Recarga de Carros Elétricos Inteligentes

Lucas de Andrade Pereira Mendes, Thiago Ramon, Jhessé Campos Rodrigues

Departamento de Tecnologia
Universidade Estadual de Feira de Santana (UEFS) – Feira de Santana, BA – Brasil

luc4s.a4@gmail.com, thiagoramon993@gmail.com, jhessecampos1412@gmail.com

Abstract. *This work proposes a TCP/IP-based solution, using native sockets and the Go programming language, to enable an intelligent interaction system between electric vehicles and charging stations, mediated by a cloud server. The architecture follows a client-server model, consisting of three main entities: the vehicle, the charging station, and the central server. The system allows vehicles to identify the nearest and available charging points, make reservations, monitor charging time, automatically release stations, and record consumption values. The solution was tested in a Docker-based environment, enabling the simulation of multiple instances and realistic scenarios. The results demonstrate the feasibility and efficiency of the proposed approach in organizing and automating the charging process.*

Resumo. *Este trabalho propõe uma solução baseada em comunicação TCP/IP, utilizando sockets nativos e a linguagem Go, para viabilizar um sistema inteligente de interação entre veículos elétricos e pontos de recarga, mediado por um servidor em nuvem. A arquitetura adotada segue o modelo cliente-servidor, sendo composta por três entidades principais: o veículo, o ponto de recarga e o servidor central. O sistema permite identificar os pontos mais próximos e disponíveis, realizar reservas, controlar o tempo de recarga, liberar automaticamente os pontos e registrar os valores consumidos. Os testes foram realizados em um ambiente dockerizado, possibilitando a simulação de múltiplas instâncias e cenários reais. Os resultados demonstram a viabilidade e a eficiência da proposta na organização e automação do processo de recarga.*

1. Introdução

O aumento expressivo da frota de veículos elétricos no Brasil nos últimos anos tem evidenciado a necessidade urgente de melhorias na infraestrutura de recarga. Segundo a Associação Brasileira do Veículo Elétrico (ABVE), em 2022 foram emplacados mais de 8 mil carros elétricos no país, número que saltou para quase 94 mil em 2023 [1] e ultrapassou 170 mil em 2024 [2]. Apesar desse crescimento, ainda persistem diversos desafios que dificultam a popularização desses veículos, como a baixa disponibilidade de pontos de recarga, a concentração desses pontos em áreas urbanas específicas, longas filas de espera e a escassez de informações em tempo real sobre localização e disponibilidade dos carregadores.

Diante desse cenário, este projeto propõe o desenvolvimento de um sistema cliente-servidor em nuvem que possibilita a comunicação inteligente, padronizada e em tempo real entre veículos elétricos e pontos de recarga. O objetivo é auxiliar motoristas

na identificação dos pontos mais adequados para recarga com base na ocupação e na distância, promover a reserva antecipada de carregadores, automatizar a liberação dos pontos após o uso e registrar os valores das recargas para posterior pagamento eletrônico.

A solução foi implementada utilizando sockets nativos TCP/IP, conforme exigido, sem o uso de frameworks de mensagens, e foi testada em um ambiente dockerizado para garantir modularidade e escalabilidade. Este relatório descreve o contexto do problema, os fundamentos teóricos utilizados, a arquitetura e implementação da solução, bem como os experimentos realizados e os resultados obtidos.

2. Fundamentação Teórica

2.1. Comunicação em Redes com Sockets TCP

A comunicação entre processos em redes de computadores pode ser realizada por meio de diferentes protocolos, sendo o TCP (Transmission Control Protocol) um dos mais utilizados por oferecer uma conexão confiável, orientada à conexão e com controle de fluxo e erros. A interface de programação de sockets, introduzida no sistema UNIX, permite a implementação direta da comunicação entre processos através de canais de comunicação definidos por pares de endpoints (endereços IP e portas).

No contexto deste projeto, a utilização de sockets TCP é fundamental para estabelecer a comunicação entre os módulos do sistema (cliente, nuvem e ponto de recarga), conforme exigido nas restrições do problema. A escolha pelo TCP justifica-se pela necessidade de garantir a entrega correta e ordenada das mensagens, principalmente para operações críticas como reserva de pontos, liberação e confirmação de pagamentos [6].

2.2. Arquitetura Cliente-Servidor

A arquitetura cliente-servidor é amplamente adotada em sistemas distribuídos, caracterizando-se pela separação entre entidades que solicitam serviços (clientes) e aquelas que os fornecem (servidores). Essa arquitetura facilita a escalabilidade, modularidade e centralização do controle, sendo adequada para aplicações que exigem sincronização e coordenação entre múltiplos agentes.

Neste projeto, o servidor representa o componente "nuvem", responsável por coordenar o estado dos pontos de recarga e responder às requisições dos clientes (veículos). Os pontos de recarga também se comunicam com o servidor para reportar disponibilidade e registrar eventos. Essa organização promove uma visão centralizada do sistema, simplificando o balanceamento de carga e a reserva de recursos [7].

2.3. Contêineres Docker

O Docker é uma plataforma que permite empacotar, distribuir e executar aplicações de forma isolada e reprodutível por meio de contêineres. Cada contêiner é uma instância

leve de um sistema de arquivos com todos os binários, bibliotecas e dependências necessários para executar uma aplicação. Isso garante que o ambiente de desenvolvimento seja idêntico ao de produção ou teste.

A utilização de Docker neste projeto permite o isolamento de cada componente do sistema (carro, nuvem e ponto de recarga), facilitando a simulação de múltiplas instâncias para testes em laboratório. Além disso, a portabilidade proporcionada pelo Docker viabiliza a implantação do sistema em diferentes ambientes com mínima configuração [4].

2.4. Sistemas de Reservas Distribuídas

Sistemas de reservas distribuídas são comuns em contextos como redes de transporte, hospedagem e energia. Esses sistemas visam garantir a alocação eficiente de recursos compartilhados entre usuários concorrentes. No caso da recarga de veículos elétricos, a reserva de pontos precisa ser gerenciada de forma a evitar conflitos (ex.: duas reservas simultâneas para o mesmo ponto) e minimizar o tempo de espera dos usuários.

A abordagem utilizada no projeto implementa a reserva de pontos a partir da comunicação direta entre os clientes e o servidor central, que atua como mediador e mantém o estado atualizado de cada ponto. Técnicas de sincronização e controle de concorrência foram consideradas para garantir a consistência do sistema.

3. Metodologia

3.1. Proposta de Solução

A proposta deste projeto é o desenvolvimento de um sistema distribuído baseado na arquitetura cliente-servidor para intermediar a comunicação entre veículos elétricos, pontos de recarga e um servidor em nuvem. O sistema visa atender aos principais desafios enfrentados atualmente por usuários de veículos elétricos no Brasil, como filas de espera, pontos indisponíveis e falta de controle sobre os custos de recarga.

A solução é composta por três entidades principais:

- **Servidor Central (Nuvem);**
- **Veículo Elétrico (Cliente);**
- **Ponto de Recarga;**

Cada entidade foi implementada como um contêiner Docker separado e utiliza **comunicação via sockets TCP**, com mensagens formatadas em texto puro. As conexões são gerenciadas de forma assíncrona para permitir a simulação de múltiplos clientes e pontos conectados simultaneamente.

3.1.1. Arquitetura Geral

A arquitetura do sistema pode ser representada pela Figura 1, onde o servidor central atua como um hub de comunicação, recebendo e processando solicitações dos veículos e dos pontos de recarga.

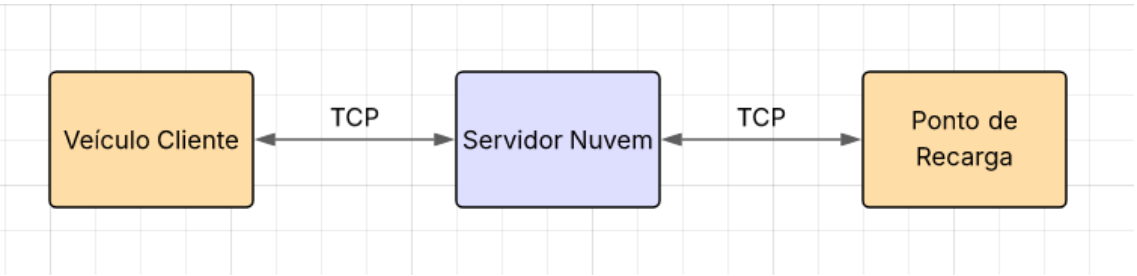

Figura 1

3.1.2. Componentes do Sistema

a) Servidor Central (Nuvem)

O servidor implementa a lógica de controle do sistema. Ele:

- Mantém uma lista com os **pontos de recarga disponíveis e ocupados**
- Recebe solicitações de reserva de recarga dos veículos
- Valida a solicitação com base na ocupação atual dos pontos
- Encaminha a reserva confirmada para o ponto e para o veículo
- Gerencia a **liberação automática** dos pontos após uso
- Registra o valor consumido por cada veículo (simulação de pagamento)

Internamente, o servidor armazena os estados dos pontos em uma estrutura `map[int]bool`, onde o número do ponto é a chave e o valor booleano representa se ele está ocupado.

b) Veículo Elétrico (Cliente)

Cada veículo é executado como um processo independente que:

- Solicita ao servidor o ponto de recarga mais próximo e disponível;
- Aguarda confirmação e estabelece a reserva;
- Simula o processo de recarga (com `sleep`);
- Envia confirmação de liberação após o tempo de recarga;
- Recebe informações sobre o custo da recarga;

As mensagens seguem um protocolo simples baseado em comandos como `SOLICITAR`, `RESERVAR`, `LIBERAR` e `CONFIRMAR`.

c) Ponto de Recarga

O ponto de recarga é um processo passivo que:

- Informa sua disponibilidade ao servidor;
- Recebe comandos de reserva e liberação;
- Atualiza seu status conforme as solicitações recebidas;

Os pontos são simulados com tempo de ocupação arbitrário, permitindo testes de simultaneidade e ocupação concorrente.

3.1.3. Lógica de Comunicação

A troca de mensagens ocorre sempre entre cliente-servidor e servidor-ponto. Todas as mensagens seguem um formato textual padronizado, permitindo fácil decodificação e registro.

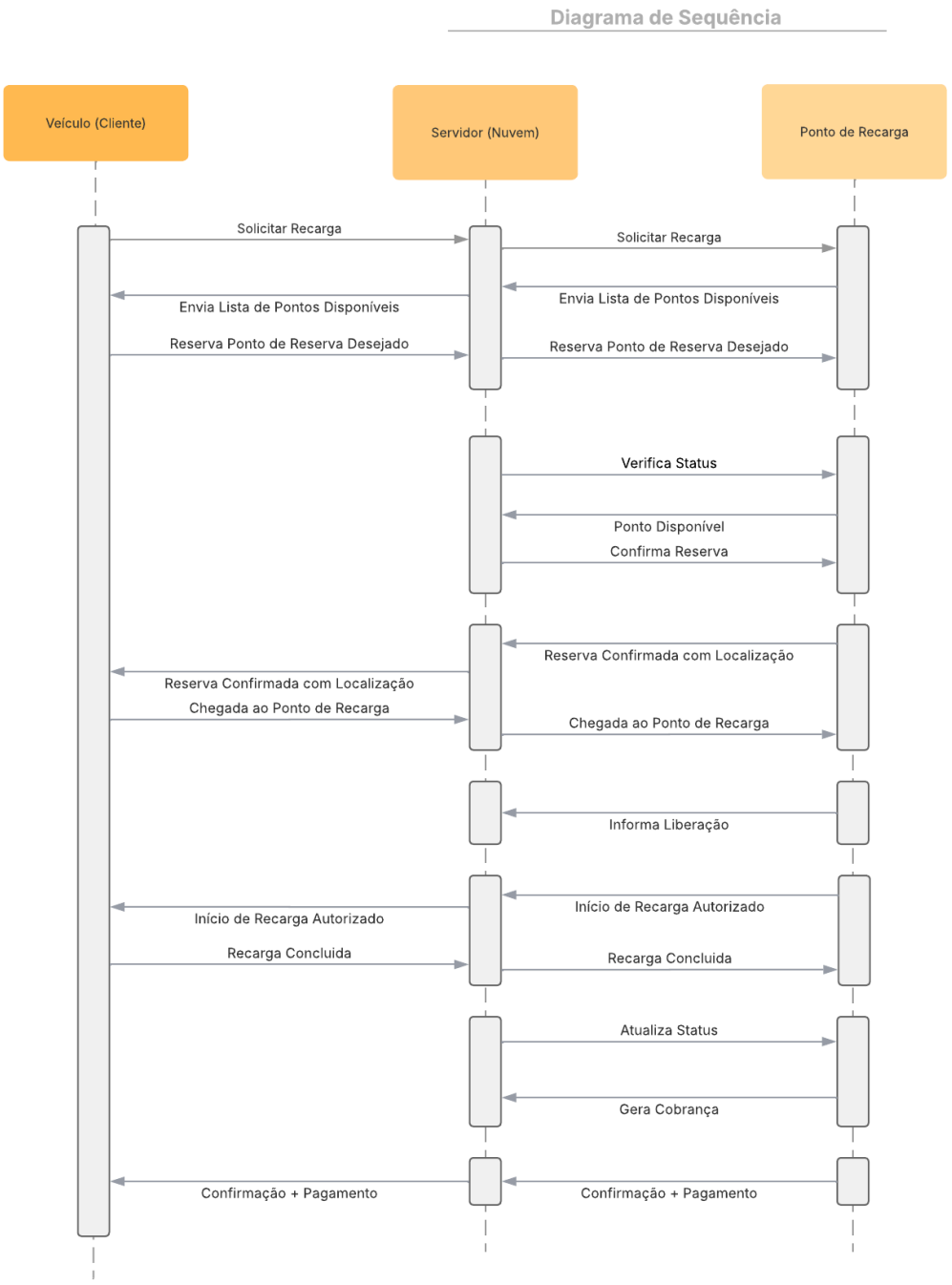

Figura 2.

Exemplo de sequência de mensagens:

1. Cliente envia `SOLICITAR` → Servidor responde com ponto mais próximo;
2. Cliente envia `RESERVAR X` → Servidor valida e encaminha para o ponto;
3. Após tempo de uso, cliente envia `LIBERAR X` → Servidor marca ponto como livre;

4. Servidor envia `VALOR` → Cliente registra ou simula pagamento;

3.1.4. Justificativas Técnicas

- A escolha por **Go (Golang)** deve-se à sua leveza, excelente suporte a concorrência via goroutines e canais, além da facilidade para trabalhar com sockets nativos.
- O uso de **Docker** permite simular múltiplos clientes e pontos em um mesmo ambiente de testes.
- A comunicação **TCP/IP com sockets nativos** cumpre a restrição do enunciado e garante robustez nas conexões.

3.2. Implementação

A implementação do sistema é composta por três componentes principais: o Cliente (Veículo), o Servidor e o Ponto de Recarga. Cada um desses componentes foi desenvolvido em Go, utilizando sockets TCP para comunicação. A seguir, detalharemos a estrutura e o funcionamento de cada parte do sistema.

3.2.1. Estrutura do Cliente (Veículo)

O cliente é responsável por interagir com o servidor e solicitar pontos de recarga. A estrutura do cliente é definida na classe `Client`, que contém os seguintes atributos:

- **Conexão e Comunicação:** Utiliza `net.Conn` para estabelecer a conexão com o servidor, e `bufio.Reader` e `bufio.Writer` para leitura e escrita de mensagens.
- **Localização e Bateria:** Armazena as coordenadas do veículo (*CoordenadaX* e *CoordenadaY*) e o nível de bateria (*Bateria*).
- **Métodos Principais:**
    - `NewClient`**:** Inicializa a conexão com o servidor e envia uma mensagem de conexão**.**
    - `solicitaPontos`**:** Envia uma solicitação ao servidor para obter pontos de recarga disponíveis.
    - `receberPontos`**:** Recebe e processa a lista de pontos de recarga retornados pelo servidor.
    - `trocaDeMensagens`**:** Implementa um loop de interação com o usuário, permitindo solicitar pontos de recarga ou encerrar a conexão.

3.2.2. Estrutura do Servidor

O servidor é responsável por gerenciar as conexões dos clientes e os pontos de recarga.

A classe `Server` contém:

- **Gerenciamento de Conexões:** Utiliza `net.Listener` para aceitar conexões de clientes e pontos de recarga.
- **Armazenamento de Clientes e Pontos:** Mantém um mapa de clientes (*clients*) e pontos de recarga (*pontos*), permitindo acesso rápido e eficiente.
- **Métodos Principais:**
    - `Start`**:** Inicia o servidor e aguarda conexões.
    - `acceptLoop`**:** Aceita novas conexões e inicia um loop de leitura para cada cliente.
    - `processarMensagens`**:** Processa as mensagens recebidas dos clientes, como solicitações de pontos de recarga e reservas.

3.2.3. Estrutura do Ponto de Recarga

Os pontos de recarga são responsáveis por gerenciar a fila de carros que desejam recarregar. A classe `Ponto` possui:

- **Conexão e Fila:** Armazena a conexão com o servidor e uma fila de carros (*fila*) que aguardam para recarregar.
- **Métodos Principais:**
    - `processarReserva`**:** Processa a reserva de um carro, adicionando-o à fila e enviando uma confirmação.
    - `processarFilaRecarga`**:** Monitora a fila de recarga e atualiza os carros sobre sua posição na fila.

3.2.4. Comunicação entre Componentes

A comunicação entre o cliente, servidor e pontos de recarga é realizada através de mensagens JSON, que são enviadas e recebidas via sockets TCP. Cada mensagem contém um tipo (*Tipo*), conteúdo (*Conteúdo*) e origem (*OrigemMensagem*), permitindo que cada componente entenda a natureza da comunicação.

Cada componente possui um loop de leitura que aguarda mensagens. Quando uma mensagem é recebida, ela é decodificada e processada de acordo com seu tipo. Isso é feito através de um switch-case que determina a ação a ser tomada com base no tipo da mensagem.

3.2.5 Tratamento de concorrência

Tendo em vista a necessidade de controle e tratamento da concorrência, algumas rotinas foram implementadas. Um bom exemplo é a rotina de monitoramento da bateria. Esta rotina monitora se o nível da bateria do veículo chegou em nível crítico, ou seja, 20%. Vale destacar, a estimativa de descarregamento durante o deslocamento até o ponto considerado por essa rotina. Dessa forma, tanto o nível da bateria quanto a distância até o ponto são atualizadas de tempo em tempo garantindo a sincronização.

4. Experimentos e Resultados

Para validar o funcionamento e a robustez do sistema desenvolvido, foram realizados diversos testes que simulam cenários reais de uso com múltiplos veículos e pontos de recarga. Os testes tiveram como objetivo avaliar o comportamento do sistema em situações de concorrência, movimentação contínua, reserva de pontos de recarga e gerenciamento de filas.

4.1 Ambiente de Testes

Os experimentos foram conduzidos utilizando técnicas de conteinerização com Docker, permitindo a simulação de um número ilimitado de carros, desde que gerenciados por um único servidor central. Essa abordagem possibilitou a criação de um ambiente controlado e escalável, favorecendo a replicação dos testes e a simulação de cenários com diferentes quantidades de veículos atuando simultaneamente.

Inicialmente, as configurações de rede foram restritas à máquina local, o que limitou a comunicação entre componentes a um único host. Contudo, a adoção de contêineres foi uma escolha estratégica, pois torna o sistema compatível com futuras expansões para ambientes distribuídos, onde diferentes máquinas possam se conectar à rede de simulação.

4.2 Simulação de Movimentação e Recarga

No componente responsável pelos carros, foram implementadas rotinas específicas que simulam o deslocamento de um veículo ao longo do tempo. Esse movimento é acompanhado por um consumo contínuo de bateria, o que introduz um elemento realista ao comportamento dos carros.

A lógica de recarga foi projetada para se assemelhar ao funcionamento do mundo real. Quando a bateria atinge um nível crítico (por exemplo, 20%), o carro automaticamente inicia uma solicitação de reserva em um ponto de recarga. Alternativamente, o próprio motorista pode acionar manualmente uma recarga, mesmo que a bateria não esteja esgotada. Esse modelo de decisão possibilita múltiplos fluxos de interação com o sistema, o que contribui para uma simulação mais abrangente.

4.3 Testes de Concorrência e Fila de Espera

Um dos principais focos dos testes foi a análise do sistema sob condições de concorrência, especialmente quando vários carros tentam reservar o mesmo ponto de recarga simultaneamente. Inicialmente, foram identificados problemas relacionados à gestão de disponibilidade e organização da fila de espera. Esses problemas foram solucionados com a implementação de uma lógica mais precisa no controle de reservas, garantindo que apenas um carro por vez utilize o ponto de recarga, enquanto os demais são ordenados em fila.

A fila de espera passou a ser processada continuamente, com notificações em tempo real sobre a posição de cada carro na fila, garantindo maior transparência e previsibilidade na experiência do usuário.

4.4 Seleção de Pontos Próximos

Durante o processo de solicitação de recarga, o sistema foi configurado para sempre sugerir ao carro os pontos de recarga mais próximos, com base na sua posição atual. Esse critério visa otimizar o tempo de deslocamento e reduzir o tempo total de espera para a recarga, além de contribuir para uma distribuição mais equilibrada da carga entre os pontos disponíveis.

4.5 Avaliação Geral

Os testes demonstraram que o sistema é capaz de lidar com múltiplos carros em movimento, com solicitações simultâneas de reserva, e com o gerenciamento eficaz de filas em pontos de recarga. A arquitetura baseada em contêineres mostrou-se adequada para a simulação de ambientes realistas e escaláveis. Apesar das limitações iniciais relacionadas à comunicação em rede, o projeto está estruturado de forma que facilmente poderá ser adaptado para suportar ambientes distribuídos, interligando diversas máquinas em uma mesma simulação.

5. Conclusão

Dado o exposto, o projeto é coerente com os requisitos estabelecidos, demonstrando capacidade de gerenciar múltiplas conexões simultâneas de forma eficiente, por meio da utilização de sockets TCP e tratamento concorrente com goroutines. A arquitetura adotada permite a comunicação entre entidades distintas, como carros e pontos de recarga, garantindo troca de mensagens em tempo real e manutenção da integridade dos dados.

Apesar da solidez da base implementada, o projeto ainda não representa sua versão final ou mais otimizada. Há espaço considerável para evolução, especialmente em aspectos como o refinamento do tratamento de conectividade — por exemplo, detecção e recuperação automática de desconexões inesperadas, timeouts e reconexões. Além disso, a infraestrutura atual pode ser aprimorada para lidar melhor com conexões externas.

Por fim, há também oportunidades de modularização e abstração de responsabilidades no código, o que aumentaria a manutenção para facilitar futuras extensões, como interfaces gráficas ou integração com bancos de dados e APIs externas.

REFERÊNCIAS:

[1] ASSOCIAÇÃO BRASILEIRA DO VEÍCULO ELÉTRICO (ABVE). *94 mil eletrificados: 2023 bate todas as previsões*. Disponível em: https://abve.org.br/2023-supera-todas-as-previsoes-94-mil-eletrificados. Acesso em: 4 abr. 2025.

[2] ASSOCIAÇÃO BRASILEIRA DO VEÍCULO ELÉTRICO (ABVE). *Eletrificados superam previsões, passam de 170 mil e batem todos os recordes em 2024*. Disponível em: https://abve.org.br/eletrificados-superam-previsoes-passam-de-170-mil-e-batem-todos-os-recordes-em-2024. Acesso em: 4 abr. 2025.

[3] VITALINO, J. F. N.; CASTRO, M. A. N. *Descomplicando o Docker*. São Paulo: Novatec Editora, 2021.

[4] DONOVAN, A. A. A.; KERNIGHAN, B. W. *A linguagem de programação Go*. 1. ed. São Paulo: Novatec Editora, 2016.

[5] ALMEIDA, E. *Trabalhando com o protocolo TCP/IP usando Go*. iMasters, 2020. Disponível em: https://imasters.com.br/back-end/trabalhando-com-o-protocolo-tcpip-usando-go. Acesso em: 4 abr. 2025.

[6] RIBEIRO, F. *Sockets TCP na prática: comunicação cliente-servidor*. Embarcados, 2016. Disponível em: https://embarcados.com.br/socket-tcp/. Acesso em: 4 abr. 2025.

[7] CONTROLE.NET. *Cliente-servidor: uma estrutura para a computação centralizada*. ControleNet. Disponível em: https://www.controle.net/faq/cliente-servidor-uma-estrutura-para-a-computacao-centralizada#:~:text=Uma%20estrutura%20cliente%2Dservidor%20%C3%A9. Acesso em: 4 abr. 2025.