

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Redes de Computadores

Trabalho prático 2

Sistema de preços de combustíveis

Aluno: Caíque Bruno Fortunato
Matrícula: 2013062731

Belo Horizonte, Maio de 2017

Introdução

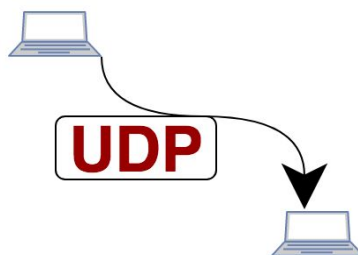


Imagem 1: UDP

Socket é um componente do sistema que permite a comunicação entre processos através da rede por meio dos protocolos de transporte, sendo também uma abstração computacional que meia diretamente a uma porta de transporte mais um endereço de rede.

A comunicação de processos por meio de rede é um tópico da computação que está cada vez mais comum, uma vez que atualmente muitos recursos ficam armazenados em servidores e acessados por clientes. Sendo assim, navegadores web utilizam sockets para requisitar páginas, sistemas também usam o recurso para comunicar com banco de dados, entre várias outras aplicações.

Existem dois lados para a comunicação: a do cliente e a do servidor, que precisam estar usando o mesmo protocolo. O UDP, que foi utilizado na implementação do programa, é um serviço sem conexão que não fornece um canal pré-estabelecido entre os processos, desconsiderando a ordem de pacotes, recuperação de falhas e garantia de ordem.

O principal objetivo do trabalho é colocar em prática os conceitos listados acima através da construção de uma aplicação que fornece o menor preço de combustível dada uma lista de preços e localizações de postos e um raio, dado um ponto inicial.

Nas próximas seções serão detalhados: descrição da implementação do programa e da conexão de rede, testes e conclusão.

Sistema de preços de combustíveis

Funcionamento

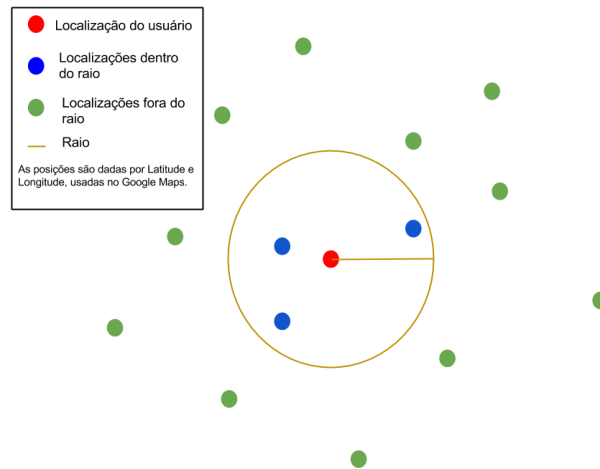


Imagem 2: Exemplo do funcionamento da pesquisa

O sistema de preço de combustíveis funciona através da comunicação entre o cliente e servidor da seguinte maneira:

1. O cliente envia uma mensagem para o servidor que pode conter:
 - a. **Dados:** Possui como objetivo informar as coordenadas dos postos existentes e o preço do combustível.
 - b. **Pesquisa:** Possui como objetivo pesquisar o menor preço de combustível que está dentro de um raio estabelecido de uma localização.
2. O servidor responde o cliente da seguinte maneira:
 - a. **Dados:** Confirma a inserção e o recebimento da mensagem.
 - b. **Pesquisa:** Confirma o recebimento da mensagem e informa o menor preço encontrado.

A entrada dos dados precisa obedecer o seguinte padrão:

1. **Tipo de mensagem:** [D] Para inserção de dados e [P] para pesquisa
 - a. Para mensagem de dados:
 - i. **Tipo de combustível:** [0] Diesel, [1] Álcool, [2] Gasolina

- ii. **Preço do combustível:** Valor x 1000. Exemplo: 4991
- iii. **Coordenadas geográficas:** Latitude e longitude.
- b. Para mensagem de pesquisa:
 - i. **Tipo de combustível:** [0] Diesel, [1] Álcool, [2] Gasolina
 - ii. **Raio de busca:** Valor em metros. Exemplo: 5 metros
 - iii. **Coordenadas do ponto de origem:** Latitude e longitude.

Implementação

Neste tópico serão listados e apresentados os métodos que manipulam o funcionamento do programa para encontrar o menor preço de combustível, tanto pela ótica do cliente quanto pela do servidor. O algoritmo foi escrito na linguagem de programação Python 3.6.

Cliente

- **recebe_dados_user:** Método que manipula as entradas do usuário de acordo com o que foi detalhado no tópico anterior e monta a mensagem final que será enviada para o servidor. Exemplos:
 - *P|0|1|100|-19.768042|-43.88599*, onde os valores separados por “|” significam respectivamente: tipo de mensagem, id da mensagem, tipo de combustível, raio, latitude e longitude.
 - *D|1|1|7662|-19.774147|-43.881119*, onde os valores separados por “|” significam respectivamente: tipo de mensagem, id da mensagem, tipo de combustível, preço do combustível, latitude e longitude.
 - S: Encerra o cliente.
- **manipula_resposta:** Método que manipula a resposta enviada pelo servidor. A resposta pode ser:
 - *Id*. Neste caso, o id recebido deverá ser maior ou igual a 0 e idêntico a da mensagem que foi enviada. Se for -1 ou não corresponder ao último envio houve erro.
 - *Id, preço e coordenadas*. O id representa a mesma regra do tópico anterior. Caso a resposta tenha mais informações a mensagem foi de

pesquisa e deverá conter também o menor preço e as coordenadas correspondentes.

- Exemplo: 0|0999|-43.881119|-19.774147: Menor valor: 0999 na latitude -19.774147 e longitude -43.881119

Servidor

- **interpreta_mensagem:** Identifica o tipo de mensagem recebida, se é para pesquisa ou para inserção de dados.
- **escreve_arquivo:** Interpreta a mensagem recebida e salva no arquivo correspondente os dados de: preço, latitude e longitude.
- **retorna_combustivel:** Retorna o nome do arquivo correspondente ao combustível que o cliente escolheu para realizar a operação (P/D). Para cada tipo de combustível o programa cria um arquivo para inserção/busca que mantêm o dados salvos para pesquisas ou inserções futuras.
- **pesquisa_posto:** Pesquisa o posto que possui menor valor do preço do combustível escolhido dentro do raio estabelecido. Para isso existem: coordenadas do ponto de origem e do posto de combustível. Sendo assim, é feito um cálculo da distância. Caso a distância entre os pontos seja menor que o raio, o menor preço é comparado e definido. (Imagem 2)
- **distance:** dado lat1, lat2, lon1 e lon2 é feito o seguinte cálculo da distância:
 - $d = 6378.137 * \text{acos}(\cos(\text{lat1}) * \cos(\text{lat2}) * \cos(\text{lon2} - \text{lon1}) + \sin(\text{lat1}) * \sin(\text{lat2}))$

Comunicação via socket

A comunicação entre o cliente e servidor é feita utilizando a biblioteca socket presente na linguagem Python 3.

Como já detalhado na introdução, socket é um componente do sistema que permite a comunicação entre processo através da rede por meio dos protocolos de transporte. É também uma abstração computacional que mapeia diretamente a uma porta de transporte (UDP, neste caso) mais um endereço de rede. Sendo assim, é

possível identificar unicamente um aplicativo ou servidor na rede de comunicação IP.

Ao utilizar o UDP os dados transmitidos podem ser recebidos fora de ordem ou perdidos. Isso porque na comunicação não há conexão entre o cliente e servidor, o transmissor envia explicitamente endereço IP e porta de destino em cada mensagem e o servidor deve extrair o endereço IP e porta do transmissor de cada pacote recebido.

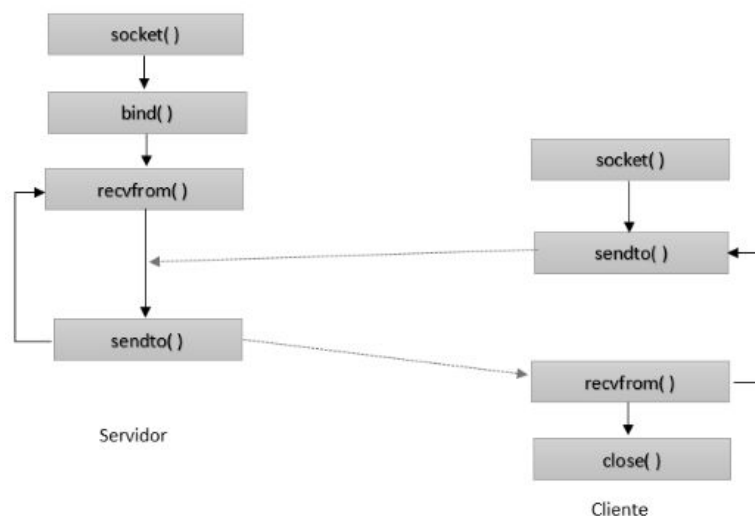


Imagem 3: Comunicação via UDP

Funcionamento e implementação

De maneira geral, foram utilizadas as funções básicas da biblioteca socket.

O servidor cria um socket em IPv6 e UDP, faz o uso do bind com a porta (::) e a enviada por parâmetro. Após isso ele entra em Loop. Se receber a mensagem do cliente, ele interpreta e envia uma mensagem de volta (utilizando os métodos `recvfrom` e `sendto` respectivamente).

O cliente também cria um socket em IPv6 e UDP. Entra em Loop até a conexão ser encerrada. Ele envia uma mensagem para o servidor e recebe uma confirmação ou dados de volta (utilizando os métodos `recvfrom` e `sendto` respectivamente).

Retransmissão de mensagens

Para contornar o problema da perda de dados, conforme definido na especificação, o cliente deverá fazer o reenvio da mensagem para o servidor pelo menos uma vez, caso a resposta não seja recebida. Para resolver esse problema foi utilizada a seguinte estratégia:

1. O usuário insere os dados, o programa interpreta.
2. Try:
 - a. Tenta enviar a mensagem
 - b. Utiliza a função settimeout para temporizar 5 segundos
 - c. Espera a resposta do cliente
 - d. Se não houver retorno em 5 segundos, entra no Except
3. Except:
 - a. Possui um novo Try / Except. Se entrar no segundo except o programa descarta a mensagem e solicita que o usuário entre com os dados novamente.

```
def envio_mensagem(client_socket, msg_envio, server_name, server_port, id_resp)
:
    client_socket.sendto(msg_envio.encode(), (server_name, server_port))
    print("Mensagem enviada: "+msg_envio)
    client_socket.settimeout(5.0)
    recebe_mensagem(client_socket, id_resp)
    client_socket.settimeout(None)

while True :
    msg_envio = recebe_dados_user(id_msg)
    if(msg_envio == "S") : break

    try:
        envio_mensagem(client_socket, msg_envio, server_name, server_port,
id_resp)
    except Exception as e:
        print("Resposta nao recebida, tentando novamente")
        try :
            envio_mensagem(client_socket, msg_envio, server_name, server_port,
```

```
id_resp)
    except Exception as e:
        print("Impossível receber uma resposta. Digite novamente\n")
```

Conversão de IPv4 para IPv6

O IPv4 transfere endereços de protocolos de 32 bits enquanto no IPv6 os IPs trabalham com 128 bits. Embora todos os serviços estejam sendo migrados para o novo padrão, ainda existe o problema das técnicas de adaptação do IPv4 para IPv6. Até alguns anos atrás estima-se que cerca de 90% dos computadores usavam o padrão antigo.

Neste trabalho o programa utiliza IPv6 como padrão de comunicação entre o cliente e servidor por padrão. A técnica utilizada para não ter problema com as versões foi a seguinte: caso uma entrada do host do cliente seja IPv4 o programa irá converter para IPv6 (Ex: 127.0.0.1 → ::FFFF:127.0.0.1), assim a conexão pode ser estabelecida independente do tipo de IP utilizado.

Como saber a arquitetura do ip digitado?

Conforme descrito na documentação, o IP pode ser informado por número ou por nome, exemplo: login.dcc.ufmg.br. Além disso, pode ser tanto na arquitetura IPv4 quanto IPv6. A biblioteca socket do Python 3 possui uma função chamada getaddrinfo que informa diversas informações sobre o endereço informado.

Segundo a documentação a função: *“Translate the host/port argument into a sequence of 5-tuples that contain all the necessary arguments for creating a socket connected to that service. host is a domain name, a string representation of an IPv4/v6 address or None. port is a string service name such as 'http', a numeric port number or None. By passing None as the value of host and port, you can pass NULL to the underlying C API.”*

Sendo assim, o primeiro argumento possui o tipo de arquitetura, IPv4 ou IPv6 e o último a sequência numérica do IP. Se o IP informado for somente do tipo IPv4

(exemplo 127.0.0.1) é realizada a conversão, caso contrário é mantido o endereço obtido.

Testes

Foram realizados diferentes testes para verificar o funcionamento correto do programa. Entre eles um caso real onde foi feita uma pesquisa de postos de combustíveis próximos da localidade atual no Google Maps. Para cada posto foi obtido a latitude e longitude e definido um preço aleatório.

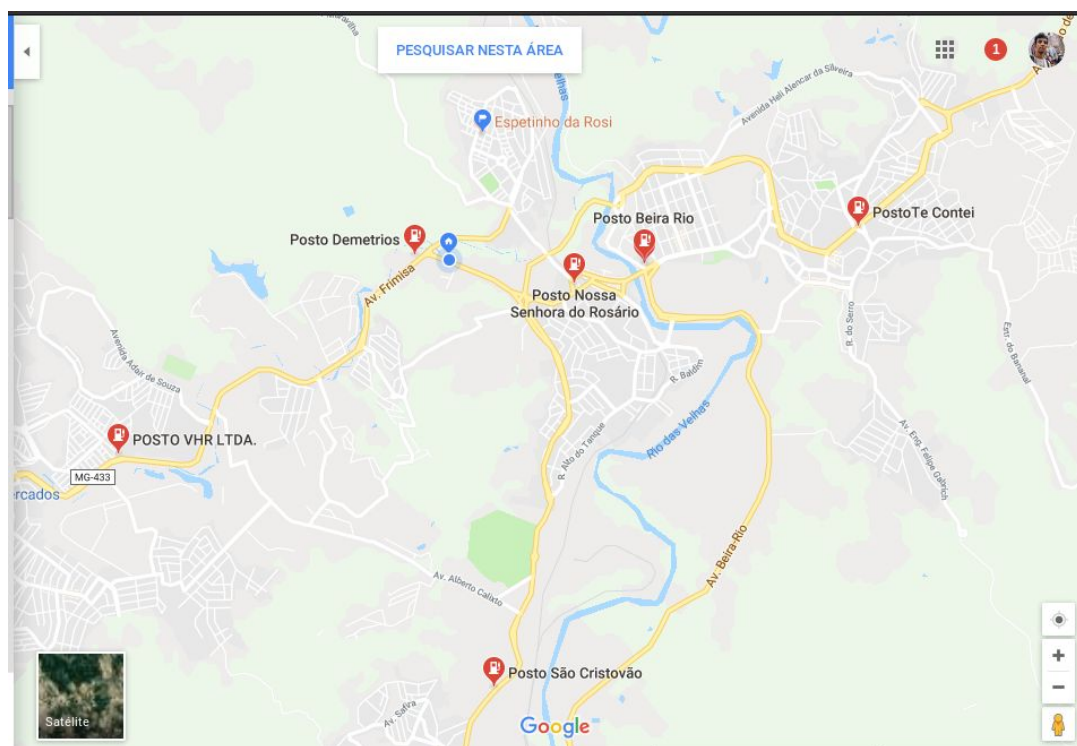


Imagem 4: Postos de combustíveis

Além dos postos presentes na imagem abaixo foi definido também um posto que se encontra mais distante e em outra cidade.

Inicialização dos programas

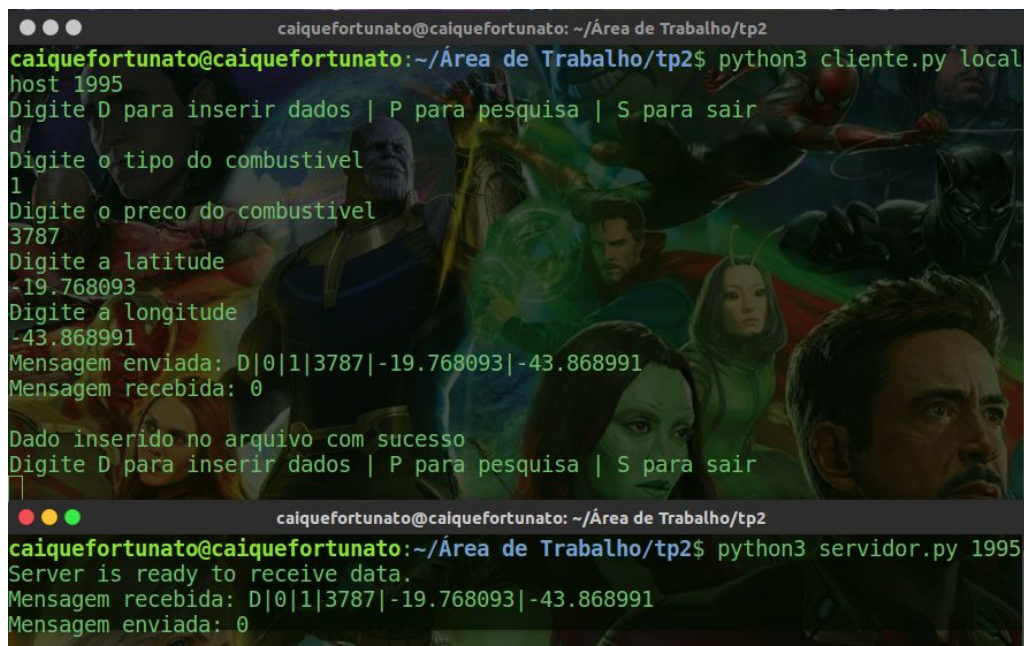
Primeiro foi inicializado o servidor, que deve ter como parâmetro a porta. Sendo assim, foi executado na linha de comando: ***python3 servidor.py 1995***

Logo depois, em outro terminal foi inicializado o cliente, que deve ter como parâmetro o ip/nome e a porta. Sendo assim, foi executado na linha de comando:

python3 cliente.py localhost 1995

Funcionamento da inserção de dados

O primeiro dado inserido foi o do Posto Beira Rio. Foi definido que o álcool custa 3787. O envio foi feito para o servidor que retornou o id da mensagem.



```
caiquefortunato@caiquefortunato: ~/Área de Trabalho/tp2
caiquefortunato@caiquefortunato:~/Área de Trabalho/tp2$ python3 cliente.py localhost 1995
Digite D para inserir dados | P para pesquisa | S para sair
d
Digite o tipo do combustível
1
Digite o preço do combustível
3787
Digite a latitude
-19.768093
Digite a longitude
-43.868991
Mensagem enviada: D|0|1|3787|-19.768093|-43.868991
Mensagem recebida: 0
Dado inserido no arquivo com sucesso
Digite D para inserir dados | P para pesquisa | S para sair
]

caiquefortunato@caiquefortunato: ~/Área de Trabalho/tp2
caiquefortunato@caiquefortunato:~/Área de Trabalho/tp2$ python3 servidor.py 1995
Server is ready to receive data.
Mensagem recebida: D|0|1|3787|-19.768093|-43.868991
Mensagem enviada: 0
```

Imagem 5: Teste 1

Logo depois, foram inseridos os dados dos demais postos, que correspondem ao da imagem 4 e mais um, que é o mais distante que não aparece na imagem, mas que possui os seguintes dados: tipo de combustível: 1, preço: 1523, latitude: -19.805947 e longitude -43.971081.

```
caiquefortunato@caiquefortunato: ~/Área de Trabalho/tp2
Server is ready to receive data.para pesquisa | S para sair
Mensagem recebida: D|0|1|3787|-19.768093|-43.868991
Mensagem enviada: 0;bustivel
1
Mensagem recebida: D|1|1|3788|-19.765124|-43.889161
Mensagem enviada: 1
Digite a latitude
Mensagem recebida: D|2|1|3765|-19.768436|-43.875342
Mensagem enviada: 2
-43.970605
Mensagem recebida: D|3|1|3789|-19.763670|-43.847962
Mensagem enviada: 3
Mensagem recebida: D|4|1|3790|-19.801145|-43.881436
Mensagem enviada: 4;ar dados | P para pesquisa | S para sair
Mensagem recebida: D|5|1|4191|-19.813351|-43.875193
Mensagem enviada: 5
Mensagem recebida: D|6|1|4101|-19.783431|-43.912186
Mensagem enviada: 6
```

Imagem 6: Teste 2

Funcionamento do cálculo da distância

Para conferir o funcionamento correto do cálculo da distância foi utilizado a ferramenta do Google.

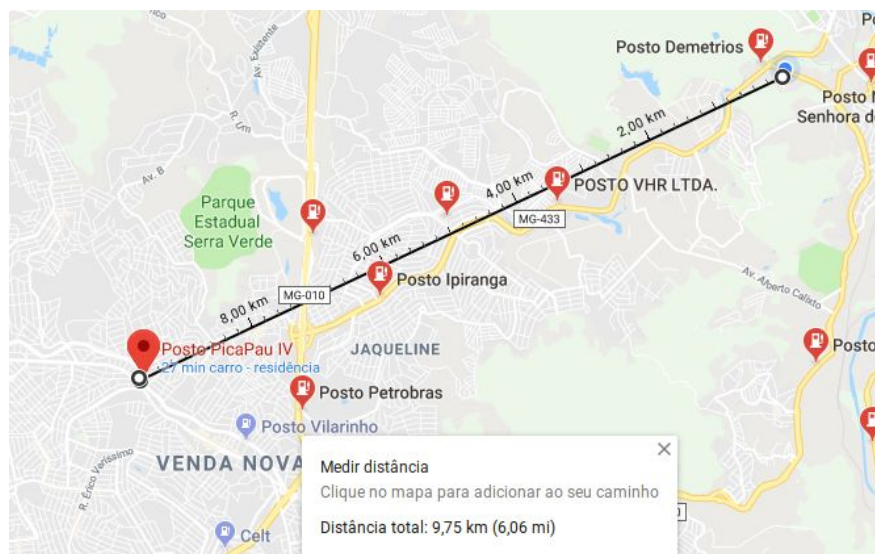


Imagem 7: Teste 3

Ao colocar um print no cálculo da distância foi conferido que o Posto Pica Pau (último da lista) está a aproximadamente 9,86 km do local inicial.


```

caiquefortunato@caiquefortunato:~/Área de Trabalho/tp2$ python3 servidor.py 1995
Server is ready to receive data.
Mensagem recebida: P|1|1|7|-19.768042|-43.88599
distancia 1.7808135379352132
distancia 0.46461623408726016
distancia 1.1163384793977251
distancia 4.0134627011067545
distancia 3.715756266751264
distancia 5.169009423710179
distancia 3.2349690653189906
distancia 9.861372182191284
Mensagem enviada: 1|3765|-43.875342|-19.768436

```

Imagem 8: Teste 4

Funcionamento da pesquisa de dados

Para a pesquisa foi definido as coordenadas: -19.768042 e -43.88599. O raio inicial foi de 7km, de modo que não alcançasse o posto mais distante que está a aproximadamente 9km de distância, mas que alcançasse todos os outros. O menor preço é o do Posto Nossa Senhora do Rosário, de coordenada -19.768436 e -43.875342. Ao rodar o programa foi conferido que o funcionamento está correto.

```

caiquefortunato@caiquefortunato: ~/Área de Trabalho/tp2
caiquefortunato@caiquefortunato:~/Área de Trabalho/tp2$ python3 cliente.py local
host 1995
Digite D para inserir dados | P para pesquisa | S para sair
p
Digite o tipo do combustível
1
Digite o raio de busca
7
Digite a latitude
-19.768042
Digite a longitude
-43.88599
Mensagem enviada: P|0|1|7|-19.768042|-43.88599
Mensagem recebida: 0|3765|-43.875342|-19.768436
Menor valor: 3765 na latitude -19.768436 e longitude -43.875342

caiquefortunato@caiquefortunato: ~/Área de Trabalho/tp2
caiquefortunato@caiquefortunato:~/Área de Trabalho/tp2$ python3 servidor.py 1995
Server is ready to receive data.
Mensagem recebida: P|0|1|7|-19.768042|-43.88599
Mensagem enviada: 0|3765|-43.875342|-19.768436

```

Imagem 9: Teste 5

Ao aumentar o raio para 10km, alcançando o posto mais distante, que possui o menor preço de combustível é possível ver também a resposta correta.

```
Digite D para inserir dados | P para pesquisa | S para sair
p
Digite o tipo do combustivel
1
Digite o raio de busca
10
Digite a latitude
-19.768042
Digite a longitude
-43.88599
Mensagem enviada: P|1|1|10|-19.768042|-43.88599
Mensagem recebida: 1|1523|-43.971081|-19.805947
Menor valor: 1523 na latitude -19.805947 e longitude -43.971081

caiquefortunato@caiquefortunato: ~/Área de Trabalho/tp2
caiquefortunato@caiquefortunato:~/Área de Trabalho/tp2$ python3 servidor.py 1995
Server is ready to receive data.
Mensagem recebida: P|0|1|7|-19.768042|-43.88599
Mensagem enviada: 0|3765|-43.875342|-19.768436

Mensagem recebida: P|1|1|10|-19.768042|-43.88599
Mensagem enviada: 1|1523|-43.971081|-19.805947
```

Imagem 10: Teste 6

Funcionamento da retransmissão

Não foi possível encontrar um caso de teste onde houvesse falha na transmissão da mensagem. Sendo assim, uma alternativa foi: iniciar o cliente e enviar uma mensagem para o servidor. Dentro de 5 segundos iniciar o cliente. A primeira mensagem irá falhar, a segunda será enviada.

```
caiquefortunato@caiquefortunato:~/Área de Trabalho/tp2$ python3 cliente.py local
host 1995
Digite D para inserir dados | P para pesquisa | S para sair
d
Digite o tipo do combustivel
0
Digite o preco do combustivel
4532
Digite a latitude
-19.768042
Digite a longitude
-43.88599
Mensagem enviada: D|0|0|4532|-19.768042|-43.88599
Resposta nao recebida, tentando novamente
Mensagem enviada: D|0|0|4532|-19.768042|-43.88599
Mensagem recebida: 0

Dado inserido no arquivo com sucesso

caiquefortunato@caiquefortunato: ~/Área de Trabalho/tp2
caiquefortunato@caiquefortunato:~/Área de Trabalho/tp2$ python3 servidor.py 1995
Server is ready to receive data.
Mensagem recebida: D|0|0|4532|-19.768042|-43.88599
Mensagem enviada: 0
```

Imagem 11: Teste 7

Os testes foram realizados em uma máquina com sistema operacional Linux com as seguintes configurações: 8GB de memória RAM, 1TB de HD, Processador Intel Core i5 5200U com 2.2GHz e distribuição Ubuntu 16.04 LTS.

Conclusão

A implementação deste trabalho foi bem interessante, principalmente pela coincidência da falta de combustíveis nos postos devido à greve dos caminhoneiros. Além disso, o trabalho apresenta uma proposta bem interessante e que pode ser útil no dia-a-dia e implementado em massa de uma maneira mais robusta para que as pessoas em geral possam utilizar e consultar o menor valor do combustível dado o raio estabelecido.

Além do tema, foi possível perceber como a comunicação via UDP funciona na prática, quais as diferenças com o TCP, vantagens e desvantagens. Ou seja, passando da teoria vista em sala para a prática. No entanto, não foi implementado nenhum tipo de verificação de integração das mensagens, somente se a mesma não chega para o servidor, o que fica para trabalhos futuros, uma vez que é bem interessante.

As principais dificuldades encontradas foram na estratégia para retransmissão das mensagens, uma vez que a ideia inicial foi o uso de threads, no entanto foi encontrada uma solução melhor que está na própria biblioteca do socket. Além disso, foi necessário ler as funções de tal biblioteca para a conversão de IP.

Referências

1. ***How to convert IPv4-mapped-IPv6 address to IPv4 (string format)?***

Disponível em:

https://stackoverflow.com/questions/11233246/how-to-convert-ipv4-mapped-ipv6-address-to-ipv4-string-format?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa> Último acesso: 6 de maio de

2018.

2. Material da disciplina.

3. Distância entre dois pontos. Disponível em:

<<https://www.mapanet.eu/PT/resources/Script-Distance.htm>> Último acesso:
30 de maio de 2018.