UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

RUAN FLANETO CARTIER

SOLUÇÕES DE BAIXO CUSTO PARA IMPLEMENTAÇÃO DO PROTOCOLO MMS EM SISTEMAS EMBARCADOS

Recife
2024

RUAN FLANETO CARTIER

SOLUÇÕES DE BAIXO CUSTO PARA IMPLEMENTAÇÃO DO PROTOCOLO MMS EM SISTEMAS EMBARCADOS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Engenheiro Eletricista com ênfase em Sistemas de Potência.

Orientador(a): Prof. Dr. Marcio Rodrigo Santos de Carvalho

Coorientador(a): Prof. Msc. Geraldo Leite Maia Junior

Recife
2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Cartier, Ruan Flaneto.
Soluções de baixo custo para implementação do protocolo MMS em sistemas embarcados / Ruan Flaneto Cartier. - Recife, 2024.
91 p. : il., tab.

Orientador(a): Marcio Rodrigo Santos de Carvalho
Cooorientador(a): Geraldo Leite Maia Junior
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Elétrica - Bacharelado, 2024.
Inclui referências, apêndices.

1. IEC 61850. 2. Dispositivo Eletrônico Inteligente. 3. protocolo MMS. 4. Raspberry Pi. 5. ESP32. I. Carvalho, Marcio Rodrigo Santos de. (Orientação). II. Junior, Geraldo Leite Maia. (Coorientação). IV. Título.

620 CDD (22.ed.)

RUAN FLANETO CARTIER

SOLUÇÕES DE BAIXO CUSTO PARA IMPLEMENTAÇÃO DO PROTOCOLO MMS EM SISTEMAS EMBARCADOS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Engenheiro Eletricista com ênfase em Sistemas de Potência.

Aprovado em: 21/03/2024.

BANCA EXAMINADORA

_____
Prof. Dr. Marcio Rodrigo Santos de Carvalho
Universidade Federal de Pernambuco

_____
Prof. Msc. Geraldo Leite Maia Junior
Universidade Federal de Pernambuco

_____
Prof. Msc. Eduardo Oliveira Barbosa
Universidade Federal de Pernambuco

_____
Prof. Dr. Eduardo José Barbosa
Universidade Federal de Pernambuco

Este trabalho é dedicado à minha família, aos professores do departamento de Engenharia Elétrica e Sistemas de Potência da Universidade Federal de Pernambuco e à Gerência de Procedimentos Operativos do Operador Nacional do Sistema Elétrico.

AGRADECIMENTOS

Ao concluir este trabalho, reconheço que ele reflete a contribuição das pessoas que estiveram presente na minha trajetória. Diante disso, surge a necessidade de expressar minha gratidão, pois cada um contribuiu positivamente para o meu desenvolvimento.

Primeiramente, dedico este agradecimento à minha família, pilar fundamental em minha vida. À minha mãe, Cláudia Maria Brasil Flaneto, e ao meu pai, Ivo Ramiro Ferreira Cartier, agradeço o amor e cuidado que serviram como alicerce para seguir em frente. E ao meu irmão, Renan Flaneto Cartier, que, apesar de trilharmos caminhos separados, seguimos unidos.

Ao Professor Márcio Rodrigo Santos de Carvalho, meu orientador do TCC, expresso minha gratidão pela sua orientação e por acreditar em mim mesmo quando eu não acreditava ser possível atender aos prazos.

Aos queridos professores do Departamento de Engenharia Elétrica (DEE), meu reconhecimento pela dedicação e compromisso com a formação de novos profissionais. Através do trabalho conjunto realizado ao longo dos últimos anos, foi possível trilhar o caminho que me levou à formação como engenheiro eletricista.

Meu agradecimento especial se estende para além do departamento. À Paula Suemy Arruda Michima, sou profundamente grato pelo apoio e pelas oportunidades proporcionadas, especialmente pelo seu apoio no LAB3C. A Guilherme Nunes Neto e a João Paulo Cerquinho Cajueiro, minha gratidão pelo apoio durante a participação no projeto Maracatronics e pela experiência da publicação no WRE.

Por último, mas não menos importante, desejo expressar meu mais sincero agradecimento à equipe Nordeste da Gerência de Procedimentos Operativos do ONS (PDP-Recife), e em particular a Robson Luis Da Silva. Sua disposição, seu respeito e seu bom humor tornaram a experiência de estágio ainda mais enriquecedora. Que nossos laços continuem fortalecidos e que possamos compartilhar muitos mais momentos de aprendizado e sucesso juntos.

Com profundo apreço,

Ruan Flaneto Cartier

You can never understand everything.
But, you should push yourself to understand the system.

- Ryan Dahl (Criador do Node JS)

RESUMO

Como meio de garantir a disponibilidade de equipamentos elétricos em uma subestação, em Função Transmissão por exemplo, é essencial manter a comunicação entre os dispositivos operacional. Para gerir a qualidade da supervisão e operação dos ativos de uma instalação elétrica, sistemas e padrões de comunicação foram desenvolvidos ao longo do tempo para promover o desenvolvimento de Sistemas de Automação de Subestações (SAS). Estes padrões carregam consigo um crescente grau de complexidade, tornando imprescindível a execução de testes ou simulações que validem o comportamento dos elementos no sistema. Portanto, para fins de pesquisa, este trabalho tem como proposta o emprego de sistemas embarcados munidos de conexão Wi-Fi, como o Raspberry Pi e o ESP32, para a emulação de Dispositivos Eletrônicos Inteligentes (IEDs) como meio de validar conceitos e avaliar desempenhos dos recursos estabelecidos pela norma IEC 61850. Especificamente, o objetivo é contemplar uma alternativa acessível aos dispositivos comerciais empregados em campo, que possa ao menos interagir com um cliente pelo protocolo MMS *(Manufacturing Message Specification)* via Rede de Área Local (LAN) sem fio atendendo aos requisitos de temporização na transmissão de dados previsto na referida norma. A metodologia consiste em fazer um levantamento bibliográfico da norma IEC 61850 e sua implementação em *software*. Os resultados obtidos atendem aos requisitos de temporização previstos na norma IEC 61850.

Palavras-chave: IEC 61850; Dispositivo Eletrônico Inteligente; protocolo MMS; Raspberry Pi, ESP32.

ABSTRACT

To ensure the electrical equipment availability in a substation, in a set of functionally interdependent installations for example, it is essential to maintain the devices communication. To handle the quality of the electrical system assets supervision quality, communication systems and standards have been developed over the time to promote the development of Substation Automation Systems (SAS). These standards carry with themselves high degrees of complexity so that the execution of tests and simulations that validate the behavior of the elements in a system is a must. Therefore, for means of researching, this text proposes the use of embedded systems equipped with Wi-Fi connectivity, such as Raspberry Pi and ESP32 microcontrollers, to emulate Intelligent Electronic Devices (IEDs) as means of validating concepts and evaluating the resources performance specified in the IEC 61850 standard. To be specific, the goal of this text is to cognize a low-cost alternative to expensive commercial commonly used devices, that can at only interact with a client through MMS *(Manufacturing Message Specification)* protocol by means of Wireless Local Area Network (WLAN) and matching time requirements set out in the standard. The methodology consists of a bibliographical survey of the IEC 61850 standard and its implementation in *software*. The obtained results meet the time requirements set out in the IEC 61850 standard.

Keywords: IEC 61850; Intelligent Electronic Device, MMS protocol, Raspberry Pi, ESP32.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ACSI	Abstraction Comunication Service Interface
BRCB	Buffered Report Control Block
CID	Configured IED Description
COS	Centro de Operação
CRC	Cyclic Redundancy Check
DA	Data Attribute
DEE	Departamento de Engenharia Elétrica
DNP	Distributed Network Protocol
DO	Data Object
DS	Dataset
EPA	Enhanced Performance Architecture
ESP-IDF	Espressif IoT Development Framework
gcc	GNU Compiler Collection
GNU	GNU's Not Unix!
GOOSE	Generic Object Oriented Substation Event
hal	Hardware Abstraction Layer
IEC	International Electrotechnical Commission
ICD	IED Capability Description
IED	Inteligent Electronic Device
IHM	Interface Homem Máquina
JRE	Java Runtime Environment
LC	Logical connection
LD	Logical Device
LN	Logical Node
MMS	Manufacturing Message Specification
MU	Merging Units
OSI	Operation System Interconnection
PC	Physical Connection
PD	Physical Device
POSIX	Portable Operating System Interface
PSRAM	Pseudostatic Random Access Memory
PTP	Precision Time Protocol
RAM	Random Access Memory
RCB	Report Control Block
RTU	Remote Transmission Unit
SAS	Sistema de Automação de Subestações
SCD	Substation Configuration Description
SCL	Substation Configuration Language
SCL	Substation Configuration Language
SDK	System Development Kit
SIN	Sistema Interligado Nacional

SNTP	Simple Network Time Protocol
SPI	Serial Peripheral Interface
SRAM	Static Read Access Memory
SSD	System Specification Description
SSEG	Small Scale Embedded Generators
SV	Sampled Values
TC	Transformador de Corrente
TH	Transport Header
TP	Transformador de Potencial
URCB	Unbuffered Report Control Buffer
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Na última década, o Sistema Elétrico Brasileiro tem demonstrado constantes evoluções em sua capacidade de geração impulsionadas pela utilização de energias renováveis em larga escala, ao passo que as infraestruturas de transmissão não acompanharam o ritmo, permanecendo defasadas [1], perto de seus limites de carregamento. Adicionalmente, no âmbito da distribuição de energia elétrica, apesar da MMGD (Mini e Microgeração Distribuída) se apresentar como um agente disruptivo e diversificador da matriz elétrica, a sua rápida penetração aumenta a complexidade no gerenciamento e expansão da rede, além de causar variações na oferta e demanda do mercado de energia elétrica [2]. Entre outros motivos, a operação do Sistema Interligado Nacional (SIN) oferece desafios no que se refere ao ponto de operação e estabilidade, tornando imprescindível a adoção de padrões que assegurem a confiabilidade da comunicação entre equipamentos em uma subestação, equipamentos e Centro de Operação (COS) e entre subestações [3].

As subestações de energia elétrica possuem muitas variáveis associadas a dispositivos que precisam ser monitorados, tais como estado de disjuntores, medições em sensores de corrente e transformadores de tensão. Os operadores frequentemente precisam conectar ou desconectar seções da rede elétrica. Os dispositivos eletrônicos inteligentes (IEDs) da subestação transmitem dados do processo para as unidades terminais remotas (RTUs) localizadas na estação principal do COS. Os IEDs também são utilizados para energizar ou desenergizar os disjuntores e reguladores de tensão. A Figura 1 e a Figura 2 mostram exemplos ilustrativos de IED e RTU respectivamente.

Figura 1 – SIPROTEC 7SJ64 – Relé de proteção multifunção com sincronização

Fonte: Catálogo SIPROTEC 4 [4]

Figura 2 – RTU530 da Hitachi

Fonte: Guia de ativação de licença [5]

A norma IEC 61850 foi criada, inicialmente, para fornecer interoperabilidade (capacidade de dois ou mais dispositivos, independentemente de serem ou não do mesmo fabricante, de trocar informações sem que o processo comprometa funcionalidades relacionadas) entres os dispositivos em um Sistema de Automação de Subestações (SAS) [6] e para estabelecer requisitos funcionais e de desempenho que venham a atender sistemas de automação, cuidando do monitoramento, do controle e da operação. Entretanto sua aplicação foi estendida para subestações inteligentes de distribuição vindo a competir com o protocolo DNP3 com a vantagem de poder integrar equipamentos de diferentes classes, que é condição necessária para promover eficiência e aprimorar procedimentos e manutenção dos SAS [7].

Alguns pontos a destacar que a IEC 61850 traz, no quesito de interoperabilidade, em relação a padrões legados são:

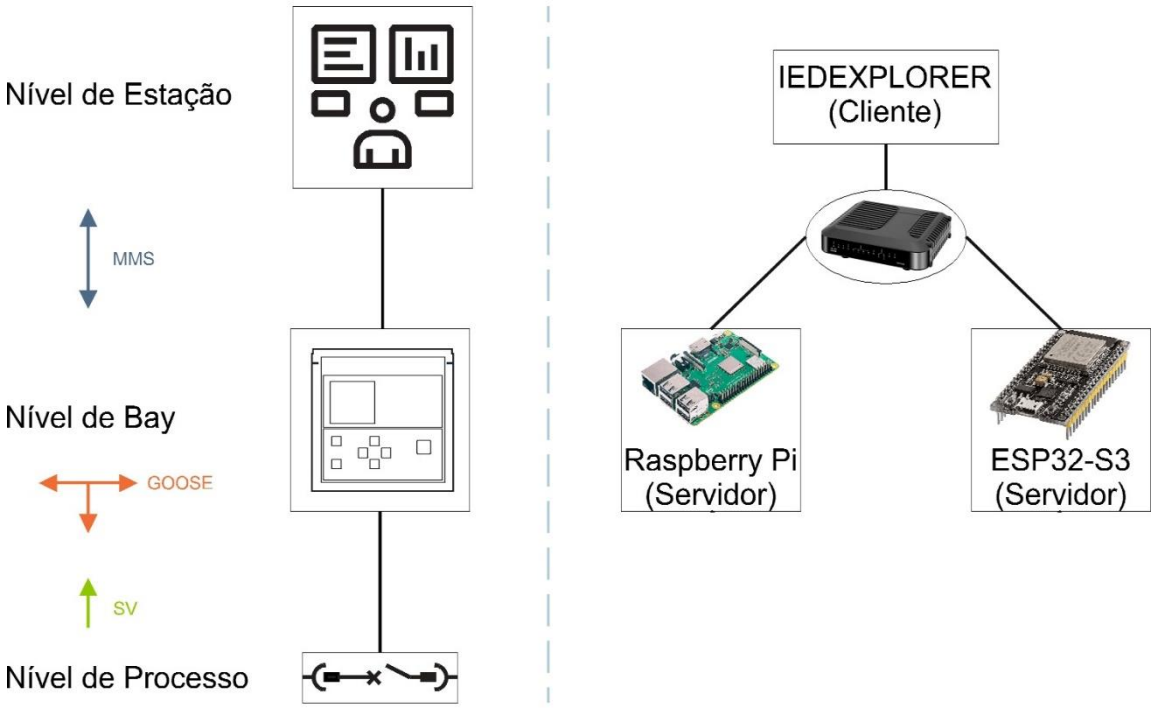
- Sistema a prova de futuro – *future-proof*: isto é feito com o desacoplamento das aplicações no SAS em relação à tecnologia de comunicação empregada por meio da aplicação de serviços e objetos abstratos (IEC 61850-7-2). Assim, por mais que os sistemas de comunicação evoluam, a aplicação ainda pode ser mantida ou adaptada apenas seguindo os requisitos de comunicação (IEC 61850-5);
- O perfil de comunicação é adaptável, possibilitando, inclusive o uso de protocolos aplicados em redes de comunicação modernas como principal meio de transmissão de dados. Isto permite altas velocidades de transferências e redução no cabeamento da infraestrutura da subestação;

- Autodescrição de dados, ocasionando configuração simplificada: isto é feito com a definição de uma linguagem padrão de configuração SCL, como uma variação da metalinguagem de marcação XML (IEC 61850-6) juntamente com a definição de uma série de modelos para funções de aplicação em um SAS (IEC 61850-7-4).

A IEC 61850 utiliza protocolos que permitem comunicação entre equipamentos heterogêneos (diferentes classes e fabricantes) de um SAS. Dentre eles: MMS (Manufacturing Message Specification), GOOSE (Generic Objected Oriented Substation Event) e SV (Sampled Values), relacionados ao SAS como no lado esquerdo da Figura 3 e os níveis hierárquicos serão representados no presente projeto conforme o lado direito da mesma figura.

Devido ao baixo custo de aquisição e à facilidade de logística de instalação, sistemas embarcados são costumeiramente utilizados para emular IEDs com a finalidade de explorar as características e benefícios da IEC 61850. É importante pontuar que o sistema computacional empregado possui recurso Wi-Fi, como trazido por Hwang [8].

Figura 3 – Associação dos níveis hierárquicos de um SAS com os recursos explorados no projeto



Fonte: Autor, 2024.

1.1 Objetivos

1.1.1 Geral

Investigar soluções de baixo custo para estabelecer comunicação entre IEDs (*Inteligent Electronics Devices* – Dispositivos Eletrônicos Inteligentes) através do protocolo MMS via rede LAN (Local Area Network – Rede de Área Local) sem fio em conformidade com os requisitos de temporização na transmissão de dados na norma IEC 61850.

1.1.2 Específicos

Dentre os objetivos específicos:

- Apresentar a norma IEC 61850, o protocolo MMS e os requisitos de temporização;
- Pesquisar trabalhos similares;
- Apresentar recursos (hardware, software e padrões) implementados;
- Detalhar o passo-a-passo da implementação de servidor MMS que represente um IED.

1.2 Organização do Trabalho

Este trabalho está estruturado com os tópicos de fundamentação teórica, desenvolvimento e resultados. A primeira parte da fundamentação teórica apresenta um escopo geral do protocolo DNP3 e da norma IEC61850, dando foco no protocolo MMS e os requisitos de temporização na comunicação; a segunda parte descreve as soluções encontradas na literatura; e a terceira parte apresenta os recursos utilizados: libIEC61850, ESP32, Raspberry Pi e ESP-IDF. O desenvolvimento do trabalho descreve a solução proposta, bem como as dificuldades enfrentadas durante a implementação/configuração dos recursos mencionados. Em seguida os resultados são apresentados e discutidos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 O protocolo DNP3 em subestações de energia elétrica

O protocolo DNP (Distributed Network Protocol) foi desenvolvido pela Westronic, Inc. (agora GE Harris) em 1990 para permitir a comunicação entre IEDs e RTUs em companhias de serviços essenciais, como por exemplo de eletricidade e água. Em 1993, a propriedade do protocolo “DNP 3.0 Basic 4” foi transferida para o recém-formado DNP Users Group, e suas especificações disponibilizadas para uso público [9]. Desde então, o DNP3 ganhou aceitação mundial, sobretudo na China, América Latina e Austrália, sendo bastante utilizado para a troca de mensagens de supervisão em subestações [9].

2.1.1 Características do protocolo DNP3

O DNP3 é um protocolo aberto, robusto e eficiente que utiliza o método de comunicação mestre-escravo com suporte para o envio de mensagens não solicitadas (assíncronas) por iniciativa de estações escravas [9]. Esta importante característica do protocolo DNP3 é denominada de envio espontâneo de informação. Adicionalmente, o DNP3 apresenta as seguintes características:

- Suporta múltiplos tipos de dados em uma única mensagem de requisição ou de resposta;
- Suporta a transmissão de grandes pacotes de dados;
- Inclui apenas dados alterados nas mensagens de resposta;
- Atribui prioridades a itens de dados;
- Suporta transferência de arquivos;
- Suporta múltiplos mestres e operações ponto a ponto, multiponto ou hierárquica;
- Permite objetos definidos pelo usuário.

2.1.1.1 Tipos de dados

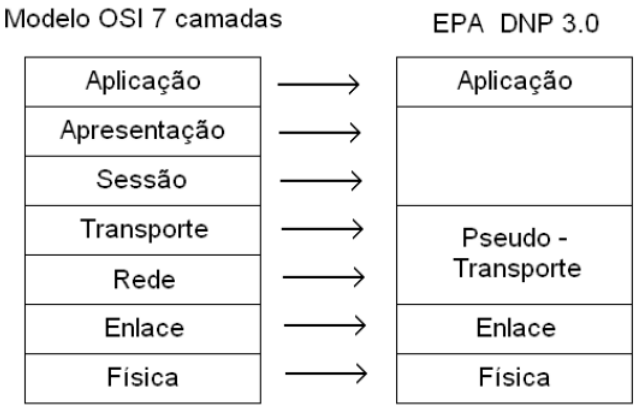
No DNP3, cada tipo de dado é um grupo de objetos que inclui:

- Entradas binárias (útil para monitorar dispositivos de dois estados, por exemplo, se um disjuntor está fechado ou desarmado);
- Saídas binárias (útil para enviar ações de controle, tais como fechar ou desarmar um disjuntor);
- Entradas analógicas (útil para monitorar valores de corrente, potência, tensão);
- Saídas analógicas (útil para enviar o *set point* da variável do processo, por exemplo);
- Contadores (útil para monitorar valores de kWh)
- Hora e data;
- Objetos de transferência de arquivos;
- Entre outros.

2.1.1.2 Arquitetura de camadas

O modelo de camadas do DNP3 é baseado na arquitetura EPA (Enhanced Performance Architecture), uma versão simplificada do modelo de referência OSI que contém apenas as camadas de aplicação, enlace de dados e física [9]. Entretanto, para suportar funções avançadas das RTUs, o DNP3 implementa uma camada chamada de pseudo-transporte. A arquitetura de camadas do DNP3 é apresentada na figura Figura 4.

Figura 4 – DNP3 e o modelo OSI



Fonte: Autor, 2024.

2.1.1.3 Camada de aplicação

A camada de aplicação é responsável por criar mensagens de requisições/respostas e mensagens não-solicitadas. Cada fragmento (unidade de dados da camada de aplicação) começa com o cabeçalho da camada de aplicação seguido por uma ou mais combinações de cabeçalho/dados do objeto [9].

O cabeçalho da camada de aplicação contém um código de controle da aplicação e um código de função da aplicação. O código de controle deve indicar:

- Se o fragmento é parte de uma mensagem;
- Se o recebimento do fragmento deve ser reconhecido/confirmado;
- Se o fragmento foi não solicitado;
- O número de sequência do fragmento;

O código de função indica o propósito da mensagem. Exemplos incluem *confirm*, *read*, *write*, *restart* etc.

2.1.1.4 Camada de pseudo-transporte

É responsável por decompor os fragmentos em unidades menores chamadas de segmentos. Cada segmento começa com um código de função de um byte que indica se a unidade de dados é o primeiro segmento da mensagem, o último segmento da mensagem ou ambos (para mensagens de um único segmento). O código de função

também inclui um número de sequência que permite detectar segmentos fora de ordem ou perdido [9].

2.1.1.5 Camada de enlace de dados

A camada de enlace de dados gerencia a conexão lógica entre o transmissor e o receptor das informações além de prover serviços para a detecção de erros de transmissão. Para o DNP3, isso é realizado começando cada quadro de enlace de dados com um cabeçalho de enlace de dados e inserindo um CRC (verificação cíclica de redundância - algoritmo de identificação de erros nos dados) de 16 bits a cada 16 bytes do quadro. Um quadro é uma parte de uma mensagem completa comunicada pela camada física. Cada quadro possui um endereço de origem de 16 bits e um endereço de destino de 16 bits, que pode ser um endereço de transmissão broadcast (0xffff). As informações de endereço, juntamente com um código de início de 16 bits, o comprimento do quadro e um byte de controle de enlace de dados, estão contidas no cabeçalho de enlace de dados de 10 bytes.

O byte de controle de enlace de dados indica se o quadro é de reconhecimento, não-reconhecimento, se o quadro requer ou não confirmação de recebimento e se a conexão lógica precisa ser refeita [9].

2.1.1.6 Camada física

Geralmente, o DNP é especificado em uma camada física serial simples, usando meios físicos como cobre por meios dos padrões RS-232 e RS-485, fibra, rádio ou satélite. Alternativamente, pode-se implementar o DNP3 por meio de uma conexão Ethernet encapsulando na pilha TCP/IP, como descrito na Seção 2.1.2.

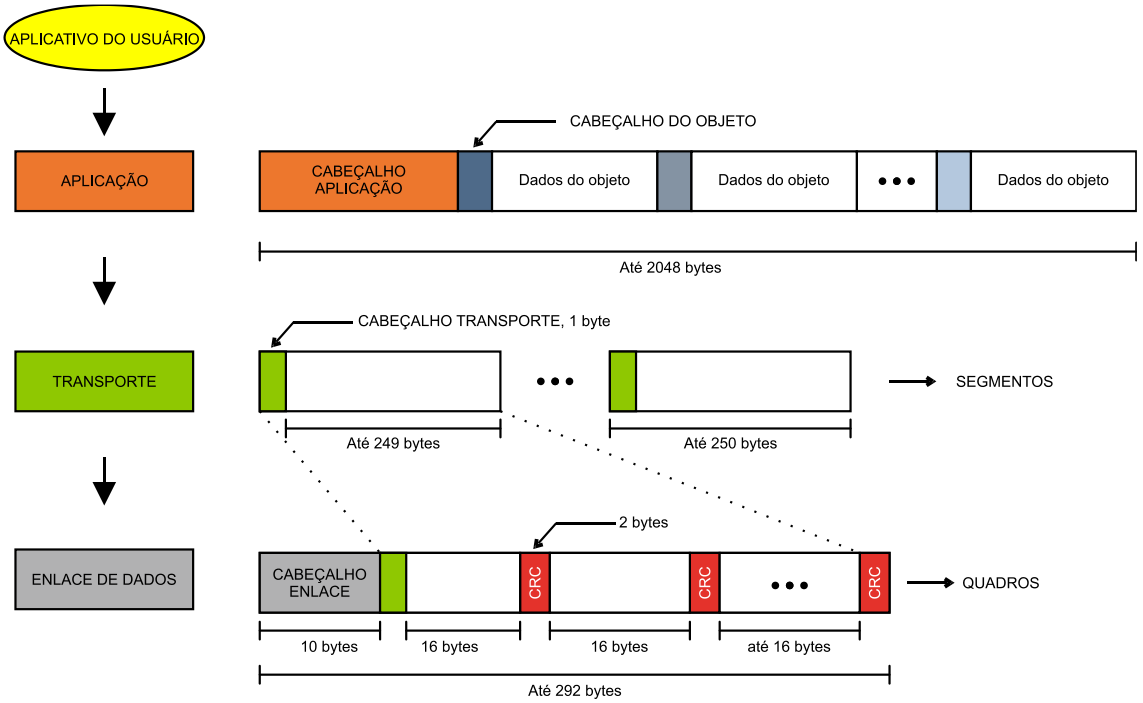
2.1.1.7 Unidade de dados

A Figura 4 ilustra as unidades de dados das camadas do protocolo DNP3, cujos detalhes são descritos a seguir e esboçados conforme a Figura 5. Os dados gerados na camada de aplicação (até 2048 bytes), chamados de fragmento, descem à próxima camada. A camada de pseudo-transporte decompõe a mensagem proveniente da

camada de aplicação em segmentos de até 250 bytes, incluindo o cabeçalho de transporte (TH). Esses segmentos são “etiquetados” com uma numeração sequencial por 6 bits do cabeçalho TH, para que, chegando ao destino, a camada de pseudo-transporte do destinatário possa reordenar a informação [9].

Na camada de enlace, cada segmento (250 bytes) recebe um novo cabeçalho de 10 bytes (sendo 2 bytes para o endereço de origem da mensagem e dois bytes para o endereço de destino) e mais 32 bytes de CRC. Na verdade, cada CRC é composto por 2 bytes, os quais são alocados em intervalos de 16 em 16 bytes de cada segmento. O frame DNP3 vai para o meio físico com até 292 bytes [9].

Figura 5 – Formato da mensagem DNP3



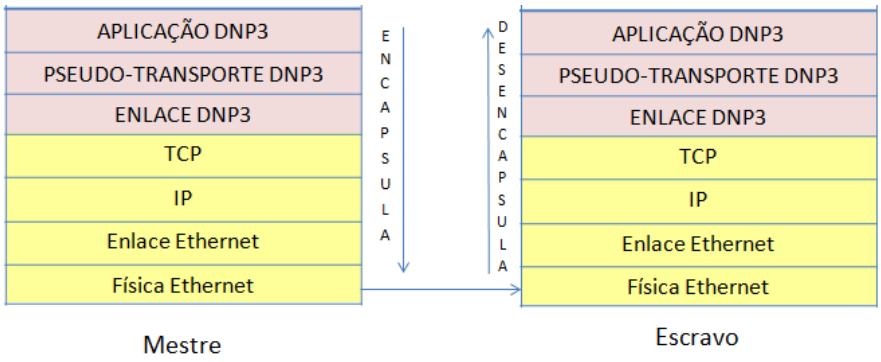
Fonte: Autor, 2024.

2.1.2 DNP3 sobre TCP/IP

Pode-se implementar o DNP3 sobre o conjunto de protocolos do padrão TCP/IP [9]. Neste caso, o frame proveniente da camada de enlace de dados anteriormente descrito (até 292 bytes) é encapsulado em TCP/IP. A Figura 6 ilustra o modelo de

camadas do DNP3 sobre TCP/IP. Neste caso, a arquitetura em camadas do DNP3 é empilhada no conjunto de protocolos do padrão TCP/IP, como mostrado na Figura 6.

Figura 6 – DNP3 encapsulado em TCP/IP



Fonte: Retirado de [10]

2.2 Visão Geral da IEC 61850

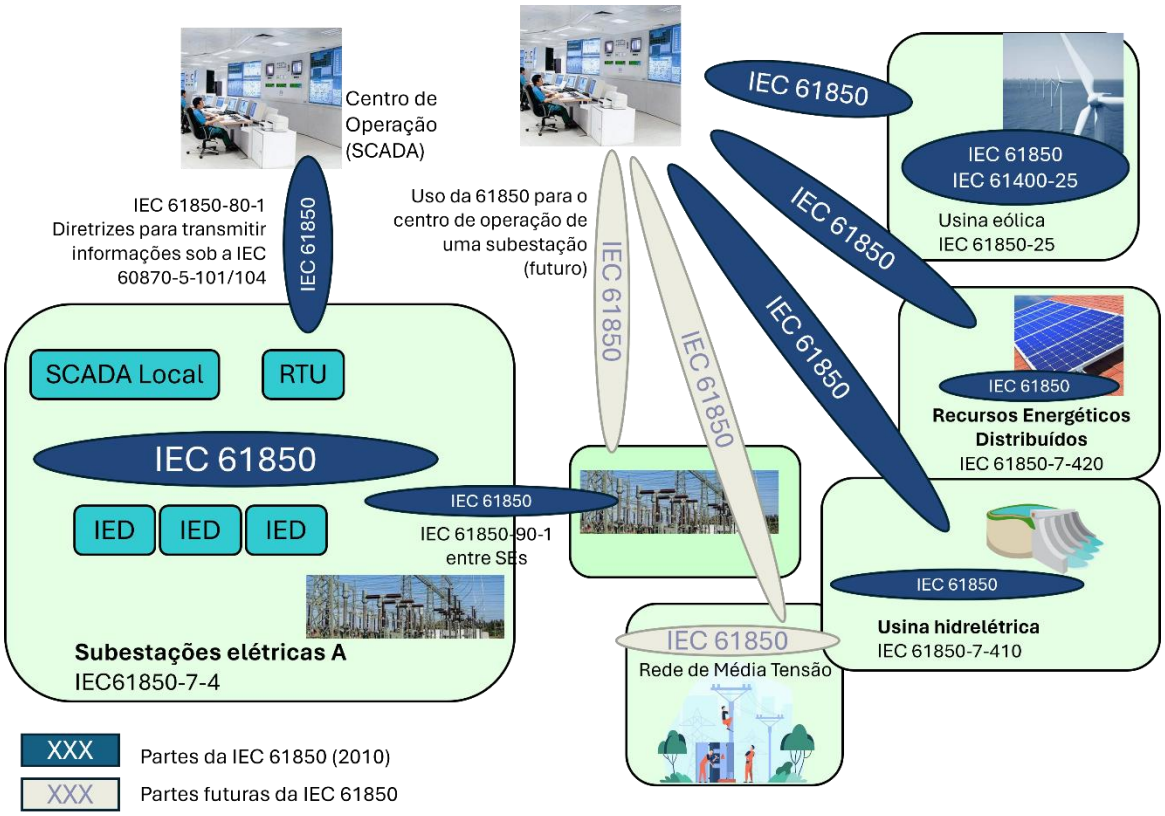
O DNP3 é um protocolo bem completo, porém nos quesitos de interoperabilidade há uma limitação na estrutura de modelagem de dados, bem como também, o seu encapsulamento sobre TCP/IP provoca o uso mais intenso da rede, sendo necessário fazer um *upgrade* da largura de banda [11]. Alternativamente, o protocolo DNP3 sem o encapsulamento tornaria necessário uma grande quantidade de cabos para interligar os equipamentos.

A Norma IEC 61850 foi publicada em 2004 pela IEC (Internactional Electrotechnical Commission) – organização que define padrões internacionais de tecnologias voltadas para geração, transmissão, distribuição de energia elétrica e assuntos afins, surgiu para tratar da padronização de comunicação entre equipamentos de proteção, medição, controle e supervisão de diferentes fabricantes em um SAS. No Brasil, a ABNT (Associação Brasileira de Normas Técnicas) toma os aspectos mais relevantes desta norma como base para definir a NBR 16932, a qual, sempre que for necessário avaliar alguma modificação ou implantação de algum Sistema de Automação em Subestação (SAS), deve ser consultada.

É importante ressaltar que o escopo da norma não se restringe ao domínio das subestações, trazendo conceitos de modelagem de usinas hidrelétricas (IEC 61850-

7-420), eólicas (IEC 6100-25), comunicação entre subestações (IEC 61850-90-1) e Recursos Energéticos Distribuídos (IEC 61850-7-420) [7]. Seu escopo de aplicação é ilustrado na Figura 7.

Figura 7 – Escopo de aplicação da norma IEC 61850



Fonte: Adaptado e traduzido da IEC 61850-1 [7]

Antes de prosseguir, é importante esclarecer a definição de um SAS. De acordo com a IEC61850-1 [7], em uma rede elétrica, um SAS é tido como um conjunto de dispositivos interligados que realiza a operação, a proteção e o monitoramento de outros equipamentos elétricos. Cabe destacar que além dos IEDs, a infraestrutura de rede de comunicação também é incluída no SAS. Então, em particular, esta norma define a comunicação entre os IEDs dentro das subestações e quais serão os requisitos de temporização.

A existência dos sistemas de automação em subestações foi possível graças ao surgimento e evolução dos microprocessadores embarcados em um IED, que veio a substituir os equipamentos eletromecânicos. Com esta tendência de automatização, houve a necessidade de garantir a interoperabilidade entre os equipamentos, isto é,

definir protocolos padrões de modo a estabelecer comunicação eficaz entre os dispositivos, uma vez que, eventualmente, em um SAS pode haver dispositivos de diferentes fabricantes, e se cada fabricante definir a sua própria forma de transmitir e receber dados, então haveria um problema de incompatibilidade ou, no mínimo, seriam aplicadas adaptações, podendo comprometer o desempenho da comunicação.

Nos subtópicos seguintes são listadas algumas definições de conhecimento geral sobre a norma, cujas principais partes de interesse para este trabalho estão resumidas na Tabela 1.

Tabela 1 – Partes da Norma IEC 61850 de interesse para este trabalho

Parte	**Título**
IEC 61850-1	Introdução e visão geral
IEC 61850-5	Requisitos de comunicação para funções e modelos de dispositivos
IEC 61850-6	Linguagem de descrição de em subestações elétricas relacionadas a IEDs
IEC 61850-7.1	Estrutura Básica de comunicação – Princípios e modelos
IEC 61850-7.2	Estrutura Básica de comunicação – Interface de serviço de comunicação abstrata (ACSI)
IEC 61850-7.3	Estrutura Básica de comunicação – Classes de dados comuns
IEC 61850-7.4	Estrutura Básica de comunicação – Classes de nós lógicos e classes de dados
IEC 61850-8.1	Mapeamento de comunicação específica (SCSM) – Mapeamento ao MMS (ISO 9506-1 e ISO 9506-2) e ISO/IEC 8802-3.

NBR 16932	Redes e Sistemas de Comunicação para Automação de Sistemas de Potência — Orientações sobre Engenharia de Rede

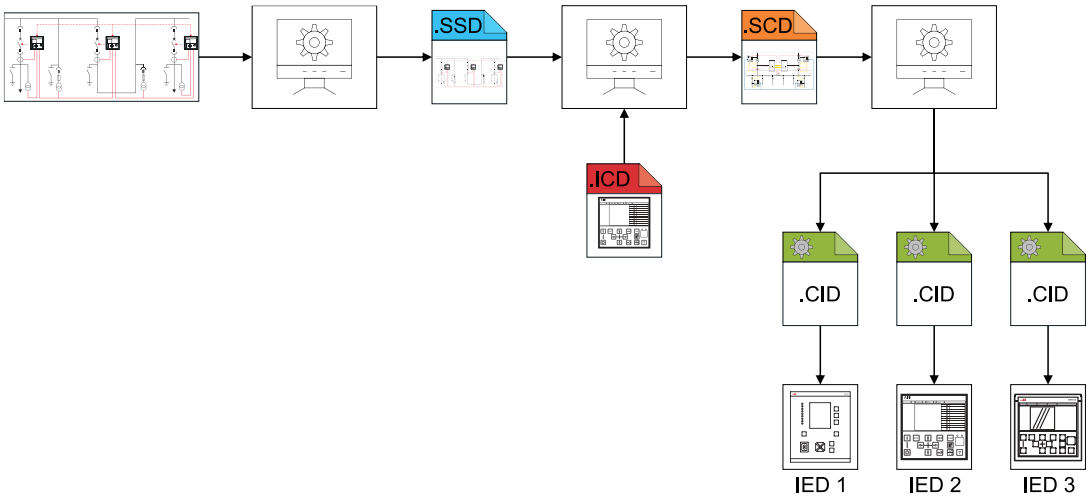
2.2.1 Linguagem e arquivos de configuração padronizados

A IEC 61850-6 [12] define uma linguagem de configuração da subestação (SCL – *Substation Configuration Language*), uma extensão de XML capaz de descrever o diagrama unifilar do sistema (através de um arquivo chamado SSD – *System Specification Description*), e as capacidades dos IEDs (arquivo ICD – *IED Capability Description*). Os fabricantes fornecem o arquivo ICD, enquanto o arquivo SSD é gerado a partir de ferramentas de softwares dedicados.

Como ilustrado na Figura 8, esses dois arquivos são integrados em um configurador do sistema para criar mais um arquivo chamado SCD. Deste, são extraídos CIDs (*Configured IED Description*) utilizados para configurar os IEDs individualmente.

Apesar de exigir experiência, essa praticidade de configuração é um dos diferenciais da IEC 61850 em relação às demais arquiteturas de rede.

Figura 8 – Geração de arquivos de extensão CID



Fonte: Autor, 2024.

Os quatro tipos de arquivos são sumarizados como segue:

- SSD (System Specification Description): descreve as funções da instalação elétrica;
- SCD (Substation Configuration Description): possui a arquitetura da rede de comunicação junto com a definição da toda a subestação;
- ICD (IED Capability Description): lista os dados suportados pelo IED;
- CID (Configured IED Description): especifica as configurações do IED.

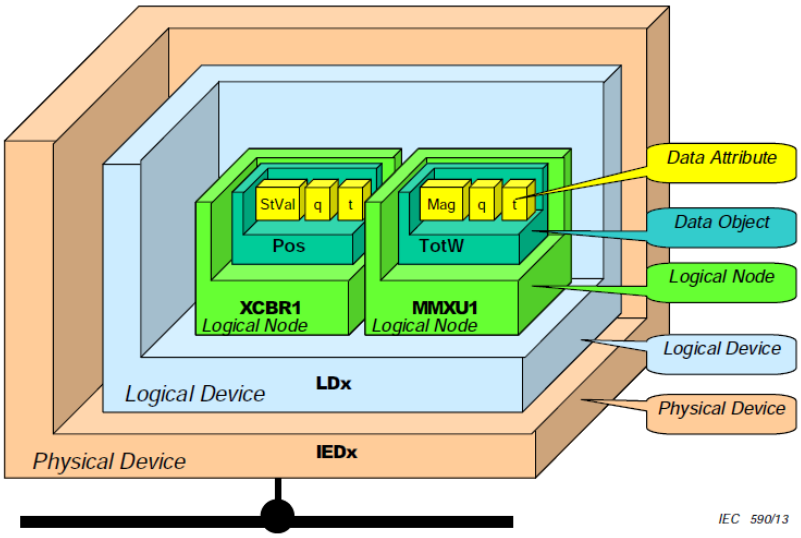
2.2.2 ACSI (Interface de Serviços Abstratos de Comunicação)

Entre os aspectos da IEC 61850, tem-se a interface de serviços de comunicação abstratos (ACSI), que especifica duas estratégias para padronizar os serviços e o acesso às informações de um SAS e então permitir a comunicação entre dispositivos de diferentes fabricantes: modelagem de dados (Seção 2.2.3) e de serviços (Seção 2.2.4).

2.2.3 Modelagem de dados na norma IEC 61850

A IEC 61850-7-1 [13] modela as principais funções dos dispositivos em uma subestação utilizando o paradigma de orientação a objeto, com uma hierarquia de quatro níveis na ordem dispositivo lógico (LD – Logical Device), nó lógico (LN – Logical Node), objeto de dados (DO – Data Object) e atributo (DA – Data Attribute), toda esta estrutura contida em um dispositivo físico (PD – Physical Device), como ilustrado na Figura 9.

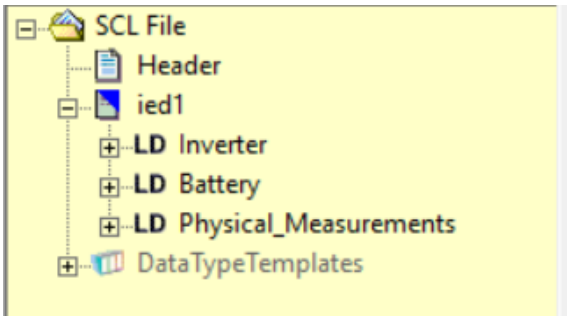
Figura 9 – Modelagem de dados na IEC 61850



Fonte: Retirado da IEC 61850-1 [7].

Cada dispositivo físico deve possuir pelo menos um LD, representando um grupo de utilidades que estão interrelacionadas. Não existe qualquer padrão que determine o número ou a forma que os LD estão arranjados dentro dos dispositivos. A única restrição é a existência de um único dispositivo físico para cada LD. Por exemplo, a biblioteca libIEC61850 fornece um modelo em um arquivo SCL (discutido na Seção 2.2.1) nomeado `complexModel.icd` que ilustra um inversor como um dispositivo físico dividido entre três LDs, como na Figura 10.

Figura 10 – Estrutura base de um arquivo SCL, com `complexModel.icd` como exemplo

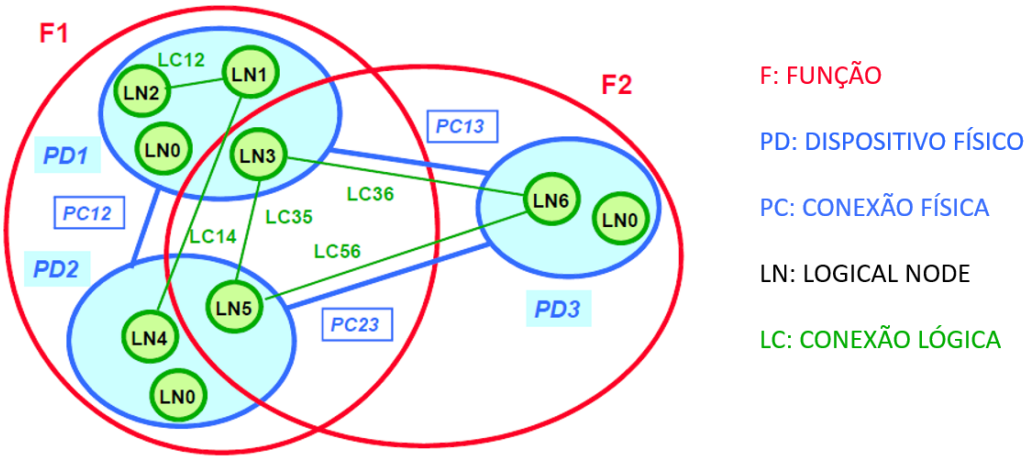


Fonte: Autor, 2024.

A segunda estrutura de dados da hierarquia são os LNs. Cada nó lógico representa uma funcionalidade dentro de um IED, por isso é frequente a existência de apenas um LD em um dispositivo físico. Não existem restrições de nomenclaturas dos

LNs, apesar de existirem convenções que associam o LN a um tipo de funcionalidade, por exemplo: funções de proteção em disjuntores ou medição em um TC ou em um TP, entre outros. Um dispositivo pode conter vários nós lógicos, que são alocados a um ou mais dispositivos do SAS. Esses conceitos estão ilustrados na Figura 11.

Figura 11 – Decomposição funcional em um SAS



- Os nós lógicos são ligados por conexões lógicas (LC) e os dispositivos por conexões físicas (PC);
- Qualquer nó lógico faz parte de um dispositivo físico;
- Qualquer conexão lógica é parte de uma conexão física;

Fonte: Adaptado da IEC 61850-5 [6]

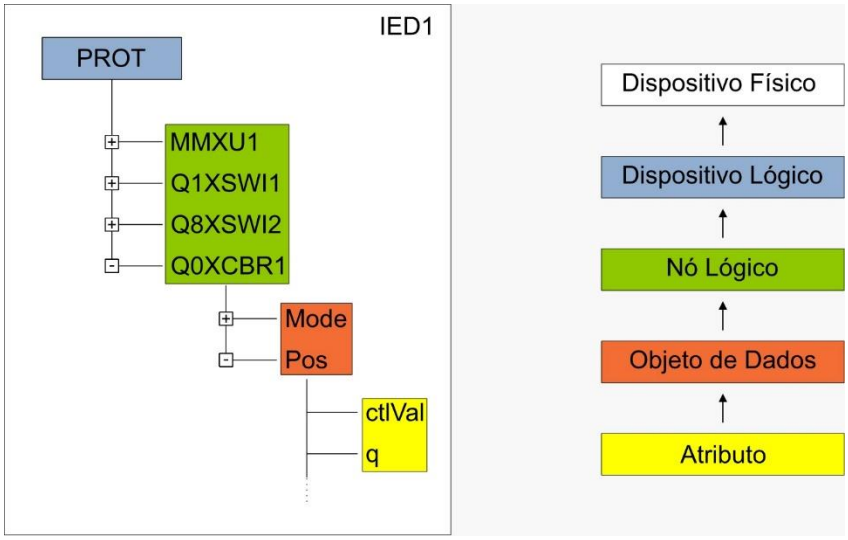
Outro ponto relevante é a existência de um LN principal chamado de Logical Node zero, que em termos de sintaxe XML é representado pela tag `<LN0 ... >` ao invés de `<LN ... >`, como os demais. Abrindo o mesmo modelo `complexModel.icd` em um editor de texto, verifica-se, na Figura 12, que o LD Physical_Measurements possui dois LNs: LLN0 e LPHD1 na linha 80.

Figura 12 – Arquivo `complexModel.icd` aberto em um editor de texto

Fonte: Autor, 2024.

Data Objects, por sua vez, são estruturas de dados que carregam variáveis que representam alguma informação específica em um nó lógico, seja algum estado binário ou uma medição. Este é o tipo de entidade que pode ser vista como uma instância de uma classe sob o ponto de vista de orientação a objeto. Os Data Atributes nada mais são do que as variáveis ou estruturas padronizadas que compõe um DO. O exemplo da Figura 13 traz como exemplo de DOs *Mode* e *Pos,* além de *ctlVal* e *q* como exemplos de atributo de forma organizada.

Figura 13 – Hierarquia das estruturas de dados conforme a IEC 61850



Fonte: Autor, 2024.

Estes dois tipos de entidades de nível mais baixo na hierarquia podem ser referenciados, e então, as referências são agrupadas ordenadamente para formar uma coleção chamada Dataset. Os membros de um Dataset podem ser objetos ou atributos e são agrupados para unificar as mensagens de requisição/resposta. Existem dois tipos de Datasets, os persistentes e os não persistentes. O primeiro tipo compreende àqueles que são previamente configurados no arquivo SCL e o segundo tipo é criado pelo serviço *CreateDataSet* e só é visível para o cliente que o cria.

A parte 7-2 [14] da norma define cinco serviços ACSI para possibilitar a interação entre os dispositivos: *GetDataSetValues* e *SetDataSetValaues* para ler e escrever valores de todos os membros; *CreateDataSet* e *DeleteDataSet* para criar e apagar datasets; *GetDataSetDirectory* para retornar ao cliente uma lista dos Datasets existentes no servidor.

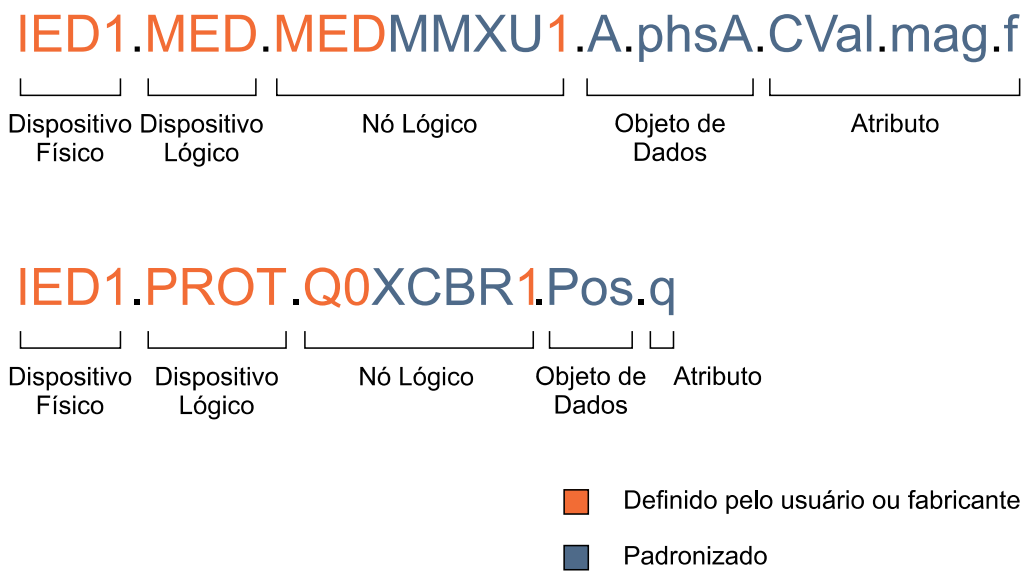
Na IEC 61850-7-2 [14] ainda, é delineado um mecanismo de relatório utilizando transmissão de dados não solicitada para minimizar a carga na rede de comunicação. Este mecanismo é constituído pelo uso dos chamados *Report Control Blocks* (RCB), onde cada RCB se relaciona unicamente com um Dataset e um cliente e, nele se define alguns atributos de interesse para intermédio da transmissão dos dados, como as condições de disparo de eventos, período de integridade que define o envio periódico de relatórios, *buffer time* (define quanto tempo o relatório deve esperar por um novo evento antes de ser enviado), dentre outros.

Existem dois tipos de RCB a depender do seu comportamento na ocorrência de um evento que depende de como é feita configuração para enviar o relatório. Um dos tipos de RCB é o *Unbuffered Report Control Block* (URCB), cujo comportamento após evento é imediatamente mandar o relatório sem se preocupar com a integridade dos dados ou se a informação será perdida ocasionada por uma desconexão repentina. O outro tipo de RCB é o *Buffered Report Control Block* (BRCB), que quando da ocorrência de um evento, ele salva os dados em um buffer e aguarda um tempo antes de enviar o relatório. Após enviado, se não houver resposta do cliente, se assume que a informação foi perdida e então se recupera os dados do Buffer para enviar novamente o relatório.

2.2.3.1 Semântica padronizada

Do modelo de dados e da semântica padronizados provém uma das principais vantagens do Padrão IEC 61850: autodescrição dos dados, ou seja, é possível identificar não só a categoria do dado, como também a sua fonte. Um exemplo de informação normatizada é apresentado na Figura 14.

Figura 14 – Normatização de nomenclatura de entidades estabelecidas no modelo de dados da IEC 61850.



Fonte: Autor, 2024.

É importante pontuar que o nome do dispositivo lógico é definido pelo usuário, enquanto os nomes dos nós lógicos, objetos de dados e seus atributos são padronizados pela IEC 61850. A primeira letra do nó lógico designa o grupo funcional que o contém. Assim, “P” refere-se à proteção, “M” refere-se à medição, “C” refere-se à controle. As demais definem o nó lógico específico.

2.2.4 Modelagem de serviços

A ACSI define diversos serviços de comunicação abstratos, que separam a tecnologia de comunicação do modelo de objetos. Dessa forma, é possível incorporar evoluções da tecnologia de comunicação à norma IEC 61850, característica referida como “a prova de futuro”.

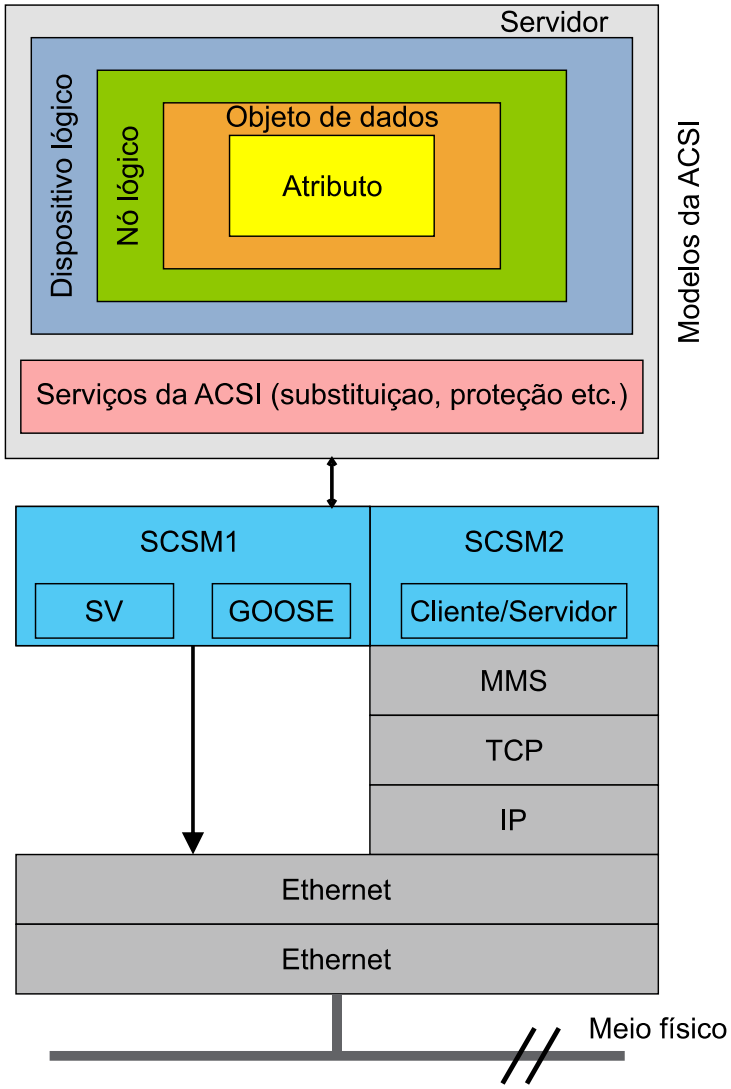
Os serviços especificam as operações que podem ser executadas durante a troca de mensagens entre os dispositivos, independente do protocolo de comunicação adjacente. São exemplos de serviços padronizados: agrupamento de dados e atributos, substituição de valores ativos nos processos em tempo de execução, alteração de conjunto dinâmico de valores de configuração, condições de geração e relatórios, configurações para transmissão confiável de valores ultrarrápidos, transferência de valores analógicos ultrarrápidos em tempo real, definição de serviços de controle, serviços de sincronização de dispositivos, troca de arquivos em grandes blocos de dados e rastreamento de informações.

Adicionalmente, a IEC 61850 padroniza procedimentos chamados mapeamento para serviço de comunicação específico (Specific Communication Service Mapping, SCSM). Cada SCSM contém perfis de comunicação e protocolos de aplicação, que são usados para mapear os serviços da ACSI em uma pilha de protocolos particular da rede de comunicação.

É importante destacar que apenas os dispositivos com o mesmo SCSM implementado são interoperáveis.

Na figura abaixo, tem-se um esquema simplificado representando os SCSMs integrando os serviços da ASCI à rede de comunicação.

Figura 15 – Integração do modelo de dados com a rede de comunicação



Fonte: Autor, 2024.

2.2.5 Níveis hierárquicos e barramentos em um SAS

A norma IEC 61850 define três níveis hierárquicos em um SAS: nível de estação, nível de bay e nível de processo.

O nível de estação é o nível superior. Contém as unidades do sistema SCADA (IHM, computadores), e gateways para permitir a comunicação com os dispositivos de controle para fins de supervisão.

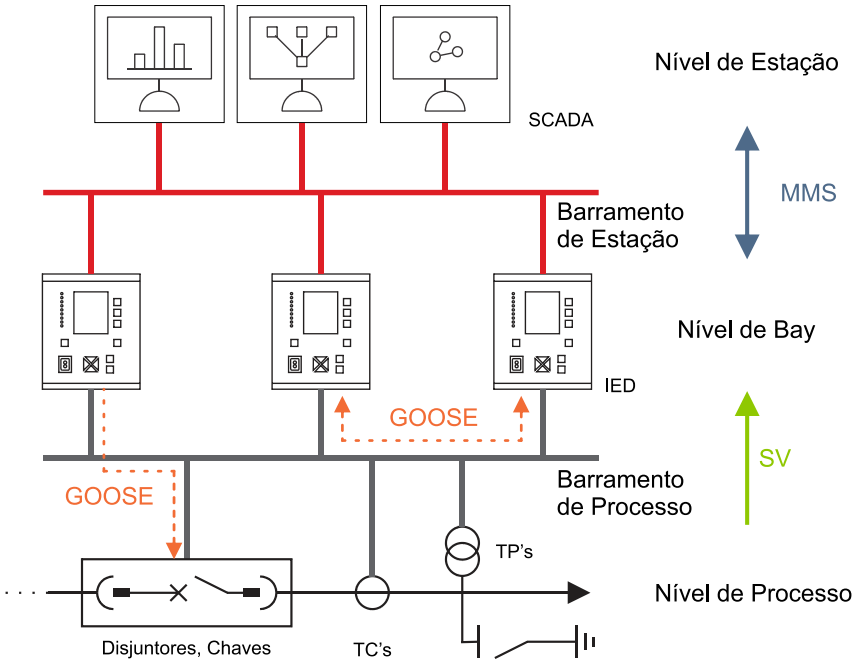
O nível de bay é o nível intermediário. Contém os IEDs, equipamentos de controle da subestação. O nível de processo é o nível inferior.

Contém os equipamentos de campo da subestação (Disjuntores, seccionadoras, transformadores etc.) e as Merging Units (Mus).

As Mus são dispositivos responsáveis por coletar e digitalizar os valores amostrados (Sampled Values) dos transformadores de instrumentos não convencionais, tais como TPs e TCs ópticos, encapsular os valores digitais em um frame Ethernet definido pela norma e então transmitir via rede Ethernet.

A IEC 61850 define dois barramentos em um SAS, o de estação e o de processo, mostrados na Figura 16. O barramento de estação interliga os dispositivos do nível de estação com os do nível de bay, permitindo a comunicação vertical entre eles. O barramento de processo interliga os dispositivos de campo com os IEDs, permitindo a comunicação entre eles. Este barramento é também utilizado para a comunicação horizontal entre os IEDs.

Figura 16 – Barramentos de estação e de processo definidos na IEC 61850 e a interação dos dispositivos



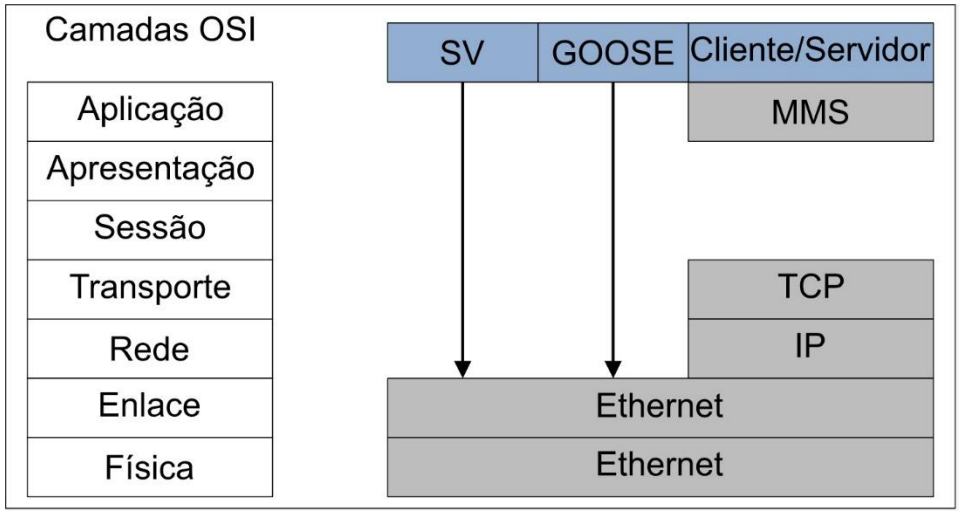
Fonte Autor, 2024.

2.2.6 Protocolos de comunicação

Como mencionado anteriormente, a IEC 61850 utiliza protocolos que permitem a comunicação entre equipamentos heterogêneos de um SAS. Os principais são eles: MMS, GOOSE e SV. Suas aplicações são baseadas no padrão OSI (Figura 17), descritas a seguir e localizadas em um SAS conforme a Figura 18.

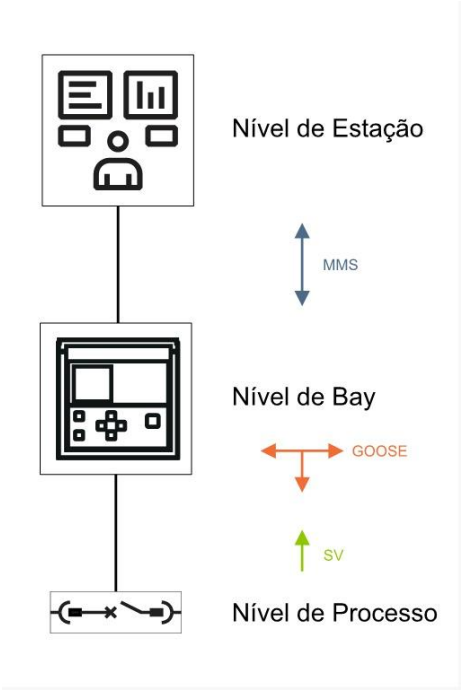
- MMS (Manufacturing Message Specification): padronizado na ISO 9506 e especificado na IEC 61850-8-1, permite a comunicação vertical entre os equipamentos do nível de estação e do nível de bay. As mensagens protocoladas são basicamente, de supervisão, portanto com prioridade menor do que as mensagens de proteção e tempos de transmissão na ordem de segundos. O MMS é um protocolo de camada de aplicação, localizado acima do conjunto de protocolos TCP/IP. O modelo de comunicação é o cliente/servidor, com transmissão unicast.
- GOOSE (Generic Object Oriented System Events): padronizado na IEC 61850-8-1, permite a comunicação horizontal e vertical entre IEDs e entre equipamentos do nível de processo. As mensagens protocoladas são, basicamente, de proteção, portanto com prescrições de prioridade e tempos de transmissão na ordem de milissegundos. Assim, as mensagens GOOSE são mapeados diretamente na camada de enlace de dados. O modelo de comunicação é o publicador/assinante, com transmissão multicast.
- SV (Sampled Values): padronizado na IEC 61850-9-2 permite a comunicação vertical entre os equipamentos do nível de processo e as IEDs. As mensagens protocoladas são, basicamente, os valores amostrados dos sinais analógicos (corrente, tensão...), portanto, com prescrições de prioridade e tempo de transmissão de milissegundos. Assim, as mensagens SV são mapeadas diretamente no padrão Ethernet. O modelo de comunicação é o publicador/assinante, também com transmissão multicast e a norma também prevê transmissão unicast, porém raramente utilizado.

Figura 17 – Protocolos de comunicação da IEC 61850 e o padrão OSI



Fonte: Autor, 2024

Figura 18 – Protocolos de comunicação definidos em um SAS



Fonte: Autor, 2024

De particular interesse, além desses três há um quarto protocolo, que diferente dos outros, não é de comunicação, mas sim de sincronismo de tempo entre dispositivos em um SAS: o SNTP (Simple Network Time Protocol). Seu propósito se limita a registrar eventos com precisão na ordem de 1 milissegundo, sendo suficiente para aplicações com baixos requisitos de temporização, como em mensagens MMS. Porém, se tratando de sincronia entre Mus e PMUs, sua aplicação é imprópria, sendo

necessário utilizar algum protocolo com maior precisão temporal, como o PTP (Precision Time Protocol), conforme especificado na IEC 61850-5 e na NBR 16932 [15].

2.2.7 Requisitos de temporização na comunicação

O padrão IEC 61850 define requisitos de desempenho dependendo do tipo de mensagem. As mensagens transmitidas dentro de uma subestação inteligente incluem os protocolos citados na Seção 2.2.6, e podem ser classificadas em sete tipos, listados na primeira coluna da Tabela 2.

Tabela 2 – Requisitos de temporização de transmissão para diferentes tipos de mensagens

Tipo de mensagem	**Mensagem**	**Protocolo**	**Atraso máximo**	**Aplicação**
1ª. Mensagens rápidas, Trip	GOOSE	Layer-2 Multicast	<3 ms	Proteção
1B. Mensagens rápidas, outros	GOOSE	Layer-2 Multicast	<20 ms	Proteção
2. Velocidade média	MMS	IP/TCP	<100 ms	Controle
3. Velocidade baixa	MMS	IP/TCP	<500 ms	Controle
4. Dados brutos	SV	Layer-2 Multicast	<3 ms	Barramento de Processo
5. Transferência de arquivos	MMS	IP/TCP/FTP	>1000 ms	Gerenciamento
6. Sincronismo de tempo	Time Sync	SNTP (IP) ou PTP (layer 2)		Uso geral; Fasores, SV
7. Comandos	MMS	IP		Controle

Fonte: Adaptado de [8] e da NBR 16932 [15].

As mensagens do tipo 1 são voltadas para interação direta com dispositivos de proteção e são divididas em duas subclasses: tipo 1ª e tipo 1B. A primeira é aplicada em mensagens de “trip”, enquanto qualquer outra mensagem, como abertura e alteração de estados de dispositivos é enquadrada no tipo 1B, todas usando o protocolo GOOSE. O protocolo SV também tem alto requisito de tempo de transmissão, uma vez que sua aplicação é voltada para a transmissão periódica de mensagens de transformadores de instrumentação.

O protocolo MMS, por sua vez, atende a três tipos diferentes de mensagens: o tipo 2 se aplica a mensagens automáticas de transmissão em caráter periódico ou orientado a eventos; os tipos 3 e 7 são de requisitos de temporização associados a tempo de resposta humana, voltados para aplicações apresentação de dados a uma estação de trabalho (em IHMs por exemplo), como registro de eventos, configuração de set-points e funcionalidades lentas de autocontrole [16]

2.3 Soluções disponíveis

No trabalho de Marissa Wong [17], algumas tentativas foram realizadas para embarcar a biblioteca libIEC1850 em um sistema computacional de baixo custo. Em particular para o ESP32, a autora tentou três metodologias, a primeira foi exportar bibliotecas estáticas para serem consumidas em um projeto ESP-IDF, metodologia a que será descrita em 3.2. A segunda tentativa foi não utilizar o framework ESP-IDF e embarcar o Sistema operacional NuttX – de grande aceitação na comunidade desenvolvedora por ser compatível com uma gama de microcontroladores, baseado no kernel Linux, que a princípio teria compatibilidade direta com a biblioteca libIEC61850, o que [17] destaca não ser possível.

Sua terceira tentativa foi baixar o repositório da libIEC61850 pelo OrangePi OS configurado no Orange Pi 3 e executar a montagem do código de forma padrão. Esta foi uma solução bem-sucedida e, além de simples, de custo acessível. Seu trabalho serviu de referência ao reportar dificuldades encontradas, e que, portanto, poderiam se repetir durante a execução deste trabalho, além de mostrar ferramentas apropriadas para esta aplicação.

O trabalho de Mayamiko Hara [18] se encaixa no contexto de adoção continuada de pequenas gerações embarcadas (SSEGs), que seria similar com a Minigeração distribuída no Brasil, porém com limitação de 1MW. Este contexto gera a demanda de uma infraestrutura de controle e monitoramento confiável. Então o seu objetivo é fornecer uma solução de baixo custo para os dispositivos de medição nas instalações seguindo os padrões da norma IEC 61850. Como resultado, o autor apresenta a modelagem e implementação do dispositivo usando a biblioteca

libiec61850, embarcando na Raspberry Pi, incluindo um frontend de medição analógica com o CI ADE9000 de aplicação fundamental para qualímetros.

Sungho Hwang [8] trouxe, sem mostrar detalhes de implementação, uma investigação profunda sobre o uso do protocolo MMS em conformidade com a norma IEC61850 usando o Raspberry Pi. Porém, ao invés de apenas um dispositivo, usando um conjunto de 5 unidades para representar minimamente uma subestação. Sua metodologia consistiu em primeiramente usar o protocolo PTP conforme a IEEE1588 para sincronizar os temporizadores de todos os componentes em rede e então capturar e registrar os pacotes de dados de rede usando o software Wireshark.

2.4 A biblioteca libIEC61850

A biblioteca libIEC61850 é uma implementação cliente/servidor dos protocolos MMS, GOOSE e SV seguindo os padrões estabelecidos na norma IEC 61850 na linguagem C no padrão C99, para fornecer máxima portabilidade [19]. O projeto é cedido pela MZ Automation, sob a licença GPLv3, com suporte para Windows, Linux, MacOS e Arm-linux.

A utilização desta biblioteca simplifica a implementação de programas voltados para comunicação entre dispositivos que seguem a norma IEC 61850, uma vez que ela já dispõe as estruturas e funções necessárias para emular os dispositivos de uma subestação, além de fornecer o alto desempenho e exploração de recursos que a linguagem C oferece. Alguns exemplos de recursos fornecidos pela biblioteca são: log service, MMS files services, suporte a TLS e gerenciamento de blocos de controle GOOSE e SV. Dentre os variados recursos, para este trabalho foi necessário usar a apenas o servidor MMS.

2.5 Raspberry Pi

O Raspberry Pi é uma família de microcomputadores *single-board computer* de baixo custo voltado para aplicações embarcadas e IoT. Possui todos os periféricos que seriam indispensáveis para um computador pessoal: entradas USB, Ethernet, saída P2 e HDMI, Bluetooth e Wi-Fi [20]. Sua popularização na comunidade DIY se deve a

um atraente equilíbrio entre desempenho, custo, compatibilidade e capacidade de expansão, oferecendo facilidades pela grande disponibilidade de tutoriais e documentação, além de sua disponibilidade no mercado.

Neste trabalho, será utilizado o modelo Pi 3 Model B+, mostrado na Figura 19, servindo para emular o cliente no nível de estação – centros de controle para fins de supervisão.

Figura 19 – RaspberryPi 3 Model B+

Fonte: Adaptado de [21].

2.6 O microcontrolador ESP32 e o framework ESP-IDF

ESP-IDF (Espressif IOT Development Framework) é o SDK (conjunto de ferramentas e recursos) disponibilizado pela Espressif para programar os microcontroladores da família ESP32. O SDK é inteiramente disponível no Github, que é um concentrador de várias outras ferramentas/bibliotecas. Dentre uma variedade de SDKs disponíveis na internet para se programar o ESP32, como pela IDE do Arduino, MicroPython, Espruino (JavaScript), Lua-RTOS, Rust, NuttX ..., o esp-idf possui as seguintes características de interesse:

- ESP-IDF é o framework nativo: é uma grande vantagem, pois sempre que um SoC (sistema-em-um-chip) novo da mesma família é lançado, o mesmo código terá compatibilidade assegurada.

- Linguagem: O framework esp-idf tem suporte para C/C++, condição necessária para suportar o código da libIEC61850, que é inteiramente escrita em C como descrito em [22]. Além disso, é importante observar que o padrão C99 não suporta funções aninhadas, logo não existe uma preocupação de padronização da linguagem, uma vez que o framework esp-idf admite o padrão C17 com exceção para o recurso de funções aninhadas.
- Recursos integrados: há muitos componentes disponíveis, incluindo pilhas integradas de rede que adotam padrões POSIX – Portable Operating System Interface, que serve para manutenção de compatibilidade entre sistemas operacionais. Este ponto tem seu destaque baseado no fato de que, combinando com o esp-idf, a biblioteca libIEC61850 tem suas funções e estruturas aderentes ao padrão POSIX, facilitando a sua implementação evitando que o usuário precise criar códigos do zero;
- Facilidades de uso: Suporte por linha de comando e extensão do VSCode e ao Eclipse IDE, bem como documentação organizada e quantidade significativa de problemas encontrados reportados no repositório e em fóruns e suas respectivas soluções;
- Recursos de depuração: o framework possui suporte a GDB, sendo essencial para contornar erros inesperados;

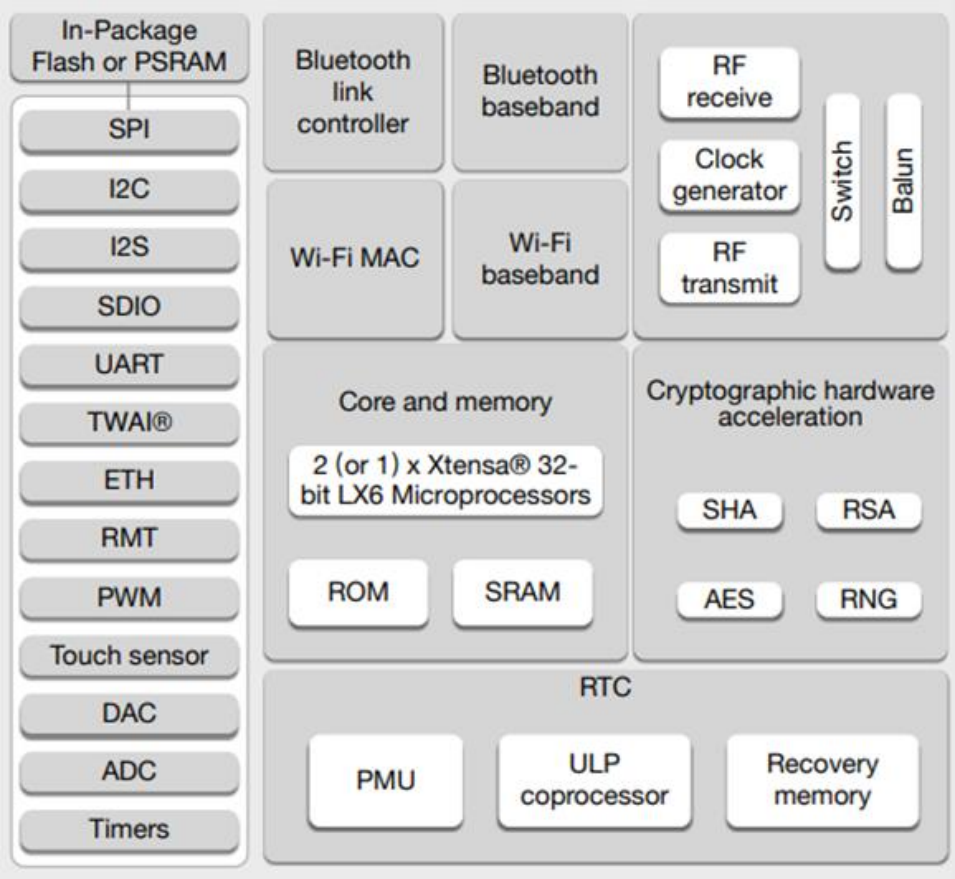
2.6.1 Visão geral do hardware ESP32

ESP32 é uma família de microcontroladores que se destaca por integrar Bluetooth e Wi-Fi, projetada para maximizar o desempenho de potência e rádio frequência com robustez, versatilidade e confiabilidade em diferentes cenários e aplicações. Por ser de baixo consumo e possuir o recurso de Deep-sleep (10 uA), o ESP32 é uma escolha ideal para IoT em aplicações de: automação residencial e industrial, dispositivos de áudio, agregadores de sensores IoT, data loggers genéricos, streaming de vídeo, reconhecimento de imagem, entre outros.

Quanto aos recursos de hardware, além de integrar Wi-Fi (802.11b/g/n e 802.11n de 2.4 GHz até 150 Mbps), e Bluetooth (v4.2), o microcontrolador emprega um microprocessador Xtensa de 32 bits LX6 de até 2 cores com frequência de clock de

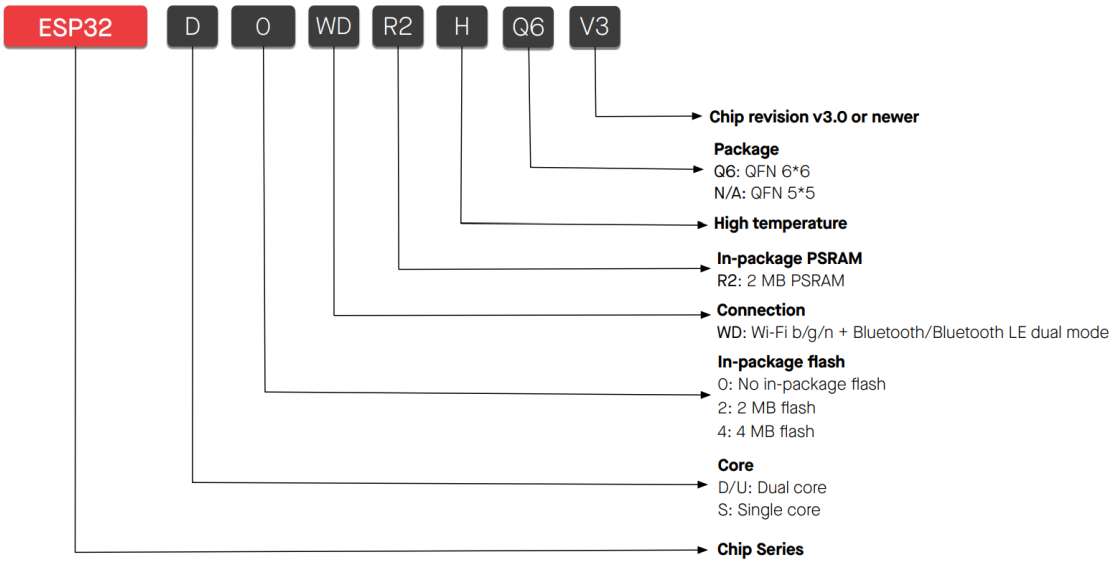
até 240 Mhz e 520 kB de SRAM. O diagrama de blocos funcional da Figura 20 apresenta os recursos disponíveis no ESP32. Vale ressaltar que, por ser uma família de microcontroladores, sua nomenclatura se estende conforme a Figura 21 a depender da especificação de alguns periféricos.

Figura 20 – Diagrama de blocos funcional do ESP32



Fonte: ESP32 Series Diagram [23]

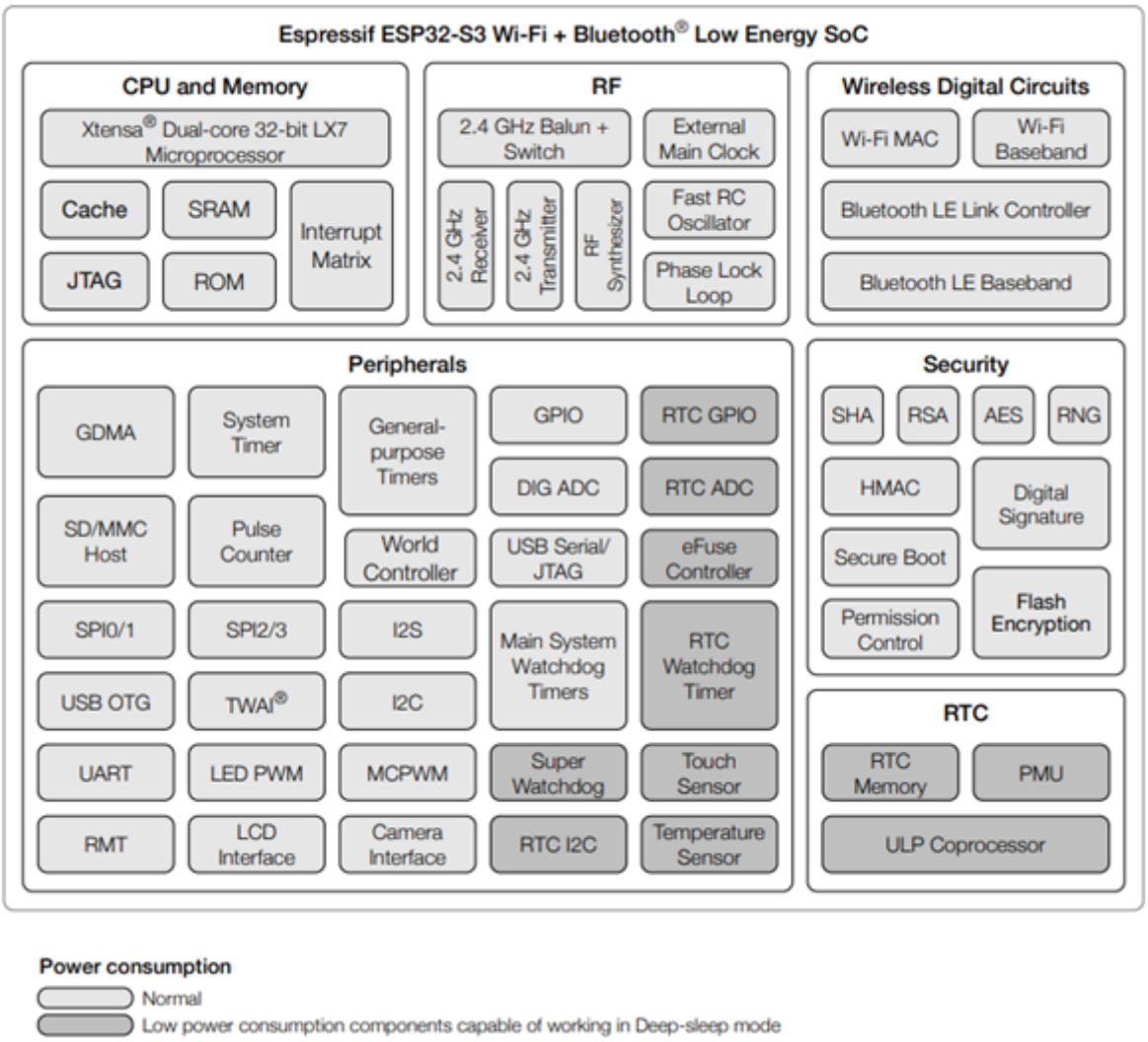
Figura 21 – Nomenclatura adotada dos microcontroladores para a família ESP32



Fonte: ESP32 Series Datasheet [23]

Conforme detalhado na Seção 3.2.5, houve a necessidade de migrar do ESP32 para o ESP32-S3 para dar prosseguimento ao desenvolvimento do projeto. Suas principais diferenças, se comparado ao ESP32 são: *upgrade* de Bluetooth para versão 5.0, pequeno incremento na memória SRAM interna em 8 kB, 9 GPIOs a mais e emprego do processador de 32 bit Xtensa LX7 single/dual-core. Seus recursos são esboçados no diagrama da Figura 22.

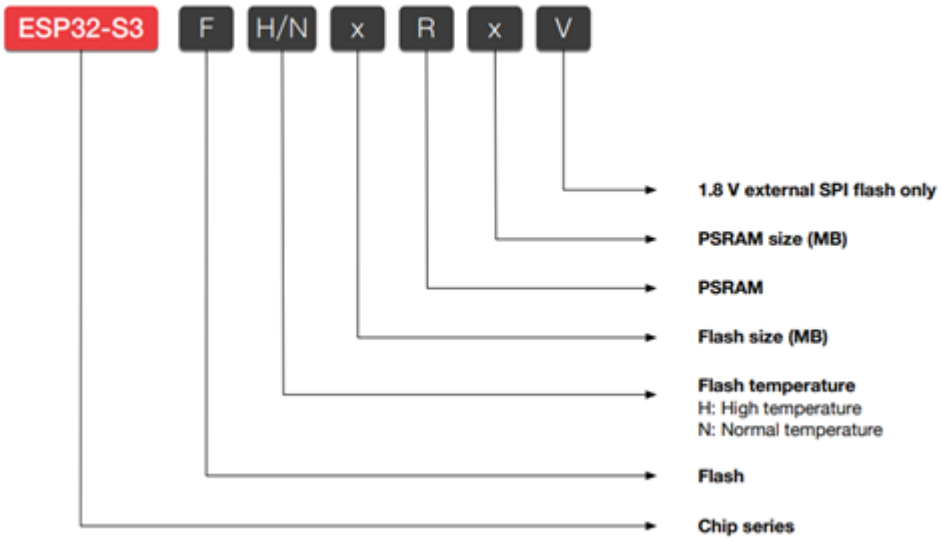
Figura 22 – Diagrama de blocos funcional do ESP32-S3



Fonte: ESP32-S3 Datasheet [24]

Assim como no caso do ESP32, existe uma nomenclatura que diferencia os microcontroladores quanto as suas especificações, como na Figura 23.

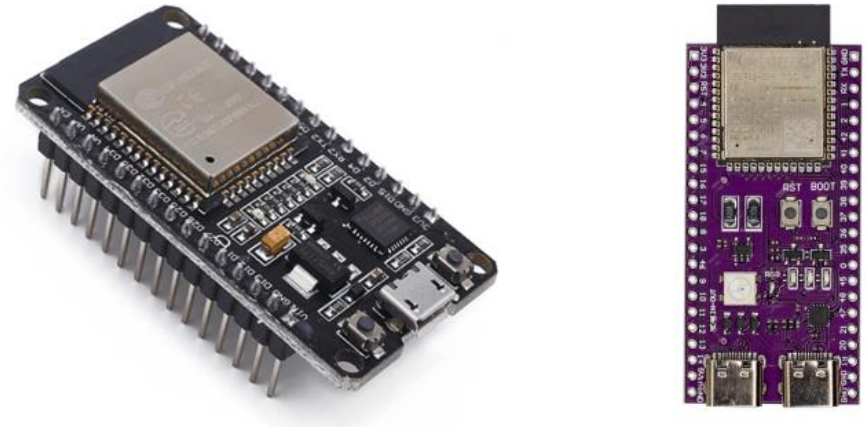
Figura 23 – Nomenclatura adotada dos microcontroladores para a família ESP32-S3



Fonte: ESP32-S3 Datasheet [23]

Os modelos utilizados neste trabalho foram o ESP32-D0WD-V3 e ESP32-S3 N16R8. Como apresentados na Figura 24 nos lados esquerdo e direito respectivamente.

Figura 24 – ESP-WROOM-32 DevKit no lado esquerdo e DevKitC-1 ESP32-S3 N16R8 no lado direito



Fonte: retirado de [25] e [26]

2.6.2 Estrutura básica de um projeto com o ESP-IDF

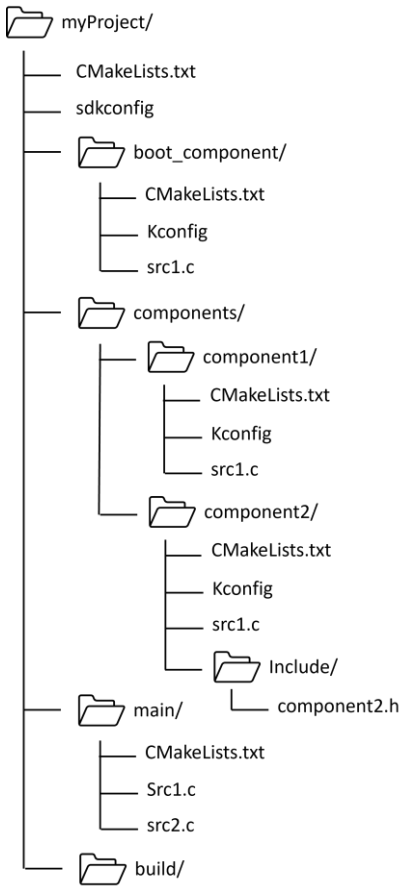
O ESP-IDF (Espressif IoT Development Framework) é um aglomerado de ferramentas disponibilizadas pela Espressif para se desenvolver aplicações voltadas para a família de microcontroladores ESP32. Este framework é desenvolvido de modo que os usuários não precisem conhecer cada uma das ferramentas. Ao invés disso, é mais produtivo que se conheça a estrutura básica de um projeto ESP-IDF, alguns comandos básicos do framework, bem como alguns conceitos e detalhes relacionados à montagem.

Os projetos ESP-IDF seguem uma estrutura orientada a componentes conforme a árvore de arquivos mostrada na Figura 25, onde cada componente é uma biblioteca que pode representar algum periférico de hardware ou também pode representar a abstração de algum recurso ou funcionalidade em software. Esta estrutura é composta por:

- Arquivos "`CMakeLists.txt`": são arquivos que servem de insumo para o Cmake criar os arquivos de geração de binários. O arquivo que se encontra na raiz do projeto se destaca em relação aos outros de mesmo nome porque é nele que se concentram todos os detalhes referentes a definições geras do projeto além da sua montagem. Os demais, tem a importância apenas de definir dependências obrigatórias a outros componentes e a estrutura de que o componente segue quanto aos seus *sources* e *headers*.
- Arquivo `sdkconfig`: é um arquivo de texto que armazena configurações gerais tanto do projeto como dos componentes. Por parte dos componentes, estas configurações são arquitetadas nos arquivos de *kconfig* de cada componente e são selecionadas com a ferramenta *menuconfig*;
- Cada componente possui seu próprio diretório com seu nome e, por padrão, ficam localizados na pasta "`componentes/`", entretanto podem ser colocados em qualquer diretório, desde que devidamente especificados no arquivo "`CMakeLists.txt`" da raiz do projeto.
- Diretório "`build/`": Este diretório armazena, por padrão, os binários gerados quando da montagem do projeto.

- `boot_component/` é um diretório opcional de uso pouco frequente que serve para sobrescrever o bootloader padrão referente a um dado *target*.

Figura 25 – Estrutura básica de um projeto ESP-IDF



Fonte: adaptado de [27]

2.6.2.1 Alternativas para montagem de um projeto ESP-IDF

A Espressif define a estrutura da Seção O microcontrolador ESP32 e o framework ESP-IDF2.6.2 por conveniência, pois torna os componentes compatíveis entre si em diferentes projetos e de fácil portabilidade. Entretanto, uma estrutura fixa como tal pode oferecer dificuldades em situações que o usuário precisa reaproveitar código de terceiros de origem não relacionada ao ESP32. Pensando nisso, a Espressif mantém possibilidades, e descreve na documentação [27], para facilitar a devida montagem:

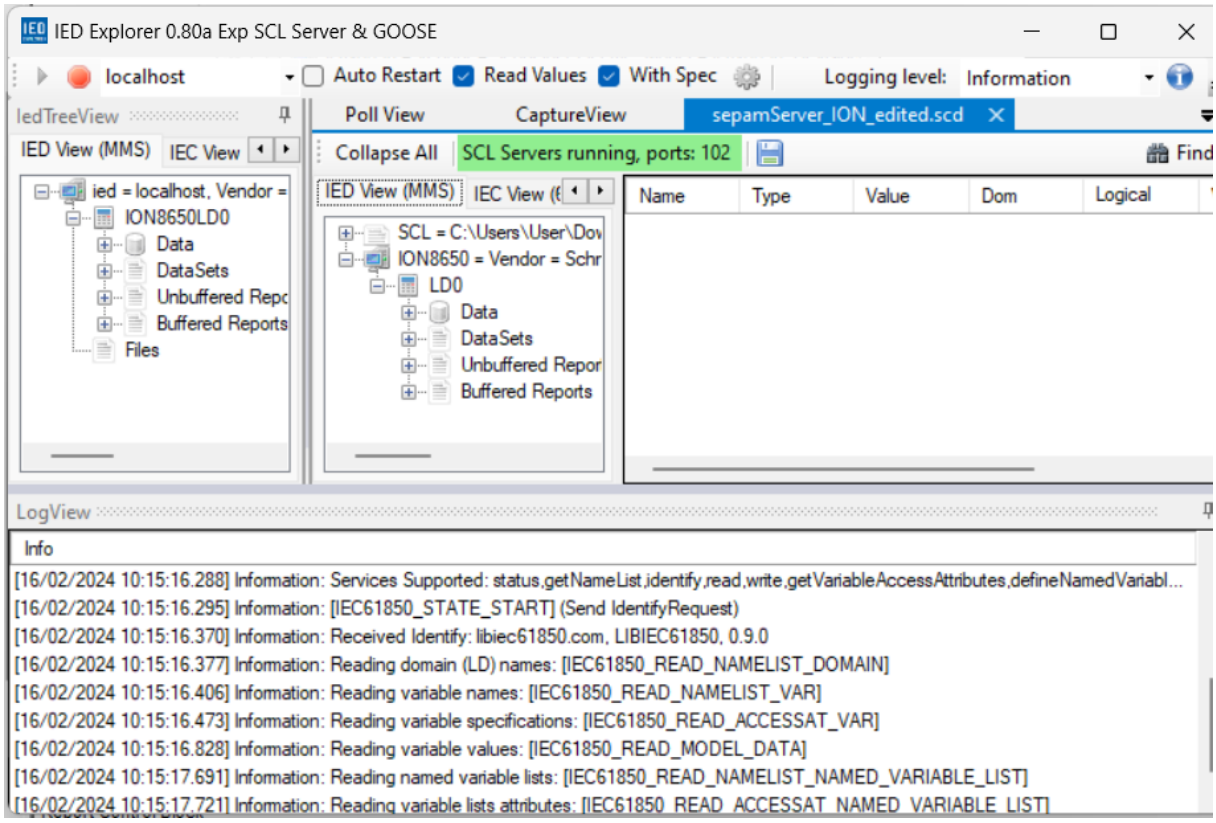
- Sobrepor completamente a montagem de um componente: estabelece uma forma de aproveitar uma biblioteca que não foi originalmente idealizada para a montagem pelo ESP-IDF e isto é feito a partir de um recurso do Cmake chamado *ExternalProject*. Cabe destacar que se deve atenção à possível necessidade de suprir o correto *toolchain.*
- Criação de componentes através do Cmake puro: substitui os comandos em *CMakeLists.txt* por comandos originais do Cmake;
- Aproveitar códigos de terceiros baseados em Cmake para serem usados como componentes;
- Usar binários no lugar de componentes: O ESP-IDF disponibiliza uma função *add_prebuilt_library* para facilmente importar e consumir bibliotecas estáticas (arquivos *.a*);
- Usar ESP-IDF em projetos personalizados: prepara um projeto para montagem com poucas alterações no *CmakeLists.txt,* consumindo os componentes IDF como bibliotecas. Vale destacar que esta alternativa foi implementada no desenvolvimento do projeto, conforme especificado no parágrafo imediatamente anterior à Seção 3.2.1.

Para mais detalhes, consultar [27].

2.7 IEDExplorer

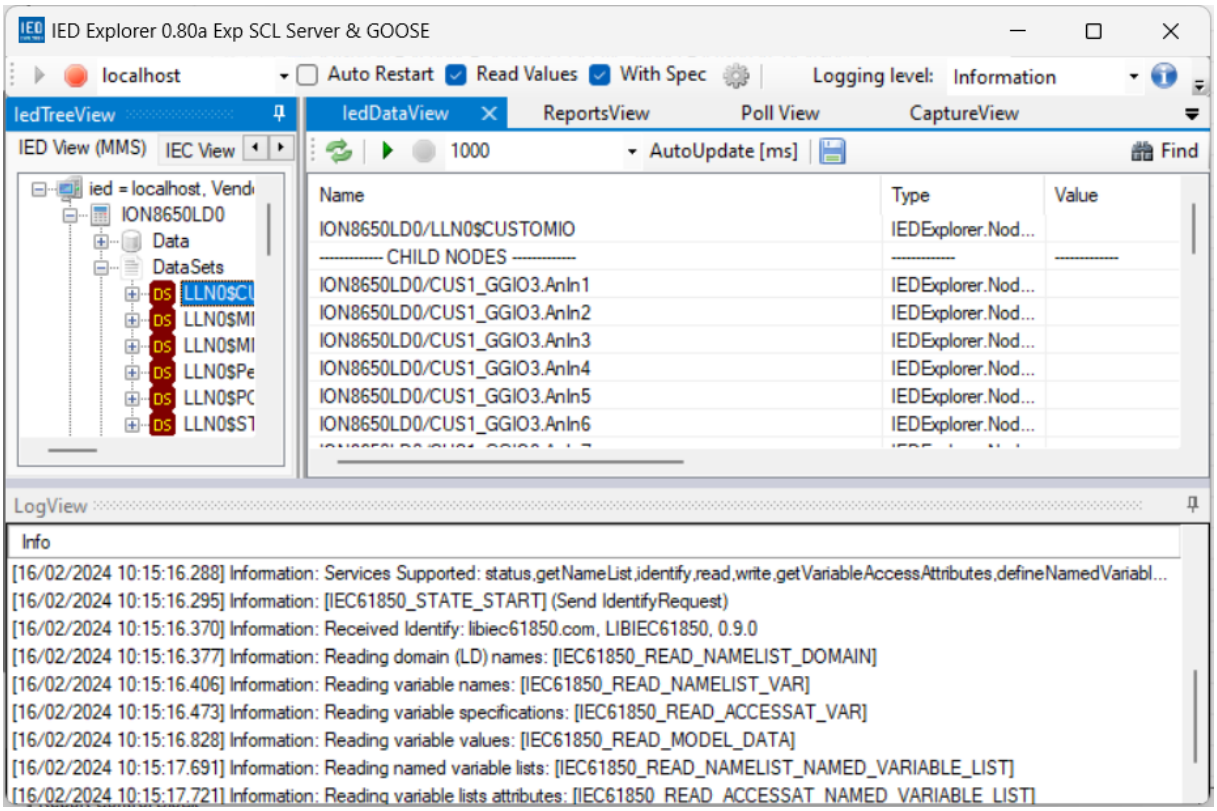
IEDExplorer é um projeto multiplataforma .NET open-source desenvolvido por Pavel Charvat. Foi projetado para auxiliar o aprendizado e o contato com os recursos de comunicação da norma IEC 61850 com foco majoritário no protocolo MMS. Apesar de não ser tão completo em termos de recursos, é um grande facilitador em um primeiro contato com a norma por ser uma alternativa gratuita e por fornecer interface simplista para interagir com dispositivos que seguem a IEC 61850. A Figura 26 mostra o recurso de emulação de servidor interno sendo acessado por uma instância de cliente no próprio localhost e a Figura 27 mostra a aba *IedDataView*, que possibilita acessar qualquer dado provido por um arquivo SCL, um Dataset, por exemplo.

Figura 26 – Servidor iniciado no IED Explorer a partir de um arquivo SCD



Fonte: Autor, 2024

Figura 27 – Acesso a dados do servidor criado no IED Explorer pela aba *IedDataView*



Fonte: Autor, 2024

3 DESENVOLVIMENTO DO TRABALHO

Para facilitar a compreensão do texto, o desenvolvimento do trabalho é divido em duas partes principais, a primeira trata do porte da biblioteca libiec61850 para o RaspberryPi, e a segunda parte é sobre a compilação cruzada da libIEC61850 para o ambiente de desenvolvimento do esp-idf. As palavras destacadas em itálico são representações incompletas, que facilitam a leitura de comandos ou termos técnicos, enquanto nomes de arquivos e trechos de código são representados com a fonte Courier New.

3.1 Implantação da biblioteca libiec61850 no RaspberryPI

A implantação do código no Raspberry Pi é de grande praticidade, no sentido de que o sistema operacional padrão é o Raspberry Pi OS, que é uma distribuição Linux baseada no Debian ao passo que a biblioteca tem suporte para sistemas Unix. Sendo assim, para fins de praticidade, a configuração de conectividade por SSH juntamente à definição de um IP estático por meio do arquivo `/etc/dhcpcd.conf`, foi ativada para que um computador pessoal possa acessar o RaspberryPi através de um emulador de terminal que suporte SSH, como o PuTTY por exemplo.

Feitas estas configurações iniciais, após realizar a conexão, basta selecionar um diretório de preferência, clonar e montar o código como segue:

```
$git clone https://github.com/mz-automation/libiec61850.git

$cd libiec61850

$make examples
```

Ao rodar o comando ***make*** com o *target examples*, o arquivo *makefile* standalone criará, se não houver, um diretório chamado `build/` e organiza os binários em seu subdiretório `build/examples/`. Esta é a maneira mais direta e simples de se realizar a montagem, conforme é descrita na documentação do código, entretanto nada o impede de aproveitar os `Cmakefiles.txt` disponíveis através do Cmake para fazer a devida montagem dos binários, o que seria o procedimento mais generalista, e será

abordado com mais detalhes quando da compilação cruzada do código para a plataforma esp-idf.

Após o último comando *shell* explicitamente citado, o usuário pode rodar qualquer exemplo de preferência. No caso do exemplo *server_example_basic_io*, é procedido como segue:

```
$cd examples/server_examples_basic_io
$sudo ./server_examples_basic_io
```

O último comando cria e roda um servidor na rede a qual o dispositivo está conectado e, se for usado a porta padrão 102, é necessário permissões de *superusuário*, por isso o comando `sudo`. Caso não seja de interesse do usuário rodar o comando com permissões `root`, então basta especificar uma porta a qual o servidor será vinculado que não seja de uso restrito ao usuário `root`, ou seja, uma porta de número não inferior a 1024. Isto é feito passando o número da porta como argumento do comando através da entrada padrão `stdin`. Por exemplo, se for desejado especificar a porta 3333, o último comando explicitamente citado poderia ser substituído por:

```
$./server_examples_basic_io 3333
```

Uma vez que o código esteja rodando em um terminal, outra instância de um terminal pode rodar o código de um servidor no mesmo dispositivo ou em outro dispositivo conectado à mesma rede local.

Este procedimento pode ser adotado para montar e rodar um exemplo de cliente no Raspberry Pi, com as únicas diferenças de que a pasta em que o código objeto do cliente se localiza em um diretório distinto, e que na entrada padrão `stdin`, o servidor não recebe apenas a porta a qual o respectivo socket é associado, mas também o IP do servidor. Quanto à posição das duas entidades, o IP é o primeiro argumento e o número da porta do socket é o segundo argumento.

Além dos recursos disponibilizados voltados para compilação, baseados em Make e Cmake, a biblioteca libIEC61850 disponibiliza na pasta `tools/model_generator/`, um código em Java para gerar os *headers* e *sources* (`static_model.h` e `static_model.c` respectivamente) necessários para a tradução em linguagem C de um arquivo SCL que modela um IED.

Se o usuário utiliza o gerenciador de pacotes `apt`, presente no Debian e suas distribuições derivadas, basta rodar o comando "`sudo apt install default-jre`"[1]. Vale lembrar que as referências do gerenciador de pacotes devem ser previamente atualizadas com os comandos de "`apt update`" e "`apt upgrade`". Após satisfeito o requisito, assumindo que o presente diretório é a mesma pasta `tools/model_generator/`, o usuário deve inserir no terminal:

```
$java -jar genmodel.jar my_model.icd
```

, onde `my_model.icd` é um nome qualquer, que serve de exemplo para obter o modelo de IED na linguagem C. Feito, os arquivos `static_model.h` e `static_model.c` serão gerados no diretório atual, cujos binários podem ser linkados na próxima construção do projeto. Para fins práticos, no caso de aproveitamento de um dos exemplos fornecidos pela biblioteca, o arquivo SCL pode ser trazido para o mesmo subdiretório do referido exemplo e seu nome especificado no *Makefile* associado, como apresentado na linha 8 da Figura 26.

Figura 28 – Modificação no *Makefile* contido no exemplo *server_example_basic_io*

```
1  LIBIEC_HOME=../..
2
3  PROJECT_BINARY_NAME = server_example_basic_io
4  PROJECT_SOURCES = server_example_basic_io.c
5  PROJECT_SOURCES += static_model.c
6
7  # PROJECT_ICD_FILE = simpleIO_direct_control.cid
8  PROJECT_ICD_FILE = ION8650.scd
9
10 include $(LIBIEC_HOME)/make/target_system.mk
11 include $(LIBIEC_HOME)/make/stack_includes.mk
12
13 all:    $(PROJECT_BINARY_NAME)
14
15 include $(LIBIEC_HOME)/make/common_targets.mk
16
```

Fonte: Autor, 2024

Seguindo este procedimento alternativo, o comando `$make model` pode ser executado para gerar os mesmos arquivos `static_model.h` e `static_model.c`.

[1] É necessário instalar o JRE (Java Runtime Environment) na versão 6 ou superior

3.2 Implantação da biblioteca libIEC61850 no ESP32

Antes de voltar a atenção para os procedimentos de implantação em si, é preciso fazer as considerações sobre os procedimentos de instalação do ESP-IDF. Além destes passos fugirem do escopo do trabalho, a documentação traz o passo-a-passo bem detalhado para cada plataforma, inclusive para Windows. Apesar disso, por preferência do autor, escolheu-se a instalação no ambiente Linux por meio do WSL2 por trazer facilidades quanto ao gerenciamento de pacotes, variáveis do ambiente e interação do Git com o Github.

O ponto de partida para a implantação da biblioteca libIEC61850 nos microcontroladores da Espressif compreende a realização de que o ESP32 e o ESP32-S3 empregam as unidades de processamento chamadas Tensilica Xtensa LX6 e Tensilica Xtensa LX7 respectivamente, cujas arquiteturas não são suportadas pelas toolchains disponibilizados por padrão nos arquivos agregados ao *makefile* standalone da libIEC61850, que são listados conforme Figura 29, sendo necessário ou fazer as adequações necessárias aos Makefiles para seu uso ou simplesmente ignorá-los, dualidade esta que separa o desenvolvimento deste trabalho em duas abordagens de implementação, descritas como segue.

Figura 29 – Toolchains suportados pela biblioteca libIEC61850

```
1   UNAME := $(shell uname)
2
3   MIPSEL_TOOLCHAIN_PREFIX=mipsel-openwrt-linux-
4   #ARM_TOOLCHAIN_PREFIX=arm-linux-
5   #ARM_TOOLCHAIN_PREFIX=arm-linux-gnueabi-
6   #ARM_TOOLCHAIN_PREFIX=arm-poky-linux-gnueabi-
7   ARM_TOOLCHAIN_PREFIX=arm-linux-gnueabihf-
8   UCLINUX_ARM_TOOLCHAIN_PREFIX=arm-uclinux-elf-
9   UCLINUX_XPORT_TOOLCHAIN_PREFIX=m68k-uclinux-
10  MINGW_TOOLCHAIN_PREFIX=i586-mingw32msvc-
11  #MINGW_TOOLCHAIN_PREFIX=x86_64-w64-mingw32-
12  MINGW64_TOOLCHAIN_PREFIX=x86_64-w64-mingw32-
13
14  #POWERPC_TOOLCHAIN_PREFIX=powerpc-poky-linux-
15  POWERPC_TOOLCHAIN_PREFIX=powerpc-linux-gnu-
```

Fonte: Repositório oficial da libIEC61850

A primeira abordagem de adaptar o código foi fazer uma análise do *makefile* e suas dependências de modo que os binários, gerados por uma compilação da

biblioteca libIEC61850 com o devido *toolchain* do microcontrolador, sejam linkados aos binários de um projeto de autoria própria com o ESP-IDF após sua fase de compilação. Esta metodologia está descrita no trecho "*Using third party Cmake projects*" em [27], que propõe o uso de código de terceiros como um componente do projeto, e usa comandos Cmake *vanilla* (em sua versão pura) como forma de sobrepor os passos que ocorreriam em uma compilação padrão em projetos com o ESP-IDF, permitindo ao usuário maior controle do processo de montagem do referido componente desde a preparação.

Com esta abordagem implementada, foi possível obter a biblioteca estática *libIEC61850.a*, mas durante o processo de *link*, notou-se que os *headers* atribuídos vinham do Sistema Operacional subjacente (Linux) ao invés do SDK da Espressif. Adicionalmente, investigou-se a documentação do esp-idf e a ferramenta Cmake de como resolver o problema mencionado, mas mesmo após algumas adequações, a montagem continuava a buscar os *headers* incorretos.

A segunda abordagem, baseada no último item da lista da Seção 2.6.2.1, que consiste em desprezar a filosofia estrutural de um projeto com esp-idf e então, a partir de um projeto existente que emprega Cmake, os componentes IDF seriam consumidos como biblioteca, sendo linkados aos targets (bibliotecas e executáveis) no processo final da montagem. Esta abordagem é de grande conveniência para o presente caso, pois pequenas modificações são adicionadas ao arquivo `CMakelists.txt` localizado na raiz do repositório da libIEC61850, sendo o acréscimo dos comandos ***idf_build_process*** e ***target_link_libraries*** as principais alterações. Mais detalhes são tratados no próximo tópico.

3.2.1 Modificações do arquivo CMakeLists.txt na raiz do projeto

Antes mesmo de modificar o arquivo `CMakeLists.txt` propriamente dito, procedeu-se definindo as variáveis `$TARGET` e `$TOOLCHAIN` de ambiente no Shell, conforme mostrado abaixo:

```
$export TARGET=esp32s3

$export TOOLCHAIN=$HOME/.espressif/tools/xtensa-esp-
elf/esp-13.2.0_20230928/xtensa-esp-elf/bin/xtensa-$TARGET-elf-
```

Para não precisar configurar manualmente estas variáveis outra vez, é interessante acrescentar as definições no final do arquivo `~/.bashrc`. A numeração `13.2.0_20230928` é consequência da instalação do ESP-IDF da *master branch* para o início de janeiro de 2024. Novas versões incluirão outra numeração, basta averiguar no respectivo diretório para obter a numeração correta. A definição destas variáveis de ambiente é importante para o Cmake encontrar a correta localização conforme as linhas **4 a 8** do `CMakeLists.txt`, que se encontra no apêndice A.

A primeira inserção em `CMakeLists.txt` é a condição (linhas **13 e 23** do apêndice A), que serve para identificar o uso de uma *toolchain* relacionada ao microcontrolador, além de redefinir a variável `CMAKE_SYSTEM_NAME` para "`Generic`", pois sem isso, quando o comando ***idf_build_process*** é executado, há condições internas que não são atendidas.

O segundo trecho de código inserido em CMakeLists.txt é o bloco condicional (linhas **169 e 195** do apêndice A), que destaca o uso do comando ***idf_build_process***, apresentado em Using ESP-IDF in custom Cmake projects [27] e desenvolvido no exemplo *idf_as_lib* do repositório do ESP-IDF.

O terceiro trecho de código inserido em `CMakeLists.txt` corresponde às linhas **220 e 229** do apêndice A, responsável pelo link dos binários gerados para a camada de abstração de hardware (hal) aos componentes idf de dependência direta. Além disso, são adicionadas duas flags ao *gcc* para a compilação do hal: `-mlongcalls` e `-mfix-esp32-psram-cache-issue`.

A primeira flag `-mlongcalls`, instrui ao compilador a realizar chamadas longas, isto é, armazenar o endereço das funções em um registrador antes de fazer a chamada de sub-rotina. Vale observar que o compilador não realiza as chamadas longas em funções declaradas com o especificador *static* ou quando da precedência da diretiva `#pragma no_long_calls`. Seu uso é necessário no ESP-IDF quando se utiliza códigos de terceiros que não são identificados como componentes IDF. Nesses casos, conforme reportado em [28], várias mensagens de erro são exibidas, sendo elas similares à seguinte:

```
dangerous relocation: call8: call target out of range: malloc
```

Por outro lado, a segunda flag `-mfix-esp32-psram-chache-issue` é necessária apenas para versões antigas dos microcontroladores da Espressif quando o projeto depende do recurso de memória RAM externa por SPI (Serial Peripheral Interface), portanto, seu uso representa apenas uma boa prática visando compatibilidade, ao invés uma real necessidade.

A quarta inserção em `CMakeLists.txt` é o bloco entre as linhas **437 a 442** do apêndice A. Analogamente ao caso anterior, este trecho é responsável pela definição das bibliotecas estáticas a serem incluídos pelo *linker* no *target* iec61850. Note que os binários do hal são inclusos juntamente com os componentes *newlib* e *lwip* que correspondem à implementação da biblioteca padrão da linguagem C e ao stack TCP/IP respectivamente. Novamente a flag `-mlongcalls` é incluída para habilitar chamadas longas.

A quinta inserção em `CMakeLists.txt` é o bloco entre as linhas **451 e 478** do apêndice A, responsável pelas diretivas que definem o binário executável a ser carregado na memória flash do microcontrolador e por associar os *headers* e *sources*, bem como as bibliotecas estáticas e os componentes IDF necessários.

O leitor pode estar questionando a razão pela qual foi decidido colocar todas as alterações apenas no CMakeLists.txt na raíz do projeto libIEC61850. De início, tentou-se aproveitar a diretiva *add_directory* do Cmake para distribuir as adequações pelos outros CMakeLists.txt nos demais diretórios, mas graças ao bug reportado em [29], é esclarecido que a diretiva *add_executable* deve ser chamada no `CMakeLists.txt` na raiz do repositório. Então, devido a esta condição, foi decidido que o código ficaria mais organizado concentrando as alterações em apenas um arquivo.

Cabe ressaltar que o suporte ao protocolo GOOSE foi **desabilitado** pela flag "`CONFIG_INCLUDE_GOOSE_SUPPORT`". A variável `DEBUG` também foi habilitada, sendo um importante recurso para localizar e resolver os erros encontrados.

3.2.2 Adequações no código de exemplo de servidor

Conforme mencionado anteriormente, é objetivo do projeto formar um servidor MMS com o ESP32-S3, que representa o nível de Bay em um SAS. Para facilitar

o desenvolvimento, pode-se utilizar um dos códigos de servidor disponíveis como exemplo no repositório da libIEC61850 – neste trabalho, o exemplo *server_example_basic_io* foi escolhido. Seis adequações foram necessárias para o seu devido uso. Dentre elas:

a) Mover as diretivas de pré-processamento para um arquivo separado `sever_example_basic_io.h` que contém os protótipos das funções;

b) Remover a chamada "`signal(SIGINT, sigint_handler)`", que serviria para interromper o fluxo do programa com a inserção de "*ctrl*+*c*" no teclado. Isto se justifica pelo fato de o esp-idf não possuir conceito de *handlers* para sinais POSIX, portanto é um comando que não tem cabimento para esta aplicação;

c) Remediar o problema de derreferência a um ponteiro nulo durante a chamada da função *printf* como descrito na Seção 3.2.5.

d) Substituir a declaração da função `int main(int argc, char** argv)` por `int server_example_basic_io(void)` para depois ser chamada em outro arquivo.

e) Compatibilizar os nomes dos objetos de dados (DO) nas linhas 46 a 63 e 145 a 157, referentes às variáveis controláveis SPCSO.

f) Compatibilizar os nomes dos objetos de dados (DO) nas linhas 208 a 218, referentes às variáveis de medição analógica.

Os itens e) e f), são adequações necessárias para tornar o código compatível com o arquivo de configuração SCL que modela o IED, portanto, estas alterações são necessárias apenas quando é de interesse do usuário substituir o arquivo SCL fornecido no exemplo.

Os itens a) a d) são estritamente necessárias para aproveitar o exemplo de servidor, evitando ao máximo causar alterações. A ideia aqui é encapsular o exemplo para externamente resolver outras definições pertinentes ao microcontrolador, tais como configuração do Wi-Fi e instância do SNTP. Para simplificar estas providências básicas, o exemplo de código `esp-idf/examples/protocols/sntp` foi tomado como template. Deste arquivo, outras 3 adequações foram necessárias:

g) Substituição do comando *esp_deep_sleep* na linha 121 e associados por um *vTaskDelay* de 1 segundo.

h) Incremento do comando *server_example_basic_io* ao final da função *void app_main* e da respectiva diretiva `#include "server_example_basic_io.h"` no início do arquivo para incluir o código de servidor encapsulado;

i) Remoção das inicializações dos componentes no início da definição da função `static void obtain_time(void)` e remoção das desinicializações no final da definição da mesma função.

Um detalhe que pode ter passado despercebido é que com o uso do template *sntp*, é preciso garantir que as constantes ainda sejam definidas bem como são originalmente feitas com o suporte de um arquivo `Kconfig.projbuild`. Como solução, um componente trivial chamado *kcomp* foi criado, apenas com o intuito de estabelecer estas definições.

3.2.3 Adequações na camada de abstração de hardware (hal)

Na camada de abstração de hardware, apenas algumas modificações pontuais foram realizadas, todas foram alterações dos headers incluídos no início dos arquivos dos subdiretórios `hal/filesystem/`, `hal/serial/`, `hal/socket/` e `hal/thread/`. Cabe destacar que apenas `hal/socket/` foi baseado no arquivo *`socket_bsd.c`*, como detalhado na documentação [30], enquanto todos os outros permaneceram baseados nos códigos preparados para Linux.

Dentre os arquivos .c na camada hal, o único que se manteve com erros persistentes foi o arquivo contido em `hal/ethernet/`. Isto se deve a diferenças significativas de hardware, de modo que o padrão POSIX não tenha sido atendido no desenvolvimento do esp-idf. Esta pendência, no entanto, faz com que o protocolo GOOSE não possa ser habilitado, o que, na verdade, não é um problema, tendo em vista que a proposta deste trabalho apenas se limita aos protocolos MMS.

3.2.4 Adequações no ESP-IDF

Seguindo os procedimentos de instalação mencionados na Seção 3.2, como consequência, dois novos diretórios foram adicionados ao ambiente de desenvolvimento. Um deles é `~/esp/esp-idf/` clonado do repositório do repositório do framework, o qual é referido como `${ESP_IDF}` e o outro é `~/.espressif/` resultado da execução do script de instalação.

Ao atender o último item da lista da Seção 2.6.2.1, o *bug* reportado em [31] foi encontrado, isto é, os componentes idf não estavam sendo encontrados quando da preparação dos arquivos para compilação. O que se observa deste problema é que os componentes estão sendo importados para o projeto de forma não global. A solução, então é adicionar a *flag* `GLOBAL` à linha 168 do arquivo `${ESP_IDF}/tools/cmake/component.cmake`, onde `${ESP_IDF}` é o diretório `esp-idf/` clonado do repositório na máquina.

Outro problema encontrado é relacionado diretamente aos padrões POSIX especificados para manipulações de dados de tempo. Em particular, a função *timegm* [2] não foi representada no ESP-IDF. Portanto, como a libIEC61850 depende desta função, mensagens de erro são exibidas durante a montagem do projeto. Para resolver este problema, dois arquivos sofreram adequações:

- Adicionar o trecho de código da Figura 30 ao final do arquivo `${ESP_IDF}componentes/newlib/time.c`, conforme [32];
- Adicionar o protótipo `time_t timegm(struct tm const* t)` à linha 58 do arquivo `xtensa_esp_elf/xtensa-esp_idf/sys_include/time.h`.

[2] *Timegm* manpage: https://man7.org/linux/man-pages/man3/timegm.3.html

Figura 30 – Código portátil para implementar a função `time_gm` especificada pelo padrão POSIX

```
// Algorithm: http://howardhinnant.github.io/date_algorithms.html
int days_from_epoch(int y, int m, int d)
{
    y -= m <= 2;
    int era = y / 400;
    int yoe = y - era * 400;                                   // [0, 399]
    int doy = (153 * (m + (m > 2 ? -3 : 9)) + 2) / 5 + d - 1;  // [0, 365]
    int doe = yoe * 365 + yoe / 4 - yoe / 100 + doy;           // [0, 146096]
    return era * 146097 + doe - 719468;
}

// It  does not modify broken-down time
time_t timegm(struct tm const* t)
{
    int year = t->tm_year + 1900;
    int month = t->tm_mon;          // 0-11
    if (month > 11)
    {
        year += month / 12;
        month %= 12;
    }
    else if (month < 0)
    {
        int years_diff = (11 - month) / 12;
        year -= years_diff;
        month += 12 * years_diff;
    }
    int days_since_epoch = days_from_epoch(year, month + 1, t->tm_mday);

    return 60 * (60 * (24L * days_since_epoch + t->tm_hour) + t->tm_min) + t->tm_sec;
}
```

Fonte: Retirado de [32]

3.2.5 Outros erros importantes (issues)

Durante a implementação do presente projeto, uma série de erros foram encontrados, que representaram dificuldades para a obtenção dos resultados. Nas sessões anteriores alguns erros foram sendo naturalmente retratados ao longo das últimas sessões. A seguir serão listados apenas os principais erros remanescentes.

O primeiro erro é referente à definição de tipos definidos no framework esp-idf. A biblioteca libIEC61850, possui alguns trechos que admitem que os tipos int32_t e int são os mesmos. Por outro lado, como descrito em [33] e reportado em [34], existe uma peculiaridade quanto aos tipos int32_t e uint32_t para o compilador gcc na plataforma Xtensa. Esta peculiaridade é que inicialmente, o tipo int32_t era outrora um

nome alternativo para o tipo int tal qual o tipo uint32_t era definição alternativa para unsigned int. Porém, a partir da versão gcc 11, o tipo int32_t passa a ser definido como long int e uint32_t se tornou unsigned long int para se adequar à arquitetura riscv32, conforme a Tabela 3. Para os compiladores associados ao esp-idf, tanto int como long int e suas versões unsigned são todos tipos inteiros de 32 bits, porém existe uma diferença de *rank*. A combinação destes dois fatores gera contradições nas definições de ponteiros ou protótipos de funções que dependem de int32_t ou de uint32_t.

Tabela 3 – A atualização do compilador GCC11 da arquitetura Xtensa combatibilizou tipagem com a arquitetura riscv32

	GCC 8	GCC 11
Xtensa	(unsigned) int	(unsigned) long int
riscv32	(unsigned) long int	(unsigned) long int

Por sorte, este problema pôde ser facilmente resolvido realizando as seguintes substituições:

- Substituir “int frsmId” por “int32_t frsm_Id” na linha 734 do arquivo src/mms/iso_mms/cliente/mms_client_files.c do repositório da biblioteca libiec61850.
- Substituir “int frsmId” por “int32_t frsm_Id” na linha 3873 do arquivo src/mms/iso_mms/mms_files_connection.c do repositório da biblioteca libiec61850.

O segundo erro encontrado foi que o arquivo “.elf” com os binários a serem carregados na memória flash é maior que os 1MB disponíveis na tabela de partição padrão. A solução para este problema é habilitar a flag `PARTITION_TABLE_SINGLE_APP_LARGE` através do *menuconfig*, que toma o espaço de memória reservado para carregamento OTA, expandindo a tabela de partição para 1.5 MB [35]. Aplicada a correção, o código pôde ser carregado na memória flash, sobrando 15% de espaço livre na tabela de partição.

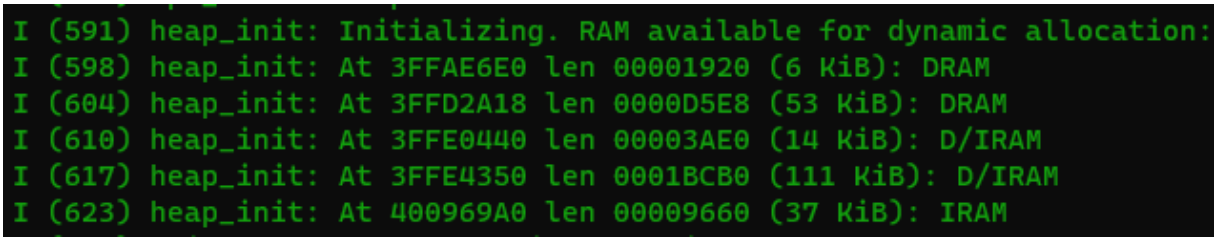
O terceiro erro é um problema de alocação dinâmica de memória no *heap*. Isto ocorreu devido à seguinte mensagem de erro da

Figura 31 – Espaço de memória insuficiente para alocação dinâmica

Fonte: Autor, 2024

, ou seja, as chamadas de funções exigem 64 kiB de memória RAM, enquanto a capacidade disponível é de 6 kiB. Nota-se que que este espaço em memória disponível é um número bem específico e que remete à primeira seção de memória RAM disponível para alocação dinâmica quando da inicialização padrão do *heap*, conforme [36], ilustrado com a Figura 32. Adicionalmente, a terceira seção ocupa 111 kiB, espaço mais que suficiente para alocar os mencionados 64 kiB. Então, aproveitar este trecho de memória seria a medida imediata, porém nada foi encontrado sobre isso na documentação.

Figura 32 – Blocos de memória disponíveis para alocação dinâmica de memória RAM quando da inicialização do *heap*



Fonte: Autor, 2024

A resolução deste problema foi crítica para o desenvolvimento do projeto. Uma solução possível seria simplificar a biblioteca por meio de redução de recursos desnecessários, conforme descrito por [37], e outra solução, que foi a adotada, é simplesmente usar o recurso de PSRAM (RAM pseudoestática) disponibilizado apenas pelo hardware especificado com a letra R (consultar Figura 21 e Figura 23), que consiste em incorporar (com certas restrições) uma memória RAM externa ao mapa de memória por meio do periférico de SPI. Este recurso é interessante pois estende a memória RAM disponível em pelo menos 2 Mbytes a depender da versão do chip.

No caso deste projeto, o desenvolvimento foi iniciado com o microcontrolador ESP32-D0WD-V3, porém, como este não suporta o recurso de PSRAM, foi necessário migrar para o modelo ESP32-S3 N16R8. Este modelo foi escolhido porque era o único com estoque no Brasil disponível para envio rápido que possui o recurso necessário.

O quarto erro importante foi sobre o ajuste do tamanho do stack. Por padrão, a stack é configurada para o tamanho de 3 kBytes. Com este valor, ao rodar o projeto, a seguinte mensagem de erro é impressa na tela:

ERROR A stack overflow in task pthread has been detected.

Após mudar o tamanho do stack de 3 para 15 kBytes [38], esta mensagem de erro não tornou a aparecer novamente.

O quinto e último erro de grande importância ocorreu quando do emprego de um ponteiro nulo para um *specifier* “%s” na função *printf*. Por inspeção, os desenvolvedores da libIEC61850, admitem, provavelmente por influência das versões mais modernas no gcc, que o resultado desta chamada resulta em `(null)`, enquanto no padrão C99 este caso não é especificado, portanto o comportamento é indefinido. No caso do esp-idf, o vetor nulo é entendido como uma derreferência a um espaço inválido na memória. Note que este problema é de mesma natureza do famoso erro *NullPointerException* da linguagem Java. Este problema não foi resolvido de fato neste trabalho, foi apenas remediado adequando-se o argumento da primeira chamada da função *printf* na função *rcbEventHandler*, cuja declaração se encontra no arquivo *server_example_basic_io.c*.

Todos os erros das últimas sessões estão listados na Tabela 4.

Tabela 4 – Principais erros e referências

Erro encontrado	Provisão	Referência
Toolchain não encontrada	Aplicar `CMAKE_SYSTEM_NAME` como genérico	-
`Call8: call target out of range`	Aplicar flag `–mlongcalls` nos targets que não são reconhecidos como componentes IDF.	[28]
Autodependência do componente `__idf_extensa`	Usar diretiva *add_executable* apenas no arquivo `CMakeLists.txt` raiz.	[29]
Componentes idf não encontrados durante a montagem do projeto.	Aplicar *flag* `GLOBAL` na importação dos componentes.	[31]
Função *timegm* não definida.	Definir a função no componente *newlib*	[32]
Conversão de `long int` para `in32_t`	Adequar trechos da biblioteca libIEC61850	[33]. [34]
Tamanho da tabela de partição	Ajustes com *menuconfig*	[35]
Erro de alocação de memória (Figura 32)	**Substituição do ESP32 pelo ESP32-S3**	Figura 21 e Figura 23
Erro de Stack Overflow	Ajustes com *menuconfig*	[38]
printf não aceita ponteiro nulo	Incrementar condições nas chamadas printf quando a libIEC61850 receber ponteiro nulo como argumento.	-

3.3 Testes de performance

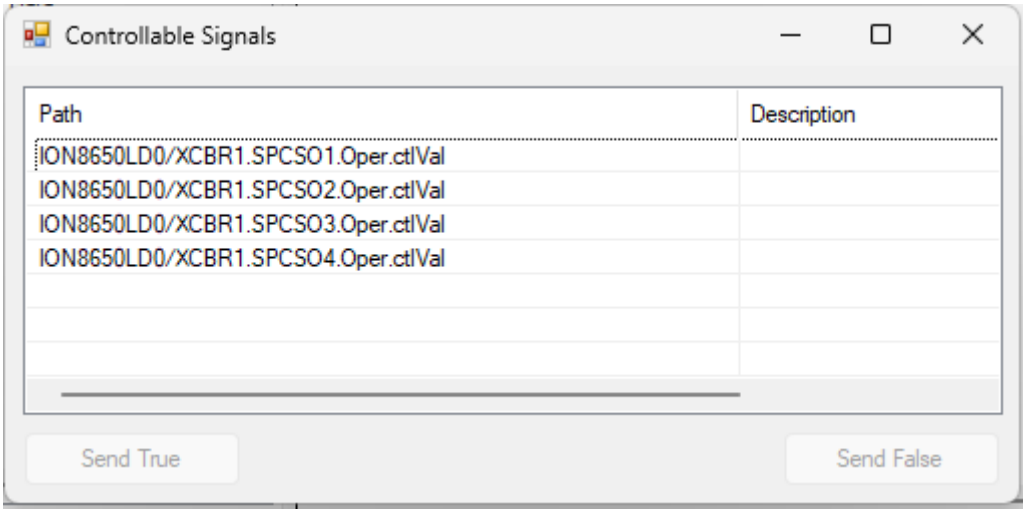
Após o desenvolvimento das últimas seções, foi verificada a necessidade de avaliar o comportamento do exemplo de servidor criado em duas abordagens. Ambas as abordagens seguem a topologia da Figura 3, onde um computador pessoal se comporta como o cliente através do IEDExplorer (que, além de mandar pacotes MMS, também possui a capacidade de registrar o intervalo das transmissões de dados) e a plataforma ESP32-S3 / Raspberry Pi assume a função de servidor. Os dispositivos são interligados por Wi-fi através de um roteador doméstico em uma rede LAN. A primeira abordagem consiste no envio de requisições em intervalos fixos de 1 segundo solicitando atualização de todos os dados presentes no modelo de IED do servidor, ou seja, todos os Logical Nodes que constam no arquivo SCL fornecido. A segunda abordagem consiste em ativar todos os *report buffers* presentes e avaliar a quantidade de dados que migram do servidor ao cliente. Alguns ensaios são descritos na próxima seção.

Durante os testes, dois arquivos SCL diferentes foram usados no servidor. Um dos arquivos utilizados foi o *simpleIO_direct_control.cid*, o qual é fornecido junto ao exemplo do código da biblioteca libIEC61850. O outro arquivo foi uma adaptação do modelo do multimedidor ION8650 obtido da Schneider Electric [39].

A mencionada adaptação se resume em três pontos:

- Para cada `j` de 1 a 16 os DA (Data Atributes) reunidos no Dataset (DS) `ION8650LLN0/CUSTOMIO.CUS1_GGIO.AnInj.mag` foram substituídos por `ION8650LLN0/CUSTOMIO.CUS1_GGIO.AnInj.mag.f`;
- Um novo Logical Node XCBR foi criado com as variáveis booleanas SPCO1, SPCO2, SPCO3 e SPCO4 do tipo Single Point Controllable (SPC) para que o software IEDExplorer possa tornar disponível a opção de “*Controllable Signals*”, conforme a Figura 33.
- Adição do Dataset PersonalEvent referente ao ponto anterior com os respectivos blocos Report Control com e sem buffer.

Figura 33 – Recurso do IEDExplorer para manipulação de variáveis em tempo real



Fonte: Autor, 2024.

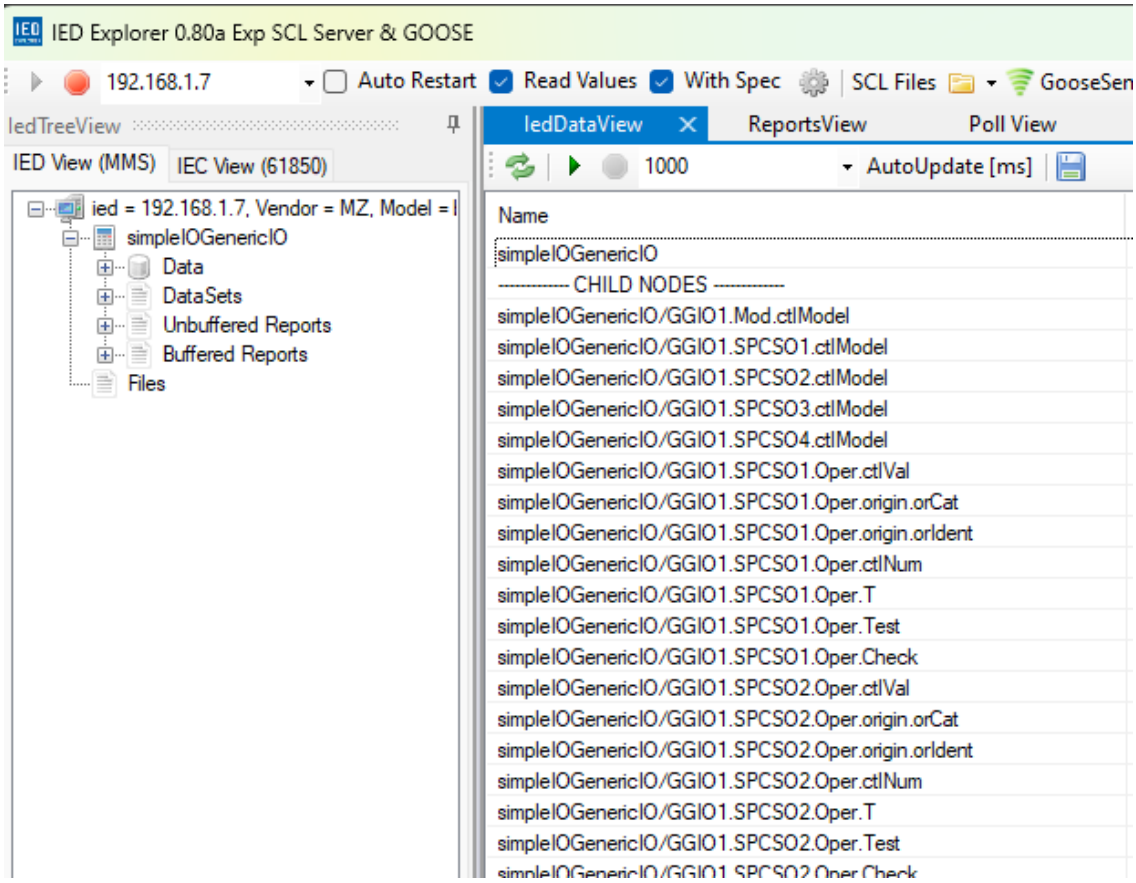
4 RESULTADOS E DISCUSSÕES

A inicialização do cliente é combinada com busca de todos os Logical Devices presentes no servidor. No caso do uso do Logical Device *simpleIOGenericIO* o tempo necessário para o envio dos dados é de 1,01 segundos, enquanto a inicialização da emulação do qualímetro ION8650 é de 7,3 segundos.

Antes de realizar os procedimentos da Seção 3.3, alguns testes de leitura e escrita com poucas quantidades de dados. Dentre estes testes, um que se destacou consistiu na requisição do dataset *ION8650LD0/CUSTOMIO* com o servidor rodando no ESP32-S3, que consistiu em 26 mensagens de requisição com 205 bytes e mais 26 mensagens de resposta com 43 bytes por mensagem. Este ensaio prévio número zero, obtendo da Figura 37, indicou intervalo médio de transmissão de 48 milissegundos e um desvio padrão de 2,7321E-04 ms.

O ensaio de número 1, conforme primeira abordagem, executa leitura de todas as informações do Logical Device (LD) simpleIOGenericIO (do SCL padrão) com o ESP32-S3, resultando em requisições de 472 bytes com retorno de 1586 bytes. O intervalo médio entre as requisições e as respostas, num conjunto de 18 pares, foi de 104 ms e o desvio padrão de 5,38E-4 ms. A Figura 34 mostra o painel principal do IEDExplorer com algumas das variáveis que constam neste ensaio. A Figura 38, por outro lado, mostra os pares requisição/respostas relacionados a este ensaio.

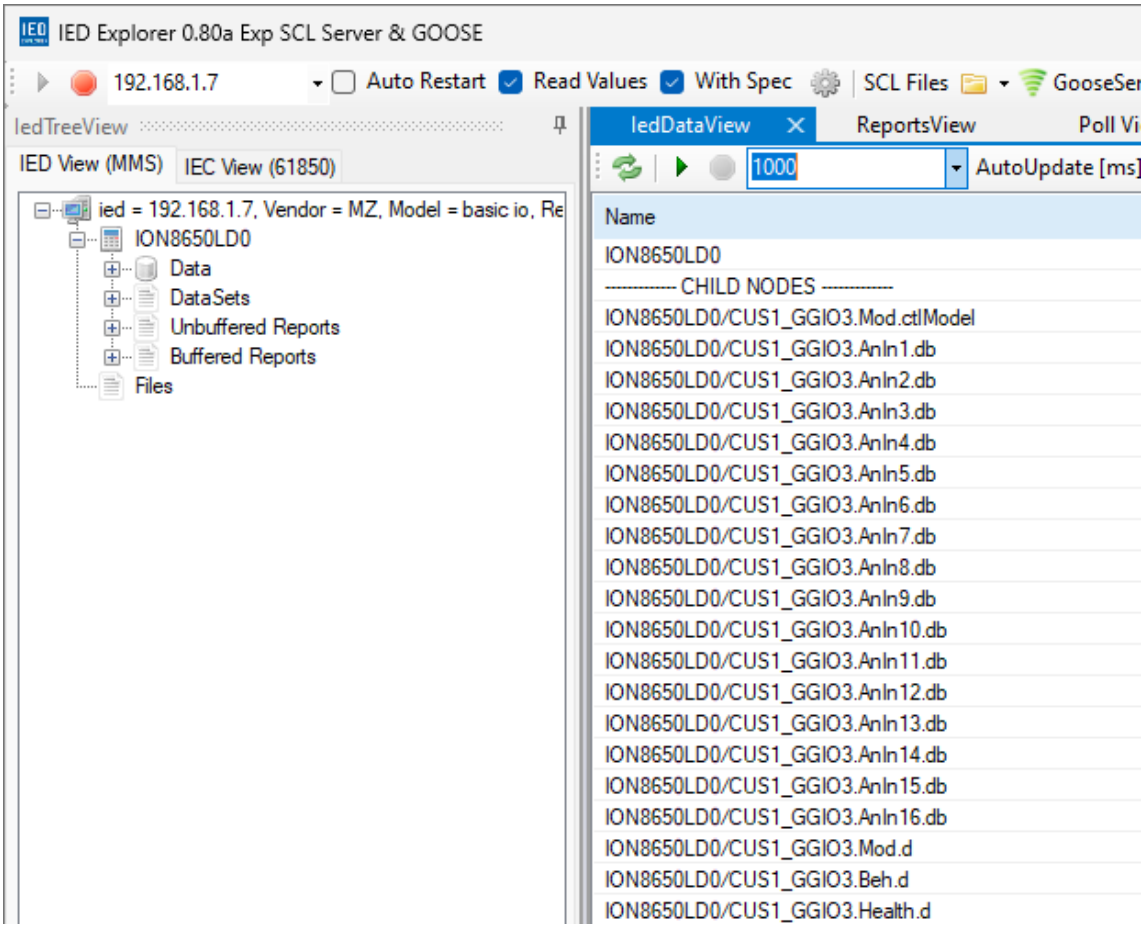
Figura 34 – Painel de leitura de dados no ensaio 1 – ESP32-S3 e *simpleIOGenericIO* como LD



Fonte: Autor, 2024.

O ensaio de número 2 consiste na leitura, usando o ESP32-S3, de todas as informações referentes ao Logical Device LD0 do qualímetro ION8650, resultando em requisições de 1327 bytes com respostas de 23713 bytes. O intervalo médio entre as requisições e as repostas, num conjunto de 49 pares, foi de 380 ms e o desvio padrão de 1,149E-3 ms. A Figura 35 mostra o painel do IEDExplorer com algumas das variáveis contidas no LD *ION8650LD0* e o intervalo de requisições em 1 segundo. A Figura 39 apresenta as amostras coletadas neste ensaio.

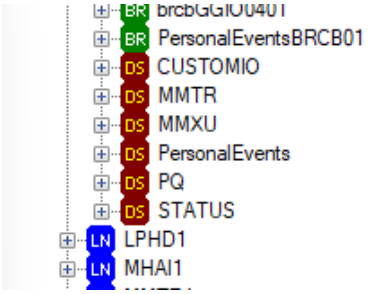
Figura 35 – Painel de leitura de dados no ensaio 2 – ESP32-S3 e *ION8650LD0* como LD



Fonte: Autor, 2024.

. O ensaio de número 3, emprega novamente o LD0 do qualímetro ION8650, habilitou-se todos Buffers de controle de Report associados a todos os *datasets* no principal LN chamado LNN0, recebendo de forma contínua pacotes de dados do servidor embarcado no microcontrolador. O intervalo médio entre os pacotes com as amostras de 871 bytes é de 117 ms com desvio padrão de 1,528E-4 ms. A Figura 36 lista todos os DS envolvidos e a Figura 40 contém as amostras coletadas neste ensaio.

Figura 36 – *Datasets* que tiveram os respectivos Report Control Blocks ativados



Fonte: Autor, 2024.

Os ensaios 4, 5 e 6 são análogos aos 3 anteriores, com a diferença de que a plataforma empregada foi o Raspberry Pi ao invés do ESP32-S3. Os resultados das 3 amostras junto dos resultados anteriores estão organizados na Tabela 5.

Tabela 5 – Síntese de resultados

Ensaio	Plataforma	Arquivo SCL	Método de ensaio	Intervalo Médio (ms)	Desvio Padrão (segundos)
0	ESP32-S3	*ION8650LD0*	AutoUpdate	48	2,732E-07
1	ESP32-S3	*simpleIOGenericIO*	Auto Update	104	5,38E-07
2		*ION8650LD0*	Auto Update	380	11,99E-07
3			Relatórios periódicos	117	1,528E-07
4	Raspberry PI 3 Model B+	*simpleIOGenericIO*	Auto Update	55	4,007E-07
5		*ION8650LD0*	Auto Update	134	7,509E-07
6			Relatórios periódicos	102	2,730E-07

A tabela Tabela 5 permite fazer algumas constatações imediatas:

- Dentre os ensaios realizado, não foram observadas restrições de uso do Raspberry Pi para a aplicação;
- Para pequenas quantidades de dados, o emprego do ESP32-S3 é razoável pois é possível trocar mensagens em menos de 100 milissegundos, conforme o ensaio zero;
- Para quantidades maiores de dados, como no segundo ensaio, o ESP32 pode ser usado para mensagens de velocidade média apenas;
- Se tempo de transmissão for usado como parâmetro de desempenho, então claramente o Raspberry Pi apresentou resultados melhores.
- Mesmo com as limitações apresentadas, o uso de *Report Control Blocks* se mostra efetivo quanto à redução do uso de banda na rede local. Note que o tempo de resposta do ESP32-S3 é de poucos milissegundos a mais que o Raspberry Pi.

Faz-se necessário compreender qual plataforma é mais adequada em uma dada circunstância. Primeiro, ao longo do texto, nota-se que, a implementação no Raspberry Pi é simplificada por ser baseado no Linux, além de demonstrar os menores tempos na transmissão de dados, o que o torna uma opção prática e suficiente para mensagens de velocidade média (vide Tabela 2).

Por outro lado, o uso do ESP32 se mostrou oneroso por não ter compatibilidade direta com a biblioteca libIEC61850, ocasionando em algumas dificuldades, como os sintetizados na Tabela 4 além de sua performance inferior, se comparado com o Raspberry Pi. Entretanto, se o recurso de Report Control Blocks for bem aproveitado, ainda assim o ESP32 se mostra uma alternativa viável, sobretudo em situações em que se deseja desenvolver um produto comercial, e, portanto, de fabricação em larga escala por apresentar custos de aquisição de ao menos 5 vezes mais barato, ainda, em uma estimativa conservadora.

5 CONCLUSÕES E PROPOSTAS DE CONTINUIDADE

Conforme apresentado ao longo do texto, a norma IEC 61850 tem fundamental importância na garantia do bom funcionamento da infraestrutura de um SAS. O grau de complexidade da norma representa um dos desafios de validação projetos/protótipos, sendo então conveniente ter em mãos do projetista possibilidades de baixo custo de se avaliar o comportamento de um IED sob determinadas configurações. Por questões de popularidade e de disponibilidade no mercado, as plataformas selecionadas para implementação do protocolo MMS especificado pela norma foram o ESP32 da Espressif e o Raspberry Pi 3 Model B+.

O desenvolvimento demonstrou com sucesso que é possível o uso das plataformas embarcadas ESP32 e Raspberry PI para se comportar como um IED no nível de bay conforme a estrutura padrão de uma Sistema de Automação de Subestação. Os resultados encontrados foram satisfatórios para o Raspberry Pi e satisfatórios para o ESP32 sob as condições descritas na seção de resultados.

Quanto às dificuldades encontradas para o Raspberry Pi não foram encontrados erros. Por outro lado, no caso do ESP32, devido a especificidades de hardware, foi decidido migrar para o modelo ESP32-S3 e, como esperado, várias dificuldades foram encontradas no sentido de tornar a biblioteca libIEC61850 compatível com o framework (ESP-IDF) do microcontrolador.

A partir do progresso obtido, lista-se as seguintes propostas de continuidade:

- Realizar as adequações necessárias no arquivo `hal/ethernet.c` para habilitar os protocolos GOOSE e Sampled Values, sendo, inclusive, necessário substituir o uso do protocolo SNTP pelo PTP junto ao driver *netif* do ESP-IDF;
- Explorar recursos do ESP-IDF ou simplificar o código da libIEC61850 para ser possível alocar o *heap* na memória RAM interna, dispensando o uso de PSRAM. Ou, como alternativa, avaliar se a velocidade de acesso à PSRAM, de 40 MHz, é um fator limitante na manipulação de dados;
- Reimplementar a biblioteca libiec61850 em um microcontrolador que possua dois periféricos MAC para aumentar a confiabilidade de transmissão de dados.

REFERÊNCIAS

1. ONS. **Sumário Executivo PAR/PEL 2023 - Plano de Operação Elétrica de Médio Prazo do SIN - Ciclo 2024 - 2028**. ONS. Rio de Janeiro. 2024.

2. SANTANA, R.; MONASTÉRIO, L.; CALDEIRA, T. C. M. Regulação da microgeração e minigeração distribuída de energia elétrica no Brasil. **Revista do Serviço Público**, 21 dez. 2023. 24.

3. ONS. Submódulo 2.11 - Requisitos mínimos para os sistemas de proteção, de registro de perturbações e de teleproteção. **Procedimentos de Rede**, 31 mar. 2021. 22.

4. SIEMENS. SIPROTEC 4 - Devices - Catalog SIP. **SIPROTEC 7SJ64**, Nuremberg, set. 2016. Disponivel em: <https://www.siemens.com/global/en/products/energy/energy-automation-and-smart-grid/protection-relays-and-control/siprotec-4/overcurrent-and-motor-protection/multifunction-protection-with-synchronization-siprotec-7sj64.html>. Acesso em: 21 fev. 2024.

5. HITACHI. RTU530 License Activation Process. **RTU500 series function and software**, 11 abr. 2023. Disponivel em: <https://www.hitachienergy.com/products-and-solutions/substation-automation-protection-and-control/products/remote-terminal-units/rtu500-series-function-and-software>. Acesso em: 21 fev. 2024.

6. IEC. **IEC 61850-5 - Communication networks and systems for power utility automation - Part 7-1: Basic communication structure - Principles and models - Part 5: Communication requirements for functions and device models**. Geneva, Switzerland. 2013.

7. IEC. **IEC/TR 61850-1 - Communication networks and systems for power utility automation - Part 1: Introduction and overview**. Geneva, Switzerland. 2013.

8. HWANG, S. Investigation of Wireless IEC 61850 MMS using Raspberry Pi. **TURCOMAT**, 13, Jul. 2021. pp. 5770-5778. Disponivel em: <https://web.archive.org/web/20240211091535/https://turcomat.org/index.php/turkbilmat/article/view/9843>. Acesso em: 20 fev. 2024.

9. GROUP, D. U. DNP Primer, Revision A. **dnp.org**, 18 fev. March, 2005. Disponivel em: <https://www.dnp.org/Portals/0/AboutUs/DNP3%20Primer%20Rev%20A.pdf>. Acesso em: 30 mar. 2024.

10. COUTINHO, P. S. Protocolos para Sistemas Elétricos: DNP 3.0. **Industria Automática**, . 2015. Disponivel em: <https://industriaautomatica.wordpress.com/2015/09/26/protocolos-para-sistemas-eletricos-dnp-3-0/>. Acesso em: 30 mar. 2024.

11. ERICKSON, A. What are The Pros and Cons of The DNP3 Protocol? **DPS Telecom**, . 2019. Disponivel em: <https://www.dpstele.com/blog/dnp3-pros-and-cons.php>. Acesso em: 30 mar. 2024.

12. IEC. **IEC 61850-6 - Communication networks and systems for power utility automation - Part 6: Configuration description language for communication in power utility automation systems related to IEDs**. Geneva, Swizterland. 2018.

13. IEC. **IEC 61850-7-1 - Communication networks and systems for power utility automation - Part 7-1: Basic communication structure - Principles and models**. Geneva, Switzerland. 2011.

14. IEC. **IEC 61850-7-2 - Communication networks and systems for power utility automation – Part 7-2: Basic information and communication structure – Abstract communication service interface (ACSI)**. Geneva, Switzerland. 2020.

15. ABNT. **NBR 16932 - Redes e sistemas de comunicação para automação de sistemas de potência - Orientações sobre engenharia de rede**. ABNT. Rio de janeiro. 2020.

16. PHAM, G. T. **Integration of IEC 61850 MMS and LTE to support smart metering communications**. [S.l.]: [s.n.], 2013.

17. NA, M. W. **Emulação de IED embarcando o protocolo MMS em um sistema computacional de baixo custo**. Niterói: SDC/BEE (UFF), 2022.

18. HARA, M.; KRIGER, C. Implementation of an IEC 61850-Based Metering Device Using Open-Source Software. **Universal Journal of Electrical and Electronic Engineering 8(2)**, maio 2021. pp. 17-34.

19. MZ AUTOMATION. Official repository for libIEC61850, the open-source library for the IEC 61850 protocols, 18 dez. 2023. Disponivel em: <https://github.com/mz-automation/libiec61850>. Acesso em: 20 fev. 2024.

20. RASPBERRY PI LTD. Raspberry Pi 3 Model B+. **raspberrypi.com**, 2023. Disponivel em: <https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf>. Acesso em: 30 mar. 2024.

21. RASPBERRY PI FOUNDATION. Raspberry Pi 3 Model B+. **raspberrypi.com**. Disponivel em: <https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/>. Acesso em: 30 mar. 2024.

22. ESPRESSIF. ESP-IDF C Support, 12 nov. 2023. Disponivel em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/c.html>. Acesso em: 20 fev. 2024.

23. ESPRESSIF. ESP32 Series Datasheet. **espressif.com**. Disponivel em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em: 30 mar. 2024.

24. ESPRESSIF. ESP32-S3 Series Datasheet. **espressif.com**, 2023. Disponivel em: <https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf>. Acesso em: 30 mar. 2024.

25. DATASHEET HUB. WROOM ESP32 Wifi Based Microcontroller Deveopment Board. **datasheethub.com**, 2022. Disponivel em: <https://www.datasheethub.com/wroom-esp32-wifi-based-microcontroller-development-board/#google_vignette>. Acesso em: 30 mar. 2024.

26. SAVARATI. Placa ESP32-S3 N16R8: Placa versátil com baixo custo ideal para automações e Iot com comunicação sem fio. **blog.savarati.com.br**. Disponivel em: <https://blog.saravati.com.br/placa-esp32-s3-n16r8-automacoes-iot-sem-fio/>. Acesso em: 30 mar. 2024.

27. ESPRESSIF. ESP-IDF Build System, 05 dez. 2023. Disponivel em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system.html>. Acesso em: 20 fev. 2024.

28. ESPRESSIF. Static library - call target out of range, 11 nov. 2017. Disponivel em: <https://esp32.com/viewtopic.php?t=549>. Acesso em: 20 fev. 2023.

29 ESPRESSIF. IDF-8677, 7 nov. 2022. Disponivel em: <https://github.com/espressif/esp-
. idf/issues/10122>. Acesso em: 20 fev. 2023.

30 ESPRESSIF. lwIP, 1 dez. 2023. Disponivel em: <https://docs.espressif.com/projects/esp-
. idf/en/stable/esp32/api-guides/lwip.html>. Acesso em: 20 fev. 2024.

31 ESPRESSIF. IDFGH11470 - Scope of CMake component targets is not global. **Github**,
. 2023. Disponivel em: <https://github.com/espressif/esp-idf/issues/12599>. Acesso em:
21 fev. 2024.

32 STÖR, M. Time cross platform. **stackoverflow.com**, 2023. Disponivel em:
. <http://howardhinnant.github.io/date_algorithms.html>. Acesso em: 21 fev. 2024.

33 ESPRESSIF. GCC - Migration from 4.4. to 5.0, 16 ago. 2022. Disponivel em:
. <https://github.com/espressif/esp-idf/commits/v5.0/docs/en/migration-guides/release-
5.x/gcc.rst>. Acesso em: 20 fev. 2024.

34 ESPRESSIF. IDFGH-5124 - int32_t should be int and not long int, 20 abr. 2021.
. Disponivel em: <https://github.com/espressif/esp-idf/issues/6906>. Acesso em: 20 fev.
2024.

35 ESPRESSIF. Partition Tables, 31 jan. 2024. Disponivel em:
. <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/partition-
tables.html>. Acesso em: 20 fev. 2024.

36 ESPRESSIF. Heap Memory Allocation, 03 jan. 2024. Disponivel em:
. <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-
reference/system/mem_alloc.html>. Acesso em: 20 fev. 2024.

37 MEKKANEN, M. **Light-Weight IEC 61850 IEDs - Reducing complexity - SESP**. Vaasa.
. 2019.

38 ESPRESSIF. Stack Overflow in task pthread, 28 jun. 2018. Disponivel em:
. <https://esp32.com/viewtopic.php?t=6253>. Acesso em: 20 fev. 2024.

39 SCHNEIDER. ION61850 - Multimedidor, 2024. Disponivel em:
. <https://www.se.com/br/pt/product-range/61053-ion8650/>. Acesso em: 21 fev. 2024.

APÊNDICES

APÊNDICE A – CMakeLists.txt da raiz do projeto modificado

```
cmake_minimum_required(VERSION 3.16)

# automagically detect if we should cross-compile
if(DEFINED ENV{TOOLCHAIN})
    set(CMAKE_C_COMPILER    $ENV{TOOLCHAIN}gcc)
    set(CMAKE_CXX_COMPILER  $ENV{TOOLCHAIN}g++)
    set(CMAKE_AR    "$ENV{TOOLCHAIN}ar" CACHE FILEPATH "CW archiver" FORCE)
endif()

project(libiec61850)
ENABLE_TESTING()

if($ENV{TOOLCHAIN} MATCHES ".*xtensa-esp.*")
    set(ESP32 1)
    set(CMAKE_SYSTEM_NAME "Generic")
    # TARGET must be defined else use a default one
    set(targets "esp32" "esp32s2" "esp32s3" "esp32c3" "esp32c2" "esp32c6"
"esp32h2" "esp32p4")
    IF("${TARGET}" IN_LIST targets)
    ELSE()
        message(STATUS "Unknown target ${TARGET}, substutuiting for
esp32s3")
        set(TARGET "esp32s3")
    ENDIF()
endif()

set(LIB_VERSION_MAJOR "1")
set(LIB_VERSION_MINOR "5")
set(LIB_VERSION_PATCH "3")
set(LIB_VERSION
"${LIB_VERSION_MAJOR}.${LIB_VERSION_MINOR}.${LIB_VERSION_PATCH}")

set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH}
"${CMAKE_SOURCE_DIR}/third_party/cmake/modules/")

# feature checks
include(CheckLibraryExists)
check_library_exists(rt clock_gettime "time.h"
CONFIG_SYSTEM_HAS_CLOCK_GETTIME)

# check if we are on a little or a big endian
include (TestBigEndian)
test_big_endian(PLATFORM_IS_BIGENDIAN)

set(CONFIG_MMS_MAXIMUM_PDU_SIZE "65000" CACHE STRING "Configure the maximum
size of an MMS PDU (default 65000)"    )
set(CONFIG_MAXIMUM_TCP_CLIENT_CONNECTIONS 5 CACHE STRING "Configure the
maximum number of clients allowed to connect to the server")

option(BUILD_EXAMPLES "Build the examples" OFF)
option(BUILD_PYTHON_BINDINGS "Build Python bindings" OFF)

option(CONFIG_MMS_SINGLE_THREADED "Compile for single threaded version" ON)
```

```
54  option(CONFIG_MMS_THREADLESS_STACK "Optimize stack for threadless operation
55  (warning: single- or multi-threaded server will not work!)" OFF)
56  set(CONFIG_MMS_SERVER_MAX_GET_FILE_TASKS 5 CACHE STRING "Configure the
57  maximum number of get file tasks")
58  set(CONFIG_MMS_MAX_NUMBER_OF_DATA_SET_MEMBERS 100 CACHE STRING "Configure
59  the maximum number of dataSet members")
60
61  option(CONFIG_ACTIVATE_TCP_KEEPALIVE "Activate TCP keepalive" ON)
62  option(CONFIG_INCLUDE_GOOSE_SUPPORT "Build with GOOSE support" OFF)
63
64  option(CONFIG_USE_EXTERNAL_MBEDTLS_DYNLIB "Build with pre-compiled mbedtls
65  dynamic library" OFF)
66
67  set(CONFIG_EXTERNAL_MBEDTLS_DYNLIB_PATH
68  "${CMAKE_CURRENT_LIST_DIR}/third_party/mbedtls/mbedtls-2.28/library" CACHE
69  STRING "Path to search for the mbedtls dynamic libraries" )
70  set(CONFIG_EXTERNAL_MBEDTLS_INCLUDE_PATH
71  "${CMAKE_CURRENT_LIST_DIR}/third_party/mbedtls/mbedtls-2.28/include" CACHE
72  STRING "Path to search for the mbedtls include files" )
73
74  # choose the library features which shall be included
75  option(CONFIG_IEC61850_CONTROL_SERVICE "Build with support for IEC 61850
76  control features" ON)
77  option(CONFIG_IEC61850_REPORT_SERVICE "Build with support for IEC 61850
78  reporting services" ON)
79  option(CONFIG_IEC61850_LOG_SERVICE "Build with support for IEC 61850
80  logging services" ON)
81  option(CONFIG_IEC61850_SERVICE_TRACKING "Build with support for IEC 61850
82  service tracking" ON)
83  option(CONFIG_IEC61850_SETTING_GROUPS "Build with support for IEC 61850
84  setting group services" ON)
85  option(CONFIG_IEC61850_SUPPORT_USER_READ_ACCESS_CONTROL "Allow user
86  provided callback to control read access" ON)
87  option(CONFIG_IEC61850_RCB_ALLOW_ONLY_PRECONFIGURED_CLIENT "allow only
88  configured clients (when pre-configured by ClientLN)" OFF)
89  set(CONFIG_IEC61850_SG_RESVTMS 300 CACHE STRING "Configure the maximum
90  number of SG RESVTMS")
91
92  set(CONFIG_REPORTING_DEFAULT_REPORT_BUFFER_SIZE "65536" CACHE STRING
93  "Default buffer size for buffered reports in byte" )
94
95  # advanced options
96  option(DEBUG "Enable debugging mode (include assertions)" OFF)
97  option(DEBUG_SOCKET "Enable printf debugging for socket layer" ${DEBUG})
98  option(DEBUG_COTP "Enable COTP printf debugging" ${DEBUG})
99  option(DEBUG_ISO_SERVER "Enable ISO SERVER printf debugging" ${DEBUG})
100 option(DEBUG_ISO_CLIENT "Enable ISO CLIENT printf debugging" ${DEBUG})
101 option(DEBUG_IED_SERVER "Enable IED SERVER printf debugging" ${DEBUG})
102 option(DEBUG_IED_CLIENT "Enable IED CLIENT printf debugging" ${DEBUG})
103 option(DEBUG_MMS_SERVER "Enable MMS SERVER printf debugging" ${DEBUG})
104 option(DEBUG_MMS_CLIENT "Enable MMS CLIENT printf debugging" ${DEBUG})
105 option(DEBUG_GOOSE_SUBSCRIBER "Enable GOOSE subscriber printf debugging"
106 ${DEBUG})
107 option(DEBUG_GOOSE_PUBLISHER "Enable GOOSE publisher printf debugging"
108 ${DEBUG})
109 option(DEBUG_SV_SUBSCRIBER "Enable Sampled Values subscriber debugging"
110 ${DEBUG})
111 option(DEBUG_SV_PUBLISHER "Enable Sampled Values publisher debugging"
112 ${DEBUG})
113 option(DEBUG_HAL_ETHERNET "Enable Ethernet HAL printf debugging" ${DEBUG})
114
```

```
115 include_directories(
116     ${CMAKE_CURRENT_BINARY_DIR}/config
117     ${CMAKE_CURRENT_LIST_DIR}/src/common/inc
118     ${CMAKE_CURRENT_LIST_DIR}/src/goose
119     ${CMAKE_CURRENT_LIST_DIR}/src/sampled_values
120     ${CMAKE_CURRENT_LIST_DIR}/src/hal/inc
121     ${CMAKE_CURRENT_LIST_DIR}/src/iec61850/inc
122     ${CMAKE_CURRENT_LIST_DIR}/src/iec61850/inc_private
123     ${CMAKE_CURRENT_LIST_DIR}/src/mms/inc
124     ${CMAKE_CURRENT_LIST_DIR}/src/mms/inc_private
125     ${CMAKE_CURRENT_LIST_DIR}/src/mms/iso_mms/asn1c
126     ${CMAKE_CURRENT_LIST_DIR}/src/logging
127 )
128
129 set(API_HEADERS
130     hal/inc/hal_base.h
131     hal/inc/hal_time.h
132     hal/inc/hal_thread.h
133     hal/inc/hal_filesystem.h
134     hal/inc/hal_ethernet.h
135     hal/inc/hal_socket.h
136     hal/inc/tls_config.h
137     src/common/inc/libiec61850_common_api.h
138     src/common/inc/linked_list.h
139     src/iec61850/inc/iec61850_client.h
140     src/iec61850/inc/iec61850_common.h
141     src/iec61850/inc/iec61850_server.h
142     src/iec61850/inc/iec61850_model.h
143     src/iec61850/inc/iec61850_cdc.h
144     src/iec61850/inc/iec61850_dynamic_model.h
145     src/iec61850/inc/iec61850_config_file_parser.h
146     src/mms/inc/mms_value.h
147     src/mms/inc/mms_common.h
148     src/mms/inc/mms_types.h
149     src/mms/inc/mms_type_spec.h
150     src/mms/inc/mms_client_connection.h
151     src/mms/inc/mms_server.h
152     src/mms/inc/iso_connection_parameters.h
153     src/goose/goose_subscriber.h
154     src/goose/goose_receiver.h
155     src/goose/goose_publisher.h
156     src/sampled_values/sv_subscriber.h
157     src/sampled_values/sv_publisher.h
158     src/logging/logging_api.h
159 )
160
161 if(MSVC AND MSVC_VERSION LESS 1800)
162     include_directories(
163         ${CMAKE_CURRENT_LIST_DIR}/src/vs
164     )
165 endif(MSVC AND MSVC_VERSION LESS 1800)
166
167 if(CONFIG_USE_EXTERNAL_MBEDTLS_DYNLIB)
168 set(WITH_MBEDTLS 1)
169 set(USE_PREBUILD_MBEDTLS 1)
170 set(MBEDTLS_INCLUDE_DIR ${CONFIG_EXTERNAL_MBEDTLS_INCLUDE_PATH})
171 endif(CONFIG_USE_EXTERNAL_MBEDTLS_DYNLIB)
172
173 if(EXISTS ${CMAKE_CURRENT_LIST_DIR}/third_party/mbedtls/mbedtls-2.28)
174 set(WITH_MBEDTLS 1)
```

```
175 set(MBEDTLS_INCLUDE_DIR
176 "${CMAKE_CURRENT_LIST_DIR}/third_party/mbedtls/mbedtls-2.28/include")
177 endif(EXISTS ${CMAKE_CURRENT_LIST_DIR}/third_party/mbedtls/mbedtls-2.28)
178
179 if(WITH_MBEDTLS)
180
181 add_definitions(-DCONFIG_MMS_SUPPORT_TLS=1)
182
183 endif(WITH_MBEDTLS)
184
185 include(CheckCCompilerFlag)
186
187 check_c_compiler_flag("-Wredundant-decls" SUPPORT_REDUNDANT_DECLS)
188 # if (SUPPORT_REDUNDANT_DECLS)
189 #     set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wredundant-decls")
190 # endif(SUPPORT_REDUNDANT_DECLS)
191
192 # write the detected stuff to this file
193 configure_file(
194     ${CMAKE_CURRENT_LIST_DIR}/config/stack_config.h.cmake
195     ${CMAKE_CURRENT_BINARY_DIR}/config/stack_config.h
196 )
197
198 if (ESP32) #build process
199     include($ENV{IDF_PATH}/tools/cmake/idf.cmake)
200
201 idf_build_component($ENV{IDF_PATH}/examples/common_components/protocol_exam
202 ples_common)
203     idf_build_component(${CMAKE_CURRENT_LIST_DIR}/kcomp)
204     set(components
205         freertos esptool_py esp_wifi
206         newlib
207         esp_system
208         esp_event
209         log
210         esp_common
211         esp_hw_support
212         nvs_flash
213         protocol_examples_common
214         esp_netif
215         lwip
216         console
217         kcomp
218         esp_gdbstub
219         esp_psram
220     )
221     idf_build_process("${TARGET}"
222         COMPONENTS ${components}
223         BUILD_DIR ${CMAKE_BINARY_DIR}
224     )
225     set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
226 endif(ESP32)
227
228 include_directories(
229     ${CMAKE_CURRENT_LIST_DIR}/hal/inc
230 )
231
232 if(!ESP32) #HALL
233     add_subdirectory("${CMAKE_CURRENT_LIST_DIR}/hal")
234 else()
235
```

```
236         set (libhal_esp32_SRCS
237         ${CMAKE_CURRENT_LIST_DIR}/hal/socket/esp32/socket_bsd.c
238         #${CMAKE_CURRENT_LIST_DIR}/hal/ethernet/esp32/ethernet_linux.c
239         ${CMAKE_CURRENT_LIST_DIR}/hal/thread/linux/thread_linux.c
240         ${CMAKE_CURRENT_LIST_DIR}/hal/filesystem/esp32/file_provider_linux.c
241         ${CMAKE_CURRENT_LIST_DIR}/hal/time/unix/time.c
242         ${CMAKE_CURRENT_LIST_DIR}/hal/serial/esp32/serial_port_linux.c
243         ${CMAKE_CURRENT_LIST_DIR}/hal/memory/lib_memory.c
244         )
245
246         set (libhal_SRCS
247             ${libhal_esp32_SRCS}
248             )
249
250         add_library (hal STATIC ${libhal_SRCS})
251         target_compile_options(hal PRIVATE "-mlongcalls")
252         #target_compile_options(hal PRIVATE "-mfix-esp32-psram-cache-issue")
253         target_link_libraries(hal
254             idf::lwip
255             idf::esp_netif
256             idf::esp_wifi
257             idf::vfs
258             idf::newlib
259         )
260     endif(!ESP32) #HALL
261
262     if(DEBUG)
263         set(CMAKE_BUILD_TYPE Debug)
264     endif(DEBUG)
265
266     if(BUILD_EXAMPLES)
267         add_subdirectory(${CMAKE_CURRENT_LIST_DIR}/examples)
268     endif(BUILD_EXAMPLES)
269
270     if(!ESP32) #SRC
271         add_subdirectory(${CMAKE_CURRENT_LIST_DIR}/src)
272     else()
273         set (lib_common_SRCS
274         ./src/common/string_map.c
275         ./src/common/map.c
276         ./src/common/linked_list.c
277         ./src/common/byte_buffer.c
278         ./src/common/string_utilities.c
279         ./src/common/buffer_chain.c
280         ./src/common/conversions.c
281         ./src/common/mem_alloc_linked_list.c
282         ./src/common/simple_allocator.c
283         ./src/mms/iso_server/iso_connection.c
284         ./src/mms/iso_server/iso_server.c
285         ./src/mms/iso_acse/acse.c
286         ./src/mms/iso_mms/common/mms_type_spec.c
287         ./src/mms/iso_mms/common/mms_value.c
288         ./src/mms/iso_mms/common/mms_common_msg.c
289         ./src/mms/iso_mms/client/mms_client_initiate.c
290         ./src/mms/iso_mms/client/mms_client_write.c
291         ./src/mms/iso_mms/client/mms_client_identify.c
292         ./src/mms/iso_mms/client/mms_client_status.c
293         ./src/mms/iso_mms/client/mms_client_named_variable_list.c
294         ./src/mms/iso_mms/client/mms_client_connection.c
295         ./src/mms/iso_mms/client/mms_client_files.c
296         ./src/mms/iso_mms/client/mms_client_get_namelist.c
```

```
297         ./src/mms/iso_mms/client/mms_client_get_var_access.c
298         ./src/mms/iso_mms/client/mms_client_common.c
299         ./src/mms/iso_mms/client/mms_client_read.c
300         ./src/mms/iso_mms/client/mms_client_journals.c
301         ./src/mms/iso_mms/server/mms_read_service.c
302         ./src/mms/iso_mms/server/mms_file_service.c
303         ./src/mms/iso_mms/server/mms_association_service.c
304         ./src/mms/iso_mms/server/mms_identify_service.c
305         ./src/mms/iso_mms/server/mms_status_service.c
306         ./src/mms/iso_mms/server/mms_named_variable_list_service.c
307         ./src/mms/iso_mms/server/mms_value_cache.c
308         ./src/mms/iso_mms/server/mms_get_namelist_service.c
309         ./src/mms/iso_mms/server/mms_access_result.c
310         ./src/mms/iso_mms/server/mms_server.c
311         ./src/mms/iso_mms/server/mms_server_common.c
312         ./src/mms/iso_mms/server/mms_named_variable_list.c
313         ./src/mms/iso_mms/server/mms_domain.c
314         ./src/mms/iso_mms/server/mms_device.c
315         ./src/mms/iso_mms/server/mms_information_report.c
316         ./src/mms/iso_mms/server/mms_journal.c
317         ./src/mms/iso_mms/server/mms_journal_service.c
318         ./src/mms/iso_mms/server/mms_server_connection.c
319         ./src/mms/iso_mms/server/mms_write_service.c
320         ./src/mms/iso_mms/server/mms_get_var_access_service.c
321         ./src/mms/iso_cotp/cotp.c
322         ./src/mms/iso_presentation/iso_presentation.c
323         ./src/mms/asn1/ber_decode.c
324         ./src/mms/asn1/asn1_ber_primitive_value.c
325         ./src/mms/asn1/ber_encoder.c
326         ./src/mms/asn1/ber_integer.c
327         ./src/mms/iso_client/iso_client_connection.c
328         ./src/mms/iso_common/iso_connection_parameters.c
329         ./src/mms/iso_session/iso_session.c
330         ./src/iec61850/client/client_control.c
331         ./src/iec61850/client/client_report_control.c
332         ./src/iec61850/client/client_goose_control.c
333         ./src/iec61850/client/client_sv_control.c
334         ./src/iec61850/client/client_report.c
335         ./src/iec61850/client/ied_connection.c
336         ./src/iec61850/common/iec61850_common.c
337         ./src/iec61850/server/impl/ied_server.c
338         ./src/iec61850/server/impl/ied_server_config.c
339         ./src/iec61850/server/impl/client_connection.c
340         ./src/iec61850/server/model/model.c
341         ./src/iec61850/server/model/dynamic_model.c
342         ./src/iec61850/server/model/cdc.c
343         ./src/iec61850/server/model/config_file_parser.c
344         ./src/iec61850/server/mms_mapping/control.c
345         ./src/iec61850/server/mms_mapping/mms_mapping.c
346         ./src/iec61850/server/mms_mapping/reporting.c
347         ./src/iec61850/server/mms_mapping/mms_goose.c
348         ./src/iec61850/server/mms_mapping/mms_sv.c
349         ./src/iec61850/server/mms_mapping/logging.c
350         ./src/logging/log_storage.c
351         )
352
353         set (lib_asn1c_SRCS
354         ./src/mms/iso_mms/asn1c/DataAccessError.c
355         ./src/mms/iso_mms/asn1c/DeleteNamedVariableListRequest.c
356         ./src/mms/iso_mms/asn1c/constr_SET_OF.c
357         ./src/mms/iso_mms/asn1c/MmsPdu.c
```

```
./src/mms/iso_mms/asn1c/GetNamedVariableListAttributesResponse.c
./src/mms/iso_mms/asn1c/BIT_STRING.c
./src/mms/iso_mms/asn1c/ber_tlv_tag.c
./src/mms/iso_mms/asn1c/constr_SEQUENCE_OF.c
./src/mms/iso_mms/asn1c/asn_SET_OF.c
./src/mms/iso_mms/asn1c/ReadResponse.c
./src/mms/iso_mms/asn1c/InformationReport.c
./src/mms/iso_mms/asn1c/ConfirmedServiceRequest.c
./src/mms/iso_mms/asn1c/DeleteNamedVariableListResponse.c
./src/mms/iso_mms/asn1c/asn_SEQUENCE_OF.c
./src/mms/iso_mms/asn1c/VariableAccessSpecification.c
./src/mms/iso_mms/asn1c/GetVariableAccessAttributesRequest.c
./src/mms/iso_mms/asn1c/xer_support.c
./src/mms/iso_mms/asn1c/ObjectName.c
./src/mms/iso_mms/asn1c/NativeEnumerated.c
./src/mms/iso_mms/asn1c/per_encoder.c
./src/mms/iso_mms/asn1c/constr_SEQUENCE.c
./src/mms/iso_mms/asn1c/GetNameListResponse.c
./src/mms/iso_mms/asn1c/MMSString.c
./src/mms/iso_mms/asn1c/InitiateErrorPdu.c
./src/mms/iso_mms/asn1c/IndexRangeSeq.c
./src/mms/iso_mms/asn1c/ConfirmedErrorPDU.c
./src/mms/iso_mms/asn1c/UnconfirmedService.c
./src/mms/iso_mms/asn1c/UTF8String.c
./src/mms/iso_mms/asn1c/ServiceError.c
./src/mms/iso_mms/asn1c/TimeOfDay.c
./src/mms/iso_mms/asn1c/GetNameListRequest.c
./src/mms/iso_mms/asn1c/asn_codecs_prim.c
./src/mms/iso_mms/asn1c/Data.c
./src/mms/iso_mms/asn1c/ScatteredAccessDescription.c
./src/mms/iso_mms/asn1c/ReadRequest.c
./src/mms/iso_mms/asn1c/per_decoder.c
./src/mms/iso_mms/asn1c/Identifier.c
./src/mms/iso_mms/asn1c/ServiceSupportOptions.c
./src/mms/iso_mms/asn1c/Integer8.c
./src/mms/iso_mms/asn1c/ConfirmedServiceResponse.c
./src/mms/iso_mms/asn1c/ParameterSupportOptions.c
./src/mms/iso_mms/asn1c/Integer16.c
./src/mms/iso_mms/asn1c/ber_tlv_length.c
./src/mms/iso_mms/asn1c/OCTET_STRING.c
./src/mms/iso_mms/asn1c/DefineNamedVariableListRequest.c
./src/mms/iso_mms/asn1c/FloatingPoint.c
./src/mms/iso_mms/asn1c/xer_encoder.c
./src/mms/iso_mms/asn1c/Unsigned8.c
./src/mms/iso_mms/asn1c/BOOLEAN.c
./src/mms/iso_mms/asn1c/INTEGER.c
./src/mms/iso_mms/asn1c/UnconfirmedPDU.c
./src/mms/iso_mms/asn1c/DataSequence.c
./src/mms/iso_mms/asn1c/constraints.c
./src/mms/iso_mms/asn1c/der_encoder.c
./src/mms/iso_mms/asn1c/VisibleString.c
./src/mms/iso_mms/asn1c/InitiateResponsePdu.c
./src/mms/iso_mms/asn1c/StructComponent.c
./src/mms/iso_mms/asn1c/Address.c
./src/mms/iso_mms/asn1c/Unsigned16.c
./src/mms/iso_mms/asn1c/ber_decoder.c
./src/mms/iso_mms/asn1c/per_support.c
./src/mms/iso_mms/asn1c/WriteResponse.c
./src/mms/iso_mms/asn1c/InitRequestDetail.c
./src/mms/iso_mms/asn1c/InitiateRequestPdu.c
./src/mms/iso_mms/asn1c/DefineNamedVariableListResponse.c
```

```
419         ./src/mms/iso_mms/asn1c/NULL.c
420         ./src/mms/iso_mms/asn1c/ListOfVariableSeq.c
421         ./src/mms/iso_mms/asn1c/UtcTime.c
422         ./src/mms/iso_mms/asn1c/ConcludeResponsePDU.c
423         ./src/mms/iso_mms/asn1c/AccessResult.c
424         ./src/mms/iso_mms/asn1c/Integer32.c
425         ./src/mms/iso_mms/asn1c/GetNamedVariableListAttributesRequest.c
426         ./src/mms/iso_mms/asn1c/VariableSpecification.c
427         ./src/mms/iso_mms/asn1c/Unsigned32.c
428         ./src/mms/iso_mms/asn1c/constr_CHOICE.c
429         ./src/mms/iso_mms/asn1c/AlternateAccess.c
430         ./src/mms/iso_mms/asn1c/ObjectClass.c
431         ./src/mms/iso_mms/asn1c/InitResponseDetail.c
432         ./src/mms/iso_mms/asn1c/ConfirmedResponsePdu.c
433         ./src/mms/iso_mms/asn1c/GetVariableAccessAttributesResponse.c
434         ./src/mms/iso_mms/asn1c/NativeInteger.c
435         ./src/mms/iso_mms/asn1c/xer_decoder.c
436         ./src/mms/iso_mms/asn1c/AlternateAccessSelection.c
437         ./src/mms/iso_mms/asn1c/ConfirmedRequestPdu.c
438         ./src/mms/iso_mms/asn1c/ConcludeRequestPDU.c
439         ./src/mms/iso_mms/asn1c/WriteRequest.c
440         ./src/mms/iso_mms/asn1c/RejectPDU.c
441         ./src/mms/iso_mms/asn1c/TypeSpecification.c
442         ./src/mms/iso_mms/asn1c/constr_TYPE.c
443         ./src/mms/iso_mms/asn1c/GeneralizedTime.c
444         )
445
446         set (lib_goose_SRCS
447         ./src/goose/goose_subscriber.c
448         ./src/goose/goose_receiver.c
449         ./src/goose/goose_publisher.c
450         )
451
452         set (lib_sv_SRCS
453         ./src/sampled_values/sv_subscriber.c
454         ./src/sampled_values/sv_publisher.c
455         )
456
457         set (library_SRCS
458             ${lib_common_SRCS}
459             ${lib_asn1c_SRCS}
460             ${lib_goose_SRCS}
461             ${lib_sv_SRCS}
462             )
463
464         add_library(iec61850 STATIC ${library_SRCS})
465
466         target_compile_options(iec61850 INTERFACE "-mlongcalls")
467
468         target_link_libraries(iec61850
469             hal
470             idf::lwip
471             idf::newlib
472         )
473     endif(!ESP32) #SRC
474
475     install(FILES ${API_HEADERS} DESTINATION include/libiec61850 COMPONENT
476     Development)
477
478     if(BUILD_PYTHON_BINDINGS)
479         add_subdirectory(${CMAKE_CURRENT_LIST_DIR}/pyiec61850)
```

```
480 endif(BUILD_PYTHON_BINDINGS)
481
482 # Generate executable for esp32
483 if (ESP32)
484     include_directories(
485         examples/server_example_basic_io/include
486         examples/server_example_basic_io
487     )
488     add_executable(example_main.elf
489         examples/server_example_basic_io/server_example_basic_io.c
490         examples/server_example_basic_io/static_model.c
491         main/example_main.c
492     )
493     target_compile_options(example_main.elf INTERFACE "-mlongcalls")
494     target_link_libraries(example_main.elf
495         iec61850
496         hal
497         idf::newlib
498         idf::esp_system
499         idf::esp_event
500         idf::log
501         idf::esp_common
502         idf::esp_hw_support
503         idf::nvs_flash
504         idf::protocol_examples_common
505         idf::esp_netif
506         idf::lwip
507         idf::esp_phy
508     )
509     idf_build_executable(example_main.elf)
510 endif(ESP32)
511
512 set(CPACK_PACKAGE_DESCRIPTION "IEC 61850 MMS/GOOSE client and server
513 library")
514 set(CPACK_PACKAGE_DESCRIPTION_SUMMARY "IEC 61850 MMS/GOOSE client and
515 server library")
516 set(CPACK_PACKAGE_VENDOR "MZ Automation GmbH")
517 set(CPACK_PACKAGE_CONTACT "info@libiec61850.com")
518 set(CPACK_PACKAGE_VERSION_MAJOR "${LIB_VERSION_MAJOR}")
519 set(CPACK_PACKAGE_VERSION_MINOR "${LIB_VERSION_MINOR}")
520 set(CPACK_PACKAGE_VERSION_PATCH "${LIB_VERSION_PATCH}")
521 set(CPACK_PACKAGE_FILE_NAME
522 "${CMAKE_PROJECT_NAME}_${LIB_VERSION_MAJOR}.${LIB_VERSION_MINOR}.${CPACK_PA
523 CKAGE_VERSION_PATCH}_${CMAKE_SYSTEM_PROCESSOR}")
524 set(CPACK_SOURCE_PACKAGE_FILE_NAME
525 "${CMAKE_PROJECT_NAME}_${LIB_VERSION_MAJOR}.${LIB_VERSION_MINOR}.${CPACK_PA
526 CKAGE_VERSION_PATCH}")
527
528 set(CPACK_COMPONENTS_ALL Libraries Development Applications)
529 #set(CPACK_PACKAGE_INSTALL_DIRECTORY "${CMAKE_PROJECT_NAME}")
530
531 if(EXISTS "${CMAKE_ROOT}/Modules/CPack.cmake")
532     include(InstallRequiredSystemLibraries)
533
534     include(CPack)
535 endif(EXISTS "${CMAKE_ROOT}/Modules/CPack.cmake")
```

APÊNDICE B – Dados não processados dos ensaios 0 a 6

Figura 37 - Capturas do ensaio zero

Packet Nr.	Time	Dir	MMS Pdu	MMS Service	Size
0	14.02.2024 02:46:21.841	>>	Confirmed_ResponsePDU	Read	205
1	14.02.2024 02:46:22.725	<<	Confirmed_RequestPDU	Read	43
2	14.02.2024 02:46:22.769	>>	Confirmed_ResponsePDU	Read	205
3	14.02.2024 02:46:23.726	<<	Confirmed_RequestPDU	Read	43
4	14.02.2024 02:46:23.788	>>	Confirmed_ResponsePDU	Read	205
5	14.02.2024 02:46:24.725	<<	Confirmed_RequestPDU	Read	43
6	14.02.2024 02:46:24.823	>>	Confirmed_ResponsePDU	Read	205
7	14.02.2024 02:46:25.725	<<	Confirmed_RequestPDU	Read	43
8	14.02.2024 02:46:25.768	>>	Confirmed_ResponsePDU	Read	205
9	14.02.2024 02:46:26.726	<<	Confirmed_RequestPDU	Read	43
10	14.02.2024 02:46:26.748	>>	Confirmed_ResponsePDU	Read	205
11	14.02.2024 02:46:27.725	<<	Confirmed_RequestPDU	Read	43
12	14.02.2024 02:46:27.796	>>	Confirmed_ResponsePDU	Read	205
13	14.02.2024 02:46:28.727	<<	Confirmed_RequestPDU	Read	43
14	14.02.2024 02:46:28.789	>>	Confirmed_ResponsePDU	Read	205
15	14.02.2024 02:46:29.724	<<	Confirmed_RequestPDU	Read	43
16	14.02.2024 02:46:29.743	>>	Confirmed_ResponsePDU	Read	205
17	14.02.2024 02:46:30.725	<<	Confirmed_RequestPDU	Read	43
18	14.02.2024 02:46:30.752	>>	Confirmed_ResponsePDU	Read	205
19	14.02.2024 02:46:31.726	<<	Confirmed_RequestPDU	Read	43
20	14.02.2024 02:46:31.758	>>	Confirmed_ResponsePDU	Read	205
21	14.02.2024 02:46:32.723	<<	Confirmed_RequestPDU	Read	43
22	14.02.2024 02:46:32.814	>>	Confirmed_ResponsePDU	Read	205
23	14.02.2024 02:46:33.727	<<	Confirmed_RequestPDU	Read	43
24	14.02.2024 02:46:33.753	>>	Confirmed_ResponsePDU	Read	205
25	14.02.2024 02:46:34.725	<<	Confirmed_RequestPDU	Read	43
26	14.02.2024 02:46:34.788	>>	Confirmed_ResponsePDU	Read	205
27	14.02.2024 02:46:35.728	<<	Confirmed_RequestPDU	Read	43
28	14.02.2024 02:46:35.776	>>	Confirmed_ResponsePDU	Read	205
29	14.02.2024 02:46:36.726	<<	Confirmed_RequestPDU	Read	43
30	14.02.2024 02:46:36.789	>>	Confirmed_ResponsePDU	Read	205
31	14.02.2024 02:46:37.724	<<	Confirmed_RequestPDU	Read	43
32	14.02.2024 02:46:37.747	>>	Confirmed_ResponsePDU	Read	205
33	14.02.2024 02:46:38.726	<<	Confirmed_RequestPDU	Read	43
34	14.02.2024 02:46:38.747	>>	Confirmed_ResponsePDU	Read	205
35	14.02.2024 02:46:39.727	<<	Confirmed_RequestPDU	Read	43
36	14.02.2024 02:46:39.766	>>	Confirmed_ResponsePDU	Read	205
37	14.02.2024 02:46:40.724	<<	Confirmed_RequestPDU	Read	43
38	14.02.2024 02:46:40.786	>>	Confirmed_ResponsePDU	Read	205
39	14.02.2024 02:46:41.726	<<	Confirmed_RequestPDU	Read	43
40	14.02.2024 02:46:41.802	>>	Confirmed_ResponsePDU	Read	205
41	14.02.2024 02:46:42.726	<<	Confirmed_RequestPDU	Read	43
42	14.02.2024 02:46:42.764	>>	Confirmed_ResponsePDU	Read	205
43	14.02.2024 02:46:43.728	<<	Confirmed_RequestPDU	Read	43
44	14.02.2024 02:46:43.749	>>	Confirmed_ResponsePDU	Read	205
45	14.02.2024 02:46:44.726	<<	Confirmed_RequestPDU	Read	43

Fonte: Autor, 2024.

Figura 38 - Capturas do ensaio 1

Packet Nr.	Time	Dir	MMS Pdu	MMS Service	Size
0	6.02.2024 12:33:04.501	<<	Confirmed_RequestPDU	Read	472
1	6.02.2024 12:33:04.703	>>	Confirmed_ResponsePDU	Read	1586
2	6.02.2024 12:33:05.502	<<	Confirmed_RequestPDU	Read	472
3	6.02.2024 12:33:05.572	>>	Confirmed_ResponsePDU	Read	1586
4	6.02.2024 12:33:06.498	<<	Confirmed_RequestPDU	Read	472
5	6.02.2024 12:33:06.566	>>	Confirmed_ResponsePDU	Read	1586
6	6.02.2024 12:33:07.498	<<	Confirmed_RequestPDU	Read	472
7	6.02.2024 12:33:07.575	>>	Confirmed_ResponsePDU	Read	1586
8	6.02.2024 12:33:08.499	<<	Confirmed_RequestPDU	Read	472
9	6.02.2024 12:33:08.625	>>	Confirmed_ResponsePDU	Read	1586
10	6.02.2024 12:33:09.500	<<	Confirmed_RequestPDU	Read	472
11	6.02.2024 12:33:09.725	>>	Confirmed_ResponsePDU	Read	1586
12	6.02.2024 12:33:10.498	<<	Confirmed_RequestPDU	Read	472
13	6.02.2024 12:33:10.593	>>	Confirmed_ResponsePDU	Read	1586
14	6.02.2024 12:33:11.499	<<	Confirmed_RequestPDU	Read	472
15	6.02.2024 12:33:11.569	>>	Confirmed_ResponsePDU	Read	1586
16	6.02.2024 12:33:12.499	<<	Confirmed_RequestPDU	Read	472
17	6.02.2024 12:33:12.590	>>	Confirmed_ResponsePDU	Read	1586
18	6.02.2024 12:33:13.500	<<	Confirmed_RequestPDU	Read	472
19	6.02.2024 12:33:13.628	>>	Confirmed_ResponsePDU	Read	1586
20	6.02.2024 12:33:14.500	<<	Confirmed_RequestPDU	Read	472
21	6.02.2024 12:33:14.574	>>	Confirmed_ResponsePDU	Read	1586
22	6.02.2024 12:33:15.499	<<	Confirmed_RequestPDU	Read	472
23	6.02.2024 12:33:15.570	>>	Confirmed_ResponsePDU	Read	1586
24	6.02.2024 12:33:16.500	<<	Confirmed_RequestPDU	Read	472
25	6.02.2024 12:33:16.591	>>	Confirmed_ResponsePDU	Read	1586
26	6.02.2024 12:33:17.500	<<	Confirmed_RequestPDU	Read	472
27	6.02.2024 12:33:17.621	>>	Confirmed_ResponsePDU	Read	1586
28	6.02.2024 12:33:18.502	<<	Confirmed_RequestPDU	Read	472
29	6.02.2024 12:33:18.566	>>	Confirmed_ResponsePDU	Read	1586
30	6.02.2024 12:33:19.499	<<	Confirmed_RequestPDU	Read	472
31	6.02.2024 12:33:19.656	>>	Confirmed_ResponsePDU	Read	1586
32	6.02.2024 12:33:20.502	<<	Confirmed_RequestPDU	Read	472
33	6.02.2024 12:33:20.570	>>	Confirmed_ResponsePDU	Read	1586
34	6.02.2024 12:33:21.501	<<	Confirmed_RequestPDU	Read	472
35	6.02.2024 12:33:21.577	>>	Confirmed_ResponsePDU	Read	1586

Fonte: Autor, 2024.

Figura 39 - Capturas do ensaio 2

Packet Nr.	Time	Dir	MMS Pdu	MMS Service	Size
0	14.02.2024 03:27:45.583	<<	Confirmed_RequestPDU	Read	1327
1	14.02.2024 03:27:45.866	>>	Confirmed_ResponsePDU	Read	23713
2	14.02.2024 03:27:50.586	<<	Confirmed_RequestPDU	Read	1327
3	14.02.2024 03:27:50.921	>>	Confirmed_ResponsePDU	Read	23713
4	14.02.2024 03:27:55.583	<<	Confirmed_RequestPDU	Read	1327
5	14.02.2024 03:27:55.998	>>	Confirmed_ResponsePDU	Read	23713
6	14.02.2024 03:28:00.585	<<	Confirmed_RequestPDU	Read	1327
7	14.02.2024 03:28:00.879	>>	Confirmed_ResponsePDU	Read	23713
8	14.02.2024 03:28:05.583	<<	Confirmed_RequestPDU	Read	1327
9	14.02.2024 03:28:05.967	>>	Confirmed_ResponsePDU	Read	23713
10	14.02.2024 03:28:10.586	<<	Confirmed_RequestPDU	Read	1327
11	14.02.2024 03:28:11.164	>>	Confirmed_ResponsePDU	Read	23713
12	14.02.2024 03:28:14.630	<<	Confirmed_RequestPDU	Read	1327
13	14.02.2024 03:28:14.891	>>	Confirmed_ResponsePDU	Read	23713
14	14.02.2024 03:28:15.631	<<	Confirmed_RequestPDU	Read	1327
15	14.02.2024 03:28:16.003	>>	Confirmed_ResponsePDU	Read	23713
16	14.02.2024 03:28:16.631	<<	Confirmed_RequestPDU	Read	1327
17	14.02.2024 03:28:16.994	>>	Confirmed_ResponsePDU	Read	23713
18	14.02.2024 03:28:17.632	<<	Confirmed_RequestPDU	Read	1327
19	14.02.2024 03:28:18.075	>>	Confirmed_ResponsePDU	Read	23713
20	14.02.2024 03:28:18.632	<<	Confirmed_RequestPDU	Read	1327
21	14.02.2024 03:28:19.183	>>	Confirmed_ResponsePDU	Read	23713
22	14.02.2024 03:28:19.634	<<	Confirmed_RequestPDU	Read	1327
23	14.02.2024 03:28:19.984	>>	Confirmed_ResponsePDU	Read	23713
24	14.02.2024 03:28:20.630	<<	Confirmed_RequestPDU	Read	1327
25	14.02.2024 03:28:20.945	>>	Confirmed_ResponsePDU	Read	23713
26	14.02.2024 03:28:21.631	<<	Confirmed_RequestPDU	Read	1327
27	14.02.2024 03:28:21.955	>>	Confirmed_ResponsePDU	Read	23713
28	14.02.2024 03:28:22.631	<<	Confirmed_RequestPDU	Read	1327
29	14.02.2024 03:28:22.997	>>	Confirmed_ResponsePDU	Read	23713
30	14.02.2024 03:28:23.632	<<	Confirmed_RequestPDU	Read	1327
31	14.02.2024 03:28:23.992	>>	Confirmed_ResponsePDU	Read	23713
32	14.02.2024 03:28:24.634	<<	Confirmed_RequestPDU	Read	1327
33	14.02.2024 03:28:25.049	>>	Confirmed_ResponsePDU	Read	23713
34	14.02.2024 03:28:25.632	<<	Confirmed_RequestPDU	Read	1327
35	14.02.2024 03:28:25.947	>>	Confirmed_ResponsePDU	Read	23713
36	14.02.2024 03:28:26.632	<<	Confirmed_RequestPDU	Read	1327
37	14.02.2024 03:28:27.061	>>	Confirmed_ResponsePDU	Read	23713
38	14.02.2024 03:28:27.631	<<	Confirmed_RequestPDU	Read	1327
39	14.02.2024 03:28:28.026	>>	Confirmed_ResponsePDU	Read	23713
40	14.02.2024 03:28:28.631	<<	Confirmed_RequestPDU	Read	1327
41	14.02.2024 03:28:28.927	>>	Confirmed_ResponsePDU	Read	23713
42	14.02.2024 03:28:29.631	<<	Confirmed_RequestPDU	Read	1327
43	14.02.2024 03:28:29.943	>>	Confirmed_ResponsePDU	Read	23713
44	14.02.2024 03:28:30.631	<<	Confirmed_RequestPDU	Read	1327
45	14.02.2024 03:28:31.302	>>	Confirmed_ResponsePDU	Read	23713
46	14.02.2024 03:28:31.632	<<	Confirmed_RequestPDU	Read	1327
47	14.02.2024 03:28:31.932	>>	Confirmed_ResponsePDU	Read	23713
48	14.02.2024 03:28:32.633	<<	Confirmed_RequestPDU	Read	1327

Fonte: Autor, 2024.

Figura 40 - Capturas do ensaio 3

Packet Nr.	Time	Dir	MMS Pdu	MMS Service	Size
0	14.02.2024 04:08:51.836	>>	Unconfirmed_PDU	InformationReport	871
1	14.02.2024 04:08:51.930	>>	Unconfirmed_PDU	InformationReport	871
2	14.02.2024 04:08:52.056	>>	Unconfirmed_PDU	InformationReport	871
3	14.02.2024 04:08:52.165	>>	Unconfirmed_PDU	InformationReport	871
4	14.02.2024 04:08:52.291	>>	Unconfirmed_PDU	InformationReport	871
5	14.02.2024 04:08:52.417	>>	Unconfirmed_PDU	InformationReport	871
6	14.02.2024 04:08:52.541	>>	Unconfirmed_PDU	InformationReport	871
7	14.02.2024 04:08:52.648	>>	Unconfirmed_PDU	InformationReport	871
8	14.02.2024 04:08:52.772	>>	Unconfirmed_PDU	InformationReport	871
9	14.02.2024 04:08:52.880	>>	Unconfirmed_PDU	InformationReport	871
10	14.02.2024 04:08:52.991	>>	Unconfirmed_PDU	InformationReport	871
11	14.02.2024 04:08:53.115	>>	Unconfirmed_PDU	InformationReport	871
12	14.02.2024 04:08:53.224	>>	Unconfirmed_PDU	InformationReport	871
13	14.02.2024 04:08:53.348	>>	Unconfirmed_PDU	InformationReport	871
14	14.02.2024 04:08:53.456	>>	Unconfirmed_PDU	InformationReport	871
15	14.02.2024 04:08:53.580	>>	Unconfirmed_PDU	InformationReport	871
16	14.02.2024 04:08:53.704	>>	Unconfirmed_PDU	InformationReport	871
17	14.02.2024 04:08:53.829	>>	Unconfirmed_PDU	InformationReport	871
18	14.02.2024 04:08:53.936	>>	Unconfirmed_PDU	InformationReport	871
19	14.02.2024 04:08:54.060	>>	Unconfirmed_PDU	InformationReport	871
20	14.02.2024 04:08:54.183	>>	Unconfirmed_PDU	InformationReport	871
21	14.02.2024 04:08:54.277	>>	Unconfirmed_PDU	InformationReport	871
22	14.02.2024 04:08:54.402	>>	Unconfirmed_PDU	InformationReport	871
23	14.02.2024 04:08:54.525	>>	Unconfirmed_PDU	InformationReport	871
24	14.02.2024 04:08:54.649	>>	Unconfirmed_PDU	InformationReport	871
25	14.02.2024 04:08:54.756	>>	Unconfirmed_PDU	InformationReport	871
26	14.02.2024 04:08:54.865	>>	Unconfirmed_PDU	InformationReport	871
27	14.02.2024 04:08:55.006	>>	Unconfirmed_PDU	InformationReport	871
28	14.02.2024 04:08:55.098	>>	Unconfirmed_PDU	InformationReport	871
29	14.02.2024 04:08:55.221	>>	Unconfirmed_PDU	InformationReport	871
30	14.02.2024 04:08:55.345	>>	Unconfirmed_PDU	InformationReport	871
31	14.02.2024 04:08:55.471	>>	Unconfirmed_PDU	InformationReport	871
32	14.02.2024 04:08:55.593	>>	Unconfirmed_PDU	InformationReport	871
33	14.02.2024 04:08:55.700	>>	Unconfirmed_PDU	InformationReport	871
34	14.02.2024 04:08:55.810	>>	Unconfirmed_PDU	InformationReport	871
35	14.02.2024 04:08:55.934	>>	Unconfirmed_PDU	InformationReport	871
36	14.02.2024 04:08:56.058	>>	Unconfirmed_PDU	InformationReport	871
37	14.02.2024 04:08:56.167	>>	Unconfirmed_PDU	InformationReport	871
38	14.02.2024 04:08:56.290	>>	Unconfirmed_PDU	InformationReport	871
39	14.02.2024 04:08:56.397	>>	Unconfirmed_PDU	InformationReport	871
40	14.02.2024 04:08:56.507	>>	Unconfirmed_PDU	InformationReport	871
41	14.02.2024 04:08:56.631	>>	Unconfirmed_PDU	InformationReport	871
42	14.02.2024 04:08:56.787	>>	Unconfirmed_PDU	InformationReport	871
43	14.02.2024 04:08:56.865	>>	Unconfirmed_PDU	InformationReport	871
44	14.02.2024 04:08:56.990	>>	Unconfirmed_PDU	InformationReport	871
45	14.02.2024 04:08:57.116	>>	Unconfirmed_PDU	InformationReport	871
46	14.02.2024 04:08:57.209	>>	Unconfirmed_PDU	InformationReport	871
47	14.02.2024 04:08:57.334	>>	Unconfirmed_PDU	InformationReport	871
48	14.02.2024 04:08:57.441	>>	Unconfirmed_PDU	InformationReport	871
49	14.02.2024 04:08:57.564	>>	Unconfirmed_PDU	InformationReport	871
50	14.02.2024 04:08:57.673	>>	Unconfirmed_PDU	InformationReport	871

Fonte: Autor, 2024.

Figura 41 - Capturas do ensaio 4

Packet Nr.	Time	Dir	MMS Pdu	MMS Service	Size
0	21.02.2024 07:17:02.600	<<	Confirmed_RequestPDU	Read	471
1	21.02.2024 07:17:02.658	>>	Confirmed_ResponsePDU	Read	1585
2	21.02.2024 07:17:03.570	<<	Confirmed_RequestPDU	Read	471
3	21.02.2024 07:17:03.622	>>	Confirmed_ResponsePDU	Read	1585
4	21.02.2024 07:17:04.568	<<	Confirmed_RequestPDU	Read	471
5	21.02.2024 07:17:04.603	>>	Confirmed_ResponsePDU	Read	1585
6	21.02.2024 07:17:05.573	<<	Confirmed_RequestPDU	Read	471
7	21.02.2024 07:17:05.633	>>	Confirmed_ResponsePDU	Read	1585
8	21.02.2024 07:17:06.571	<<	Confirmed_RequestPDU	Read	471
9	21.02.2024 07:17:06.612	>>	Confirmed_ResponsePDU	Read	1585
10	21.02.2024 07:17:07.577	<<	Confirmed_RequestPDU	Read	471
11	21.02.2024 07:17:07.661	>>	Confirmed_ResponsePDU	Read	1585
12	21.02.2024 07:17:08.573	<<	Confirmed_RequestPDU	Read	471
13	21.02.2024 07:17:08.614	>>	Confirmed_ResponsePDU	Read	1585
14	21.02.2024 07:17:09.644	<<	Confirmed_RequestPDU	Read	471
15	21.02.2024 07:17:09.671	>>	Confirmed_ResponsePDU	Read	1585
16	21.02.2024 07:17:10.571	<<	Confirmed_RequestPDU	Read	471
17	21.02.2024 07:17:10.635	>>	Confirmed_ResponsePDU	Read	1585
18	21.02.2024 07:17:11.571	<<	Confirmed_RequestPDU	Read	471
19	21.02.2024 07:17:11.621	>>	Confirmed_ResponsePDU	Read	1585
20	21.02.2024 07:17:12.568	<<	Confirmed_RequestPDU	Read	471
21	21.02.2024 07:17:12.603	>>	Confirmed_ResponsePDU	Read	1585
22	21.02.2024 07:17:13.572	<<	Confirmed_RequestPDU	Read	471
23	21.02.2024 07:17:13.618	>>	Confirmed_ResponsePDU	Read	1585
24	21.02.2024 07:17:14.583	<<	Confirmed_RequestPDU	Read	471
25	21.02.2024 07:17:14.628	>>	Confirmed_ResponsePDU	Read	1585
26	21.02.2024 07:17:15.630	<<	Confirmed_RequestPDU	Read	471
27	21.02.2024 07:17:15.664	>>	Confirmed_ResponsePDU	Read	1585
28	21.02.2024 07:17:17.467	<<	Confirmed_RequestPDU	Read	471
29	21.02.2024 07:17:17.501	>>	Confirmed_ResponsePDU	Read	1585
30	21.02.2024 07:17:17.725	<<	Confirmed_RequestPDU	Read	471
31	21.02.2024 07:17:17.893	>>	Confirmed_ResponsePDU	Read	1585
32	21.02.2024 07:17:18.573	<<	Confirmed_RequestPDU	Read	471

Fonte: Autor, 2024.

Figura 42 - Capturas do ensaio 5

Packet Nr.	Time	Dir	MMS Pdu	MMS Service	Size
0	21.02.2024 09:36:14.428	<<	Confirmed_RequestPDU	Read	1327
1	21.02.2024 09:36:14.518	>>	Confirmed_ResponsePDU	Read	23712
2	21.02.2024 09:36:15.427	<<	Confirmed_RequestPDU	Read	1327
3	21.02.2024 09:36:15.557	>>	Confirmed_ResponsePDU	Read	23712
4	21.02.2024 09:36:16.434	<<	Confirmed_RequestPDU	Read	1327
5	21.02.2024 09:36:16.591	>>	Confirmed_ResponsePDU	Read	23712
6	21.02.2024 09:36:17.432	<<	Confirmed_RequestPDU	Read	1327
7	21.02.2024 09:36:17.541	>>	Confirmed_ResponsePDU	Read	23712
8	21.02.2024 09:36:18.428	<<	Confirmed_RequestPDU	Read	1327
9	21.02.2024 09:36:18.540	>>	Confirmed_ResponsePDU	Read	23712
10	21.02.2024 09:36:19.432	<<	Confirmed_RequestPDU	Read	1327
11	21.02.2024 09:36:19.552	>>	Confirmed_ResponsePDU	Read	23712
12	21.02.2024 09:36:20.432	<<	Confirmed_RequestPDU	Read	1327
13	21.02.2024 09:36:20.616	>>	Confirmed_ResponsePDU	Read	23712
14	21.02.2024 09:36:21.433	<<	Confirmed_RequestPDU	Read	1327
15	21.02.2024 09:36:21.538	>>	Confirmed_ResponsePDU	Read	23712
16	21.02.2024 09:36:22.428	<<	Confirmed_RequestPDU	Read	1327
17	21.02.2024 09:36:22.529	>>	Confirmed_ResponsePDU	Read	23712
18	21.02.2024 09:36:23.431	<<	Confirmed_RequestPDU	Read	1327
19	21.02.2024 09:36:23.589	>>	Confirmed_ResponsePDU	Read	23712
20	21.02.2024 09:36:24.431	<<	Confirmed_RequestPDU	Read	1327
21	21.02.2024 09:36:24.542	>>	Confirmed_ResponsePDU	Read	23712
22	21.02.2024 09:36:25.429	<<	Confirmed_RequestPDU	Read	1327
23	21.02.2024 09:36:25.532	>>	Confirmed_ResponsePDU	Read	23712
24	21.02.2024 09:36:26.429	<<	Confirmed_RequestPDU	Read	1327
25	21.02.2024 09:36:26.525	>>	Confirmed_ResponsePDU	Read	23712
26	21.02.2024 09:36:27.428	<<	Confirmed_RequestPDU	Read	1327
27	21.02.2024 09:36:27.542	>>	Confirmed_ResponsePDU	Read	23712
28	21.02.2024 09:36:28.429	<<	Confirmed_RequestPDU	Read	1327
29	21.02.2024 09:36:28.562	>>	Confirmed_ResponsePDU	Read	23712
30	21.02.2024 09:36:29.431	<<	Confirmed_RequestPDU	Read	1327
31	21.02.2024 09:36:29.557	>>	Confirmed_ResponsePDU	Read	23712
32	21.02.2024 09:36:30.433	<<	Confirmed_RequestPDU	Read	1327
33	21.02.2024 09:36:30.564	>>	Confirmed_ResponsePDU	Read	23712
34	21.02.2024 09:36:31.437	<<	Confirmed_RequestPDU	Read	1327
35	21.02.2024 09:36:31.854	>>	Confirmed_ResponsePDU	Read	23712
36	21.02.2024 09:36:32.434	<<	Confirmed_RequestPDU	Read	1327
37	21.02.2024 09:36:32.626	>>	Confirmed_ResponsePDU	Read	23712
38	21.02.2024 09:36:33.430	<<	Confirmed_RequestPDU	Read	1327
39	21.02.2024 09:36:33.530	>>	Confirmed_ResponsePDU	Read	23712
40	21.02.2024 09:36:34.430	<<	Confirmed_RequestPDU	Read	1327
41	21.02.2024 09:36:34.523	>>	Confirmed_ResponsePDU	Read	23712
42	21.02.2024 09:36:35.429	<<	Confirmed_RequestPDU	Read	1327
43	21.02.2024 09:36:35.528	>>	Confirmed_ResponsePDU	Read	23712
44	21.02.2024 09:36:36.428	<<	Confirmed_RequestPDU	Read	1327
45	21.02.2024 09:36:36.534	>>	Confirmed_ResponsePDU	Read	23712
46	21.02.2024 09:36:37.429	<<	Confirmed_RequestPDU	Read	1327
47	21.02.2024 09:36:37.544	>>	Confirmed_ResponsePDU	Read	23712
48	21.02.2024 09:36:38.440	<<	Confirmed_RequestPDU	Read	1327
49	21.02.2024 09:36:38.587	>>	Confirmed_ResponsePDU	Read	23712
50	21.02.2024 09:36:39.430	<<	Confirmed_RequestPDU	Read	1327

Fonte: Autor, 2024.

Figura 43 - Capturas do ensaio 6

Packet Nr.	Time	Dir	MMS Pdu	MMS Service	Size
0	21.02.2024 08:54:57.669	>>	Unconfirmed_PDU	InformationReport	871
1	21.02.2024 08:54:57.809	>>	Unconfirmed_PDU	InformationReport	871
2	21.02.2024 08:54:57.840	>>	Unconfirmed_PDU	InformationReport	871
3	21.02.2024 08:54:57.935	>>	Unconfirmed_PDU	InformationReport	871
4	21.02.2024 08:54:58.028	>>	Unconfirmed_PDU	InformationReport	871
5	21.02.2024 08:54:58.138	>>	Unconfirmed_PDU	InformationReport	871
6	21.02.2024 08:54:58.232	>>	Unconfirmed_PDU	InformationReport	871
7	21.02.2024 08:54:58.339	>>	Unconfirmed_PDU	InformationReport	871
8	21.02.2024 08:54:58.431	>>	Unconfirmed_PDU	InformationReport	871
9	21.02.2024 08:54:58.541	>>	Unconfirmed_PDU	InformationReport	871
10	21.02.2024 08:54:58.633	>>	Unconfirmed_PDU	InformationReport	871
11	21.02.2024 08:54:58.742	>>	Unconfirmed_PDU	InformationReport	871
12	21.02.2024 08:54:58.835	>>	Unconfirmed_PDU	InformationReport	871
13	21.02.2024 08:54:58.941	>>	Unconfirmed_PDU	InformationReport	871
14	21.02.2024 08:54:59.036	>>	Unconfirmed_PDU	InformationReport	871
15	21.02.2024 08:54:59.145	>>	Unconfirmed_PDU	InformationReport	871
16	21.02.2024 08:54:59.251	>>	Unconfirmed_PDU	InformationReport	871
17	21.02.2024 08:54:59.343	>>	Unconfirmed_PDU	InformationReport	871
18	21.02.2024 08:54:59.435	>>	Unconfirmed_PDU	InformationReport	871
19	21.02.2024 08:54:59.543	>>	Unconfirmed_PDU	InformationReport	871
20	21.02.2024 08:54:59.651	>>	Unconfirmed_PDU	InformationReport	871
21	21.02.2024 08:54:59.745	>>	Unconfirmed_PDU	InformationReport	871
22	21.02.2024 08:54:59.854	>>	Unconfirmed_PDU	InformationReport	871
23	21.02.2024 08:54:59.976	>>	Unconfirmed_PDU	InformationReport	871
24	21.02.2024 08:55:00.070	>>	Unconfirmed_PDU	InformationReport	871
25	21.02.2024 08:55:00.146	>>	Unconfirmed_PDU	InformationReport	871
26	21.02.2024 08:55:00.256	>>	Unconfirmed_PDU	InformationReport	871
27	21.02.2024 08:55:00.394	>>	Unconfirmed_PDU	InformationReport	871
28	21.02.2024 08:55:00.564	>>	Unconfirmed_PDU	InformationReport	871
29	21.02.2024 08:55:00.694	>>	Unconfirmed_PDU	InformationReport	871
30	21.02.2024 08:55:00.815	>>	Unconfirmed_PDU	InformationReport	871
31	21.02.2024 08:55:00.861	>>	Unconfirmed_PDU	InformationReport	871
32	21.02.2024 08:55:00.939	>>	Unconfirmed_PDU	InformationReport	871
33	21.02.2024 08:55:01.031	>>	Unconfirmed_PDU	InformationReport	871
34	21.02.2024 08:55:01.140	>>	Unconfirmed_PDU	InformationReport	871
35	21.02.2024 08:55:01.248	>>	Unconfirmed_PDU	InformationReport	871
36	21.02.2024 08:55:01.326	>>	Unconfirmed_PDU	InformationReport	871
37	21.02.2024 08:55:01.482	>>	Unconfirmed_PDU	InformationReport	871
38	21.02.2024 08:55:01.589	>>	Unconfirmed_PDU	InformationReport	871
39	21.02.2024 08:55:01.685	>>	Unconfirmed_PDU	InformationReport	871
40	21.02.2024 08:55:01.773	>>	Unconfirmed_PDU	InformationReport	871
41	21.02.2024 08:55:01.870	>>	Unconfirmed_PDU	InformationReport	871
42	21.02.2024 08:55:01.954	>>	Unconfirmed_PDU	InformationReport	871
43	21.02.2024 08:55:02.040	>>	Unconfirmed_PDU	InformationReport	871
44	21.02.2024 08:55:02.179	>>	Unconfirmed_PDU	InformationReport	871
45	21.02.2024 08:55:02.285	>>	Unconfirmed_PDU	InformationReport	871
46	21.02.2024 08:55:02.383	>>	Unconfirmed_PDU	InformationReport	871
47	21.02.2024 08:55:02.485	>>	Unconfirmed_PDU	InformationReport	871

Fonte: Autor, 2024.

Uma gravação sobre a coleta de dados pode ser encontrada no seguinte link.

O fork relacionado a este projeto é o https://github.com/Ruanfc/libiec61850-1