Problema 1 - EASYCHARGER

[1] **Robson Jones,** [2] **Henrique Zéu,** [3] **Luan Barbosa**

[1]Departamento de Tecnologia – Universidade Estadual de Feira de Santana (UEFS)
44036–900 – Feira de Santana – Bahia

robson.uefs.2013@gmail.com, henrique.zeuu@gmail.com, luanbscosta@hotmail.com

Resumo. *A frota de veículos elétricos vem aumentando consideravelmente a cada ano. A quantidade elevada de veículos elétricos causou uma carência de postos de recarga. Foi proposto a criação de uma aplicação que gerencia as filas de carros em cada um dos postos de recarga. O sistema implementa em Python uma arquitetura cliente-servidor, utilizando Sockets, sincronização e Threads para tratamento de concorrência e gerenciamento desses postos. Os veículos podem fazer reservas e entrar em uma fila de recarga para o posto mais próximo reduzindo a possibilidade de espera, além de receber notificações para estados de bateria.*

1. Introdução

Com o objetivo de facilitar o uso na utilização de carros elétricos foi proposto o desenvolvimento de uma aplicação capaz de gerenciar um sistema de postos de recarga. O sistema deve ser capaz de adicionar novos carros, identificar o status de bateria de cada carro e por fim informar o posto de recarga mais próximo e que também esteja desocupado para agilidade na recarga.

O sistema deve ser implementado com foco em exercitar os conhecimentos sobre comunicação em redes utilizando *Sockets* e tratamento do concorrência de dados. Para o exercício do conhecimento deve-se implementar o sistema de forma que ele possua um Servidor Principal para comunicação e direcionamento de cada veiculo para os respectivos postos necessários e uma aplicação para o cliente, que é representado pelos próprios veículos que farão a recarga. Os postos responsáveis já são pré-cadastrados na inicialização do sistema.

Este trabalho está organizado da seguinte forma. A Seção 2 apresenta os fundamentos teóricos relacionados com o problema. A Seção 3 discute aspectos de implementação e testes da solução. A Seção 4 apresenta e avalia os resultados. No final, na Seção 5, as conclusões e reflexões sobre os conhecimentos adquiridos.

2. Fundamentação Teórica

O problema tem como fundamentação o uso de arquitetura Cliente-Servidor para sistemas distribuídos, a aplicação do Protocolo TCP/IP, o uso de *Sockets* para envio de pacotes na redes e o uso de *Threads* para lidar com a concorrência de pacotes simultâneos.

2.1. Arquitetura Cliente-Servidor

A arquitetura cliente-servidor é um modelo de construção de um *software* que tem como característica a divisão em 2 tipos de aplicação. O cliente é responsável por mandar

informações sobre os atributos necessários para o funcionamento do sistema ao servidor, este possui a função de receber esses dados, processá-los e devolver o resultado aos clientes, conforme houver necessidade.

2.2. Protocolo TCP/IP

O protocolo TCP/IP é um conjunto de instruções que possibilita a comunicação entre computadores e servidores combinando o *Transmission Control Protocol*(TCP) com o *Internet Protocol*(IP). Este protocolo combina as camadas de de aplicação, transporte, de rede e a de enlace para identificar o tipo de pacote que será enviado, definir a porta que o pacote será transmitido os dados, localizar o destino de envio do pacote através do IP e por fim concretizar o envio do conteúdo da transmissão.

2.3. Utilização de Sockets

Socket é uma *interface* de comunicação utilizada para troca de dados entre dispositivos em uma rede, seu funcionamento é dependente de um endereço IP e um número de porta. Os Sockets possuem uma conexão bidirecional entre processos, podendo ser de uma mesma máquina ou de uma máquina diferente, existem diversos tipos porém os principais são os de *Stream*(TCP) e os de datagrama(UDP).

2.4. Utilização de Threads

Threads são uma forma de código capaz de executar múltiplas tarefas simultaneamente, isso é útil no âmbito de redes pois é possível tratar múltiplas requisições simultaneamente e tratar a prioridade para conexão com o Servidor para caso vários clientes solicitem uma requisição ao mesmo tempo.

3. Metodologia, Implementação e Testes

O problema foi resolvido através de várias reuniões em grupo, execução de testes e consulta de informações em materiais oficiais. Na primeira reunião foi discutido sobre o fato de que os postos devem possuir reservas e que o cliente, os veículos, devem possuir um sinalizador e um índice de bateria para consultar a capacidade dela.

Na segunda sessão foi discutido o fato da usabilidade do protocolo TCP/IP, se tornando necessário para o prosseguimento da aplicação, além da utilização da ferramenta *Docker* para virtualização do projeto. Na terceira sessão foi constatado que não seria possível utilizar *Frameworks* para facilitar a troca de dados pois anularia o objetivo do projeto.

Na quarta sessão foi confirmado que o veiculo precisaria de um *Timer* para simular o consumo da bateria no trajeto. Após sessões de estudos individuais, houve o esclarecimento de que o projeto precisaria de ser dividido em 3 aplicações, uma para o posto, uma para o servidor central e a última para o veiculo. A implementação de um servidor TCP para a conexão do carro para o servidor principal foi confirmada para esse tipo de comunicação enquanto que para a comunicação entre o posto e o servidor foi decidido o desenvolvimento de um servidor UDP para que fosse realizado constantes requisições com destino ao servidor principal.

O projeto foi implementado utilizando a linguagem de programação *Python3*, os membros utilizaram o editor de texto *VS Code*, com complementos certificados pela *Microsoft* para desenvolvimento da aplicação. Testes unitários foram realizados para testar

cada função, e um servidor VPS (*Virtual Private Server*) foi utilizado para testes de rede fora do ambiente laboratorial. O laboratório *Larsid* foi o local final para os testes finais de desempenho da aplicação.

4. Resultados e Discussões

Inicialmente, o sistema foi desenvolvido utilizando apenas dois componentes principais: o cliente (representado pelos veículos) e o servidor (responsável por registrar os veículos e alocar os postos de recarga). Nessa primeira versão, os postos eram fixos e carregados diretamente na inicialização do servidor, o que limitava a escalabilidade e dificultava o gerenciamento dinâmico dos recursos.

Após feedback e revisão do projeto, foi solicitada uma alteração na arquitetura, de forma que os postos fossem executados como containers independentes e enviassem periodicamente seus dados de status ao servidor via comunicação UDP. Essa mudança tornou o sistema mais modular e escalável, além de permitir que os postos fossem distribuídos em diferentes máquinas ou redes.

Com essa nova abordagem, cada posto envia uma requisição UDP ao servidor contendo seu identificador e estado atual (livre ou ocupado). O servidor mantém uma lista atualizada dos postos disponíveis com base nas últimas mensagens recebidas, o que reduz significativamente o tempo de resposta no momento da escolha do posto mais próximo e disponível para cada veículo.

Para avaliar o desempenho do sistema, realizamos testes em uma VPS do tipo KVM1 da Hostinger, onde conseguimos simular com sucesso 150 postos e 150 clientes simultaneamente. O sistema manteve-se estável durante os testes, com baixo consumo de memória e boa capacidade de resposta, demonstrando viabilidade técnica e eficiência mesmo em um ambiente com recursos limitados.

Essa mudança na arquitetura provou-se essencial para lidar com múltiplos acessos simultâneos, tornando a comunicação entre cliente-servidor e postos mais fluida e confiável. A utilização do protocolo UDP para os postos contribuiu para uma maior performance, visto que sua natureza não-conexa permite o envio de informações sem a sobrecarga de conexão constante.

5. Conclusão

O desenvolvimento deste projeto permitiu aplicar e consolidar conhecimentos importantes sobre redes de computadores, especialmente no que diz respeito à arquitetura cliente-servidor, uso de sockets TCP e UDP, e manipulação de concorrência com threads. Ao longo da implementação, enfrentamos desafios reais de design de sistemas distribuídos, como escalabilidade, sincronização e desempenho.

A evolução da arquitetura do sistema, saindo de uma abordagem monolítica para uma baseada em containers independentes que se comunicam via UDP, demonstrou na prática como pequenas decisões de design podem impactar positivamente a flexibilidade e robustez de uma aplicação em rede. A capacidade de executar testes com 150 postos e 150 usuários simultâneos em uma VPS simples mostra que o sistema é promissor e possui potencial de crescimento.

Além do ganho técnico, o projeto também contribuiu para o desenvolvimento de habilidades em trabalho em equipe, organização de código, testes e documentação. Consideramos que o problema proposto foi interessante, desafiador na medida certa, e teve grande valor didático. Como sugestão, poderíamos melhorar ainda mais o sistema adicionando autenticação dos postos e persistência de dados para garantir maior segurança e confiabilidade a longo prazo.

Em suma, a experiência foi enriquecedora e certamente será útil na formação acadêmica e profissional de todos os envolvidos.

Referências

Hostinger (2024). O que é protocolo tcp/ip e como ele funciona? Disponível em: https://www.hostinger.com.br/tutoriais/tcp-ip. Acessado em: 06 fev. de 2025.

IBM (2025). O modelo cliente/servidor. Disponível em: https://www.ibm.com/docs/pt-br/cics-ts/5.6.0?topic=programs-clientserver-model. Acessado em: 06 fev. de 2025.

Python. Socket progamming howto. Disponível em: https://docs.python.org/3/howto/sockets.html. Acessado em: 06 fev. de 2025.

[IBM 2025] [Hostinger 2024] [Python]