SmartCharge: Sistema Cliente-Servidor de Recarga de Veículos Elétricos

Kauan Caio de A. Farias[1], **Nathielle C. Alves**[1], **Vitória T. dos Santos**[1]

[1]UEFS – Universidade Estadual de Feira de Santana
Av. Transnordestina, s/n, Novo Horizonte
Feira de Santana – BA, Brasil – 44036-900

{fariak, nalves, vtsantos}@ecomp.uefs.br

Resumo. *O número de veículos elétricos tem crescido rapidamente, evidenciando sua expansão no mercado. No entanto, esse crescimento traz desafios, como a escassez de pontos de carregamento e a grande distância entre eles. Este trabalho apresenta o desenvolvimento de uma comunicação inteligente, padronizada e em tempo real entre veículos e os pontos de carregamento. O sistema adota o modelo cliente-servidor em nuvem para auxiliar os motoristas por meio da internet utilizando recursos na linguagem Golang, com foco em concorrência, comunicação via TCP/IP e sincronização de dados. Utilizou-se um protocolo JSON padronizado para garantir uma comunicação funcional e estruturada, obtendo resultados satisfatórios, mesmo diante de desafios de concorrência e sincronização.*

1. Introdução

O mercado de veículos elétricos (VEs) impulsionou um crescimento acelerado nas últimas décadas, impulsionado por avanços tecnológicos, políticas governamentais e conscientização ambiental. Segundo S&P Global Mobility, as vendas mundiais de carros elétricos devem aumentar cerca de 30% em 2025, atingindo a marca de 15,1 milhões de unidades comercializadas em relação aos anos anteriores.

Contudo, apesar do crescimento acelerado, essa inovação enfrenta obstáculos significativos. A expansão da infraestrutura de carregamento não tem acompanhado a demanda, resultando em escassez de pontos de recarga e em uma distribuição desigual, o que dificulta o uso dos veículos elétricos de forma eficiente.

Grande parte dos carregadores está concentrada em áreas urbanas e ao longo de algumas rodovias principais, o que limita o uso desses veículos em trajetos mais longos ou em regiões afastadas [Hardinghaus et al. 2019]. Além disso, mesmo nos centros urbanos, a quantidade de estações de recarga ainda é insuficiente para atender à crescente demanda, ocasionando filas e longos períodos de espera que acabam desestimulando os motoristas.

Neste cenário, este trabalho propõe um protocolo de comunicação inteligente e em tempo real entre veículos elétricos e pontos de carregamento, utilizando um modelo cliente-servidor baseado em nuvem. A solução busca otimizar a distribuição dos carregadores ao indicar ao motorista o local mais adequado para recarga, considerando tanto a ocupação atual dos postos quanto a distância até eles. Essa proposta visa reduzir filas, minimizar o tempo de espera e melhorar a eficiência no uso da infraestrutura existente.

Durante o desenvolvimento, surgiram desafios relacionados à concorrência e à sincronização das conexões TCP, como o conflito de leitura simultânea em goroutines e perda de mensagens entre os clientes e o servidor. Esses problemas exigiram uma reestruturação no fluxo de mensagens e no controle de acesso às conexões. O TCP (Transmission Control Protocol) é um protocolo orientado à conexão que estabelece uma comunicação confiável entre dois pontos finais, garantindo a entrega ordenada e íntegra dos dados [Billington and Han 2007].

Como solução, foi pensado centralizar as decisões no servidor, que passou a armazenar as informações de ID, nome e localização dos pontos de recarga, deixando-os apenas com o controle da fila de espera dos carros e o status de disponibilidade dos mesmos. Com essa abordagem, o sistema apresentou um bom desempenho, permitindo que os veículos fossem direcionados de forma eficiente aos pontos de recarga disponíveis, respeitando a ordem de chegada e o nível de bateria informado.

2. Fundamentação Teórica

2.1. Protocolo TCP/IP

O protocolo TCP/IP (Transmission Control Protocol/Internet Protocol) é a base da arquitetura de comunicação da Internet. Trata-se de um conjunto de protocolos projetados para permitir a interligação de redes heterogêneas, viabilizando a comunicação entre dispositivos distribuídos em diferentes localizações geográficas, como redes locais dentro de prédios, campi universitários ou entre continentes inteiros [Comer 2006].

Esse protocolo oferece um serviço confiável, orientado à conexão, garantindo que os dados enviados de um ponto a outro cheguem de forma íntegra e ordenada. Ele opera na camada de transporte do modelo TCP/IP e se apoia no IP para o roteamento dos pacotes entre os nós da rede

Protocolo é um termo que refere-se a um conjunto de regras que define a sintaxe, semântica e sincronização da comunicação entre dispositivos. Protocolos como o TCP e o IP descrevem como pacotes devem ser formatados, transmitidos, recebidos e interpretados, independentemente do hardware subjacente. Assim como um algoritmo permite abstrair a lógica computacional de uma linguagem de programação específica, os protocolos abstraem os detalhes da rede física, promovendo interoperabilidade [Comer 2006].

O conjunto TCP/IP é concebido com uma estrutura em camadas de protocolos, onde cada camada trata de uma parte específica do processo de comunicação. Essa organização modular facilita a manutenção, evolução e interoperabilidade dos sistemas distribuídos [Comer 2006]. A comunicação entre dois dispositivos se dá pela passagem das mensagens por essas camadas, desde a aplicação do emissor até a aplicação do receptor.

As principais camadas no modelo TCP/IP são:

- Camada de Aplicação: onde atuam os protocolos utilizados por softwares finais, como HTTP, SMTP, e DNS.
- Camada de Transporte: fornece serviços de transporte de dados fim a fim. Os dois principais protocolos aqui são o TCP e o UDP.
- Camada de Internet: responsável pelo roteamento dos pacotes entre redes, utilizando o protocolo IP.

- Camada de Acesso à Rede: trata da interface com o meio físico de transmissão (Ethernet, Wi-Fi etc.).

2.1.1. Diferença entre TCP e UDP

O TCP é ideal para aplicações que exigem confiabilidade, como transmissões de arquivos, requisições web e sistemas de controle em tempo real [Kurose and Ross 2010]. Internamente, o TCP controla a taxa de transmissão (controle de fluxo), detecta e corrige erros (verificação de integridade), e reenvia pacotes perdidos.

Ao contrário do TCP, o UDP é um protocolo de transporte não confiável e sem conexão, baseado em entrega por melhor esforço. Ele permite que aplicações enviem mensagens chamadas datagramas diretamente, sem estabelecer previamente uma conexão com o destinatário [Comer 2006].

O UDP não oferece garantias de entrega, não assegura a ordem das mensagens, nem implementa controle de congestionamento [Kurose and Ross 2010]. Por essa razão, é preferido em aplicações que priorizam velocidade e simplicidade sobre confiabilidade, como transmissões de vídeo ao vivo ou jogos online, como exemplificado na Tabela 1.

Table 1. Comparação entre os protocolos TCP e UDP

Característica	TCP	UDP
Conexão	Sim (orientado à conexão)	Não (sem conexão)
Confiabilidade	Alta (garante entrega e ordem)	Baixa (sem garantia de entrega)
Ordem das mensagens	Garante ordem	Pode chegar fora de ordem
Controle de Fluxo	Sim (controle e congestionamento)	Não
Correção de erros	Sim (verificação e retransmissão)	Simples (sem retransmissão)
Velocidade	Menor (mais sobrecarga)	Maior (menos sobrecarga)
Casos de uso típicos	Web, e-mail, transferência de arquivos	Streaming, jogos, VoIP

2.2. Modelo Cliente-Servidor

A arquitetura cliente-servidor constitui o modelo fundamental de comunicação em redes baseadas na pilha de protocolos TCP/IP. Esse paradigma organiza a interação entre programas de aplicação distribuídos de maneira eficiente e escalável, sendo amplamente utilizado tanto na Internet quanto em redes locais. Em essência, um servidor é um programa de aplicação que oferece um serviço, como fornecimento de dados ou execução de processos, enquanto um cliente é um programa que requisita esse serviço, geralmente iniciando a comunicação [Comer 2000].

O modelo funciona com base em um padrão simples: o cliente envia uma requisição por meio da rede, e o servidor recebe, processa e responde a essa requisição. Isso permite que a aplicação do lado cliente permaneça relativamente leve, delegando tarefas mais pesadas ou específicas ao lado servidor. Um exemplo clássico é um servidor web, que recebe requisições HTTP de navegadores e retorna os conteúdos das páginas requisitadas.

A grande vantagem desse modelo é sua portabilidade e modularidade. Como os

servidores são programas em execução (processos), eles podem ser executados em qualquer sistema computacional que suporte comunicação TCP/IP, podendo ainda ser replicados para melhorar desempenho e tolerância a falhas. Além disso, esse modelo também é facilmente extensível e serve de base para arquiteturas mais complexas como peer-to-peer, computação em nuvem e modelos distribuídos como MapReduce — todos esses ainda dependem, em seus níveis mais básicos, da interação cliente-servidor para comunicação entre nós [Comer 2000].

As goroutines são um recurso da linguagem Golang utilizados para manter a comunicação paralela entre o servidor e múltiplos clientes, permitindo escutar e responder simultaneamente a diversas solicitações sem bloqueios. Entretanto, o uso incorreto dessas estruturas pode levar a condições de corrida (race conditions), leitura simultânea da mesma conexão e perda de dados — problemas enfrentados e corrigidos no desenvolvimento do sistema por meio da reestruturação do fluxo de mensagens.

2.3. Docker e Linguagem Golang

Docker é uma plataforma que permite empacotar, distribuir e executar aplicações em ambientes isolados chamados contêineres. Esses contêineres compartilham o kernel do sistema operacional, mas funcionam de forma independente (como ilustrado na Figura 2), garantindo portabilidade e consistência na execução, especialmente útil em sistemas distribuídos para facilitar o desenvolvimento, testes e deploy.

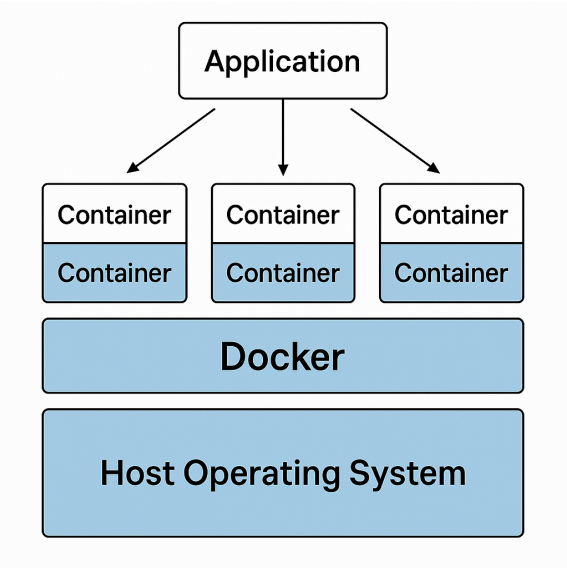

Figure 1. Diagrama de ilustração do funcionamento de containers Docker

A linguagem Go (ou Golang), desenvolvida pelo Google, é conhecida por sua simplicidade, desempenho e suporte nativo a concorrência. Uma de suas principais vantagens é o uso de goroutines, estruturas leves de execução concorrente que compartilham o mesmo espaço de memória e são gerenciadas pela própria runtime da linguagem.

Go oferece um modelo simples de concorrência com canais (chan) para comunicação segura entre goroutines, eliminando boa parte da complexidade associada

a threads tradicionais. Essas características foram determinantes para a escolha da linguagem no desenvolvimento do sistema, pois possibilitaram a criação de um servidor capaz de gerenciar múltiplas conexões simultâneas de forma eficiente e com baixo custo computacional.

Embora o sistema utilize conexões TCP puras em vez de WebSockets, o conceito de comunicação em tempo real é essencial. WebSockets são uma abstração de comunicação persistente e bidirecional sobre TCP, amplamente utilizada em aplicações web modernas. Em Go, esse comportamento pode ser simulado por meio de conexões TCP tradicionais combinadas com goroutines para leitura e escrita assíncronas.

3. Metodologia

A metodologia adotada neste projeto visa a implementação e avaliação de um sistema de comunicação entre veículos elétricos e estações de recarga, utilizando uma arquitetura cliente-servidor baseada no protocolo TCP. O objetivo é simular um ambiente onde carros elétricos monitoram seus níveis de bateria e, ao atingir um nível crítico, buscam a estação de recarga mais próxima para reabastecimento.

3.1. Ferramentas e Tecnologias

O projeto foi desenvolvido em linguagem Go, utilizando as bibliotecas padrão para comunicação via rede, manipulação de JSON e controle concorrente. A execução foi feita em um ambiente virtualizado com Docker para simular os diferentes componentes do sistema.

3.2. Arquitetura do Sistema

O sistema é composto por três componentes principais:

- Cliente (Carro): Simula um carro elétrico que se movimenta aleatoriamente, monitora seu nível de bateria e se comunica com o servidor.
- Servidor: Coordena a comunicação entre os carros e as estações de recarga, processando requisições e identificando o posto de recarga mais próximo.
- Estação de Recarga: Responsável por receber solicitações de carregamento e gerenciar a fila de atendimento.

3.2.1. Fluxo de Comunicação

A interação entre os componentes do sistema segue um fluxo estruturado para garantir a troca eficiente de informações entre os carros elétricos, o servidor e as estações de recarga. O diagrama a seguir ilustra esse fluxo, detalhando desde a verificação do nível da bateria até a finalização do carregamento e pagamento.

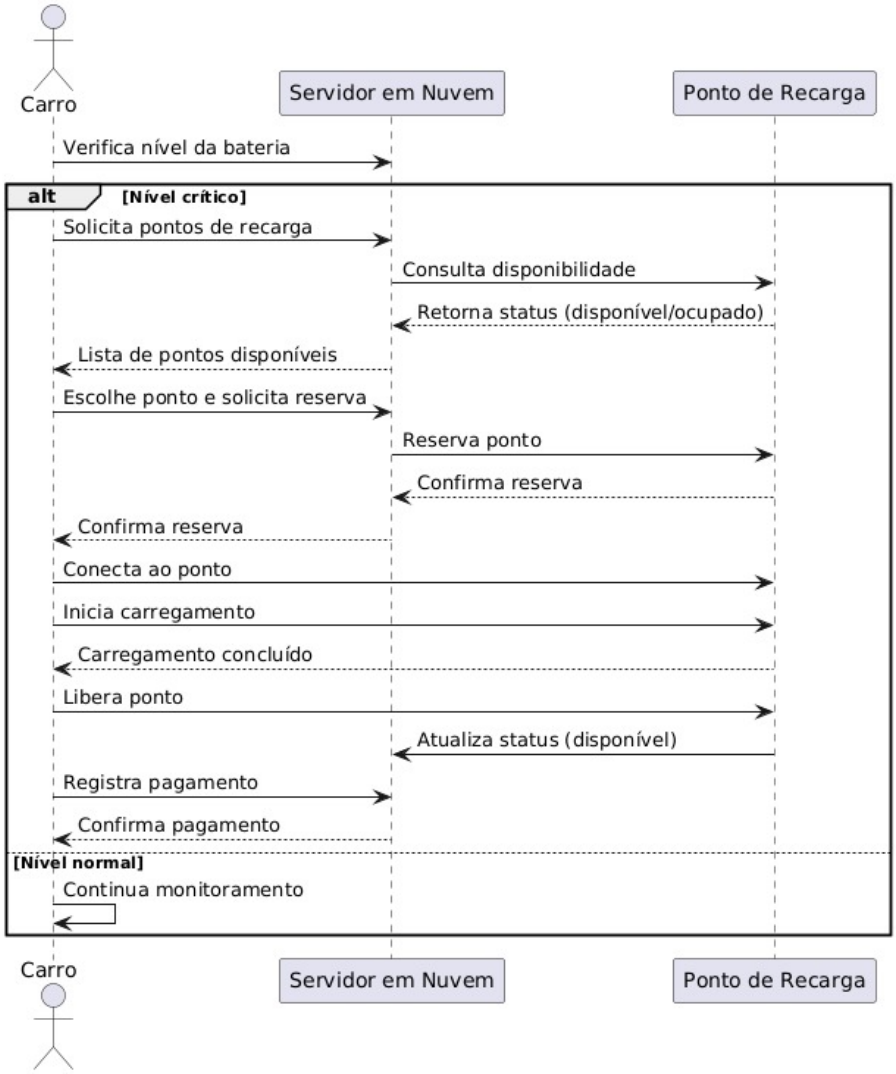

Figure 2. Diagrama de sistemas

3.3. Implementação

3.3.1. Cliente (Carro)

Cada carro gera um identificador único e inicia a comunicação com o servidor TCP. Ele simula o movimento dentro de uma área delimitada, reduzindo progressivamente seu nível de bateria. Quando a bateria atinge um nível crítico de 20%, o carro solicita assistência ao servidor para que o posto de recarga mais próximo seja designado.

Principais funcionalidades:

- Gerenciamento de estado do veículo (localização, nível de bateria, nível crítico);
- Comunicação com o servidor via protocolo TCP;
- Simulação de movimentação e consumo de bateria;
- Envio de alertas ao servidor em caso de bateria crítica.

3.3.2. Servidor

O servidor gerencia as conexões dos carros e estações de recarga, armazenando suas informações em uma estrutura compartilhada protegida por mutexes. Ele processa os dados recebidos dos carros e calcula a melhor estação com base na distância e disponibilidade. Após a identificação da estação adequada, o servidor envia a resposta ao carro informando onde ele deve se dirigir para recarga.

Principais funcionalidades:

- Registro de conexões de carros e estações;
- Processamento de dados dos clientes;
- Cálculo da melhor estação de recarga com base na localização do carro;
- Encaminhamento de solicitações de recarga para as estações disponíveis.

3.3.3. Estação de Recarga

Cada estação de recarga adota um algoritmo de fila FIFO (First In, First Out) para gerenciar os pedidos de recarga, garantindo que os carros sejam atendidos na ordem de chegada. Além disso, elas recebem os pedidos de recarga e informam sua disponibilidade.

Principais funcionalidades:

- Registro e gerenciamento de carros na fila de recarga;
- Comunicação com o servidor para atualização de disponibilidade;
- Processamento de solicitações de atendimento.;

3.4. Técnicas e Métodos

- **Algoritmo de Seleção da Estação de Recarga:** Um algoritmo de menor caminho baseado na distância euclidiana é utilizado para identificar a estação mais próxima disponível.
- **Gerenciamento de Conexões:** Utilização de *goroutines* para permitir que múltiplos clientes sejam processados simultaneamente sem bloqueios.
- **Serialização de Dados:**Os dados trocados entre os componentes são estruturados em JSON para garantir compatibilidade entre sistemas.
- **Simulação de Movimentação:** Os carros simulam deslocamento aleatório dentro de uma área predefinida, alterando suas coordenadas ao longo do tempo.

4. Resultados

Os testes demonstraram que o terminal exibe corretamente a redução progressiva dos níveis de bateria e a atualização das coordenadas dos carros elétricos. Quando a bateria atinge um nível crítico, uma mensagem de alerta é exibida no terminal, e os dados do carro são enviados ao servidor. O servidor então processa a solicitação, identifica a estação de recarga mais próxima disponível e retorna a resposta para o carro, que recebe a informação e pode simular sua movimentação até a estação.

As estações de recarga também respondem corretamente às requisições, gerenciando a fila de atendimento e atualizando seu status de disponibilidade.

4.1. Discussão sobre Falhas

Apesar do funcionamento adequado, algumas limitações e falhas foram observadas:

- **Latência na comunicação:** Em cenários com um grande número de carros simultâneos, houve aumento no tempo de resposta do servidor devido ao gerenciamento das conexões e ao processamento das requisições.
- **Gerenciamento da fila de recarga:** Em alguns casos, a distribuição dos carros para as estações não foi a mais eficiente, resultando em tempos de espera elevados para alguns veículos.
- **Condições de corrida:** Durante os testes, verificou-se a necessidade de aprimorar o controle de concorrência para evitar acesso simultâneo não sincronizado a certas estruturas de dados.

5. Conclusão

A metodologia adotada permitiu a implementação eficiente da comunicação entre os carros elétricos e as estações de recarga. A arquitetura baseada em sockets TCP se mostrou adequada para a troca de informações em tempo real, possibilitando uma simulação realista do funcionamento de um sistema de recarga de veículos elétricos.

Entretanto, algumas melhorias poderiam ser realizadas, como a otimização do gerenciamento da fila de recarga e aprimoramento da concorrência para reduzir latências e evitar condições de corrida. A implementação de um algoritmo mais eficiente para seleção de estações também poderia contribuir para reduzir o tempo de resposta do sistema. Ao longo do projeto, foi possível adquirir experiência em programação concorrente, manipulação de sockets e serialização de dados, além de aprender sobre o impacto da concorrência e da sincronização em sistemas distribuídos.

References

Billington, J. and Han, B. (2007). Modelling and analysing the functional behaviour of tcp's connection management procedures. *STTT*, 9:269–304.

Comer, D. E. (2000). *Internetworking with TCP/IP: Client-Server Programming and Applications*. Prentice Hall, Upper Saddle River, NJ.

Comer, D. E. (2006). *Interligação em Rede com TCP/IP: Princípios, Protocolos e Arquitetura*, volume 1. Campus/Elsevier, Rio de Janeiro, 5 edition.

Hardinghaus, M., Seidel, C., and Anderson, J. E. (2019). Estimating public charging demand of electric vehicles. *Sustainability*, 11:5925.

Kurose, J. F. and Ross, K. W. (2010). *Redes de Computadores e a Internet: Uma Abordagem Top-Down*. Pearson, São Paulo, 5 edition.