

Rodrigo Faria Argondizo

Desenvolvimento de um Gateway Modbus TCP/DNP3 para Aplicações em Sistemas de Energia

Uberlândia, MG

2025

Rodrigo Faria Argondizo

Desenvolvimento de um Gateway Modbus TCP/DNP3 para Aplicações em Sistemas de Energia

Trabalho de Conclusão de Curso da Engenharia de Controle e Automação da Universidade Federal de Uberlândia - UFU - Campus Santa Mônica, como requisito para a obtenção do título de Graduação em Engenharia de Controle e Automação.

Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Elétrica - FEELT

Orientador Prof. Dr. Renato Ferreira Fernandes Júnior

Uberlândia, MG

2025

Agradecimentos

Em primeiro lugar, agradeço a Deus, que me concedeu a dádiva da existência.

Aos meus pais, expresso minha eterna gratidão por todo o apoio incondicional, por sempre fazerem o possível para que eu chegasse até aqui e não desistisse no caminho.

À minha namorada, Mariana Calixto, dedico meu sincero reconhecimento pelo incentivo constante e por estar ao meu lado nos momentos mais desafiadores.

Aos meus avós, agradeço por sempre me inspirarem e torcerem por mim ao longo da graduação.

À minha família e amigos, deixo meu muito obrigado pelos inúmeros incentivos que tornaram possível a realização desta conquista.

À toda a equipe de professores da Universidade Federal de Uberlândia (UFU), cujo comprometimento com o ensino, dedicação e excelência acadêmica foram fundamentais para minha formação.

Ao meu orientador, Professor Renato Ferreira Fernandes, sou grato pelo apoio fundamental durante a elaboração deste trabalho. Sua disponibilidade, orientação e contribuições sempre enriquecedoras foram essenciais para o desenvolvimento deste projeto.

Por fim, agradeço a toda a equipe de Engenharia de Operações e Manutenção da (re)energisa, pelo compartilhamento diário de conhecimentos e experiências tão valiosas. Um reconhecimento especial a Guilherme Couto Damasceno, Paulo Vicktor Felix Dias e Rafael Alexandre Silva de Carvalho, cuja colaboração e apoio foram fundamentais para o enriquecimento da minha jornada profissional.

*“Eu sei o que tenho que fazer agora, tenho que continuar respirando porque amanhã o sol
vai nascer. Quem sabe o que a maré pode trazer?”*
(Náufrago)

Resumo

A crescente adoção da geração distribuída por meio de usinas solares fotovoltaicas impulsionou a demanda por soluções de automação industrial. Esses sistemas exigiram a integração eficiente e confiável de diferentes dispositivos de campo. Nesse contexto, os padrões de comunicação Modbus TCP e DNP3 destacam-se como tecnologias essenciais para sistemas SCADA, garantindo interoperabilidade e monitoramento centralizado. Este trabalho apresentou o desenvolvimento de um gateway embarcado entre Modbus TCP e DNP3, implementado em C++. A solução teve como objetivo facilitar a integração de equipamentos típicos de plantas solares a sistemas supervisórios, reduzindo custos com licenciamento de softwares de mercado e aumentando a eficiência operacional. O gateway foi implantado em uma Raspberry Pi, atuando como concentrador de dados e convertendo mensagens Modbus TCP, simuladas por um dispositivo ESP8266, para o padrão DNP3. Testes com equipamentos reais de usinas fotovoltaicas foram realizados para validar a solução em condições práticas. Após a análise de desempenho, concluiu-se que o gateway operou conforme o esperado, oferecendo uma alternativa acessível, confiável e de fácil implementação, com potencial para contribuir significativamente para a supervisão remota em tempo real no setor de energia solar.

Palavras-chave: automação industrial; energia solar fotovoltaica; geração distribuída; gateway embarcado; SCADA.

Abstract

The growing adoption of distributed generation through photovoltaic solar power plants has driven the demand for industrial automation solutions. These systems required the efficient and reliable integration of various field devices. In this context, the Modbus TCP and DNP3 communication standards stand out as essential technologies for SCADA systems, ensuring interoperability and centralized monitoring. This work presented the development of an embedded gateway between Modbus TCP and DNP3, implemented in C++. The solution aimed to facilitate the integration of typical solar plant equipment into supervisory systems, reducing costs with commercial software licenses and improving operational efficiency. The gateway was deployed on a Raspberry Pi, acting as a data concentrator and converting Modbus TCP messages, simulated by an ESP8266 device, to the DNP3 standard. Tests with real equipment from photovoltaic power plants were conducted to validate the solution under practical conditions. After performance analysis, it was concluded that the gateway operated as expected, offering an affordable, reliable, and easy-to-implement alternative, with the potential to significantly contribute to real-time remote supervision in the solar energy sector.

Keywords: distributed generation; embedded gateway; industrial automation; photovoltaic solar energy; SCADA.

Lista de ilustrações

Figura 1 – Usina fotovoltaica.	15
Figura 2 – Centro de operações com a utilização de um sistema SCADA.	17
Figura 3 – Tela de um supervisório desenvolvido em Elipse Power.	18
Figura 4 – Fluxograma de mensagens no protocolo Modbus TCP.	19
Figura 5 – Arquitetura de rede Modbus TCP.	19
Figura 6 – Estrutura pergunta e resposta Modbus na rede TCP/IP.	20
Figura 7 – Cabeçalho de aplicação do protocolo Modbus.	20
Figura 8 – Tabelas de funções do protocolo Modbus.	21
Figura 9 – Campo de dados das mensagens no protocolo Modbus.	21
Figura 10 – Relação mestre-remota protocolo DNP3.	22
Figura 11 – Arquiteturas usuais no protocolo DNP3.	23
Figura 12 – Camadas do protocolo DNP3.	24
Figura 13 – Composição do DNP3 e seu cabeçalho.	24
Figura 14 – Tipos de objetos protocolo DNP3.	26
Figura 15 – Comando do tipo Direct Operate.	27
Figura 16 – Comando do tipo Select Before Operate.	27
Figura 17 – Desmonstração do funcionamento de um Gateway.	28
Figura 18 – ESP8266.	29
Figura 19 – Pinagem ESP8266+.	29
Figura 20 – Raspberry Pi 3 Model B+.	30
Figura 21 – Visão geral Raspberry Pi 3 Model B+.	30
Figura 22 – Pinagem de uma Raspberry Pi 3 Model B+.	31
Figura 23 – Datasheet do SmartLogger3000.	32
Figura 24 – Arquitetura atual de mercado.	33
Figura 25 – Arquitetura proposta.	34
Figura 26 – Topologia de rede com Raspberry Pi e ESP8266 conectados via rede local.	36
Figura 27 – Esquema de ligação elétrica do ESP8266.	37
Figura 28 – Montagem física.	37
Figura 29 – Fluxograma do algoritmo ESP8266.	38
Figura 30 – Callback de comandos ESP8266.	39
Figura 31 – Software utilizado para validação dos endereços Modbus.	40
Figura 32 – Software para instalação do Sistema Operacional em placas Raspberry.	41
Figura 33 – Fluxograma principal de funcionamento do programa embarcado.	43
Figura 34 – Fluxograma da função de leitura Modbus.	44
Figura 35 – Fluxograma da função de Reconexão Modbus.	45

Figura 36 – Fluxograma do Manipulador de Comandos.	46
Figura 37 – Topologia de rede com Raspberry Pi e os equipamentos fotovoltaicos conectados via rede local.	49
Figura 38 – Fluxograma principal do Algoritmo.	50
Figura 39 – Fluxograma da Thread de Polling Modbus detalhada.	51
Figura 40 – Topologia utilizando Elipse SCADA como mestre.	55
Figura 41 – Pontos DNP3 cadastrados no driver do Elipse SCADA.	55
Figura 42 – Tela de testes no SCADA.	56
Figura 43 – Topologia utilizando simulador DNP3 como mestre.	57
Figura 44 – Pontos DNP3 cadastrados no simulador da Kepware.	57
Figura 45 – Pontos DNP3 cadastrados no simulador da Kepware.	58
Figura 46 – Pontos DNP3 dos equipamentos fotovoltaicos cadastrados no driver do Elipse SCADA.	58
Figura 47 – Tela de testes no SCADA com equipamentos fotovoltaicos.	59
Figura 48 – Tráfego Modbus TCP entre Raspberry Pi e ESP8266.	62
Figura 49 – Tráfego DNP3 entre Raspberry Pi e Elipse SCADA.	62
Figura 50 – Terminal reportando a comunicação em tempo real.	63
Figura 51 – Terminal reportando falha de comunicação com o escravo Modbus TCP.	64
Figura 52 – Tela do Elipse SCADA com pontos DNP3 ativos.	65
Figura 53 – Tela do simulador Kepware com pontos DNP3 ativos.	65
Figura 54 – Monitoramento dos valores lidos do gateway no ambiente do Kepware.	66
Figura 55 – Recebimento de comandos no terminal da Raspberry Pi.	66
Figura 56 – Recebimento do comando Direct Operate pelo mestre DNP3.	67
Figura 57 – Execução do comando via escrita em coil Modbus TCP.	67
Figura 58 – Captura Wireshark: Leitura periódica Modbus TCP de dispositivos reais.	69
Figura 59 – Captura Wireshark: Troca de dados entre o SCADA Elipse e o gateway.	69
Figura 60 – Interface SCADA exibindo dados DNP3 de dispositivos reais.	70
Figura 61 – Tratamento de falha: Substituição do valor Modbus após falhas consecutivas.	71

Lista de tabelas

Tabela 1	–	Classes de Objetos no Protocolo DNP3	25
Tabela 2	–	Especificações técnicas da Raspberry Pi 3 Model B+	31
Tabela 3	–	Endereçamento Modbus TCP configurado no ESP8266	40
Tabela 4	–	Endereçamento Modbus TCP configurados na Raspberry Pi	47
Tabela 5	–	Endereçamento DNP3 configurado na Raspberry Pi	48
Tabela 6	–	Dispositivos da Rede Industrial	49
Tabela 7	–	Endereçamento dos dados Modbus TCP monitorados	52
Tabela 8	–	Endereçamento dos pontos DNP3 configurados	53
Tabela 9	–	Dispositivos reais integrados para testes	68

Lista de abreviaturas e siglas

CA	Corrente Alternada
CC	Corrente Contínua
COI	Centro de Operação Integrado
CROB	Control Relay Output Block
DNP3	Distributed Network Protocol – Protocolo de Rede Distribuída versão 3
GC	Geração Centralizada
GD	Geração Distribuída
IHM	Interface Homem-Máquina
IP	Internet Protocol – Protocolo de Internet
IOT	Internet of Things – Internet das Coisas
LED	Light Emitting Diode – Diodo emissor de luz
MBAP	Modbus Application Protocol Header – Cabeçalho de aplicação do protocolo Modbus
SCADA	Supervisory Control and Data Acquisition – Sistemas de Supervisão e Aquisição de Dados
SO	Sistema Operacional
SSH	Secure Shell
TCP	Transmission Control Protocol – Protocolo de Controle de Transmissão
UFV	Usina Fotovoltaica

Lista de Códigos

3.1	Comandos para instalação das dependências no terminal Linux	41
3.2	Comandos para instalação das dependências no terminal Linux	41
3.3	Comandos para instalação das dependências no terminal Linux	42
3.4	Conversão de dois registradores Modbus de 16 bits em um valor de 32 bits	53
3.5	Comando para instalação do tcpdump no terminal Linux	60
A.1	Escravo Modbus TCP implementado com uma ESP8266	77
B.1	Implementação 1 - Estação Remota DNP3 integrada a estação de testes com ESP8266	80
C.1	Implementação 2 - Estação Remota DNP3 integrada a dispositivos reais . .	87
D.1	CMakeLists para compilação dos projetos	93

Sumário

1	INTRODUÇÃO	13
1.1	Justificativas	14
1.2	Objetivos	14
2	REFERENCIAIS TEÓRICOS	15
2.1	Geração distribuída	15
2.1.1	Usinas solares fotovoltaicas	15
2.1.2	Automação em usinas solares fotovoltaicas	16
2.2	SCADA	17
2.3	Protocolos industriais de rede	18
2.3.1	Modbus TCP	18
2.3.2	DNP3	21
2.3.2.1	Classes de objetos no protocolo DNP3	25
2.3.2.2	Objetos e suas variações no protocolo DNP3	25
2.3.2.3	Modos de comando no protocolo DNP3	26
2.4	Gateways de comunicação	27
2.5	Sistemas embarcados	28
2.5.1	ESP8266	28
2.5.2	Raspberry Pi	30
3	METODOLOGIA	32
3.1	Entendendo a necessidade do gateway em UFVs	32
3.2	Arquitetura geral	33
3.2.1	Cenário atual	33
3.2.2	Solução proposta	34
3.3	Gateway Modbus TCP/DNP3	34
3.3.1	Integração da estação remota DNP3 com o dispositivo simulado no ESP8266	35
3.3.1.1	ESP8266	36
3.3.1.2	Raspberry Pi	40
3.3.2	Estação remota DNP3 integrada com dispositivos reais	48
3.4	Observações gerais de ambas as implementações	54
3.5	Configuração do mestre DNP3 para validar o gateway	54
3.5.1	Implementação 1	54
3.5.1.1	Mestre DNP3 utilizando Elipse SCADA	54
3.5.1.2	Mestre DNP3 utilizando simulador DNP3	56

3.5.2	Implementação 2	58
3.6	Captura da rede para análise dos resultados	59
3.6.1	Ferramentas utilizadas	59
3.6.2	Metodologia de captura	60
4	RESULTADOS E DISCUSSÕES	61
4.1	Resultados do ambiente simulado	61
4.1.1	Resultados com o mestre Elipse SCADA	61
4.1.1.1	Análise do tráfego de rede	61
4.1.1.2	Comunicação em tempo real	63
4.1.2	Resultados com o mestre Kepware	65
4.1.3	Resultados dos comandos Direct Operate	66
4.1.4	Considerações finais da etapa simulada	67
4.2	Resultados com dispositivos reais de UFVs	68
4.2.1	Análise do tráfego	68
4.2.2	Apresentação dos dados em interface SCADA	70
4.2.3	Estabilidade e tratamento de falhas	70
4.2.4	Conclusão da validação com dispositivos reais	71
5	CONCLUSÃO	72
	REFERÊNCIAS	73
	ANEXO A – ANEXOS	77
	ANEXO B – ANEXOS	80
	ANEXO C – ANEXOS	87
	ANEXO D – ANEXOS	93

1 Introdução

Com o crescimento acelerado da busca por fontes de energia limpa e renovável, a geração de energia fotovoltaica tem se consolidado como uma alternativa viável e promissora no cenário energético mundial. No Brasil, o avanço desse setor tem sido significativo. De acordo com a Associação Brasileira de Geração Distribuída ([ABGD, 2024](#)): “impulsionada pelo ‘boom’ da geração fotovoltaica, a geração distribuída cresceu mais de 26 gigawatts nos últimos 4 anos”.

À medida que a implantação de usinas fotovoltaicas aumenta, surgem também desafios relacionados ao monitoramento, controle e gerenciamento eficiente dessas plantas, que geralmente são compostas por múltiplos equipamentos de diferentes fabricantes. Nesse contexto, os sistemas de automação e supervisão tornam-se indispensáveis para garantir a operação segura e eficiente, permitindo que operadores acompanhem o desempenho dos ativos em tempo real e tomem decisões com base em dados confiáveis.

Soluções como sistemas SCADA (Supervisory Control And Data Acquisition) e interfaces homem-máquina (IHMs) têm papel fundamental nesse cenário, pois possibilitam a centralização das informações coletadas em campo e a execução de comandos remotos. No entanto, um dos principais desafios enfrentados em ambientes de geração distribuída, como usinas fotovoltaicas, está na interoperabilidade entre diferentes protocolos industriais adotados pelos dispositivos de campo, dificultando a integração eficiente desses dados em plataformas de supervisão e controle.

Entre os protocolos mais relevantes nesse tipo de aplicação destacam-se o Modbus TCP e o DNP3. O Modbus TCP é amplamente utilizado devido à sua simplicidade de implementação e à compatibilidade com uma grande variedade de equipamentos industriais. Segundo ([CALDIERI, 2016](#)): “no campo das Redes Industriais, este é talvez o protocolo de mais larga utilização”. Já o DNP3 (Distributed Network Protocol) é amplamente adotado em aplicações que exigem alta confiabilidade e robustez na comunicação, sendo comum em sistemas SCADA para o setor elétrico. Conforme apontado por ([SIGGINS, 2020](#)), “o DNP3 é um protocolo amplamente utilizado em sistemas de controle e monitoramento remoto confiável”.

Nesse cenário, surge a necessidade de integrar dispositivos que operam com protocolos distintos. É aqui que os gateways de comunicação se tornam fundamentais. Um gateway é capaz de realizar a conversão entre diferentes protocolos, possibilitando a interoperabilidade entre sistemas heterogêneos. Conforme ([MESTANZA; FERNANDES, 2022](#)), “gateways permitem a integração de dispositivos que operam em redes distintas e utilizam protocolos diferentes, ampliando a conectividade entre esses dispositivos”.

1.1 Justificativas

Na indústria, especialmente em setores críticos como energia, água e petróleo, a diversidade de equipamentos de diferentes fabricantes aumenta a complexidade no desenvolvimento de sistemas SCADA. Isso ocorre devido à despadronização dos protocolos de comunicação, o que torna o processo de implementação mais desafiador. Considerando a importância de ter um centro de operações em tempo real, com baixo tempo de detecção de falhas nos setores mencionados, a implementação de um gateway embarcado pode ser uma solução eficaz. Especificamente, um gateway Modbus TCP/DNP3 facilita a centralização dos equipamentos em um único ambiente, permitindo o controle remoto e integrado dos processos, o que é essencial para a operação eficiente de sistemas fotovoltaicos e outras infraestruturas críticas.

Além disso, o desenvolvimento independente de recursos de baixo custo pode incentivar o uso de soluções de código aberto, ampliando a difusão da conversão e adaptação de protocolos industriais. Essa abordagem contribui tanto para a inovação tecnológica, quanto para a redução de custos em sistemas de monitoramento e controle, além de fomentar o uso de protocolos confiáveis, como o DNP3, na geração distribuída, especialmente em plantas fotovoltaicas.

No contexto de geração fotovoltaica, a integração de diferentes equipamentos em uma rede eficiente de comunicação possibilita a centralização do monitoramento, tornando-o mais eficiente em termos operacionais. Portanto, este trabalho se justifica pela necessidade de simplificar a integração entre sistemas fotovoltaicos e sistemas supervisórios, garantindo qualidade e eficiência na gestão de ativos.

1.2 Objetivos

O objetivo deste trabalho é desenvolver uma solução embarcada para integrar um sistema SCADA, baseado no protocolo DNP3, a dispositivos de campo que utilizam Modbus TCP. A proposta visa otimizar o monitoramento e o controle de sistemas de energia, como usinas fotovoltaicas (UFVs). Para isso, será projetado um gateway embarcado DNP3/Modbus TCP, que atuará como uma estação remota Ethernet, permitindo a comunicação eficiente entre os dispositivos de campo e o sistema SCADA, contribuindo para a implementação de um centro de operação centralizado, eficiente e de baixo custo.

2 Referenciais Teóricos

Este tópico tem como objetivo a explanação da fundamentação científica necessária para a compreensão do desenvolvimento do trabalho, visando proporcionar uma base conceitual sólida para o entendimento dos principais conceitos abordados.

2.1 Geração distribuída

O setor elétrico global está passando por grandes transformações, isso vem sendo impulsionado por avanços tecnológicos e por mudanças regulatórias essenciais para viabilizar novos modelos de negócio. Tradicionalmente, a geração de energia elétrica tem sido centralizada, com grandes usinas situadas longe dos centros consumidores. No entanto, ganha força um novo modelo baseado na geração descentralizada, conhecido como geração distribuída, em que o próprio consumidor produz energia no local de consumo ([SANTOS; CUNHA; TORRES, 2021](#)).

2.1.1 Usinas solares fotovoltaicas

Usinas fotovoltaicas, ou centrais fotovoltaicas, são grandes sistemas projetados para gerar e comercializar energia elétrica. A produção ocorre por meio de painéis solares, que realizam a conversão direta da luz solar em eletricidade, a qual é então injetada na rede de distribuição ([Portal Solar, 2024](#)).

Figura 1 – Usina fotovoltaica.



Fonte: ([Portal Solar, 2024](#)).

As usinas solares fotovoltaicas geram energia a partir da luz solar captada por painéis, produzindo eletricidade em corrente contínua (CC). Para ser utilizada na rede

elétrica, essa energia é convertida em corrente alternada (CA) por um inversor solar, tornando-a compatível com o padrão nacional de distribuição ([Portal Solar, 2024](#)).

Em resumo, as usinas solares fotovoltaicas são uma forma prática e eficiente de geração distribuída, permitindo que a energia seja gerada próxima ao local de consumo e comercializada diretamente por empresas ou consórcios com acesso à rede elétrica.

2.1.2 Automação em usinas solares fotovoltaicas

Além da conversão de energia realizada pelos inversores, as usinas fotovoltaicas modernas contam com sistemas de automação que possibilitam o controle e o monitoramento remoto de seus principais componentes. Esses sistemas frequentemente integram inversores solares, medidores de energia, relés de proteção, sensores de irradiância solar, entre outros, todos conectados a um sistema supervisório central.

Embora muitos desses dispositivos operem de forma autônoma, a automação permite que sejam gerenciados remotamente, viabilizando ações como ligar ou desligar inversores, alterar limites operacionais e coletar dados em tempo real para análise de desempenho e diagnóstico de falhas.

As informações adquiridas por esses dispositivos são geralmente transmitidas via protocolos industriais, como o Modbus TCP, para um sistema SCADA. Através dessa estrutura supervisória, os operadores conseguem visualizar parâmetros como tensão, corrente, potência ativa e reativa, frequência e alarmes, garantindo atualizações constantes e supervisão eficaz da planta.

Um exemplo prático é o sistema utilizado pela empresa Clemar Engenharia, que implementou a plataforma Elipse E3 para monitorar e controlar o desempenho de suas usinas solares. Com o auxílio do SCADA, a empresa realiza a leitura de dados de inversores, controladores e sensores ambientais, além de comandar dispositivos remotamente, garantindo maior eficiência operacional e redução de custos com manutenções presenciais ([ELIPSE, 2024](#)).

Outro exemplo relevante é a solução implementada pelo Grupo MTR em parceria com a Ineer Energia, voltada para o monitoramento de 132 MW de usinas solares. O sistema, com suporte ao protocolo Modbus TCP/IP, coleta dados operacionais, gera alertas de falhas em tempo real e realiza análises preditivas, reduzindo significativamente o tempo de inatividade e contribuindo para a otimização da geração fotovoltaica distribuída ([Grupo MTR, 2024](#)).

2.2 SCADA

A sigla SCADA - Supervisory Control and Data Acquisition, em português Sistemas de Supervisão e Aquisição de Dados trata-se de um software que permite o monitoramento e o rastreamento de uma ou mais informações de um processo produtivo ou planta física (CONSTAIN, 2011).

Em um centro de operações, o SCADA desempenha um papel crucial, pois, permite o controle e a operação de ativos de forma centralizada. Possibilita que os utilizadores consigam realizar mudanças em variáveis, chaves, monitorar alarmes e consolidar informações de uma planta em apenas um local (URZÊDA, 2006).

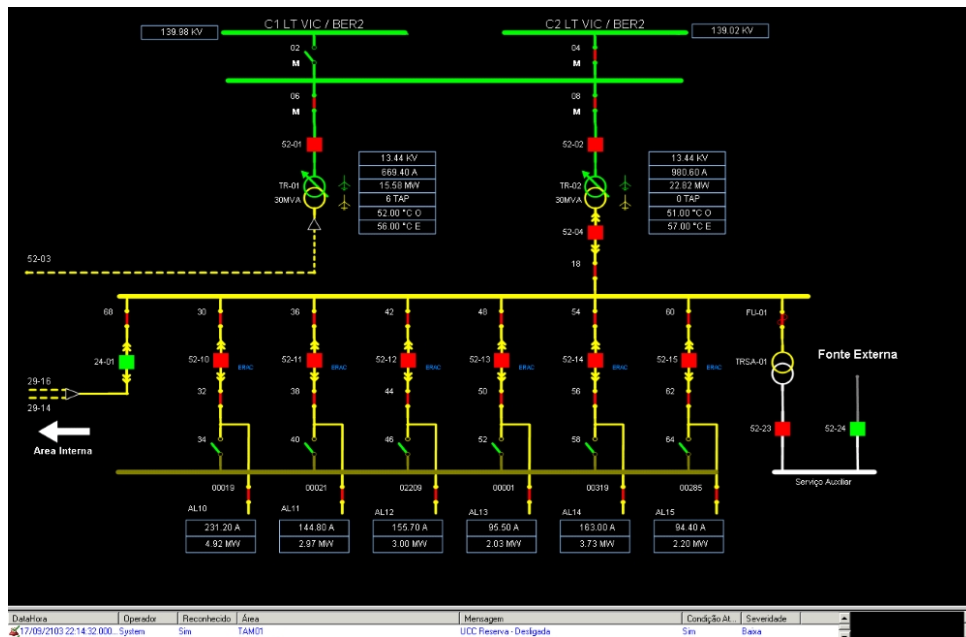
A Figura 2 e 3 mostram exemplos de sistemas supervisórios e seu ambiente de utilização, os denominados COIs.

Figura 2 – Centro de operações com a utilização de um sistema SCADA.



Fonte: (Eclipse Software, 2024).

Figura 3 – Tela de um supervisor desenvolvido em Elipse Power.



Fonte: (Elipse Software, 2024).

Em suma, um sistema SCADA possibilita a integração entre homem e máquina, proporcionando aos operadores acesso completo às informações do processo, além de permitir a execução de comandos como paradas de emergência e acionamentos de equipamentos de campo quando necessário (TIBURSKI; MOREIRA; MISAGHI, 2017).

2.3 Protocolos industriais de rede

Protocolos são meios essenciais para a troca de informações entre diferentes dispositivos e sistemas em ambientes industriais. No presente trabalho, serão utilizados dois protocolos amplamente empregados na indústria, o Modbus TCP e o DNP3, que têm aplicação significativa no setor energético e em sistemas relacionados a SCADA. Esses protocolos serão detalhados nos próximos subtópicos.

2.3.1 Modbus TCP

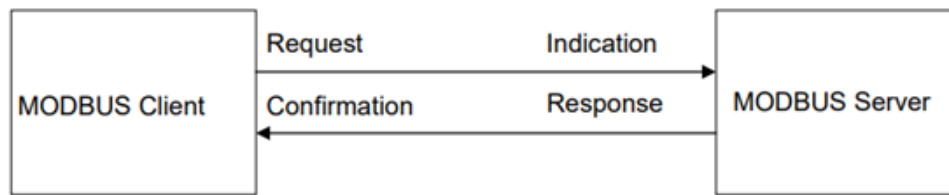
O Modbus TCP/IP é uma evolução do protocolo Modbus, projetada para operar em redes que utilizam o TCP/IP, um padrão confiável amplamente adotado na Internet. A queda nos custos do Ethernet, aliada à sua popularidade, favoreceu sua aplicação em ambientes industriais, viabilizando a integração direta com sistemas corporativos (Modbus Organization, 2024).

Introduzido em 1999, o Modbus TCP/IP une a flexibilidade do Ethernet ao padrão universal do TCP/IP, oferecendo uma solução aberta e eficiente para a troca de dados em

processos industriais, com fácil implementação em dispositivos que suportam o modelo TCP/IP ([Modbus Organization, 2024](#)).

O protocolo Modbus TCP funciona basicamente na utilização de uma técnica mestre-escravo ou, comumente chamada de cliente-servidor. Seu funcionamento consiste no modo que apenas um dispositivo, o mestre/cliente pode iniciar transações. Sendo assim os outros dispositivos, os escravos/servidores respondem fornecendo os dados solicitados ao mestre ([BANERJI, 2013](#)).

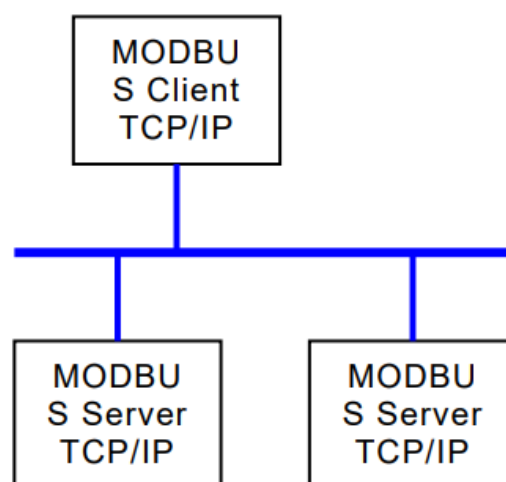
Figura 4 – Fluxograma de mensagens no protocolo Modbus TCP.



Fonte: ([Modbus Organization, 2006](#)).

Um sistema de comunicação que utiliza o protocolo MODBUS TCP/IP inclui dispositivos Cliente e Servidor conectados diretamente a uma rede TCP/IP para realizar a comunicação, onde se tem a porta 502 reservada para conexões Modbus. Além disso, a rede pode contar com outros dispositivos de interconexão, como bridges, roteadores ou gateways, que operam exclusivamente na rede TCP/IP. Essa arquitetura garante flexibilidade para integrar com as redes modernas ([Modbus Organization, 2006](#)).

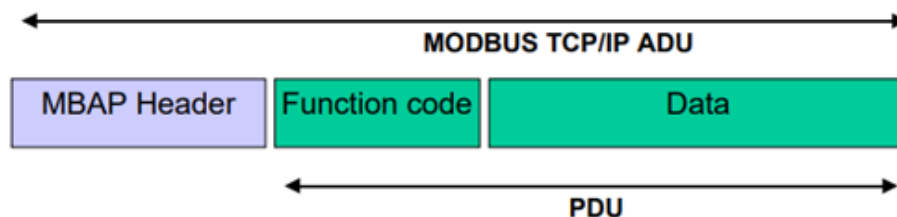
Figura 5 – Arquitetura de rede Modbus TCP.



Fonte: Adaptado de ([Modbus Organization, 2006](#)).

O Modbus TCP, possui uma estrutura definida para organizar e transmitir dados em uma rede TCP/IP. Essa estrutura pode ser visualizada na Figura 6.

Figura 6 – Estrutura pergunta e resposta Modbus na rede TCP/IP.



Fonte: (Modbus Organization, 2006).

No caso do Modbus TCP, diferentemente de sua outra versão, o Modbus RTU, a estrutura de dados é composta por um cabeçalho denominado MBAP. Segundo (Modbus Organization, 2006): “Esse cabeçalho possui quatro campos fundamentais: o Identificador de Transação, que vincula solicitações e respostas, sendo replicado pelo servidor; o Identificador de Protocolo, responsável pela multiplexação intra-sistema, com o valor 0 reservado para o protocolo Modbus; o Comprimento, que especifica a quantidade de bytes dos campos subsequentes, incluindo o Unit ID e Data; e o Identificador de Unidade, utilizado no roteamento intra-sistema, especialmente em comunicações entre redes Ethernet TCP/IP e linhas seriais Modbus, sendo configurado pelo cliente e retornado inalterado pelo servidor na resposta”.

Figura 7 – Cabeçalho de aplicação do protocolo Modbus.

Fields	Length	Description -	Client	Server
Transaction Identifier	2 Bytes	Identification of a MODBUS Request / Response transaction.	Initialized by the client	Recopied by the server from the received request
Protocol Identifier	2 Bytes	0 = MODBUS protocol	Initialized by the client	Recopied by the server from the received request
Length	2 Bytes	Number of following bytes	Initialized by the client (request)	Initialized by the server (Response)
Unit Identifier	1 Byte	Identification of a remote slave connected on a serial line or on other buses.	Initialized by the client	Recopied by the server from the received request

Fonte: (Modbus Organization, 2006).

Além do cabeçalho MBAP, a estrutura do Modbus TCP também inclui dois componentes essenciais para a comunicação: o Function Code e o Data Field. O Function Code é um campo de 1 byte que define a operação a ser executada pelo servidor, que pode ser visualizada na Figura 8. Esse código indica ao servidor o tipo de ação que deve ser realizada e, dependendo do valor, a resposta do servidor também será ajustada para refletir a operação solicitada (MODBUS Organization, 2012).

Figura 8 – Tabelas de funções do protocolo Modbus.

Code	1/16-bit	Description	I/O Range
01	1-bit	Read coils	00001 – 10000
02	1-bit	Read contacts	10001 – 20000
05	1-bit	Write a single coil	00001 – 10000
15	1-bit	Write multiple coils	00001 – 10000
03	16-bit	Read holding registers	40001 – 50000
04	16-bit	Read input registers	30001 – 40000
06	16-bit	Write single register	40001 – 50000
16	16-bit	Write multiple registers	40001 – 50000
22	16-bit	Mask write register	40001 – 50000
23	16-bit	Read/write multiple registers	40001 – 50000
24	16-bit	Read FIFO queue	40001 – 50000

Fonte: (THOMAS, 2008).

Já o Data Field, que varia em tamanho de acordo com a operação solicitada, contém os dados específicos da solicitação ou resposta. Esses dados podem incluir informações como valores de registros ou status de dispositivos, sendo fundamentais para a realização da comunicação entre os dispositivos. Ambos os campos são determinantes para garantir que os dados sejam corretamente processados e retornados pelo servidor ao cliente (MODBUS Organization, 2012).

Figura 9 – Campo de dados das mensagens no protocolo Modbus.

Start	1	2	3	4	5	6	7	Parity	Stop
-------	---	---	---	---	---	---	---	--------	------

Start	1	2	3	4	5	6	7	Stop	Stop
-------	---	---	---	---	---	---	---	------	------

Start	1	2	3	4	5	6	7	8	Parity	Stop
-------	---	---	---	---	---	---	---	---	--------	------

Start	1	2	3	4	5	6	7	8	Stop	Stop
-------	---	---	---	---	---	---	---	---	------	------

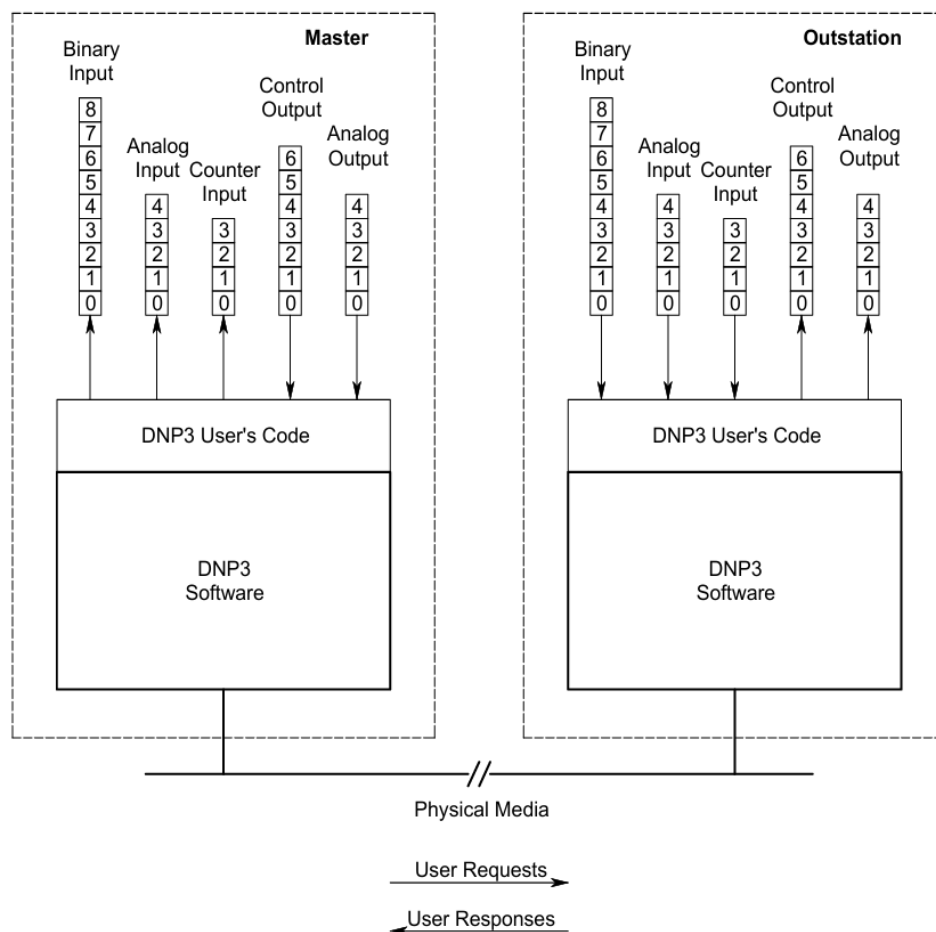
Fonte: (THOMAS, 2008).

2.3.2 DNP3

De acordo com o (DNP Users Group, 2024), o DNP3 (Distributed Network Protocol) é um protocolo de comunicação amplamente utilizado em sistemas de automação, especialmente em redes de energia elétrica e outras indústrias críticas. Desenvolvido inicialmente pela Harris Corporation em 1993, o DNP3 foi projetado para superar as limitações de outros protocolos da época, oferecendo maior confiabilidade, segurança e flexibilidade, especialmente em ambientes industriais complexos e distribuídos. O DNP3 utiliza uma arquitetura mestre-escravo (ou cliente-servidor), semelhante ao protocolo Modbus, mas com

capacidades estendidas, como o suporte a múltiplas mensagens simultâneas, a comunicação bidirecional eficiente e a gestão de eventos em tempo real. O DNP3 foi desenvolvido com o objetivo de aprimorar a transmissão de dados e comandos de controle entre computadores. Ao contrário dos protocolos de uso geral, como os utilizados na Internet para envio de e-mails, documentos, consultas SQL, multimídia e arquivos grandes, o DNP3 é especificamente voltado para aplicações SCADA (Supervisory Control and Data Acquisition) (DNP Users Group, 2005).

Figura 10 – Relação mestre-remota protocolo DNP3.

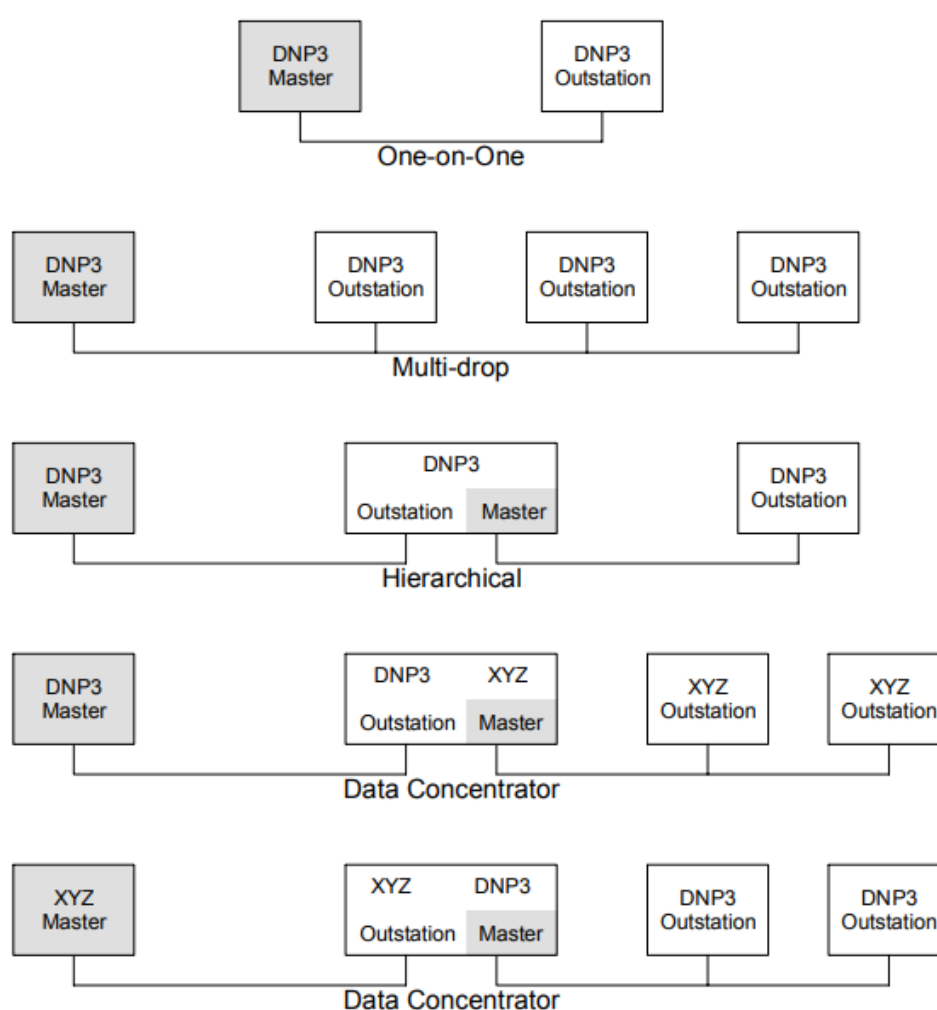


Fonte: (DNP Users Group, 2005).

Na estação remota, os dados são armazenados em blocos organizados como arrays, representando diferentes tipos de informações. Os arrays de entradas binárias indicam os estados de dispositivos booleanos físicos ou lógicos, enquanto os de entradas analógicas representam quantidades medidas ou calculadas pela estação. Há também arrays de contadores, que representam valores de contagem, como quilowatt-hora, que aumentam até um valor máximo e reiniciam. As saídas de controle são organizadas em arrays de pontos de controle, como ligar/desligar ou aumentar/diminuir, e as saídas analógicas representam valores usados para ajustes, como setpoints. Cada elemento dentro desses arrays é numerado de 0 a N-1, com N sendo o número de blocos de dados e os índices

sendo baseados em zero (DNP Users Group, 2005). O mestre DNP3 possui um banco de dados similar para armazenar dados de entrada (binário, analógico e contador). O mestre usa essas informações para monitorar o sistema, realizar controle, notificar alarmes e outras funções. Para manter seu banco de dados atualizado, o mestre envia solicitações à estação remota, pedindo os dados mais recentes, em um processo chamado polling. A estação remota responde transmitindo seu banco de dados para o mestre. Em sistemas mais avançados, a estação remota pode também enviar respostas sem ser solicitada (DNP Users Group, 2005).

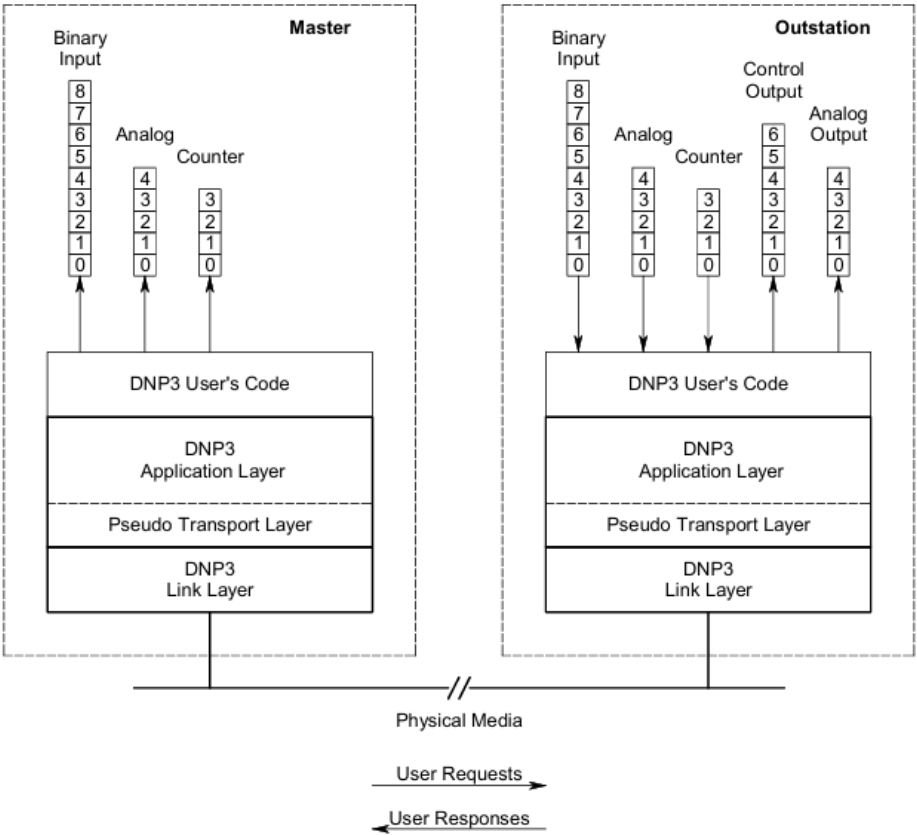
Figura 11 – Arquiteturas usuais no protocolo DNP3.



Fonte: (DNP Users Group, 2005).

Segundo (DNP Users Group, 2005), o DNP3 utiliza uma arquitetura em camadas, que estrutura de forma eficiente o envio de dados e comandos. Essa estrutura pode ser visualizada na Figura 12.

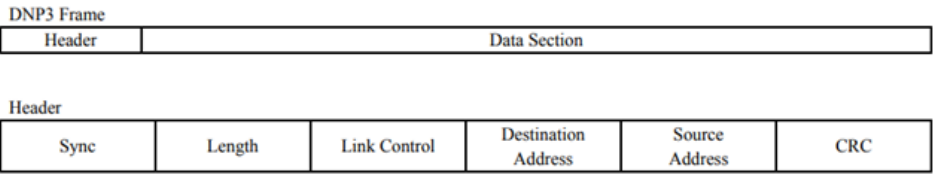
Figura 12 – Camadas do protocolo DNP3.



Fonte: (DNP Users Group, 2005).

A camada de enlace de dados é responsável por garantir a confiabilidade da comunicação na conexão física. Isso é feito por meio da detecção de erros e da identificação de quadros duplicados. Quando os dados são transmitidos, a camada de enlace envia e recebe pacotes, conhecidos como quadros no protocolo DNP3. Em algumas situações, é necessário enviar vários quadros para transmitir todas as informações de um dispositivo para outro (DNP Users Group, 2005). Um quadro DNP3 é composto por duas partes: o cabeçalho e a seção de dados. O cabeçalho contém informações sobre o tamanho do quadro, controle de enlace de dados e os endereços dos dispositivos de origem e destino. Já a seção de dados, chamada de payload, contém as informações que são transferidas entre os dispositivos, provenientes das camadas superiores do protocolo (DNP Users Group, 2005).

Figura 13 – Composição do DNP3 e seu cabeçalho.



Fonte: (DNP Users Group, 2005).

2.3.2.1 Classes de objetos no protocolo DNP3

O protocolo DNP3 organiza os dados em quatro classes de objetos, que determinam como as informações são armazenadas, monitoradas e transmitidas entre um mestre e um escravo. Essas classes são usadas para otimizar o tráfego de dados, priorizando eventos críticos e reduzindo a carga desnecessária na comunicação (Real Time Automation, Inc., 2024). Essas classes podem ser visualizadas na Tabela 1.

Tabela 1 – Classes de Objetos no Protocolo DNP3

Classe	Tipo de Dados	Prioridade	Comportamento no Polling
Classe 0	Dados estáticos	–	Retorna todos os valores atuais (Classes 0, 1, 2 e 3) quando solicitado.
Classe 1	Dados dinâmicos (eventos)	Alta	Retorna apenas eventos não reportados (mudanças de valor) da Classe 1.
Classe 2	Dados dinâmicos (eventos)	Média	Retorna apenas eventos não reportados (mudanças de valor) da Classe 2.
Classe 3	Dados dinâmicos (eventos)	Baixa	Retorna apenas eventos não reportados (mudanças de valor) da Classe 3.

Fonte: Adaptado de (Real Time Automation, Inc., 2024)

Portanto segundo (Real Time Automation, Inc., 2024), na utilização do protocolo DNP3, são utilizados quatro parâmetros distintos para configurar de forma independente os diferentes tipos de solicitações de polling. Como exemplo, em um cenário típico, as requisições de polling para as Classes 1, 2 e 3 podem ser enviadas em intervalos curtos, por exemplo a cada 10 segundos, enquanto a sondagem de integridade (referente à Classe 0) — que recupera os valores atualizados de todos os objetos (Classes 0 a 3) — é executada em períodos mais espaçados.

Como citado anteriormente, em alguns sistemas algumas outstation DNP3 podem reportar dados de eventos ao mestre DNP3, independentemente dos ciclos de polling desse mestre. Esses dados de eventos são transmitidos das estações remotas por meio da transmissão de mensagens não solicitadas (Schneider Electric, 2020).

2.3.2.2 Objetos e suas variações no protocolo DNP3

Os Grupos de Objetos no DNP3 determinam a forma como as entradas e saídas são acessadas para leitura e escrita. Em suma pode-se resumir que cada tipo de entrada ou saída possui dois grupos principais: um destinado às leituras regulares, através de varredura

e outro voltado para os eventos, enviados apenas quando ocorrem alterações. No caso das saídas binárias e analógicas, os comandos, existe ainda um grupo específico utilizado para operações de escrita ([PcVue Solutions, 2024](#)). Na Figura 14, pode-se observar os tipos de objetos.

Figura 14 – Tipos de objetos protocolo DNP3.

Type	Group	Description	Variables tree
Binary input	1	A two state value such as a simple on off switch.	Bit
Binary input (event)	2		
Binary output	10	A two state output such as the open and close control for a circuit breaker.	Bit
Binary output (event)	11		
Binary output (CROB)	12	Control Relay Output Block. The mechanism used to write a binary output.	
Running counters	20	A value that counts the number of occurrences of an event. For example the number of times a circuit breaker has tripped.	Register
Running counters (event)	22		
Frozen counters	21	A value that counts the number of occurrences of an event at a particular point in time.	Register
Frozen counters (event)	23		
Analog input	30	A single analog input such as a phase voltage.	Register
Analog input (event)	32		
Analog output	40	A single analog output such as a set-point.	Register
Analog output (event)	42		
Analog output (AOB)	41	Analog Output Block. The mechanism used to write an analog output.	
Internal indications	80	A two state value that indicate the operational status of the device.	Bit
Device attributes	0	String values representing static properties, and analog values representing configuration attributes, of the device. For example the manufacturer's name or the quantity of analog inputs.	Register or text as appropriate

Fonte: ([PcVue Solutions, 2024](#)).

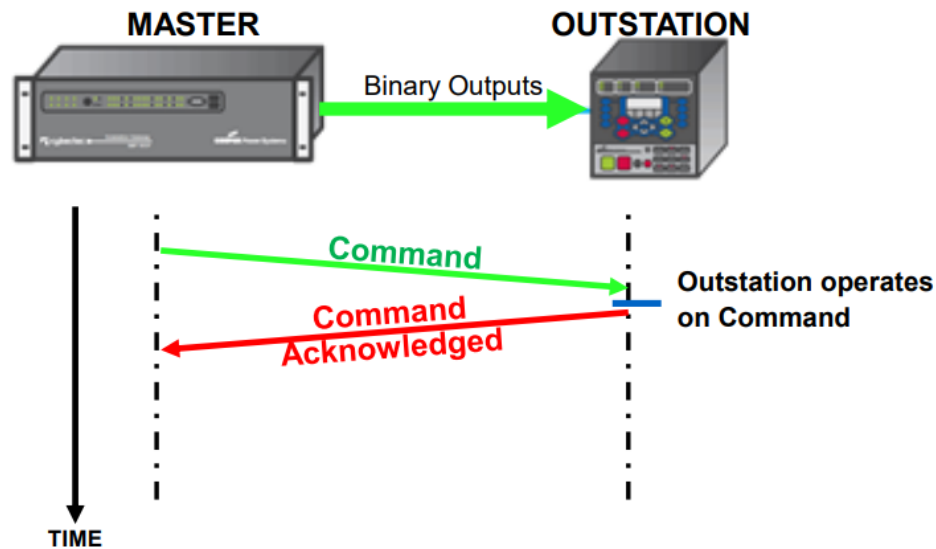
Segundo ([PcVue Solutions, 2024](#)), as variações representam os diferentes formatos em que os dados podem ser apresentados dentro de cada Grupo. Por exemplo, entradas analógicas podem ser transmitidas como um inteiro de 32 bits com sinalizador (flag), como um inteiro de 16 bits sem sinalizadores, entre outras formas.

2.3.2.3 Modos de comando no protocolo DNP3

No protocolo DNP3, os comandos de controle de dispositivos são realizados por meio da função Control Relay Output Block (CROB), que encapsula instruções como ligar, desligar ou pulsar, além de parâmetros adicionais como a duração e o número de pulsos. Em dispositivos inteligentes, como relés de proteção, esses parâmetros geralmente são ignorados, sendo considerado apenas o tipo de comando (ON ou OFF) ([ABB, 2021](#)).

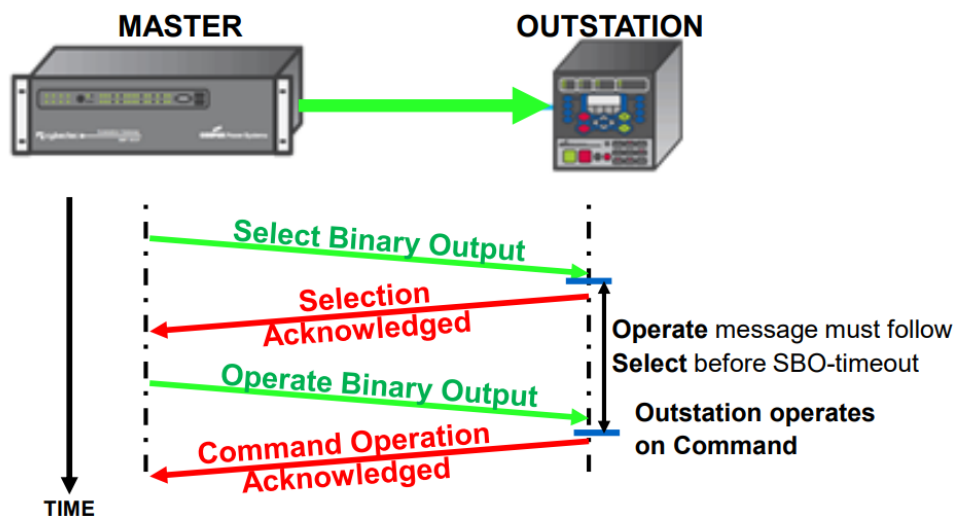
Dois modos principais de execução são suportados: Direct Operate e Select-Before-Operate. No modo Direct Operate, o comando é executado imediatamente após o envio. Já no modo Select-Before-Operate, a operação ocorre em duas etapas: primeiro o ponto é selecionado, e em seguida o comando é confirmado, o que aumenta a segurança ao evitar operações acidentais ([ABB, 2021](#)).

Figura 15 – Comando do tipo Direct Operate.



Fonte: (GROUP, 2015).

Figura 16 – Comando do tipo Select Before Operate.



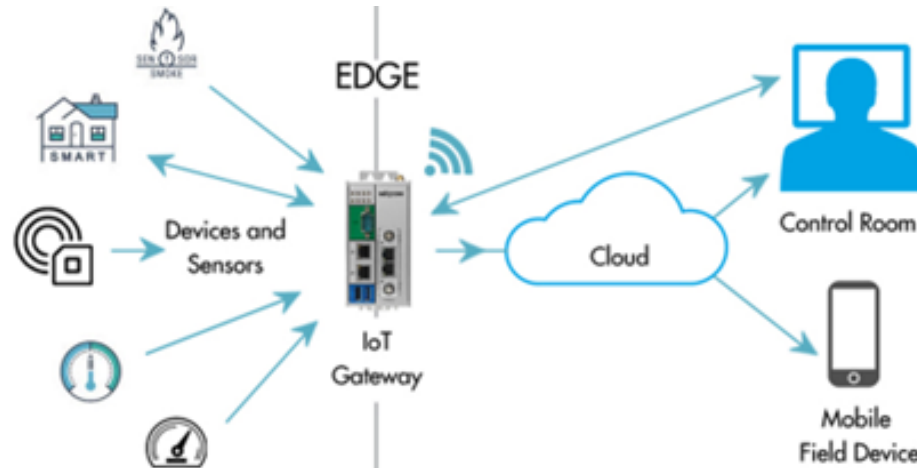
Fonte: (GROUP, 2015).

2.4 Gateways de comunicação

Um gateway industrial é um dispositivo instalado no chão de fábrica, próximo aos equipamentos de produção e sensores, com a função principal de coletar e encaminhar dados. Ele se conecta a diferentes fontes de dados na borda e pode utilizar diferentes protocolos industriais (Dell Technologies, 2024). De maneira geral, um gateway, é um dispositivo responsável por permitir a comunicação entre redes que utilizam protocolos distintos. Para isso, ele realiza a conversão dos formatos dos dados recebidos, adaptando-os ao protocolo de destino. No contexto da Internet das Coisas (IoT), o gateway desempenha

o papel de intermediar a comunicação entre dispositivos, sendo capaz de suportar diferentes protocolos de comunicação e formatos de dados (MACIAS et al., 2021).

Figura 17 – Desmonstração do funcionamento de um Gateway.



Fonte: (Ezenroute, 2024).

2.5 Sistemas embarcados

Sistemas embarcados são aqueles que realizam o processamento de dados dentro de sistemas ou produtos mais complexos, com a finalidade de executar tarefas específicas. Esses sistemas são geralmente desenvolvidos para cumprir uma ou mais funções pré-definidas, sem a necessidade de programação pelo usuário final. Eles frequentemente interagem com o ambiente ao seu redor, podendo coletar dados de diversos sensores, além de poder acionar atuadores. (VARGAS, 2007).

Segundo (RODRIGUES; PEDÓ; TEDESCO, 2013), “Os sistemas embarcados se mostram presentes em diversas áreas da sociedade, com tendência de cada vez mais aumentar sua presença no dia-a-dia das pessoas, através dos celulares, smartphones e tablets, por exemplo. Além disto, pode-se observar também a presença de sistemas embarcados nas máquinas de lavar roupas, nos micro-ondas, ou nos sistemas de automóveis, por exemplo.”

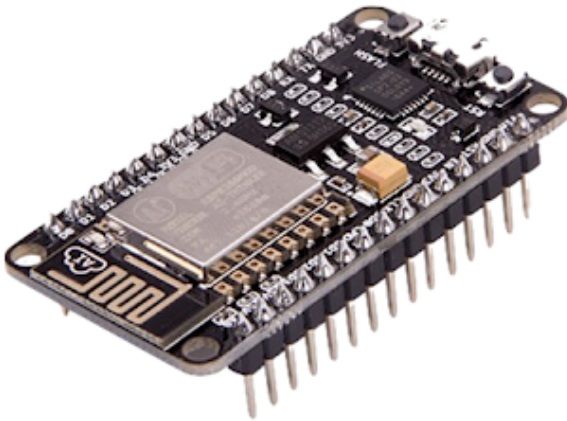
2.5.1 ESP8266

O ESP8266 é um microcontrolador de baixo custo e alto desempenho, conhecido principalmente por sua capacidade de conexão Wi-Fi integrada. Desenvolvido pela Espressif Systems, ele se tornou popular por permitir a conexão de dispositivos à Internet de forma acessível, impulsionando projetos de IoT e automação residencial. Sua combinação de processamento eficiente e conectividade sem fio o torna ideal para aplicações que exigem comunicação em rede com baixo consumo de energia (SYSTEMS, 2024).

A integração de conectividade Wi-Fi em sistemas embarcados tem se tornado cada vez mais comum, e o ESP8266 surge como uma solução acessível e versátil para essa demanda. Com ele, é possível desenvolver desde dispositivos IoT simples, como sensores remotos e controladores inteligentes, até sistemas mais complexos, como gateways de comunicação e automação industrial. Sua facilidade de programação e compatibilidade com plataformas como o Arduino contribuíram para sua rápida adoção no mercado (SYSTEMS, 2024).

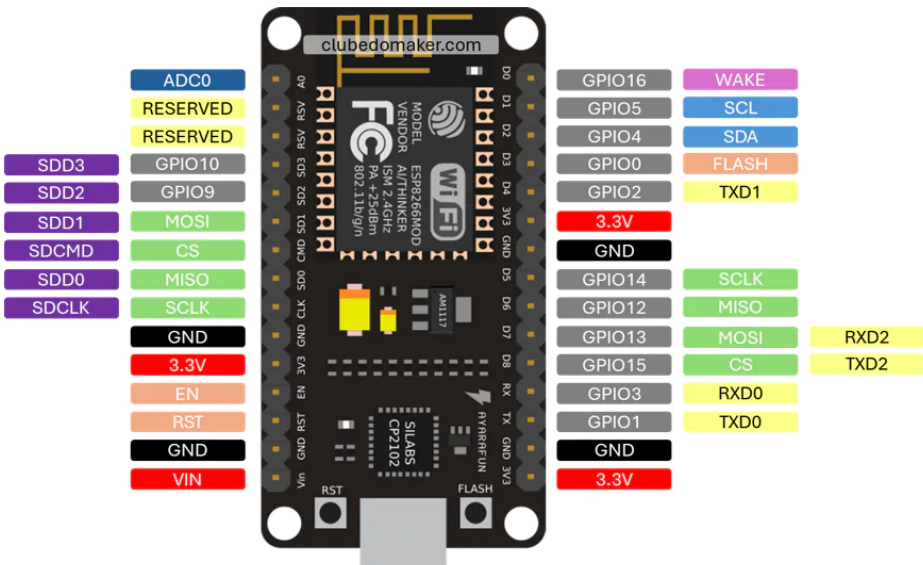
O ESP8266 e a sua pinagem podem ser visualizados na Figura 18 e 19 respectivamente.

Figura 18 – ESP8266.



Fonte: (K, 2018).

Figura 19 – Pinagem ESP8266+.



Fonte: (PEDRO, 2024).

2.5.2 Raspberry Pi

Desenvolvido pela Raspberry Pi Foundation, o Raspberry Pi 3 Model B+ é um computador de placa única de tamanho pequeno, baseado em Linux e de código aberto. (Raspberry Pi Foundation, 2019). O modelo descrito pode ser visualizado na Figura 20.

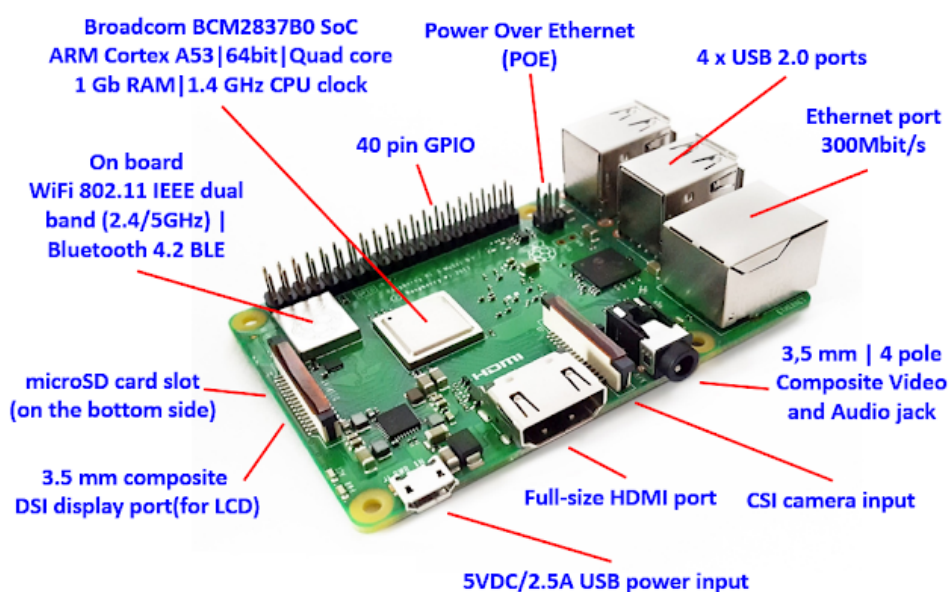
Figura 20 – Raspberry Pi 3 Model B+.



Fonte: (Raspberry Pi Foundation, 2018).

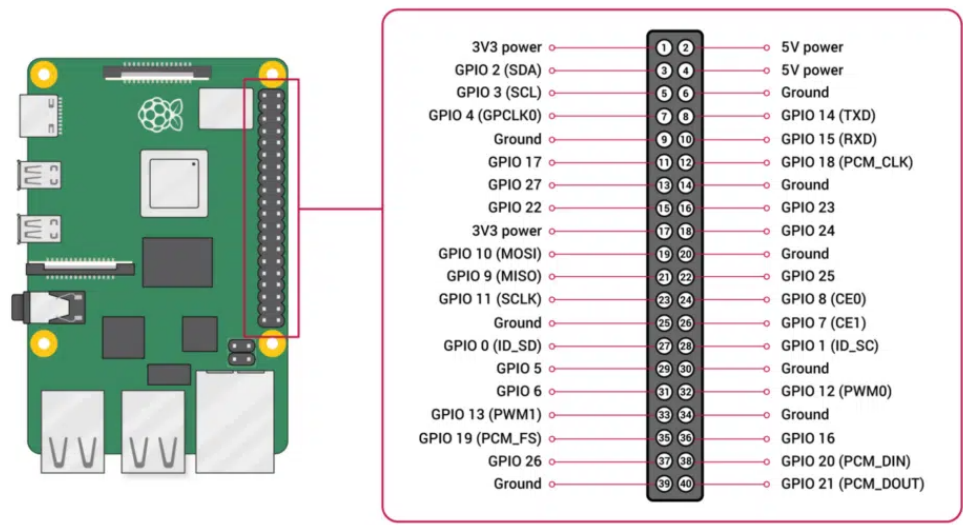
A Raspberry Pi 3 Model B+ se consolida como uma das melhores opções para desenvolvedores, estudantes e entusiastas que buscam uma solução acessível, porém robusta, para projetos de computação, automação e IoT. (MakerHero, 2024). A visão geral da placa, assim como sua pinagem podem ser observadas nas Figuras 21 e 22, respectivamente.

Figura 21 – Visão geral Raspberry Pi 3 Model B+.



Fonte: Adaptado de (Robotics University, 2018).

Figura 22 – Pinagem de uma Raspberry Pi 3 Model B+.



Fonte: ([MakerHero](#), 2024).

As especificações técnicas da Raspberry Pi 3 Model B+ podem ser visualizadas na Tabela 2.

Tabela 2 – Especificações técnicas da Raspberry Pi 3 Model B+

Categoria	Descrição
Processador	Broadcom BCM2837B0, SoC Cortex-A53 64-bit @ 1.4GHz
Memória	1GB
Conectividade	Wi-Fi 2.4 GHz e 5 GHz IEEE 802.11b/g/n/ac; Bluetooth 4.2, BLE; Ethernet Gigabit via USB 2.0 (taxa máxima de 300 Mbps); 4 portas USB 2.0
Vídeo e Áudio	1 saída HDMI full-size; Porta de display MIPI DSI; Porta de câmera MIPI CSI; Saída estéreo de 4 polos e porta de vídeo composto
Multimídia	Decodificação H.264, MPEG-4 (1080p30); Codificação H.264 (1080p30); Gráficos OpenGL ES 1.1, 2.0
Armazenamento	Cartão Micro SD (para sistema operacional e dados)
Alimentação	5V/2.5A DC via conector micro USB; 5V DC via cabeçalho GPIO; Suporte a Power over Ethernet (PoE) (requer PoE HAT separado)
Temperatura de Operação	0–50°C

Fonte: Adaptado de ([Raspberry Pi Foundation](#), 2018)

3 Metodologia

Ao longo do desenvolvimento deste projeto, foram realizadas diversas etapas, desde a análise do problema e a identificação da necessidade de um gateway de comunicação, até o desenvolvimento final e os testes da solução proposta. Cada uma dessas etapas será detalhada nos subtópicos a seguir, com o objetivo de descrever o processo metodológico adotado de forma clara e estruturada.

3.1 Entendendo a necessidade do gateway em UFVs

Como brevemente abordado nos capítulos anteriores, o desenvolvimento de um gateway de comunicação torna-se essencial para a integração dos diferentes dispositivos de campo presentes em uma usina fotovoltaica (UFV). A maioria desses dispositivos, como inversores, medidores de energia e controladores, oferece suporte ao protocolo industrial Modbus TCP, enquanto é raro encontrar equipamentos compatíveis com o protocolo DNP3, frequentemente utilizado em sistemas SCADA para o setor elétrico.

Essa incompatibilidade dificulta a centralização das informações, uma vez que o SCADA de um centro de operações integrado (COI) comumente exige comunicação via DNP3. A Figura 23 apresenta um exemplo real desse cenário, destacando o SmartLogger3000, da fabricante Huawei, amplamente utilizado em UFVs de geração distribuída, mas que não oferece suporte nativo ao protocolo DNP3.

Figura 23 – Datasheet do SmartLogger3000.

Communication Protocol	
Ethernet	Modbus-TCP, IEC 60870-5-104
RS485	Modbus-RTU, IEC 60870-5-103 (standard), DL / T645

Fonte: Adaptado de ([HUAWEI, 2025](#)).

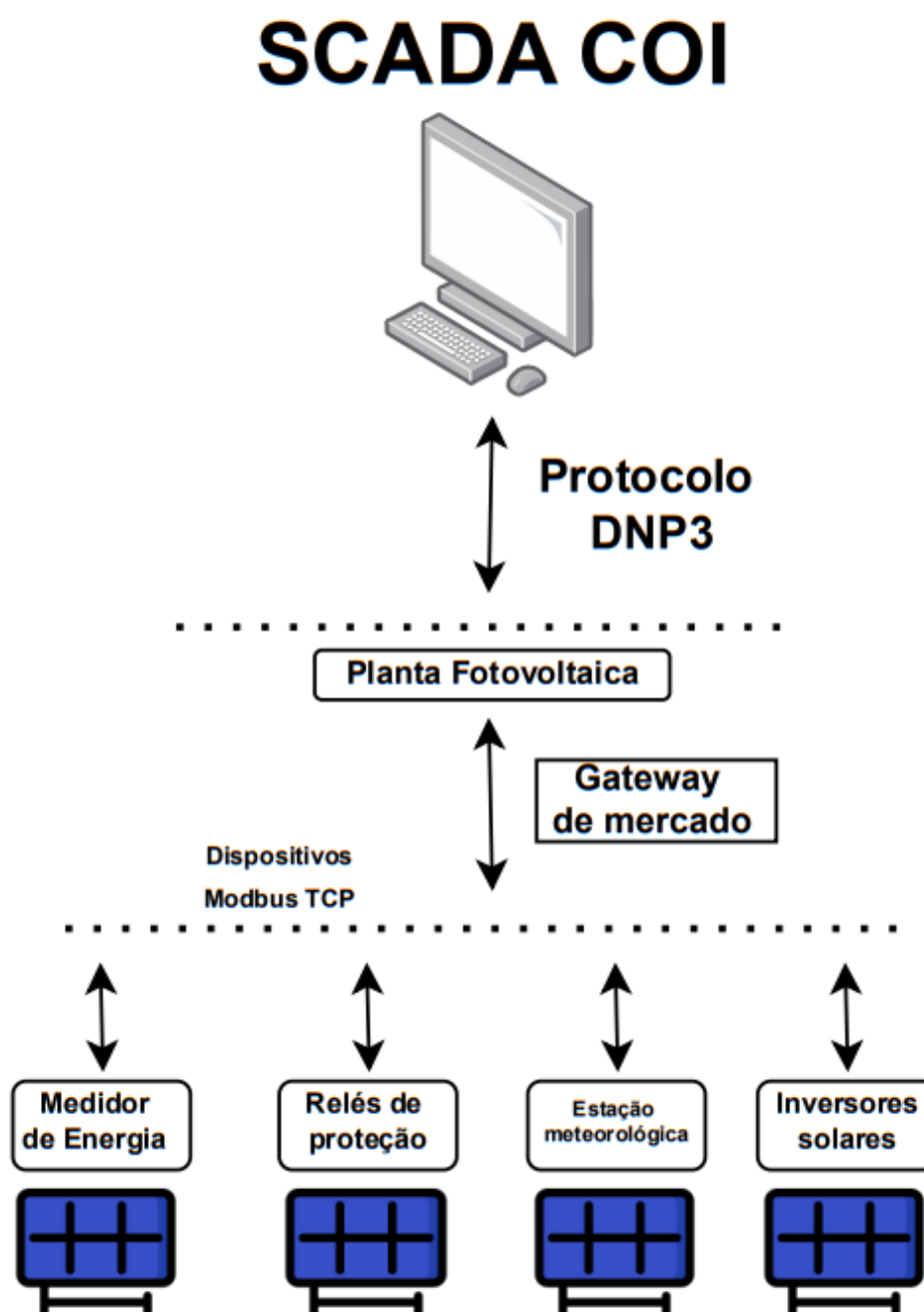
Para contornar essa limitação, existem soluções comerciais que funcionam como gateways, realizando a conversão entre Modbus TCP e DNP3. No entanto, tais soluções apresentam alto custo, especialmente em função da necessidade de múltiplas licenças para atuação como master Modbus TCP, sendo uma para cada dispositivo da planta, além da aquisição de drivers específicos para operar como slave DNP3. Esse cenário impõe barreiras financeiras significativas à implementação de um centro de operação eficiente, justificando a proposta de desenvolvimento de uma solução embarcada e de baixo custo, como apresentada neste trabalho.

3.2 Arquitetura geral

3.2.1 Cenário atual

Como apresentado anteriormente, na arquitetura do cenário atual, utilizando soluções comerciais, é necessário a aquisição de várias licenças de software a fim da centralização de um SCADA em um centro de operações. Como pode ser visto na Figura 24, para cada planta fotovoltaica, seria necessário um gateway de mercado, com várias licenças de software do protocolo Modbus TCP e 1 licença slave do protocolo DNP3.

Figura 24 – Arquitetura atual de mercado.

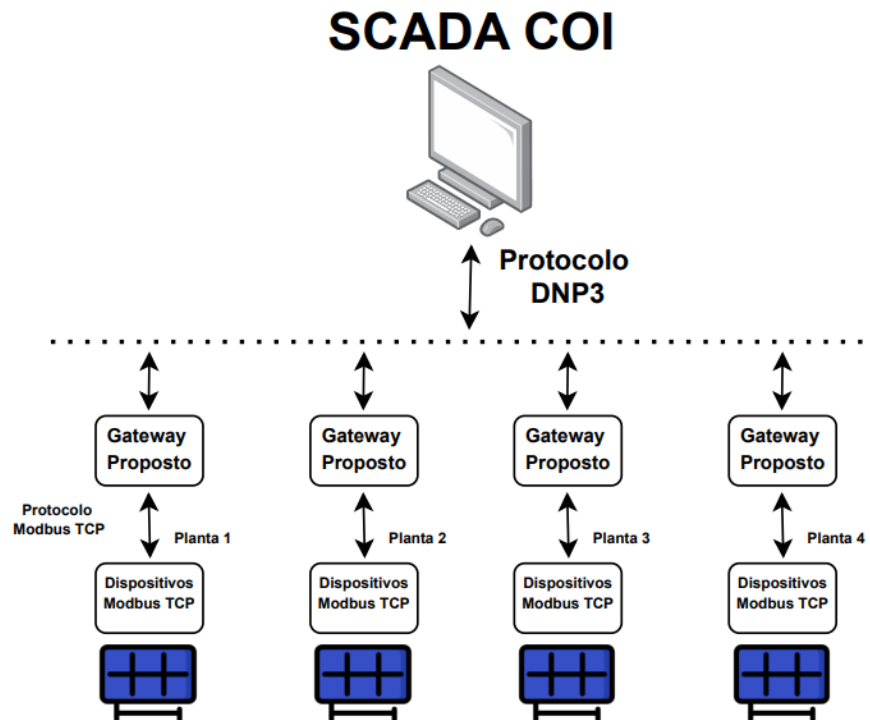


Fonte: Autoria própria.

3.2.2 Solução proposta

Já para a solução apresentada no presente trabalho, o gateway embarcado elimina a necessidade de aquisição de licenças de softwares para as plantas fotovoltaicas, permitindo consideravelmente a redução de custos nos casos onde se é necessário integrar uma ou mais usinas solares em um só ambiente supervisório. A Figura 25 exemplifica isso.

Figura 25 – Arquitetura proposta.



Fonte: Autoria própria.

A arquitetura proposta da Figura 25 baseia-se na inserção de um gateway embarcado de código aberto, desenvolvido em uma Raspberry Pi 3 Model B, que coleta informações dos equipamentos de campo e responde ao SCADA. Essa combinação permite atuar como um concentrador único, convertendo mensagens Modbus TCP (dispositivos de campo) para DNP3 (SCADA) sem custos adicionais por licença. A escolha da Raspberry Pi justifica-se pelo baixo custo e alto desempenho.

3.3 Gateway Modbus TCP/DNP3

O gateway desenvolvido neste projeto atua como um concentrador de dados Modbus TCP, responsável por coletar informações de dispositivos de campo e traduzi-las para o protocolo DNP3, permitindo a comunicação com o sistema SCADA. Para isso, o gateway foi implementado como uma **estação remota DNP3 (Outstation)**, operando sobre TCP/IP, garantindo interoperabilidade com a camada de supervisão. Esse dispositivo foi implementado utilizando a biblioteca *opendnp3*, para o protocolo DNP3 e a *libmodbus*

([RAIMBAULT, 2023](#)), para o protocolo Modbus TCP, ambas em linguagem C++. A estação remota foi configurada para representar pontos analógicos e binários relevantes ao contexto fotovoltaico. Na sua implementação, foram conduzidas duas abordagens distintas para validação da solução proposta.

A primeira abordagem consistiu no uso de uma estação de testes desenvolvida especificamente para simular dispositivos Modbus TCP e avaliar a funcionalidade do escravo DNP3, testando os principais pontos de leitura e escrita. Essa configuração permitiu um ambiente controlado e reproduzível, favorecendo ajustes de implementação e testes de estabilidade do sistema.

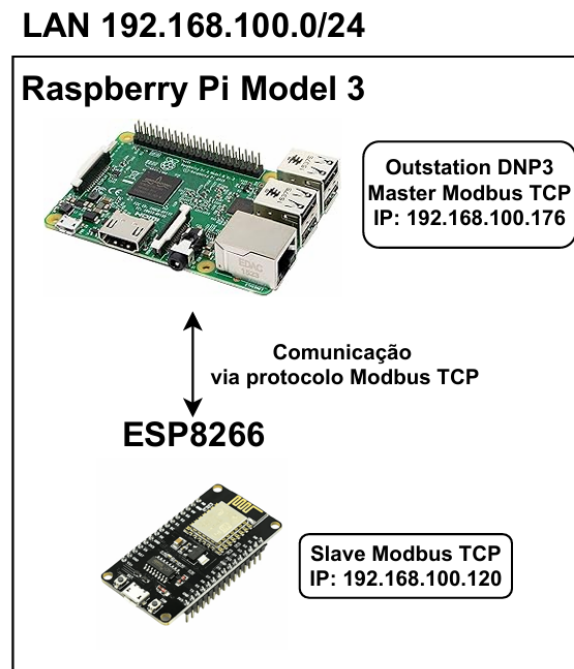
A segunda abordagem teve foco na aplicação prática, com a integração de múltiplos dispositivos reais amplamente utilizados em usinas fotovoltaicas. Nessa etapa, o gateway foi configurado para se comunicar com equipamentos operando exclusivamente sob o protocolo Modbus TCP, simulando um cenário real de campo e permitindo validar o funcionamento da solução em condições próximas às de produção.

3.3.1 Integração da estação remota DNP3 com o dispositivo simulado no ESP8266

A primeira abordagem da estação remota DNP3 embarcada na Raspberry Pi foi realizada visando a comunicação com um dispositivo Modbus TCP simulado em uma placa de desenvolvimento ESP8266. A escolha da ESP8266 se deu por sua facilidade de programação, baixo custo e conectividade nativa via Wi-Fi, o que facilitou o desenvolvimento de um ambiente de testes.

Nesta topologia, a Raspberry Pi atua como um mestre Modbus TCP e escravo DNP3 (estação remota) e o ESP8266 como um escravo Modbus TCP. Todo o sistema foi configurado em uma rede local, visando amenizar possíveis problemas de comunicação. A Figura 26 exemplifica a arquitetura de rede.

Figura 26 – Topologia de rede com Raspberry Pi e ESP8266 conectados via rede local.



Fonte: Autoria própria.

3.3.1.1 ESP8266

Antes da implementação completa do gateway, foi desenvolvido um código embarcado na ESP8266 com a finalidade de simular um equipamento Modbus TCP *slave*, permitindo validar as requisições de leitura e escrita provenientes do *master*. Essa etapa preliminar foi fundamental para garantir o correto funcionamento da comunicação e facilitar os testes do gateway em um ambiente controlado.

Primeiramente, para o desenvolvimento do código embarcado, foi necessário configurar o ambiente. Para isso, utilizou-se a plataforma Arduino IDE, por ser gratuita e amplamente difundida na comunidade de sistemas embarcados. Para possibilitar sua utilização com a placa ESP8266, foram realizadas as seguintes configurações iniciais:

- Instalação das bibliotecas e dependências específicas para os microcontroladores ESP8266;
- Configuração do *Board Manager* para o reconhecimento da placa ESP8266;
- Instalação da biblioteca *ModbusIP_ESP8266.h*, utilizada para a implementação do protocolo Modbus TCP.

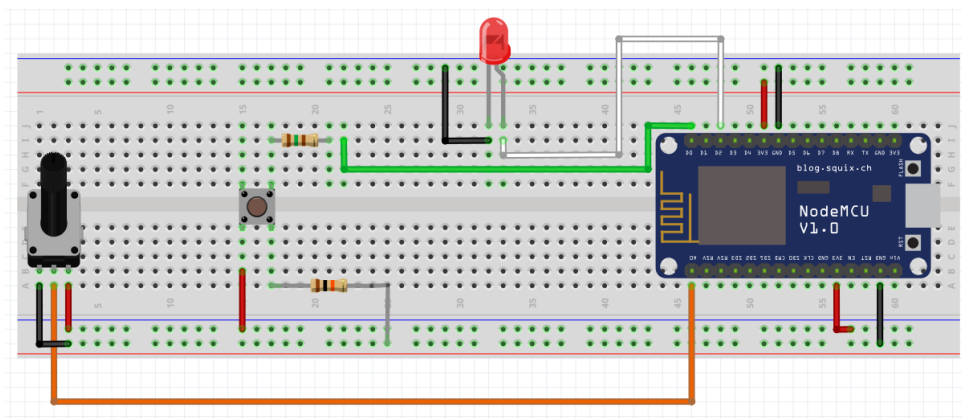
Após as configurações iniciais do ambiente de programação, visando simular variáveis típicas em sistemas supervisórios, foram conectados à placa os seguintes componentes

eletrônicos: um potenciômetro, representando uma entrada analógica; um botão, representando uma variável binária de entrada; e um LED, utilizado como atuador, permitindo testar comandos de escrita enviados pelo *master* para ligar ou desligar o dispositivo.

Essa configuração permitiu que fosse realizada a criação de um ambiente de testes funcional, simulando condições reais de monitoramento e controle, comuns em aplicações SCADA.

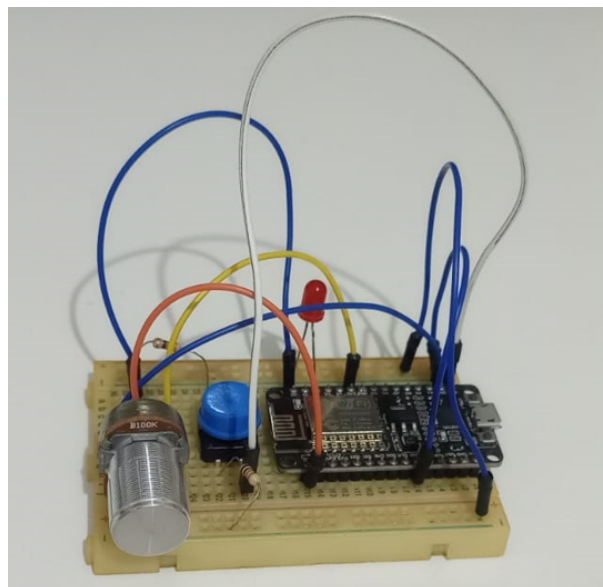
A montagem em bancada do ESP8266 pode ser visualizada nas Figura 27 e 28:

Figura 27 – Esquema de ligação elétrica do ESP8266.



Fonte: Autoria própria.

Figura 28 – Montagem física.

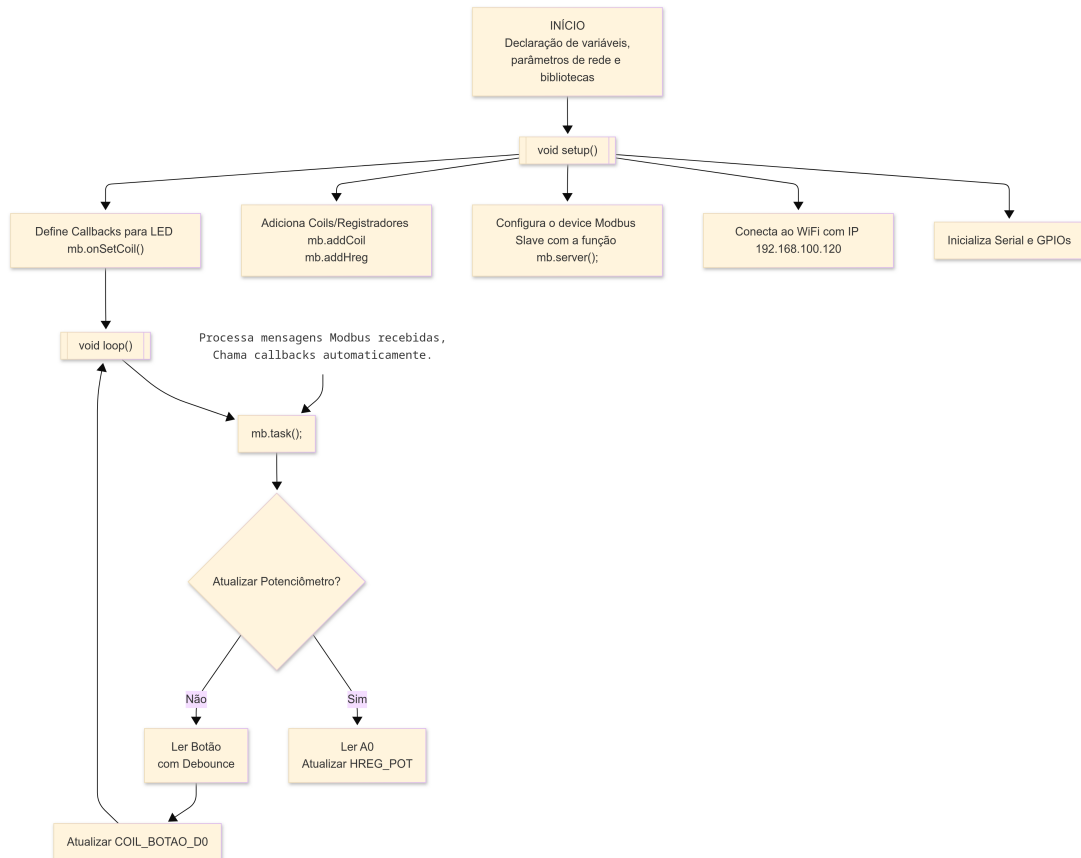


Fonte: Autoria própria.

Com o ambiente de desenvolvimento configurado e a integração dos componentes físicos concluída, foi possível elaborar o algoritmo para que a ESP8266 atuasse como um dispositivo *slave* Modbus TCP. O código completo pode ser visualizado no Anexo A.

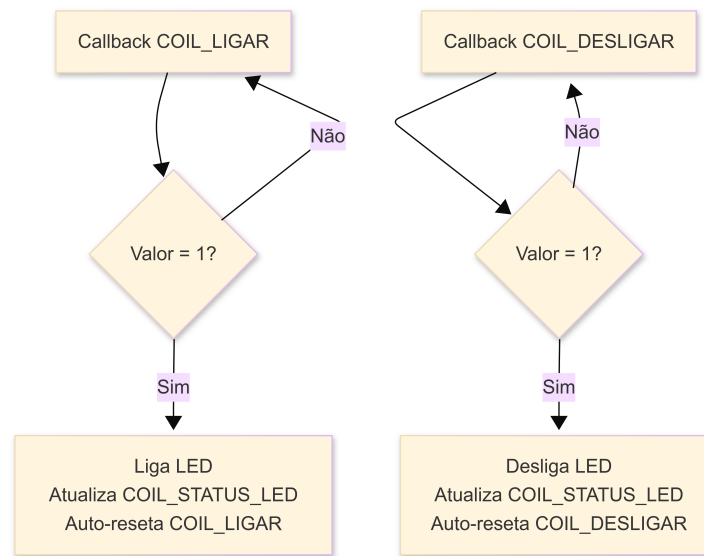
A fim de facilitar a compreensão do funcionamento do sistema, dois fluxogramas foram elaborados: o primeiro Figura 29 apresenta uma visão geral do funcionamento do algoritmo, enquanto o segundo Figura 30 detalha o processamento dos comandos recebidos para controle do LED.

Figura 29 – Fluxograma do algoritmo ESP8266.



Fonte: Autoria própria.

Figura 30 – Callback de comandos ESP8266.



Fonte: Autoria própria.

Como visualizado nas Figuras 26 a 29, logo no início do programa a placa foi configurada com IP fixo 192.168.100.120, a configuração do IP estático, possibilita uma fácil identificação do equipamento na rede local de testes. O ID do dispositivo Modbus, é implicitamente tratado como 1 pela biblioteca, não sendo necessária configuração.

Como observado na Figura 29 o servidor Modbus TCP é iniciado com a função `mb.server()`, além disso, em `void setup()`, é disponibilizado os pontos configurados para leitura e escrita via funções como `mb.addCoil()` e `mb.addHreg()`.

A lógica de controle dos comandos Modbus, visualizada em Figura 30 foi implementada através de funções de *callback*, utilizando a função `mb.onSetCoil()`. Esses callbacks são responsáveis por detectar escritas nas coils específicas e executar as ações correspondentes, como ligar ou desligar o LED. Após a execução, os comandos são automaticamente resetados para evitar reacionamento acidental.

O algoritmo principal, definidos em `void loop()` realiza varreduras contínuas utilizando a função `mb.task()`, que verifica se há requisições Modbus pendentes. Além disso, um controle de tempo interno foi adicionado para que a leitura do potenciômetro seja atualizada a cada 500 ms, e a leitura do botão seja feita com debounce de 50 ms, garantindo estabilidade nas leituras digitais. Enquanto o LED é controlado assincronamente via callbacks Modbus, a leitura do potenciômetro e do botão utiliza polling periódico.

E por fim, para complementar o entendimento, a Tabela 3 apresenta o endereçamento Modbus TCP implementado no ESP8266 para simular as variáveis de um inversor. Os endereços foram configurados para representar comandos e parâmetros típicos de um inversor real, como o acionamento (coils de escrita) e a leitura de grandezas elétricas (holding registers). Por exemplo, o endereço 00001 corresponde ao comando de partida do

inversor, enquanto 40001 permite a leitura da potência ativa simulada. Essa abordagem transforma o ESP8266 em um simulador funcional para testes de integração com sistemas SCADA via DNP3.

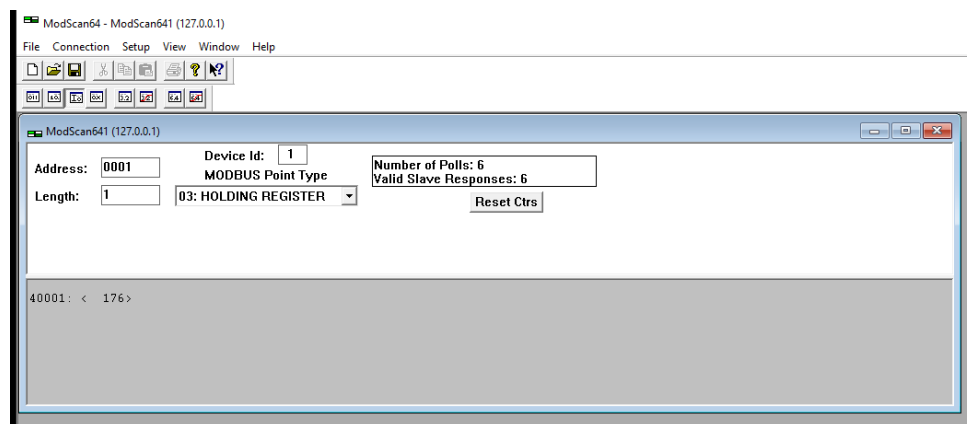
Tabela 3 – Endereçamento Modbus TCP configurado no ESP8266

Endereço Modbus	Tipo	Acesso (Master)	Descrição
00001	Coil	Escrita	Comando para ligar o LED (auto-reset após escrita)
00002	Coil	Escrita	Comando para desligar o LED (auto-reset após escrita)
00003	Coil	Leitura	Estado atual do LED, atualizado internamente
00004	Coil	Leitura	Estado do botão físico (com lógica invertida)
40001	Holding Register	Leitura	Valor analógico lido do potenciômetro (0–1023)

Fonte: Autoria própria.

Para validar os endereços criados na ESP8266, descritos na Tabela 3, foi utilizado a versão de testes do Software *ModScan64* para ler e escrever nos endereços configurados.

Figura 31 – Software utilizado para validação dos endereços Modbus.



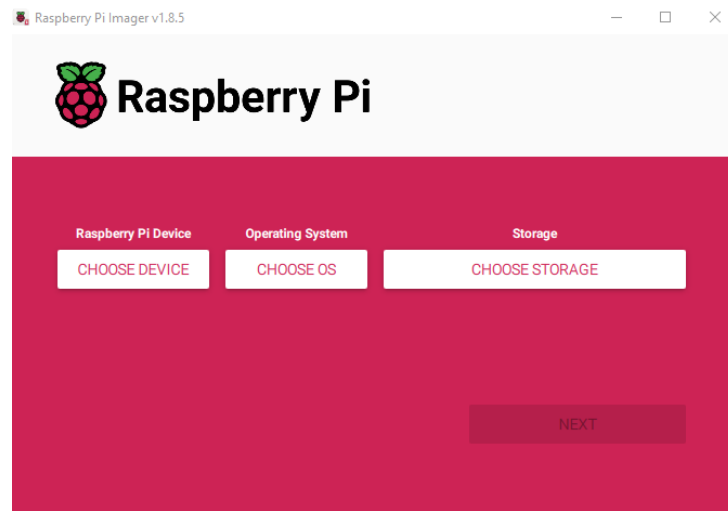
Fonte: (WIN-TECH, 2025).

3.3.1.2 Raspberry Pi

Então, após obter um dispositivo *slave* Modbus 100% funcional o desenvolvimento da Raspberry Pi como gateway Modbus TCP/DNP3 foi iniciado.

Inicialmente, foi necessário configurar o sistema operacional da Raspberry, para isso utilizou-se o software *Raspberry Pi Imager*, Figura 32, para baixar e instalar o sistema operacional em um cartão de memória.

Figura 32 – Software para instalação do Sistema Operacional em placas Raspberry.



Fonte: ([Raspberry Pi Foundation, 2025](#)).

O sistema operacional escolhido foi o Raspberry Pi OS, uma distribuição baseada no Debian Linux. Visando um desempenho mais otimizado, optou-se pela versão Lite, que não possui interface gráfica, sendo mais leve e adequada para aplicações embarcadas. Após a instalação do sistema operacional, foram instaladas todas as dependências necessárias para a utilização das bibliotecas *OpenDNP3* e *libmodbus*. As bibliotecas foram configuradas de forma global no sistema da Raspberry Pi, permitindo que qualquer aplicação desenvolvida no ambiente tenha acesso a elas.

Adicionalmente, foram instaladas as ferramentas **g++** e **cmake**, essenciais para a compilação de projetos em C++ no ambiente Linux, bem como o **git**, utilizado para clonar repositórios hospedados no *GitHub*.

Para a instalação das dependências e bibliotecas, os seguintes comandos foram executados no terminal Linux:

Código 3.1 – Comandos para instalação das dependências no terminal Linux

```
1 sudo apt update && sudo apt install git
2 sudo apt update && sudo apt install cmake
3 sudo apt update && sudo apt install g++
4 sudo apt update && sudo apt install libmodbus-dev
```

Nota-se que a biblioteca *libmodbus*, permite a instalação de forma oficial pelo sistema linux, porém, o mesmo não se aplica para a biblioteca *OpenDNP3*. Nesse caso, foi preciso clonar o repositório do github e instalar a biblioteca de forma universal no sistema, para isso foram utilizados os comandos:

Código 3.2 – Comandos para instalação das dependências no terminal Linux

```
1 git clone https://github.com/dnp3/opendnp3
2 cd opendnp3
```

```
3 mkdir build
4 cd build
5 cmake ..
6 make
7 sudo make install
8 echo 'export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH' >> ~/.bashrc
9 source ~/.bashrc
```

Com os requisitos mínimos adequados, foi feita a configuração de um endereço IP fixo para a Raspberry Pi, por meio da edição do arquivo de sistema `dhcpcd.conf`. Como visualizado na Figura 26, o IP definido foi: 192.168.100.176, garantindo consistência na comunicação com os demais dispositivos da rede local.

Ainda antes do início efetivo do desenvolvimento do código do gateway, buscou-se otimizar o fluxo de trabalho entre o computador pessoal de desenvolvimento e a Raspberry Pi. Para isso, foi instalado o Samba, permitindo o compartilhamento de arquivos entre os dois dispositivos. A instalação foi feita com o comando:

Código 3.3 – Comandos para instalação das dependências no terminal Linux

```
1 sudo apt update && sudo apt install samba -y
```

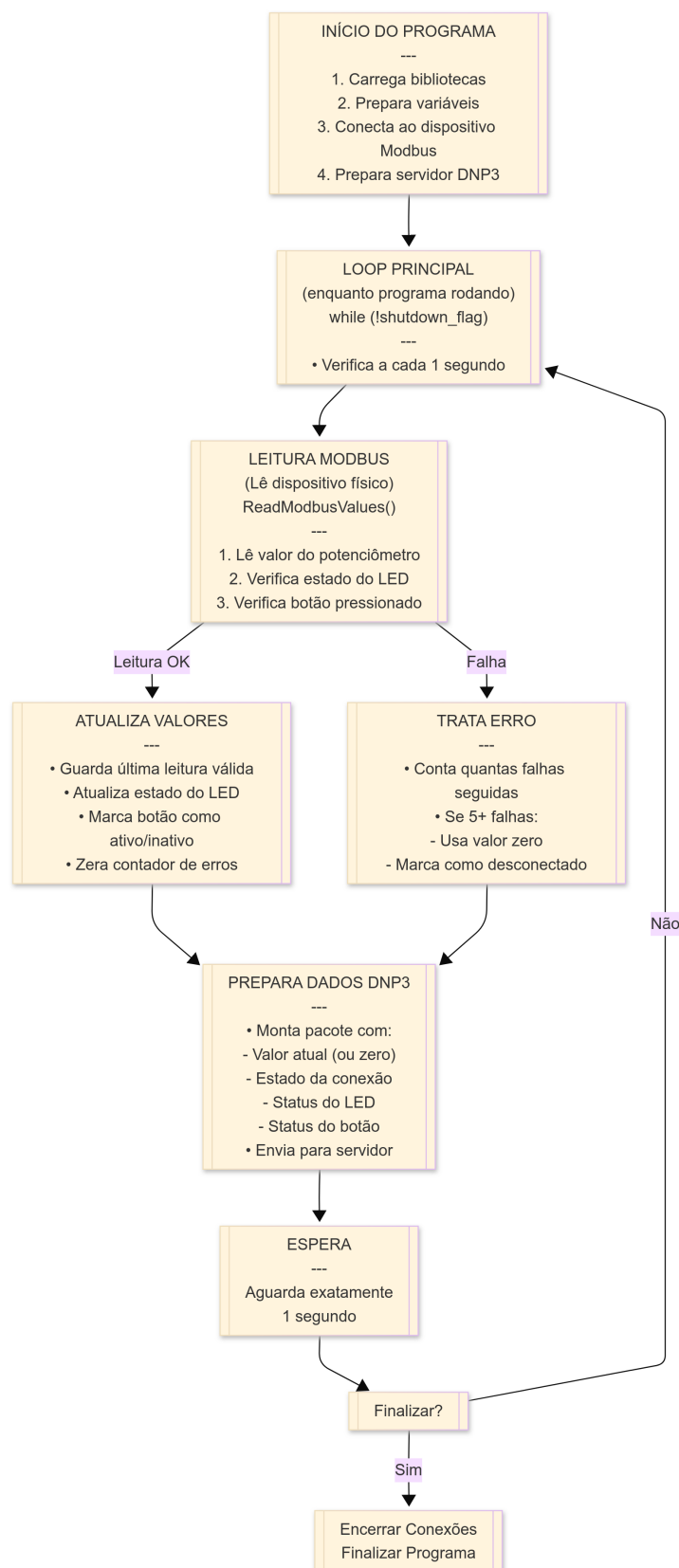
Após a instalação, o serviço foi configurado editando o arquivo `smb.conf`, de forma a disponibilizar os diretórios necessários para edição dos arquivos do programa.

Com isso, tornou-se possível editar os arquivos da Raspberry diretamente a partir do ambiente de desenvolvimento pessoal no Windows, utilizando o software Visual Studio Code. A conexão com a Raspberry foi feita por meio do protocolo *SSH*, que foi ativado no momento da instalação do SO, através do software apresentado na Figura 32. Dessa forma, foi possível o acesso ao terminal e os arquivos da placa, facilitando o processo de edição, compilação e testes do código.

Em seguida, após a finalização do ambiente de desenvolvimento e a validação do funcionamento das bibliotecas — por meio de exemplos fornecidos pelos próprios repositórios —, finalmente foi desenvolvido a primeira implementação do gateway proposto. O algoritmo completo pode ser visualizado no Anexo B.

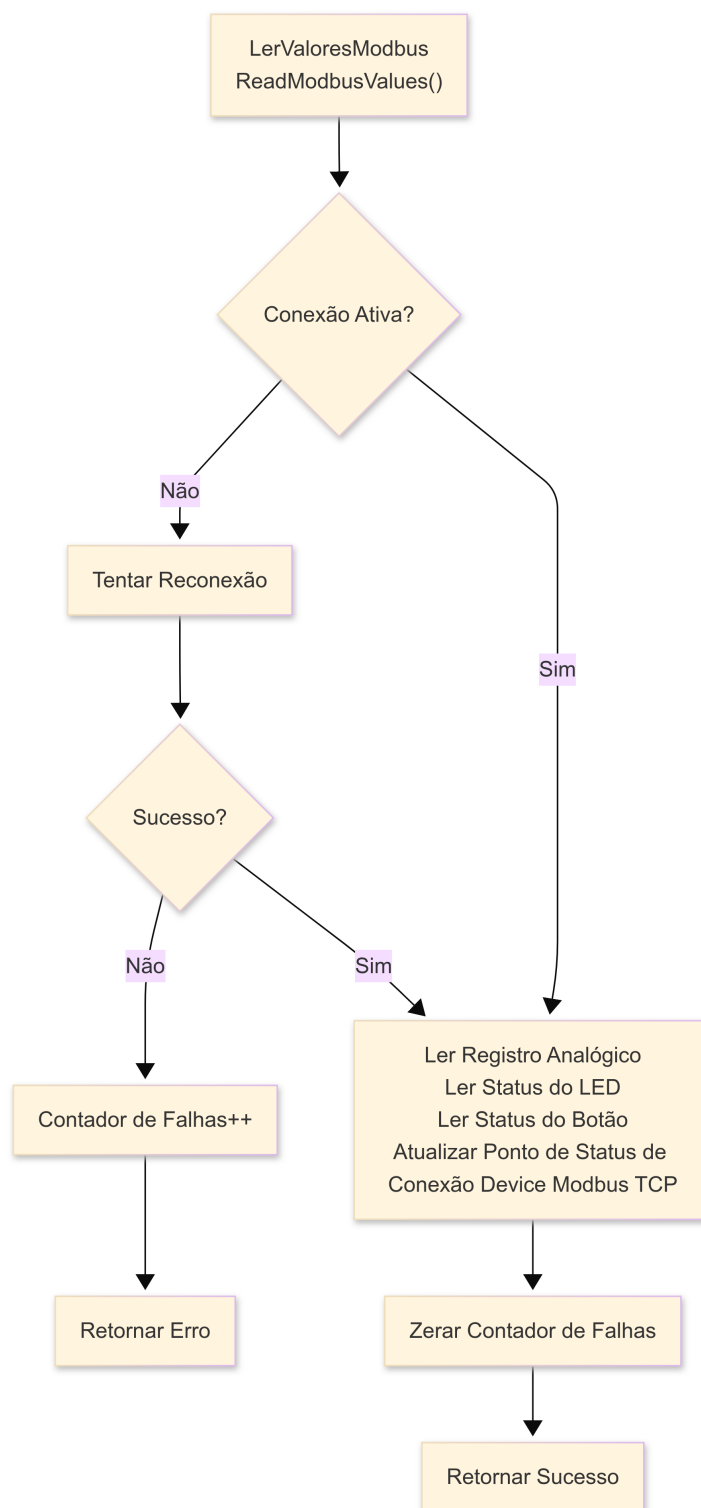
Com o objetivo de facilitar o entendimento do programa embarcado na Raspberry Pi, foram elaborados fluxogramas de funcionamento, os fluxogramas consistem no fluxograma do algoritmo principal, fluxograma de leitura Modbus, fluxograma de reconexão Modbus e manipulador de comandos DNP3. Respectivamente os fluxogramas podem ser visualizados nas Figuras 33, 34, 35 e 36.

Figura 33 – Fluxograma principal de funcionamento do programa embarcado.



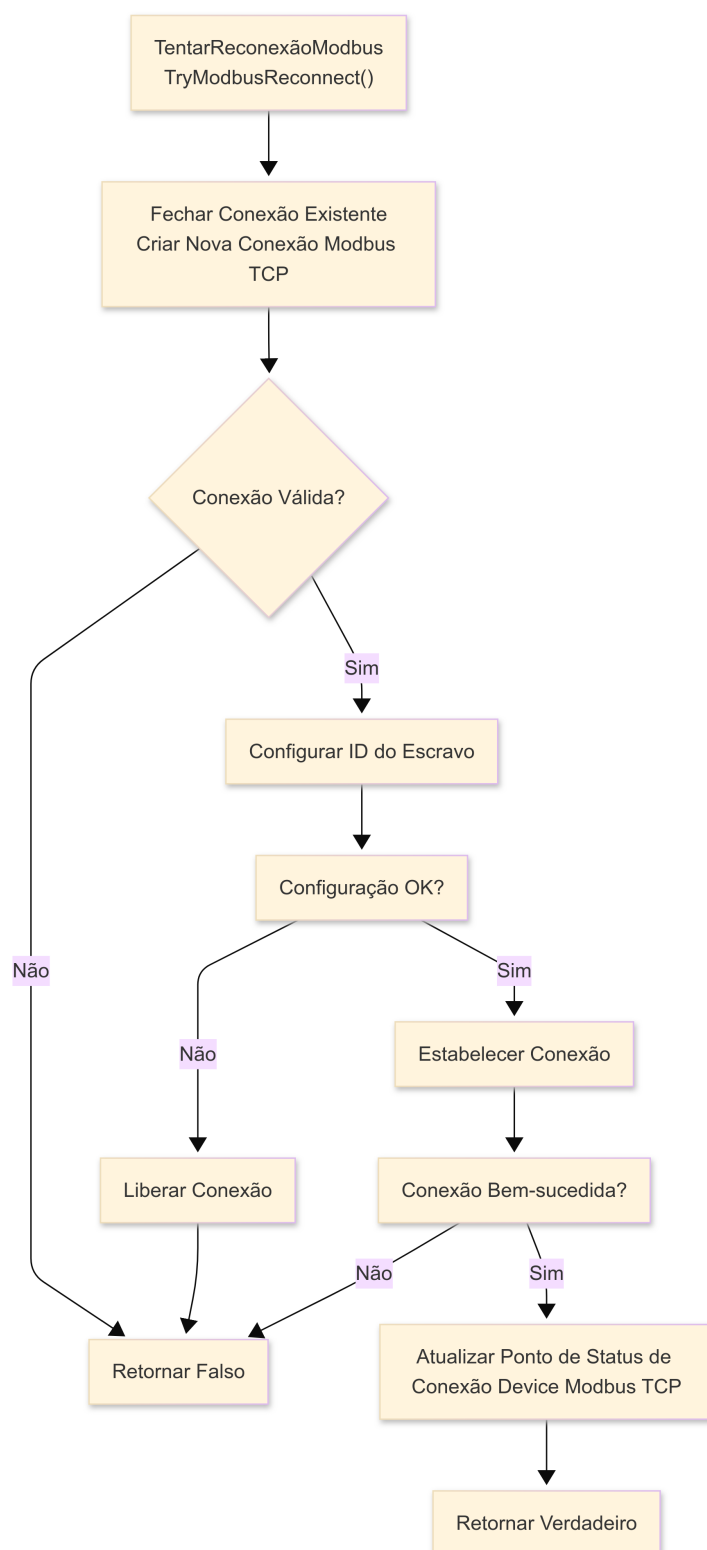
Fonte: Autoria própria.

Figura 34 – Fluxograma da função de leitura Modbus.



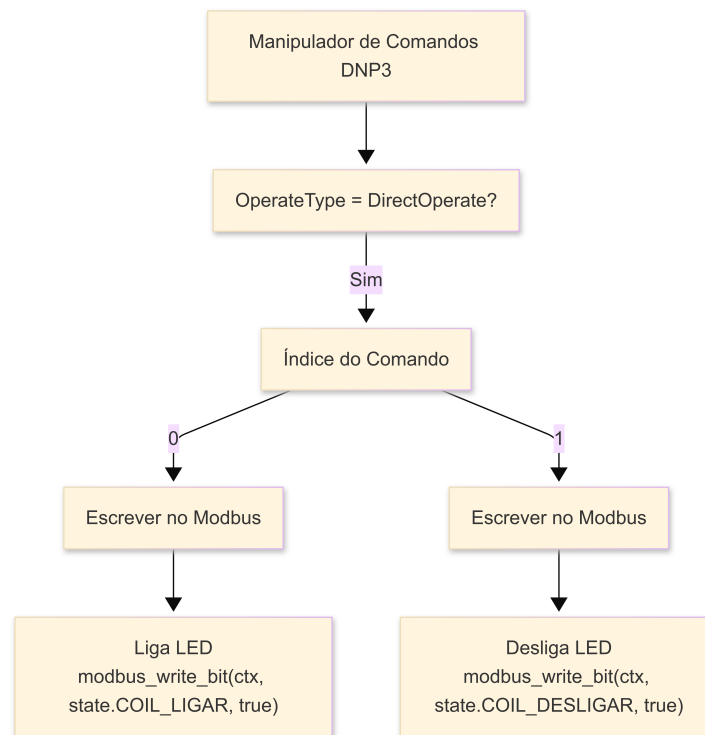
Fonte: Autoria própria.

Figura 35 – Fluxograma da função de Reconexão Modbus.



Fonte: Autoria própria.

Figura 36 – Fluxograma do Manipulador de Comandos.



Fonte: Autoria própria.

Basicamente, o código apresentado no Anexo B implementa um gateway embarcado que integra um dispositivo Modbus TCP a uma outstation DNP3, configurada por meio da biblioteca *OpenDNP3*. Para isso, o gateway utiliza a biblioteca *libmodbus* para se conectar a um escravo Modbus TCP, realizando a leitura de dados analógicos e digitais a cada 1000ms, conforme ilustrado na Figura 34. Os dados obtidos do dispositivo Modbus são convertidos em pontos DNP3 e disponibilizados a um mestre DNP3 através de uma conexão TCP.

Além da leitura de dados, o sistema também é capaz de executar comandos de controle — como ligar e desligar um LED — ao receber instruções do mestre DNP3 por meio de comandos do tipo Direct Operate, conforme representado na Figura 36.

É importante destacar que a outstation foi configurada para suportar eventos não solicitados, permitindo que as informações sejam publicadas de forma “não solicitada”, sem a necessidade de requisições e varreduras do mestre. Outro ponto relevante é a implementação de um mecanismo de reconexão automática com o dispositivo Modbus TCP, garantindo maior robustez ao sistema em caso de falhas de comunicação. Esse processo está detalhado na Figura 35.

No código demonstrado, a comunicação com o dispositivo Modbus TCP ocorre via endereço IP 192.168.100.120, utilizando a porta padrão 502 e o ID de escravo 1. Por sua vez, a outstation DNP3 é configurada para operar no IP 192.168.100.176, escutando

na porta 20000 para conexões com um mestre DNP3. Os endereços lógicos utilizados pelo protocolo DNP3 são definidos por meio dos parâmetros `config.link.LocalAddr`, endereço local da outstation, configurado como 2 e `config.link.RemoteAddr` endereço do mestre, configurado como 1.

Essa configuração completa permite que o gateway atue de forma eficaz como uma ponte entre os dois protocolos, traduzindo comandos recebidos do mestre DNP3 em operações Modbus e de forma inversa, convertendo leituras Modbus em pontos DNP3 publicáveis.

Vale ressaltar ainda que, caso o programa seja iniciado sem que a comunicação com o dispositivo Modbus esteja estabelecida, o sistema entra automaticamente em um modo de espera, realizando tentativas de reconexão a cada 5 segundos. Durante esse período, a criação da outstation DNP3 é adiada, evitando que o sistema entre em operação em um estado inconsistente ou sem dados válidos.

Por fim, para complementar o entendimento, as Tabelas 4 e 5 apresentam os endereços e pontos configurados via código na Raspberry Pi, tanto para a comunicação Modbus TCP quanto para o cadastro dos pontos do protocolo DNP3, respectivamente.

Tabela 4 – Endereçamento Modbus TCP configurados na Raspberry Pi

Endereço	Tipo	Acesso	Descrição
HR0	Holding Register	Leitura	Valor do potenciômetro (0-1023)
Coil 2	Coil	Leitura	Status do LED (1 = ligado)
Coil 3	Coil	Leitura	Status do botão (1 = solto)
Coil 0	Coil	Escrita	Comando ligar LED (1 = acionar)
Coil 1	Coil	Escrita	Comando desligar LED (1 = acionar)

Fonte: Autoria própria

Tabela 5 – Endereçamento DNP3 configurado na Raspberry Pi

Índice	Tipo de Ponto	Acesso	Descrição	Fonte Modbus	Tipo de Objeto/Classe
0	Analog Input	Leitura	Valor do potenciômetro (0-1023)	HR0	Group30Var2 - Class 2
0	Binary Input	Leitura	Status de conexão (0 = OK, 1 = Falha)	Gerado internamente	Group1Var2 - Class 1
1	Binary Input	Leitura	Status do LED	Coil 2	Group1Var2 - Class 1
2	Binary Input	Leitura	Status do botão	Coil 3	Group1Var2 - Class 1
0	Binary Output	Escrita	Comando para ligar LED	Coil 0	Group12Var1
1	Binary Output	Escrita	Comando para desligar LED	Coil 1	Group12Var1

Fonte: Autoria própria.

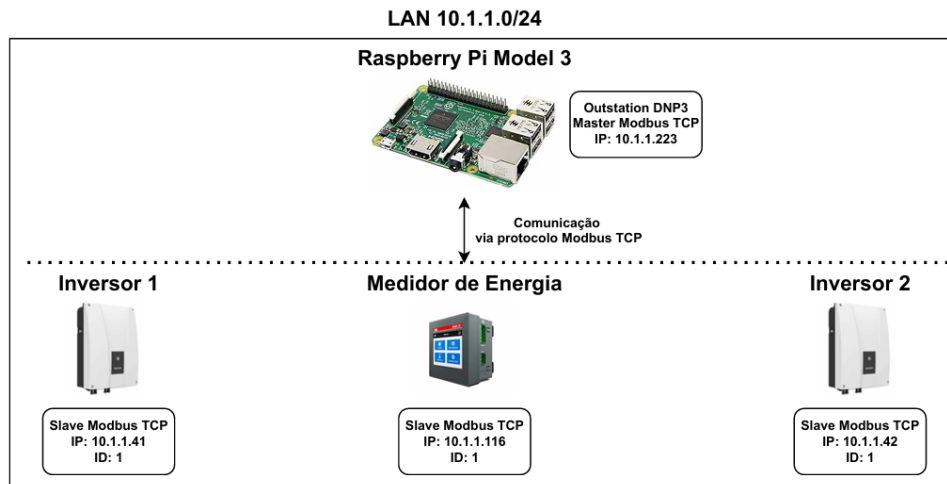
Essa abordagem com estação de testes permitiu simular um cenário controlado, viabilizando o desenvolvimento e a validação das funcionalidades do gateway. A utilização do ESP8266 como dispositivo Modbus TCP permitiu bastante flexibilidade para implementar os principais tipos de dados do protocolo DNP3, dessa forma foi possível verificar o funcionamento correto da “ponte” entre os dois protocolos. Essa configuração foi essencial para testar o sistema antes de sua aplicação com dispositivos reais.

3.3.2 Estação remota DNP3 integrada com dispositivos reais

A segunda abordagem do gateway embarcado foi implementada com foco na comunicação entre dispositivos reais comumente utilizados em projetos fotovoltaicos. Para isso, foram utilizados como dispositivos escravos, dois inversores solares e um medidor de energia. Nessa topologia, a Raspberry Pi atua como mestre Modbus TCP para os dispositivos de campo, enquanto desempenha a função de estação remota no protocolo DNP3, de forma semelhante à implementação descrita na Seção 3.3.1.

Como mencionado anteriormente, essa abordagem tem um foco prático, integrando três dispositivos Modbus TCP, que são amplamente utilizados em aplicações fotovoltaicas. A topologia implementada nesta etapa pode ser visualizada na Figura 37.

Figura 37 – Topologia de rede com Raspberry Pi e os equipamentos fotovoltaicos conectados via rede local.



Fonte: Autoria própria.

Para facilitar a visualização da arquitetura geral da implementação 2, foi criada a Tabela 6.

Tabela 6 – Dispositivos da Rede Industrial

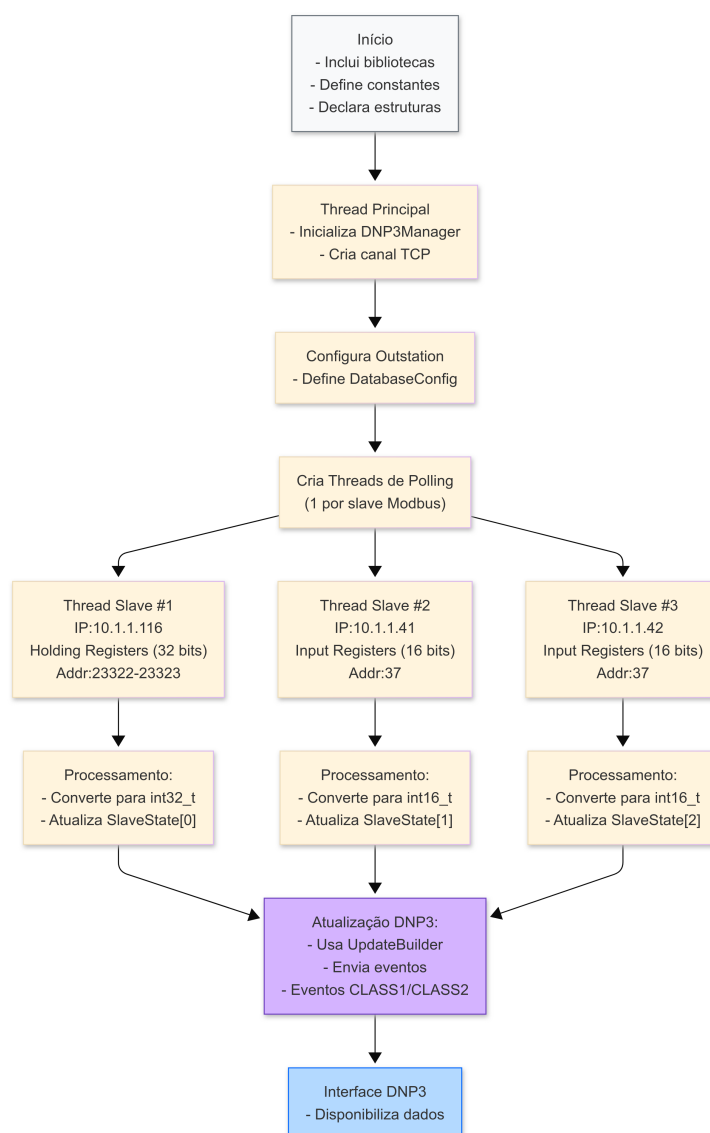
Modelo do Dispositivo	Endereço IP
Raspberry Pi (Gateway)	10.1.1.223
Multimedidor de Energia ABB M4M 30	10.1.1.116
Inversor Ingeteam Ingecon Sun Storage 1Play TL M	10.1.1.41
Inversor Ingeteam Ingecon Sun Storage 1Play TL M	10.1.1.42

Fonte: Autoria própria.

Nesta segunda implementação, não foram necessárias as configurações iniciais descritas na Seção 3.3.1, uma vez que as bibliotecas utilizadas permanecem sendo as mesmas descritas anteriormente. As únicas alterações realizadas foram a configuração de um novo IP estático na Raspberry Pi, com o objetivo de estabelecer comunicação com os escravos Modbus TCP, como visualizado na Figura 37. Além, foi preciso realizar modificações consideráveis no código embarcado da placa.

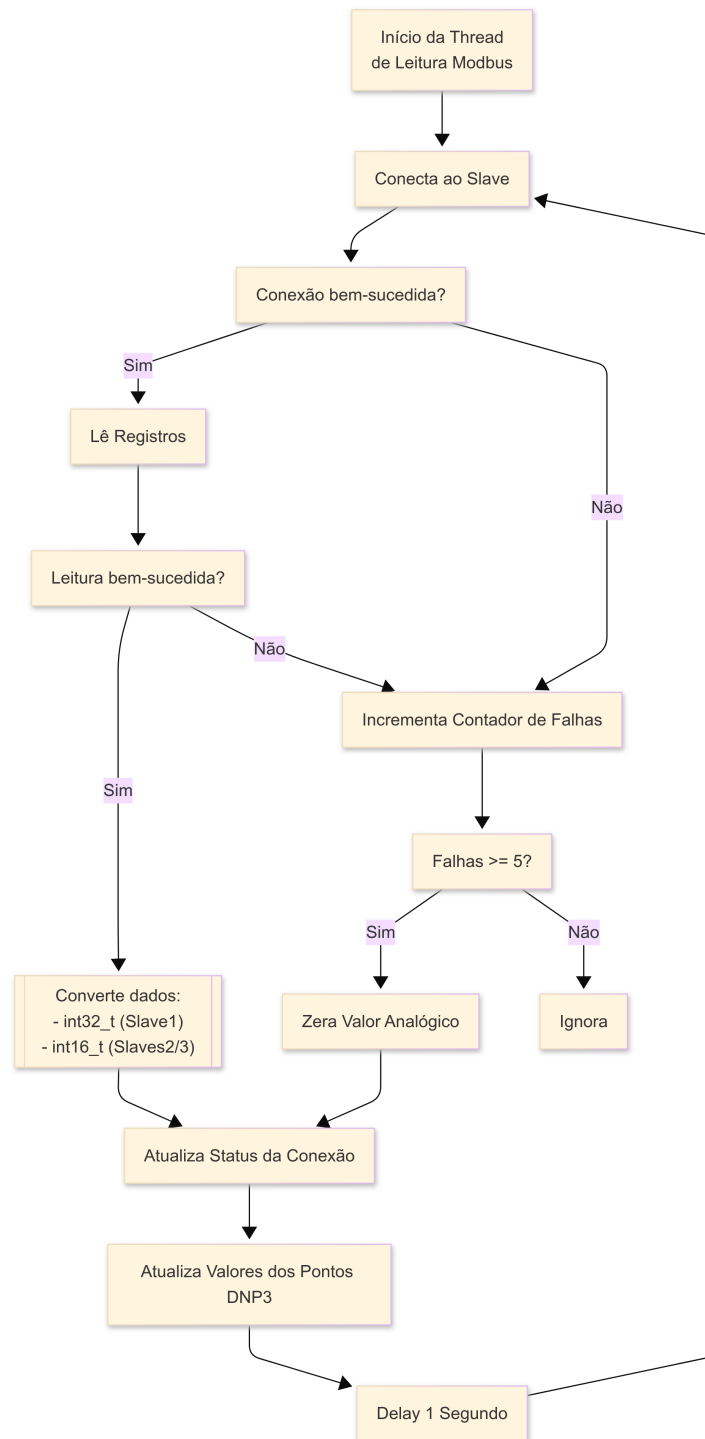
O programa desenvolvido na presente seção, pode ser visualizado no Anexo C. Para facilitar a compressão do algoritmo desenvolvido, análogamente à seção anterior, foram realizados fluxogramas das principais funções do software, que podem ser observados na Figura 38 e 39.

Figura 38 – Fluxograma principal do Algoritmo.



Fonte: Autoria própria.

Figura 39 – Fluxograma da Thread de Polling Modbus detalhada.



Fonte: Autoria própria.

A visão geral do funcionamento do algoritmo é apresentada na Figura 38, na qual observa-se que o gateway proposto efetua a cada 1000ms a leitura dos registros Modbus de potência ativa provenientes dos três dispositivos, realiza tratamentos nos dados e os converte para o protocolo DNP3. As configurações gerais da Outstation nesta segunda implementação permanecem análogas às descritas na Seção 3.3.2, sendo o endereço local

da Outstation configurado como 2 e o endereço do mestre como 1. Ressalta-se ainda que os dados não solicitados, *unsolicited responses* seguem habilitados nesta etapa. As únicas alterações realizadas referem-se ao endereço IP da remota, conforme descrito anteriormente.

O Gateway desenvolvido, realiza o monitoramento contínuo dos dispositivos Modbus listados na Tabela 6, por meio de múltiplas threads de polling. Cada thread é responsável por se conectar periodicamente a um dispositivo, realizar a leitura dos dados via Modbus TCP e publicar os valores na outstation DNP3, conforme ilustrado na Figura 39. Essa abordagem permite que os dados da planta sejam atualizados em tempo real e sejam disponibilizados ao mestre DNP3. Além disso, a aplicação implementa um mecanismo de tratamento de falhas: em caso de perda de conexão com o escravo, é registrado o valor lógico 1 no ponto Binary Input, sinalizando falha de comunicação. A partir disso, inicia-se uma contagem de até cinco falhas consecutivas; ao final dessa contagem, o valor associado ao ponto Analog Input correspondente é zerado automaticamente.

Assim como no tópico anterior, as Tabelas 7 e 8 demonstram os pontos de leitura cadastrados no código, tanto no protocolo Modbus TCP, quanto no DNP3 respectivamente. É importante lembrar que os registros Modbus foram retirados do manual de operação dos equipamentos.

Tabela 7 – Endereçamento dos dados Modbus TCP monitorados

Endereço Modbus	Tipo	Acesso	Descrição
23322	Holding Register (2x 16 bits)	Leitura	Potência Ativa do Medidor
37	Input Register (16 bits)	Leitura	Potência Ativa do Inversor 1
37	Input Register (16 bits)	Leitura	Potência Ativa do Inversor 2

Fonte: Autoria própria.

Tabela 8 – Endereçamento dos pontos DNP3 configurados

Índice	Tipo	Descrição	Fonte Modbus	Objeto/Classe	Acesso
0	Analog Input	Potência ativa do medidor	Holding Register 23322	G30V1 / Classe 2	Leitura
1	Analog Input	Potência ativa do inversor 1	Input Register 37	G30V2 / Classe 2	Leitura
2	Analog Input	Potência ativa do inversor 2	Input Register 37	G30V2 / Classe 2	Leitura
0	Binary Input	Status de conexão do dispositivo 1	Interno	G1V2 / Classe 1	Leitura
1	Binary Input	Status de conexão do dispositivo 2	Interno	G1V2 / Classe 1	Leitura
2	Binary Input	Status de conexão do dispositivo 3	Interno	G1V2 / Classe 1	Leitura

Fonte: Autoria própria.

Nota-se na Tabela 7 que a potência ativa do medidor de energia é um valor de 32 bits. Dessa forma, foi necessária a leitura de dois registradores Modbus de 16 bits consecutivos (endereços 23322-23323) para a reconstrução do valor completo.

Para isso, implementou-se uma função no código-fonte responsável por realizar a conversão dos dois registradores de 16 bits em um valor de 32 bits. O trecho de código responsável por essa operação é apresentado a seguir:

Código 3.4 – Conversão de dois registradores Modbus de 16 bits em um valor de 32 bits

```

1 int32_t modbusRegistersToInt32(uint16_t* regs, bool is_signed) {
2     int32_t value = (regs[0] << 16) | regs[1];
3     if (is_signed && (value & 0x80000000)) {
4         value |= 0xFFFFFFFF00000000;
5     }
6     return value;
7 }

```

Essa função realiza a junção dos dois registradores utilizando operações de deslocamento e bitwise. Caso o valor seja assinado e o bit mais significativo esteja ativado, é feita a extensão de sinal para preservar corretamente a representação negativa em 32 bits.

É necessário frisar também que o ponto DNP3 do medidor precisou ser ajustado para suportar um dado de 32 bits, para isso foi preciso ajustar seu objeto e variação descrito na Tabela 8.

Nesta etapa, foram empregados dispositivos Modbus reais utilizados em usinas fotovoltaicas, com o objetivo de verificar a compatibilidade do gateway com equipamentos presentes em ambientes operacionais. Apesar de se ter mudanças claras no algoritmo, a configuração da outstation DNP3 foi mantida, adaptando apenas os parâmetros de conexão para os dispositivos em questão. Essa fase complementou os testes em ambiente simulado, permitindo a avaliação do comportamento do sistema frente às particularidades de equipamentos industriais.

3.4 Observações gerais de ambas as implementações

Ambas as implementações apresentadas nas seções anteriores, requerem a utilização de um arquivo `CMakeLists.txt` para a configuração e automatização do processo de compilação. O uso do CMake foi fundamental para gerenciar dependências externas, como as bibliotecas *OpenDNP3* e *libmodbus*.

O arquivo `CMakeLists.txt` pode ser visualizado no Anexo [D](#).

3.5 Configuração do mestre DNP3 para validar o gateway

Visando validar a comunicação via protocolo DNP3, foram utilizados softwares capazes de simular um mestre DNP3 para testar a transmissão de dados entre o gateway embarcado e o sistema supervisor. Nos tópicos seguintes, serão apresentadas as formas como esses testes foram executados.

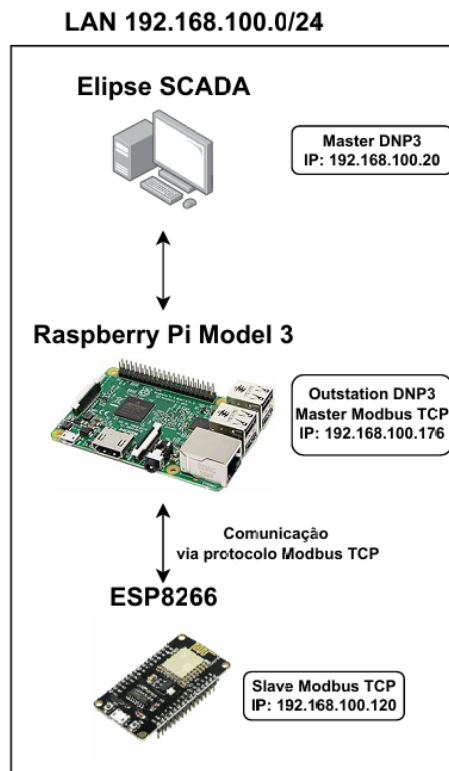
3.5.1 Implementação 1

Para a leitura e a escrita dos pontos DNP3 disponibilizados no gateway desenvolvido na Seção [3.3.1](#), dois mestres foram configurados. O primeiro consistiu no uso da solução comercial Elipse SCADA ([ELIPSE, 2023](#)), com licença de desenvolvedor, enquanto o segundo foi realizado por meio da versão de testes do software KEPServerEX ([INC., 2023](#)), desenvolvido pela Kepware, utilizado como simulador do protocolo DNP3.

3.5.1.1 Mestre DNP3 utilizando Elipse SCADA

Para uma visualização geral, a Figura [40](#) ilustra a topologia da arquitetura empregada, com o Elipse SCADA atuando como mestre DNP3.

Figura 40 – Topologia utilizando Elipse SCADA como mestre.



Fonte: Autoria própria.

Durante os testes, foi configurado o driver DNP3 com o IP e os parâmetros de endereço corretos do gateway. Em seguida, foram cadastrados os pontos descritos na Tabela 5, conforme ilustrado na Figura 46.

Figura 41 – Pontos DNP3 cadastrados no driver do Elipse SCADA.

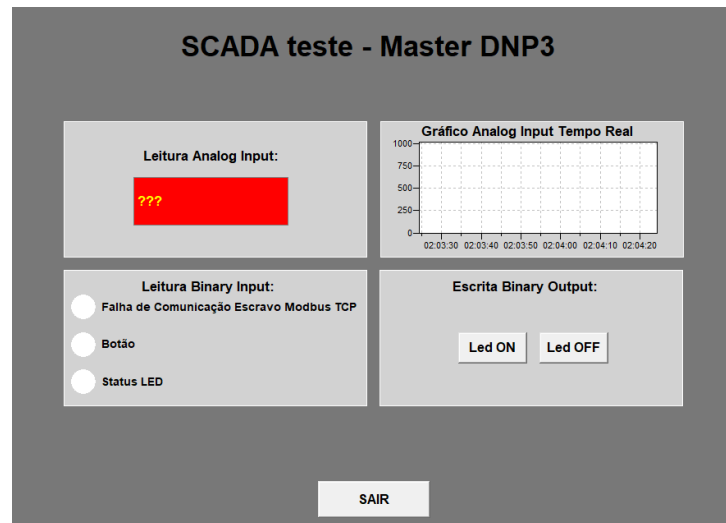
Nome	Dispo...	Item	P1/N1/B1	P2/N2/B2	P3/N3/B3	P4/N4/B4	Taman...	Varredura	Leitura?	Escrita?
DriverMestre			0	0	0	0				
Command										
led_on			0	5	1201	0		1000	☐	☒
led_off			0	5	1201	1		1000	☐	☒
DIGITAL										
status_communication			0	1	102	0		1000	☒	☐
led_status			0	1	102	1		1000	☒	☐
button_status			0	1	102	2		1000	☒	☐
ANALOG										
analog_read			0	1	3002	0		1000	☒	☐

Fonte: Autoria própria.

Nota-se que, na coluna “P3/N3/B3”, constam o grupo (objeto) e a variação dos dados. O driver foi configurado para operar por varredura, com todos os pontos sendo lidos a cada 1 segundo.

Para facilitar a visualização em tempo real, foi desenvolvida uma interface simples com todos os pontos monitorados. Essa tela pode ser observada na Figura 42.

Figura 42 – Tela de testes no SCADA.



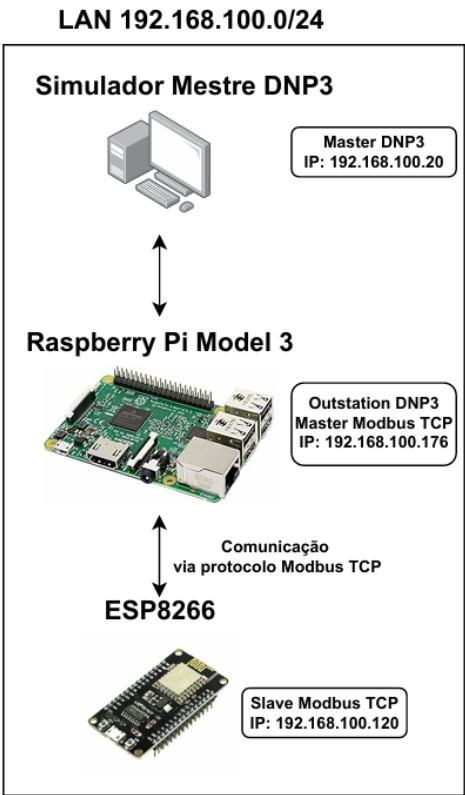
Fonte: Autoria própria.

Após a configuração do ambiente do mestre DNP3 com o Elipse SCADA, foram realizados os testes de comunicação com o gateway embarcado, confirmando a integridade e atualização dos dados em tempo real.

3.5.1.2 Mestre DNP3 utilizando simulador DNP3

De forma análoga ao descrito na Seção 3.5.1.1, a Figura 43 apresenta a topologia da arquitetura geral utilizando o simulador DNP3 da Kepware como mestre.

Figura 43 – Topologia utilizando simulador DNP3 como mestre.



Fonte: Autoria própria.

Da mesma forma, os pontos DNP3 definidos na Tabela 5 foram cadastrados no driver DNP3 da Kepware, como mostrado na Figura 44.

Figura 44 – Pontos DNP3 cadastrados no simulador da Kepware.

Tag Name	Address	Data Type	Scan Rate	Scaling
Analog Input (Potenciometro)	30.2.0.Value	Short	1000	None
Binary Input (Status Botao)	1.1.2.Value	Boolean	1000	None
Binary Input (Status Comunicacao)	1.1.0.Value	Boolean	1000	None
Binary Input (Status LED)	1.1.1.Value	Boolean	1000	None
Binary Output (LED OFF)	10.0.1 Operate.Set	Boolean	1000	None
Binary Output (LED ON)	10.0.0 Operate.Set	Boolean	1000	None

Fonte: Autoria própria.

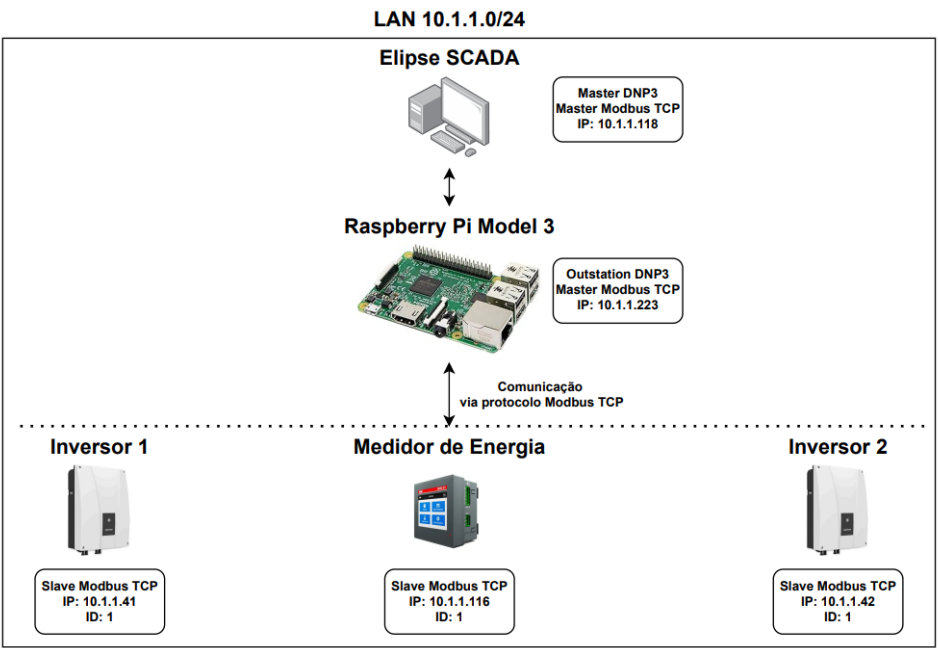
Ressalta-se que a varredura foi configurada para ocorrer a cada 1 segundo, e que os endereços de objeto, variação e índice também foram especificados, seguindo a sintaxe **OBJETO.VARIAÇÃO.ÍNDICE.TIPO**. Destaca-se ainda uma particularidade do software Kepware no tratamento de comandos CROB (Objeto 12, variação 1), em que os pontos de controle foram configurados como **10.0.0.ÍNDICE.Operate.Set**, conforme orientação do manual da ferramenta.

Posteriormente à configuração, foram realizados testes de comunicação visando verificar a correta varredura e controle dos dados, o que permitiu validar o comportamento da gateway desenvolvido.

3.5.2 Implementação 2

Para os testes de leitura DNP3 do Gateway desenvolvido no Tópico 3.3.2, foi utilizado apenas o Elipse SCADA, visando simplificar o desenvolvimento e obter uma abordagem mais visual da medição dos equipamentos. A arquitetura realizada pode ser visualizada na Figura 45.

Figura 45 – Pontos DNP3 cadastrados no simulador da Kepware.



Fonte: Autoria própria.

O Driver DNP3 Elipse utilizado anteriormente foi reaproveitado, apenas com a mudança do IP do escravo e o novo cadastro de pontos utilizando a Tabela 8 que pode ser observado na Figura 46.

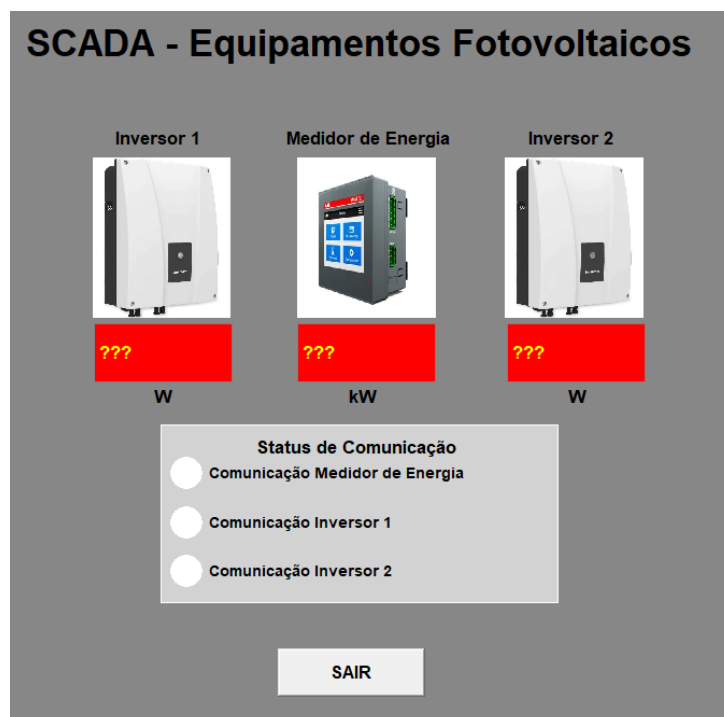
Figura 46 – Pontos DNP3 dos equipamentos fotovoltaicos cadastrados no driver do Elipse SCADA.

Nome	Dispo...	Item	P1/N1/B1	P2/N2/B2	P3/N3/B3	P4/N4/B4	Taman...	Varredura
DriverMestre			0	0	0	0		
DIGITAL								
status_communication1			0	1	102	0		1000
status_communication2			0	1	102	1		1000
status_communication3			0	1	102	2		1000
ANALOG								
analog_read1			0	1	3001	0		1000
analog_read2			0	1	3002	1		1000
analog_read3			0	1	3002	2		1000

Fonte: Autoria própria.

Após a configuração, foi elaborada uma nova tela de SCADA, visando a comunicação e visualização dos dados em tempo real. A tela desenvolvida pode ser contemplada através da Figura 47.

Figura 47 – Tela de testes no SCADA com equipamentos fotovoltaicos.



Fonte: Autoria própria.

Em suma, após toda a configuração, foram realizados testes de leitura dos pontos DNP3 com o objetivo de validar a segunda implementação do concentrador proposto neste trabalho.

3.6 Captura da rede para análise dos resultados

Com o objetivo de validar a correta operação do gateway embarcado entre os protocolos Modbus TCP e DNP3, foram realizadas capturas de tráfego de rede durante os testes práticos utilizando os métodos configurados nas seções anteriores. Essa etapa foi fundamental para verificar a integridade dos dados transmitidos, a conformidade com os protocolos utilizados e a correlação entre os eventos registrados nas diferentes interfaces de comunicação. A seguir, são apresentadas as ferramentas empregadas e a metodologia adotada para a captura e análise dos pacotes.

3.6.1 Ferramentas utilizadas

Para garantir uma análise completa e precisa do tráfego de rede, foram utilizadas as seguintes ferramentas:

Wireshark: Aplicativo amplamente utilizado para análise de pacotes em redes (FOUNDATION, 2025), permitindo a inspeção detalhada das mensagens DNP3. Com ele, foi possível observar os cabeçalhos dos pacotes, os tipos de objetos de dados transmitidos,

as mensagens do tipo `unsolicited response` e a sequência de confirmações exigida pelo protocolo.

Tcpdump: Ferramenta de linha de comando ([GROUP, 2025](#)), empregada diretamente na Raspberry Pi para capturar pacotes Modbus TCP em tempo real. Utilizou-se um filtro específico para monitorar apenas os pacotes direcionados à porta padrão do protocolo (502), assegurando foco nas comunicações relevantes ao experimento.

3.6.2 Metodologia de captura

A metodologia adotada para a captura do tráfego foi dividida conforme os dois protocolos utilizados na arquitetura do sistema:

Captura DNP3: O Wireshark foi executado em um computador conectado à mesma rede do mestre DNP3, implementado tanto nos casos da utilização do ambiente SCADA (Eclipse), quanto no uso do simulador da Kepware. O filtro aplicado durante a análise foi “`dnp3`”, de forma a restringir apenas aos pacotes DNP3. Dessa forma, foi possível avaliar se as respostas geradas pelo gateway estavam de acordo com as requisições do mestre e com os dados obtidos via Modbus TCP.

Captura Modbus TCP: Para monitorar o tráfego do protocolo Modbus TCP, utilizou-se o comando `tcpdump -i eth0 'tcp port 502' -w modbus.pcap`, executado diretamente no terminal da Raspberry Pi. Antes disso, foi necessária a instalação da ferramenta `tcpdump`, realizada com o seguinte comando:

Código 3.5 – Comando para instalação do `tcpdump` no terminal Linux

```
1 sudo apt update && sudo apt install tcpdump
```

Esse procedimento possibilitou a criação de um arquivo no formato `.pcap`, contendo todas as transações Modbus estabelecidas entre o gateway embarcado e os dispositivos simulados ou reais. Posteriormente, esse arquivo pôde ser analisado no Wireshark, permitindo a inspeção detalhada dos pacotes e a verificação da integridade da comunicação.

A combinação dessas ferramentas e da metodologia adotada possibilitou uma análise robusta das trocas de mensagens entre os protocolos, evidenciando o correto funcionamento do gateway, o atendimento às requisições do sistema SCADA e a fidelidade dos dados transmitidos entre os dispositivos envolvidos.

4 Resultados e Discussões

Nesta seção, apresentam-se os resultados obtidos com a implementação do **gateway embarcado Modbus TCP/DNP3**, validados através de testes práticos e análise de tráfego de rede. Os testes foram conduzidos seguindo as duas abordagens complementares desenvolvidas no capítulo 3:

- Ambiente simulado com a ESP8266 (validação funcional básica)
- Dispositivos reais de usinas fotovoltaicas (validação operacional)

4.1 Resultados do ambiente simulado

A primeira etapa de testes foi realizada com um **ESP8266** configurado como *escravo* Modbus TCP. Essa simulação foi escolhida por proporcionar um ambiente controlado, de fácil reprodução, ideal para ajustes iniciais e validação do funcionamento da conversão entre protocolos.

Conforme descrito no Capítulo 3, essa implementação foi testada por meio de duas abordagens distintas: a primeira utilizando o software Elipse SCADA como mestre DNP3, e a segunda utilizando o simulador Kepware.

4.1.1 Resultados com o mestre Elipse SCADA

Nessa topologia, o gateway embarcado (Raspberry Pi) atuou simultaneamente como:

- **Mestre Modbus TCP**, responsável por realizar leituras e escritas no ESP8266;
- **Estação remota DNP3 (Outstation)**, que disponibiliza os dados lidos ao mestre DNP3 (Elipse SCADA).

4.1.1.1 Análise do tráfego de rede

As capturas de tráfego realizadas no Wireshark, representadas nas Figuras 48 e 49, confirmaram o funcionamento esperado da comunicação entre os protocolos Modbus TCP e DNP3, validando a operação do gateway como intermediário entre os dispositivos de campo e o sistema SCADA.

Figura 48 – Tráfego Modbus TCP entre Raspberry Pi e ESP8266.

No.	Time	Source	Destination	Protocol	Length	Info
9	1.032374	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3515; Unit: 1, Func: 3: Read Holding Registers
10	1.043242	192.168.100.120	192.168.100.176	Modbus..	65	Response: Trans: 3515; Unit: 1, Func: 3: Read Holding Registers
12	1.043384	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3515; Unit: 1, Func: 1: Read Coils
13	1.054765	192.168.100.120	192.168.100.176	Modbus..	64	Response: Trans: 3515; Unit: 1, Func: 1: Read Coils
14	1.054874	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3517; Unit: 1, Func: 1: Read Coils
15	1.065697	192.168.100.120	192.168.100.176	Modbus..	64	Response: Trans: 3517; Unit: 1, Func: 1: Read Coils
17	2.066243	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3518; Unit: 1, Func: 3: Read Holding Registers
18	2.078825	192.168.100.120	192.168.100.176	Modbus..	65	Response: Trans: 3518; Unit: 1, Func: 3: Read Holding Registers
20	2.078170	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3519; Unit: 1, Func: 1: Read Coils
22	2.089955	192.168.100.120	192.168.100.176	Modbus..	64	Response: Trans: 3519; Unit: 1, Func: 1: Read Coils
23	2.089156	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3520; Unit: 1, Func: 1: Read Coils
24	2.099077	192.168.100.120	192.168.100.176	Modbus..	64	Response: Trans: 3520; Unit: 1, Func: 1: Read Coils
26	3.100429	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3521; Unit: 1, Func: 3: Read Holding Registers
27	3.111415	192.168.100.120	192.168.100.176	Modbus..	65	Response: Trans: 3521; Unit: 1, Func: 3: Read Holding Registers
29	3.111595	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3522; Unit: 1, Func: 1: Read Coils
30	3.122357	192.168.100.120	192.168.100.176	Modbus..	64	Response: Trans: 3522; Unit: 1, Func: 1: Read Coils
31	3.122466	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3523; Unit: 1, Func: 1: Read Coils
32	3.133525	192.168.100.120	192.168.100.176	Modbus..	64	Response: Trans: 3523; Unit: 1, Func: 1: Read Coils
34	4.134734	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3524; Unit: 1, Func: 3: Read Holding Registers
35	4.145309	192.168.100.120	192.168.100.176	Modbus..	65	Response: Trans: 3524; Unit: 1, Func: 3: Read Holding Registers
37	4.145434	192.168.100.176	192.168.100.120	Modbus..	66	Query: Trans: 3525; Unit: 1, Func: 1: Read Coils
38	4.155329	192.168.100.120	192.168.100.176	Modbus..	64	Response: Trans: 3525; Unit: 1, Func: 1: Read Coils

Fonte: Autoria própria.

Figura 49 – Tráfego DNP3 entre Raspberry Pi e Elipse SCADA.

No.	Time	Source	Destination	Protocol	Length	Info
203	7.989535	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
204	7.990539	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
205	7.991787	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
206	7.992690	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
207	7.993991	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Analog Input
208	7.994900	192.168.100.176	192.168.100.20	DNP 3.0	79	Response
214	8.987075	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
215	8.988253	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
216	8.988942	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
217	8.989960	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
218	8.990606	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
219	8.991485	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
220	8.992142	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Analog Input
221	8.993054	192.168.100.176	192.168.100.20	DNP 3.0	79	Response
238	9.987209	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
239	9.988440	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
240	9.989750	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
241	9.990704	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
242	9.992029	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
243	9.992962	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
247	9.994296	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Analog Input
248	9.995217	192.168.100.176	192.168.100.20	DNP 3.0	79	Response

Fonte: Autoria própria.

Na captura do tráfego Modbus TCP visualizado na Figura 48, o gateway atua como mestre Modbus, realizando as seguintes operações de leitura nos dispositivos escravos:

- Leitura de registradores (função 0x03), para obter valores analógicos;
- Leitura/escrita de *coils* (função 0x05), para estados binários.

Por outro lado, na captura do protocolo DNP3 observada na Figura 49, o mestre SCADA solicita os dados ao gateway, que agora atua como escravo DNP3. Os dados lidos via Modbus são convertidos e disponibilizados no padrão DNP3 da seguinte forma:

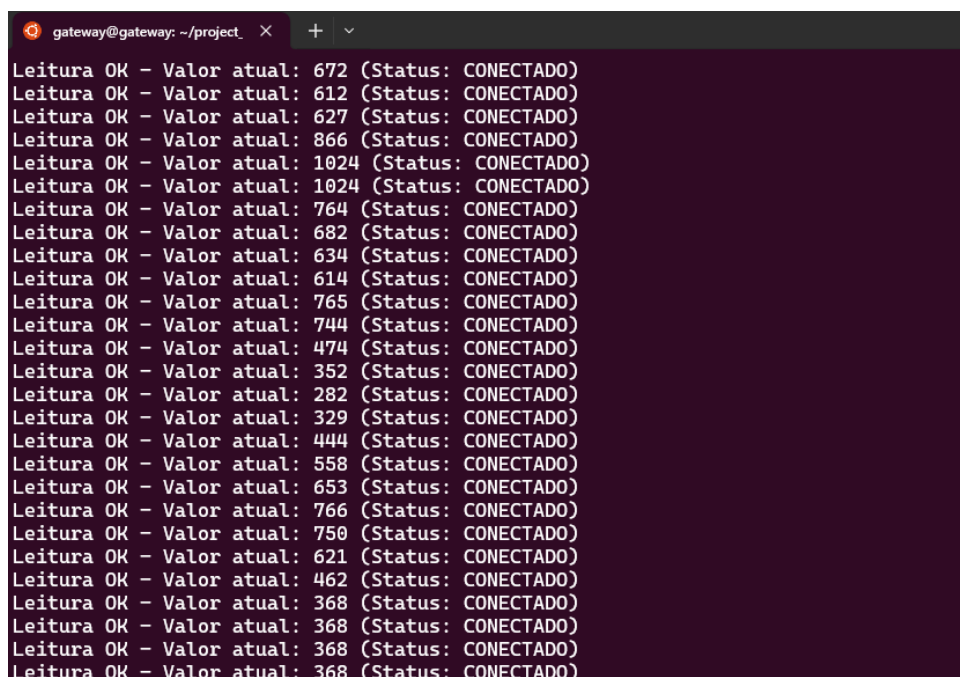
- Variáveis analógicas (valores lidos de registradores Modbus) são mapeadas para o tipo **Group30Var2**;
- Variáveis binárias (*coils* Modbus) são representadas como **Group1Var2**.

Essas capturas confirmam que a tradução entre os protocolos está ocorrendo conforme o esperado: os dados coletados via Modbus TCP são corretamente convertidos e disponibilizados no formato DNP3, garantindo a interoperabilidade entre os dispositivos de campo e o sistema SCADA.

4.1.1.2 Comunicação em tempo real

Além da análise de pacotes, foram observadas as mensagens no terminal da Raspberry Pi indicando o funcionamento contínuo do sistema e, em momentos simulados de perda de conexão, o acionamento do mecanismo de reconexão. Essas situações são exemplificadas nas Figuras 50 e 51.

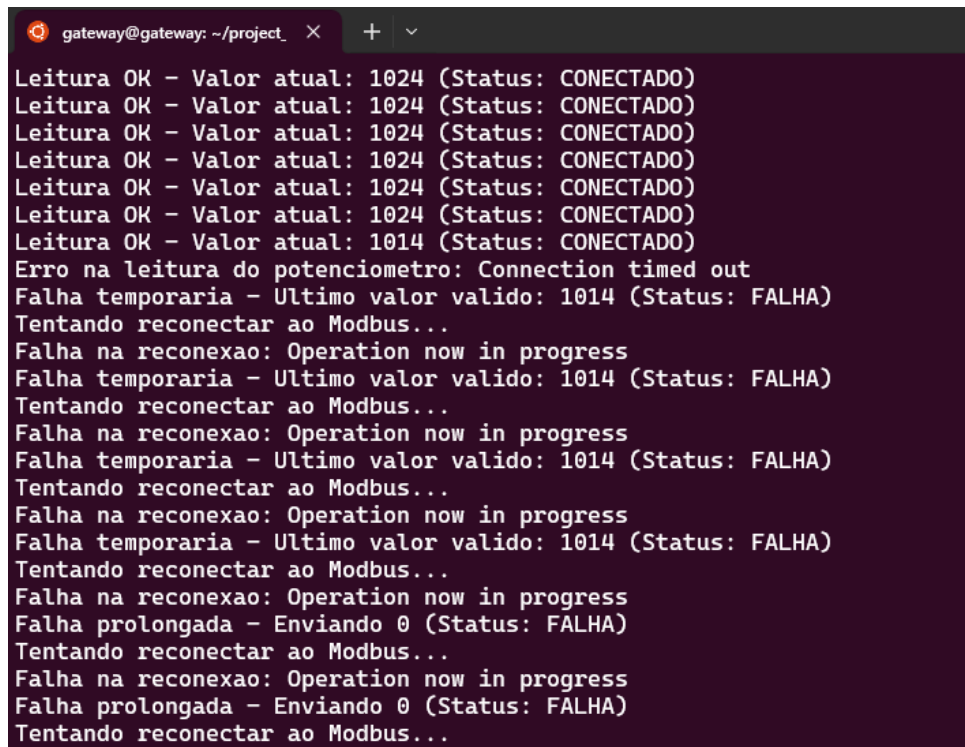
Figura 50 – Terminal reportando a comunicação em tempo real.

A screenshot of a terminal window with a dark background. The window title bar shows 'gateway@gateway: ~/project' and standard window controls. The terminal displays a list of 25 lines of text, each representing a status report. Each line follows the format 'Leitura OK - Valor atual: [number] (Status: CONECTADO)'. The values for 'Valor atual' are: 672, 612, 627, 866, 1024, 1024, 764, 682, 634, 614, 765, 744, 474, 352, 282, 329, 444, 558, 653, 766, 750, 621, 462, 368, 368, 368, and 368. The status is consistently 'CONECTADO' for all entries.

```
gateway@gateway: ~/project
Leitura OK - Valor atual: 672 (Status: CONECTADO)
Leitura OK - Valor atual: 612 (Status: CONECTADO)
Leitura OK - Valor atual: 627 (Status: CONECTADO)
Leitura OK - Valor atual: 866 (Status: CONECTADO)
Leitura OK - Valor atual: 1024 (Status: CONECTADO)
Leitura OK - Valor atual: 1024 (Status: CONECTADO)
Leitura OK - Valor atual: 764 (Status: CONECTADO)
Leitura OK - Valor atual: 682 (Status: CONECTADO)
Leitura OK - Valor atual: 634 (Status: CONECTADO)
Leitura OK - Valor atual: 614 (Status: CONECTADO)
Leitura OK - Valor atual: 765 (Status: CONECTADO)
Leitura OK - Valor atual: 744 (Status: CONECTADO)
Leitura OK - Valor atual: 474 (Status: CONECTADO)
Leitura OK - Valor atual: 352 (Status: CONECTADO)
Leitura OK - Valor atual: 282 (Status: CONECTADO)
Leitura OK - Valor atual: 329 (Status: CONECTADO)
Leitura OK - Valor atual: 444 (Status: CONECTADO)
Leitura OK - Valor atual: 558 (Status: CONECTADO)
Leitura OK - Valor atual: 653 (Status: CONECTADO)
Leitura OK - Valor atual: 766 (Status: CONECTADO)
Leitura OK - Valor atual: 750 (Status: CONECTADO)
Leitura OK - Valor atual: 621 (Status: CONECTADO)
Leitura OK - Valor atual: 462 (Status: CONECTADO)
Leitura OK - Valor atual: 368 (Status: CONECTADO)
Leitura OK - Valor atual: 368 (Status: CONECTADO)
Leitura OK - Valor atual: 368 (Status: CONECTADO)
Leitura OK - Valor atual: 368 (Status: CONECTADO)
```

Fonte: Autoria própria.

Figura 51 – Terminal reportando falha de comunicação com o escravo Modbus TCP.

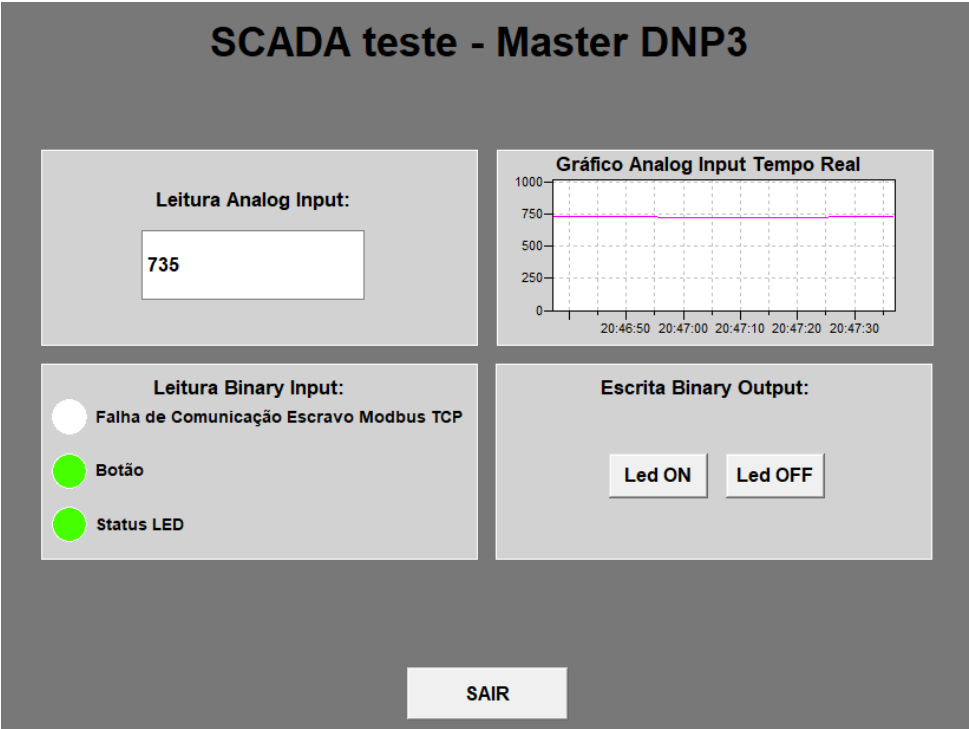


```
gateway@gateway: ~/project  x  +  v
Leitura OK - Valor atual: 1024 (Status: CONECTADO)
Leitura OK - Valor atual: 1024 (Status: CONECTADO)
Leitura OK - Valor atual: 1024 (Status: CONECTADO)
Leitura OK - Valor atual: 1024 (Status: CONECTADO)
Leitura OK - Valor atual: 1024 (Status: CONECTADO)
Leitura OK - Valor atual: 1024 (Status: CONECTADO)
Leitura OK - Valor atual: 1014 (Status: CONECTADO)
Erro na leitura do potenciometro: Connection timed out
Falha temporaria - Ultimo valor valido: 1014 (Status: FALHA)
Tentando reconectar ao Modbus...
Falha na reconexao: Operation now in progress
Falha temporaria - Ultimo valor valido: 1014 (Status: FALHA)
Tentando reconectar ao Modbus...
Falha na reconexao: Operation now in progress
Falha temporaria - Ultimo valor valido: 1014 (Status: FALHA)
Tentando reconectar ao Modbus...
Falha na reconexao: Operation now in progress
Falha temporaria - Ultimo valor valido: 1014 (Status: FALHA)
Tentando reconectar ao Modbus...
Falha na reconexao: Operation now in progress
Falha prolongada - Enviando 0 (Status: FALHA)
Tentando reconectar ao Modbus...
Falha na reconexao: Operation now in progress
Falha prolongada - Enviando 0 (Status: FALHA)
Tentando reconectar ao Modbus...
```

Fonte: Autoria própria.

Complementando os testes, a Figura 52 mostra a tela do Elipse SCADA, exibindo os dados DNP3 atualizados em tempo real, demonstrando que a leitura e a publicação dos dados estavam em pleno funcionamento.

Figura 52 – Tela do Elipse SCADA com pontos DNP3 ativos.



Fonte: Autoria própria.

4.1.2 Resultados com o mestre Kepware

Nesta topologia, o software Kepware foi utilizado no lugar do Elipse SCADA como mestre DNP3, permitindo a simulação do sistema supervisor e a verificação da comunicação com o gateway embarcado. A configuração foi realizada conforme descrito anteriormente no capítulo 3, com o correto mapeamento dos pontos DNP3 estabelecidos no gateway.

Durante os testes, observou-se que a comunicação entre o gateway e o simulador ocorreu conforme o esperado, com leitura e escrita dos pontos DNP3 ocorrendo de forma contínua e estável. Apesar do sucesso na validação da troca de dados, não foram realizadas capturas de pacotes com o Wireshark nesta etapa, com o objetivo de manter o foco nas evidências mais relevantes e otimizar a apresentação dos resultados.

As Figuras 53 e 54 demonstram o funcionamento correto da comunicação, com os pontos sendo atualizados em tempo real no ambiente do simulador Kepware.

Figura 53 – Tela do simulador Kepware com pontos DNP3 ativos.

Channel1.Device1.Analog Input (Potenciometro)	Short	668	20:38:08.556	Good	3
Channel1.Device1.Binary Input (Status Botao)	Boolean	1	20:38:05.556	Good	2
Channel1.Device1.Binary Input (Status Comunicacao)	Boolean	0	20:37:57.579	Good	1
Channel1.Device1.Binary Input (Status LED)	Boolean	0	20:37:57.579	Good	1
Channel1.Device1.Binary Output (LED OFF)	Boolean	0	20:37:57.579	Good	1
Channel1.Device1.Binary Output (LED ON)	Boolean	0	20:37:57.579	Good	1

Fonte: Autoria própria.

Figura 54 – Monitoramento dos valores lidos do gateway no ambiente do Kepware.

Channel1.Device1.Analog Input (Potenciometro)	Short	728	20:51:51.512	Good	2
Channel1.Device1.Binary Input (Status Botao)	Boolean	1	20:51:57.512	Good	2
Channel1.Device1.Binary Input (Status Comunicacao)	Boolean	0	20:51:40.527	Good	1
Channel1.Device1.Binary Input (Status LED)	Boolean	1	20:51:57.512	Good	2
Channel1.Device1.Binary Output (LED OFF)	Boolean	0	20:51:40.527	Good	1
Channel1.Device1.Binary Output (LED ON)	Boolean	0	20:51:53.512	Good	3

Fonte: Autoria própria.

4.1.3 Resultados dos comandos Direct Operate

A funcionalidade de comandos DNP3 do tipo **Direct Operate** foi validada com sucesso. Após o recebimento do comando por parte do mestre DNP3, o gateway interpreta a solicitação e executa a ação correspondente através da função `write_single_coil` do protocolo Modbus TCP, acionando ou desligando o LED conectado ao escravo Modbus.

Essa operação pôde ser confirmada por diferentes meios: via saída no terminal da Raspberry Pi, conforme mostrado na Figura 55, e também por meio das capturas de tráfego realizadas com o Wireshark. A Figura 56 ilustra a recepção do comando DNP3 pela Raspberry Pi, enquanto a Figura 57 mostra a correspondente escrita Modbus realizada logo em seguida.

Figura 55 – Recebimento de comandos no terminal da Raspberry Pi.

```
gateway@gateway: ~/project
Leitura OK - Valor atual: 678 (Status: CONECTADO)
Leitura OK - Valor atual: 678 (Status: CONECTADO)
Leitura OK - Valor atual: 678 (Status: CONECTADO)
Leitura OK - Valor atual: 678 (Status: CONECTADO)
Direct Operate: LED ON
Leitura OK - Valor atual: 678 (Status: CONECTADO)
Leitura OK - Valor atual: 670 (Status: CONECTADO)
Leitura OK - Valor atual: 687 (Status: CONECTADO)
Leitura OK - Valor atual: 701 (Status: CONECTADO)
Leitura OK - Valor atual: 764 (Status: CONECTADO)
Leitura OK - Valor atual: 751 (Status: CONECTADO)
Leitura OK - Valor atual: 757 (Status: CONECTADO)
Leitura OK - Valor atual: 750 (Status: CONECTADO)
Leitura OK - Valor atual: 750 (Status: CONECTADO)
Leitura OK - Valor atual: 756 (Status: CONECTADO)
Leitura OK - Valor atual: 756 (Status: CONECTADO)
Leitura OK - Valor atual: 756 (Status: CONECTADO)
Leitura OK - Valor atual: 756 (Status: CONECTADO)
Leitura OK - Valor atual: 750 (Status: CONECTADO)
Leitura OK - Valor atual: 750 (Status: CONECTADO)
Leitura OK - Valor atual: 756 (Status: CONECTADO)
Leitura OK - Valor atual: 756 (Status: CONECTADO)
Leitura OK - Valor atual: 750 (Status: CONECTADO)
Leitura OK - Valor atual: 750 (Status: CONECTADO)
Direct Operate: LED OFF
Leitura OK - Valor atual: 756 (Status: CONECTADO)
```

Fonte: Autoria própria.

Figura 56 – Recebimento do comando Direct Operate pelo mestre DNP3.

No.	Time	Source	Destination	Protocol	Length	Info
439	21.992010	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
440	21.992932	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
441	21.994203	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Analog Input
442	21.995101	192.168.100.176	192.168.100.20	DNP 3.0	79	Response
446	22.746889	192.168.100.20	192.168.100.176	DNP 3.0	89	Direct Operate
449	22.758198	192.168.100.176	192.168.100.20	DNP 3.0	91	Response
451	22.987151	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
452	22.988265	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
453	22.989618	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
454	22.990609	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
455	22.991881	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Binary Input
456	22.992771	192.168.100.176	192.168.100.20	DNP 3.0	77	Response
457	22.994044	192.168.100.20	192.168.100.176	DNP 3.0	76	Read, Analog Input


```

Frame 446: 89 bytes on wire (712 bits), 89 bytes captured (712 bits) on interface \Device\NPF_{4331BA3D-EC47-4...}
Ethernet II, Src: ASUSTekCOMPU_e7:d0:58 (d8:50:e6:e7:d0:58), Dst: RaspberryPiF_b2:5d:b5 (b8:27:eb:b2:5d:b5)
Internet Protocol Version 4, Src: 192.168.100.20, Dst: 192.168.100.176
Transmission Control Protocol, Src Port: 56591, Dst Port: 20000, Seq: 2280, Ack: 2594, Len: 35
Distributed Network Protocol 3.0
  Data Link Layer, Len: 26, From: 1, To: 2, DIR, PRM, Unconfirmed User Data
  Transport Control: 0xc0, Final, First(FIR, FIN, Sequence 0)
  Data Chunks
  [1 DNP 3.0 AL Fragment (20 bytes): #446(20)]
  Application Layer: (FIR, FIN, Sequence 11, Direct Operate)
    Application Control: 0xcb, First, Final(FIR, FIN, Sequence 11)
    Function Code: Direct Operate (0x05)
    DIRECT OPERATE Request Data Objects
      Object(s): Control Relay Output Block (Obj:12, Var:01) (0x0c01), 1 point
        Qualifier Field, Prefix: 2-Octet Index Prefix, Range: 16-bit Single Field Quantity
        Number of Items: 1
        Point Number 1 [Pulse On] [NUL]

```

Fonte: Autoria própria.

Figura 57 – Execução do comando via escrita em coil Modbus TCP.

No.	Time	Source	Destination	Protocol	Length	Info
146	17.559753	192.168.100.120	192.168.100.176	Modbus...	65	Response: Trans: 3564; Unit: 1, Func: 3: Read Holding Registers
148	17.559904	192.168.100.176	192.168.100.120	Modbus...	66	Query: Trans: 3565; Unit: 1, Func: 1: Read Coils
149	17.574208	192.168.100.120	192.168.100.176	Modbus...	64	Response: Trans: 3565; Unit: 1, Func: 1: Read Coils
150	17.574322	192.168.100.176	192.168.100.120	Modbus...	66	Query: Trans: 3566; Unit: 1, Func: 1: Read Coils
151	17.581619	192.168.100.120	192.168.100.176	Modbus...	64	Response: Trans: 3566; Unit: 1, Func: 1: Read Coils
153	18.282112	192.168.100.176	192.168.100.120	Modbus...	66	Query: Trans: 3567; Unit: 1, Func: 5: Write Single Coil
154	18.291678	192.168.100.120	192.168.100.176	Modbus...	66	Response: Trans: 3567; Unit: 1, Func: 5: Write Single Coil
156	18.582281	192.168.100.176	192.168.100.120	Modbus...	66	Query: Trans: 3568; Unit: 1, Func: 3: Read Holding Registers
157	18.595818	192.168.100.120	192.168.100.176	Modbus...	65	Response: Trans: 3568; Unit: 1, Func: 3: Read Holding Registers
159	18.595989	192.168.100.176	192.168.100.120	Modbus...	66	Query: Trans: 3569; Unit: 1, Func: 1: Read Coils
160	18.605050	192.168.100.120	192.168.100.176	Modbus...	64	Response: Trans: 3569; Unit: 1, Func: 1: Read Coils
161	18.605165	192.168.100.176	192.168.100.120	Modbus...	66	Query: Trans: 3570; Unit: 1, Func: 1: Read Coils
162	18.615901	192.168.100.120	192.168.100.176	Modbus...	64	Response: Trans: 3570; Unit: 1, Func: 1: Read Coils
164	19.616973	192.168.100.176	192.168.100.120	Modbus...	66	Query: Trans: 3571; Unit: 1, Func: 3: Read Holding Registers
165	19.628312	192.168.100.120	192.168.100.176	Modbus...	65	Response: Trans: 3571; Unit: 1, Func: 3: Read Holding Registers
167	19.628478	192.168.100.176	192.168.100.120	Modbus...	66	Query: Trans: 3572; Unit: 1, Func: 1: Read Coils


```

Frame 153: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{4331BA3D-EC47-4...}
Ethernet II, Src: RaspberryPiF_b2:5d:b5 (b8:27:eb:b2:5d:b5), Dst: Espressif_92:18:bd (e8:9f:6d:92:18:bd)
Internet Protocol Version 4, Src: 192.168.100.176, Dst: 192.168.100.120
Transmission Control Protocol, Src Port: 59400, Dst Port: 502, Seq: 661, Ack: 571, Len: 12
Modbus/TCP
  Modbus
    Function Code: Write Single Coil (5)
    Reference Number: 1
    Data: ff00
    Padding: 0x00

```

Fonte: Autoria própria.

Analisando as figuras apresentadas, observa-se que o comando é originado do endereço IP 192.168.100.20 (mestre DNP3) e destinado ao IP 192.168.100.176 (gateway). O gateway interpreta corretamente a requisição e, em seguida, realiza a escrita no dispositivo Modbus TCP, cujo IP de destino é 192.168.100.120. Esse fluxo de comunicação confirma o funcionamento esperado da cadeia de conversão de comandos no gateway embarcado.

4.1.4 Considerações finais da etapa simulada

Com base nas evidências apresentadas, é possível afirmar que a comunicação entre o gateway embarcado e o mestre DNP3 (Eclipse SCADA e simulador da Kepware) ocorreu

conforme o esperado. Todos os dados foram transmitidos corretamente, os comandos do tipo **Direct Operate** foram interpretados com sucesso, e as respostas do sistema puderam ser validadas tanto por ferramentas de rede quanto pela interface do supervisor e do simulador DNP3.

4.2 Resultados com dispositivos reais de UFVs

A segunda etapa de testes teve como objetivo validar a aplicação do gateway embarcado em um cenário mais próximo da realidade de uma usina fotovoltaica. Foram utilizados três dispositivos reais amplamente empregados em instalações de geração distribuída, conforme listados na Tabela 9.

Tabela 9 – Dispositivos reais integrados para testes

Dispositivo	Endereço IP	Tipo de Dado Modbus
Medidor ABB	10.1.1.116	Holding Register (32 bits)
Inversor 1	10.1.1.41	Input Register (16 bits)
Inversor 2	10.1.1.42	Input Register (16 bits)

Fonte: Autoria própria.

De forma análoga à implementação anterior, nesta topologia o gateway (Raspberry Pi) atuou como:

- **Mestre Modbus TCP** (para leitura nos dispositivos da Tabela 9)
- **Estação remota DNP3** (para disponibilizar dados ao Mestre Elipse SCADA)

4.2.1 Análise do tráfego

Durante os testes, o gateway embarcado realizou a leitura periódica dos três dispositivos, convertendo os dados Modbus TCP em pontos DNP3 disponíveis para o sistema supervisor. A Figura 58 mostra a captura do tráfego Modbus realizada no Wireshark, evidenciando as leituras em tempo real.

Figura 58 – Captura Wireshark: Leitura periódica Modbus TCP de dispositivos reais.

No.	Time	Source	Destination	Protocol	Length	Info
4	0.000657	10.1.1.223	10.1.1.116	Modbus...	66	Query: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
5	0.034485	10.1.1.116	10.1.1.223	Modbus...	67	Response: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
14	0.306700	10.1.1.223	10.1.1.41	Modbus...	78	Query: Trans: 1; Unit: 1, Func: 4: Read Input Registers
19	0.652243	10.1.1.223	10.1.1.42	Modbus...	78	Query: Trans: 1; Unit: 1, Func: 4: Read Input Registers
21	0.673840	10.1.1.41	10.1.1.223	Modbus...	77	Response: Trans: 1; Unit: 1, Func: 4: Read Input Registers
26	0.973661	10.1.1.42	10.1.1.223	Modbus...	77	Response: Trans: 1; Unit: 1, Func: 4: Read Input Registers
34	1.036284	10.1.1.223	10.1.1.116	Modbus...	66	Query: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
35	1.070297	10.1.1.116	10.1.1.223	Modbus...	67	Response: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
44	1.676674	10.1.1.223	10.1.1.41	Modbus...	78	Query: Trans: 1; Unit: 1, Func: 4: Read Input Registers
49	1.976570	10.1.1.223	10.1.1.42	Modbus...	78	Query: Trans: 1; Unit: 1, Func: 4: Read Input Registers
54	2.071576	10.1.1.223	10.1.1.116	Modbus...	66	Query: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
55	2.105182	10.1.1.116	10.1.1.223	Modbus...	67	Response: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
61	2.370155	10.1.1.41	10.1.1.223	Modbus...	77	Response: Trans: 1; Unit: 1, Func: 4: Read Input Registers
66	2.686926	10.1.1.42	10.1.1.223	Modbus...	77	Response: Trans: 1; Unit: 1, Func: 4: Read Input Registers
74	3.107059	10.1.1.223	10.1.1.116	Modbus...	66	Query: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
75	3.138725	10.1.1.116	10.1.1.223	Modbus...	67	Response: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
84	3.372884	10.1.1.223	10.1.1.41	Modbus...	78	Query: Trans: 1; Unit: 1, Func: 4: Read Input Registers
89	3.689635	10.1.1.223	10.1.1.42	Modbus...	78	Query: Trans: 1; Unit: 1, Func: 4: Read Input Registers
91	3.692389	10.1.1.41	10.1.1.223	Modbus...	77	Response: Trans: 1; Unit: 1, Func: 4: Read Input Registers
96	4.021011	10.1.1.42	10.1.1.223	Modbus...	77	Response: Trans: 1; Unit: 1, Func: 4: Read Input Registers
104	4.140541	10.1.1.223	10.1.1.116	Modbus...	66	Query: Trans: 1; Unit: 1, Func: 3: Read Holding Registers
105	4.183033	10.1.1.116	10.1.1.223	Modbus...	67	Response: Trans: 1; Unit: 1, Func: 3: Read Holding Registers

Fonte: Autoria própria.

Foram observadas requisições contínuas nas portas Modbus TCP (502), alternando entre os três IPs configurados, com tempo de leitura média de aproximadamente 1 segundo por ciclo.

Também, foram analisadas as capturas dos pacotes DNP3, onde é possível visualizar o SCADA realizando as varreduras dos dados. A captura da comunicação DNP3 pode ser visualizada na Figura 59.

Figura 59 – Captura Wireshark: Troca de dados entre o SCADA Elipse e o gateway.

No.	Time	Source	Destination	Protocol	Length	Info
211	7.491398	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Analog Input
213	7.496271	10.1.1.223	10.1.1.118	DNP 3.0	81	Response
216	7.539692	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Analog Input
217	7.541708	10.1.1.223	10.1.1.118	DNP 3.0	79	Response
218	7.543283	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Binary Input
219	7.545340	10.1.1.223	10.1.1.118	DNP 3.0	77	Response
220	7.546636	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Binary Input
221	7.548518	10.1.1.223	10.1.1.118	DNP 3.0	77	Response
222	7.549912	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Binary Input
223	7.553234	10.1.1.223	10.1.1.118	DNP 3.0	77	Response
224	7.554668	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Analog Input
225	7.558367	10.1.1.223	10.1.1.118	DNP 3.0	79	Response
230	8.391035	10.1.1.223	10.1.1.118	DNP 3.0	85	Unsolicited Response
231	8.392604	10.1.1.118	10.1.1.223	DNP 3.0	69	Confirm
234	8.485446	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Analog Input
236	8.491784	10.1.1.223	10.1.1.118	DNP 3.0	81	Response
237	8.493479	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Analog Input
238	8.495260	10.1.1.223	10.1.1.118	DNP 3.0	79	Response
239	8.496096	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Binary Input
240	8.499664	10.1.1.223	10.1.1.118	DNP 3.0	77	Response
241	8.500553	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Binary Input
243	8.501671	10.1.1.223	10.1.1.118	DNP 3.0	85	Unsolicited Response
244	8.502548	10.1.1.118	10.1.1.223	DNP 3.0	69	Confirm
245	8.504276	10.1.1.223	10.1.1.118	DNP 3.0	77	Response
246	8.505289	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Binary Input
247	8.508196	10.1.1.223	10.1.1.118	DNP 3.0	77	Response
248	8.509128	10.1.1.118	10.1.1.223	DNP 3.0	76	Read, Analog Input
249	8.515780	10.1.1.223	10.1.1.118	DNP 3.0	79	Response

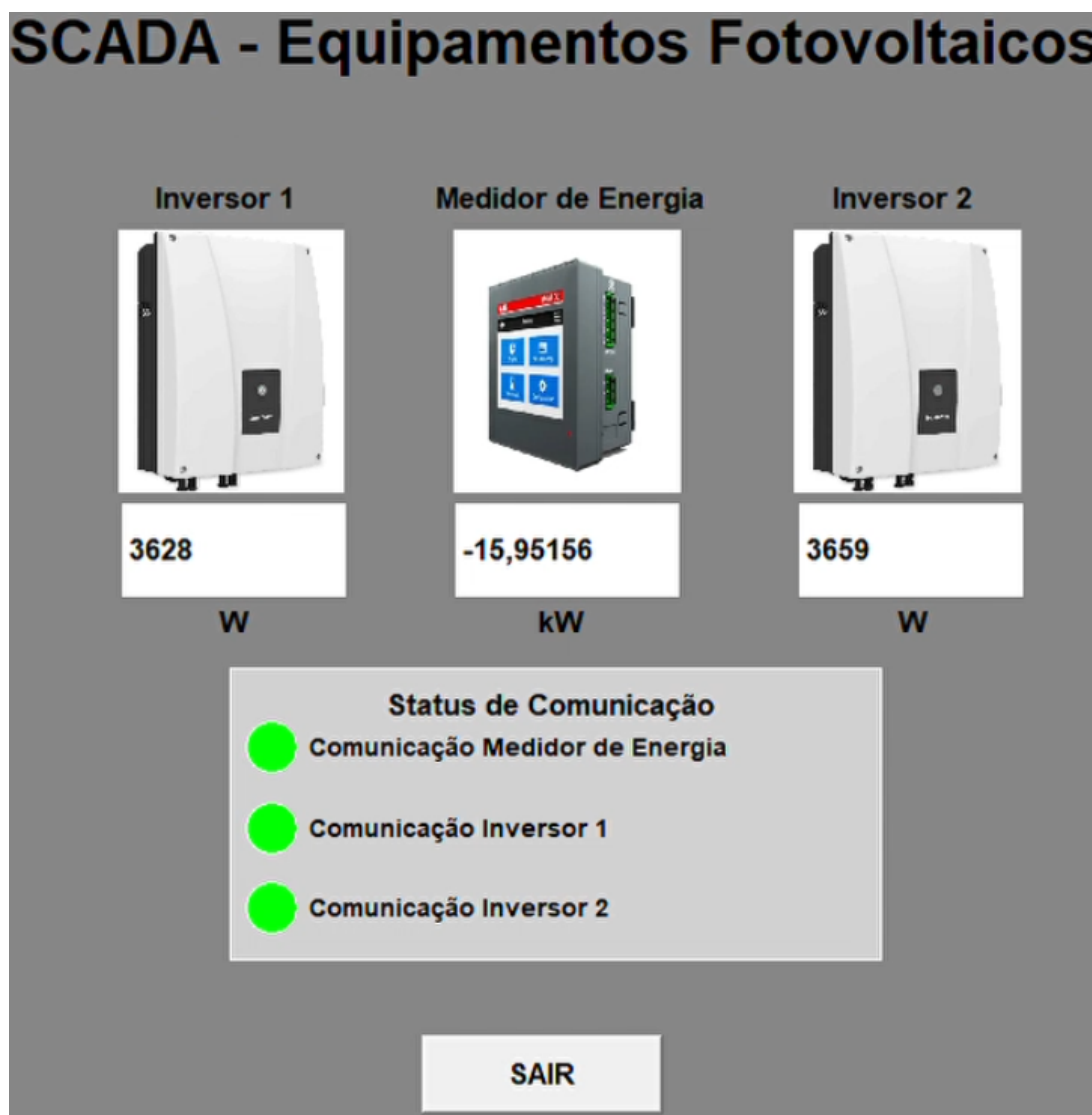
Fonte: Autoria própria.

É importante destacar que, na Figura 59, observa-se a transmissão de dados do gateway para o mestre DNP3 de forma não solicitada. Esse comportamento resulta da ativação da funcionalidade de envio de mensagens **unsolicited** na configuração da estação remota DNP3, conforme discutido nas seções do Capítulo 3. Essa funcionalidade permite que o dispositivo envie atualizações espontâneas ao mestre, sem a necessidade de uma requisição prévia, otimizando a atualização de variáveis críticas em sistemas SCADA.

4.2.2 Apresentação dos dados em interface SCADA

A Figura 60 apresenta a tela do sistema SCADA Elipse comunicando com os dados provenientes dos três dispositivos reais, já convertidos para o protocolo DNP3.

Figura 60 – Interface SCADA exibindo dados DNP3 de dispositivos reais.

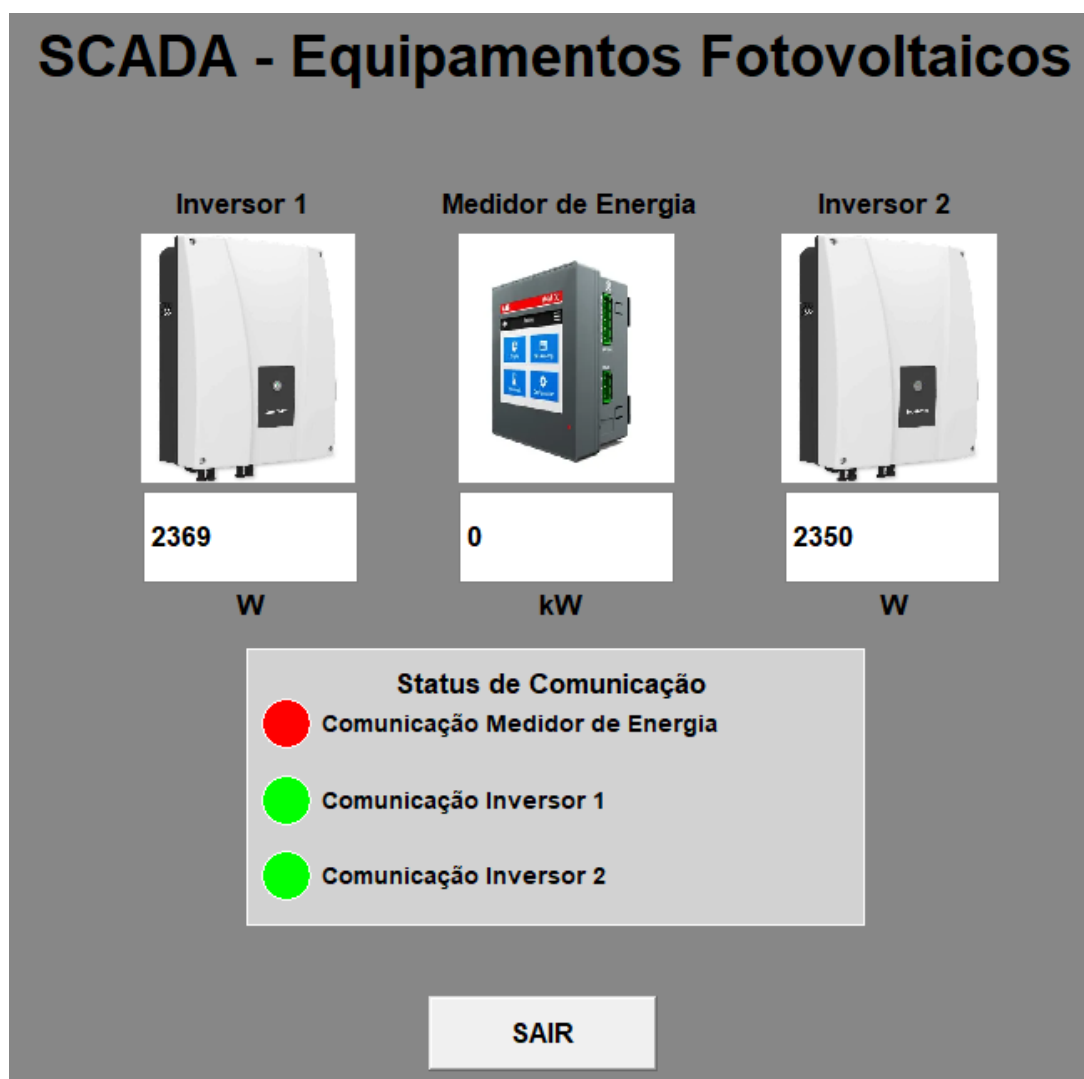


Fonte: Autoria própria.

4.2.3 Estabilidade e tratamento de falhas

Durante os testes, o sistema se manteve estável mesmo diante de falhas simuladas nos dispositivos. A Figura 61 mostra o comportamento do gateway quando um dos dispositivos fica inacessível por mais de cinco tentativas consecutivas. Nesse caso, o valor analógico correspondente é substituído por zero e o estado de falha é refletido no ponto binário DNP3.

Figura 61 – Tratamento de falha: Substituição do valor Modbus após falhas consecutivas.



Fonte: Autoria própria.

Esse comportamento demonstra a robustez da lógica embarcada no gateway, garantindo que o sistema supervisório receba dados coerentes, mesmo em condições adversas de comunicação.

4.2.4 Conclusão da validação com dispositivos reais

Com a integração bem-sucedida dos três dispositivos reais e a correta publicação dos dados via protocolo DNP3, foi possível validar a aplicação do gateway embarcado em ambientes industriais. A estabilidade observada, o tratamento eficiente de falhas e a compatibilidade com sistemas SCADA consolidados demonstram a viabilidade da solução para o monitoramento de usinas fotovoltaicas de geração distribuída.

5 Conclusão

O presente trabalho teve como objetivo o desenvolvimento de um gateway embarcado para integração entre dispositivos que utilizam o protocolo Modbus TCP e sistemas supervisórios que operam com o protocolo DNP3, com foco em aplicações no setor de geração fotovoltaica distribuída. A proposta foi motivada pela dificuldade de integração entre equipamentos de diferentes fabricantes, que comumente não oferecem suporte ao protocolo DNP3, exigindo soluções de conversão geralmente onerosas.

Para isso, foi desenvolvida uma solução baseada em uma Raspberry Pi, utilizando as bibliotecas `OpenDNP3` e `libmodbus`, com toda a lógica implementada em linguagem C++. Foram realizadas duas abordagens complementares para validação do sistema: a primeira com um ambiente simulado utilizando uma ESP8266 configurada como escravo Modbus TCP, e a segunda com dispositivos reais comumente encontrados em usinas fotovoltaicas, como inversores e medidores de energia.

Os testes comprovaram a eficácia da solução, evidenciada tanto por capturas de tráfego em ferramentas como o Wireshark, quanto pela correta atualização e controle das variáveis no ambiente supervisório. Os comandos DNP3 do tipo *Direct Operate* também foram processados corretamente, demonstrando a bidirecionalidade da comunicação e a capacidade de controle do gateway. Além disso, os testes demonstraram que o gateway embarcado atendeu ao objetivo proposto, fornecendo uma solução de baixo custo, confiável e eficiente para integração de equipamentos Modbus TCP com sistemas SCADA via DNP3. A implementação mostrou-se escalável para mais dispositivos, limitando-se apenas pela capacidade de hardware da Raspberry Pi e pela otimização do tempo de varredura.

Além da funcionalidade validada, a solução se mostrou viável economicamente, apresentando um custo significativamente inferior ao de gateways comerciais, sem comprometer a robustez exigida por aplicações industriais.

Como continuidade deste trabalho, propõe-se o desenvolvimento de uma interface gráfica baseada em web para permitir a configuração dinâmica dos dispositivos escravos Modbus TCP e o mapeamento dos pontos DNP3, eliminando a necessidade de modificar diretamente o código-fonte. Essa interface possibilitaria o cadastro e exclusão de novos dispositivos Modbus, a associação de registradores e coils a índices DNP3, a seleção de tipos de dados, classes e variações do protocolo, além do monitoramento em tempo real do estado dos dispositivos. A incorporação dessa funcionalidade aumentaria significativamente a usabilidade e a flexibilidade da solução proposta, tornando-a mais atrativa para aplicações comerciais e industriais que exigem agilidade, escalabilidade e simplicidade na configuração de redes de automação.

Referências

ABB. *REC615 and RER615 DNP3 Communication Protocol Manual*. 2.0. ed. [S.l.], 2021. Acesso em: 21 abr. 2025. Disponível em: <<https://techdoc.relays.protection-control.abb/r/REC615-and-RER615-DNP3-Communication-Protocol-Manual/2.0/en-US/Binary-outputs-and-control-relay-output-block>>. Citado na página 26.

ABGD, A. B. D. G. D. Geração própria de energia atinge 30 gw no brasil. 2024. Disponível em: <<https://www.abgd.com.br/portal/geracao-propria-de-energia-atinge-30-gw-no-brasil/>>. Acesso em: 12 de fevereiro de 2025. Citado na página 13.

BANERJI, S. *Study and development of a data acquisition & control (DAQ) system using TCP/Modbus protocol*. 2013. Citado na página 19.

CALDIERI, M. R. *Implementação do Modbus para aplicação em sistemas de controle via rede sem fio*. Dissertação (Mestrado) — Universidade Estadual Paulista "Júlio de Mesquita Filho", Bauru, SP, Brasil, 2016. Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica. Citado na página 13.

CONSTAIN, N. B. P. *Integração de sistemas SCADA com a implementação de controle supervisorio em CLP para sistemas de manufatura*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina, Florianópolis, 2011. Dissertação (Mestrado em Engenharia de Automação e Sistemas). Citado na página 17.

Dell Technologies. *What is an industrial gateway?* 2024. Acesso em: 22 dez. 2024. Disponível em: <<https://infohub.delltechnologies.com/pt-br/1/dell-validated-design-for-manufacturing-edge-with-ptc-design-guide/what-is-an-industrial-gateway-7/>>. Citado na página 27.

DNP Users Group. *DNP3 Primer, Revision A*. 2005. <<https://www.dnp.org/Portals/0/AboutUs/DNP3%20Primer%20Rev%20A.pdf>>. Acesso em: 22 de dezembro de 2024. Citado 3 vezes nas páginas 22, 23 e 24.

DNP Users Group. *Overview of DNP3 Protocol*. 2024. Acesso em: 22 dez. 2024. Disponível em: <<https://www.dnp.org/About/Overview-of-DNP3-Protocol>>. Citado na página 21.

ELIPSE. *Eclipse E3 Software*. 2023. <<https://www.elipse.com.br/produto/elipse-e3/>>. Acesso em: 15 abr. 2025. Citado na página 54.

ELIPSE. *Eclipse E3 monitora o desempenho das usinas solares da Clemar Engenharia*. 2024. Acesso em: 15 abr. 2025. Disponível em: <<https://www.elipse.com.br/case/elipse-e3-monitora-o-desempenho-das-usinas-solares-da-clemar-engenharia/>>. Citado na página 16.

Elipse Software. *Soluções da Elipse conferem agilidade à operação da Elektro*. 2024. Disponível em: <<https://www.elipse.com.br/case/solucoes-da-elipse-conferem-agilidade-a-operacao-da-elektro/>>. Acesso em: 26 de dezembro de 2024. Citado 2 vezes nas páginas 17 e 18.

- Ezenroute. *IoT Gateway*. 2024. Disponível em: <https://www.ezenroute.com/iot_gateway.php>. Acesso em: 22 dez. 2024. Citado na página 28.
- FOUNDATION, W. *Wireshark: the world's foremost and widely-used network protocol analyzer*. 2025. <<https://www.wireshark.org>>. Acesso em: 25 abr. 2025. Citado na página 59.
- GROUP, D. U. DNP3 introduction. In: *Event Name*. [s.n.], 2015. Acesso em: 21 abr. 2025. Disponível em: <https://na.eventscloud.com/file_uploads/b68188f3ce5b22895a67b1afe5e51b6a_DNP3IntroductionHORS.PDF>. Citado na página 27.
- GROUP, T. T. *Tcpdump & libpcap*. 2025. <<https://www.tcpdump.org>>. Acesso em: 25 abr. 2025. Citado na página 60.
- Grupo MTR. *MTR monitora usinas da Ineer Energia com tecnologia nacional*. 2024. <<https://www.pv-magazine-brasil.com/2024/09/03/grupo-mtr-fecha-acordo-para-monitorar-132-mw-de-usinas-da-ineer-energia/>>. Acesso em: 15 abr. 2025. Citado na página 16.
- HUAWEI. *Smart PV Management SystemSmartLogger*. 2025. Disponível em: <<https://support.huawei.com/enterprise/br/portuguese-documents/smartlogger-pid-21294677>>. Acesso em: 09 de abril de 2025. Citado na página 32.
- INC., P. *O que é o KEPServerEX*. 2023. <<https://www.kepserverex.pt/o-que-e-o-kepserverex-como-funciona-ecomo-pode-ajuda-lo/>>. Acesso em: 15 abr. 2025. Citado na página 54.
- K, F. *NodeMCU ESP8266: Detalhes e Pinagem*. 2018. Acesso em: 22 dez. 2024. Disponível em: <<https://www.fernandok.com/2018/05/nodemcu-esp8266-detalhes-e-pinagem.html>>. Citado na página 29.
- MACIAS, J. et al. Diseño e implementación de un gateway iot multiprotocolo. *Vínculos: Ciencia, Tecnología y Sociedad*, 2021. Enviado em: 16 jan. 2021. Recebido em: 12 fev. 2021. Aceito em: 26 mai. 2021. Citado na página 28.
- MakerHero. *Raspberry Pi 3 Model B+*. 2024. <<https://www.makerhero.com/produto/raspberry-pi-3-model-b/>>. Acesso em: 22 dez. 2024. Citado 2 vezes nas páginas 30 e 31.
- MESTANZA, T. F. C.; FERNANDES, R. F. Desenvolvimento e implementação de um gateway comunicador entre protocolos modbus tcp/rtu. *Conferência de Estudos em Engenharia Elétrica (CEEL)*, 2022. Citado na página 13.
- Modbus Organization. *Modbus Messaging Implementation Guide V1.0b*. 2006. Disponível em: <https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf>. Acesso em: 19 de dezembro de 2024. Citado 2 vezes nas páginas 19 e 20.
- MODBUS Organization. *Modbus Application Protocol Specification V1.1b3*. 2012. Acesso em: 19 dez. 2024. Disponível em: <https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>. Citado 2 vezes nas páginas 20 e 21.
- Modbus Organization. *Modbus FAQ*. 2024. Acesso em: 19 dez. 2024. Disponível em: <<https://modbus.org/faq.php>>. Citado 2 vezes nas páginas 18 e 19.

PcVue Solutions. *DNP3 Inputs and Outputs, Object Groups and Variations*. [S.l.], 2024. Acesso em: 22 abr. 2025. Citado na página 26.

PEDRO. *ESP8266 Pinout: Quais GPIOs devo usar?* 2024. Acesso em: 22 dez. 2024. Disponível em: <<https://clubedomaker.com/esp8266-pinout>>. Citado na página 29.

Portal Solar. *Usina Solar: o que é, como funciona e como construir a sua*. 2024. Disponível em: <<https://www.portalsolar.com.br/usina-solar.html>>. Acesso em: 19 de dezembro de 2024. Citado 2 vezes nas páginas 15 e 16.

RAIMBAULT, S. *libmodbus - A Modbus library for Linux, Mac OS, FreeBSD and Windows*. 2023. <<https://libmodbus.org/>>. Acessado em: 02 mai. 2025. Citado na página 35.

Raspberry Pi Foundation. *Raspberry Pi 3 Model B+ Product Brief*. [S.l.], 2018. Acesso em: 22 dez. 2024. Disponível em: <<https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf>>. Citado 2 vezes nas páginas 30 e 31.

Raspberry Pi Foundation. *Raspberry Pi Foundation Annual Review 2019*. 2019. Disponível em: <<https://static.raspberrypi.org/files/about/RaspberryPiFoundationReview2019.pdf>>. Acesso em: 22 dez. 2024. Disponível em: <<https://static.raspberrypi.org/files/about/R>>. Citado na página 30.

Raspberry Pi Foundation. *Raspberry Pi Imager: Versão 1.8.4*. 2025. <<https://www.raspberrypi.com/software/>>. Acesso em: 15 abr. 2025. Citado na página 41.

Real Time Automation, Inc. *DNP3 Overview*. 2024. Disponível em: <<https://www.rtautomation.com/technologies/dnp3-overview/>>. Acesso em: 22 abr. 2025. Disponível em: <<https://www.rtautomation.com/technologies/dnp3-overview/>>. Citado na página 25.

Robotics University. *Introducing Raspberry Pi 3 Model B+*. 2018. Disponível em: <<https://www.robotics-university.com/2018/06/introducing-raspberry-pi-3-model-bplus.html>>. Acesso em: 22 dez. 2024. Citado na página 30.

RODRIGUES, E.; PEDÓ, R.; TEDESCO, L. P. Sistemas embarcados e sua aplicação na indústria. In: *Anais do II WSPI – Workshop em Sistemas e Processos Industriais*. Santa Cruz do Sul, RS: Universidade de Santa Cruz do Sul (UNISC), 2013. 8–10 maio 2013. Citado na página 28.

SANTOS, J. A. F. d. A.; CUNHA, F. B. F.; TORRES, E. A. Geração distribuída brasileira: aspectos regulatórios, evolução e estudo de caso em Juazeiro/ba. In: PRIMO, R. G. B.; KÁLID, R. d. A. (Ed.). *Aporias no Desenvolvimento da América Latina*. 1. ed. Barcelona-ES; Salvador-BR: CEALA - Centro de Estudios por la Amistad de Latinoamérica, Asia y África, 2021. único, p. 180–209. Citado na página 15.

Schneider Electric. *DNP3 Driver Guide: Unsolicited Messages*. [S.l.], 2020. Acesso em: 21 abr. 2025. Citado na página 25.

SIGGINS, M. How to understand the dnp3 protocol. 2020. Disponível em: <<https://www.dpstele.com/blog/how-to-understand-dnp3-protocol.php>>. Acesso em: 19 de dezembro de 2024. Citado na página 13.

SYSTEMS, E. *ESP8266 Wi-Fi SoC*. 2024. Acesso em: 22 dez. 2024. Disponível em: <<https://www.espressif.com/en/products/socs/esp8266>>. Citado 2 vezes nas páginas 28 e 29.

THOMAS, G. Introduction to the modbus protocol. *Contemporary Control Systems*, Contemporary Control Systems, Inc., v. 9, n. 4, jul.-ago. 2008. © 2008 Contemporary Control Systems, Inc. Citado na página 21.

TIBURSKI, G.; MOREIRA, G. T.; MISAGHI, M. Supervisory systems integration SCADA and ERP for production control in real time. *Revista Espacios*, v. 38, n. 4, p. 5, 2017. Received: 08 Aug. 2016. Approved: 28 Aug. 2016. Citado na página 18.

URZÊDA, C. C. *Software SCADA como Plataforma para Racionalização Inteligente de Energia Elétrica em Automação Predial*. 96 p. Dissertação (Dissertação de Mestrado em Engenharia Elétrica), Brasília, DF, 2006. Publicação PPGENE.DM-281/06. Citado na página 17.

VARGAS, R. S. *Sistemas embarcados: acoplamento do soft-core Plasma ao barramento OPB de um PowerPC 405*. 2007. 100 f. Monografia (Bacharelado em Ciências da Computação) — Universidade Federal de Santa Catarina, Florianópolis, 2007. Citado na página 28.

WIN-TECH. *WIN-TECH: Home*. 2025. Acesso em: 07 abr. 2025. Disponível em: <<https://www.win-tech.com/>>. Citado na página 40.

ANEXO A – Código - ESP8266

Código A.1 – Escravo Modbus TCP implementado com uma ESP8266

```

1
2 // Inclui bibliotecas necessarias
3 #include <Arduino.h>           // Biblioteca basica do Arduino
4 #include <ESP8266WiFi.h>       // Para conexao WiFi no ESP8266
5 #include <ModbusIP_ESP8266.h> // Para protocolo Modbus via IP
6
7 // Configuracoes de rede
8 const IPAddress ip(192, 168, 100, 120);    // IP fixo do dispositivo
9 const IPAddress gateway(192, 168, 100, 1);  // Gateway da rede
10 const IPAddress subnet(255, 255, 255, 0);  // Mascara de rede
11 const char* ssid = "casa.exe";           // Nome da rede WiFi
12 const char* password = "AfxJ457T";      // Senha da rede WiFi
13
14 // Definicoes dos registradores Modbus
15 const uint16_t COIL_LIGAR = 0;           // Coil para ligar LED (endereço 00001)
16 const uint16_t COIL_DESLIGAR = 1;       // Coil para desligar LED (endereço 00002)
17 const uint16_t COIL_STATUS_LED = 2;     // Coil para status do LED (endereço
    00003)
18 const uint16_t COIL_BOTAO_D0 = 3;       // Coil para estado do botao (endereço
    00004)
19 const uint16_t HREG_POT = 0;            // Registrador para valor do potenciometro
20
21 // Definicoes dos pinos
22 const uint8_t LED_PIN = D2;             // Pino do LED
23 const uint8_t POT_PIN = A0;             // Pino do potenciometro
24 const uint8_t BUTTON_PIN = D0;         // Pino do botao
25
26 ModbusIP mb; // Objeto ModbusIP
27
28 // Variaveis para debounce do botao
29 bool lastButtonState = HIGH;             // Ultimo estado do botao
30 unsigned long lastDebounceTime = 0;      // Tempo do ultimo debounce
31 const unsigned long debounceDelay = 50;  // Tempo de debounce em ms
32
33 void setup() {
34     Serial.begin(115200); // Inicia serial com baud rate 115200
35
36     // Configura pinos
37     pinMode(LED_PIN, OUTPUT);             // Define pino do LED como saida
38     digitalWrite(LED_PIN, LOW);          // Inicia LED desligado
39     pinMode(BUTTON_PIN, INPUT_PULLUP);    // Define pino do botao como
        entrada com pull-up
40
41     // Configura conexao WiFi
42     WiFi.config(ip, gateway, subnet);     // Define configuracao de rede
43     WiFi.begin(ssid, password);          // Conecta na rede WiFi
44     while (WiFi.status() != WL_CONNECTED) { // Espera conexao
45         delay(500);
46         Serial.print(".");
47     }

```

```

48     Serial.println("\nConectado! IP: " + WiFi.localIP().toString());
49
50     mb.server(); // Inicia servidor Modbus
51
52     // Adiciona registradores Modbus
53     mb.addCoil(COIL_LIGAR); // Coil para ligar LED
54     mb.addCoil(COIL_DESLIGAR); // Coil para desligar LED
55     mb.addCoil(COIL_STATUS_LED); // Coil para status do LED
56     mb.addCoil(COIL_BOTAO_DO); // Coil para estado do botao
57     mb.addHreg(HREG_POT); // Registrador para valor do potenciometro
58
59     // Callbacks para alteracao das coils
60     mb.onSetCoil(COIL_LIGAR, [](TRegister* reg, uint16_t val) {
61         if (val) { // Se coil for ativada
62             digitalWrite(LED_PIN, HIGH); // Liga LED
63             mb.Coil(COIL_STATUS_LED, true); // Atualiza status do LED
64             Serial.println("Ligado via coil 00001");
65             mb.Coil(COIL_LIGAR, false); // Reseta coil
66         }
67         return val;
68     });
69
70     mb.onSetCoil(COIL_DESLIGAR, [](TRegister* reg, uint16_t val) {
71         if (val) { // Se coil for ativada
72             digitalWrite(LED_PIN, LOW); // Desliga LED
73             mb.Coil(COIL_STATUS_LED, false); // Atualiza status do LED
74             Serial.println("Desligado via coil 00002");
75             mb.Coil(COIL_DESLIGAR, false); // Reseta coil
76         }
77         return val;
78     });
79 }
80
81 void loop() {
82     mb.task(); // Processa requisicoes Modbus
83
84     // Leitura periodica do potenciometro
85     static uint32_t lastUpdate = 0;
86     if (millis() - lastUpdate >= 500) { // A cada 500ms
87         lastUpdate = millis();
88         uint16_t val = analogRead(POT_PIN); // Le valor do potenciometro
89
90         // Atualiza registrador se diferenca maior que 5 (para evitar ruido)
91         if (abs(val - mb.Hreg(HREG_POT)) > 5) {
92             mb.Hreg(HREG_POT, val); // Atualiza registrador
93             Serial.println("Pot: " + String(val));
94         }
95     }
96
97     // Leitura do botao com debounce
98     bool reading = digitalRead(BUTTON_PIN); // Le estado atual do botao
99
100     if (reading != lastButtonState) { // Se estado mudou
101         lastDebounceTime = millis(); // Reseta tempo do debounce
102     }
103
104     // Apos tempo de debounce, atualiza coil do botao

```

```
105     if ((millis() - lastDebounceTime) > debounceDelay) {  
106         mb.Coil(COIL_BOTA0_D0, !reading); // Inverte logica (pull-up)  
107     }  
108  
109     lastButtonState = reading; // Atualiza ultimo estado do botao  
110     delay(10); // Pequeno delay para estabilidade  
111 }
```

ANEXO B – Código - Gateway 1ª Implementação

Código B.1 – Implementação 1 - Estação Remota DNP3 integrada a estação de testes com ESP8266

```

1
2 // Bibliotecas necessarias para comunicacao DNP3 e Modbus
3 #include <opendnp3/ConsoleLogger.h>
4 #include <opendnp3/DNP3Manager.h>
5 #include <opendnp3/logging/LogLevels.h>
6 #include <opendnp3/outstation/DefaultOutstationApplication.h>
7 #include <opendnp3/outstation/IUpdateHandler.h>
8 #include <opendnp3/outstation/UpdateBuilder.h>
9 #include <opendnp3/outstation/OutstationStackConfig.h>
10 #include <opendnp3/outstation/SimpleCommandHandler.h>
11 #include <opendnp3/channel/IPEndpoint.h>
12 #include <opendnp3/channel/PrintingChannelListener.h>
13 #include <modbus/modbus.h>
14 #include <iostream>
15 #include <memory>
16 #include <thread>
17 #include <chrono>
18
19 using namespace std;
20 using namespace opendnp3;
21
22 // Estrutura para armazenar o estado da aplicacao
23 struct State {
24     int16_t analog = 0; // Valor analogico de entrada
25     bool led_status = 0; // Estado do LED
26     bool button_status = 0; // Estado do botao
27     int16_t last_valid_value = 0; // Ultimo valor analogico valido
28     const int COIL_LIGAR = 0; // Endereco da bobina para ligar
29     const int COIL_DESLIGAR = 1; // Endereco da bobina para desligar
30     const int COIL_STATUS_LED = 2; // Endereco da bobina do LED
31     const int COIL_STATUS_BUTTON = 3; // Endereco da bobina do botao
32
33     bool modbus_connected = false; // Status da conexao Modbus
34     bool last_connection_state = false; // Estado anterior da conexao
35     int failure_count = 0; // Contador de falhas consecutivas
36     const int max_failures_before_zero = 5; // Maximo de falhas antes de enviar
        zero
37 };
38
39 // Configura os pontos da base de dados DNP3
40 DatabaseConfig ConfigureDatabase()
41 {
42     DatabaseConfig config;
43
44     // Configura ponto analogico de entrada

```

```

45     config.analog_input[0] = AnalogConfig();
46     config.analog_input[0].clazz = PointClass::Class2;
47     config.analog_input[0].svariation = StaticAnalogVariation::Group30Var2;
48
49     // Configura pontos binarios de entrada
50     config.binary_input[0] = BinaryConfig();
51     config.binary_input[0].clazz = PointClass::Class1;
52     config.binary_input[0].svariation = StaticBinaryVariation::Group1Var2;
53
54     config.binary_input[1] = BinaryConfig();
55     config.binary_input[1].clazz = PointClass::Class1;
56     config.binary_input[1].svariation = StaticBinaryVariation::Group1Var2;
57
58     config.binary_input[2] = BinaryConfig();
59     config.binary_input[2].clazz = PointClass::Class1;
60     config.binary_input[2].svariation = StaticBinaryVariation::Group1Var2;
61
62     return config;
63 }
64
65 // Adiciona atualizacoes ao outstation DNP3
66 void AddUpdates(UpdateBuilder& builder, State& state) {
67     // Atualiza valor analogico
68     if (state.failure_count >= state.max_failures_before_zero) {
69         builder.Update(Analog(0), 0);
70     } else {
71         builder.Update(Analog(state.last_valid_value), 0);
72     }
73
74     // Atualiza status da conexao
75     bool connection_failed = !state.modbus_connected;
76     builder.Update(Binary(connection_failed), 0);
77
78     // Atualiza flag de status se houve mudanca
79     if (state.modbus_connected != state.last_connection_state) {
80         builder.Update(Binary(connection_failed, Flags(0x01)), 0);
81         state.last_connection_state = state.modbus_connected;
82     }
83
84     // Atualiza entradas binarias
85     builder.Update(Binary(state.led_status), 1);
86     builder.Update(Binary(state.button_status), 2);
87 }
88
89 // Tenta reconectar ao dispositivo Modbus
90 bool TryModbusReconnect(modbus_t* ctx, const char* ip, int port, int slave_id,
91     State& state) {
92     cout << "Tentando reconectar ao Modbus..." << endl;
93
94     // Fecha conexao existente se houver
95     if (state.modbus_connected) {
96         modbus_close(ctx);
97         state.modbus_connected = false;
98     }
99
100    // Cria novo contexto se necessario
101    if (ctx == nullptr) {

```

```

101     ctx = modbus_new_tcp(ip, port);
102     if (ctx == nullptr) {
103         cerr << "Falha ao criar novo contexto Modbus" << endl;
104         return false;
105     }
106     modbus_set_response_timeout(ctx, 1, 0);
107     modbus_set_byte_timeout(ctx, 1, 0);
108 }
109
110 // Configura ID do escravo
111 if (modbus_set_slave(ctx, slave_id) == -1) {
112     cerr << "Erro ao configurar ID do escravo: " << modbus_strerror(errno)
113         << endl;
114     modbus_free(ctx);
115     return false;
116 }
117
118 // Tenta conexao
119 if (modbus_connect(ctx) == -1) {
120     cerr << "Falha na reconexao: " << modbus_strerror(errno) << endl;
121     return false;
122 }
123
124 state.modbus_connected = true;
125 cout << "Conexao Modbus restabelecida!" << endl;
126 return true;
127 }
128
129 // Le valores do dispositivo Modbus
130 bool ReadModbusValues(modbus_t* ctx, const char* ip, int port, int slave_id,
131     State& state) {
132     uint16_t tab_reg[1];
133     uint8_t led_status, button_status;
134
135     // Tenta reconectar se nao estiver conectado
136     if (!state.modbus_connected) {
137         if (!TryModbusReconnect(ctx, ip, port, slave_id, state)) {
138             state.failure_count++;
139             return false;
140         }
141     }
142
143     // Le registro analogico
144     modbus_flush(ctx);
145     int rc = modbus_read_registers(ctx, 0, 1, tab_reg);
146
147     if (rc == -1) {
148         cerr << "Erro na leitura do potenciometro: " << modbus_strerror(errno)
149             << endl;
150         modbus_close(ctx);
151         state.modbus_connected = false;
152         state.failure_count++;
153         return false;
154     }
155
156     // Le status do LED
157     rc = modbus_read_bits(ctx, state.COIL_STATUS_LED, 1, &led_status);

```

```

155
156     if (rc == -1) {
157         cerr << "Erro na leitura do status do LED: " << modbus_strerror(errno)
158             << endl;
159         modbus_close(ctx);
160         state.modbus_connected = false;
161         state.failure_count++;
162         return false;
163     }
164
165     // Le status do botao
166     rc = modbus_read_bits(ctx, state.COIL_STATUS_BUTTON, 1, &button_status);
167
168     if (rc == -1) {
169         cerr << "Erro na leitura do coil do Botao: " << modbus_strerror(errno)
170             << endl;
171         modbus_close(ctx);
172         state.modbus_connected = false;
173         state.failure_count++;
174         return false;
175     }
176
177     // Atualiza estado com novos valores
178     state.analog = static_cast<int16_t>(tab_reg[0]);
179     state.last_valid_value = state.analog;
180     state.led_status = (led_status == 1);
181     state.button_status = !(button_status == 1);
182     state.failure_count = 0;
183     return true;
184 }
185
186 // Flag para tratamento de sinal
187 volatile sig_atomic_t shutdown_flag = 0;
188
189 // Funcao para tratamento de sinal
190 void signal_handler(int signal) {
191     shutdown_flag = 1;
192 }
193
194 // Liga dispositivo via Modbus
195 bool LigarDispositivo(modbus_t* ctx, const State& state) {
196     if (modbus_write_bit(ctx, state.COIL_LIGAR, true) == -1) {
197         std::cerr << "Erro ao ligar: " << modbus_strerror(errno) << std::endl;
198         return false;
199     }
200     modbus_flush(ctx);
201     return true;
202 }
203
204 // Desliga dispositivo via Modbus
205 bool DesligarDispositivo(modbus_t* ctx, const State& state) {
206     if (modbus_write_bit(ctx, state.COIL_DESLIGAR, true) == -1) {
207         std::cerr << "Erro ao desligar: " << modbus_strerror(errno) << std::endl;
208         ;
209         return false;
210     }
211     modbus_flush(ctx);

```

```

209     return true;
210 }
211
212 // Manipulador de comandos personalizado para DNP3
213 class DirectOperateOnlyHandler : public SimpleCommandHandler {
214 private:
215     modbus_t* ctx;
216     State state;
217
218 public:
219     DirectOperateOnlyHandler(modbus_t* modbus_ctx, const State& initialState)
220         : SimpleCommandHandler(CommandStatus::SUCCESS),
221         ctx(modbus_ctx),
222         state(initialState) {}
223
224     // Manipula comandos de controle
225     CommandStatus Operate(const ControlRelayOutputBlock& command,
226                          uint16_t index,
227                          IUpdateHandler& handler,
228                          OperateType opType) override {
229         if (opType == OperateType::DirectOperate) {
230             switch (index) {
231                 case 0:
232                     std::cout << "Direct Operate: LED ON" << std::endl;
233                     return LigarDispositivo(ctx, state) ?
234                         CommandStatus::SUCCESS :
235                         CommandStatus::HARDWARE_ERROR;
236                 case 1:
237                     std::cout << "Direct Operate: LED OFF" << std::endl;
238                     return DesligarDispositivo(ctx, state) ?
239                         CommandStatus::SUCCESS :
240                         CommandStatus::HARDWARE_ERROR;
241                 default:
242                     return CommandStatus::NOT_SUPPORTED;
243             }
244         }
245         return CommandStatus::NOT_SUPPORTED;
246     }
247 };
248
249 int main() {
250     // Configura handlers de sinal
251     signal(SIGINT, signal_handler);
252     signal(SIGTERM, signal_handler);
253
254     // Parametros de conexao Modbus
255     const char* modbus_ip = "192.168.100.120";
256     const int modbus_port = 502;
257     const int modbus_slave_id = 1;
258
259     // Cria contexto Modbus
260     modbus_t* ctx = modbus_new_tcp(modbus_ip, modbus_port);
261
262     // Tenta conexao inicial
263     while (!shutdown_flag) {
264         if (modbus_connect(ctx) == -1) {

```

```

265         std::cerr << "Erro ao conectar ao dispositivo: " << modbus_strerror(
266             errno) << std::endl;
267         std::cerr << "Tentando novamente em 5 segundos..." << std::endl;
268
269         std::this_thread::sleep_for(std::chrono::seconds(5));
270     } else {
271         std::cout << "Conexao Modbus estabelecida com sucesso!" << std::endl
272             ;
273         break;
274     }
275 }
276
277 // Configura ID do escravo Modbus
278 if (modbus_set_slave(ctx, 1) == -1)
279 {
280     std::cerr << "Erro ao configurar ID do escravo: " << modbus_strerror(
281         errno) << std::endl;
282     modbus_free(ctx);
283     return -1;
284 }
285
286 // Verifica conexao
287 if (modbus_connect(ctx) == -1)
288 {
289     std::cerr << "Erro ao conectar ao dispositivo: " << modbus_strerror(
290         errno) << std::endl;
291     modbus_free(ctx);
292     return -1;
293 }
294
295 // Inicializa estado da aplicacao
296 State state;
297
298 // Configura niveis de log DNP3
299 const auto logLevels = levels::NORMAL | levels::NOTHING;
300
301 // Cria gerenciador DNP3
302 DNP3Manager manager(1, ConsoleLogger::Create());
303
304 // Configura canal DNP3
305 auto channel = std::shared_ptr<IChannel>(nullptr);
306 try
307 {
308     channel = manager.AddTCPServer("server", logLevels, ServerAcceptMode::
309         CloseExisting, IPEndpoint("192.168.100.176", 20000),
310         PrintingChannelListener::Create());
311 }
312 catch(const std::exception& e)
313 {
314     std::cerr << "Erro ao configurar canal DNP3: " << e.what() << '\n';
315     return -1;
316 }
317
318 // Configura stack DNP3 outstation
319 OutstationStackConfig config(ConfigureDatabase());
320
321 config.outstation.eventBufferConfig = EventBufferConfig::AllTypes(10);

```

```
317
318     config.outstation.params.allowUnsolicited = true;
319
320     config.link.LocalAddr = 2;
321     config.link.RemoteAddr = 1;
322
323     config.link.KeepAliveTimeout = TimeDuration::Seconds(30);
324
325     // Cria instancia outstation
326     auto outstation = channel->AddOutstation(
327         "outstation",
328         std::make_shared<DirectOperateOnlyHandler>(ctx, state),
329         DefaultOutstationApplication::Create(),
330         config
331     );
332
333     outstation->Enable();
334
335     // Loop principal da aplicacao
336     while (!shutdown_flag) {
337         // Le valores do Modbus
338         bool read_success = ReadModbusValues(ctx, modbus_ip, modbus_port,
339             modbus_slave_id, state);
340
341         // Atualiza pontos DNP3
342         UpdateBuilder builder;
343         AddUpdates(builder, state);
344         outstation->Apply(builder.Build());
345
346         // Log de status
347         if (!read_success) {
348             if (state.failure_count >= state.max_failures_before_zero) {
349                 cout << "Falha prolongada - Enviando 0 (Status: FALHA)" << endl;
350             } else {
351                 cout << "Falha temporaria - Ultimo valor valido: "
352                     << state.last_valid_value << " (Status: FALHA)" << endl;
353             }
354         } else {
355             cout << "Leitura OK - Valor atual: " << state.analog
356                 << " (Status: CONECTADO)" << endl;
357         }
358
359         this_thread::sleep_for(chrono::seconds(1));
360     }
361
362     // Limpeza da conexao Modbus
363     if (state.modbus_connected) {
364         modbus_close(ctx);
365     }
366     modbus_free(ctx);
367
368     cout << "Encerrando programa..." << endl;
369     return 0;
370 }
```

ANEXO C – Código - Gateway 2ª Implementação

Código C.1 – Implementação 2 - Estação Remota DNP3 integrada a dispositivos reais

```

1
2 // Inclui bibliotecas necessarias
3 #include <opendnp3/ConsoleLogger.h>
4 #include <opendnp3/DNP3Manager.h>
5 #include <opendnp3/logging/LogLevels.h>
6 #include <opendnp3/outstation/DefaultOutstationApplication.h>
7 #include <opendnp3/outstation/IUpdateHandler.h>
8 #include <opendnp3/outstation/UpdateBuilder.h>
9 #include <opendnp3/outstation/OutstationStackConfig.h>
10 #include <opendnp3/outstation/SimpleCommandHandler.h>
11 #include <opendnp3/channel/IPEndpoint.h>
12 #include <opendnp3/channel/PrintingChannelListener.h>
13
14 #include <stdio.h>
15 #include <stdlib.h>
16 #include <modbus/modbus.h>
17 #include <errno.h>
18 #include <string.h>
19 #include <unistd.h>
20 #include <iostream>
21 #include <memory>
22 #include <thread>
23 #include <chrono>
24 #include <vector>
25 #include <mutex>
26 #include <atomic>
27
28 using namespace std;
29 using namespace opendnp3;
30
31 // Constantes de configuracao do sistema
32 #define NUM_SLAVES 3 // Numero de dispositivos Modbus (slaves) para
    monitorar
33 #define NUM_HOLDING_REGISTERS 2 // Numero de registros holding para leitura (32
    bits)
34 #define NUM_INPUT_REGISTERS 1 // Numero de registros input para leitura (16
    bits)
35
36 // Mutexes para sincronizacao
37 mutex data_mutex; // Protege dados compartilhados entre threads
38 mutex dnp3_update_mutex; // Protege atualizacoes DNP3
39
40 // Estrutura para armazenar estado de cada slave Modbus
41 struct SlaveState {
42     int32_t analog_value = 0; // Valor analogico atual do slave
43     bool connection_status = false; // Status atual da conexao
44     bool last_connection_state = true; // Status anterior da conexao

```

```

45     bool connection_changed = true;        // Flag indicando mudanca no status
46     int failure_count = 0;                 // Contador de falhas consecutivas
47     const int max_failures_before_zero = 5; // Max falhas antes de zerar valor
48     chrono::system_clock::time_point last_change_time; // Timestamp da ultima
        mudanca
49 };
50
51 // Funcao para configurar banco de dados DNP3
52 DatabaseConfig ConfigureDatabase() {
53     DatabaseConfig config;
54
55     // Configura entradas analogicas e binarias para cada slave
56     for(int i = 0; i < NUM_SLAVES; i++) {
57         // Configuracao entrada analogica
58         config.analog_input[i] = AnalogConfig();
59         config.analog_input[i].clazz = PointClass::Class2;
60         config.analog_input[i].svariation = (i == 0) ? StaticAnalogVariation::
            Group30Var1
61                                     : StaticAnalogVariation::
            Group30Var2;
62
63         // Configuracao entrada binaria
64         config.binary_input[i] = BinaryConfig();
65         config.binary_input[i].clazz = PointClass::Class1;
66         config.binary_input[i].svariation = StaticBinaryVariation::Group1Var2;
67         config.binary_input[i].evariation = EventBinaryVariation::Group2Var2;
68     }
69
70     return config;
71 }
72
73 // Estrutura para configuracao de slave Modbus
74 typedef struct {
75     const char* ip;                // Endereco IP do slave
76     int port;                      // Porta do slave
77     int slave_id;                  // ID do slave
78     int is_first_slave;            // Flag para primeiro slave (tipo de registro
        diferente)
79     int dnp3_analog_index;         // Indice DNP3 para valor analogico
80     int dnp3_status_index;        // Indice DNP3 para status
81 } SlaveConfig;
82
83 // Atualiza valores DNP3 baseado no estado do slave
84 void UpdateDNP3Values(UpdateBuilder& builder, SlaveState& state, int index) {
85     // Atualiza valor analogico (zera se muitas falhas)
86     builder.Update(Analog((state.failure_count >= state.max_failures_before_zero
        ) ? 0 : state.analog_value), index);
87
88     // Atualiza status da conexao se houve mudanca
89     if(state.connection_changed || state.last_change_time == chrono::
        system_clock::time_point()) {
90         bool connection_failed = !state.connection_status;
91         builder.Update(Binary(connection_failed, Flags(0x01)), index);
92     }
93 }
94
95 // Trata falha de comunicacao com slave

```

```

96 void HandleCommunicationFailure(SlaveState& state) {
97     bool previous_status = state.connection_status;
98     state.connection_status = false;
99     state.connection_changed = (state.connection_status != previous_status);
100     state.failure_count++;
101
102     // Zera valor analogico se muitas falhas
103     if (state.failure_count >= state.max_failures_before_zero) {
104         state.analog_value = 0;
105     }
106
107     // Registra hora da mudanca se status alterou
108     if(state.connection_changed) {
109         state.last_change_time = chrono::system_clock::now();
110     }
111 }
112
113 // Trata comunicacao bem sucedida com slave
114 void HandleCommunicationSuccess(SlaveState& state) {
115     bool previous_status = state.connection_status;
116     state.connection_status = true;
117     state.connection_changed = (state.connection_status != previous_status);
118     state.failure_count = 0;
119
120     // Registra hora da mudanca se status alterou
121     if(state.connection_changed) {
122         state.last_change_time = chrono::system_clock::now();
123     }
124 }
125
126 // Converte registros Modbus para inteiro 32 bits
127 int32_t modbusRegistersToInt32(uint16_t* regs, bool is_signed) {
128     int32_t value = (regs[0] << 16) | regs[1];
129     if (is_signed && (value & 0x80000000)) {
130         value |= 0xFFFFFFFF00000000;
131     }
132     return value;
133 }
134
135 // Funcao de thread para monitorar um slave Modbus
136 void PollSlave(int slave_index, SlaveConfig* slaves, vector<SlaveState>*
137 slave_states, shared_ptr<opendnp3::IOutstation> outstation) {
138     const int HOLDING_REG_OFFSET = 23322; // Offset holding register para
139     primeiro slave
140     const int INPUT_REG_OFFSET = 37; // Offset input register para outros
141     slaves
142     uint16_t tab_reg[max(NUM_HOLDING_REGISTERS, NUM_INPUT_REGISTERS)];
143
144     // Atraso inicial escalonado para evitar congestionamento
145     this_thread::sleep_for(chrono::milliseconds(100 * slave_index));
146
147     while (true) {
148         bool read_success = false;
149
150         // Cria novo contexto Modbus para cada tentativa
151         modbus_t* ctx = modbus_new_tcp(slaves[slave_index].ip, slaves[
152             slave_index].port);

```

```

149     if (ctx) {
150         modbus_set_slave(ctx, slaves[slave_index].slave_id);
151         modbus_set_response_timeout(ctx, 1, 0);
152         modbus_set_byte_timeout(ctx, 0, 500000);
153
154         if (modbus_connect(ctx) != -1) {
155             try {
156                 // Primeiro slave usa holding registers, outros usam input
157                 registers
158                 if (slaves[slave_index].is_first_slave) {
159                     int rc = modbus_read_registers(ctx, HOLDING_REG_OFFSET,
160                     NUM_HOLDING_REGISTERS, tab_reg);
161                     if (rc != -1) {
162                         lock_guard<mutex> lock(data_mutex);
163                         (*slave_states)[slave_index].analog_value =
164                             modbusRegistersToInt32(tab_reg, true);
165                         cout << "Slave " << slave_index << " (Holding): " <<
166                             (*slave_states)[slave_index].analog_value <<
167                             endl;
168                         HandleCommunicationSuccess((*slave_states)[
169                             slave_index]);
170                         read_success = true;
171                     }
172                 } else {
173                     int rc = modbus_read_input_registers(ctx,
174                     INPUT_REG_OFFSET, NUM_INPUT_REGISTERS, tab_reg);
175                     if (rc != -1) {
176                         lock_guard<mutex> lock(data_mutex);
177                         (*slave_states)[slave_index].analog_value =
178                             static_cast<int16_t>(tab_reg[0]);
179                         cout << "Slave " << slave_index << " (Input): " <<
180                             (*slave_states)[slave_index].analog_value <<
181                             endl;
182                         HandleCommunicationSuccess((*slave_states)[
183                             slave_index]);
184                         read_success = true;
185                     }
186                 }
187             } catch (const char* e) {
188                 cerr << "Erro lendo slave " << slave_index << ": " << e <<
189                 endl;
190             }
191             modbus_close(ctx);
192         }
193         modbus_free(ctx);
194     }
195
196     if (!read_success) {
197         lock_guard<mutex> lock(data_mutex);
198         cerr << "Falha comunicacao com slave " << slave_index << endl;
199         HandleCommunicationFailure((*slave_states)[slave_index]);
200     }
201
202     // Atualiza valores DNP3
203     {
204         lock_guard<mutex> dnp_lock(dnp3_update_mutex);
205         UpdateBuilder builder;

```

```

194
195         {
196             lock_guard<mutex> data_lock(data_mutex);
197             UpdateDNP3Values(builder, (*slave_states)[slave_index],
198                             slave_index);
199
200             // Reseta flag de mudanca se foi enviada
201             if ((*slave_states)[slave_index].connection_changed) {
202                 auto now = chrono::system_clock::now();
203                 auto elapsed = chrono::duration_cast<chrono::milliseconds>(
204                     now - (*slave_states)[slave_index].last_change_time).
205                     count();
206                 if (elapsed > 100) { // Espera pelo menos 100ms antes de
207                     resetar
208                         (*slave_states)[slave_index].connection_changed = false;
209                 }
210             }
211
212             outstation->Apply(builder.Build());
213         }
214     }
215
216     this_thread::sleep_for(chrono::seconds(1));
217 }
218
219 int main() {
220     // Inicializa gerenciador DNP3 com logging
221     const auto logLevels = levels::NORMAL | levels::NOTHING;
222     DNP3Manager manager(1, ConsoleLogger::Create());
223
224     // Cria canal TCP server
225     auto channel = manager.AddTCPServer("server", logLevels, ServerAcceptMode::
226         CloseExisting,
227         IPEndpoint("10.1.1.223", 20000),
228         PrintingChannelListener::Create());
229
230     // Configura stack outstation DNP3
231     OutstationStackConfig stackConfig(ConfigureDatabase());
232     stackConfig.outstation.eventBufferConfig = EventBufferConfig::AllTypes(100);
233     stackConfig.outstation.params.allowUnsolicited = true;
234     stackConfig.link.LocalAddr = 2;
235     stackConfig.link.RemoteAddr = 1;
236
237     // Cria e ativa outstation
238     auto outstation = channel->AddOutstation("outstation", SuccessCommandHandler
239         ::Create(),
240         DefaultOutstationApplication::Create(
241             , stackConfig);
242
243     outstation->Enable();
244
245     /*
246     * Configuracao dos dispositivos Modbus (slaves) que serao monitorados:
247     * Cada entrada contem:
248     * 1. IP do dispositivo
249     * 2. Porta Modbus (normalmente 502)
250     * 3. ID do escravo Modbus
251     */

```

```
244 * 4. Tipo de registro (1 = holding registers, 0 = input registers)
245 * 5. Indice do ponto analogico no DNP3
246 * 6. Indice do ponto binario (status) no DNP3
247 *
248 * 0 primeiro slave usa holding registers (32 bits) enquanto os demais usam
    input registers (16 bits)
249 */
250 SlaveConfig slaves[NUM_SLAVES] = {
251     {"10.1.1.116", 502, 1, 1, 0, 0},
252     {"10.1.1.41", 502, 1, 0, 1, 1},
253     {"10.1.1.42", 502, 1, 0, 2, 2}
254 };
255
256 vector<SlaveState> slave_states(NUM_SLAVES);
257
258 // Envia status inicial desconectado
259 {
260     UpdateBuilder builder;
261     for (int i = 0; i < NUM_SLAVES; ++i) {
262         builder.Update(Binary(true, Flags(0x01)), i); // Status inicial =
            falha
263         slave_states[i].last_change_time = chrono::system_clock::now();
264     }
265     outstation->Apply(builder.Build());
266 }
267
268 // Cria threads para monitorar cada slave
269 vector<thread> threads;
270
271 for (int i = 0; i < NUM_SLAVES; ++i) {
272     threads.emplace_back(PollSlave, i, slaves, &slave_states, outstation);
273 }
274
275 // Desvincula threads (elas rodam indefinidamente)
276 for (auto& t : threads) {
277     t.detach();
278 }
279
280 // Mantem thread principal rodando
281 while(true) {
282     this_thread::sleep_for(chrono::hours(1));
283 }
284
285 return 0;
286 }
```

ANEXO D – CMakeLists

Código D.1 – CMakeLists para compilação dos projetos

```
1
2 cmake_minimum_required(VERSION 3.10) # Versao minima do CMake
3 project(DNP3_Modbus_Integration)     # Nome do projeto integrado
4
5 # Configuracoes do C++ (usando C++11 ou superior)
6 set(CMAKE_CXX_STANDARD 14)
7 set(CMAKE_CXX_STANDARD_REQUIRED ON)
8
9 # Procura pela biblioteca libmodbus
10 find_package(PkgConfig REQUIRED)
11 pkg_check_modules(LIBMODBUS REQUIRED libmodbus)
12
13 # Adiciona o diretorio de include da libmodbus
14 include_directories(${LIBMODBUS_INCLUDE_DIRS})
15
16 #include_directories(/usr/local/include/opensnp3/gen)
17 #include_directories(/usr/local/include/opensnp3/app)
18
19 # Adiciona o executavel principal (integra DNP3 e Modbus)
20 add_executable(dnp3_modbus_integration main.cpp)
21
22 # Linka as bibliotecas necessarias
23 target_link_libraries(dnp3_modbus_integration PRIVATE opensnp3 ${
    LIBMODBUS_LIBRARIES})
24
25 # Define propriedades adicionais para o alvo (opcional)
26 set_target_properties(dnp3_modbus_integration PROPERTIES FOLDER cpp/src)
27
28 # Instala o executavel no diretorio bin (opcional)
29 install(TARGETS dnp3_modbus_integration RUNTIME DESTINATION bin)
```