. Essa codificação é essencial para garantir que os dados binários sejam transmitidos corretamente via MQTT, que é um protocolo orientado a texto.

4. **Envio dos Blocos de Firmware**: O bloco codificado em base64 é então enviado de volta ao dispositivo através de um tópico específico relacionado ao Identidade Internacional de Equipamento Móvel (IMEI) do dispositivo. A aplicação também gerencia a resposta para garantir que todos os blocos sejam enviados na ordem correta e sem falhas, utilizando mecanismos de repetição em caso de falhas na transmissão.
5. **Finalização e Verificação**: Após o envio de todos os blocos, a aplicação monitora mensagens de confirmação para garantir que o dispositivo tenha recebido e aplicado a atualização corretamente. Caso todas as etapas sejam concluídas com sucesso, o processo é finalizado, e a conexão com o *broker* MQTT é encerrada.

3.6.2 Robustez e Manutenção de Conexão

A aplicação foi projetado para lidar com interrupções temporárias na conexão e outros erros que possam ocorrer durante a transmissão de dados. Em caso de falha, ele tenta reconectar ao *broker* MQTT e recomeçar o envio de onde parou, garantindo que a atualização seja concluída com sucesso. Além disso, a aplicação inclui verificações periódicas do status da conexão, garantindo que a comunicação seja mantida até o final do processo.

3.7 Alvo do OAD

3.7.1 OAD Core Image Header

O cabeçalho principal contém as informações essenciais necessárias para o OAD. Sua presença é imprescindível, pois o BIM depende dele para inicializar e realizar o OAD da imagem. Além disso, o cálculo do CRC toma como base o valor contido nesse cabeçalho. As posições de cada informação do cabeçalho para o caso do microcontrolador CC1312 estão contidas na tabela 2

Tabela 2 – Descrição do cabeçalho principal

Campo	Tamanho (em bytes)	Descrição
Valor de Identificação da Imagem OAD	8	Número único para identificar o início de uma imagem OAD
CRC	4	Verificação de Redundância Cíclica
Versão do BIM	1	Versão necessária para suportar o formato da imagem
Versão do Cabeçalho da Imagem	1	Versão do cabeçalho da imagem contida na imagem
Tecnologia Sem Fio	2	Tipo de conectividade usada na imagem
Informações da Imagem	4	Bytes de informação da imagem
Validação da Imagem	4	Verifica se a imagem é válida para execução
Comprimento da Imagem	4	Comprimento total da imagem, incluindo o cabeçalho
Endereço de Entrada do Programa	4	Endereço de entrada de inicialização da aplicação
Versão do Software da Imagem	4	Versão de software da imagem
Endereço Final da Imagem	4	Endereço final da imagem
Comprimento do Cabeçalho	2	Comprimento total do cabeçalho da imagem

Fonte: Autor, adaptado de (Texas Instruments, s.d.)

3.7.2 Layout Flash Externa

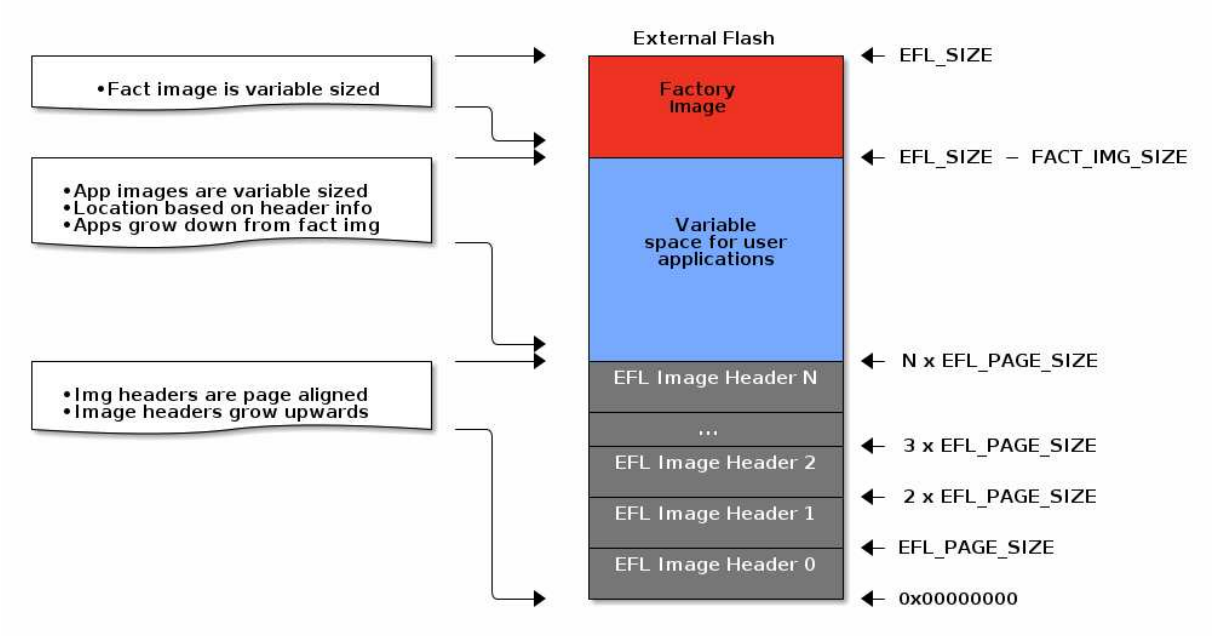
Baseando-se no layout descrito na Figura 5, o tamanho total de armazenamento da memória flash externa é de 1MB no microcontrolador CC1312R1. Este espaço permite armazenar imagens de firmware com um tamanho máximo considerado de 151.552 bytes, sendo que para todos os casos analisados, as imagens não ultrapassaram 120.000 bytes. Isso garante que múltiplas versões de firmware possam coexistir na memória, evitando a sobrecarga do sistema.

Além disso, os metadados — informações sobre as versões e endereços do firmware — são alocados em espaços dedicados, facilitando a gestão e prevenindo a sobreposição de dados. Essa organização permite que áreas específicas sejam reservadas para dados do usuário e firmware, garantindo um uso seguro da memória. Assim, respeitando o espaço dedicado às aplicações do usuário, foi definido que o início da imagem de transferência começaria na posição de memória "0x20000". Outra posição crítica para este trabalho é a variável que armazena o número do bloco que foi transferido por último. Essa informação é vital em caso de falhas ou interrupções durante o processo de transferência, permitindo que a NIC retome de onde parou.

3.7.3 Inicialização do Sistema

A inicialização do sistema é responsável por configurar os componentes essenciais para o funcionamento do processo, incluindo a inicialização do modem e a configuração de *timers*, que controlam intervalos de operação. Antes de qualquer operação, o sistema reinicia o *watchdog*, um mecanismo que monitora o funcionamento e evita travamentos inesperados. Se o

Figura 5 – Layout Flash Externa



Fonte: (Texas Instruments, s.d.).

sistema falhar em responder dentro de um intervalo específico, o *watchdog* reinicia o dispositivo. Após configurar o *watchdog*, o modem é inicializado para estabelecer a conexão MQTT. Em seguida, o IMEI do dispositivo é coletado e exibido para assegurar que o dispositivo esteja identificado corretamente na rede.

Além disso, diversos *timers* são configurados para controlar os intervalos de tempo entre as operações críticas, como o tempo de duração da subscrição e o tempo de duração das operações durante o FOTA.

3.7.4 Configuração do MQTT

Nesta etapa, o *broker* MQTT é configurado no modem, utilizando os comandos AT descritos na Seção 3.5.1, estabelecendo a conexão necessária para a troca de mensagens durante o processo FOTA.

3.8 Preparo do Firmware

Antes do processo de OAD, é necessário preparar uma imagem válida para que o Distribuidor OAD possa enviar os dados corretamente.

3.8.0.1 OAD Image Tool

O *OAD Image Tool* da Texas Instruments é uma ferramenta crucial para a criação de imagens de *firmware* compatíveis com o processo de OAD em dispositivos da família CC13x2. Esta ferramenta é utilizado como parte do fluxo de trabalho de compilação para garantir que as imagens geradas sejam válidas para o OAD. Especificamente, a ferramenta é responsável por converter o arquivo de saída da compilação, como um .out ou .hex, em um arquivo binário `.bin`. Além disso, o OAD Image Tool alinha os dados de imagem para garantir que estejam corretamente formatados, calcula o CRC da imagem, e insere informações necessárias no cabeçalho da imagem, como o tipo de imagem e a versão (Texas Instruments, s.d.).

3.9 Download

Após as configurações iniciais, o processo FOTA é executado, envolvendo a inscrição em tópicos MQTT, a recepção do novo *firmware*, e a atualização do dispositivo. Esta seção descreve o fluxo de atualização remota do *firmware* via MQTT. O processo inicia com a publicação da solicitação de download e a subscrição ao tópico, seguida pelo envio dos parâmetros necessários. Em cada etapa, o *parser* processa as mensagens para extrair os dados relevantes, enquanto o laço de download garante a sequência correta dos blocos. O *payload*, codificado em Base64, é decodificado e armazenado no sistema. Ao final, o processo é concluído com a verificação da integridade do *firmware* atualizado.

3.9.1 Payload

O *payload* refere-se aos dados do bloco de *firmware* que estão sendo transferidos para a NIC durante o processo de atualização. Esses dados estão codificados em base64 e são parte do conteúdo do novo *firmware* que está sendo baixado e aplicado ao dispositivo. Cada bloco de dados é identificado por um número de bloco, denominado como bl_num, e, junto com o *payload*, compõe a sequência necessária para reconstituir o *firmware* completo no dispositivo. Um exemplo de *payload* antes e após o *parsing*, respectivamente, pode ser identificado abaixo, sendo "dados"um termo para representar a sequência de caracteres em base64 e 385 o número do bloco:

```
1    "payload": dados "bl_num": 385
```

```
token 1: "payload":
token 2: "dados"
token 3: "bl_num":
token 4: 385
```

3.9.2 Parser

Ele é responsável por analisar e processar uma mensagem recebida para extrair informações específicas. O processo pode ser descrito nas seguintes etapas principais:

1. Preparação da Mensagem: A função inicia limpando um *buffer* e copiando a mensagem recebida para esse *buffer* para garantir que o processamento não altere o conteúdo original.
2. Processamento da mensagem: A função verifica se a mensagem contém um delimitador específico que indica que a linha deve ser processada. Para o caso da primeira mensagem o delimitador é uma palavra que indica o total de blocos, enquanto que para o restante dos blocos é apenas a palavra "*payload*". Se a linha contém o delimitador esperado, ela é dividida em partes menores (*tokens*) usando um delimitador de espaço.
3. Extração de Informações: Os *tokens* extraídos são analisados para encontrar os dados relevantes, que nesse caso incluem:
   - Um *payload* (informação principal da mensagem), que consiste no que precisa ser extraído;
   - O número total de bloco, para o caso da primeira mensagem;
   - O bloco atual, o qual é comparado com o bloco esperado pela NIC, de forma que caso sejam divergentes a operação é cancelada e o bloco é requisitado novamente.
4. Validação dos Dados:
   - A função valida se o *payload* foi corretamente extraído e se o número do bloco corresponde ao esperado;
   - Se qualquer uma das validações falhar, a função retorna uma mensagem de erro;
   - Se todas as validações forem bem-sucedidas, a função retorna o *payload* extraído.

3.9.3 Publicação Inicial e Subscrição

Uma mensagem MQTT é publicada no tópico do IMEI, utilizando as instruções da Seção 3.5.3, para sinalizar que o dispositivo está pronto para ouvir comandos via MQTT. Essa mensagem inclui detalhes como o status "Ready". Em seguida, o dispositivo entra em modo de subscrição, respeitando os comandos expostos na Seção 3.5.2, para aguardar comandos, como "DOWN" para iniciar o download do *firmware* ou "RESET" para reiniciar o sistema.

3.9.4 Envio de parâmetros

Quando a NIC recebe uma mensagem solicitando o download de uma imagem, ela primeiro verifica qual bloco está armazenado usando a função $get()$, que realiza a leitura da memória flash externa. Em seguida, a NIC define o tópico "oad" como destino para suas publicações e envia uma mensagem que inclui seu IMEI, solicitando informações sobre a imagem. Por fim, se inscreve no tópico associado ao seu IMEI. Dessa maneira, a aplicação que está inscrito no tópico "oad" armazena o IMEI da NIC e responde à solicitação publicando os dados no tópico correspondente. Esta abordagem, que utiliza tópicos distintos para cada unidade NIC, é importante para permitir que múltiplas unidades realizem o processo de OAD simultaneamente para um mesmo distribuidor.

3.9.5 Laço de Download

- Controle de Tentativas de Publicação:
  - O laço começa com o número do bloco inicializado a partir do valor armazenado na flash externa e continua por enquanto que o bloco seja menor que o total de blocos;
  - Se contagem de tentativas de publicação atingir 10, significa que a aplicação não está respondendo as requisições, então a função imprime uma mensagem de erro e retorna -1, indicando falha na tentativa de download.
- Publicação e Assinatura:
  - Configura uma mensagem de publicação e a envia via MQTT para o tópico "oad";
  - Em seguida, a função se inscreve no tópico relativo ao seu IMEI e reinicia um temporizador para aguardar a resposta.
- Recebimento e Processamento da Resposta:
  - Dentro de um loop, a função lê a resposta do tópico MQTT.

  - A resposta é analisada para extrair o *payload* usando a função de "parsing";
  - Se ocorrer um erro na análise, *payload* conterá "ERROR", a função aumenta a contagem de tentativas e reinicia a tentativa de publicação do bloco.
- Validação e Processamento de Dados:
  - Se a resposta for válida e o *payload* não estiver vazio, a função:
    * Decodifica o *payload* do formato Base64;
    * Se o bloco atual for o bloco 0, realiza operações adicionais para processar o cabeçalho da imagem e atualizar a versão disponível do *firmware*;
    * Processa o bloco de dados, atualiza o número do bloco, e aguarda a escrita em memória.
- Atualização e Preparação para o Próximo Bloco:
  - Após processar o bloco, o número do bloco é incrementado;
  - A função então prepara para o próximo bloco, garantindo que o buffer de dados esteja limpo para o próximo ciclo de processamento.

3.9.6 Decode Base64

Após o processamento da mensagem, o *payload* bruto precisa ser convertido para o formato binário antes de ser armazenado. O processo é descrito abaixo:

1. Validação: Primeiro é verificado se o comprimento do conjunto de caracteres base64 (`b64`) é um múltiplo de 4. Se não for, a função retorna 0, indicando um erro na entrada.
2. Decodificação:
   a) A *string* base64 é processada em blocos de 4 caracteres por vez;
   b) Cada bloco de 4 caracteres é convertido em até 3 bytes binários, utilizando a função auxiliar `get_number_b64_from_symbol_0_64` para mapear os caracteres base64 para seus valores numéricos correspondentes;
   c) Os bytes decodificados são armazenados no array `imageBytes`.
3. Interrupção: Se a função `get_number_b64_from_symbol_0_64` retornar `0xff`, a decodificação é interrompida.

A conversão de um caractere base64 em seu valor numérico correspondente consiste em retornar um valor entre 0 e 63 (0x00 a 0x3F), que é o intervalo usado para representar os 64 caracteres válidos do alfabeto base64. Esta função opera da seguinte forma:

1. Se o caractere está entre `0x41` (A) e `0x5a` (Z), o valor retornado é `b - 0x41`, mapeando

letras maiúsculas para valores de 0 a 25;

2. Se o caractere está entre `0x61` (a) e `0x7a` (z), o valor retornado é `b - 0x47`, mapeando letras minúsculas para valores de 26 a 51;
3. Se o caractere está entre `0x30` (0) e `0x39` (9), o valor retornado é `b + 0x4`, mapeando dígitos para valores de 52 a 61;
4. O caractere + (`0x2b`) é mapeado para 62;
5. O caractere / (`0x2f`) é mapeado para 63;
6. O caractere de *padding* = (`0x3d`) é mapeado para `0xff`, indicando o *padding*;
7. Qualquer outro caractere retorna `0xff`, indicando um caractere inválido ou inesperado.

3.9.7 Armazenamento de Blocos

O processo de armazenamento de blocos, após a conversão em binário, durante a atualização de *firmware* considera os seguintes pontos:

1. **Identificação do Tipo de Bloco**: Primeiro, verifica-se se o bloco convertido contém informações de cabeçalho (o primeiro bloco da sequência) ou se é um bloco de dados regular.
   - Se for um bloco de cabeçalho, ele contém metadados importantes como a versão do *firmware* e o tamanho total da imagem.
2. **Armazenamento Temporário do Cabeçalho**: Se o bloco pertence ao cabeçalho da imagem (definido pelo número do bloco), ele é armazenado temporariamente em um *buffer* na memória RAM até que todo o cabeçalho seja recebido.
3. **Validação do Cabeçalho Completo**: Quando o último bloco do cabeçalho é recebido, o cabeçalho completo é validado. Se a validação falhar, o processo de atualização é cancelado.
4. **Cálculo de Páginas e Pré-apagamento da Memória Flash**: Após a validação do cabeçalho, o sistema calcula o número de páginas de memória flash externa que serão necessárias para armazenar a nova imagem. Este cálculo considera dois pontos principais:
   - **Tamanho da imagem**: Este é incluso no cabeçalho da imagem e varia dependendo do conteúdo da nova imagem;
   - **Número de bytes por bloco**: É o número fixo definido tanto na NIC, quanto na aplicação que enviará os blocos. Neste trabalho foi definido o total de 256 bytes para cada bloco.

Em seguida, é realizado um pré-apagamento dessas páginas na memória flash externa para garantir que não haverá sobreposição de dados durante a escrita.

5. **Escrita do Cabeçalho na Memória Flash**: Caso a validação seja bem-sucedida, o cabeçalho da imagem é então escrito na memória flash externa. Se o bloco de dados recebido contiver bytes adicionais que não pertencem ao cabeçalho, esses bytes são imediatamente gravados na memória flash externa após o cabeçalho.
6. **Cálculo de Endereço para Blocos Subsequentes**: Para os blocos subsequentes, que contêm os dados reais da imagem, o sistema calcula o endereço na memória flash externa onde os dados devem ser escritos, com base na posição do bloco dentro da sequência total de blocos e no tamanho da memória.
7. **Gerenciamento de Erros durante a Escrita**: Se qualquer erro ocorrer durante a escrita na memória flash externa, o processo de atualização é interrompido e um erro é retornado. Caso contrário, o sistema retorna um status de sucesso e solicita o próximo bloco de dados.
8. **Atualização do Progresso**: Após armazenar o bloco, a NIC atualiza os registros de progresso, indicando que aquele bloco foi gravado com sucesso. Isso é essencial para permitir a retomada da atualização a partir do ponto correto caso o processo seja interrompido.
9. **Repetição do Processo para Blocos Subsequentes**: Esse ciclo de recepção, conversão, cálculo do endereço e escrita é repetido para cada bloco subsequente até que todos os blocos que compõem a imagem de *firmware* tenham sido armazenados na memória flash externa.

3.9.8 Finalização do OAD

A finalização do OAD consiste na validação final da imagem de *firmware* e o armazenamento de metadados. O processo é descrito em detalhes a seguir:

1. **Verificação de CRC:**
   - Primeiro é feito a verificação do CRC da nova imagem utilizando uma função criada pela própria desenvolvedora do MCU. Esta função primeiro valida os parâmetros fornecidos, depois calcula o número total de páginas necessárias para armazenar a imagem e determina o número de bytes na última página. O cálculo do CRC é iniciado com um valor inicial de 0xFFFFFFFF. A função então itera sobre todas as páginas e buffers da imagem e para cada um desses *buffers*, lê os dados e atualiza o valor do CRC byte a byte. Se um *buffer* não estiver completo, o tamanho da leitura é

ajustado de acordo.

- Após processar todos os bytes de todas as páginas e buffers, a função lê *buffers* adicionais, se necessário, especialmente quando se utiliza memória flash externa. Finalmente, a função complementa o valor do CRC (fazendo XOR com 0xFFFFFFFF) e retorna o valor CRC32 calculado.
- Se o CRC retornar um erro (ou seja, a verificação falhar), sinalizando que os dados da imagem estão corrompidos, o processo de validação é interrompido.

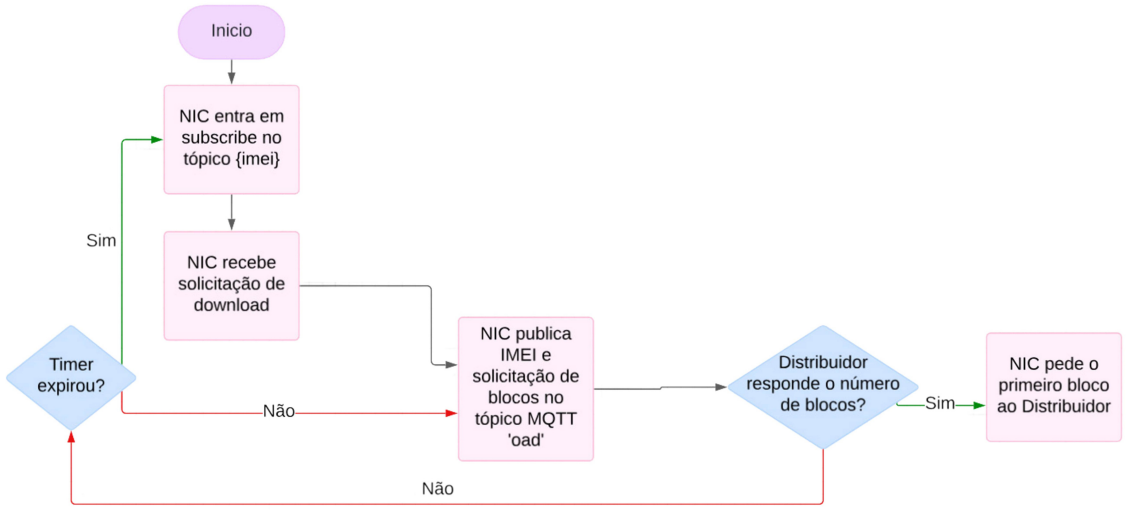
2. **População da Estrutura de Metadados Externos:** Se o CRC for bem-sucedido, a estrutura de metadados da imagem externa é então preenchida com informações extraídas do cabeçalho da imagem. Além disso, O ID da imagem externa é adicionado e o endereço da imagem e o contador são configurados. Por fim, novos campos são definidos nesse cabeçalho para indicar que a imagem precisa ser copiada e que o CRC é válido. Essas duas novas informações indicam que a imagem é válida para o BIM, de forma que caso os metadados contenham essas informações, quando o sistema for reiniciado o BIM copiará a imagem para a memoria interna e inicializará o *firmware* a partir dessa nova imagem.
3. **Gravação e Substituição de Metadados:** Os metadados antigos da memória flash são apagados para garantir que os novos metadados sejam armazenados corretamente. Os novos metadados, definidos no item anterior são então gravados na memória flash externa.
4. **Finalização:**
   - Se todas as operações forem bem-sucedidas, é correto afirmar a validação da imagem e a atualização dos metadados foram concluídas com sucesso.
   - Considerando que o processo de transferência e armazenamento foi concluído, é enviado uma nova publicação para a aplicação, o qual estava enviando os blocos, indicando que seja interrompido o envio de blocos para esta NIC.

3.10 Fluxo do OAD

3.10.1 Conexão com o Script e Número de Blocos

Esta figura mostra o processo de conexão inicial com a aplicação distribuidora e a quantidade de blocos que serão transferidos durante o processo de OAD (Figura 6).

Figura 6 – Conexão com o Script

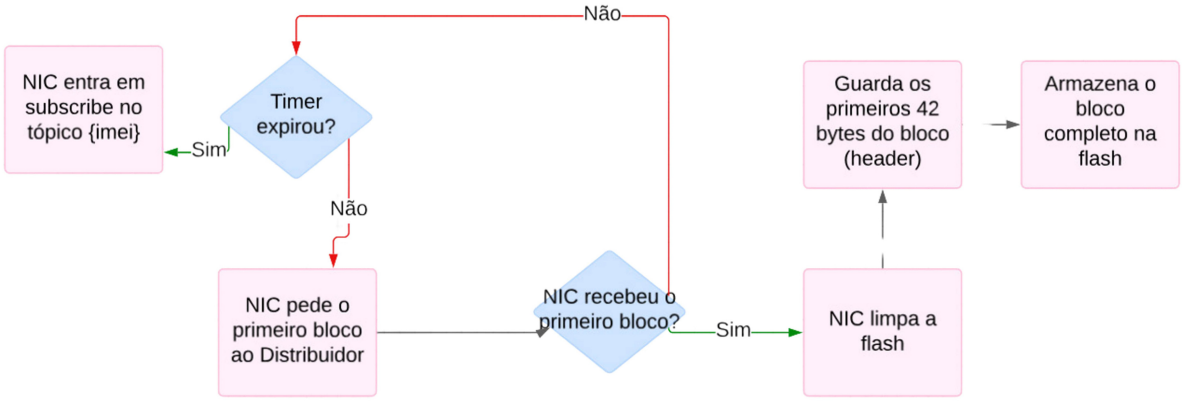


Fonte: Autor.

3.10.2 Primeiro Bloco

Esta figura ilustra a transferência do primeiro bloco de dados no processo de OAD (Figura 7).

Figura 7 – Primeiro Bloco

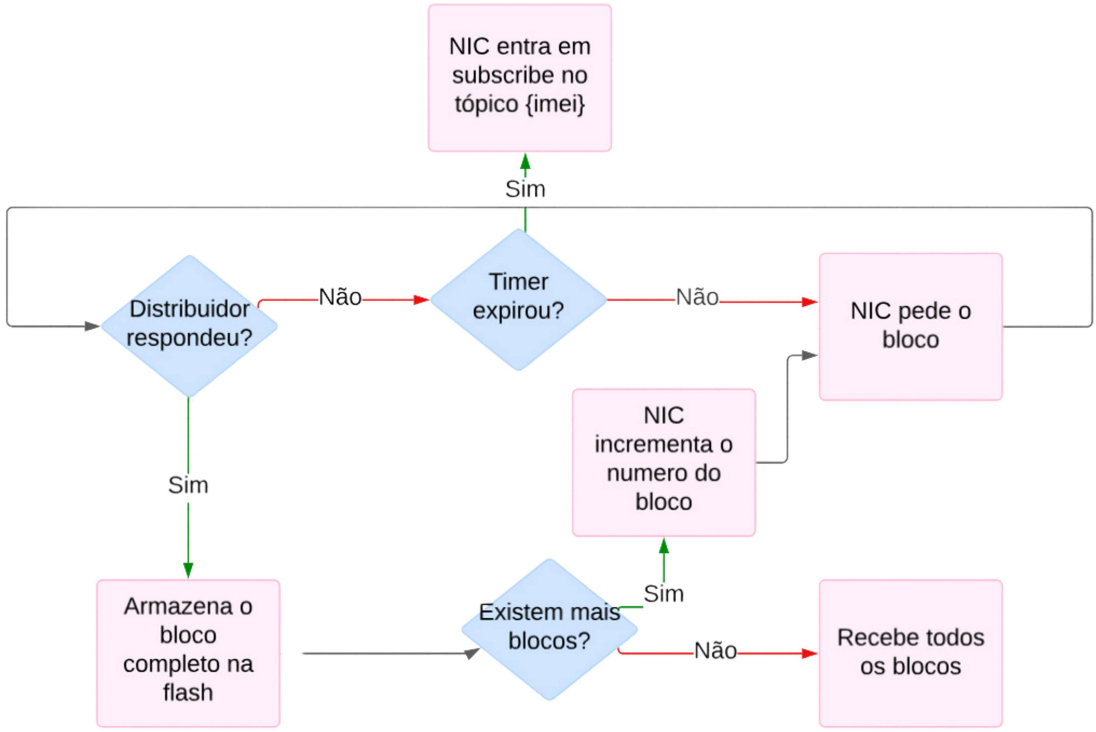


Fonte: Autor.

3.10.3 Blocos Restantes

Esta figura mostra a transferência dos blocos subsequentes após o primeiro bloco (Figura 8).

Figura 8 – Blocos Restantes

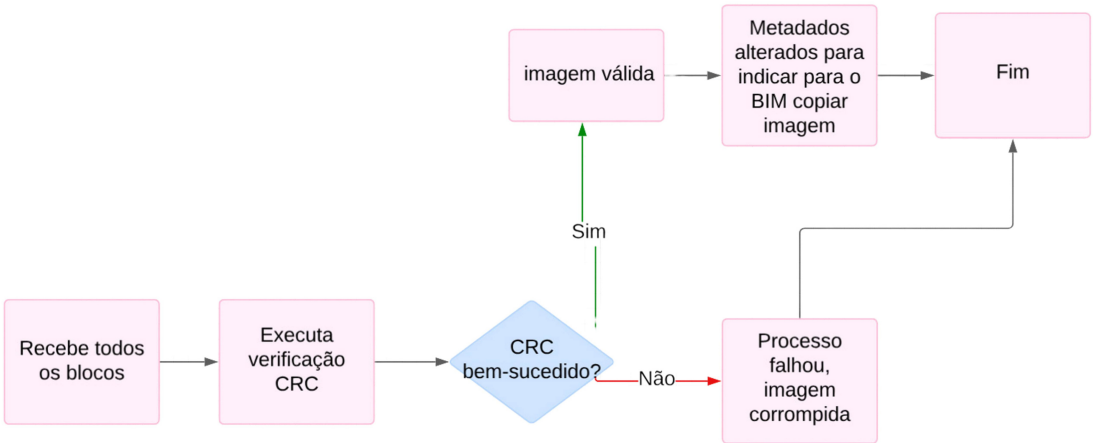


Fonte: Autor.

3.10.4 Validação da Imagem

Esta figura descreve o processo de validação da imagem de firmware após a transferência de todos os blocos (Figura 9).

Figura 9 – Validação da Imagem



Fonte: Autor.

4 RESULTADOS

Nesta seção, são apresentados os resultados obtidos com a implementação do sistema de atualização remota de *firmware* utilizando o protocolo MQTT. A análise dos resultados é realizada com base em critérios estabelecidos nas seções anteriores. A Seção 4.1 discute o tempo de atualização, enquanto a Seção 4.2 aborda a integridade do *firmware*. A confiabilidade das mensagens trocadas durante o processo de atualização é explorada na Seção 4.3. O cenário de teste é descrito na Seção 4.4, e a comparação com trabalhos relacionados é apresentada na Seção 4.5.

4.1 Tempo de Atualização

O tempo de atualização do *firmware* é um fator crucial para avaliar a eficiência do sistema de OAD. Esse tempo pode ser dividido em três componentes principais: o tempo para requisitar um bloco via rotina de publicação, o tempo para entrar em modo de escuta via rotina de subscrição e o tempo para receber a mensagem correspondente ao bloco solicitado.

Cada um desses componentes pode ser influenciado por diversas variáveis, como a qualidade do sinal de comunicação e a eficiência do processamento da NIC. Em condições ideais, nas quais a comunicação ocorre sem interrupções ou perda de pacotes, o tempo para requisitar e receber cada bloco é de aproximadamente 60 segundos. Esse valor inclui o tempo necessário para processar os comandos `AT` na função de publicar, o tempo gasto processando as respostas, incluindo o tempo de exibição de mensagens de depuração e o tempo para a recepção do bloco.

Entretanto, em situações nas quais a qualidade do sinal é comprometida, o sistema pode enfrentar dificuldades na comunicação, resultando em reenvios de blocos que não foram recebidos corretamente. Nessas condições adversas, foi considerado que o tempo de atualização dobrou, pois alguns blocos poderão ser requisitados mais de duas vezes, porém haverá blocos que serão enviados na primeira tentativa.

Para ilustrar o impacto dessas variáveis no tempo total de atualização, foram considerados cenários para quatro tamanhos de blocos diferentes. Em cada cenário, variou-se a condição da comunicação.

O tempo total para cada cenário foi calculado multiplicando-se o número de blocos necessários para transferir um arquivo de 98.948 bytes, que foi utilizado para testes, pelo tempo gasto para requisitar e receber cada bloco. Em condições perfeitas, o tempo é de aproxima-

damente 1 minuto por bloco, enquanto em condições ruins o tempo é de aproximadamente 2 minutos por bloco. O gráfico gerado a seguir ilustra esses tempos, permitindo uma comparação visual clara dos diferentes cenários analisados.

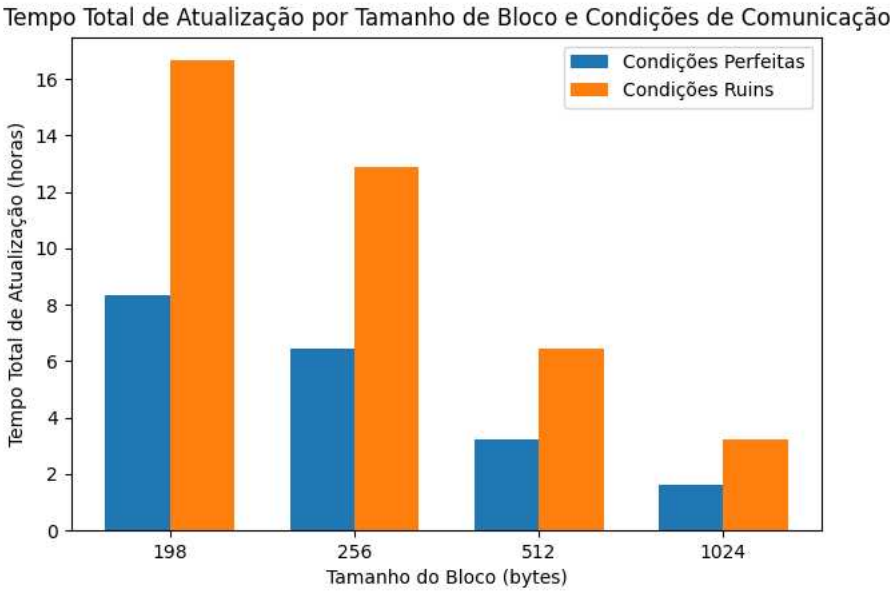


Figura 10 – Tempo de atualização em função do tamanho do firmware.

4.2 Integridade do Firmware

A integridade do *firmware* após a atualização foi verificada utilizando a função de validação CRC. Os testes mostraram que o sistema consegue verificar com precisão se o *firmware* foi atualizado corretamente. Para os casos de blocos de 256 e 512 bytes foram coletados registros de atividade e estão expostos, respectivamente, na Listagem 1 e na Listagem 2. A Tabela 3 condensa os resultados obtidos na validação do *firmware* para diferentes cenários de teste.

Código-fonte 1 – Log de validação para blocos de 256 bytes

```
1 crcFromHdr = 0x8d1c28de (2367432926)
2 crcCalculated = 0x8d1c28de (2367432926)
3 OADStorage_imgFinalise: Writing Meta Data in the ext flash
    ...
4 I am inside readFlashPg
5   addr = 0x20000, len = 0x2c
6 flashStat = 0
```

```
OAD_IMAGE: sv:0004 bv:03 imgCpStat:fe crcStat:fe extFlAddr
   :00020000, counter:00000000
Erasing the old meta data...
I am inside eraseFlashPg
  addr = 0x2000 (8192), EFL_PAGE_SIZE = 4096 (0x1000)
Storing the new meta data...
I am in writeFlashPg in page = 2 and offset = 0
  addr = 0x2000 (8192)

OADStorage_imgFinalise() with success...
```

Código-fonte 2 – Log de validação para blocos de 512 bytes

```
crcFromHdr = 0x8d1c28de (2367432926)
crcCalculated = 0xd21c245c (3525059676)

END END END...
```

Tabela 3 – Taxas de sucesso na validação do firmware.

tamanho dos blocos (bytes)	Sucesso
198	Sim
256	Sim
512	Não
1024	Não

4.3 Confiabilidade das Mensagens

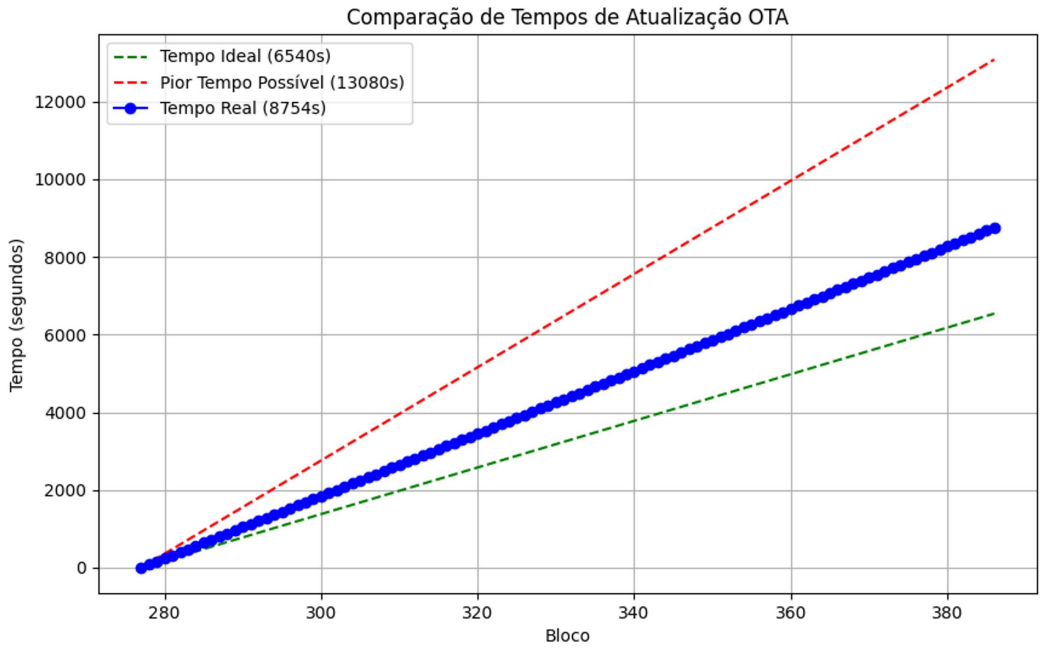
A confiabilidade das mensagens enviadas via MQTT foi avaliada com base na taxa de entrega e na integridade das mensagens. Os testes confirmaram que o protocolo MQTT mantém uma alta taxa de entrega e integridade das mensagens, com uma taxa de falhas mínima. Estes resultados foram possíveis porque cada mensagem trocada continha um padrão específico, com a palavra "*payload*"no início e o número do bloco no final. Esse formato permitiu garantir a integridade das mensagens, assegurando que, ao serem entregues, reproduzissem com exatidão o

conteúdo esperado.

4.4 Cenário de Teste

Esta seção apresenta o cenário de teste utilizado para avaliar a atualização remota de *firmware* por meio do processo de OAD, conforme ilustrado na Figura 11.

Figura 11 – Cenário de Teste



Fonte: Autor.

Para a avaliação do processo de OAD, foram utilizados blocos de 256 bytes. Inicialmente, a NIC foi reiniciada e o *uptime* foi reiniciado. Ao retomar o processo de OAD, a NIC encontrava-se no bloco 277 e o *uptime*, o qual se refere ao tempo total durante o qual um dispositivo permanece operacional desde a última reinicialização, indicava 35 segundos. O monitoramento do processo continuou até a conclusão do bloco 386, momento em que o *uptime* registrava 8789 segundos.

Análise do Tempo: Durante o experimento, foram transferidos 109 blocos ao longo de 8754 segundos. O tempo de transferência observado mostrou-se compatível com as previsões:

- No cenário ideal, a transferência de 109 blocos esperava-se ocorrer em 6540 segundos.
- No cenário limite, a transferência dos 109 blocos poderia se estender até 13080 segundos.

A análise dos tempos de transferência indica que o processo de OAD operou dentro dos parâmetros esperados, validando, assim, a eficiência do sistema implementado para a

atualização remota de *firmware*.

4.5 Comparação com Trabalhos Relacionados

Não foram encontrados trabalhos acadêmicos com proposta semelhante ao exposto neste trabalho, devido à natureza comercial do projeto. Contudo, a empresa OneRF também possui outra solução para o FOTA, a qual consiste em utilizar outro protocolo de comunicação para uma NIC similar, com o mesmo microcontrolador e mesmo sistema operacional. O protocolo adotado no projeto é o UDP, de forma que há vantagens e desvantagens em comparação.

A abordagem utilizando o protocolo UDP apresenta uma implementação simples, com uma vantagem significativa em termos de velocidade no processo de transferência dos blocos. No cenário de testes, o UDP foi capaz de completar a transferência em aproximadamente 30 minutos, enquanto o método proposto utilizando MQTT levou cerca de 6 horas. Essa diferença se deve, em parte, ao fato de que o UDP minimiza o tempo gasto com a exibição de mensagens de depuração e estabelece uma conexão direta entre cliente e servidor. No UDP, o cliente solicita diretamente um bloco via pacote e o servidor o envia sem intermediários. Já no método proposto com MQTT, o processo é mais complexo, exigindo o envio de comandos AT para publicar a solicitação no tópico, seguido por comandos adicionais para subscrever e escutar a resposta do servidor, o que aumenta o tempo total de execução.

Uma desvantagem significativa do UDP, no entanto, é a maior probabilidade de perda de pacotes, o que pode comprometer a integridade dos dados. Além disso, o tamanho dos blocos é limitado a 196 bytes por pacote, enquanto o MQTT demonstrou ser capaz de enviar blocos de até 256 bytes, com potencial para atingir tamanhos ainda maiores. Isso evidencia que, embora o MQTT tenha uma sobrecarga maior em termos de tempo de processamento e comunicação, ele oferece maior flexibilidade no tamanho dos blocos e maior confiabilidade na entrega dos pacotes, sendo uma alternativa promissora para casos em que a integridade dos dados é fundamental.

5 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, foi desenvolvido e testado um sistema de atualização remota de *firmware* para sistemas embarcados utilizando o protocolo MQTT. Os resultados obtidos demonstraram que o método proposto é eficiente, especialmente em cenários onde os blocos de dados possuem um tamanho menor ou igual a 256 bytes. Nesses casos, o processo de FOTA foi concluído com sucesso, evidenciando a confiabilidade do sistema em transmitir e armazenar o *firmware*.

Nos testes com blocos de 512 e 1024 bytes, no entanto, foram observadas falhas durante a atualização. Considerando que os *payloads* eram entregues corretamente, conforme evidenciado pelo padrão das mensagens trocadas—onde cada mensagem continha a palavra "*payload*"no início e o número do bloco no final—é provável que o problema esteja relacionado ao armazenamento dos blocos de dados maiores. Isso sugere que o sistema atual pode ter dificuldades em manipular e armazenar pacotes de dados maiores que 256 bytes, o que compromete a integridade do *firmware* atualizado.

Essa limitação abre caminho para pesquisas e desenvolvimentos futuros. Primeiramente, seria interessante explorar tamanhos de blocos intermediários entre 256 e 512 bytes, para identificar um possível ponto de equilíbrio que maximize o desempenho sem comprometer a confiabilidade. Além disso, futuras investigações poderiam focar em melhorias no mecanismo de armazenamento dos blocos de dados. Duas abordagens podem ser consideradas: uma possível repartição das mensagens após a recepção via MQTT ou o desenvolvimento de uma nova metodologia de armazenamento que permita a manipulação de pacotes de dados maiores.

Essas melhorias podem aumentar a eficiência e a robustez do sistema, permitindo uma atualização remota de *firmware* mais confiável e adaptável a diferentes condições de rede e requisitos de sistema.

REFERÊNCIAS

DODDAPANENI, K.; LAKKUNDI, R.; RAO, S.; KULKARNI, S. G.; BHAT, B. Secure fota object for iot. In: **2017 IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops)**. [*S. l.*: *s. n.*], 2017. p. 154–159.

GS M2M. **AT Commands Manual for Cinterion® EXS82-W**. 01.100a. ed. [*S. l.*], 2020. Acesso em: 01 ago. 2024. Disponível em: https://www.gs-m2m.de/fileadmin/Bilder/GSM_Module/Module/EXS62_82/exs82-w_atc_v01100a.pdf.

INNOVATIONS eG. **What is Mosquitto MQTT?** 2024. Accessed: 2024-08-13. Disponível em: https://www.eginnovations.com/documentation/Mosquitto-MQTT/What-is-Mosquitto-MQTT.htm.

Mozilla Developer Network. **Base64**. n.d. Accessed on August 10, 2024. Disponível em: https://developer.mozilla.org/en-US/docs/Glossary/Base64. Acesso em: 2024-08-10.

NLT Telecom. **IoT Celular**. s.d. Accessed on August 19, 2024. Disponível em: https://www.nlt.com.br/iot-celular. Acesso em: 2024-08-19.

OASIS Committee. **MQTT Version 3.1.1**. 2014. Último acesso em 23 de julho de 2024. Disponível em: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

OIKONOMOU, G.; DUQUENNOY, S.; ELSTS, A.; ERIKSSON, J.; TANAKA, Y.; TSIFTES, N. The Contiki-NG open source operating system for next generation IoT devices. **SoftwareX**, v. 18, p. 101089, 2022. ISSN 2352-7110.

PEREIRA, M. C. **Microcontroladores e Microprocessadores: Um Enfoque Prático**.

SILBERSCHATZ PETER B. GALVIN, G. G. A. **Operating System Concepts**. 10th. ed. [*S. l.*]: Wiley, 2018.

TANENBAUM, A. S.; BOS, H. **Modern Operating Systems**. 4th. ed. [*S. l.*]: Pearson, 2015.

Telit. **EXS82 IoT Modules**. 2024. Acesso em: 23 jul. 2024. Disponível em: https://www.telit.com/devices/exs82/.

Texas Instruments. **CC1312R SimpleLink™ Sub-1 GHz Wireless Microcontroller Datasheet**. Dallas, TX, 2020. Accessed: 2024-07-23. Disponível em: https://www.ti.com/lit/ds/symlink/cc1312r.pdf?ts=1711049336354&ref_url=https%253A%252F%252Fwww.google.com%252F.

Texas Instruments. **Over-the-Air Download (OAD)**. s.d. Accessed: 20-Aug-2024. Disponível em: https://software-dl.ti.com/simplelink/esd/simplelink_cc13x2_26x2_sdk/4.20.00.35/exports/docs/proprietary-rf/proprietary-rf-users-guide/oad/tools.html.

ZHANG, Z.; OCHIAI, H.; ESAKI, H. An iot application-layer protocol modem: A case study on interfacing ieee 1888 with at commands. In: **2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)**. [*S. l.*: *s. n.*], 2017. p. 346–349.